Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Амурский государственный университет»

Акилова И.М., С.Г. Самохвалова

ПРИКЛАДНОЕ ПРОГРАММИРОВАНИЕ

Методические указания к лабораторным занятиям
для студентов очной формы обучения
факультета специального образования

Благовещенск
2020

Прикладное программирование. Методические указания к лабораторным занятиям для студентов очной формы обучения факультета специального образования. / сост. И.М., Акилова, С.Г. Самохвалова – Благовещенск.: ФГБОУ ВО «АмГУ», 2020 г. – 45 с.

Методические указания к лабораторным работам содержат описание основных конструкций языка логического программирования ПРОЛОГ, используемого при разработке логических программ. Иллюстративные примеры позволяют получить представление о приемах построения логических программ. Предлагаемые для самостоятельного решения задачи, служат для накопления практического опыта разработки программ.

Методические указания рекомендуются студентам направления подготовки 09.02.03 - Программирование в компьютерных системах, среднее профессиональное образование, а также могут быть полезны для преподавателей и студентов, преподающих и осваивающих логическое программирование рамках других направлений подготовки.

Рецензент:

Юрьева Т.А. доцент, к.п.н. доцент кафедры общей математики и информатики ФГБОУ ВО АмГУ

СОДЕРЖАНИЕ

Введение 4

Изучение работы с интегрированной оболочкой системы турбо пролог. 5

Лабораторная работа № 1 Родительские отношения 7

Лабораторная работа № 2 Логический вывод 9

Лабораторная работа № 3 Разветвления. 11

Лабораторная работа № 4 Рекурсия. 20

Лабораторная работа № 5 Решение логических задач. 23

Лабораторная работа № 6 Списки. 28

Лабораторная работа № 7 Строки

Список использованных источников 38

ВВЕДЕНИЕ

Пролог является результатом многолетней исследовательской работы. Первая официальная версия Пролога разработана Аланом Кольмароэ (ALain Colmerauer) в Марсельском университете во Франции в начале 1970-х годов как инструмент для программирования логики. В результате своего развития появился язык более мощный, чем такие хорошо известные сегодня языки программирования, как Паскаль и Си.

Пролог известен как декларативный язык. Это означает, что при заданных необходимых фактах и правилах, Пролог будет использовать дедуктивные умозаключения для решения задач программирования. Эта его отличительная особенность выгодно контрастирует с традиционными процедурными языками.

Пролог является очень важным инструментом в программировании приложений искусственного интеллекта и в разработке экспертных систем. Высокий уровень абстракции, легкость и простота в представлении сложных структур данных, возможность моделировать логические отношения между объектами и процессами существенно облегчают решение задач в различных предметных областях.

Пролог – язык программирования, в котором решения компьютерных задач выражаются с помощью фактов, представляющих отношения между объектами, и правил, специфицирующих выводимые из фактов следствия. Механизмы представления знаний объектах и отношениях в Прологе являются одновременно и высокоуровневыми и многоцелевыми. Это дает программисту два существенных преимущества. Первое, весьма ощутимое преимущество состоит в том, что программист освобождается от необходимости вникать в организацию физической памяти, отводимой для данных, которыми манипулирует программа. Второе преимущество состоит в той легкости, с которой в этом языке могут быть выражены сущности и отношения из самых различных областей человеческой деятельности.

ИЗУЧЕНИЕ РАБОТЫ С ИНТЕГРИРОВАННОЙ ОБОЛОЧКОЙ СИСТЕМЫ ТУРБО ПРОЛОГ.

Цель работы: Освоение основных режимов работы в интегрированной оболочке системы Турбо Пролог: редактирования, компиляции, отладки, ведения диалога и работы с файлами.

Краткие теоретические сведения

1. Турбо-Пролог, версия 2.0

Система Турбо Пролог версия 2.0 может работать на ПЭВМ, совместимых с IBM PC XT/AT и PS/2, с ОЗУ минимум 384 Кбайт и двумя НГМД по 360 Кбайт. Рекомендуется иметь ОЗУ 512/640 Кбайт и НМД типа "Винчестер". В файле config.sys должно быть указано files=20 buffers=40.

Программа на Турбо Прологе состоит из следующих в определенном порядке секций и имеет следующую структуру [2]:

constants /* Секция объявления констант. Может отсутствовать */

domains /* Секция объявления нестандартных и/или составных типов данных. Может отсутствовать */

database - имя_ВБД /* Необязательная секция объявления предикатов для работы с внутренней базой данных (ВБД) */

predicates /* Секция объявления предикатов */

clauses /* Секция объявления правил и фактов */

goal /* Секция объявления внутренней цели. Может отсутствовать */

При составлении программы на Прологе необходимо соблюдать следующие ограничения:

- комментарии в программе могут располагаться в программе на любом месте. Комментарий начинается либо с символа % либо с последовательности символов /* и заканчиваться */;

- в программе может использоваться только один раз секция GOAL;

- все предикаты в CLAUSES с одинаковыми именами должны записы-

ваться подряд;

- большинство стандартных предикатов выполняют несколько функций в зависимости от состояния параметров, входящих в предикат. Известные параметры называют входными (INPUT – (i)), неизвестные – выходными (OUTPUT – (o)). Совокупность входных параметров определяет работу предиката. Эта совокупность называется проточным шаблоном.

Интегрированная оболочка системы Турбо-Пролог предоставляет следующие возможности:

- создавать и редактировать тексты программ;
- выполнять и отлаживать программы;
- транслировать программы в объектные файлы;
- компоновать объектные файлы в исполняемые модули;
- получать справочную информацию, изменять размеры окон и их цвет;
- устанавливать параметры и конфигурацию системы.

При первоначальном входе в интегрированную среду Турбо-Пролога на экране монитора появляется главное меню (рис.1). В верхней строке находятся названия 6 основных режимов работы системы. Текущее положение в меню отмечено выделяющейся по цвету и яркости прямоугольной полоской. Перемещая эту полоску (курсор) с помощью клавиш с горизонтальными стрелками нажатием клавиши Enter можно выбрать необходимый режим. Это можно сделать также одновременным нажатием клавиши Alt и первой буквы названия соответствующего меню, например, для выбора режима редактирования достаточно нажать Alt-E.

Для удобства работы для наиболее часто используемых операций в оболочке Турбо Пролога вместо выбора из меню (или подменю) можно использовать нажатие функциональных клавиш, либо определенного сочетания клавиш (Hot keys). Действие той или иной функциональной клавиши может быть различным в зависимости от того, в каком режиме находится система. Более полную подсказку можно получить нажатием клавиш Alt-H.

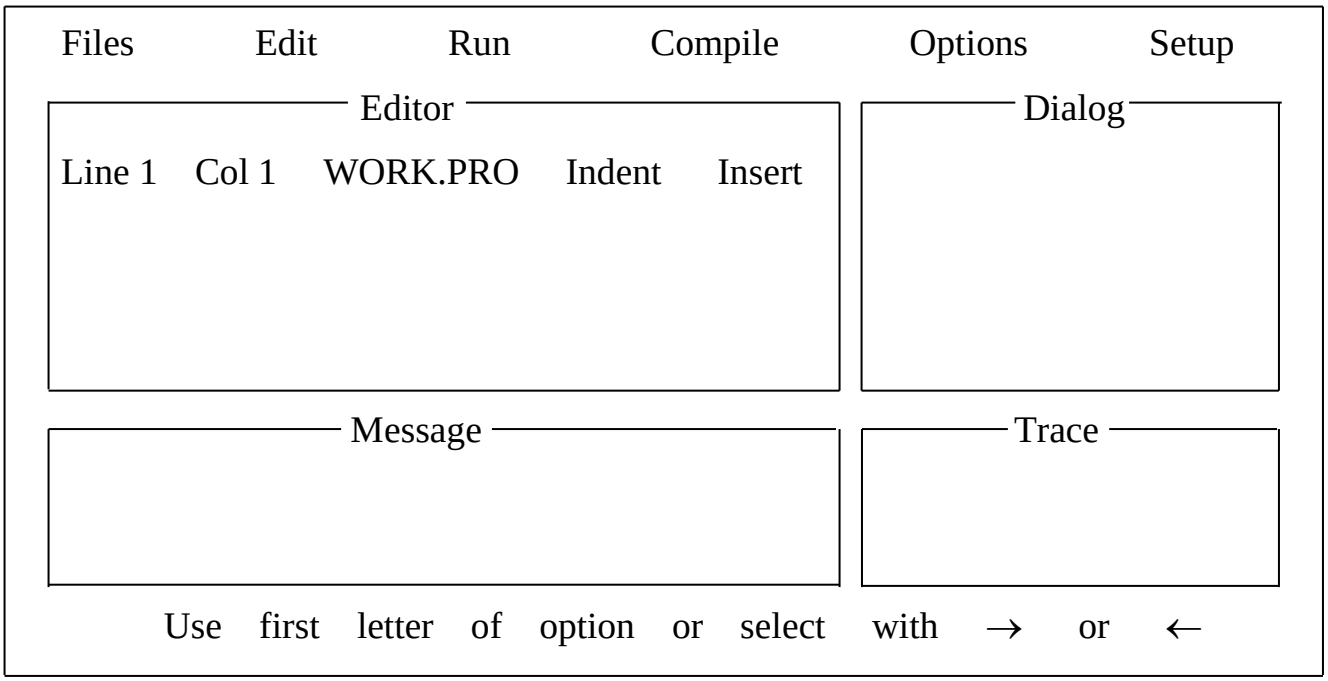


Рис. 1

Экран разделен на 4 окна:

- окно редактора текстов, в которое загружаются отлаживаемые или редактируемые программы;

- окно диалога является по умолчанию окном, если в программе не назначены другие, ввода и вывода из программы;

- окно для вывода сообщений системы;

- окно сообщений отладчика (трассировки).

Первоначальное разделение экрана может быть изменено в режиме Setup. Окно, в котором находится курсор, называется текущим окном.

Нижняя строка является строкой подсказки или иначе навигационной строкой. Навигационную строку вверху, показывающую местоположение курсора (Line - номер текущей строки, Col - позиция курсора в строке), название редактируемого файла и режимы редактирования, имеет также окно редактирования. Если имя редактируемого фала не задано, то по умолчанию он называется WORK.PRO. Если файл с таким именем находится в текущей директории, то он автоматически загружается в окно редактирования.

Все режимы главного меню, кроме Edit и Run, содержат дополнительные подменю. Выход из любого подменю осуществляется нажатием клавиши Esc.

Цикл разработки программы

Турбо Пролог поддерживает в широком смысле структурное программирование и сопровождение проекта. Цикл разработки программы (без постановочной части) можно представить следующей схемой:

1. С помощью встроенного редактора текстов исходный текст программы вводится в систему.

2. Периодически, по мере ввода новых предикатов, программа транслируется для выявления синтаксических ошибок, которые тут же устраняются.

3. С помощью встроенных средств трассировки производится отладка каждого нового предиката.

4. Периодически производится структуризация программы. Независимые части программы выделяются в отдельные модули.

5. Процесс повторяется до получения программы, удовлетворяющей внешним спецификациям, которые, в частности, сами могут быть написаны на Турбо Прологе.

2. Основные режимы работы

Основные режимы работы [2] заданы главным меню в верхней строке экрана. Такими режимами являются:

Files - работа с файлами и взаимодействие с MS-DOS

Edit - ввод и редактирование программы

Run - выполнение и отладка программы

Compile - трансляция и компоновка программы

Options - установка режимов компиляции

Setup - установка режимов работы оболочки

Работа с файлами и взаимодействие с MS-DOS

2.1. Работа с файлами и взаимодействие с MS-DOS

При выборе режима Files в верхней части экрана под словом Files появится подменю (рис. 2.).

Перемещение по подменю осуществляется также с помощью клавиш со стрелками, вводом первой (заглавной) буквы слова. Для быстрого выполнения

наиболее частых операций имеются зарезервированные функциональные клавиши или одновременное нажатие нескольких клавиш (Hot keys).

