МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»
Пятигорский институт (филиал) СКФУ

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

к выполнению лабораторных работ по дисциплине

Языки представления знаний

Направление подготовки 09.04.02 «Информационные системы и технологии»
профиль «**Технологии работы с данными и знаниями, анализ информации**»

Пятигорск 2025

СОДЕРЖАНИЕ

ПРЕДИСЛОВИЕ

Целью освоения дисциплины «Языки представления знаний» является формирование общепрофессиональных и профессиональных компетенций будущего магистра по направлению подготовки 09.04.02 «Информационные системы и технологии».

Основные задачи дисциплины:

– дать теоретические сведения о моделях представления знаний;
– дать целостное представление о различных логических средствах – языке, понятиях и приемах логических исчислений;
– сформировать умение выбирать и использовать методы решения задач искусственного интеллекта с использованием ПЭВМ;
– сформировать умение разрабатывать алгоритмы решения логических задач;
– сформировать умение использовать современные технологии и инструменты представления знаний для решения нестандартных задач.

Дисциплина «Языки представления знаний» относится к факультативным дисциплинам.

Знания, полученные в ходе изучения дисциплины «Языки представления знаний», могут быть использованы при подготовке к защите выпускной квалификационной работы и защите выпускной квалификационной работы.

В результате освоения дисциплины «Языки представления знаний» обучающийся должен:

ЗНАТЬ:

– методы, средства, алгоритмы решения нестандартных задач, в том числе в новой или незнакомой среде и в междисциплинарном контексте;
– алгоритмы и программные средства для решения профессиональных задач;
– методы анализа профессиональной информации;
– методы, средства, технологии разработки и исследования теоретических и экспериментальных моделей объектов профессиональной деятельности в различных областях;

УМЕТЬ:

– применять математические, естественнонаучные, социальноэкономические и профессиональные знания для решения нестандартных задач, в том числе в новой или незнакомой среде и в междисциплинарном контексте;
– разрабатывать оригинальные алгоритмы и программные средства для решения профессиональных задач;
– анализировать профессиональную информацию, выделять в ней главное, структурировать, оформлять и представлять в виде аналитических обзоров с обоснованными выводами и рекомендациями;
– уметь использовать инструментальные средства разработки и исследования теоретических и экспериментальных моделей объектов профессиональной деятельности в различных областях; ВЛАДЕТЬ:

– навыками применения математических, естественнонаучных, социально-экономических и профессиональных знаний для решения нестандартных задач, в том числе в новой или незнакомой среде и в междисциплинарном контексте;
– навыками применения современных интеллектуальных технологий, для решения профессиональных задач;
– методологией анализа профессиональной информации, оформления и представления в виде аналитических обзоров с обоснованными выводами и рекомендациями;
– методами, средствами, технологиями разработки и исследования теоретических и экспериментальных моделей объектов профессиональной деятельности в различных областях.

1. ИЗУЧЕНИЕ ОСНОВНЫХ ПРИЕМОВ РАБОТЫ В ПРОЛОГСИСТЕМЕ. СОЗДАНИЕ И ИСПОЛЬЗОВАНИЕ ПРОСТЫХ ОБЪЕКТОВ

Цель и содержание: ознакомление с основными приемами работы в системе Turbo Prolog 2.0, создания простых объектов в системе Turbo Prolog 2.0 и их использования.

Организационная форма занятий: решение проблемных задач, разбор конкретных ситуаций.

Вопросы для обсуждения на лабораторном занятии: примеры создания, редактирования и выполнения программы; сохранения файла; трассировки программы; примеры создания простых объектов в системе Turbo Prolog 2.0 и их использования.

Теоретическое обоснование

Запуск системы осуществляется файлом PROLOG.EXE, в результате чего на экране появляются четыре окна (рисунок 1.1):

– окно редактирования ***Editor*** (для ввода исходной программы);
– окно диалога ***Dialog*** (для ввода запросов и выдачи результатов);
– окно сообщений ***Message*** (для вывода информации об ошибках);
– окно трассировки ***Trace*** (для выдачи информации о выполнении программы).

Главное меню содержит команды:

– ***Files*** (позволяет управлять файлами; манипулировать каталогами; вызывать DOS и выходить из программы) (рисунок 1.2);
– ***Edit*** (вызывает встроенный редактор для изменения рабочего файла; если имя рабочего файла не задано в режиме ***Files***, то по умолчанию рабочим принимается WORK.PRO);

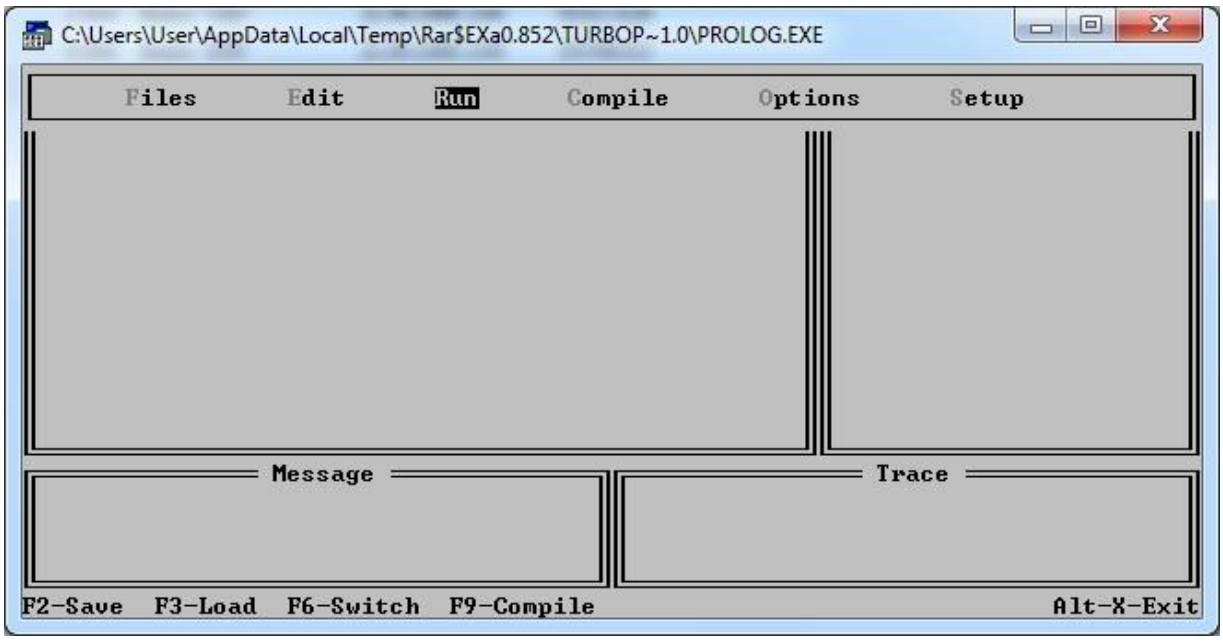

Рисунок 1.1 – Главное окно системы Turbo Prolog

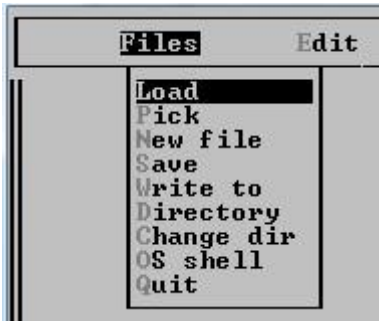

Рисунок 1.2 – Меню ***Files***

– ***Run*** (выполняет программу, содержащуюся в оперативной памяти); – ***Compile*** (управляет работой компилятора) (рисунок 1.3);

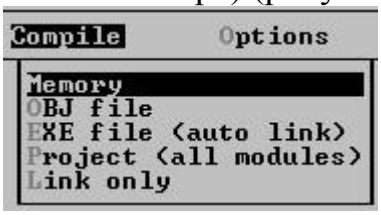

Рисунок 1.3 – Меню ***Compile***

– ***Options*** (позволяет задавать требуемые опции компилятора; опции компоновщика; имена библиотек и т. д) (рисунок 1.4);

Рисунок 1.4 – Меню ***Options***

– ***Setup*** (устанавливает среду разработки для Турбо Пролога: цвета и размеры окон; каталоги: текущий (Prolog), Turbo, Executable (исполняемый) и

Object (объектный); модифицирует конфигурацию клавиатуры и содержимое строк помощи в нижней части экрана; и т. д.) (рисунок 1.5).

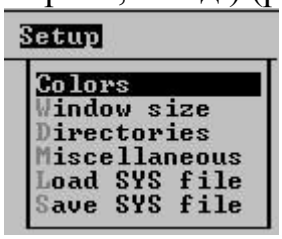

Рисунок 1.5 – Меню ***Setup***

Программа в системе Турбо Пролог обычно имеет следующую структуру:
domains /* определение типов данных */ ***global domains***
database /* определение предикатов динамической базы данных */
predicates /* определение предикатов */ ***global predicates***
goal /* определение цели */ ***clauses*** /* определение фактов и правил */ Содержимое раздела ***domains*** может отсутствовать.
Перед разделом ***clauses*** (или после него) может располагаться раздел ***goal*** (цель). Цель может состоять из нескольких подцелей. Если разрабатываемая программа предназначена для работы в пакетном режиме, раздел ***goal*** не может быть опущен.
В программе могут присутствовать два раздела, обеспечивающие межмодульный интерфейс (определение глобальных доменов и предикатов):
global domains global predicates

В разделе ***predicates*** определяются ***предикаты*** (отношения). Каждый предикат определяется со своим именем и своими ***аргументами*** (параметрами), которые должны быть описаны в разделе ***domains***.
В разделе ***clauses*** определяются ***факты*** и ***правила***. Факт представляется именем предиката, за которым следуют аргументы, заключенные в круглые скобки. Правило состоит из ***заголовка правила*** и ***тела правила***. Заголовок представляет собой предикат, тело состоит из термов, которые могут быть связаны между собой словами ***or*** (или) или ***and*** (и). Между телом и заголовком стоит слово ***if*** (если).
Имена переменных должны начинаться с прописной буквы и могут содержать только буквы, цифры или знак подчеркивания. Максимальная длина имени – 250 знаков.
Комментарии начинаются с последовательности символов /* и заканчиваются символами */ и могут располагаться в программе в произвольном месте.
Типы данных, используемых в системе Турбо Пролог, приведены в таблице 2.1

Таблица 2.1 – Типы данных

Тип	Описание
symbol	Последовательность букв, цифр и знаков подчеркивания, которая начинается со строчной буквы или заключена в кавычки

string	Последовательность символов, заключенная в кавычки
char	Отдельный символ, заключенный в апострофы
integer	Целое число в диапазоне от -32768 до 32767
real	Вещественное число
file	Имя файла

Данные типа ***symbol*** в отличие от данных типа ***string*** запоминаются в таблице символов, размещенной в оперативной памяти.

Аппаратура и материалы. Требования к техническому и программному обеспечению приведены в приложении В.

Указания по технике безопасности приведены в приложении Г.

Методика и порядок выполнения работы

Иллюстрацией создания, выполнения и редактирования Прологпрограммы служит пример 1.1, приведенный на рисунке 1.6.

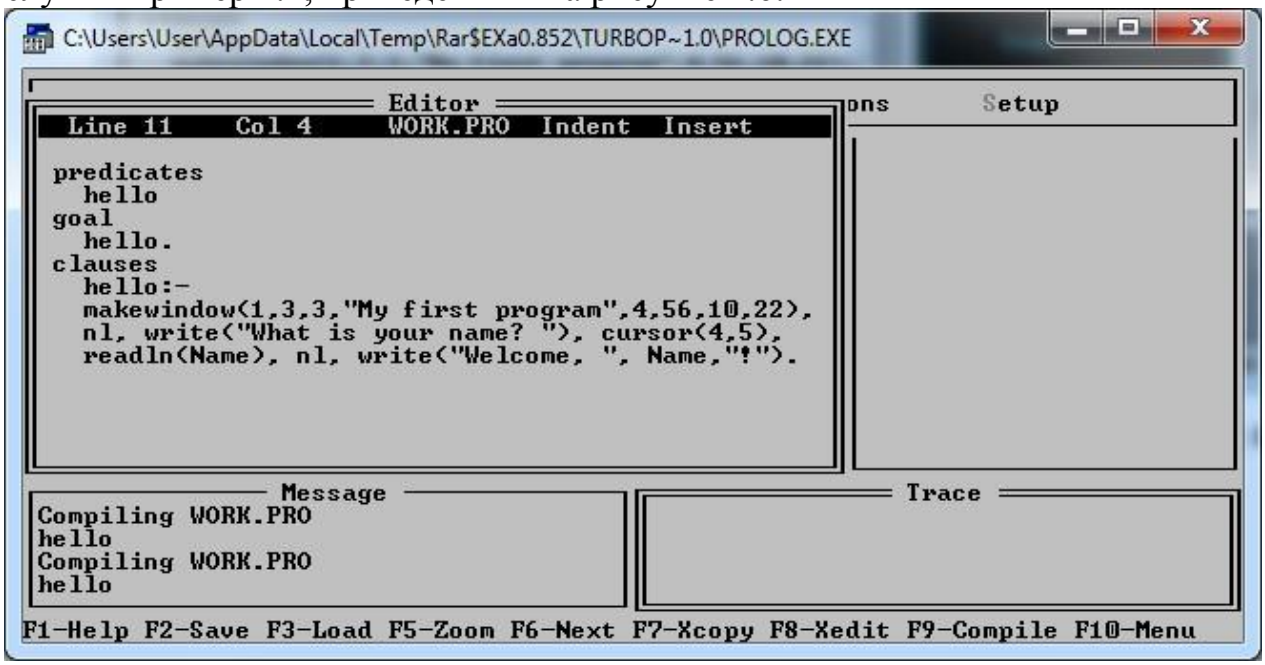

Рисунок 1.6 – Пример 1.1 (создание)

Вызов редактора производится движением курсора до слова ***Edit*** и последующим нажатием клавиши ***Enter***. Для выхода из редактора нужно нажать ***Esc***.

По требованию программы введите Ваше имя (например, Ivan) и нажмите ***Enter***. Программа ответит: «Welcome, [имя]!» и будет ожидать нажатия клавиши ***Spacebar*** (рисунок 1.7).

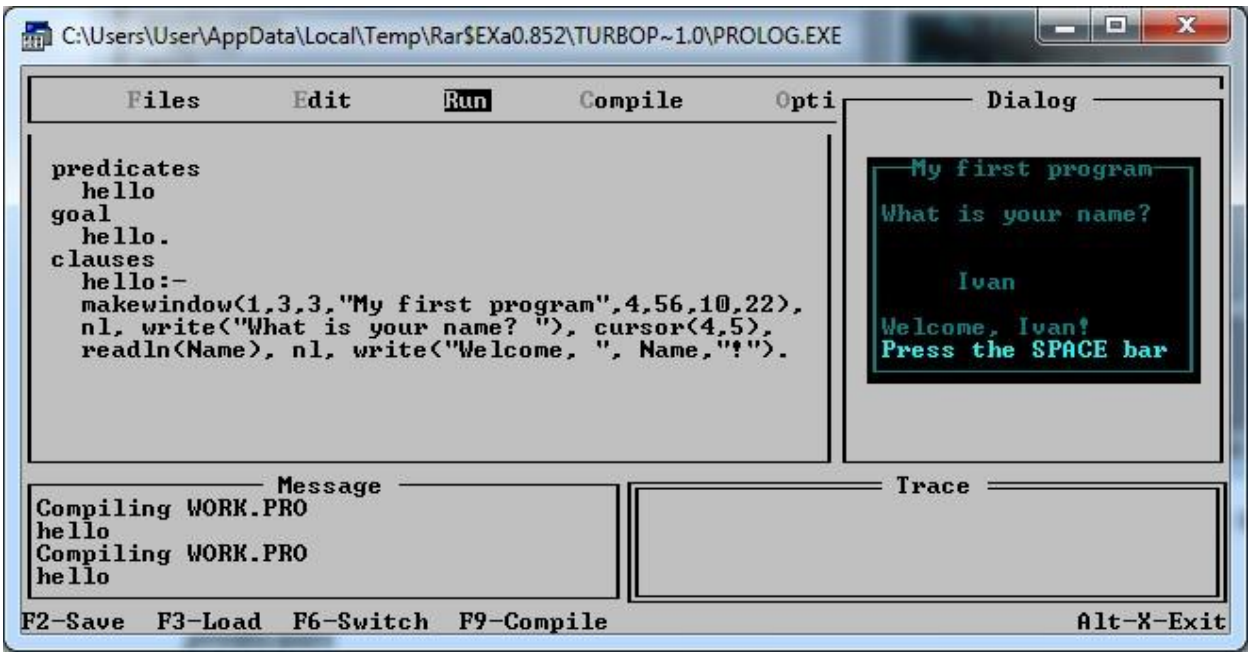

Рисунок 1.7 – Пример 1.1 (выполнение)

Сделайте две ошибки в рабочем файле, заменив второе слово *hello* на *howdy*, а третье – на *hi* (рисунок 1.8).

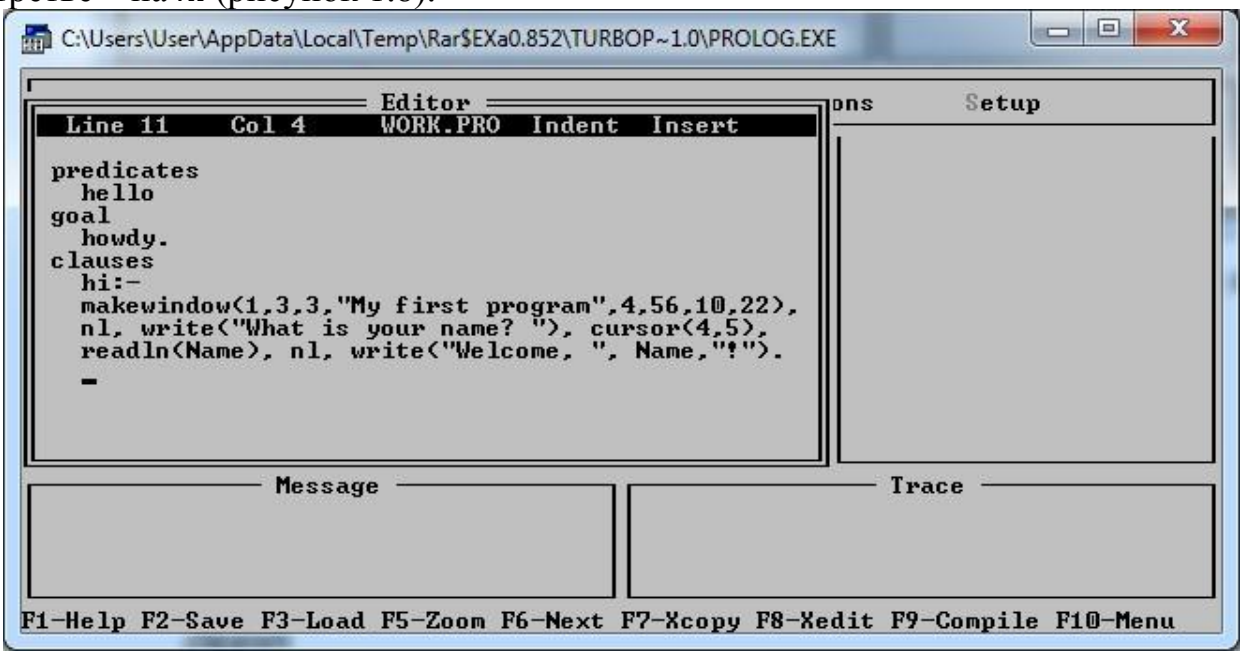

Рисунок 1.8 – Пример 1.1 (редактирование)

Запустите программу. Первая ошибка будет обнаружена, и управление будет передано редактору. Исправьте ошибку и нажмите F10. Произойдет автоматический выход из редактора и система запустит программу на выполнение. После перезапуска обнаружится вторая ошибка. Исправьте ее и нажмите F10. Программа будет оттранслирована и выполнена правильно.

Для сохранения программы следует перейти в режим ***Files*** и из меню следует выбрать режим ***Write to***. В ответ на запрос об имени файла следует ввести

Myfirst и нажать ***Enter*** (рисунок 1.9). Содержание рабочего файла будет сохранено, постфикс .PRO будет добавлен к его имени автоматически.
Для того чтобы просмотреть список всех программ в текущем каталоге, необходимо в режиме ***Files*** выбрать команду ***Directory***. После нажатия ***Enter*** Турбо Пролог покажет имя текущего каталога, маршрут доступа и наличие маски файлов. Нажмете ***Enter***, увидите список всех файлов в текущем каталоге, включая Myfirst.pro. Нажмите клавишу ***Backspace***, чтобы выйти из режима ***Directory***.