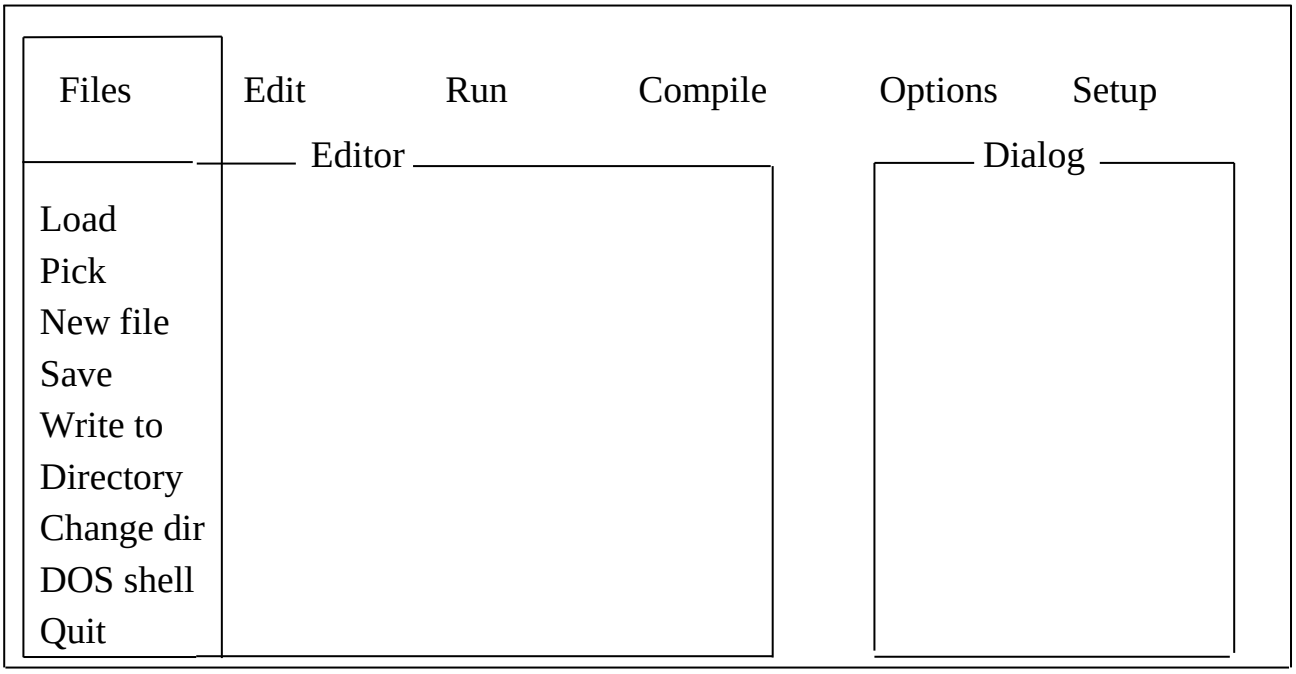


Рис. 2

Load – позволяет загрузить исходный текст программы на Прологе в окно редактора для его последующей модификации, выполнения или отладки.

Pick – позволяет работать из оболочки Турбо-Пролога одновременно с 8-ю файлами.

New file – очистка окна редактора текстов для ввода нового файла.

Save – сохранение редактируемой программы в файле на диске.

Write to – запись редактируемой программы в заданный файл на диске.

Directory – вывод содержимого указанной директории на экран.

Change dir – смена текущей директории. Равносильна команде ChDir в MS DOS.

OS shell – выполнение команды операционной системы. Выход из системы Турбо-Пролог в MS-DOS с возвратом по команде EXIT.

Quit – окончание работы. Выход из системы Турбо-Пролог в MS-DOS.

При работе с файлами приняты следующие соглашения об их типах:

.PRO – файл с исходным текстом программы

.BAK – файл сохранения после редактирования

.PRJ – файл с именами модулей, входящих в проект разрабатываемой системы

.EXE – исполняемый файл после компоновки

.OBJ – объектный файл после трансляции

.ARC – упакованный файл архивации

2.2. Ввод и редактирование программы

Встроенный редактор интегрированной оболочки Турбо-Пролога имеет набор команд, который является подмножеством команд широко известного редактора текстов WordStar. Кроме того, при редактировании можно пользоваться более короткими командами, пользуясь функциональными и управляющими клавишами.

Все команды редактирования можно переопределить, однако учитывая, что они совпадают с оболочками языков Турбо Паскаль, Турбо Си и Турбо Бейсик, это не рекомендуется делать.

Как и при работе с обычным текстовым редактором, ввод может происходить в режиме вставки текста или в режиме наложения его на уже существующий. Переход с одного режима в другой осуществляется нажатием клавиши Ins.

В случае слияния файлов, копирования блока из одного файла в другой и т.п. в системе Турбо Пролог может быть открыто дополнительное окно редактирования (Aux Editor) в нижнем правом углу экрана.

Команды редактора [2] условно делятся на команды перемещения, команды удаления текста, команды работы с блоками текста и вспомогательные. Выход из редактора в главное меню - F10 или по одновременному нажатию клавиши Alt и клавиши с первой буквой имени соответствующего режима.

Команды перемещения

Направление перемещения	Клавиши
Курсор вправо на один символ	Стрелка вправо или Ctrl-D

Курсор влево на один символ	Стрелка влево или Ctrl-S
Курсор вверх на одну строку	Стрелка вверх или Ctrl-E
Курсор вниз на одну строку	Стрелка вниз или Ctrl-X
Курсор вправо на одно слово	Ctrl-Стрелка вправо
Курсор влево на одно слово	Ctrl-Стрелка влево
Курсор в конец строки	End
Курсор в начало строки	Home
Курсор в начало окна	Ctrl-Home
Курсор в конец окна	Ctrl-End
Курсор вверх на один экран	PgUp или Ctrl-R
Курсор вниз на один экран	PgDn или Ctrl-C
Курсор в начало файла	Ctrl-PgUp или Ctrl-Q R
Курсор в конец файла	Ctrl-PgDn или Ctrl-Q C

Команды удаления текста

Название команды	Клавиши
Удаление символа над курсором	Del
Удаление символа слева от курсора	BackSpace или Ctrl-H
Удаление слова над курсором	Ctrl-T
Удаление строки над курсором	Ctrl-Y
Удаление начала строки до курсора	Ctrl-Q T
Удаление от курсора до конца строки	Ctrl-Q Y

Если при редактировании программы возникли какие-либо трудности, то нажатие клавиши F1 вызовет следующее меню системы подсказки:

Edit Help – [Помощь при редактировании] Заголовок меню

Show help file – вывод на экран файла справочника по всей системе. Нажатие F5 позволяет увеличить окно со справочником до размеров экрана. Повторное нажатие уменьшит его до прежних размеров. Выход из справочника по Esc.

Cursor movement – вывод на экран справочника по командам для перемещения курсора.

Insert & Delete – справочник по операциям вставки и удаления текста.

Block Functions – справочник по операциям над блоками текста.

WordStar-like – справочник по командам редактирования, аналогичным командам редактора текстов WordStar.

Miscellaneous – справочник по командам редактирования не входящим в какую-либо группу.

Global functions – справочник по командам, выполняющим операции над всем текстом.

Hot keys – справочник по клавишам для быстрого выполнения наиболее часто встречающихся операций. При этом выбранная с помощью клавиш со стрелками операция может быть тут же выполнена.

2.3. Выполнение и отладка программы

Режим Run обеспечивает компиляцию находящейся в окне редактора текстов программы и ее выполнение. Этот режим обеспечивает быструю интерактивную отладку программы, так как при обнаружении синтаксических и семантических ошибок имеется возможность их тут же исправить. Мощным инструментом отладки программ в системе являются встроенные средства трассировки.

Возможны следующие виды трассировки:

– пошаговая трассировка всей программы

– трассировка заданных предикатов

– трассировка в режиме оптимизации

– интерактивное включение/выключение трассировки

– интерактивный вывод результатов трассировки и выходных сообщений на печать и в дисковый файл.

Отметим, что трассировка является также важным средством изучения Турбо Пролога, так как, в случае непонимания работы какого-либо предиката, его работу можно посмотреть по шагам.

По умолчанию режим трассировки выключен (Off). Для установки/отключения режима трассировки следует выбрать подменю "Директивы компилятора" меню Options и в нем выбрать вход Trace. На экране появится небольшое меню, содержащее три входа:

Trace – включить режим полной трассировки;

ShortTrace – включить режим сокращенной трассировки;

Off – выключить трассировку.

Если выбрать Trace (выход из всех подменю по Esc), то после запуска программы на выполнение (Run или Alt-R) в окне диалога появится подсказка Goal:. После ввода цели, в окне трассировки за словом CALL: появляется вызываемый предикат.

Нажатие клавиши F10 разрешает выполнение следующего шага программы. Одновременно в окне редактирования курсор показывает на выполняемый в текущий момент предикат.

После подсказки REDO: отображаются результаты вычисления (унификации).

При успешном поиске подстановке после слова RETURN: выводится ее результат. При неудаче выводится подсказка FAIL.

Трассировка в любой момент может быть прекращена нажатием клавиши Esc.

Нажатие Alt-T позволяет изменить режим трассировки. При этом на экране появляется следующее меню:

Status	On	Запрет/разрешение отображения статуса
Trace window	On	Запрет/разрешение окна трассировки
Edit window	On	Запрет/разрешение окна редактирования

После нажатия Shift-F10 можно по желанию изменить размеры окон.

Внутри программы можно размещать передикат trace(on), который включает режим трассировки, или trace(off) – выключающий ее.

Нажатие клавиш Alt-P позволяет перенаправить вывод результатов трассировки на печать или в файл PROLOG.LOG на диске.

Режим ShortTrace отличается от Trace только тем, что компилятор производит оптимизацию программы (в режиме Trace оптимизацияне производится). Это особенно важно при отладке рекурсивных вызовов, особенно при "хвостовой рекурсии", когда рекурсивный вызов является последней подцелью в теле предиката. Этот вид рекурсивных вызовов оптимизируется Турбо Прологом и преобразуется в цикл, что экономит память при исполнении.

2.4. Трансляция и компановка программ

Подменю режима Compile содержит следующие входы:

Memory – отладочная трансляция в память

OBJ file – трансляция в файл типа .OBJ

EXE file (auto link) – трансляция с созданием исполняемого (.EXE) файла с автоматической компоновкой модулей

Project (all modules) –трансляция всех входящих в проект модулей;

Link only – выполнение компоновки модулей в исполняемый файл.

2.5. Установка режимов компиляции

Исходный текст программы на Турбо Прологе перед выполнением проходит обязательную фазу компиляции и компоновки. Для определения параметров этих процессов служит режим Options.

Меню режима Options содержит следующие входы:

Link options (Опции компоновки)

Содержит подменю:

Map file ON/OFF - создавать или нет файл с картой компоновки.

Libraries - ввод имен пользовательских библиотек для компоновки.

Edit PRJ file (Редактирование файла с описанием проекта)

Запрашивается имя файла, в котором перечислены модули, входящие в проект. Этот файл затем используется компоновщиком.

Compiler Directive (Директивы компиляции)

Содержит подменю (названия входов даны в круглых скобках) позволя-

ющие установить:

– распределение памяти (Memory allocation) под код, стек, тип и рекурсию. Размер этих областей устанавливается в параграфах (параграф равен 16 байтам);

– какие виды контроля выполнять во время исполнения программы (Run-time check): нажатие клавиши Breack, нарушение границ стека и целочисленное переполнение;

– допустимый уровень ошибок трансляции (Error level): ошибки недопустимы, по умолчанию (1), максимальный (2);

– выдачу предупреждения при наличии недетерминированных предикатов (Non-determ warning ON/OFF);

– предупреждение о наличии переменных, которые используются в предикате только один раз (Variable used once warning ON/OFF);

– уровень трассировки (Trace): полная, сокращенная и трассировка выключена;

– включение вывода диагностики по результатам трансляции (Diagnostics ON/OFF).

2.6. Получение справочной информации, изменение размеров окон и их цвета

Подменю режима SetUp содержит следующие входы:

Colors – изменение цвета окон при наличии цветного монитора. На экран выводится таблица цветовой гаммы, из которой для каждого окна выбирается цвет фона и цвет изображения;

Window size – изменение размеров окон. При выборе этой опции в нижней строке экрана появляется подсказка, как с помощью клавиш со стрелками изменить размер или местоположение окна.

Directories– установка директорий по умолчанию.

Miscellaneous – разные опции связанные с настройкой на адаптер видеомонитора;

Load SYS file – загрузка файла с параметрами интегрированной среды;

Save SYS file сохранение параметров настройка среды интегрированной среды в файле типа *.SYS.

3. Стандартные предикаты

3.1. Предикаты преобразования типов

- char_int(Символ,Число) (char,integer): прототип (i,o) – связывает параметр "Число" с кодом ASCII параметра "Символ"; прототип (o,i) – связывает параметр "Символ" с символом, код которого определяется параметром "Число"; прототип (i,i) – выполняется успешно, если код, определяемый параметром "Число", является ASCII – кодом символа, определяемого параметром "Символ".

Например:

char_int(’A’,X) – переменная X принимает значение 65.

char_int(X,66) – переменная X принимает значение ‘B’.

char_int(‘A’,65) – выполняется успешно.

- str_char(Строка,Символ) (string,char): прототип: (i,o) – заданная первым параметром строка, состоящая из единственного символа, преобразуется в символ. Символ связывается со вторым параметром; прототип (o,i) – преобразуется символ в строку, состоящую из единственного символа и связывает ее с заданной переменной; прототип: (i,i) – выполняется успешно, если параметры связаны с представлениями одного и того же символа.

Например:

str_char(“A”,X) – результат X=’A’.

str_char(X,’A’) – результат X=”A”.

str_char(“A”,‘A’) – выполняется успешно.

- str_int(Строка,Целое число) (string,integer): прототип:(i,o) – преобразует строку в целое число. Число связывается со вторым параметром; прототип (o,i) – связывает с первым параметром строку, представляющую собой запись целого числа, связанного со вторым параметром; прототип(i,i) – выполняется успешно, если связанная с первым параметром строка является представлением числа, связанного со вторым параметром.

Например:

str_int(“123”,X) – результат: X=123

str_int(X,456) – результат: X=”456”

str_int(“234”,234) – выполняется успешно.

- str_real(Строка,Действительное число) (string,real): прототип:(i,o) – связывает второй параметр с действительным числом, определяемым записью числа в строке, заданной первым параметром; прототип (o,i) – связывает с первым параметром строку, представляющую собой запись действительного числа, заданного вторым параметром; прототип:(i,i) – выполняется успешно, если связанная с первым параметром строка является представлением числа, связанного со вторым параметром.

Например:

str_real(“1.23”,X) – результат: X=1.23

str_real(X,0.56) – результат: X=”0.56”

str_real(“4.567”,4.567) – выполняется успешно.

- file_str(Имя_ файла_ DOS,Строка) (string,string): прототип (i,o) – читает строку из заданного файла и связывает ее с параметром "Строка". Максимально допустимый размер строки – 64 К. Признаком конца строки является символ Ctrl – Z (десятичный код ASCII=26).

Например:

file_str(“B:TEXT1”,X) – символы из файла TEXT1 на накопителе B будут прочитаны и связаны с переменной X.

- field_str(Строка, Столбец, Длина, Строка_символов) (integer, integer, integer, string): прототип (i,i,i,i) – записывает строку, связанную с параметром "Строка_символов", в поле, определяемое длинной и номерами строки и столбца. Если строка длиннее, чем заданное поле, то записывается только начало строки. Если строка короче, то оставшиеся позиции поля заполняются пробелами; прототип (i,i,i,o) – строка, определяемая длиной и позицией, связывается с параметром "Строка_символов". Проследите, чтобы поле в текущем окне соответствовало параметрам.

Например:

field_str(15,5,5,"hollo") – строка "hollo" записывается в поле, начинающееся с позиции (15,5).

field_str(10,30,5,X) – строка длиной 5, начинающаяся с позиции (10,30), связывается с переменной X.

- str_len(Строка,Длина) (string,integer): прототип (i,o) – с параметром "Длина" связывается количество символов в заданной строке; прототип (i,i) – выполняется успешно, если строка имеет заданную длину.

Например:

str_len("hollo", X) – результат: X=5.

str_len("book", 4) – выполняется успешно.

- isname(Строка) (string) прототип (i) - выполняется успешно, если последовательность символов, связанная с параметром, представляет собой имя, допустимое в Турбо – Прологе.

Например:

isname ("abcd") – выполняется успешно.

- upper_lower(Строка1,Строка2) (string, string): прототип (i,i) – выполняется успешно, если с первым и вторым параметром связаны идентичные строки, представленные соответственно прописными и строчными буквами; прототип (i,o) – связывает со вторым параметром строку, полученную из строки, связанной с первым параметром, заменой прописных букв на строчные; прототип (o,i) – связывает с первым параметром строку, полученную из строки, связанной со вторым параметром, заменой строчных букв на прописные.

Например:

upper_lower("A","a") – выполняется успешно.

upper_lower("ZDF",X) – результат: X="zdf"

upper_lower(X,"house") – результат: X="HOUSE"

3.2. Арифметические операции

+ сложение; – вычитание; * умножение; / деление; mod абсолютная величина; div целочисленное деление

3.2. Операторы отношений

Операторы отношений являются инфиксными операторами (т.е. должны находится между двумя сравниваемыми величинами). Свободные переменные в операторах отношений не допускаются. Для операторов отношений приняты следующие обозначения:

< меньше; > больше; = равно; <= меньше или равно; >= больше равно; <> не равно.

3.3. Математические функции

Функция	Описание
abs(X) /* (Var) (i) */	Возвращает абсолютное значение X
round(X) /* (Var) (i) */	Возвращает округленное целое значение X
sqrt(X) /* (Var) (i) */	Возвращает квадратный корень из X
trunc(X) /* (Var) (i) */	Возвращает целое значение X отбрасывая дробную часть
exp(X) /* (Var) (i) */	Возвращает значение e в степени X
log(X) /* (Var) (i) */	Возвращает десятичный логарифм X
ln(X) /* (Var) (i) */	Возвращает натуральный логарифм X
arctan(X) /* (Radians) (i) */	Возвращает арктангенс X
cos(X) /* (Radians) (i) */	Возвращает косинус X
sin(X) /* (Radians) (i) */	Возвращает синус X
tan(X) /* (Radians) (i) */	Возвращает тангенс X

Все тригонометрические функции требуют, чтобы аргумент X задавался в радианах.

ЛАБОРАТОРНАЯ РАБОТА № 1

ТЕМА: РОДИТЕЛЬСКИЕ ОТНОШЕНИЯ

*Соотношение между процедурным и декларативным смыслом***.** Создавая *Пролог*-программы всегда надо помнить о процедурном и декларативном смысле. Декларативный смысл касается отношений, определенных в программе, т.е. определяет, что должно быть результатом программы. С другой стороны, про-

цедурный смысл определяет, как этот результат может быть достигнут, т.е. как реально отношения обрабатываются *Прологом.*

Введем отношение родитель (parent) между объектами.

parent(«тимур», «борис»).

Это факт, определяющий, что Тимур является родителем Бориса.

parent - имя отношения, тимур, борис - его аргументы. Теперь можно записать программу, описывающую все дерево родственных отношений.

parent(«нина», «борис»).

parent(«тимур», «борис»).

parent(«тимур», «лиза»).

parent(«борис», «анна»).

parent(«борис», «таня»).

parent(«марина», «анна»).

parent(«таня», «юля»).

Эта программа состоит из семи предложений (утверждений) - clause. Каждый clause записан фактом в виде отношения **parent.**

При записи фактов надо соблюдать следующие правила:

1) имена всех отношений и объектов с маленькой буквы;

2) сначала записывается имя отношения, затем в круглых скобках через запятую – объекты;

3) в конце ставится точка.

Совокупность фактов в *Прологе* называют базой данных. К составленной базе данных можно задать вопросы. Вопрос в обычном *Прологе* начинается с ?. Вопрос записывается также, как и факт.

Например: ? - parent («борис», «таня»).

yes

Можно задать вопрос и узнать, кто родитель Лизы:

? - parent (X, "лиза").

X=тимур

Здесь X - переменная. Ее величина неизвестна, и она может принимать значения. В данном случае ее значением будет объект, для которого это утверждение истинно.

Можно задать более общий вопрос: кто является родителем родителя Юли? Так как нет отношения **grandparent**, то можно разбить этот вопрос на

два:

1. Кто родитель Юли? Предположим, **Y.**
2. Кто родитель **Y**? Предположим, **X.**

Тогда составной вопрос: **? - parent (Y, «юля»), parent (X, Y).**

X=борис Y=таня

При поиске решения сначала находится **Y** , а затем по второму условию **X**. Вопрос: кто внуки тома?: **? - parent («тимур», Y), parent (Y, X).**

Y=борис X=анна

Y=борис X=таня

Введем отношение ребенок **child**, обратное к **parent** – «родитель». Можно было бы определить аналогично: **child («лиза», «тимур»)**. Но можно воспользоваться тем, что отношение **child** обратно к **parent**, и записать в виде утверждения **правила: child (Y, X):-parent (X, Y)**. Правило читается так: **Для всех X и Y Y - child X, если X - parent Y**.

Правило отличается от факта тем, что факт - всегда истина, а правило описывает утверждение, которое будет истиной, если будет выполнено некоторое условие. Поэтому в правиле выделяют заключение условия **child (Y, X): parent (X, Y)**. Если условие **parent (X, Y)** выполняется, то логическим следствием из него будет утверждение **child(Y, X)**.

Как правило, используется *Прологом*? Зададим вопрос **? – child («лиза», «тимур»).** В программе нет данных о **child**. Но есть правило, которое верно для всех **X Y**, в т. ч. для Лизы и Тимура. Мы должны применить правило для этих значений. Для этого надо подставить в правило вместо **X** - Тимур, а вместо **Y** - Лиза. Говорят, что переменные будут связаны, а операция будет называться подстановкой.

Получаем конкретный случай для правила

child («лиза», «тимур»):-parent («тимур», «лиза»).

Условная часть приняла вид **parent («тимур», «лиза»).**

Теперь надо выяснить, выполняется ли это условие. Исходная цель **child («лиза», «тимур»)** заменяется подцелью **parent («тимур», «лиза»)**, которая выполняется, поэтому *Пролог* ответит «yes».

Добавим еще одно отношение в базу данных - унарное, определяющее пол.

mail («тимур»).

mail («борис»).

mail («женя»).

femail («лиза»).

femail («нина»).

femail («таня»).

femail («анна»).

Теперь определим отношение **mother**. Оно описывается следующим образом:

Дни всех X и Y X- mother Y, if X - parent Y и X - female.

Таким образом, правило будет **mother (X, Y):-parent (X, Y), female (X)**.

Запятая между двумя условиями означает конъюнкцию целей (два условия должны быть выполнены одновременно).

Как система ответит на вопрос?: **?-mother («нина», «борис»).**

yes

Находится правило **mother**, производится подстановка **X=нина Y=6орис.** Получаем правило:

mother («нина», «борис»):-parent («нина», «борис»), femail («нина»).

Сначала удовлетворяются **parent**, а затем **female**. *Пролог* отвечает: **«yes»**

Вопрос: **?-mother (X, «борис»).**

X=нина

Определим отношение **sister. Для любых X и Y X sister Y, if y X и Y есть общий родитель, и X female**

Запишем правило на *Прологе:*

sister (X, Y):- parent(Z,X), parent(Z,Y), female(X).

Здесь **Z** - общий родитель, **Z** - некоторый, любой.

Можно спросить **? – sister («анна», «таня»)**. Ответ будет **«yes»**, так как мы не потребовали, чтобы **X** и **Y** были разные.

Добавим отношение **different (X, Y)**, которое указывает, что **X** и **Y** разные.

sister (X, Y):- parent(Z,X), parent(Z,Y), female(X), different (X, Y).

5. Войдите в режим редактирования (Alt-E), изучите действие клавиши F1 (меню подсказок) и введите следующие определения:

```
domains                    /вв/
   fakt = symbol           /вв/
 predicates                /вв/
   male(fakt)              /вв/
   female(fakt)            /вв/
   mother(fakt,fakt)       /вв/
   father(fakt,fakt)       /вв/
   wife(fakt,fakt)         /вв/
clauses                          /вв/
   male(“саша”).                 /вв/
   female(“марина”).             /вв/
   mother(“наташа”,”саша”).  /вв/
   father(“петя”,”коля”).        /вв/
   wife(“наташа”,”петя”).        /вв/
  …….
```

6. Введите в секцию clauses 5-7 фактов для предикатов male, female, mother, father, wife.

7. Пользуясь средствами Турбо Пролога постройте предикаты для выражения следующих связей между объектами:

son(X,Y) /вв/

daughter(X,Y) /вв/

brother(X,Y) /вв/

grandmother(X,Y /вв/

grandfather (X,Y) /вв/

cusins(X,Y) /вв/

uncle(X,Y) /вв/

aunt(X,Y) /вв/

Например:

parent(X,Y):- father(X,Y);mother(X,Y). /вв/

husbend(X,Y):-wife(Y,X). /вв/

8. После ввода каждого нового предиката, программу следует откомпилировать, для чего нажать клавишу F9. Если при компиляции будут обнаружены синтаксические ошибки, то их следует исправить и добиться безошибочной компиляции (см. п. 2.2).