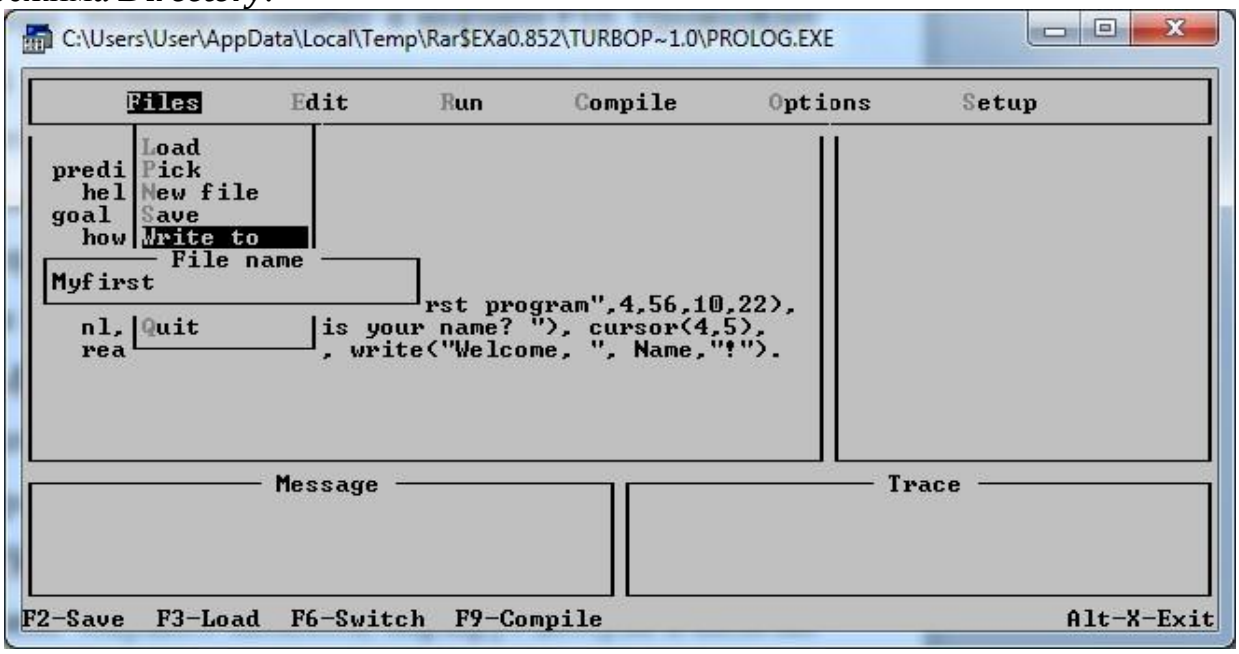

Рисунок 1.9 – Пример 1.1 (сохранение)

Для пошагового выполнения программы нужно добавить опцию ***trace*** в начало программы (рисунок 1.10).

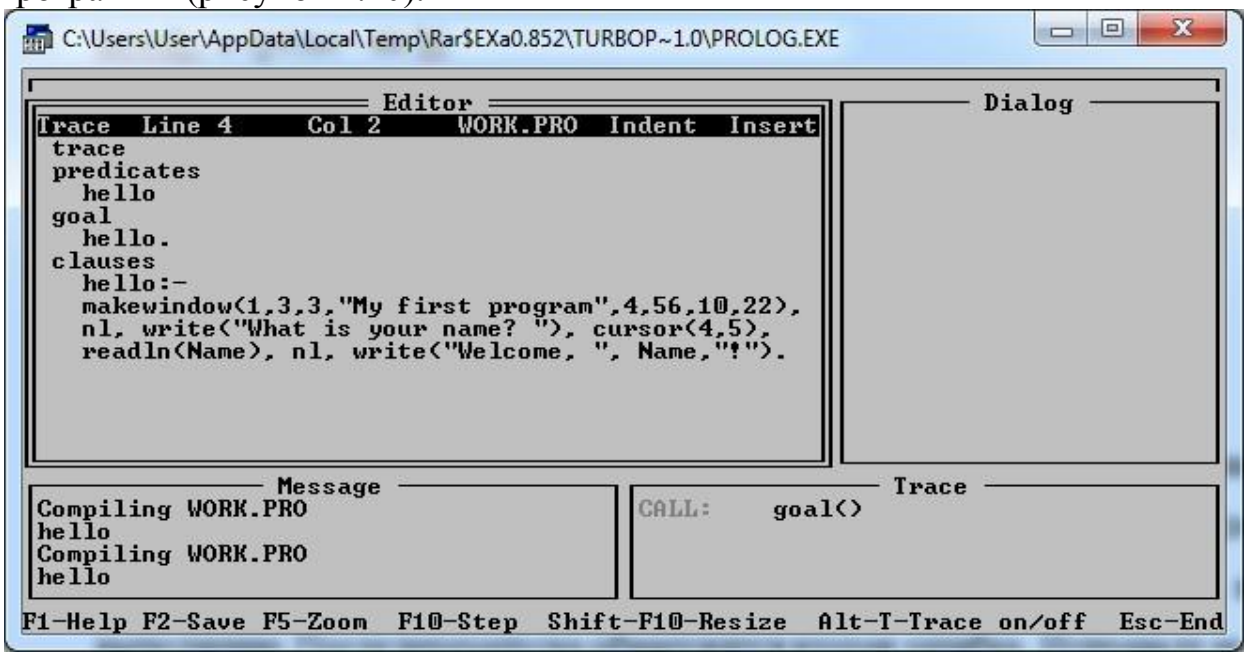

Рисунок 1.10 – Пример 1.1 (трассировка)

Выберите команду ***Run***. В окне редактора курсор будет мигать в конце задачи hello, а в окне трассировки начало выполнения этой задачи будет выглядеть следующим образом: CALL: goal().
F10 задает пошаговый режим выполнения задачи. CALL: hello() означает, что должен быть обработать предикат hello. Нажмите F10, увидите, что вызван предикат создания окон ***makewindow***. Следующее нажатие F10 вызовет выполнение этого предиката:
RETURN: makewindow(1,7,7,«*My first program*»,4,56,10,22).
Нажмите F10 и посмотрите, что будет в каждом окне, пока в окне трассировки не появится CALL: readln(_). Курсор будет находиться в окне, где выполняется программа, и где необходимо набрать имя. Наберите *Ivan* и нажмите ***Enter***. Окно трассировки покажет: RETURN: readln(«*Ivan*»).
Нажмите F10 и окно трассировки покажет следующее:
RETURN: hello()
RETURN: goal(),
что означает окончание выполнения программы. Нажмите F10 и система предложит нажать ***Spacebar*** для возврата в главное меню.
Окна системы могут быть использованы в любой конфигурации и любое окно может занимать либо целый экран, либо часть его. В любое время можно изменить вид, конфигурацию и расположение окна с помощью команды ***Setup***.
Удалите опцию ***trace*** из программы и запустите на выполнение, но не отвечайте на запрос об имени. С помощью F6 выделите окно сообщений. Изучите действие клавиш управления курсором и сделайте это окно в форме квадрата, передвиньте его по экрану, используя клавиши управления курсором с одновременным нажатием ***Shift***.
Для возобновления программы нажмите ***Backspace*** и введите имя. После выполнения программы управление будет передано основной системе через меню, но новая конфигурация окон сохранится.
Выберите опцию ***window size*** из ***Setup***-меню для приведения системных окон к их первоначальному виду.
Выберите ***Save*** из ***Setup***-меню для записи информации о размещении окон на диск. Для использования различных форматов окон выберите режим ***Read*** из ***Setup***-меню.
В примере 1.1 (рисунок 1.6) использован стандартный предикат управления окнами:

– ***makewindow***(WNo, ScrAttr, FrameAttr, Head, Row, Col, Height, Width) определяет часть площади экрана как окно; все параметры, кроме Header, должны быть целыми; Head должен быть строкой или символом и используется в качестве заголовка (на верхней линии рамки); номер (WNo), используется для определения активных окон; FrameAttr определяет цвет рамки; ScrAttr определяет цвет символов; при обработке предиката определения окна оно закрашивается цветом фона, курсор устанавливается в левый верхний угол; положение левого верхнего угла окна по отношению к

левому верхнему углу экрана определяется параметрами Row и Col; Height (высота) и Width (ширина) определяют размер окна.

Турбо Пролог имеет следующие стандартные предикаты управления окнами:

– ***shiftwindow***(WindowNo), где WindowNo – номер активизируемого окна, при этом Турбо Пролог запоминает номер предыдущего активного окна;

– ***window_attr***(Attr) позволяет изменить атрибут активного в настоящее время окна; после его обработки активное окно получает атрибут Attr;

– ***cursor***(Row, Col) передвигает курсор в позицию (Row, Col);

– ***clearwindow*** очищает окно;

– ***removewindow*** удаляет текущее окно;

– ***window_str***(StringParam) записывает строку в текущее окно или считывает строку из текущего окна;

– ***scr_char***(Row, Col, Character) используется для чтения и записи символов;

– ***scr_attr***(Row, Col, Attr) устанавливает позицию (Row,Col) символа Attr, если Attr связано; в противном случае Attr связывается с содержимым (Row,Col);

– ***field_str***(Row, Col, Lendth, String) используется при вводе / выводе полей, работает аналогично scr_char; используется для обмена информацией с полем длины Lendth, начинающимся в позиции (Row, Col) относительно верхнего левого края окна (экрана);

– ***field_attr***(Row, Col, Lendth, Attr) устанавливает или читает атрибут поля экрана; первые три параметра предиката перед его выполнением должны быть связаны; если параметр Attr свободен, то выполнение предиката приведет к связыванию Attr с величиной атрибута поля (определенной по первой позиции поля).

На рисунках 1.11–1.12 представлена Пролог-программа, которая использует стандартные предикаты обработки окон; суммирует два целых числа, выводит результат на экран.

Рисунок 1.11 – Пример программы работы с окнами

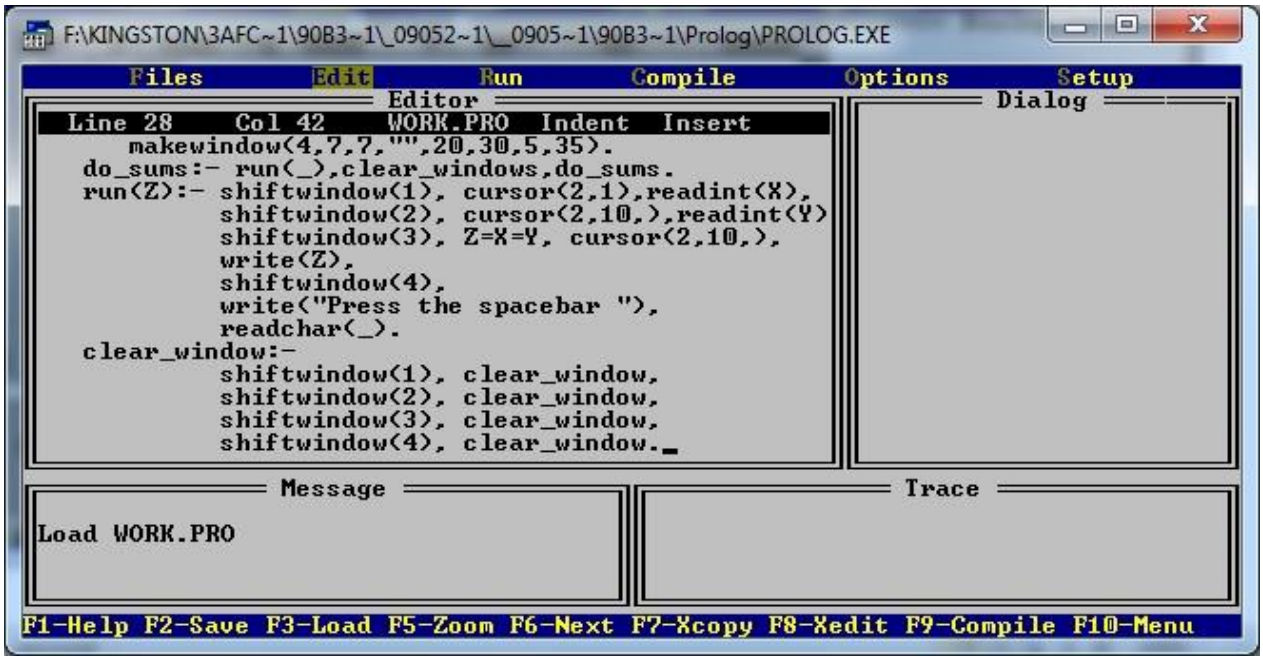

Рисунок 1.12 – Пример программы работы с окнами (продолжение)

Программа (пример 2.1) содержит базу данных, состоящую из набора фактов и правила логического вывода (рисунок 2.1).

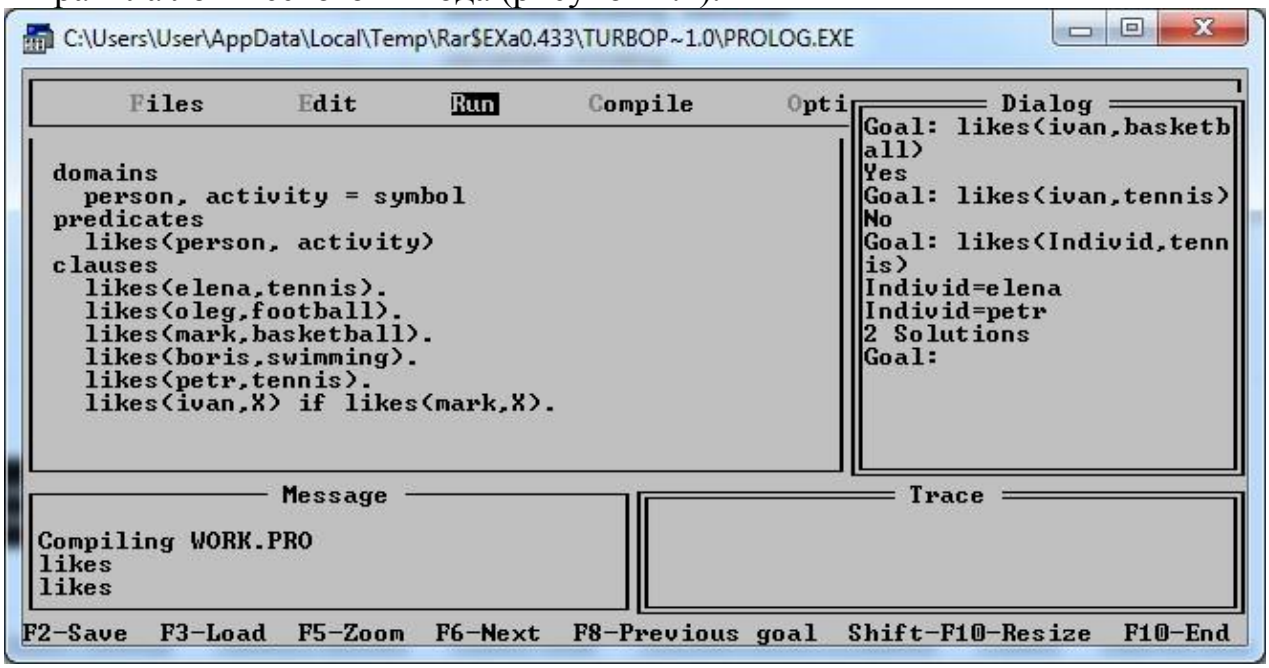

Рисунок 2.1 – Пример 2.1

Введите задачу, например:
Goal: likes (ivan, basketball).

Турбо Пролог в диалоговом окне ответит: ***Yes*** и будет ожидать введения следующей задачи, например:
Goal: likes (ivan, tennis).
Система ответит: ***No***, так как это утверждение отсутствует в наборе фактов и не может быть выведено с использованием правила.
В правиле
Likes (ivan, X) if likes (mark, X)
в качестве переменной использована переменная X, показывающая неизвестный вид игры.
При введении задачи:
Goal: likes (Individ, tennis)
Турбо Пролог ответит:
Individ = elena
Individ = petr
2 Solutions
Программа (пример 2.2) иллюстрирует поиск в базе данных всех автомобилей, стоимость которых не превышает 27000.
Когда цель задана в программе, то Пролог-система находит первое решение, «chrysler 12000», но в данном примере оно не единственное.

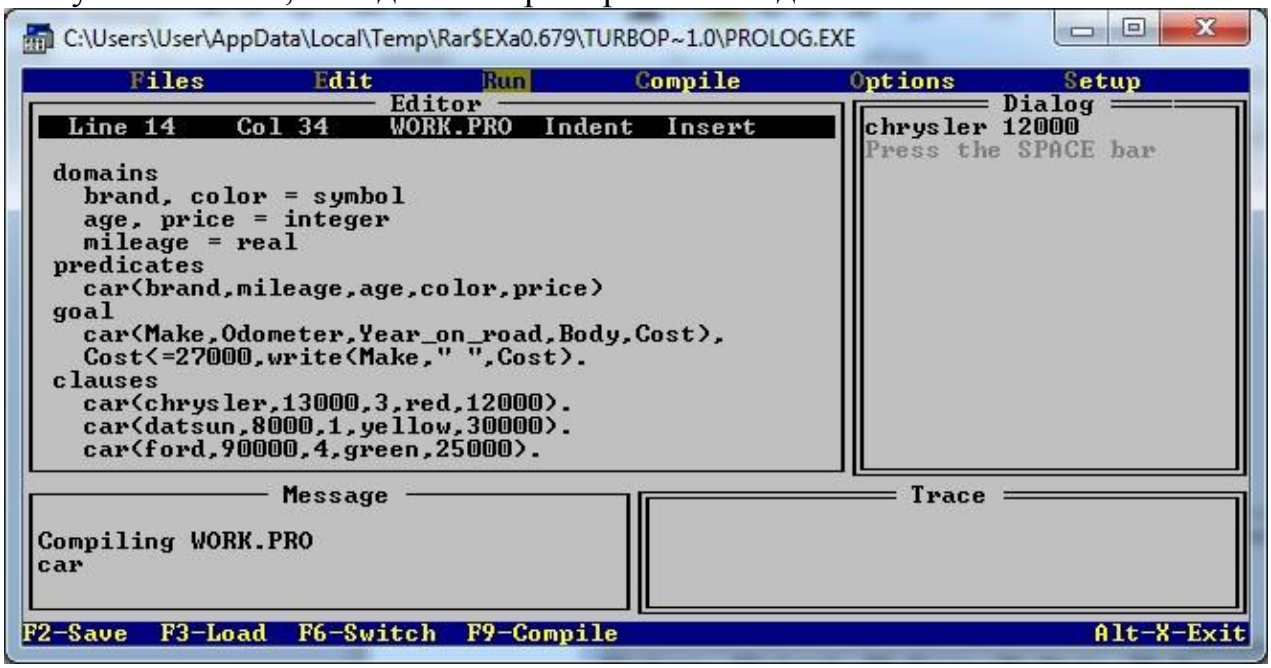

Рисунок 2.2 – Пример 2.2 (цель указана в программе)

Если ввести задачу (рисунок 2.3):
Goal: car (Make, _, _, _, Cost) and Cost < 27000, то система ответит:
Make=chrysler, Cost=12000
Make =ford, Cost=25000 2 Solutions
Предикат ***car*** содержит пять объектов (пять переменных). Если интересуют только цена и марка автомобиля, то можно использовать анонимные переменные (обозначаемые одиночным символом подчеркивания «_»),

которые могут быть использованы также как любые переменные, но они никогда не принимают каких-либо определенных значений.

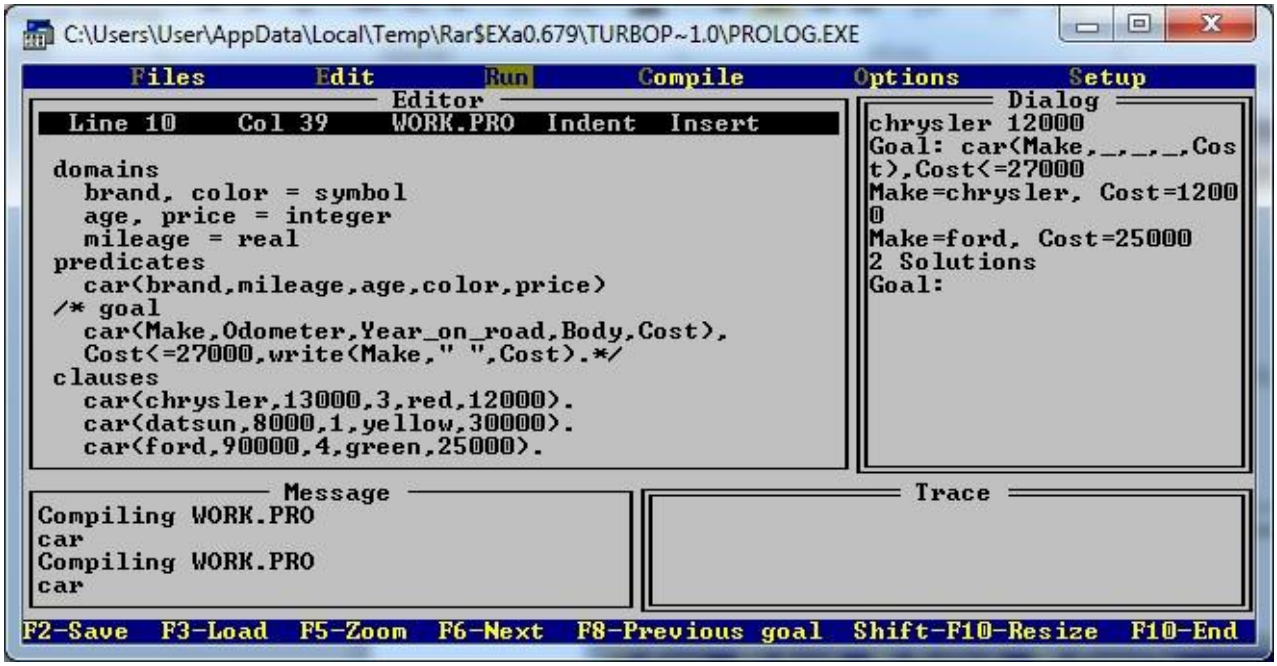

Рисунок 2.3 – Пример 2.2 (цель не указана в программе)

Турбо Пролог различает два типа переменных: «свободная» переменная (Турбо Пролог не знает ее значения) и «связанная» переменная (ее значение известно).

Рассмотрим решение задачи (пример 2.3), представленной на рисунке 2.4.

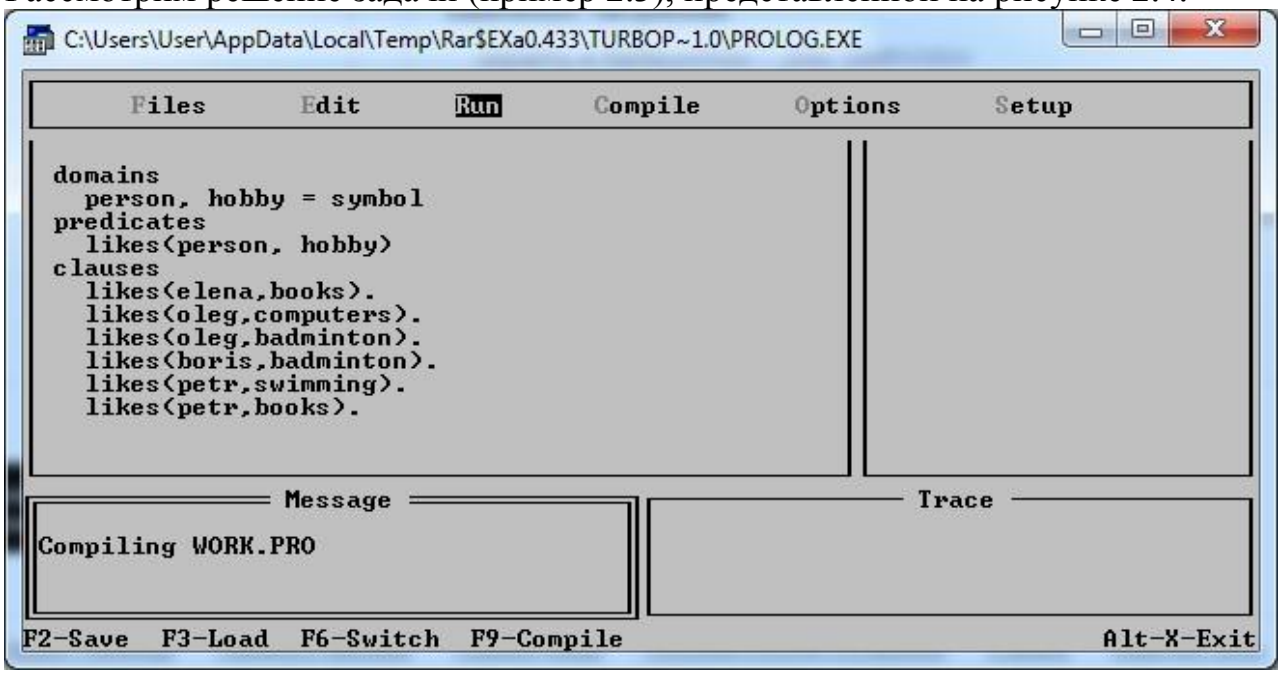

Рисунок 2.4 – Пример 2.3

Goal: likes (X, books) and likes (X, swimming).

В первой подзадаче *likes (X, books)* переменная X является свободной, а второй аргумент – *books* – определен. Первую подзадачу удовлетворяет первый факт и переменная X принимает значение первого аргумента – *elena*. Турбо Пролог оставляет против этого факта указатель. Так как X связано с именем *elena*, то Прологу необходимо отыскать факт *likes (elena, swimming)*, а так как подзадача не имеет решения, то Пролог снова начинает поиск решения первой подзадачи с факта, следующего за указателем, считая X свободным.

Содержание отчета и его форма

В отчете к лабораторной работе должно быть указано название работы; перечислены этапы выполнения лабораторной работы и краткая характеристика работ на каждом этапе решения задач, описанных в примерах.

Вопросы для защиты работы

1. Как создать файл в Пролог-системе?
2. В каком окне Пролог-системы можно внести изменения в программу?
3. Как осуществить запуск программы на выполнение?
4. Каким образом можно сохранить файл?
5. Как осуществить пошаговое выполнение программы?
6. Какова структура программы?
7. Какие разделы программы обеспечивают межмодульный интерфейс?
8. В каком разделе описываются предикаты?
9. В каком разделе описываются факты и правила?
10. Какие типы данных используются в Пролог-системе?

2. СОЗДАНИЕ И ИСПОЛЬЗОВАНИЕ СОСТАВНЫХ ОБЪЕКТОВ ДАННЫХ И ФУНКТОРОВ. СОЗДАНИЕ И ИСПОЛЬЗОВАНИЕ РЕКУРСИВНЫХ ОБЪЕКТОВ

Цель и содержание: ознакомление с основными приемами создания составных объектов данных и функторов в системе Turbo Prolog 2.0 и их использования; ознакомление с технологией создания и использования рекурсивных объектов в системе Turbo Prolog 2.0.
Организационная форма занятий: решение проблемных задач, разбор конкретных ситуаций.
Вопросы для обсуждения на лабораторном занятии: пример создания составных объектов данных и функторов; примеры создания и использования рекурсивных объектов.

Теоретическое обоснование

Турбо Пролог позволяет создавать объекты, компонентами которых являются другие объекты. Сложный объект состоит из ***функтора***, аргументами которого также являются объекты:

functor (object1, object2, ..., objectN).

Функтор может и не иметь аргументов, в этом случае он употребляется так: functor() или functor.

Рекурсия используется в тех случаях, когда отношения описаны со ссылкой на самих себя, а также когда сложные объекты являются частями других сложных объектов.

Аппаратура и материалы. Требования к техническому и программному обеспечению приведены в приложении В.

Указания по технике безопасности приведены в приложении Г.

Методика и порядок выполнения работы

Сложные объекты должны быть описаны, как показано в примере 3.1 (рисунок 3.1).

Решением задачи «Что есть у Елены?»

Goal: owns (elena, Thing) будет:

Thing = book (“eagle_feather”, “pavel_bazhov”)

Thing = dog (“chernysh”)

Решением задачи «Кто автор книги, которая есть у Елены?»

Goal: owns (elena, book(_, Name)) будет:

Name = pavel_bazhov

Решением задачи «Как зовут собаку Елены?»

Goal: owns (elena, dog(Name) будет:

Name = chernysh

Рисунок 3.1 – Пример 3.1

В приведенной ниже программе (пример 4.1) возникает ситуация, когда отношения описаны со ссылкой на самих себя. В примере описаны утверждения и правило для предиката factorial, который, будучи использован в задаче типа:

Goal: factorial(N,F) сообщит значение N! (рисунок 4.1).

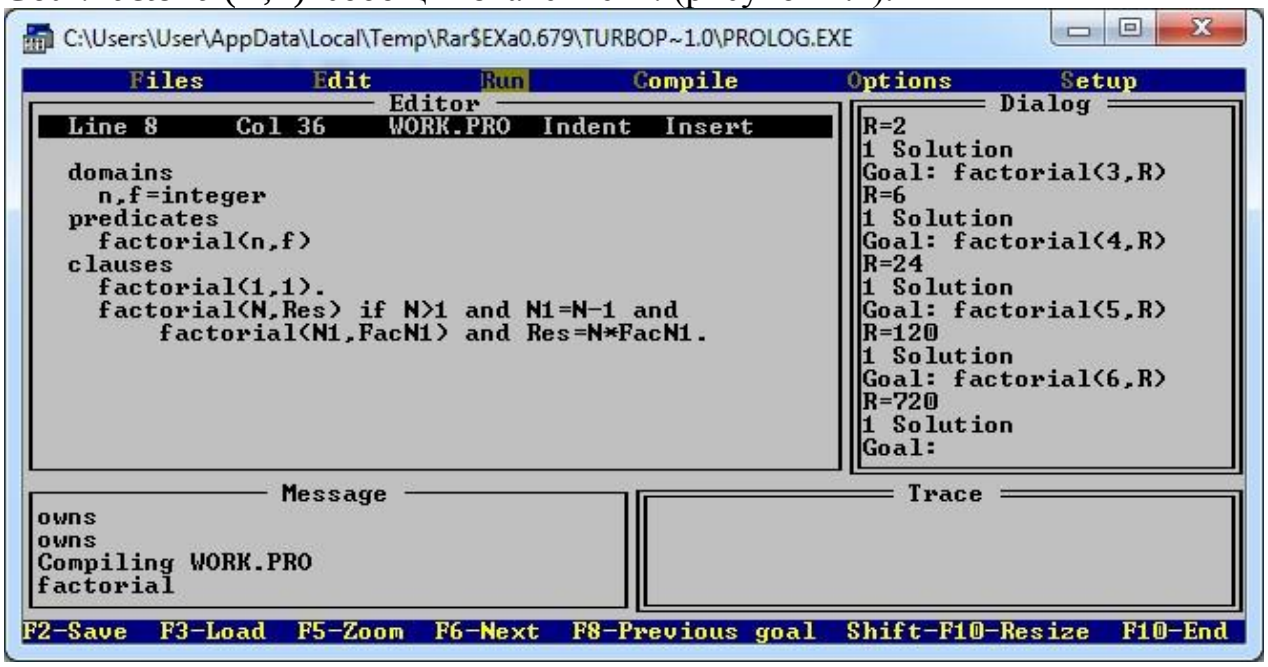

Рисунок 4.1 – Пример 4.1

Следующий пример иллюстрирует второй случай использования рекурсии: необходимо описать объект, содержащий имена учеников класса, не зная заранее численность этого класса.

Для решения этой задачи сформулируем определение области *classlist* по шагам. Формулировка начинается с пустого списка:

classlist=empty.

Затем формулируем рекурсивное описание: classlist=class (name, classlist).

Объектом такого рода может быть: class (petr, X).

Класс из двух учеников:

class (petr, class (oleg, empty))

или из трех: class (petr, class (oleg, class(mark, empty))).

class (name, classlist) – является сложным объектом. Функтором здесь является *class*, *name* – имя одного из учеников, *classlist* содержит остальные имена.

Полное определение области для classlist содержит две альтернативы: classlist=class (name, classlist); empty.

Таким же образом может быть описана последовательность чисел: integerlist=list (integer, integerlist); empty.

Список является основной структурой в Турбо Прологе. Элементы списка должны разделяться запятыми и заключаться в квадратные скобки, например: [1, 2, 3], [dog, cat, canary], [“Ivan”, “Oleg”, “Mark”].

Компонентами списка могут быть любые объекты, включая списки. Все компоненты списка должны принадлежать одной и той же области. Для формирования списка любых объектов можно пользоваться конструкцией:
domains objectlist=objects* objects=...
Турбо Пролог обрабатывает списки, разделяя их на две части – «голову» и «хвост». «Головой» списка [1, 2, 3] является элемент «1», «хвостом» – список [2, 3]. В Прологе список с головой X и с хвостом Y может быть описан следующим образом: [X|Y].
Приведенная на рисунке 4.2 программа (пример 4.2) позволяет решить следующие задачи: имеется список имен и требуется выяснить, есть ли данное имя в списке
Goal: member (ball, [cat, cut, car, foot, ball]) (ответ ***Yes***) или какие имена содержит список
Goal: member (Z, [cat, cut, car, foot, ball])
Z = cat
Z = cut
Z = car
Z = foot
Z = ball
5 Solutions

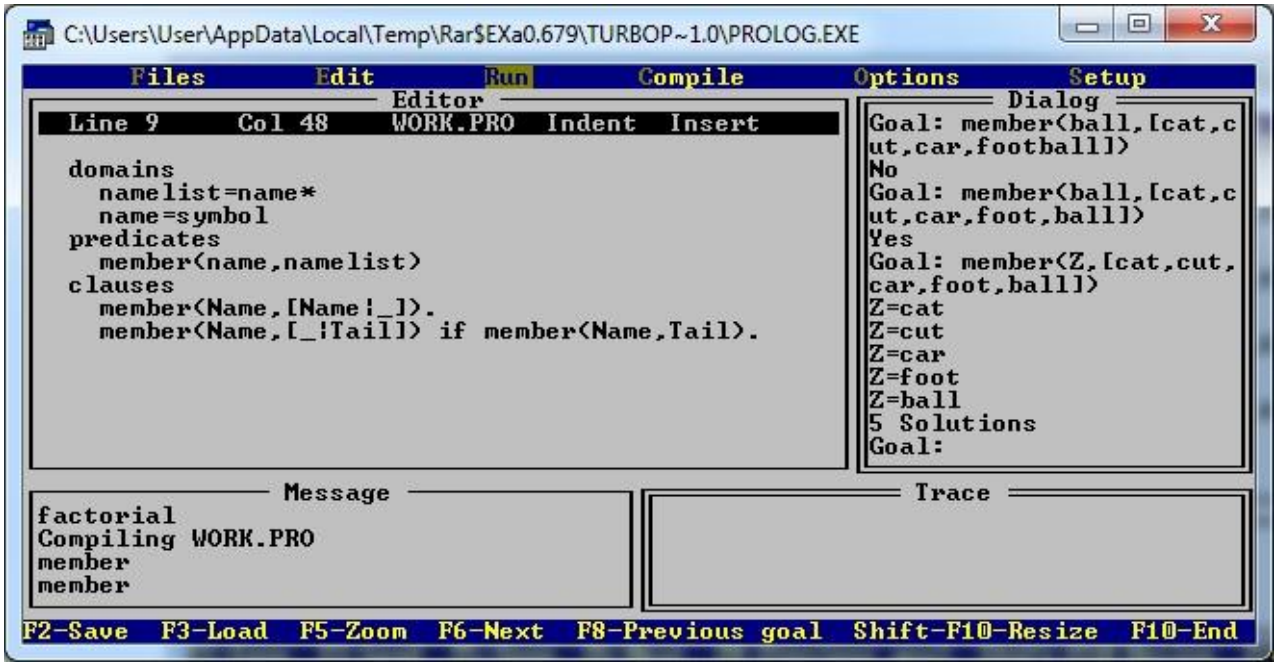

Рисунок 4.2 – Пример 4.2

Для вывода элементов списка существует предикат: write_a_list ([]).
write_a_list ([Head | Tail]) if write (Head), nl, write_a_list (Tail).
С помощью предиката *append* один список добавляется к другому. Этот предикат имеет три аргумента *append (List1, List2, List3)*, объединяет *List1* и *List2*, формирует *List3*, используя рекурсию (рисунок 4.3).
Один и тот же предикат может иметь несколько применений. Его можно использовать, если известны первый и третий аргументы, но неизвестен

второй, или, если известен только третий аргумент. Например, чтобы узнать, какие списки дают в качестве объединения данный список, нужно поставить Турбо Прологу следующую задачу:

Goal: append (X, Y, [1, 2, 3, 4]).

Результат решения данной задачи представлен на рисунке 4.4.

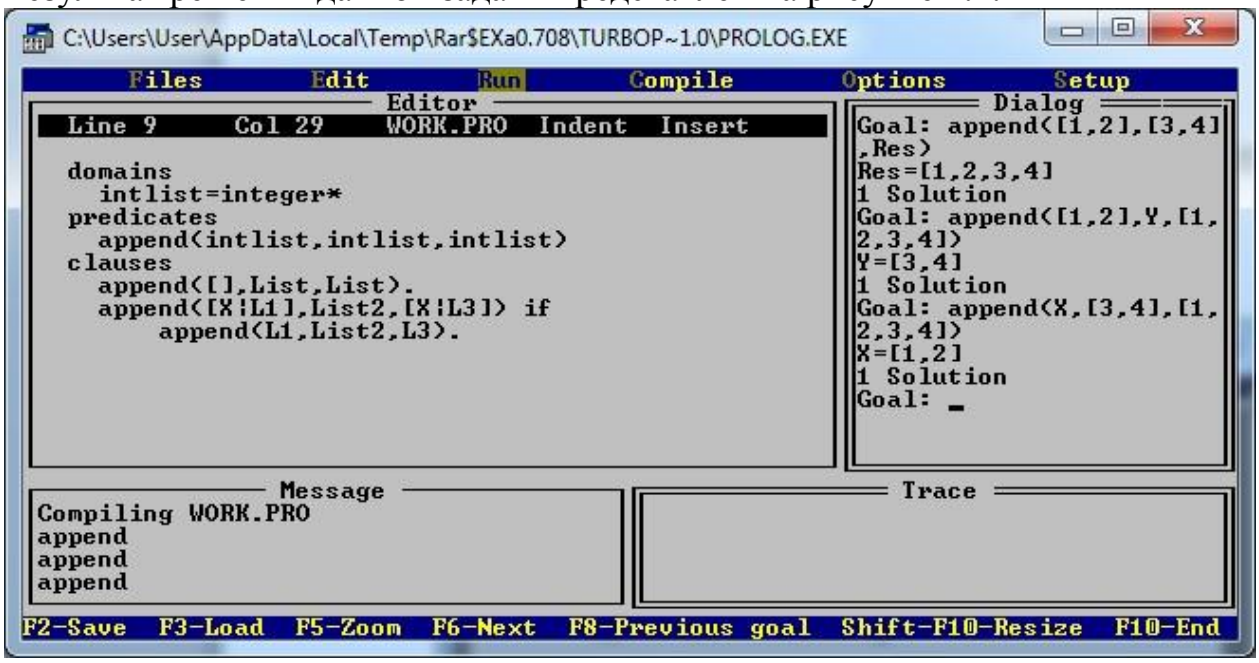

Рисунок 4.3 – Пример 4.3

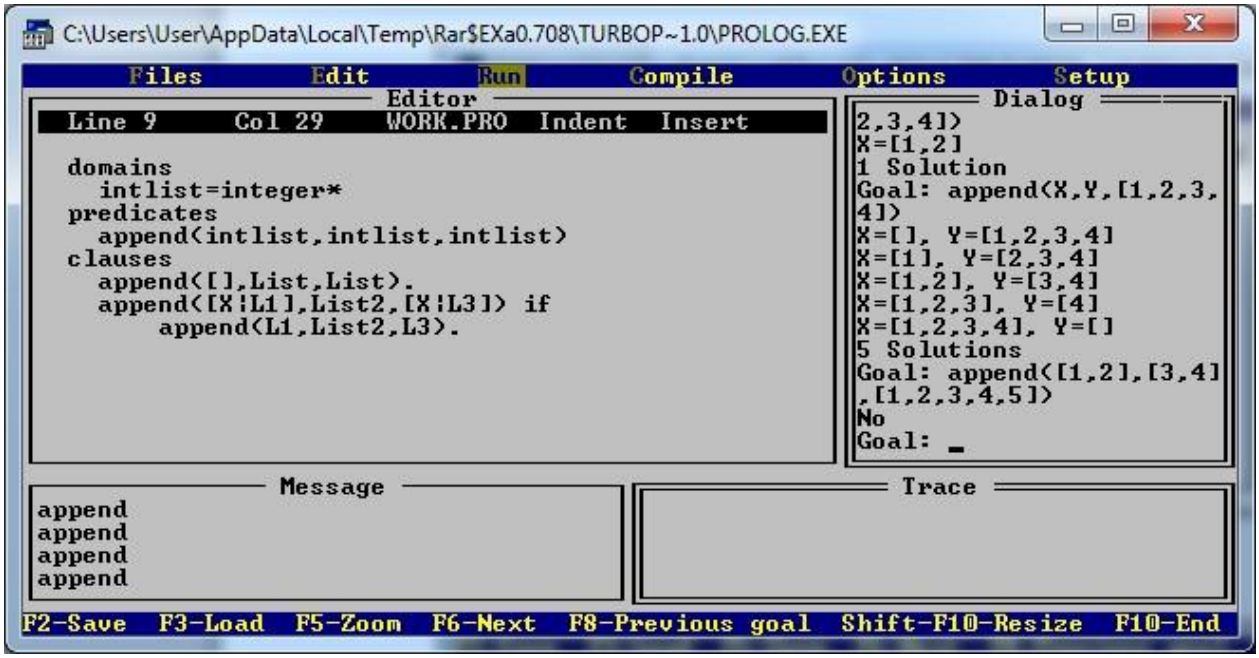

Рисунок 4.4 – Пример 4.3 (продолжение)

Содержание отчета и его форма

В отчете к лабораторной работе должно быть указано название работы; перечислены этапы выполнения лабораторной работы и краткая характеристика работ на каждом этапе решения задач, описанных в примерах.