Варианты заданий

Задание: Имеется N объектов и заданы отношения между ними: Родитель, мужчина, женщина. Требуется определить новое отношение и выявить круг лиц, ему удовлетворяющих.

Варианты:

1. Определить предикат дети и найти всех детей и детей конкретного лица.
2. Определить предикат внуки и найти всех внуков и внуков конкретного лица.
3. Определить предикат сын и найти всех сыновей и сыновей конкретного лица.
4. Определить предикат дочь и найти всех дочерей и дочерей конкретного лица.

5. Определить предикат дедушка и найти всех дедушек и дедушку конкретного лица.

6. Определить предикат бабушка и найти всех бабушек и бабушку конкретного лица

7. Определить предикат двоюродный дедушка и найти всех двоюродных дедушек и двоюродных дедушек конкретного лица.

8. Определить предикат двоюродная бабушка и найти всех двоюродных бабушек и двоюродных бабушек конкретного лица

9. Определить предикат тетя и найти всех тетей и тетей конкретного лица

10. Определить предикат дядя и найти всех дядей и дядей конкретного лица.

11. Определить предикат брат и найти всех братьев и братьев конкретного лица.

12. Определить предикат сестра и найти всех сестер и сестер конкретного лица.

13. Определить предикат двоюродный брат и найти всех двоюродных братьев и двоюродных братьев конкретного лица.

14. Определить предикат двоюродная сестра и найти всех двоюродных сестер и двоюродных сестер конкретного лица.

15. Определить предикат племянник и найти всех племянников и племянников конкретного лица.

16. Определить предикат потомок и найти всех потомков и потомков конкретного лица.

17. Определить предикат предок и найти всех предков и предков конкретного лица.

18. Определить предикат потомки мужского пола и найти всех потомков мужского пола и потомков мужского пола конкретного лица.

19. Определить предикат потомки женского пола и найти всех потомков женского пола и потомков женского пола конкретного лица.

20. Определить предикат предки мужского пола и найти всех предков

мужского пола и предков мужского пола конкретного лица.

21. Определить предикат предки женского пола и найти всех предков женского пола и предков женского пола конкретного лица.

22. Определить предикат потомки по мужской линии и найти всех потомков по мужской линии и потомков по мужской линии конкретного лица.

23. Определить предикат потомки по женской линии и найти всех потомков по женской линии и потомков по женской линии конкретного лица.

24. Определить предикат предки по мужской линии и найти всех предков по мужской линии и предков по мужской линии конкретного лица.

25. Определить предикат предки по женской линии и найти всех предков по женской линии и предков по женской линии конкретного лица.

Контрольные вопросы

1. Из каких основных секций состоит программа на языке Пролог?
2. В какой последовательности записываются необходимые для работы программы на Прологе предикаты?
3. Какие ограничения следует соблюдать при составлении программы на Прологе?
4. Какие возможности предоставляет интегрированная оболочка системы Турбо-Пролог?
5. Каков цикл разработки программы на Прологе?
6. Что такое трассировка программы на Прологе?
7. Какие виды трассировки возможны?
8. Назовите стандартные общесистемные предикаты, позволяющие использовать возможности предоставляемые операционной системой?
9. Назовите предикаты преобразования типов и их функции.
10. Какие математические функции используются при составлении программы на Прологе?

Лабораторная работа № 2

ПРАВИЛА ЛОГИЧЕСКОГО ВЫВОДА

Цель работы: Освоение декларативной и процедурной семантики языка Пролог; освоение соотношения между процедурным и декларативным смыслом; составление запросов к программе на Прологе.

Краткие теоретические сведения

Логическая программа – это множество аксиом и правил, задающих отношения между объектами. Вычислением логической программы является вывод следствий из программы.

Пролог – это язык логического программирования. У языков данного типа два основных отличия от классических алгоритмических языков:

символьное программирование – данные в таких языках суть символы, которые, как правило, представляют только себя и не подлежат интерпретации при выполнении программы;

целевое программирование – алгоритмы получения определенных результатов непосредственно не задаются, а описываются объекты, их свойства и отношения между объектами. Здесь определяются ситуации и формируются задачи вместо того, чтобы детально описывать способ решения этих задач.

Пролог, с одной стороны, является подмножеством формальной логики, а с другой, – языком программирования. Различают декларативную и процедурную семантику (смысл, понимание) *Пролог*-программ. Причины двойного прочтения заключаются в том, что выполнение *Пролог*-программы есть доказательство теорем, использующее логический аспект языка. Выполнение сводится к обоснованию противоречия между отрицанием поставленного вопроса и множеством фактов и правил.

Факты – это предикаты с аргументами-константами, обозначающие отношения между объектами или свойства объектов, именованные этими константами. Факты в программе считаются всегда и безусловно истинными и, таким образом, служат основой доказательства, возникающего при выполнении программы.

Пример:

Факты, описывающие студентов:

нравится (сергей, реп).

нравится (юрий, джаз).

носит (сергей, джинсы).

носит (юрий, пиджак).

Это означает: «Сергею нравится рэп», «Юрию нравится джаз» и т. п.

Правила – это хорновские фразы с заголовком и одной или несколькими подцелями-предикатами.

Пусть задано P:-Q, R, где P, Q, R – термы. Тогда с точки зрения декларативного смысла это предложение читается: «P – истинно, если Q и R истинны». или «из Q и R следует P». Т.е. определяются логические связи между головой предложения и целями в его теле. Таким образом, декларативный смысл программы определяет, является ли данная цель истинной (достижимой), и если является, то при каких значениях переменных она достигается.

Например:

крутой_парень(X):-нравится(X, рэп), носит(X, джинсы).

Рассмотрим декларативный смысл более подробно.

Декларативный смысл касается только отношений, определенных в программе. Декларативная семантика определяет, что должно быть результатом работы программы, не вдаваясь в подробности, как это достигается.

Основная операция, выполняемая в языке *Пролог*, - это операция сопоставления (называемая также унификацией, или согласованием). Операция сопоставления может быть успешной, а может закончиться неудачно. Определяется операция сопоставления так:

- константа сопоставляется только с равной ей константой;

- идентичные структуры сопоставляются друг с другом;

- переменная сопоставляется с константой или с ранее связанной переменной (и становится связанной с соответствующим значением);

- две свободные переменные могут сопоставляться (и связываться) друг с другом. С момента связывания они трактуются как одна переменная: если одна из них принимает какое-либо значение, то вторая немедленно принимает то же

значение.

Пример:

haschild(X):-parent(X ,Y).

Предложение C: haschild(«тимур»):-parent(«тимур,Y).

Конкретизация I: X=Тимур

Предложение C: haschild(«борис»):-parent(«борис», «анна»).

Конкретизация I: X=Борис, Y=Анна

Определение. Пусть дана некоторая программа и цель G, тогда в соответствии с декларативной семантикой можно утверждать, что цель G истинна (т. е. достижима или логически следует из программы) тогда и только тогда, когда в программе существует предложение C такое, что существует такая его (C) конкретизация I, что: (a) голова I совпадает с G (b) все цели в теле I истинны.

Пример:

female(«анна»).

parent(«анна», «борис»).

C(I):

mother(«анна»):-parent(«анна», Y), female(«анна»).

mother(X):-parent(X, Y), female(X).

C:

?- mother(«анна»).

Это определение можно распространить на вопросы следующим образом (в общем случае вопрос - список целей, разделенных запятыми).

Определение. Список целей называется истинным (достижимым), если все цели в этом списке истинны, достижимы при одинаковых конкретизациях переменных. Запятая между целями означает конъюнкцию целей, и они должны быть все истинны.

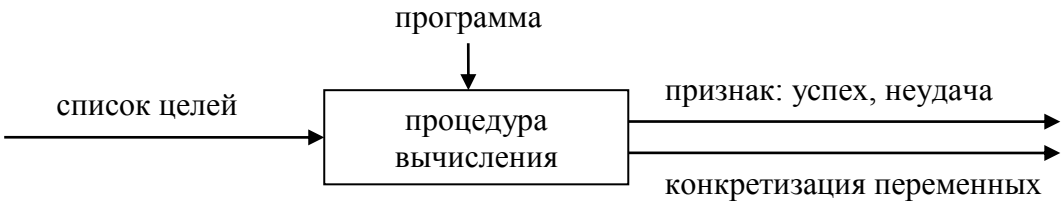
Возможна дизъюнкция целей: истинной должна быть по крайней мере одна из целей. Дизъюнкция обозначается точкой с запятой «;».

Например: P:-Q;R. Читается: P истинна, если Q – истинна или R – истинна, т.е. это то же самое, что P:-R. P:-Q.

Запятая связывает цели сильнее, чем точка с запятой.

Таким образом, предложение P;-Q,R;S,T,U. понимается как P:-(Q,R);(S,T,U). и имеет смысл P:-Q,R. P:-S,T,U.

Процедурная семантика (процедурный смысл) *Пролог*-программы определяет, как *Пролог*-программа отвечает на вопросы. Ответить на вопрос – значит удовлетворить цели. Потому процедурная семантика *Пролога* – это процедура вычисления списка целей с учетом программы.



Пример вычисления:

Рассмотрим программу и на ее примере - процедуру вычисления списка целей.

Программа 1.

1. большой (медведь).

2. большой (слон).

3. маленький (кот).

4. бурый (медведь).

5. черный (кот).

6. серый (слон).

7. темный (Z):-черный (Z).

7.1

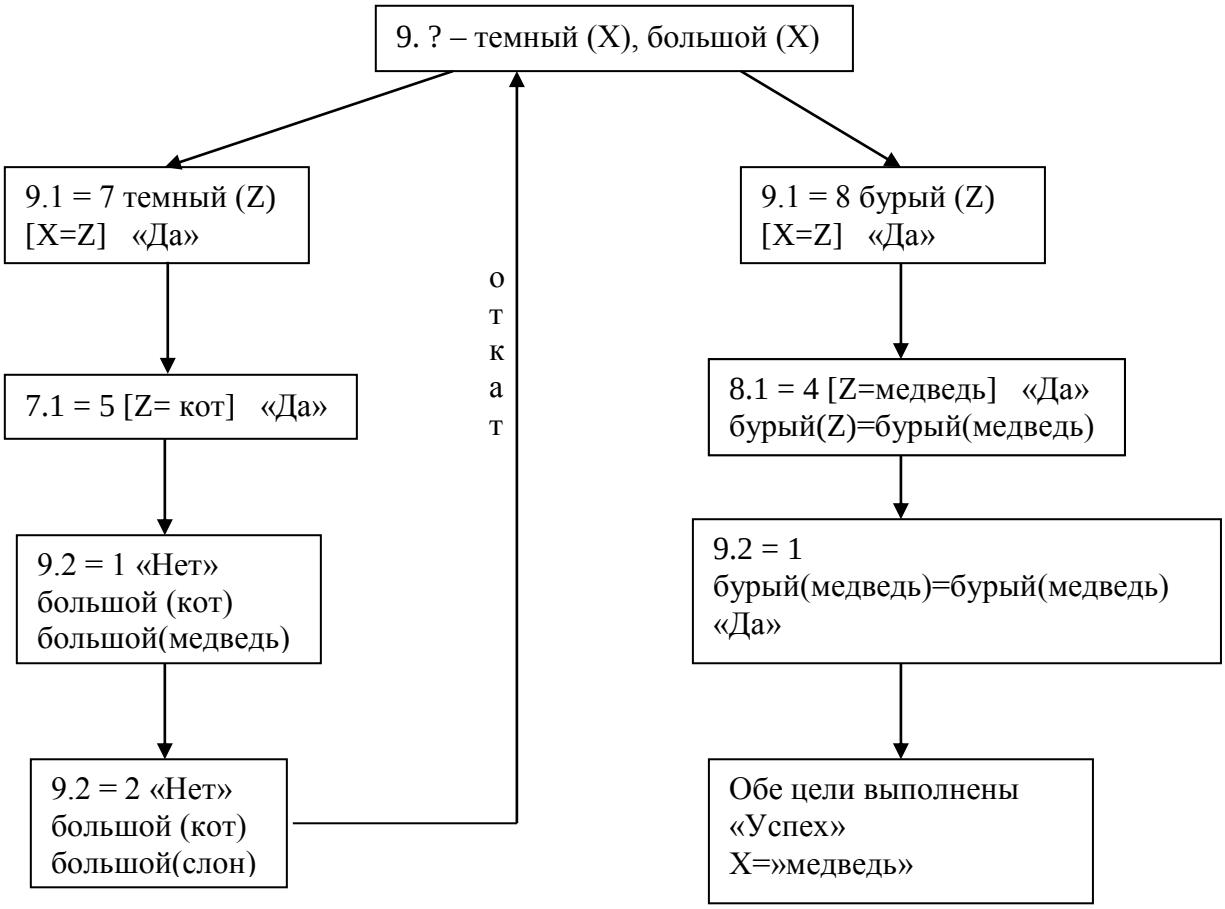
8. темный(Z):-бурый(Z).

8.1

9. ?-темный (X),большой (X).

9.1 9.2

Предложения пронумерованы для удобства. Составим схему вычисления поставленной цели (9).



Таким образом, для вычисления целей потребовалось 7 сопоставлений и один откат.

Формальное описание процедуры вычисления целей. Пусть дан список целей $G_1, G_2, ..., G_m$:

1. Если список целей пуст, вычисление дает успех; если нет, то выполнятся пункт 2.

2. Берется первая цель G_1 из списка. *Пролог* выбирает в базе данных, просматривая сначала первое предложение С, С: Н :- В1, В2,..., Вn., голова которого сопоставляется с целью G_1. Если такого предложения нет, то неудача. Если есть, то переменные конкретизируются и цель G_1 заменяется на список целей с конкретизированными значениями переменных.

3. Рассматривается рекурсивно через п.2 новый список целей. В1, В2,…,Вn, G_2, ..., G_m. Если С – факт, то новый список короче на одну цель (n=0). Если вычисление нового списка оканчивается успешно, то и исходный список

целей выполняется успешно. Если нет, то новый список целей отбрасывается, снимается конкретизация переменных и происходит возврат к просмотру программы, но начинал с предложения, следующего за предложением С. Описанный процесс возврата называется бэктрекинг (backtracking).

Составление запроса. Простейшая *Пролог*-программа - это множество фактов, которое неформально называют базой данных. Рассмотрим пример базы данных, состоящей из фактов «знает»:

знает (катя, сергей).

знает (костя, сергей)

знает (костя, марина)

После того как база знаний составлена, можно написать *запрос к* ней. *Простой запрос* состоит из имени предиката, за которым располагается список аргументов (синонимом *слова запрос* является слово *цель).* Приглашение ? - означает, что интерпретатор готов к обработке запроса.

Запросы с константами. Простейшей формой запроса является запрос верно, требующий подтверждения некоторого факта. Будем объяснять этот и другие запросы с помощью вопросов на естественном языке. После вопроса приводим его эквивалент на *Прологе* и даваемые системой *Пролог* ответы.

Верно, что Катя знает Сергея?

? знает (катя, сергей)

да

Запрос верно спрашивает, имеется ли данное высказывание в базе банных. Искомое высказывание должно быть заключено в скобки. В данном случае высказывание в запросе совпало с высказыванием знает (катя, сергей) из базы данных, поэтому система ответила «ДА», что является сокращением от «Да, факт подтвердился».

Верно, что Катя знает Костю?

? - верно (катя, костя)

нет

В данном случае высказывание в запросе не совпало ни с одним из высказываний, содержащихся в текущей базе данных, поэтому система ответила «нет» - сокращение от «Нет, факт не подтвердился».

Запросы с переменными. Переменная - это вид терма, который записывается как слово, начинающееся с большой буквы, - к примеру, X или «Кто». Если в запрос входят переменные, то интерпретатор попытается отыскать такие их значения, при которых запрос будет истинным. Запрос

?-знает (Катя, X).

означает: «Кого знает Катя?»

Квантификация переменной в запросе. Считается, что переменные в *Прологе* квантифицированы экзистенциально. Это означает, что в запросе, содержащем переменную неявно, спрашивается о том, существует ли хотя бы одно значение этой переменной, при котором запрос будет истинным. Если иметь это в виду, то приведенный выше запрос можно прочесть так: "Существует ли хотя бы один человек, которого знает Катя?". Этот запрос будет истинным, если такое лицо будет найдено в текущей базе данных "знает*.

Выполнение запроса. Интерпретатор пытается унифицировать (т. е. согласовать) аргументы запроса с аргументами фактов, входящих в базу данных "знает". В рассматриваемом случае запрос успешен при сопоставлений запроса с первым же фактом, поскольку атом "Катя" в запросе унифицируется с атомом "Катя" в этом факте, а переменная "Кто" унифицируется с атомом "сергей", входящим в факт. В результате данного процесса переменная "Кто" примет значение атома "сергей", и интерпретатор ответит: Кто = Сергей

Конкретизация. Говорят, что переменная конкретизируется, когда при выполнении запроса она унифицируется с некоторым значением.

Более чем один ответ. Следующий запрос предназначен для выдачи имен всех лиц, знающих друг друга в соответствии с фактами, содержащимися в базе данных "знает".

? - знает(X,Y).

X = катя, Y=сергей;

X = костя, Y=сергей;

X = костя, Y=марина;

No.

Интерпретатор нашел три ответа на запрос. Последний ответ - нет - означает, что интерпретатор достиг конца базы данных "знает" и больше не в состоянии отыскать еще какие-либо ответы.

Порядок выдачи ответов. Интерпретатор находит ответы а базе данных "знает" точно а том порядке, а каком факты вводились в базу. После отыскания первого ответа на запрос интерпретатор запоминает положение этого ответа в базе данных "знает". Если пользователь введет символ ;, то интерпретатор начнет поиск нового ответа со следующей фразы "знает". Но если пользователь нажмет клавишу возврата каретки (return), то интерпретатор "забудет" положение последнего ответа в базе данных "знает* и вернется к сообщению - подсказке верхнего уровня, ?–.

Продолжительность существования переменных. Запрос существует в интервале времени от момента, когда пользователь наберет текст запроса и нажмет клавишу "возврат каретки", до момента, когда на терминале снова появится сообщение – подсказка верхнего уровня. Продолжительность существования любой переменной, входящей а запрос, будет той же, что и у самого запроса. После того как интерпретатор выполнит запрос и вернется к сообщению – подсказке верхнего уровня, переменные этого запроса прекратят существование.

Составные запросы. Все приведенные выше запросы были простыми запросами. Составной запрос образуется из нескольких простых запросов, соединенных между собой запятыми. Каждый простой запрос, входящий в составной, называется "подцелью". Для того чтобы составной запрос оказался истинным, необходимо, чтобы каждая из его подцелей была истинной. К примеру, запрос

? - знает (катя, X), знает (костя, X).

соответствует вопросу: *"Есть ли такой человек, которого знают одновременно*

и Катя, и Костя?" Одна и та же переменная X входит в обе подцели этого запроса, а это означает, что для истинности всего составного запроса вторые аргументы обеих подцелей должны принимать одно и то же значение.

Интерпретатор ответит:

A = Сергей;

нет

Первый ответ, *A* = Сергей, показывает, что Сергея знают одновременно и Катя, и Костя. Второй ответ, *нет,* свидетельствует о том, что Сергей – это единственный их общий знакомый.

Аргументы как входные и выходные параметры. Выдача запроса – это единственный способ дать команду интерпретатору *Пролога* на выполнение программы. Использование константы в качестве аргумента запроса (скажем, констант "катя" или "костя" в последнем примере) эквивалентно заданию *входного параметра* программы. Любой положительный ответ должен унифицироваться с константой. Использование переменной в качестве аргумента запроса (скажем, переменной X в последнем примере) эквивалентно требованию получить от программы *выходные данные.*

Переменная – Символ подчеркивания _ выступает в качестве *анонимной переменной,* которая предписывает интерпретатору проигнорировать значение аргумента. Анонимная переменная унифицируется с чем угодно, но не обеспечивает выдачу на терминал выходных данных. Каждая переменная _, входящая в запрос, отличается от всех других переменных этого запроса.

Посмотрим, к примеру, что произойдет, если ввести запрос:

? – знает (X,_).

При выполнении этого запроса будут выданы все возможные значения первого аргумента предиката "знает", вне зависимости от того, какие значения будет принимать второй аргумент.

X=катя;

X=костя;

X=костя;

нет

Унификацией называется отожествление объектов PROLOG при сопоставлении предиката с фактами и правилами базы данных. Если такое отожествление выполнено, то говорят, что унификация завершена успешно. При таком сопоставлении свободные переменные доказываемого предиката приобретают конкретные значения. Этот процесс называется *конкретизацией.*

Пример: Пусть имеется база данных:

пишет(«Петров», портреты).

пишет(«Репин», картины).

пишет(«Суриков», картины).

пишет(«Блок», стихи).

пишет(«Аксенов», романы).

художник(X):-пишет(X, портреты).

художник(X):-пишет(X, картины).

писатель(X):- пишет(X, стихи).

писатель(X):- пишет(З, романы).

Сделаем запрос: «Кто является писателем?»: писатель(X).

Задание на лабораторную работу

Последовательность действий:

1. В соответствии с вариантом задания, определенным преподавателем, составить Пролог-программу задания.

2. Оформить отчет с указанием варианта задания, правил, текста программы и протокола выполнения программы.

Варианты заданий

2. Опишите увлечения студентов вашей группы.

3. Опишите успеваемость вашей группы (дайте определение «отличник», «хорошист»).

4. Создайте базу данных из высказываний, описывающих страны разных частей света, с помощью следующего словаря:

Имена объектов:

Вашингтон	США	Америка
Оттава	Канада	Европа
Лондон	Соединенное королевство	Африка
Рим	Италия	Европа
Лагос	Нигерия	Африка
Париж	Франция	Европа

Имена отношений:

столица_государства, страна_части_света.

Ваша база данных, например, должна содержать такие высказывания: Вашингтон – столица_государства США, США – страна_части_света Америка.

5. Создайте базу данных из простых высказываний, описывающих книги разных жанров, написанные различными людьми. Воспользуйтесь следующим словарем:

Имена объектов:

Том Сойер	Марк Твен	Роман
По ком звонит колокол	Эрнест Хемингуэй	Пьеса
Ромео и Джульетта	Шекспир	Драма

Имена отношений:

жанр, автор, писатель.

В вашей базе данных должны быть, например, такие высказывания: Том Сойер - автор Марк Твен: Том Сойер - жанр роман: Марк Твен - писатель.

6. Создайте базу данных, описывающую устройство велосипеда, воспользовавшись следующим словарем:

Имена объектов:

велосипед	колесо	педали

электропривод	седло	рама
тормозная система	фара	руль
тормозной трос	втулка	шестеренки
переключатель скоростей	цепь	спица

Имена отношений:

часть_объекта.

В вашей базе данных должны быть, например, такие высказывания:

колесо - часть_объекта велосипед: спица - часть_обьекта колесо: втулка - часть_объекта колесо т.д.

7. Создайте базу данных из высказываний, описывающих страны разных частей света, с помощью следующего словаря:

Имена объектов:

Вашингтон	США	Америка
Оттава	Канада	Европа
Лондон	Соединенное королевство	Африка
Рим	Италия	Европа
Лагос	Нигерия	Африка
Париж	Франция	Европа

Представьте следующие вопросы в виде запросов на *Прологе**

а) верно, что Рим - столица Франции?

б) верно, что Вашингтон - столица государства в Европе?

в) какие города являются столицами стран, находящихся в Европе?

г) имеется ли запись о столице Индии?

д) столицы каких государств в Америке известны системе?

е) в каких частях света находятся государства, столицы которых известны системе?

8. Создайте базу данных из простых высказываний, описывающих книги разных жанров, написанные различными людьми. Воспользуйтесь следующим словарем:

Имена объектов:

Том Сойер	Марк Твен	Роман
По ком звонит колокол	Эрнест Хемингуэй	Пьеса
Ромео и Джульетта	Шекспир	Драма

Ответьте на следующие запросы на *Прологе* и объясните смысл каждого из них:

а) верно (Шекспир автор Ромео и Джульетта)

б) верно (X автор Марк Твен и X жанр роман)

в) какие (X Y: X жанр пьеса и X автор Y)

г) какие (X: X жанр роман и X авторY)

д) какие (X: Y автор X)

9. Создайте базу данных, описывающую устройство велосипеда, воспользовавшись следующим словарем:

Имена объектов:

велосипед	колесо	педали
электропривод	седло	рама
тормозная система	фара	руль
тормозной трос	втулка	шестеренки
переключатель скоростей	цепь	спица

Представьте следующие вопросы на *Прологе:*

а) из каких частей состоит велосипед?

б) верно, что генератор постоянного тока является частью велосипеда?

в) верно, что спица является частью чего-то?

г) частью какой части велосипеда является генератор постоянного тока?

д) из каких частей состоит тормозная система?