Вопросы для защиты работы

1. Что такое «функтор»?
2. В каком разделе Пролог-программы описывается функтор?
3. Чем описание функтора отличается от описания предиката?
4. Что такое «рекурсия»?
5. В каких случаях используется рекурсия?
6. Приведите примеры создания рекурсивных объектов.
7. Как описываются списки в Пролог-программе?
8. Приведите примеры использования рекурсивных объектов.

3. ПРИМЕНЕНИЕ МЕТОДИК, ПОЗВОЛЯЮЩИХ УПРАВЛЯТЬ ПОИСКОМ РЕШЕНИЙ, СТАНДАРТНЫЕ ПРЕДИКАТЫ FAIL, CUT, СЛУЧАИ СОВМЕСТНОГО ИСПОЛЬЗОВАНИЯ ПРЕДИКАТОВ FAIL И CUT

Цель и содержание: ознакомление с методиками, позволяющими управлять поиском решений в системе Turbo Prolog 2.0.
Организационная форма занятий: решение проблемных задач, разбор конкретных ситуаций.
Вопросы для обсуждения на лабораторном занятии: примеры программ с использованием стандартных предикатов ***fail***, ***cut***; пример программы с совместным использованием стандартных предикатов ***fail*** и ***cut.***

Теоретическое обоснование

Турбо Пролог содержит специальный встроенный предикат ***fail***, всегда принимающий значение *false* («ложь»), вынуждающие к возврату, т. е. к поиску новых решений.
Операция прекращения поиска называется отсечением (***cut***) и обозначается восклицательным знаком (**!**). Предикат *cut* всегда принимает значение *true* («истина»)
Отсечение используется в следующих случаях: когда заранее известно, что некоторые возможные пути не приведут к интересующим решениям; когда отсечение требуется по логике программы.
Многие решаемые в системе Турбо Пролог задачи требуют совместного использования предиката *cut*, прекращающего поиск, и предиката *fail*, вынуждающего искать новые решений.

Аппаратура и материалы. Требования к техническому и программному обеспечению приведены в приложении В.
Указания по технике безопасности приведены в приложении Г.

Методика и порядок выполнения работы

В примере 5.1 используется предикат ***fail*** для поиска всех возможных решений (рисунок 5.1).

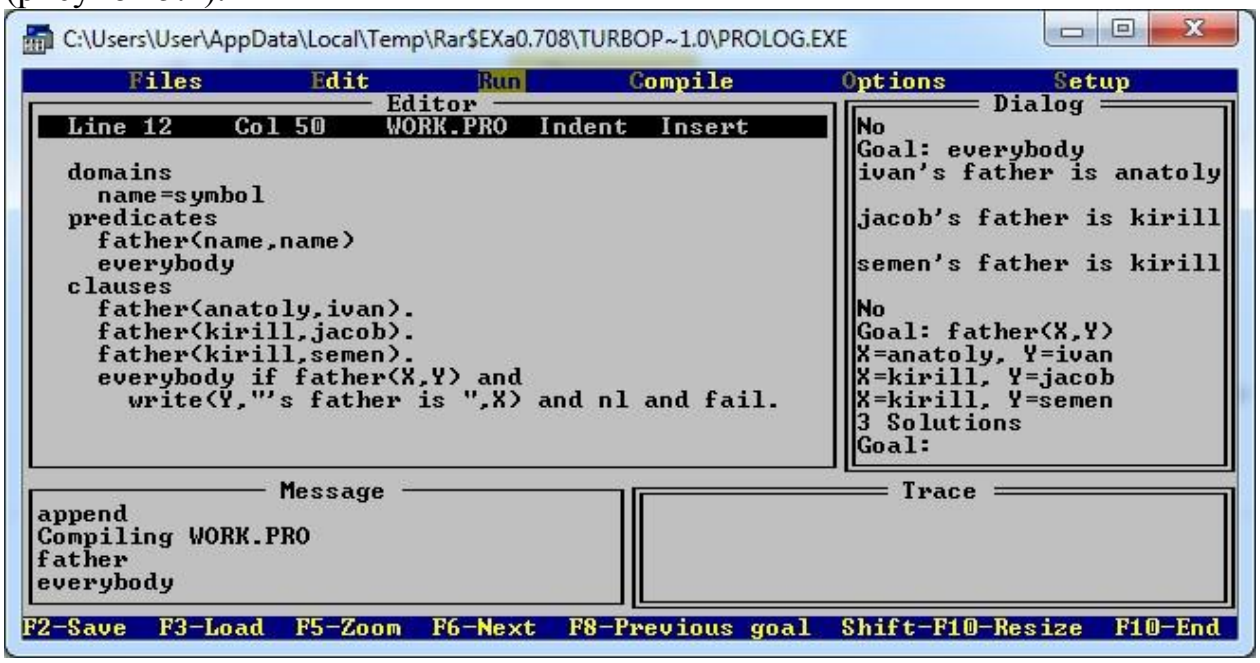

Рисунок 5.1 – Пример 5.1

Задача *father (X, Y)* может быть использована в двух различных ситуациях: как вопрос к Турбо Прологу (внешняя задача) и в правой части правила (в виде встроенной задачи), например, grandfather (X, B) if father (X, Y) and father (Y, B).

Если *father (X, Y)* является внешней задачей, Турбо Пролог сообщит все возможные решения.

Если *father (X, Y)* – встроенная задача, то Турбо Пролог после её решения перейдет к следующей задаче и сообщит всего лишь одно решение.

Цель предиката *everybody* – собрать всевозможные решения.

Если поставлена задача:

Goal: everybody,

то Турбо Пролог сообщит (рисунок 5.1): ivan's father is anatoly jacob's father is kirill semen's father is kirill No

Если задана цель:

Goal: father (X, Y), то Турбо Пролог сообщит (рисунок 5.1):

X=anatoly, Y=ivan

X=kirill, Y=jacob

X=kirill, Y=semen

3 Solutions

Следующий пример (рисунки 5.2–5.3) основан на той мысли, что два человека нравятся друг другу, если они имеют хотя бы один общий интерес.

Если поставлена задача:

Goal: findpairs, то Турбо Пролог сообщит (рисунок 5.2):

oleg + maria
igor + maria igor + inna
No

Если бы в предикате *common_interest* не было использовано отсечение, то каждая пара имен обрабатывалась бы программой столько раз, сколько общих интересов есть у этой пары. Таким образом, отсечение используется для предотвращения перехода к следующему предложению.

Если поставлена задача «Чьим общим интересом является музыка?» (рисунок 5.4):

Goal: person (X, f, L1), person (Y, f, L2), common_interest (L1, L2, music), то Турбо Пролог сообщит:

X = maria
Y = inna
2 Solutions

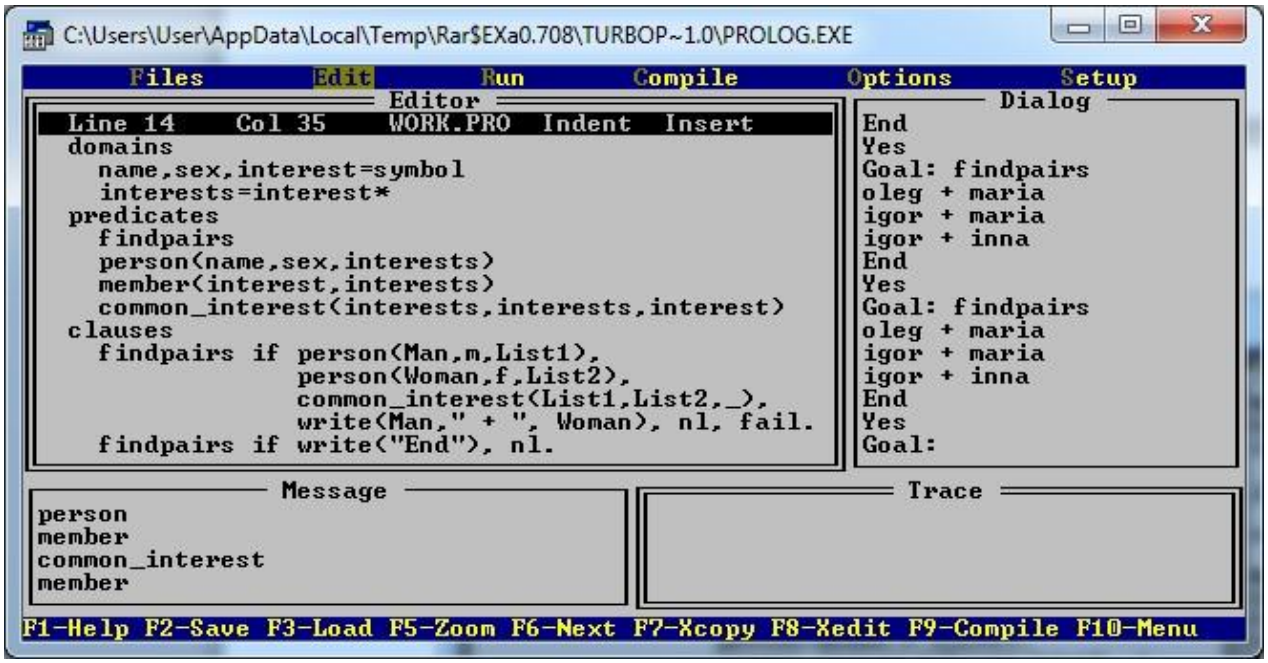

Рисунок 5.2 – Пример 5.2

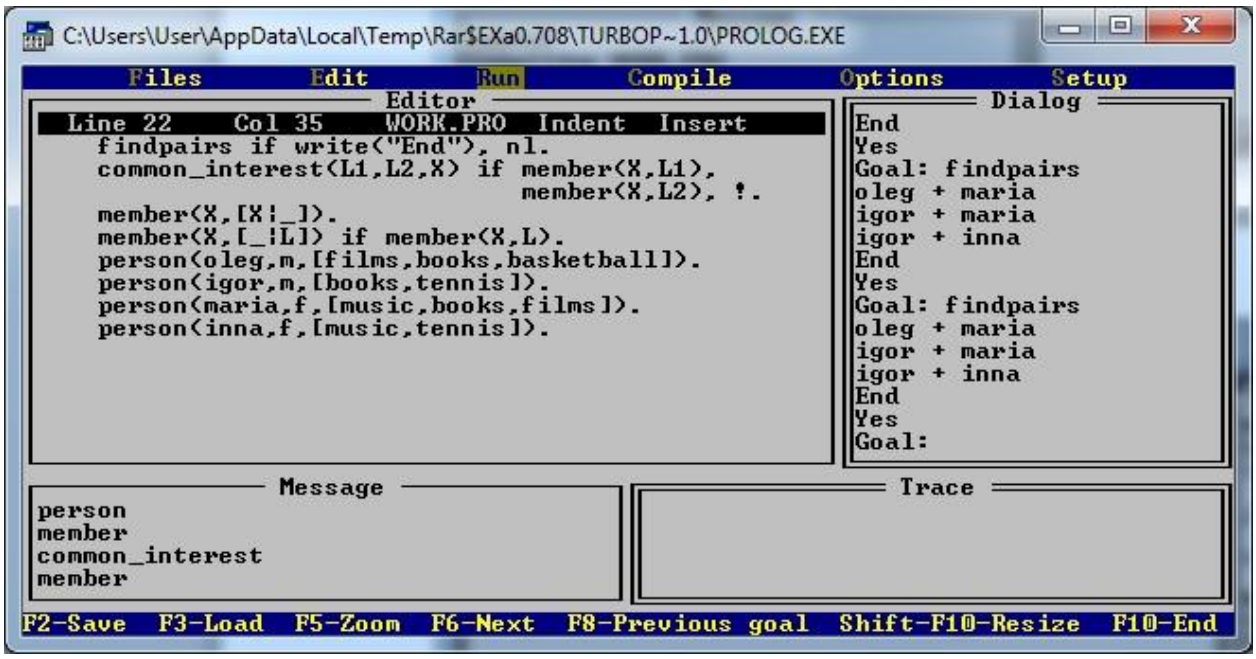

Рисунок 5.3 – Пример 5.2 (продолжение)

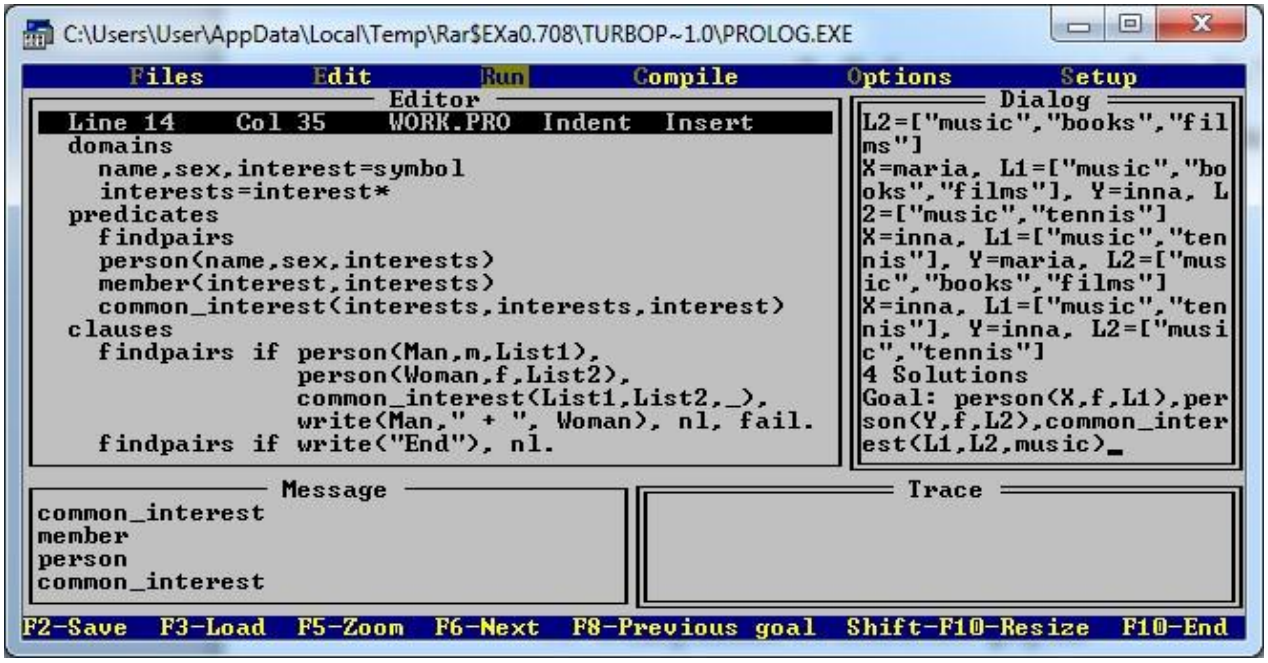

Рисунок 5.4 – Пример 5.3

Пример 3. Рассмотрим различные способы записи предиката *different*, определяющего различны ли числа.

Варианты, использующие различные сочетания встроенных предикатов *cut* (!) и *fail*:

different (X, X):- !, fail. different (_, _).

или different (X,Y):- X=Y, !, fail. different (_, _).

или different (X, Y):- X=Y, !, fail; true.

/* true – встроенный предикат, который всегда истинен */ или different (X,Y):- not (X=Y).

Содержание отчета и его форма

В отчете к лабораторной работе должно быть указано название работы; перечислены этапы выполнения лабораторной работы и краткая характеристика работ на каждом этапе решения задач, описанных в примерах.

Вопросы для защиты работы

1. Какое значение всегда принимает предикат *fail*?
2. Какое значение всегда принимает предикат *cut*?
3. В каких случаях используется предикат *fail*?
4. В каких случаях используется предикат *cut*?
5. Приведите примеры использования предиката *fail*.
6. Приведите примеры использования предиката *cut*.
7. Какие значения принимают предикаты *fail* и *cut*?
8. В каких случаях совместно используются предикаты *fail* и *cut*?
9. Приведите примеры совместного использования предикатов *fail* и *cut*.

4. ПРИМЕНЕНИЕ ОСНОВНЫХ ПРЕДИКАТОВ УПРАВЛЕНИЯ СТРОКОЙ. ОРГАНИЗАЦИЯ ВВОДА-ВЫВОДА ДАННЫХ РАЗЛИЧНЫХ ТИПОВ. ИЗУЧЕНИЕ СТАНДАРТНЫХ ПРЕДИКАТОВ РАБОТЫ С ФАЙЛАМИ

Цель и содержание: ознакомление с основными предикатами управления строкой в системе Turbo Prolog 2.0; ознакомление с основными предикатами организация ввода-вывода данных различных типов в системе Turbo Prolog 2.0; ознакомление со стандартными предикатами работы с файлами в системе Turbo Prolog 2.0.

Организационная форма занятий: решение проблемных задач, разбор конкретных ситуаций.

Вопросы для обсуждения на лабораторном занятии: примеры программ с использованием стандартных предикатов управления строкой; примеры программ с использованием стандартных предикатов организация ввода-вывода данных различных типов; примеры программ работы с файлами.

Теоретическое обоснование

Предикат ***frontchar*** (String1, CharParam, String2) действует так, как если бы он был определен с помощью равенства
String1=(сцепление символа CharParam и строки String2).
Завершается неудачно, если String1 пустая строка.
Предикат ***fronttoken***(String1, SymbolParam, Rest) используется для разделения строки в список ***лексем*** (слов). При обращении к предикату необходимо,

чтобы *String1* была связанной. Предикат находит первую лексему строки *String1* и связывает ее с переменной *Rest*. Пробелы в начале
строки игнорируются. Считается, что последовательность символов составляет лексему, если она удовлетворяет одному из следующих условий: – имя в соответствии с синтаксисом Турбо Пролога; – число (знак) рассматривается как отдельное слово; – символ, отличный от ' '.
Предикат ***concat***(String1, String2, String3) служит для сцепления строк *String1* и *Strins2* в строку *String3*. Перед обращением к предикату по меньшей мере две строки должны быть связаны.
Предикат ***frontstr***(NumberOfChars, String1, StartStr, String2) служит для разделения строки *String1* на две части так, что *StartStr* содержит первые *NumberOfChars* символов, а строка *String2* содержит остаток.
Предикат ***str_len***(StringParam, InegerLength) возвращает длину *StringParam*.
Предикат ***isname***(String) служит для проверки, является ли данная строка именем согласно синтаксиса Турбо Пролога, т. е. начинается ли она с буквы, и является ли она последовательностью букв, цифр и символов подчеркивания (пробелы до и после исследуемого слова игнорируются).
Стандартные предикаты преобразования типов:
– ***char_ascii***(Acharacter, AnInteger) – переводит символ в его код ASCII;
– ***str_char***(OneCharInAString, OneCharacter) – преобразует строку из одного символа в символ;
– ***str_int***(AString, AnInteger) – преобразует строку в целое число;
– ***str_real***(AString, AReal) – преобразует строку в действительное число;
– ***upper_lower***(UpperCaseString, LowCaseString) – переводит символ нижнего регистра в символ верхнего регистра и наоборот.
Турбо Пролог включает стандартные предикаты для чтения таких объектов как: полные строки символов; целые, вещественные и символьные величины с клавиатуры; файлы с дисков (таблица 8.1).