10 . Составить на языке *Пролог* следующую программу:

Амур - это собака

Флэш - это собака

Джерри - это кошка

Стар - это лошадь

Флэш черная

Джерри коричневая

Амур рыжая

Стар белая

X - домашнее животное, если либо X - это собака или X - это кошка.

X - это животное, если либо X - это лошадь или X - домашнее животное.

Том владеет X, если X - это собака и X не черного цвета.

Кейт владеет X, если либо X черного цвета или X - это лошадь.

Составьте запросы, позволяющие определить:

а) всех, кто владеет животными;

б) всех, кто владеет животными не белого цвета;

в) того, кто владеет Джерри;

г) клички тех животных, которыми кто-то владеет, и имена владельцев.

Контрольные вопросы

1. В чем состоят принципиальные различия процедурных и декларативных языков программирования?

2. Каковы этапы программирования на *Прологе*?

3. Какие типы данных допускает *Пролог*?

4. В чем существо операции сопоставления?

5. Как реализуются вопросы к программе на *Проло*

6. В чем существо операции сопоставления?

7. Как реализуются вопросы к программе на *Прологе?*

8. В чем заключается процесс унификации на *Прологе?*

9. Как происходит квантификация переменной в запросе?

10. Как составляются составные запросы на *Прологе?*

Лабораторная работа № 2

РЕКУРСИЯ

Цель работы: изучить понятие рекурсии и способы построения рекурсив-

ных процедур в Прологе, разработать программу с использованием рекурсии.

Краткие теоретические сведения

Рекурсия – это второе средство для организации повторяющихся действий в Prologe. *Рекурсивная процедура* – это процедура, вызывающая сама себя до тех пор, пока не будет соблюдено некоторое условие, которое остановит рекурсию. Такое условие называют граничным. Рекурсия – хороший способ для решения задач, содержащих в себе подзадачу такого же типа. Правило является *рекурсивным,* если оно принадлежит процедуре, включающей вызов самой себя в виде подцели, содержащейся в теле по крайней мере одного из ее утверждений.

В общем случае рекурсивное правило имеет следующий вид :

recursive_rule(<фактические параметры через запятую>): –<предикаты и правила>,recursive_rule(<фактические параметры рекурсивного вызова>).

Классическим примером рекурсивного определения в Прологе может служить процедура «предок», которая состоит из двух правил:

предок(X,Y):-родитель(X,Y).

предок(X,Y):-родитель(Z,Y), предок(X,Y).

Совокупность этих правил определяет два способа, в соответствии с которыми одно лицо (X) может быть предком другого лица (Y). Согласно первому правилу, X является предком Y, если X – родитель Y, т.е. X является ближайшим предком Y.

Согласно второму правилу. X будет предком Y, если есть некоторый Z, который, являясь родителем Y, имеет своим предком X. Другими словами, X – предок Y, если X – предок родителя Y, т.е. X – отдаленным предком Y. Таким образом второе правило зависит от более простой версии самого себя, т.е. от подцели «предок».

Примером рекурсивных вычислений является известный алгоритм вычисления факториала. На Прологе эта программа может иметь такой вид:

Domains

```
  N,F=real
Predicates
  factorial(N,F)
Clauses
  factorial(1,1).
  factorial(N,R):- N>0, N1=N-1,
      factorial(N1,R1), R=R1*N.
Goal
  factorial(8,F),write(F).
```

При каждом вызове дизъюнкта *factorial* генерируются новые переменные, которые действуют всегда только на своем уровне вложенности, пока не встретится условие прекращения вычислений. Только после этого на обратном пути прохождения рекурсии определяются результаты, которые передаются вверх.

Вообще, любая рекурсивная процедура должна содержать хотя бы одну из двух компонент:

1. Нерекурсивную фразу, определяющую правило, применяемое в момент прекращения рекурсии.

2. Рекурсивное правило, первая подцель которого вырабатывает новые значения аргументов, а вторая – рекурсивная подцель – использует эти значения.

База правил просматривается сверху вниз. Сначала делается попытка выполнения нерекурсивной фразы. Если условие окончания рекурсии не указано, то правило может работать бесконечно.

Любое рекурсивное определение содержит по крайней мере одно нерекурсивное правило и одно или несколько правил с рекурсией. В большинстве случаев имеется по одному правилу каждого типа. Считается, что используется *хвостовая рекурсия,* если последнее условие в последнем правиле является рекурсивным. Такая рекурсия имеет преимущество перед нехвостовой рекурсией, так как позволяет ограничить рост стека и строго контролировать процесс возврата. Это происходит благодаря очистке стека после успешного сопоставления

условия, содержащего рекурсию.

Начинающему пользователю бывает довольно трудно понять, каким образом выполняется рекурсивная процедура, будучи вызванной в качестве цели. Поэтому в последующем разделе на примере работы со списками будет продемонстрировано выполнение некоторых рекурсивных процедур.

Пример решения задачи «Ханойская башня» на ПРОЛОГе.

```
DOMAINS
loc =right;middle;left
PREDICATES
hanoi(integer)
move(integer,loc,loc,loc)
inform(loc,loc)
GOAL
hanoi(5).
CLAUSES
hanoi(N):- move(N,left,middle,right).
move(1,A,_,C):- inform(A,C),!.
move(N,A,B,C):- N1=N-1, move(N1,A,C,B), inform(A,C),
move(N1,B,A,C).
inform(Loc1, Loc2):- nl,write("Диск с", Loc1, " на ", Loc2).
```

Варианты заданий

Вариант 1. Подсчитать, сколько раз встречается некоторая буква в строке. Строка и буква должны вводиться с клавиатуры. Для разделения строки на символы использовать стандартный предикат frontchar (String, Char, StringRest), позволяющий разделять строку String на первый символ Char и остаток строки StringRest.

Вариант 2. Вычислить значение n-го члена ряда Фибоначчи: f(0)=0, f(1)=1, f(n)=f(n-1)+f(n-2).

Вариант 3. Вычислить произведение двух целых положительных чисел (используя суммирование).

Вариант 4. Подсчитать, сколько раз встречается некоторое слово в строке. Строка и слово должны вводиться с клавиатуры. Для разделения строки на слова использовать стандартный предикат fronttoken (String, Lexeme, StringRest), позволяющий разделить строку String на первое слово Lexeme и остаток строки StringRest.

Вариант 5. Поменять порядок следования букв в слове на противоположный. Для разделения строки на символы использовать стандартный предикат frontchar (String, Char, StringRest), позволяющий разделять строку String на первый символ Char и остаток строки StringRest.

Вариант 6. Вычислить сумму ряда целых нечетных чисел от 1 до n.

Вариант 7. Поменять порядок следования слов в предложении на противоположный. Для разделения строки на слова использовать стандартный предикат fronttoken (String, Lexeme, StringRest), позволяющий разделить строку String на первое слово Lexeme и остаток строки StringRest..

Вариант 8. Вычислить сумму ряда целых четных чисел от 2 до n.

Вариант 9. Организовать ввод целых положительных чисел и их суммирование до тех пор, пока сумма не превысит некоторого порогового значения. Введенные отрицательные целые числа суммироваться не должны.

Вариант 10. Организовать ввод букв и их соединение в строку до тех пор, не будет введен символ #. Для присоединения символа к строке использовать стандартный предикат frontchar (String, Char, StringRest), позволяющий присоединять символ Char к строке StringRest и получать строку String.

Вариант 11. Написать рекурсивную программу вычисления *n-го* члена геометрической прогрессии, суммы ее *n* первых членов и суммы ее членов, начиная с *i*-го по *к-й.*

Вариант 12. Описать рекурсивную функцию вычисления максимального числа Фибоначчи, ближайшего к заданному *n* по недостатку.

Вариант 13. Написать подпрограмму-функцию степени a^x, где a, x – любые числа. Воспользуемся формулой: $a^x = e^{x \ln a}$

Вариант 14. Используя рекурсию составить программу вычисления сум-

мы. Значения k и n ввести с клавиатуры. $s = \sum_{k=1}^{n} k(k+1)x^k$.

Вариант 15. Написать программу вычисления суммы n первых членов бесконечного ряда $\sum_{i=1}^{\infty} \frac{n}{n!}$.

Вариант 16. Написать программу для вычисления чисел Фибоначчи для ряда, заданного списком.

Вариант 17. Написать программу для вычисления среднего арифметического списка чисел.

Вариант 18. Написать программу для генерации ряда целых чисел от M до N в порядке возрастания.

Вариант 19. Написать программу для генерации ряда целых чисел от M до N в порядке убывания.

Вариант 20. Написать программу, которая бы воспринимала целые числа с клавиатуры и вычисляла сумму введенных десятичных чисел. Программа должна завершаться при вводе числа 0.

Отчет по лабораторной работе должен содержать: титульный лист с указанием номера варианта; текст задания; исходные тексты программы с комментариями.

Контрольные вопросы:

1. Что такое рекурсия? Привести примеры.
2. Дать рекурсивную формулировку понятиям «предок» и «потомок».
3. Дать определение следующего понятия: рекурсия, базис рекурсии, шаг рекурсии.
4. Назначение предиката fail.
5. рганизация рекурсивных вычислений в Прологе.

Лабораторная работа № 4

СПИСКИ И АЛГОРИТМЫ СОРТИРОВКИ СПИСКОВ.

Цель работы: Освоение фундаментальных операций на списках: вхождение отдельного элемента в список, вывод элементов списка, соединение двух списков, добавление элемента к списку. Применение рекурсивных процедур для обработки списков. Применение отсечения при работе со списками.

Краткие теоретические сведения

Работа со списками. Список – это специальный вид сложного терма, состоящий из головы списка и хвоста списка, где голова – элемент списка, а хвост рекурсивно определяется как остаток списка. Элементами списка могут быть любые объекты, в т. ч. тоже списки.

Синтаксически список задается следующим образом: [X|Y], где X – это голова списка, Y – хвост. При построении списков необходимо наличие константного символа, чтобы рекурсия не была бесконечной. Этот "пустой список" будем обозначать знаком [].

Точное описание списка в виде фрагмента программы имеет вид:

list([]).

tist([X|Y]):-list(Y).

Декларативная интерпретация: списком является либо пустой список либо значение функции соединения на паре, хвост которой – список.

Например, список [a,Ь,c] является примером терма [X|Y] при подстановке {X=a,Y=[b,c]}.

Рекурсивные процедуры служат наиболее удобным средством для выполнения действий с каждым из элементов списка. Рекурсивная процедура может работать со списком любой длины, что освобождает программиста от необходимости заранее знать точную длину обрабатываемого списка. Рекурсивная процедура, выполняющая обработку списков, имеет те же обычную структуру, что и любые рекурсивные процедуры. Вначале располагаются фразы, определяющие условия окончания рекурсии, а затем следует фраза, выполняющая действия с заголовком списка, которая далее вызывает рекурсивно сама себя с

аргументом, которым служит остаток списка.

Определение списка через его голову и хвост в сочетании с рекурсией лежит в основе большого числа программ, оперирующих списками. Эти программы состоят:

1) из факта, ограничивающего рекурсию и описывающего операцию для пустого списка;

2) из рекурсивного правила, определяющего операцию над списком, состоящим из головы и хвоста (в голове правила), через операцию над хвостом (в подцели).

В процессе вычисления цели *Пролог* проводит перебор вариантов в соответствии с установленным порядком. Цели выполняются последовательно, одна за другой, а в случае неудачи происходит *откат к предыдущей цели (backtracing).*

Однако для повышения эффективности выполнения программы часто требуется вмешаться в перебор, ограничить его, исключить некоторые цели. Для этого в *Пролог* введена специальная конструкция *cut – "отсечение",* обозначаемая как "!" (Читается *"cut", "кат"). Cut* подсказывает *Прологу* "закрепить" все решения, предшествующие появлению его в предложении. Это означает, что если требуется бэктрекинг, то он приведет к неудаче *(fail)* без попытки поиска альтернатив.

Пример действия cut. Пусть база данных выглядит следующим образом:

```
data(one).
data (two).
data(three).
```

Процедура для проверки:

```
cut_test_a(X):-data(X).
cut_test_a(‘last clouse’)
```

Цель: ?- cut_test_a(X),nl,write(X).

```
one;
two;
```

three;

‘last clouse’;

no.

Теперь поставим *cut* в правило:

cut_test_b(X):-data(X),!.

cut_test_b(‘last clouse’)

Цель: ?- cut_test_b(X),nl,write(X).

one;

no.

Происходит остановка бэктрекинга для левого *data* и родительского:

cut_test_b(X)..

Теперь поместим *cut* между двумя целями:

cut_test_c(X,Y):-data(X),!,data(Y).

cut_test_c(‘last clouse’)

Теперь бэктрекинг не будет для для левого data и родительского cut_test_b(X), но будет для правого data, стоящего после !.

?- cut_test_c(X,Y),nl,write(X,Y).

one-one;

one-two;

one-tree;

no

Введение отсечения повышает эффективность программы, сокращая время перебора и объем памяти, не влияет на декларативное чтение программы. После изъятия отсечения декларативный смысл не измениться – такое применение отсечения называют «зеленым отсечением». «Зеленые отсечения » лишь отбрасывают те пути вычисления, которые не приводят к новым решениям. Бывают «красные отсечения», при их изъятии меняется декларативный смысл программы.

Формальное описание действия отсечения. Рассмотрим предложение H:-B1, B2,…..., Bm, !,….,Bn. Это предложение активизируется, когда некоторая

цель G будет сопоставляться с H. Тогда G называют целью-родителем. Если B1, B2,..., Bm выполнены, а после! (например, в Bi, i>m) произошла неудача и требуется выбрать альтернативные варианты, то для B1, B2,..., Bm такие альтернативы больше не рассматриваются ч все выполнение окончится неудачей. Кроме того, G будет связана с головой H и другие предложения процедуры во внимание не принимаются. Т.е. отсечение в теле предложения отбрасывает все предложения, расположенные после этого предложения. Формально действие отсечения описывается так: пусть цель-родитель сопоставляется с головой предложения, в теле которого содержится отсечение. Когда при просмотре целей тела предложения встречается в качестве цели отсечение, то такая цель считается успешной и все альтернативы принятым решениям до отсечения отбрасываются, а любая попытка найти новые альтернативы на промежутке между целью-родителем и он оканчиваются неудачей.

Процесс поиска возвышается к последнему выбору, сделанному перед сопоставлением цели-родителя.

Замечания пои использовании отсечения. Применение отсечения за счет сокращения перебора позволяет повысить эффективность программы. Кроме того, отсечение упрощает программирование выбора вариантов. Вместе с тем часто при использовании красных отсечений может измениться декларативный смысл программы. Поэтому отсечение требует осторожности в использовании.

Следует избегать двух ошибок:

отсечения путей вычисления, которые нельзя отбрасывать;

неотсечения тех решений, которые должны были быть отброшены.

Рассмотрим программу:

```
B.
D.
A:-B,C.    (1)
C:-D,!,E.  (2)
E:-F,G,H.  (3)
?A.
```

Говорят, что дизъюнкт (1) «порождает» дизъюнкт (2), так как в правой части (1) есть литера C и эта же литера – в левой части (2). Аналогично дизъюнкт (2) «порождает» дизъюнкт (3). Если (3) неудачен, то в (2) выполнится отсечение: дизъюнкт (2) также считается неудачным, восстанавливается «родительская среда» (1), делается попытка найти альтернативное решение для B. Если, бы (2) имело вид C:-D,E., то при неудаче в (3) была бы сделана попытка найти альтернативное решение для D.

В других случаях может стать необходимым продолжение поиска дополнительных решений, даже если целевое утверждение уже согласовано. В таких случаях можно использовать встроенный предикат *fail* Этот предикат не имеет аргументов. Он считается всегда ложным.

Рассмотрим некоторые стандартные операции над списками:

1.Фундаментальной операцией на списках является определение вхождения отдельного элемента в список. Фрагмент программы, рекурсивно определяющий данное отношение имеет вид:

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

Декларативный аспект программы:

X является элементом списка, если в соответствии с первым предложением X - голова списка, или в соответствии со вторым предложением X - элемент хвоста.

Процедурный аспект программы:

для определения того, является ли X элементом списка, нужно определить, совпадает ли X с головой списка и если не совпадает, искать X в хвосте списка.

Таким образом, member(X,Y) - отношение принадлежности к списку.

Например, member(4,[4,6,7]) - да.

member(8,[4,5,7,8,9])?

Работает второе правило. Список разбирается до тех пор, пока факт не будет истинен. Затем список собирается в обратном порядке.

2. Вывод элементов списка.

Первое правило: остановится, когда исчерпаны все элементы списка.

Второе правило: вывести элемент списка и работать далее с хвостом списка.

append([]).

append([X|Y]):-write(X),append(Y).

Например, append([3,6,8,10])?

Факт ложен - работает второе правило:

[3,6,8,10] 3 [6,8,10]

[6,8,10] 6 [8,10]

[8,10] 8 [10]

[10] 10 []

Факт истинен - список собирается путем присоединения головы списка к хвосту согласно левой части правила - [3,6,8,10].

3. Рассмотрим фрагмент программы, выражающей отношение между списками и числами. Предикат leng(L,N) истинен, если список L содержит N элементов.

leng([],0).

leng([X|L),N):-leng(L,N1),N=Nl+1.

Например, leng([3,4,5,7],Y)?

Факт ложен - работает второе правило.

[3,4,5,7], [4,5,7],

[4,5,7], [5 7],

[5,7], [7],

[7], [],

Факт истинен. Правило работает следующим образом:

[7],1 1=1

[5,7],2 2=2

[4,5,7],3 3=3

[3,4,5,7],4 4=4

Таким образом, Y=4.

4. Не менее важной операцией над списками является операция соединения двух списков для получения третьего. Определим предикат списка через три аргумента:

список(L1,L2,L3)

если L1 - пустой список, то результатом добавления L1 к L2 будет список L3.

На *Прологе* можно записать:

spicok([],LI,L1).

В противном случае формируется список L3, голова которого совпадает с годовой списка L1. Если первый аргумент отношения не пуст, то он имеет голову и хвост, и записывается это так:

[X|L1]

Таким образом можно показать соединение списка [X|L1] с произвольным списком L2, результат - список L3.

На *Прологе* можно записать так:

spisok([X|Ll],L2,[X|L3]):-spisok(Ll,L2,L3).

Если посмотреть на "spisok" с декларативной точки зрения, то мы, таким образом, описали отношение между тремя списками. Эти отношения сохраняются и в том случае, если известны L1 и L3, а L2 - нет, или известен только L3.

5. Добавление элемента к списку.

Наиболее простой способ добавить элемент в список - это вставить его в самое начало так, чтобы он стал его новой головой. Если X - это новый элемент, а список, в который X добавляется, - L, тогда результирующий список - просто [X|L].

Таким образом, чтобы добавить новый элемент в начало списка, не надо никакой процедуры. В явном виде это можно представить фактом:

добавить(X,L,[X|L]).

6. Выделение подсписка.

Это отношение имеет два аргумента - список L и список S, такой, что S содержится в L в качестве подсписка. Так отношение подсписок ([c,d,e],[a,b,c,d,e,f]) имеет место, а отношение подсписок([c,e],[a,b,c,d,e,f]) - нет.

Подсписок можно сформулировать так:

S является подсписком L, если

1) L можно разбить на два списка – L1 и L2;

2) L2 можно разбить на два списка - S и L3.

На *Прологе* это можно записать следующим образом:

```
podspisok(S,L):-spisok(L1,L2,L3),spisok(S,L3,L).
```

Фрагмент программы будет иметь вид:

```
spicok([],L1,Ll).
spisok([X|L1],L2,[X|L3]):-spisok(L1,L2,L3).
podspisok(S,L):-spisok(L1,L2,L),spisok(S,L3,L).
```

Например, podspisok([2,3],[1,2,3])?

7. Удаление элемента списка.

Для удаления элемента из списка требуются три аргумента: удаляемый элемент X, список LI, в который может входить X, и список L2, из которого удалены все вхождения элемента X.

Имеем два случая:

1) если X является головой списка, то результатом удаления будет хвост этого списка.

Процедурный аспект программы:

```
del([X|L],X,L).
```

Декларативный подход к фрагменту: "Удаление X из списка [X|L] приводит к L, если удаление X из L приводит к L1". Подходящим правилом является

```
del([X|L],X,L):-del(L,X,Ll).
```

Условие совпадения головы списка и удаляемого элемента задано с помощью общей переменной в заголовке правила;

2) если X находится в хвосте списка, тогда его нужно удалить оттуда:

```
del([Y|L],X,Y|L1):-del(L,X,L1).
```

Если в списке встречается несколько вхождений элемента X, то предикат удаления сможет исключить их все при помощи возвратов. Вычисление по ка-

ждой альтернативе будет удалять лишь одно вхождение X, оставляя остальные в неприкосновенности.

Например, del([3,5,3,5,7,3],3,Y)?

Y=[5,3,5_t7,3]

Y=[3,5,5,7,3]

Y=[3,5,3,5,7]

Декларативное понимание: «Удаление X из списка [Y|L] равно [Y|L1], если X отлично от Y и удаление X из L приводит к L1». В отличие от предыдущего правила условие неравенства головы списка и удаляемого элемента явно задано в теле правила.

del([Y|L],X,Y|L1):-X<>Y,del(L,X,L1).

Исходный случай задается непосредственно. Из пустого списка не удаляется ничего, результатом будет также пустой список. Это выражается фактом

del([],X,[]).

Полный фрагмент программы имеет вид:

del([],X,[]).

del([X|L],X,L):-del(L,X,L1).

del([Y|L],X,Y|L1):-del(L,X,L 1).

Значение программы содержит примеры, в которых удаляемый элемент вообще не входит в исходный список, - например, выполнено del([l,2,3],4,[1,2,3]). Существуют приложения, в которых это нежелательно. Определим отношение dell(X,L,Ll), в котором случай списка, не содержащего элемент X, рассматривается следующим образом:

dell([X|L],X,L).

dell([Y|L],X,[Y|L1]):-dell(L,X,L1).

Декларативное понимание программы: «Выделение элемента X из списка [X|L] приводит к L или выделение X из списка [Y|L] приводит к [Y|L1], если выделение X из списка L приводит к L1». Это отношение используется как вспомогательное отношение для программы сортировки списков.

8. Добавление элемента без дублирования.

Чтобы добавить элемент в голову списка, достаточно использовать

add(X,L,[X|L]).

Но если возникает необходимость добавлять только при отсутствии элемента, то можно добавить правило:

add(X, L, L):-member(X,L),!,add(X,L,[X|L]).

Вопрос ?-add(a, [b,c], L). L=[a, b, c]

?-add(b, [b, c], L). L=[b, c]

Если отсечение убрать, то

?-add(b, [b, c], L). L=[b, c]; L=[b, b, c]

Таким образом, при изъятии отсечения изменился декларативный смысл программы - отсечение «красное».

Задание на лабораторную работу

Последовательность действий:

1. В соответствии с вариантом задания, определенным преподавателем, составить Пролог-программу задания.

2 Оформить отчет с указанием варианта задания, правил, текста программы и протокола выполнения программы.

Варианты заданий

1. Создать случайным образом список состоящий из K нулей и K единиц.
2. Написать предикат PL(L+,N-) – истинный тогда и только тогда, когда N – предпоследний элемент списка L, имеющего не менее двух элементов.
3. Определите возведение в целую степень через умножение и деление.
4. Вставить подсписок в определенное место списка.
5. Удалить все заданные элементы из списка.
6. Сложить два списка.
7. Напишите предикат subst(+V, +X, +Y, -L) – истинный тогда и только тогда, когда список L получается после взаимной замены X на Y, т.е. X → Y, Y → X.

8. Напишите предикат, который определяет, является ли данное натуральное число простым.

9. Напишите предикат p(+N, +K, -L) – истинный тогда и только тогда, когда L – список всех последовательностей (списков) длины K из чисел 1, 2, …, N.

10. Напишите предикат p(+N, -L) – истинный тогда и только тогда, когда список L содержит все последовательности (списки) из N нулей и N единиц, в которых никакая цифра не повторяется три раза подряд (нет куска вида XXX).

11. Получить элемент под номером N в списке.

12. Объедините два списка, найдите MAX и удалите его.

13. Удалите из списка элемент, найдите длину оставшегося списка.

14. Добавьте элемент к списку, вычислите среднее арифметическое его элементов.

15. Определить максимальный элемент в списке.

16. Определить минимальный элемент в списке.

17. Определить количество одинаковых элементов в списке.

18. Определить число элементов в списке.

19. Определить произведение элементов списка.

20. Исключить из списка отрицательные элементы.

21. Выполнить сортировку элементов списка по возрастанию.

22. Даны два списка, имеющие ненулевое пересечение. Построить список, включающий все элементы указанных двух списков без повторений.

23. Определить отношение STPR (+X, –Y), где Y элементы списка в обратном порядке.

24. Определить отношение PRDS (+X, –Y), где Y перевод списка чисел от 0 до 9 в список соответствующих слов.