Таблица 8.1 – Стандартные предикаты для чтения объектов

Предикат	Описание
readln (Line)	Переменная *Line* должна быть свободной перед началом ввода; тип переменной может быть, как строковым, так и символьным.
readint (X)	Переменная *X* должна быть свободной переменной целого типа.
readreal (X)	Переменная *X* должна быть свободной переменной действительного типа.
readchar (CharParam)	Переменная *CharParam* должна быть свободной переменной символьного типа.

file_str (Filename, X)	Тип переменных *Filename* и *X* должен быть символьным или строковым. Переменная *X* должна быть свободной перед обработкой предиката *file_str* считывает символы вплоть до символа *EOF* (обычно *Ctrl-Z*. Содержание файла *Filename* передается переменной *X*.

Оператор вывода может содержать любое требуемое число аргументов: ***write***(Arg1, Arg2, ...). Эти аргументы могут быть либо константами стандартных типов, либо переменными.

Стандартный предикат ***nl*** часто используется вместе с предикатом ***write*** и вызывает переход к новой строке.

В системе Turbo Prolog существуют следующие предикаты для работы с файлами:

– deletefile (DOS_filename) – удаление файла;

– save (DOS_filename) – сохранение файла;

– renamefile (old_DOS_filename, new_DOS_filename) – переименование файла;

– existfile (DOS_filename) – проверка на наличие файла с таким именем;

– flush (file_domain) – сброс данных из внутреннего буфера, отведенного для данного устройства записи;

– disk (Path) – выбор дисковода и пути доступа;

– dir (Path, File_spec, File_name) – переменой *Path* должен быть присвоен корректный путь доступа, переменная *File_spec* задает расширение представляющей интерес группы файлов. Данный предикат выдает каталог имен файлов с заданным расширением; можно выбрать среди них нужный и нажать *Enter*. Имя файла будет присвоено переменой *File_name*.

При описании файловых доменов тип домена записывается по левую сторону от знака равенства, а имя домена по правую. Например,

file =datafile file=datafile1; datafile2

– openwrite (datafile, filename) – открытие файла для записи или создания, где *datafile* – введенный пользователем файловый домен, *filename* – имя файла в DOS, теперь ссылки на *datafile* будут означать обращение к *filename*;

– writedevice (datafile) – назначение файла в качестве устройства записи;

– openread (datafile, filename) – открытие файла для чтения;

– readdevice (datafile) – назначение файла устройством чтения;

– openmodify (datafile, filename) – открытие файла для редактирования, указатель помещается в начало файла, сместить указатель можно при помощи предиката *filepos*;

– openappend (datafile, filename) – открытие файла для добавления данных в конец файла;

– closefile (datafile) – закрытие файла.

Предикаты для работы с файлами прямого доступа:

– openmodify (fn,f ilename) – связывает логическое имя файла *fn* с именем файла.
– filepos (fn, pos, mode) – устанавливает указатель файла в заданную позицию.
Таблица 9.1 – Действие системы при операциях с файлами прямого доступа

Режим mode	Действия системы
0	Смещение берется относительно начала файла
1	Смещение берется относительно текущей позиции
2	Смещение берется относительно конца файла

Аппаратура и материалы. Требования к техническому и программному обеспечению приведены в приложении В.
Указания по технике безопасности приведены в приложении Г.

Методика и порядок выполнения работы

В приведенной программе (рисунок 7.1) предикат *frontchar* используется для представления строки в виде списка символов.

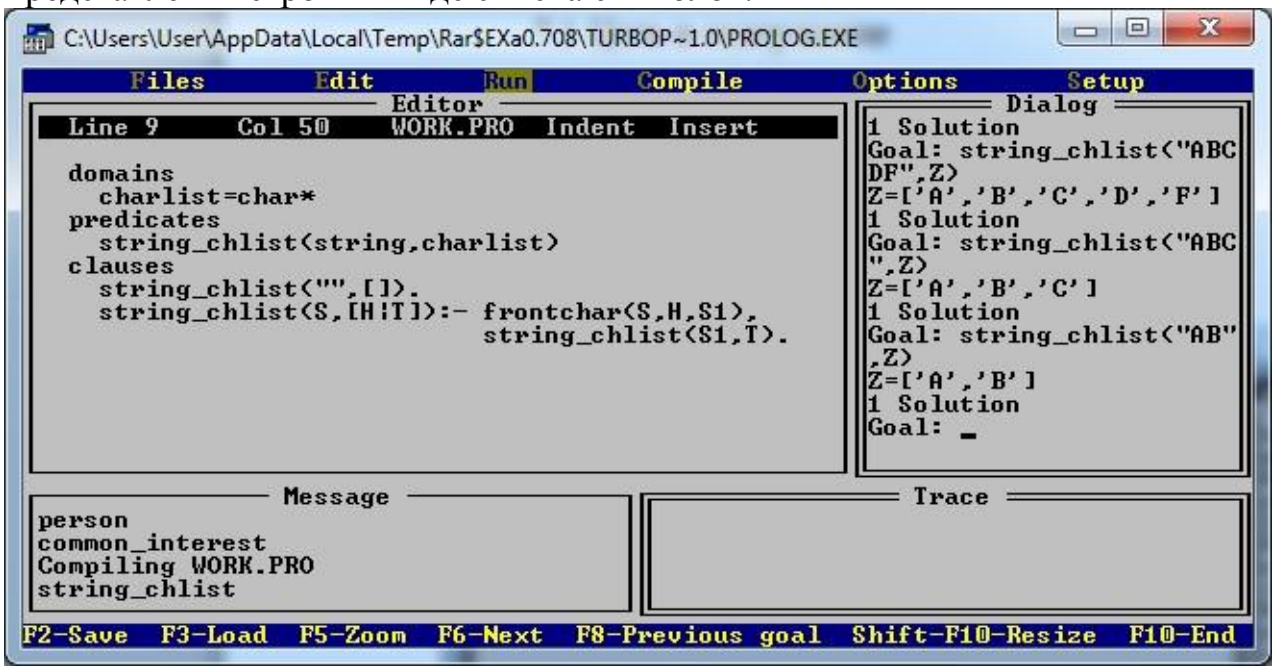

Рисунок 7.1 – Пример 7.1

На поставленную задачу:
Goal: string_chlist(“ABCDF”, Z) система ответит, что Z связана со списком [‘A’, ‘B’, ‘C’, ‘D’, ‘F’].
Следующая программа (рисунок 7.2) разделяет строку на последовательность имен.

В результате решения задачи:
Goal: string_namelist (“cat army art car ball”, X)
Пролог-система ответит:
X=[“cat”, “army”, “art”, “car”, “ball”]
1 Solution

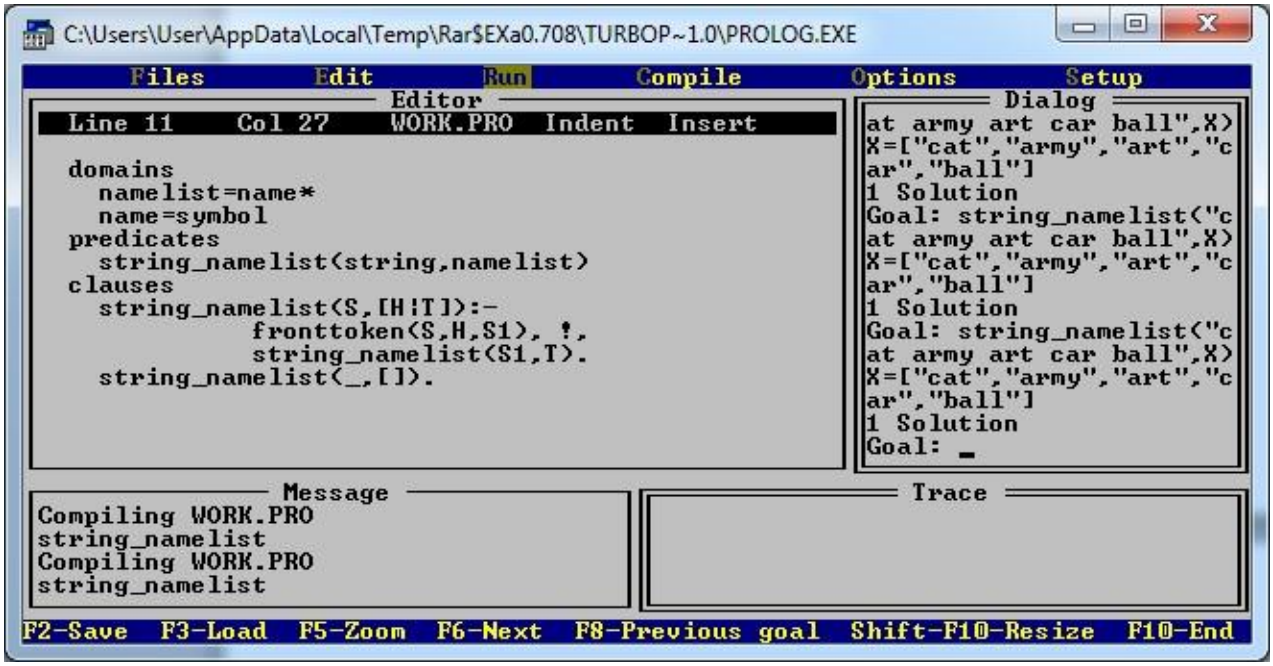

Рисунок 7.2 – Пример программы

Приведенная программа (рисунки 8.1–8.2) иллюстрирует использование предикатов *readln*, *readint*, *readchar* и выборку составных объектов из входных строк.

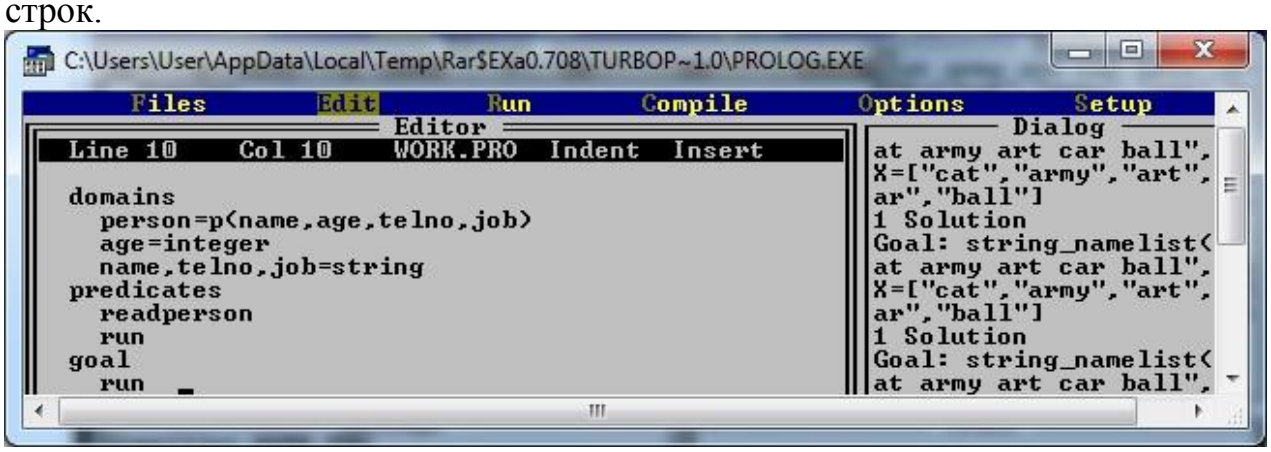

Рисунок 8.1 – Пример 8.1

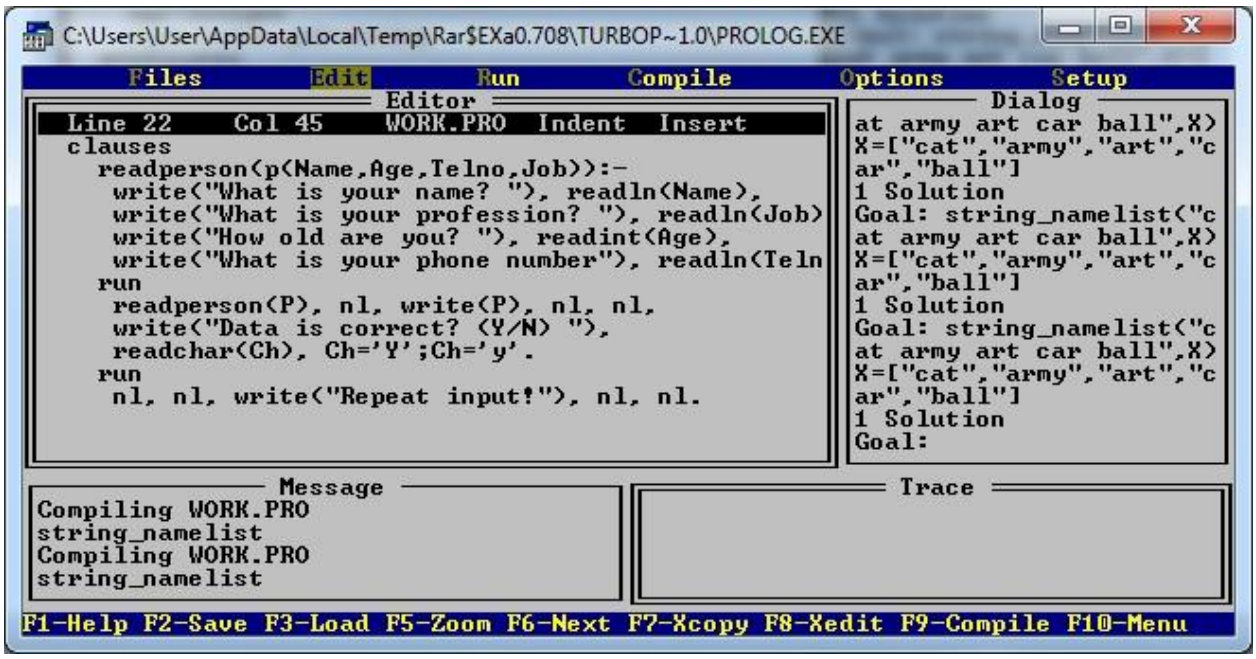

Рисунок 8.2 – Пример 8.1 (продолжение)

Рассмотрим примеры работы с файлами.

Пример 1. Вывести информацию на экран дисплея и в файл на диске.

```
/* Запись данных в файл */ domains
     str = string       file = datafile predicates
          data(str)
     write_lines goal
     openwrite(datafle,"file1.dat"),     write_lines,
closefile(datafile).
clauses
     data("Старому году оставьте печали,!").
     data("Забудьте обиду, беду.").       data("Только
успехов, здоровья и счастья,").     data("Мы Вам желаем в
Новом году!").
  write_lines:-    data(Line),    write("        ",Line),nl,
writedevice(datafile),       write("             ",Line),nl,
writedevice(screen),  fail, write_lines.
```

Пример 2. Вывести данные файла на экран.

```
/* Чтение данных из файла */ domains
     str = string       file = datafile predicates
     read_write_lines goal openread(datafile,"file1.dat"),
read_write_lines, closefile(datafile).
clauses
  read_write_lines          :-          readdevice(datafile),
not(eof(datafile)), readln(Line),       writedevice(screen),
write("  ",Line),nl,     read_write_lines.
```

Пример 3. Записать в файл данные, вводимые с клавиатуры

```
/* Запись в файл данных, вводимых с клавиатуры */ domains
     file = datafile   dstring, cstring = string predicates
          readin(dstring,cstring)
     create_a_file goal     create_a_file. clauses
          create_a_file :- nl, nl,
               write("Введите полное имя файла."), nl, nl,
readln(Filename),
openwrite(datafile,Filename),
     writedevice(datafile), readln(Dstr),
concat(Dstr,"\13\10",Cstr), readin(Dstr,Cstr),
closefile(datafile).
          readin("done",_) :- !.
/*ввод данных завершится после вода слова "done"*/
     readin(_,Cstr) :- write(Cstr), readln(Dstrl),
     concat(Dstrl, "\13\10",Cstr1),
     writedevice(datafile), readin(Dstrl,Cstr1).
```

Пример 4. Данные, вводимые с клавиатуры, записать в файл прямого доступа.

```
/* Запись данных в файл прямого доступа */ domains
     file = datafile  predicates
     create_a_random_access_file     write_read_more(real,
string)       pad_str (strIng,string,integer) qoal
          create_a_random_access_file.
clauses
     create_a_random_access_file :-           write("Please
enter filename:"),nl,           readln(Filename),
     openwrite(datafile,Filename),
     closefile(datafile),
     openmodify(datafile,Filename),           write("Введите
строку"),nl, readln(Dstr),
     write_read_more(0,Dstr), closefile(datafile).
          write_read_more(_,"done") :- nl,
write(" Press the space bar."),
               readchar(_),exit.
     write_read_more(Index,Dstr) :-
     writedevice(datafile),
     filepos(datafile,Index,0),
     pad_str(Dstr,Padstr,38),            concat(Padstr,
"\10\13", Cstr),           write(Cstr),
writedevice(screen),           write("Введите строку"),nl,
readln(Dstr1),        Index1 = Index + 40,
     write_read_more (Index1, Dstr1).
     pad_str (Instr,Instr,Length) :-
     str_len(Instr,Testlength),          Testlength >=
Length,!.
```

```
    pad_str (Instr,Padstr,Length) :-
    concat(Instr,"-",Newstr),
    pad_str(Newstr,Padstr,Length).
```

Пример 5. Вывести на экран заданную строку файла прямого доступа и выдача их на экран.

```
/* Вывод данных из файла прямого доступа */ domains
    file = datafile  predicates
    read_a_random_access_file goal
    read_a_random_access_file. clauses
    read_a_random_access_file:- write("Please enter
filename:"), nl, readln(Filename),
        openread(datafile,Filename),        write("Type 1n
record number: "),nl,
   readreal(Record),      I  ndex  =  (Record  -  1)  *  40,
readdevice(datafile),            filepos(datafile,Index,0),
readln(Cstr),    write(Cstr), nl, nl,
        write("Press the space bar."),nl,
    readdevice(keyboard), readchar(_),
        closefile(datafile),           exit.
```

Содержание отчета и его форма

В отчете к лабораторной работе должно быть указано название работы; перечислены этапы выполнения лабораторной работы и краткая характеристика работ на каждом этапе решения задач, описанных в примерах.

Вопросы для защиты работы

1. Перечислите средства работы со строками.
2. Какой стандартный предикат соединяет символ и строку?
3. Какой стандартный предикат используется для разделения строки в список лексем?
4. Какой предикат служит для сцепления строк?
5. Какой стандартный предикат служит для разделения строки на две части так, что одна содержит первые *N* символов, а другая содержит остаток?
6. Какой предикат возвращает длину строки?
7. Какой стандартный предикат служит для чтения полных строк символов?
8. Какой стандартный предикат служит для ввода целых чисел?
9. Какой стандартный предикат служит для ввода вещественных чисел?
10. Какой стандартный предикат служит для ввода символьных величин?
11. Какой стандартный предикат служит для чтения файлов с дисков?
12. Какой предикат служит для удаления файла?

13. Какой предикат служит для сохранения файла?
14. Какой предикат служит для переименования файла?
15. Какой предикат служит для проверки на наличие файла с таким именем?
16. Какой предикат служит для открытия файла для записи или создания?

5. РАЗРАБОТКА ИГРОВЫХ ПРОГРАММ

Цель и содержание: ознакомление с основными приемами разработки игровых программ в системе Turbo Prolog 2.0.
Организационная форма занятий: решение проблемных задач, разбор конкретных ситуаций.
Вопросы для обсуждения на лабораторном занятии: примеры игровых программ.

Теоретическое обоснование

В качестве примера рассмотрим следующую задачу: необходимо нарисовать конверт, не отрывая карандаша от бумаги и не проводя два раза по одной и той же линии.
Обозначим ребра латинскими буквами, вершины – цифрами.
Знания о структуре графа можно представить следующим образом:
rebro (a, 1, 2), rebro (a, 2, 1), rebro (b, 2, 3), rebro (b, 3, 2), rebro (c, 1, 3), rebro (c, 1, 2), rebro (c, 3, 1), rebro (d, 2, 4), rebro (d, 4, 2), rebro (e, 2, 5), rebro (e, 5, 2), rebro (f, 3, 4), rebro (f, 4, 3), rebro (g, 3, 5), rebro (g, 5, 3), rebro (h, 4, 5), rebro (h, 5, 4).
Решением задачи должен явиться список пройденных ребер графа, причем длина его должна быть равна 8, и в нем не должно быть повторяющихся ребер, что можно описать так: prohod (T, P):- length (P, 8), write_list (P), !. prohod (T, P):- rebro(R, T, H), not (member (R,P)), prohod (H, [R|P]). где *T* – текущая вершина графа, а *P* – список пройденных ребер.
Первое правило означает, что если длина списка *P* пройденных вершин становится равной 8, то список *P* выводится на печать. Это правило ограничивает рекурсивный перебор вершин и ребер, проводимый вторым правилом.
Второе правило является генератором перебора, оно перебирает предикаты *rebro* и находит такое ребро *R*, из текущей вершины *T* в новую *H*, чтобы оно не принадлежало списку *P*, зато это ребро добавляется в качестве головы к списку *P*, и поиск дальнейшего пути производится уже из новой вершины *H*.

Аппаратура и материалы. Требования к техническому и программному обеспечению приведены в приложении В.
Указания по технике безопасности приведены в приложении Г.

Методика и порядок выполнения работы

В следующем примере реализуется алгоритм логической игры «Угадай число».

```
/* Логическая игра «Угадай число» */ predicates
     play_the_game     give_info
          play_it
     generate_rand(integer)      play_it_sam(integer)
     test_and_tell(integer,integer)
          say_you_got_it_right(integer)
          say_too_big
     say_too_small goal     play_the_game. clauses
     play_the_game:-give_info, play_it.
     /*информация для пользователя*/     give_info:-
makewindow(1,7,7,"",0,0,25,80),
     makewindow(2,7,7,"Угадай число", 2,20,22,45),
     nl, write("Это игра Угадай число"),          nl,
write("Я загадаю число из"),             nl,
write("интервала [1,100]"),
          nl, write("Я буду подсказывать больше"),
     nl, write("или меньше задуманное число"),nl,
     nl, write("Будешь готов - нажми space bar"),
     nl,readchar(_), clearwindow.        /* выполнение игры
*/
  play_it:-generate_rand(A),  play_it_sam(A).    /*
генерация случайного числа */
          generate_rand(X):- random(R),
               X=1+R*100,
          nl,write("Я загадал число"),
     nl,write("Теперь дело за Вами"),nl.          /*запрос к
пользователю */   play_it_sam(X):-           nl, write
("Введите свой вариант"),               nl, readint(G),
test_and_tell(X,G),        play_it_sam(X).
          /* проверка и выдача результата*/
     test_and_tell(X,X):- say_you_got_it_right(X),!.
     test_and_tell(X,Y):- X>Y, say_too_big,!.
     test_and_tell(_,_):- say_too_small.          /*выдача
сообщений*/       say_too_big:- nl,
               write("Загаданное число больше").
     say_too_small:- nl,  write("Загаданное число меньше").
          say_you_got_it_right(X):- nl, write("Вы правы"),
          nl,write("Я загадал ",X),           nl, write("До
новых встреч!"),           nl, write("Нажмите space bar"),
          nl,readchar(_), exit.
```

В следующем примере реализуется алгоритм логической игры «Баше – 23 спички»: на столе 23 спички, два игрока по очереди берут от 1 до 3 спичек, проигравшим считается тот, кто возьмет последнюю спичку.

```
/* Логическая игра «Игра Баше – 23 спички» */ predicates
     play_the_game     do_windows
     play(integer,integer,integer)
     play_it_again(integer,integer,integer)
     real_int(real,integer) goal     play_the_game. clauses
     play_the_game:- do_windows,play(23,0,0).      /* правило
для образования окон */    do_windows:-
makewindow(1,7,7,"",0,0,25,80),
     makewindow(2,7,7,"Игра 23 спички", 1,5,22,40),
     nl, write("Добро пожаловать!"),               M=23, H=0,
C=0,
          nl, write("Всего ",M,"  спички"),             nl,
write("По очереди будем перемещать "),     nl,
write("спички в свои кучки"), nl,
               write("За раз можно взять 1, 2, 3 спички"),nl,
          nl,  write("Проиграет забравший последнюю
спичку"),
          nl,  write("Итак, начнем, всего ", M, "сп."),nl,
               nl,
                write("я взял ",C," сп. Вы – ",H, " сп."),nl.
     play(M,H,C):- play_it_again(M,H,C).      /*правило
определения победителя*/        play_it_again(M,_,_):- M<=0,
          nl, write("Вы выиграли!"),           nl,
write("Нажмите любую клавишу для выхода"),
     readchar(_), clearwindow,!,exit.
     play_it_again(1,_,_):- nl, write("Я выиграл!"),
     nl, write("Нажмите любую клавишу для выхода"),
     readchar(_), clearwindow,!,exit.
     play_it_again(M,_,_):- nl, write("Ваш ход"),
     nl, write("Сколько спичек вы хотите взять?"),
    readint(Hn),   M2=M-Hn,   H2=Hn,          write("Осталось
",M2,"сп."),    nl, write("Мой ход"),  random(F), Rea=1+3*F,
real_int(Rea,Rint), M3=M2-Rint,   nl, write("Я беру ",Rint,"
сп."),          nl,   write("Осталось   ",M3,   "   сп."),nl,
M7=M3,H7=H2,C7=Rint,                play_it_again(M7,H7,C7).
/*вспомогательное правило*/  real_int(Re,In):- In=trunc(Re).
```

Содержание отчета и его форма

В отчете к лабораторной работе должно быть указано название работы; перечислены этапы выполнения лабораторной работы и краткая характеристика работ на каждом этапе решения задач, описанных в примерах.

Вопросы для защиты работы

1. Какие основные приемы разработки игровых программ используются в системе Turbo Prolog?
2. Какие основные стандартные предикаты используются при разработке игровых программ в системе Turbo Prolog?

6. РАЗРАБОТКА ПРОГРАММЫ ПОИСКА НАИЛУЧШЕГО МАРШРУТА

Цель и содержание: ознакомление с основными приемами разработки программ поиска наилучшего маршрута в системе Turbo Prolog 2.0.
Организационная форма занятий: решение проблемных задач, разбор конкретных ситуаций.
Вопросы для обсуждения на лабораторном занятии: пример программы поиска наилучшего маршрута.

Теоретическое обоснование

Рассмотрим следующий принцип построения маршрута: для того, чтобы найти маршрут между городами X и Z, необходимо:
(1) если между X и Z имеются ключевые пункты Y1, Y2, …, то найти один из путей:
путь из X в Z через Y1, или путь из X в Z через Y2, или
…
(2) если между X и Z нет ключевых пунктов, то найти такой соседний с X город Y, что существует маршрут из Y в Z.

Аппаратура и материалы. Требования к техническому и программному обеспечению приведены в приложении В.
Указания по технике безопасности приведены в приложении Г.

Методика и порядок выполнения работы

В приведенной программе приведен пример реализации алгоритма поиска наилучшего маршрута.

```
domains sl=symbol*
predicates a(symbol,symbol,integer,integer,integer)
b(symbol,symbol,integer,integer,integer) мин(integer)
мин_путь(integer,integer,integer,sl)
путь(symbol,symbol,sl,integer,integer,integer)
путь1(symbol,symbol,sl,sl,integer,integer,integer,
integer,integer,integer)
принадл(symbol,sl)
добавить(integer,integer,integer,sl)
```

```
clauses a(a,b,100,10,2).  a(b,c,200,10,1).
a(d,c,800,50,4). a(d,f,300,20,3).  a(f,c,400,25,3).
a(a,c,400,15,2). a(a,f,700,42,4).
 b(X,Y,C,P,B):- a(X,Y,C,P,B); a(Y,X,C,P,B).
 путь(Н,К,Путь,Ст,Расст,Вр):-
              путь1(К,Н,[К],П,0,С,0,Р,0,В),
              добавить(С,Р,В,П),              fail;
     мин_путь(Ст,Расст,Вр,Путь).

добавить(С,Р,В,П):-
              мин(Т),Т<С,!;                мин(Т),Т>С,
     retract(мин(_)),!,
     retractall(мин_путь(_,_,_,_)),
     assert(мин_путь(С,Р,В,П)),
     assert(мин(С));              мин(_),!,
     assert(мин_путь(С,Р,В,П));
     assert(мин(С)),              assert(мин_путь(С,Р,В,П)).
 путь1(Y,Y,П,П,C,C,P,P,B,B):- !.
путь1(X,Y,П0,П,С0,С,Р0,Р,В0,В):-
              мин(Тек),               b(X,Z,C1,P1,B1),
          С2=С0+С1,С2<=Тек,                not(принадл(Z,П0)),
              Р2=Р0+Р1,
              В2=В0+В1,
              путь1(Z,Y,[Z|П0],П,С2,С,Р2,Р,В2,В).
путь1(X,Y,П0,П,С0,С,Р0,Р,В0,В):-
              not(мин(_)),            b(X,Z,C1,P1,B1),
          not(принадл(Z,П0)),
              С2=С0+С1,
              Р2=Р0+Р1,               В2=В0+В1,
     путь1(Z,Y,[Z|П0],П,С2,С,Р2,Р,В2,В).
 принадл(X,[X|_]):- !.
принадл(X,[_|T]):- принадл(X,T).
 goal X="a",Y="d",путь(X,Y,Путь,Стоим,Рас,Вр).
```

Содержание отчета и его форма

В отчете к лабораторной работе должно быть указано название работы; перечислены этапы выполнения лабораторной работы и краткая характеристика работ на каждом этапе решения задачи, описанной в примере.

Вопросы для защиты работы

1. Какие основные приемы разработки программ поиска наилучшего маршрута используются в системе Turbo Prolog?
2. Какие основные стандартные предикаты используются при разработке программ поиска наилучшего маршрута в системе Turbo Prolog?

7. РАЗРАБОТКА ЭКСПЕРТНОЙ СИСТЕМЫ

Цель и содержание: ознакомление с основными приемами разработки экспертных систем в системе Turbo Prolog 2.0.
Организационная форма занятий: решение проблемных задач, разбор конкретных ситуаций.
Вопросы для обсуждения на лабораторном занятии: пример программы разработки экспертной системы.

Теоретическое обоснование

Рассмотрим пример экспертной системы для идентификации породы собак. Система поможет потенциальному хозяину выбрать породу собаки в соответствие с определенными критериями.
В данной экспертной системе используются следующие характеристики:
1) короткая шерсть;
2) длинная шерсть;
3) рост меньше 30 дюймов;
4) рост меньше 22 дюймов;
5) низкопосаженный хвост;
6) длинные уши; 7) хороший характер
8) вес больше 100 фунтов.
Каждая характеристика для конкретной породы либо верна, либо не верна. Для каждой породы справедливы характеристики, представленные в таблице 12.1.

Таблица 12.1 – Характеристики пород собак

Порода	Характеристики							
	1	2	3	4	5	6	7	8
Английский бульдог	*			*	*		*	
Гончая	*			*		*	*	
Дог	*		*			*	*	*
Американская гончая	*				*	*	*	
Кокер-спаниель		*		*	*	*	*	
Ирландский сеттер		*	*			*		
Колли		*	*		*		*	

Сенбернар		*			*		*	*

Аппаратура и материалы. Требования к техническому и программному обеспечению приведены в приложении В.

Указания по технике безопасности приведены в приложении Г.

Методика и порядок выполнения работы

Рассмотрим пример экспертной системы «Эксперт по породам собак».

```
/* Назначение: Демонстрация работы ЭС */ domains
     n=integer     list=n*
     dog=symbol predicates
     rule(n,dog,list)       cond(n,string)     do_expert
     show_menu     do_consulting     process(n)
     test(n,list)       topic
     repeat goal        do_expert. clauses
     rule(1,"английский бульдог",[1,4,5,7]).
     rule(2,"гончая",[1,4,6,7]).
     rule(3,"дог",[1,3,6,7,8]).      rule(4,"американская
гончая",[1,5,6,7]).    rule(5,"коккер-
спаниель",[2,4,5,6,7]).     rule(6,"ирландский
сеттер",[2,3,6]).      rule(7,"колли",[2,3,5,7]).
     rule(8,"сенбернар",[2,5,7,8]).      /*Характеристики*/
     cond(1,"короткошерстная").
     cond(2,"длинношерстная").       cond(3,"рост ниже 30
дюймов").     cond(4,"рост ниже 22 дюймов").
     cond(5,"низкопосаженный хвост").     cond(6,"большие
уши").   cond(7,"хороший характер").     cond(8,"вес более
100 фунтов").

          do_expert:-
     makewindow(1,7,5 ,"ЭКСПЕРТНАЯ СИСТЕМА",0,0,25,80),
     show_menu.

     repeat.       repeat:-repeat.

          /*Вывод меню*/     show_menu:-
               repeat,
   write("***************************"),nl,
write("*****Добро   пожаловать!*****"),nl,          write("*
*"),nl,             write("*****1-консультация********"),nl,
write("*****2-список*************"),nl,      write("*****3-
выход**************"),nl,                          write("*
*"),nl,          write("****Сделайте   свой   выбор****"),nl,
readint(X), process(X),fail.
```

```
            /*Обработка 1 пункта меню “Консультация”*/
    process(1):-          do_consulting, readchar(_),
    shiftwindow(1), clearwindow.
            /*Обработка 2 пункта меню “Вывод списка”*/
    process(2):-          makewindow(2,7,7,"",5,20,12,25),
        topic, readchar(_),        shiftwindow(1),
clearwindow.          /* Обработка 3 пункта меню “Выход”*/
    process(3):-          removewindow, exit.
    /*Вывод пород собак*/
    topic:-           rule(X,Y,_), write(X,".  ",Y), nl,
fail.    topic.
   /*Консультация*/      do_consulting:-      test(1,List),
rule(_,X,List),    write("Ваш выбор:"  ,X),!.
    do_consulting:-       write("Мне жаль, что не смог Вам
помочь.").
         /*Тестирование*/       test(9,[]):-!.
    test(1,[N|List]):- cond(N,Text),
    makewindow(2,7,7,"",5,20,10,35),
    write("Вопрос:-",Text,"?"),nl,          write("1-
да"),nl,          write("0-нет"),nl,
    readint(R),R=1,!,test(3,List).      test(1,List):-
test(2,List),!.       test(N,[N|List]):- cond(N,Text),
    makewindow(2,7,7,"",5,20,10,35),
    write("Вопрос:-",Text,"?"),nl,          write("1-да"),
nl,          write("0-нет"), nl,        readint(R), M=N+1,
R=1,!,test(M,List).
         test(N,List):- M=N+1, test(M,List).
```

Содержание отчета и его форма

В отчете к лабораторной работе должно быть указано название работы; перечислены этапы выполнения лабораторной работы и краткая характеристика работ на каждом этапе решения задачи, описанной в примере.

Вопросы для защиты работы

1. Какие основные приемы разработки экспертных систем используются в системе Turbo Prolog?
2. Какие основные стандартные предикаты используются при разработке экспертных систем в Турбо Прологе?

8. ОНТОЛОГИЧЕСКАЯ ИНЖЕНЕРИЯ ЗНАНИЙ В СИСТЕМЕ PROTÉGÉ: СОЗДАНИЕ ПРОЕКТА, СОХРАНЕНИЕ ПРОЕКТА, СОЗДАНИЕ КЛАССОВ, СОЗДАНИЕ СЛОТОВ

Цель и содержание: ознакомление с основными приемами создания проекта в системе Protégé; ознакомление с основными приемами создания классов в системе Protégé; ознакомление с основными приемами создания слотов в системе Protégé.
Организационная форма занятий: решение проблемных задач, разбор конкретных ситуаций.
Вопросы для обсуждения на лабораторном занятии: пример создания проекта в системе Protégé; пример создания классов в системе
Protégé; пример создания слотов в системе Protégé.

Теоретическое обоснование

Онтология описывает основные концепции (положения) предметной области и определяет отношения между ними.
Процесс построения онтологий состоит из создания следующих блоков:
– Классов и их свойств (classes, properties).
– Свойств каждой концепции, описывающих различные функциональные возможности и атрибуты концепции (слоты (slots), иногда называемые роли).
– Ограничения по слотам (также известных как аспекты / грани (slot facets), иногда называемые ограничения ролей).
Онтология вместе с множеством индивидуальных экземпляров классов составляют базу знаний.
Онтология – формальное явное описание понятий в рассматриваемой предметной области (классов, иногда их называют понятиями), свойств каждого понятия, описывающих различные свойства и атрибуты понятия (слотов (иногда их называют ролями или свойствами)), и ограничений, наложенных на слоты (фацетов, иногда их называют ограничениями ролей). Онтология вместе с набором индивидуальных экземпляров классов образует базу знаний.
Потребность в разработке онтологии возникает по ряду причин:
– Для совместного использования людьми или программными агентами общего понимания структуры информации.
– Для возможности повторного использования знаний в предметной области.
– Для того чтобы сделать допущения в предметной области явными.
– Для отделения знаний в предметной области от оперативных знаний.
– Для анализа знаний в предметной области.
Совместное использование людьми или программными агентами общего понимания структуры информации является одной из наиболее общих целей разработки онтологий. К примеру, пусть несколько различных веб-сайтов

содержат информацию по медицине или предоставляют информацию о платных медицинских услугах, оплачиваемых через Интернет. Если эти веб-сайты совместно используют и публикуют одну и ту же базовую онтологию терминов, которыми они все пользуются, то компьютерные агенты могут извлекать информацию из этих различных сайтов и накапливать ее. Агенты могут использовать накопленную информацию для ответов на запросы пользователей или как входные данные для других приложений.

Обеспечение возможности использования знаний предметной области стало одной из движущих сил недавнего всплеска в изучении онтологий. Например, для моделей многих различных предметных областей необходимо сформулировать понятие времени. Это представление включает понятие временных интервалов, моментов времени, относительных мер времени и т. д. Если одна группа ученых детально разработает такую онтологию, то другие могут просто повторно использовать ее в своих предметных областях. Кроме того, если нам нужно создать большую онтологию, мы можем интегрировать несколько существующих онтологий, описывающих части большой предметной области.

Создание явных допущений в предметной области, лежащих в основе реализации, дает возможность легко изменить эти допущения при изменении наших знаний о предметной области. Жесткое кодирование предположений о мире на языке программирования приводит к тому, что эти предположения не только сложно найти и понять, но и также сложно изменить, особенно непрограммисту. Кроме того, явные спецификации знаний в предметной области полезны для новых пользователей, которые должны узнать значения терминов предметной области.

Отделение знаний предметной области от оперативных знаний – это еще один вариант общего применения онтологий. Можно описать задачу конфигурирования продукта из его компонентов в соответствии с требуемой спецификацией и внедрить программу, которая делает эту конфигурацию независимой от продукта и самих компонентов. После этого можно разработать онтологию компонентов и характеристик ЭВМ и применить этот алгоритм для конфигурирования нестандартных ЭВМ.

Анализ знаний в предметной области возможен, когда имеется декларативная спецификация терминов. Формальный анализ терминов чрезвычайно ценен как при попытке повторного использования существующих онтологий, так и при их расширении.

Часто онтология предметной области сама по себе не является целью. Разработка онтологии сродни определению набора данных и их структуры для использования другими программами. Методы решения задач, доменнонезависимые приложения и программные агенты используют в качестве данных онтологии и базы знаний, построенные на основе этих онтологий.

Универсального подхода к созданию онтологий, который бы привел к однозначно успешному результату, не существует. Процесс создания

онтологий обычно является итеративным, т. е. сначала создается черновой набросок, а затем по мере необходимости происходит возврат для определения деталей, и так продолжается до тех пор, пока онтология не будет отражать концепцию предметной области с определенной степенью.