25. Написать программу вычисления скалярного произведения векторов inner_product(+X,+Y, –V), где X и Y – списки целых чисел, V – скалярное произведение этих списков.

26. Определить отношение PERES (+X,+Y, –V), где V – элементы списка чисел, являющимися общими для списков X и Y.

27. Определить отношение RAZN (+X,+Y, –V), где V – элементы списка чисел, принадлежат X, но не принадлежат Y.

28. Определить отношение element_mult(+X,+Y, –V), в котором элементы списка V равны произведениям соответствующих элементов списков X и Y.

29. Определить отношение shift(+X,–V), таким образом, чтобы список V представлял собой список X, "циклически сдвинутый" влево на один символ.

30. Треугольное число с индексом N – это сумма всех натуральных чисел до N включительно. Напишите программу, задающую отношение triangle(N,T), истинное, если T – треугольное число с индексом N.

Контрольные вопросы

1. Для чего служит предикат отсечения?
2. Какое отсечение считается зеленым?
3. Какое отсечение считается красным?
4. Каких ошибок следует избегать при применении отсечения?
5. Для чего служит предикат fail?
6. Для чего служат списки и как они задаются?
7. Какую роль выполняет предикат "!" при работе со списками?
8. Какова роль рекурсии при работе со списками?

Лабораторная работа № 5

СТРОКИ

Цель работы: Изучение приемов работы со строками в Прологе.

Краткие теоретические сведения

Под *строкой* в Прологе понимается последовательность символов, заключенная в двойные кавычки.

Prolog поддерживает различные стандартные предикаты для обработки

строк. Стандартные предикаты обработки строк делятся на две группы: базовое управление строками и преобразование строковых типов. Основными предикатами для работы со строками можно назвать предикат **frontchar** (String, Char, StringRest), позволяющий разделить строку String на первый символ Char и остаток строки StringRest и предикат **fronttoken** (String, Lexeme, StringRest), который работает аналогично предикату frontchar, но только отделяет от строки String лексему Lexeme.

fronttoken(String, Token, RestString)

(string,string,string) −(i,o,o) (i,i,o)(i,o,i) (i,i,i) (o,i,i)

Разделяет строку, заданную параметром **String**, на лексему **Token** и остаток **RestString** согласно поточному шаблону. Лексемой называется последовательность символов, удовлетворяющая следующим условиям: имя в соответствии с синтаксисом Prolog'a, число или отличный от пробела символ.

(i, i, o) fronttoken ("Go to cursor", X, Y) X="Go" Y="to cursor"

Предикаты обработки строк используются для разделения строк либо на список отдельных символов, либо на список заданных групп символов.

*Пример 1***.** Теперь попробуем применить рассмотренные предикаты. Создадим предикат, который будет преобразовывать *строку* в список символов. Предикат будет иметь два аргумента. Первым аргументом будет данная *строка*, вторым − список, состоящий из символов исходной *строки***.** Решение, будет рекурсивным. Базис: пустой *строке* будет соответствовать пустой список. Шаг: с помощью встроенного предиката frontchar разобьем *строку* на первый символ и остаток *строки*, остаток *строки* перепишем в список, после чего добавим первый символ исходной *строки* в этот список в качестве первого элемента. Запишем эту идею:

```
str_list("",[]). /* пустой строке соответствует пустой список */
str_list(S,[H|T]): – frontchar(S,H,S1), str_list(S1,T).
/* H − первый символ строки S, S1 − остаток строки */
/* T − список, состоящий из символов, входящих в строку S1*/
```

*Пример 2***.** Разработаем предикат, который будет преобразовывать список символов в *строку*. Предикат будет иметь два аргумента. Первым аргументом будет список символов, вторым − *строка*, образованная из элементов списка. Базис рекурсии: пустому списку соответствует пустая *строка*. Шаг: если исходный список не пуст, то нужно перевести в *строку* его хвост, после чего, используя стандартный предикат frontchar, приписывать к первому элементу списка *строку*, полученную из хвоста исходного списка. Запишем эту идею:

list_str([],""). /* пустой строке соответствует пустой список */ list_str([H|T], S):–list_str(T,S1), frontchar(S,H,S1).

/* S1 − строка, образованная элементами списка T */

/* S − строка, полученная дописыванием строки S1 к первому элементу списка H */

Встроенный предикат **str_len**, предназначен для определения *длины строки*, т.е. количества символов, входящих в *строку*. Он имеет два аргумента: первый − *строка*, второй − количество символов.

str_len(String, Length)

(string, integer) − (i,i) (i,o) (o,i)

(i, o) – с параметром длина связывается количество символов в строке

(i ,i) – выполняется успешно, если строка имеет указанную длину.

Следующий стандартный предикат **concat** предназначен, вообще говоря, для соединения двух *строк*, или, как еще говорят, для их *конкатенации*. У него три аргумента, каждый строкового типа, по крайней мере, два из трех аргументов должны быть связаны.

concat (Стр1, Стр2, Стр3) (string, string, string) : (i, i, o) (o, i, i) (i, o, i) (i, i, i)

(i, i, o) concat ("фут", "бол", X) X="футбол"

(o, i, i) concat (X, "ball", "football") X= "foot"

(i, i, i) concat ("foot", "ball", "football") True

frontstr(Lenght, Inpstring, StartString, RestString)

(integer, string, string, string) − (i, i, o, o)

Разделяет строку Inpstring на две части. StartString будет иметь длину

Lenght первых символов исходной строки, RestString представляет собой остаток строки InpString.

isname(StringParam)

(string) - (i)

Завершается успешно, если StringParam есть имя, удовлетворяющее синтаксису Турбо-Пролога.

Стандартные предикаты преобразования типа служат для преобразования символов с десятичным кодом ASCII, строк с отдельным символом, строк с целыми и действительными числами, а также строчных букв латинского алфавита с соответствующими прописными буквами.

Предикаты, предназначенные для преобразования типа:

char_int(CharParam,IntgParam)

(char,integer) − (i,o) (o,i) (i,i)

Преобразует символ в код ASCII, согласно поточному шаблону.

str_int(StringParam, IntgParam)

(string,integer) − (i,o) (o,i) (i,i)

Строка, представляющая целое десятичное число, преобразуется в это число.

str_char(StringParam, CharParam)

(string,char) − (i,o) (o,i) (i,i)

Один знак как строка преобразуетс в символ.

str_real(StringParam, RealParam)

(string,real) − (i,o) (o,i) (i,i)

Строка, представляющая десятичное число, преобразуется в это число.

upper_lower(StringInUpperCase, StringInLowerCase)

(string,string) - (i,i) (i,o) (o,i)

Строка, записанная прописными буквами, записывается в строку со строчными буквами.

upper_lower(CharInUpperCase, CharInLowerCase)

(char,char) − (i,i) (i,o) (o,i)

Прописной символ преобразовывается в строчный.

Каждый предикат преобразования типов имеет три варианта потока параметров; в случаях (i,o) и (o,i) выполняется соответствующие преобразования, а в случае (i,i) осуществляется проверка, которая завершается успешно, только если два аргумента являются различными представлениями одного и того же объекта.

Варианты заданий

Вариант 1. Создайте предикат, который будет находить последнюю позицию вхождения символа в строку.

Вариант 2. Создайте предикат, который подсчитает общее количество латинских букв в списке символов.

Вариант 3. Создайте предикат, который будет подсчитывать количество русских гласных букв в строке.

Вариант 4. Создайте предикат, находящий в исходной строке слово, в котором наибольшее количество русских гласных букв.

Вариант 5. Создайте предикат, который будет удалять из данной строки все вхождения заданного символа.

Вариант 6. Создайте предикат, удаляющий из данной строки все повторные вхождения символов.

Вариант 7. Создайте предикат, который продублирует вхождение каждого символа в строку.

Вариант 8. Создайте предикат, "переворачивающий" строку (меняющий в строке порядок символов на обратный).

Вариант 9. Создайте предикат, проверяющий, является ли данная строка палиндромом.

Вариант 10. Создайте предикат, составляющий список символов, которые входят одновременно в обе данных строки.

Вариант 11. Создайте предикат, находящий в исходной строке слово максимальной (минимальной) длины.

Вариант 12. Создайте предикат, преобразующий строку в список слов, состоящих из четного количества символов.

Вариант 13. Создайте предикат, преобразующий строку в список слов, которые упорядочены по длине.

Вариант 14. Создайте предикат, преобразующий строку в список слов, которые упорядочены в лексикографическом порядке.

Вариант 15. Создайте предикат, преобразующий исходную строку в строку, состоящую из первых букв слов первоначальной строки.

Вариант 16. Создайте предикат, преобразующий исходную строку в строку, состоящую из последних букв слов первоначальной строки.

Вариант 17. Создайте предикат, проверяющий правильность расстановки скобок в исходной строке.

Вариант 18. Создайте предикат, меняющий местами первую и последнюю буквы в каждом слове исходной строки.

Вариант 19. Дана строка символов. Преобразовать ее, удалив каждый символ * и повторив каждый символ, отличный от *.

Вариант 20. Дана строка символов, в которой есть двоеточие. Получить все символы, расположенные до первого двоеточия включительно.

Вариант 21. Дана строка символов. Выяснить, верно ли, что в строке имеются пять идущих подряд букв *e*.

Вариант 22. Дана строка символов. Определить число вхождений в строку группы букв *abc*.

Вариант 23. Дана строка символов. Группы символов, разделенные пробелами (одним или несколькими) и не содержащие пробелов внутри себя, будем называть словами. Найти количество слов, у которых первая и последняя буквы совпадают.

Вариант 24. Дана строка символов. Группы символов, разделенные пробелами (одним или несколькими) и не содержащие пробелов внутри себя, будем называть словами. Найти длину самого длинного слова.

Вариант 25. Дана строка символов. Группы символов, разделенные про-

белами (одним или несколькими) и не содержащие пробелов внутри себя, будем называть словами. Подсчитать количество букв «*а*» в последнем слове.

Отчет по лабораторной работе должен содержать: титульный лист с указанием номера варианта; текст задания; исходные тексты программы с комментариями.

Контрольные вопросы:

1 Как в Прологе описываются строки.

2 Какие стандартные предикаты используются для работы со строками в Прологе?

3 Для каких целей используется стандартный предикат *frontchar*? Приведите примеры его использования.

4 Назовите стандартные предикаты для базового управления строками.

5 Какие стандартные предикаты используются для преобразования строковых типов.

БИБЛИОГРАФИЧЕКИЙ СПИСОК

Адаменко А.Н. Логическое программирование и Visual Prolog / А.Н. Адаменко, А.М. Кучуков − СПб.: БХВ-Питер, 2003. − 992 с.

Акилова И.М. Логическое программирование: практикум / И.М. Акилова. – Благовещенск: Амурский гос. ун-т., 2002. – 40 с.

Акилова И.М. Программирование на языке Турбо-Пролог: практикум / И.М. Акилова. – Благовещенск: Амурский гос. ун-т., 2002. – 40 с.

Бессмертный, И.А.Искусственный интеллект: учеб. пособие – СПб: СПбГУ ИТМО, 2010. –132 с.

Братко И. Программирование на языке Пролог для искусственного ин-

теллекта. / Пер. с англ.М.: Мир, 1990.

Братко И. Алгоритмы искусственного интеллекта на языке PROLOG, 3-е издание: Пер. с англ. − М.: Издательский дом «Вильяме», 2004. − 640 с.

Ин П., Соломон Д. Использование Турбо-Пролога. / Пер. с англ. М.: Мир 1993.

Клоксин, У. Программирование на языке Пролог. / У. Клоксин, К. Меллиш – М.: Мир, 1987.

Марселлус Д. Н. Программирование экспертных систем на Турбо – Прологе. / Пер. с англ. М.: Финансы и статистика, 1994.

Новицкая Ю.В. Основы логического и функционального программирования: учеб. пособие. − Новосибирск: НГТУ, 2006. − 60 с.

Стобо Д.Ж. Язык программирования Пролог. / Пер.с англ. М.:Радио и связь,1993.

Функциональное и логическое программирование. Ч. 2. Логическое программирование: лабораторный практикум / Д.В. Михайлов, Г.М. Емельянов. – Великий Новгород: НовГУ им. Ярослава Мудрого, 2007. – 88 с.

Чанышев О.Г. Программирование в Логике: Учебное пособие. - Омск: Изд-во ОмГУ, 2004. – 64 с.

Шрайнер, П.А. Основы программирования на языке Пролог: курс лекций: учеб. пособие. – М.: Изд-во: Интернет-Университет информационных технологий, 2009, – 173 с.