Практически, создание онтологий включает:

1. Определение классов в онтологии.
2. Организация классов в некоторую иерархию (базовый класс –> подкласс).
3. Определение слотов и их допустимых значений,
4. Заполнение значений слотов для экземпляров классов.

Для любой предметной области может существовать бесчисленное количество онтологий; ведь каждая новая онтология – это всего лишь еще один из способов структурирования концепций и отношений между ними. Однако существуют несколько простых принципов, которые могут помочь при принятии решений о том, как создавать те или иные онтологии:

– Не может быть только одного способа описания модели предметной области – всегда есть жизнеспособная альтернатива. Лучшее решение почти всегда будет зависеть от того, какая система разрабатывается и от возможных будущих изменений в системе.
– Процесс разработки обязательно должен быть итеративным.
– Концепции в онтологии должны быть максимально близки к объектам (логическим или физическим) и отношениям между ними в интересуемой области знаний. При правильном моделировании онтология может быть представлена предложениями, где вероятней всего в качестве существительных будут объекты, а отношений – глаголы.

Для начала необходимо понять, для чего должна быть использована онтология и как примерно выглядел бы ее детальный и общий вариант. Затем среди многих альтернатив необходимо выбрать ту, которая будет лучше всего работать для намеченной задачи, а также будет наиболее интуитивна, расширяема, поддерживаема. При этом необходимо помнить, что онтология представляет предметную область в реальном окружающем мире, и потому понятия в онтологии также должны отражать эту реальность.

После того как создан черновой вариант онтологии, можно проверить ее и откорректировать, используя Protégé. Такой процесс итеративной коррекции, будет продолжаться на протяжении всего жизненного цикла онтологии.

Предположим, что необходимо разработать систему, которая помогает управлять стоимостью и организацией печатного издания (для простоты можно взять некую газету). Система должна отвечать на следующие вопросы:

– Кто ответственный за каждый раздел в газете?
– Каково содержимое каждой статьи в разделе, и кто автор?
– Перед кем отчитывается каждый автор?
– Каково расположение и расходы на каждую статью?

После того как определена идея, следует расписать некоторые из важных положений системы. Сюда могут войти: основные концепции и их свойства, а

также отношения между ними. Для начала можно просто определить термины, независимо от роли, которую они могут играть в онтологии.
Итак, в любой газете есть разделы. Каждый раздел имеет содержимое, например, статьи, реклама и т. д. и ответственного редактора. У каждой статьи есть автор, который может быть, как работником газеты, так и быть приглашенным со стороны. Для каждого автора, работающего в газете, необходимо знать его имя и зарплату, а также перед кем он отчитывается.
По мере определения понятий, неявно определяются рамки создаваемой онтологии, а именно, что необходимо включить в создаваемую модель, а что нет. К примеру, при начальном рассмотрении термина «работник», возможно включение в это понятие вахтера или водителя грузовика из службы доставки. Однако, следует осознать, что онтология должна была сфокусирована на производственных затратах, связанных напрямую с тем что, как и где написано в газете. Таким образом, принимается решение не включать вахтера и т. п. в область рассмотрения.
Получив достаточно полный список терминов, можно разделить эти понятия по категориям в зависимости от их функции в онтологии. Понятия (концепции / термины предметной области), являющиеся объектами, такие как статья или автор, будут представлены в виде классов. Свойства классов, такие как содержимое раздела или зарплата, могут быть представлены как слоты, а ограничения на свойства или отношения между классами как грани / аспекты (slot facets).
Основное окно программы Protégé состоит из закладок (tabs) которые отображают различные аспекты модели знаний. Наиболее важной закладкой, когда Вы только начинаете делать проект, является закладка классов (Classes). Обычно классы соответствуют объектам или типам объектов, в некой предметной области. В примере с газетой, классы будут включать в себя людей, а именно, редакторов, репортеров, агентов по продаже, а также компоненты расположения информации газеты, такие как разделы, кроме того, содержимое газеты (реклама и статьи) будет также представлено в виде объектов.
Классы в Protégé отображаются в виде иерархии наследования (inheritance hierarchy), которая располагается в области просмотра называемой Class Browser (или навигатор классов) в левой части закладки классов. Свойства классов, выбранных в текущий момент в навигаторе, будут отображены в редакторе классов справа. Ниже вы узнаете, как создавать классы, подклассы, изменять иерархию классов, создавать абстрактные классы и добавлять дополнительные базовые классы к существующим классам.
В системе Protégé под классами понимаются конкретные понятия (концепции) предметной области, такие как редактор или корреспондент. В то же время классы – это больше, чем объекты, объединенные в иерархию.
Они также могут иметь атрибуты (свойства), к примеру, имя, номер телефона или уровень зарплаты и отношения между ними, такие как Автор Статьи.

Атрибуты и отношения класса описываются конструкцией под названием слот. В данном разделе будет показано, как создавать слоты, привязывать слоты к классам, описывать отношения между классами, а также будет описан механизм наследования слотов.

Аппаратура и материалы. Требования к техническому и программному обеспечению приведены в приложении В.

Указания по технике безопасности приведены в приложении Г.

Методика и порядок выполнения работы

Создание проекта. Чтобы создать новый проект в системе Protégé необходимо:

1. Запустите Protégé. Если проект уже открыт, необходимо сохранить его и перезапустить программу. После того как программа запустилась, появляется диалог приветствия, предлагающий создать новый проект, открыть последний проект или посмотреть документацию.

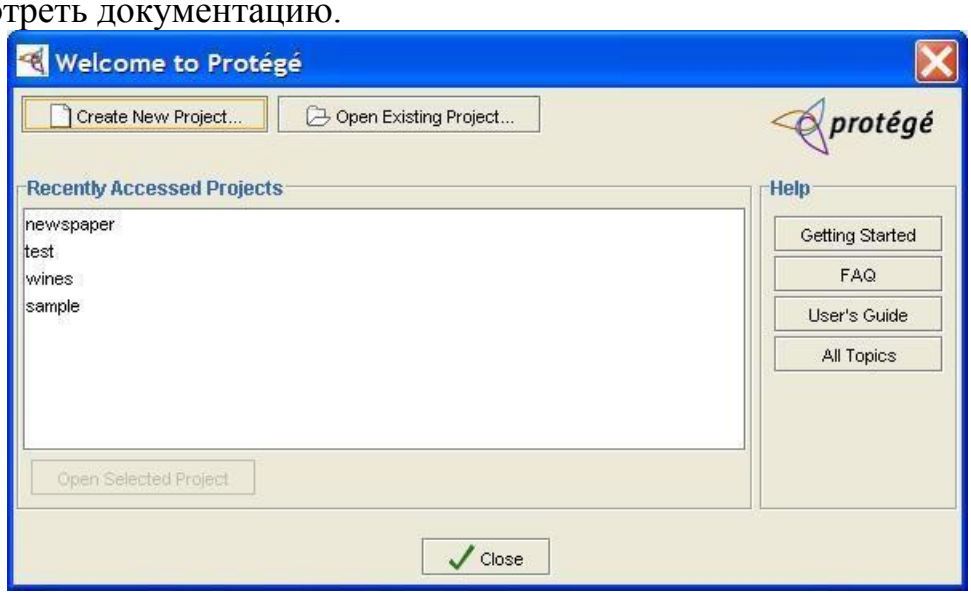

Рисунок 13.1 – Окно приветствия

2. Щелкните мышкой по кнопке **Create New Project...**. Появится диалоговое окно *Create New Project*, позволяющее выбрать тип проекта. Если нет необходимости в специальном формате для файлов, необходимо нажать кнопку **Finish** – будет выбран формат файла по умолчанию Protege Files (.pont and pins).

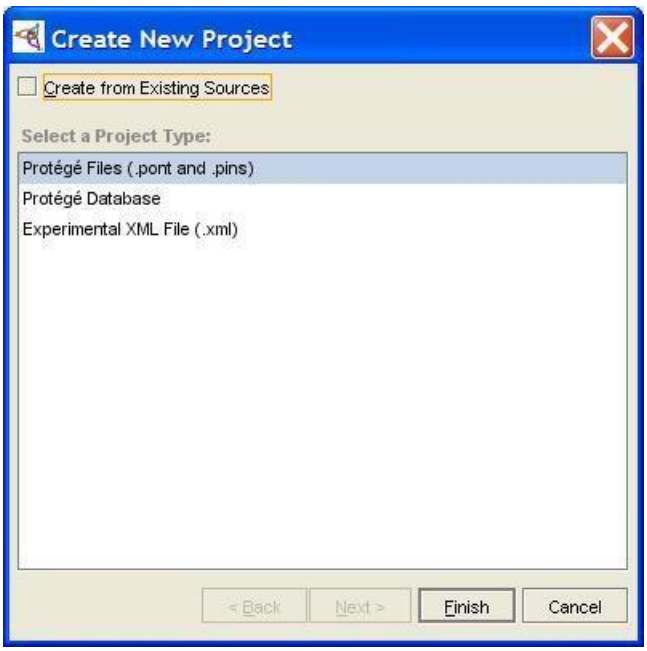

Рисунок 13.2 – Окно создания нового проекта

3. Откроется окно проекта Protégé. Новый проект всегда открывается в области просмотра классов (Classes view). Видно, что в этой области на данный момент находятся только внутренние системные классы Protégé: THING и SYSTEM-CLASS. Никаких экземпляров классов создано к этому моменту не будет.

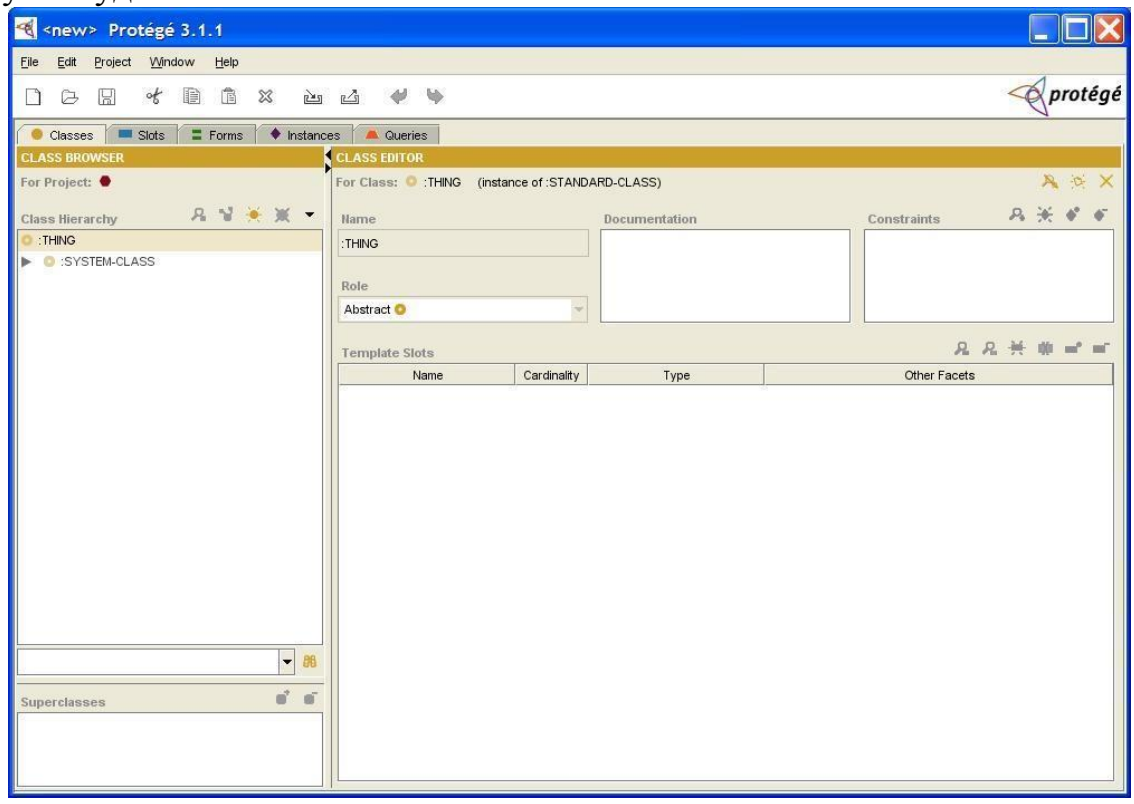

Рисунок 13.3 – Область просмотра классов

Сохранение проекта. Во время работы с программой можно сохранять промежуточные изменения, для этого:

1. Щелкните кнопку сохранить проект , также можно выбрать пункт **Save project** из меню файл **File**.

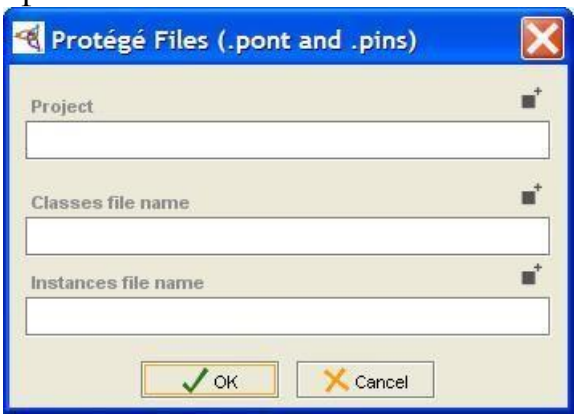

Рисунок 13.4 – Окно сохранения проекта

2. Для того чтобы указать место, куда будет сохранен Ваш проект, нажмите кнопку чуть выше самой верхней строчки (напротив надписи Project). В открывшемся диалоговом окне, перейдите по нужному пути в файловой системе (или создайте каталог, где будут храниться данные проекта и откройте созданную папку).

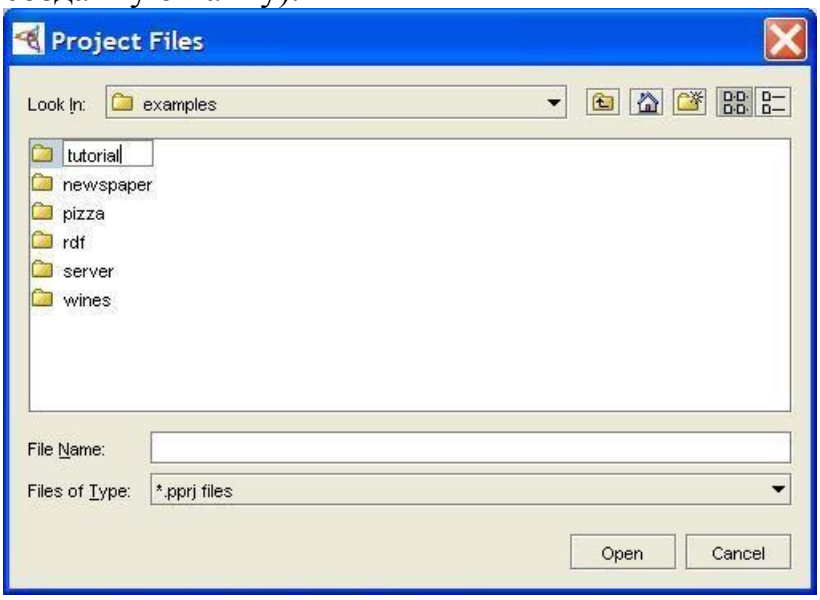

Рисунок 13.5 – Окно выбора имени файла проекта

3. Введите имя файла проекта (например, tutorial).
4. Нажмите кнопку **Select**.
5. Вы вернетесь в окно сохранения файлов проекта, нажмите **OK** и проект будет сохранен.

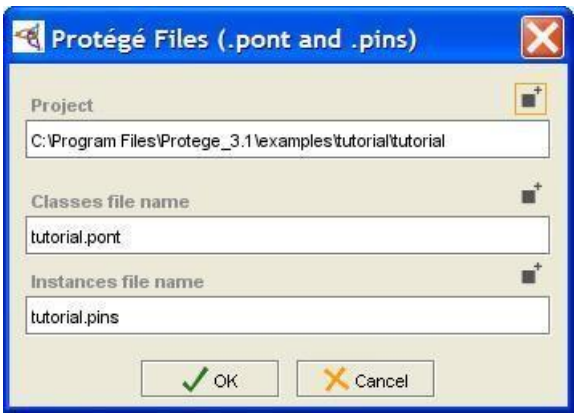

Рисунок 13.6 – Завершение сохранение проекта

Создание класса «Корреспондент». Для того, чтобы знать происхождение каждой статьи, необходимо начать с задания типов сотрудников или служб, которые могут создавать статьи. Для начала создадим новый класс «корреспондент» (Columnist):

1. Выберите закладку классов.
2. Найдите область в навигаторе классов (Class Browser), где отображается иерархия классов (Class Hierarchy, в окне Protégé слева). Эта область отображает иерархию классов, с выделенным текущим выбранным классом.

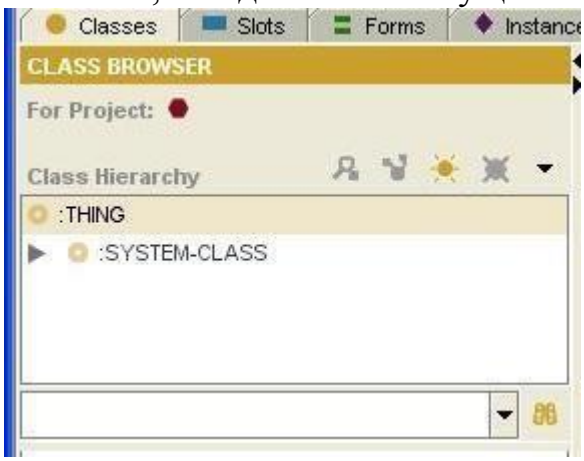

Рисунок 14.1 – Область отображения иерархии классов

3. Проверьте: по умолчанию класс :THING (вещь, нечто) должен быть выделен. Почти все классы в данном примере будут созданы на уровень ниже класса THING. Другой класс :SYSTEM_CLASS используется для определения структур различных форм Protégé.
4. Нажмите кнопку создать класс (Create Class)  в верхнем правом углу навигатора классов. Новый класс будет создан со стандартным именем (основанном на имени проекта). В нашем случае tutorial_Class_0. Вы можете увидеть, что имя нового класса в навигаторе классов после создания будет выделено, для указания того, что этот класс выбран в данный момент.
5. В активном поле редактора классов введите «Columnist». В системе Protégé приняты правила наименования, когда первая буква в каждом слове в

имени класса пишется в верхнем регистре, а остальные буквы в нижнем, при этом слова разделяются символом подчеркивания.

6. Нажмите ввод или щелкните мышью на отображаемый класс, чтобы подтвердить и отобразить свои изменения.

7. Если при изменении имени класса у вас возникли проблемы, посмотрите в панель редактора классов справа в главном окне Protégé. Стандартное имя нового класса должно быть отображено и выделено в поле Name. Если правильное стандартное имя отображается, но не выделено, просто щелкните на поле ***Name*** мышкой, для того чтобы его отредактировать. Если имя неправильное, тогда скорей всего выбран неверный класс в области отображения иерархии классов (Class Hierarchy), щелкните на нужном классе.

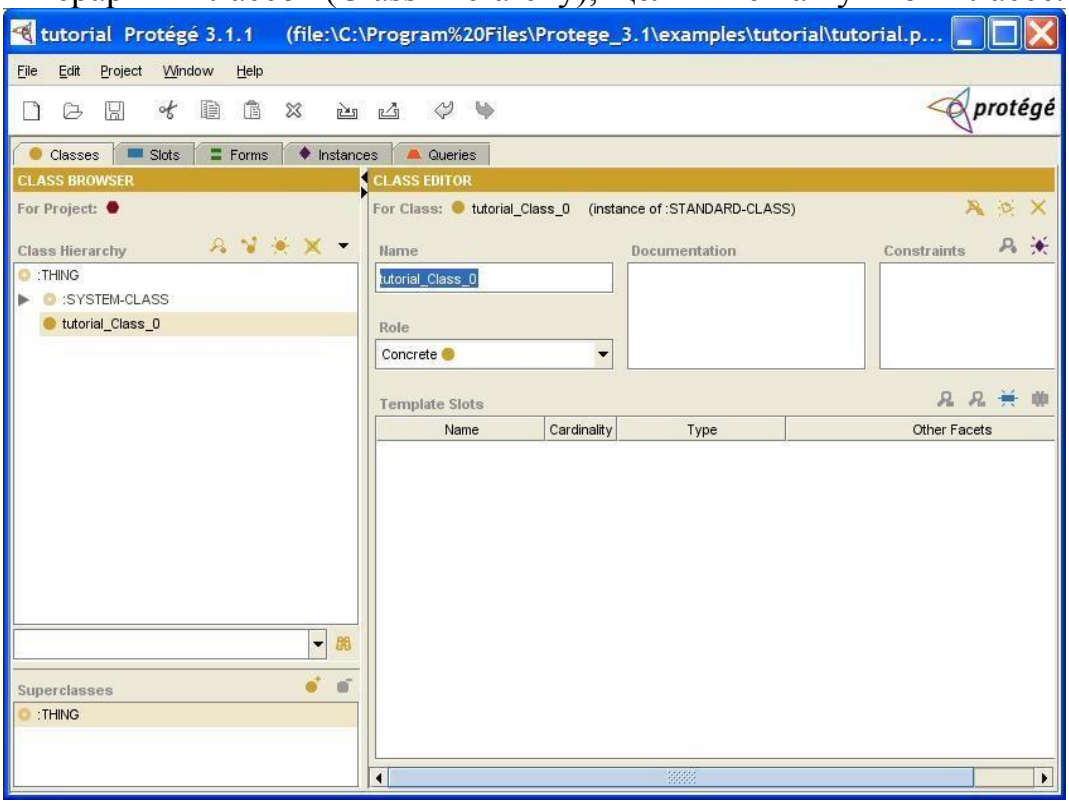

Рисунок 14.2 – Редактор класса

Создание класса «Автор». Автором может быть любой возможный источник информации для статьи, такой как новостная служба или корреспондент. Для того чтобы создать класс «Автор»:

1. Выделите класс :THING. Если вы этого не сделаете, то вы создадите класс, который будет подклассом класса Columnist.

2. Нажмите кнопку создать класс (Create class) и наберите с клавиатуры имя класса: Author.

3. Нажмите ввод для завершения создания класса.

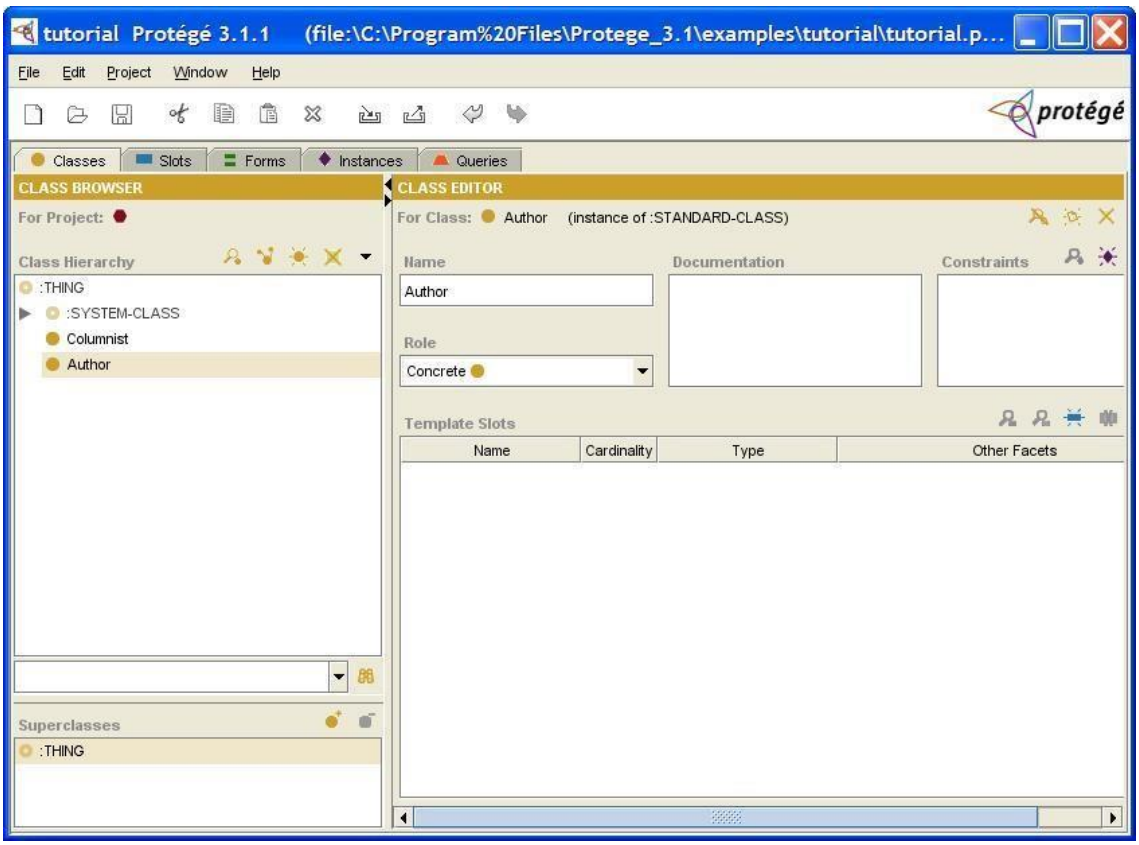

Рисунок 14.3 – Создание класса «Автор»

Создание подклассов класса «Автор». Чтобы добавить больше источников статей (служба новостей и редактор), необходимо создать их как подклассы класса автор (Author).

1. Выберите класс «Author» в области отображения иерархии классов.
2. Нажмите кнопку «создать класс» (Create class) и переименуйте новый класс в News_Service (служба новостей). Помните, что, когда бы вы не создавали новый класс, он будет создан как подкласс текущего выбранного класса. Также заметьте, что когда вы создаете первый подкласс класса, то иконки или появляются слева от него. Вы можете использовать эти иконки для того чтобы показать или скрыть подклассы класса. Продолжая разрабатывать онтологию, создадим еще один подкласс класса Автор (Author).
3. Выберите класс Автор (Author) в навигаторе классов (Class Browser).
4. Нажмите кнопку **Create Class** и переименуйте созданный класс в **Editor** (редактор).

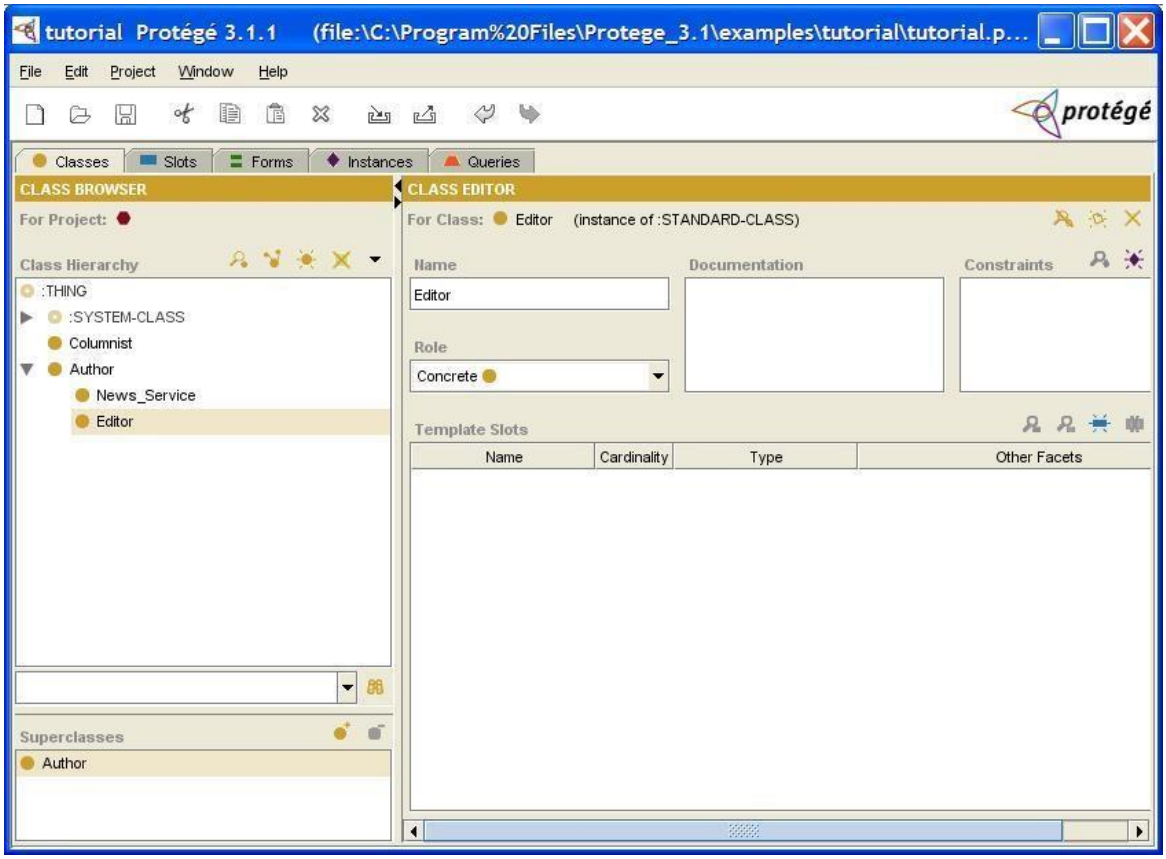

Рисунок 14.4 – Создание класса «Редактор»

Изменение иерархии классов. На данной стадии разработки, можно заметить что «Автор» (Author) и «Корреспондент» (Columnist) находятся на одном уровне в создаваемой иерархии, в то время как «Служба новостей» (News_Service) и «Редактор» (Editor) являются подклассами класса Автор (Author). С точки зрения концепции и служба новостей, и редактор, и корреспондент, все могут являться авторами (источниками) статей, т. е. понятие «Автор» является общим для всех этих трех классов. Значит, текущее расположение в иерархии противоречит принципам хорошо спроектированных онтологий – все классы, имеющие одно и то же более общее понятие, должны находиться на одном уровне в иерархии.

Таким образом, следует модифицировать иерархию, чтобы класс «Корреспондент» (Columnist) стал подклассом класса Автор (Author), правильно отражая структуру предметной области.

Для этого необходимо сделать следующее:

1. Щелкните на классе Columnist (Корреспондент) и перетащите (dragn-drop) его на класс Author (Автор).

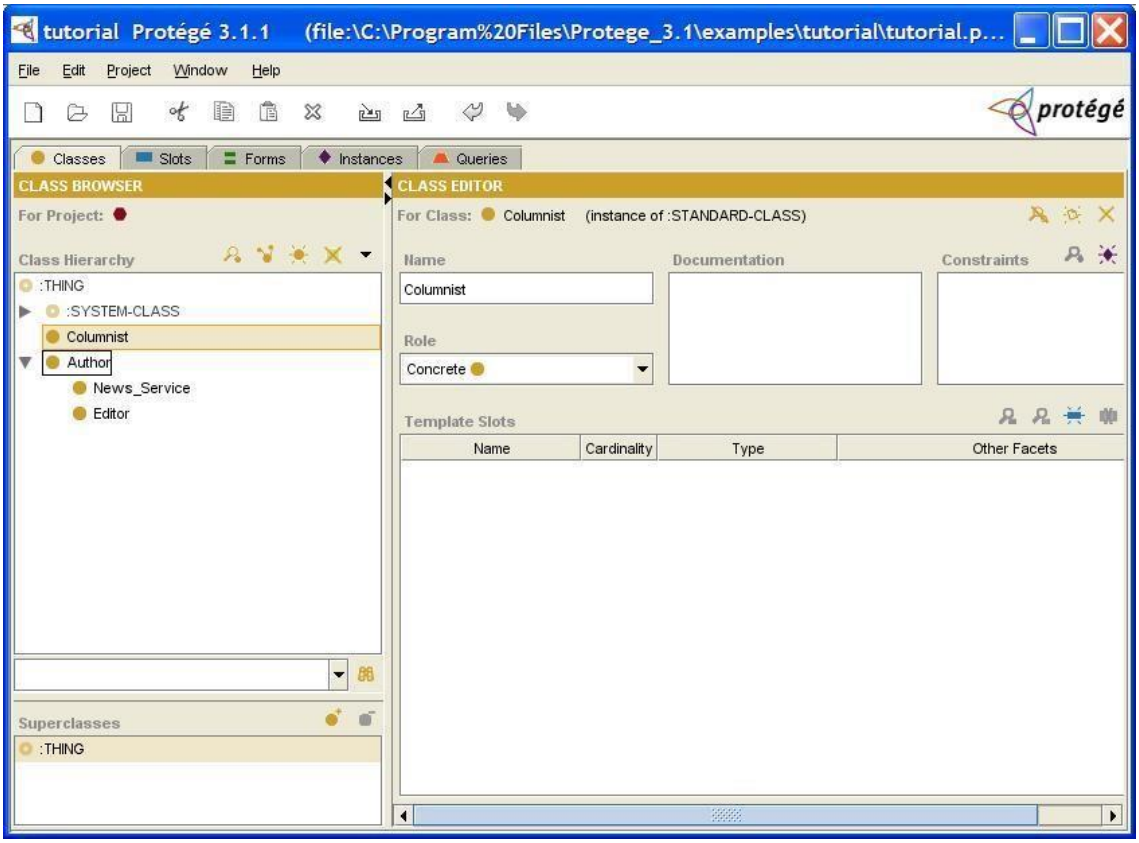

Рисунок 14.5 – Изменение иерархии классов

2. Класс **Columnist** будет удален с предыдущего места в иерархии и создан в новом, но уже как подкласс Author.

В данном случае, ошибка была введена явно, для примера. Однако, при создании собственных онтологий, в процессе разработки могут открываться отличия или сходства между классами, которые не были явно видны в начале и вам вероятней всего придется часто использовать перемещение, создание, удаление классов, для того чтобы создавать иерархии, оптимально отражающие положение вещей в предметной области.

Создание абстрактных классов. В системе Protégé классы могут быть как конкретными (Concrete), на основе таких классов система может непосредственно создавать готовые экземпляры, так и абстрактными, у таких классов не может быть экземпляров. По умолчанию, при создании класса выбирается тип класса как «конкретный класс». Так как в нашей системе понятие автора является скорее обобществляющим, нежели связанным с какой конкретной сущностью, мы делаем вывод что класс Автор (Author) не может сам по себе иметь экземпляров без более детального определения (к примеру, является ли автор службой новостей или корреспондентом).

Поэтому решаем сделать класс Автор (Author) абстрактным.

1. Выберите класс «Author» в области иерархии классов (Class Hierarchy). Далее в редакторе классов (справа от навигатора классов), найдите меню Роль (Role), оно должно находиться прямо под именем класса.

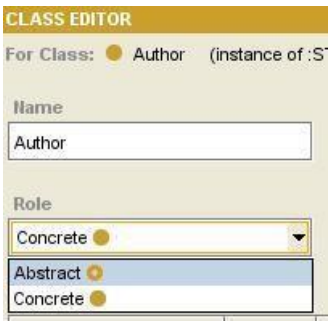

Рисунок 14.6 – Выбор меню Role

2. Щелкните по списку в меню *Role* и выберите «Abstract».
3. Когда вы меняете роль класса, иконка в перед именем класса меняется на . Такая иконка означает, что класс является абстрактным (соответственно означает конкретный класс).

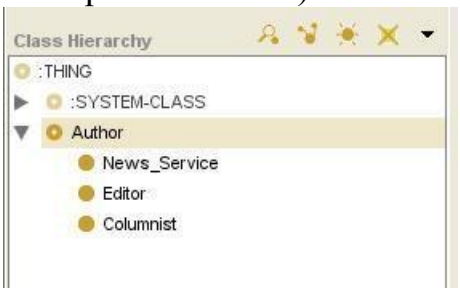

Рисунок 14.7 – Маркировка абстрактных классов

Создание класса «Работник». Теперь настало время создать класс «работник» (Employee). Этот класс подразумевает под собой любого работника газеты, независимо от того является ли он автором или нет. Заметим, однако, что нам интересны только те работники, которые как-то привлечены к созданию и управлению непосредственным содержимым газеты. Одним из выборов, который делает разработчик при проектировании онтологий, является отсечение лишних сущностей. Хотя изначально может казаться, что многие классы являются важными, все же необходимо следить затем, чтобы создаваемая иерархия не становилась слишком сложной. Применительно к нашему случаю, это означает, что хотя технически вахтер и является служащим компании, мы не будем создавать такой подкласс.

1. Выберите класс :THING в навигаторе классов. Несмотря на то, что некоторые авторы являются служащими газеты, мы не хотим, чтобы класс «Employee» (работник) был подклассом класса Автор (Author).
2. Нажмите кнопку **Create Class** и переименуйте вновь созданный класс в **Employee**.

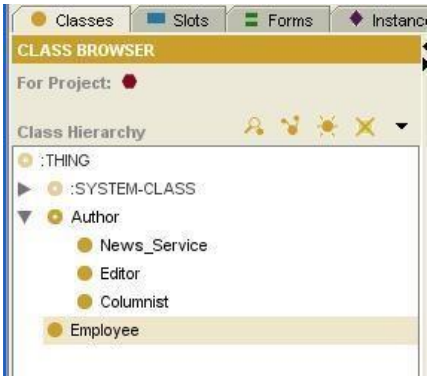

Рисунок 14.8 – Добавление класса Employee

Добавление дополнительного базового класса к существующему подклассу. Мы хотим чтобы «корреспондент» стал «работником». Но поскольку уже создан такой класс, то не будем создавать его снова как подкласс класса «работник». Вместо этого можно сделать так, чтобы существующий класс «корреспондент» (Columnist) стал подклассом «Employee» (работник). Для этого нужно сделать следующее:

1. Выберите класс Columnist (корреспондент) в навигаторе классов.

Рисунок 14.9 – Выделение класса в навигаторе иерархии классов

2. Найдите панель базовых классов (Superclasses) в нижней левой части окна системы Protégé (под навигатором классов). Заметьте, что, когда выбран класс Columnist (корреспондент), его базовый класс (Автор) отображается в панели Superclasses.

Рисунок 14.10 – Панель базовых классов

3. Нажмите кнопку **Add Superclass** (добавить базовый класс), в верхнем правом углу панели базовых классов (Superclasses). Появится диалоговое окно, отображающее все классы, созданные на тот момент времени, в виде иерархии.

Рисунок 14.11 – Выбор базового класса

4. Выберите класс «Employee» (работник) и нажмите **OK**. Теперь класс «корреспондент» (Columnist) имеет два базовых класса (Автор и Работник). Оба класса показаны в панели базовых классов.

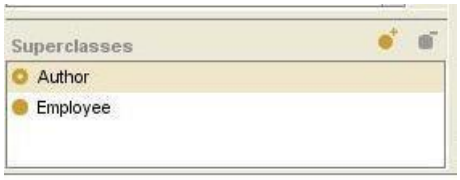

Рисунок 14.12 – Обновленный список базовых классов

5. Заметьте, что рядом с именем класса «Employee» появилась иконка ▶. Щелкните по ней, для того чтобы развернуть дерево подклассов и увидеть детей класса «работник» (Employee). Как видим, класс «корреспондент» теперь присутствует в двух местах в навигаторе классов: как подкласс класса «Автор» (Author) и еще раз как подкласс класса «Работник» (Employee).

Рисунок 14.13 – Обновленная иерархия классов

Добавление базового класса с помощью перетаскивания (drag-n-drop).

Существует еще одна возможность задания базового класса – при помощи механизма перетаскивания (drag-n-drop):

1. Выберите класс «Редактор» (Editor) в навигаторе классов.
2. Удерживая нажатой левую кнопку мыши, перетащите класс Editor (редактор), так чтобы он находился над классом Employee (работник). Класс Employee автоматически будет выделен.
3. Перед тем как отпустить кнопку мыши, нажмите дополнительно клавишу Ctrl, затем отпустите кнопку мыши, для того чтобы перенести класс.

Рисунок 14.14 – Добавление базового класса с помощью перетаскивания

Для удаления базового класса от некоторого подкласса, выберите класс, который должен быть удален в панели отображения базовых классов (Superclasses), и нажмите кнопку удаления базового класса (**Remove Superclass**).

Создание слота (используя закладку слоты (Slots tab)). Для создания слота есть несколько способов. Один из них – это создать слот, используя закладку «Slots», а затем связать его с одним или более классами. Вернемся к нашему примеру, для того, чтобы создать слот name, используя закладку Slots, необходимо:

1. Щелкнуть на закладку Slots. Заметьте, что расположение элементов управления на закладке слотов, схоже с закладкой классов, а именно, готовые слоты отображаются слева в области просмотра, а редактирование слотов возможно с помощью редактора (справа).
2. Нажмите кнопку создать слот (Create Slot) в правом верхнем углу панели отображения иерархии слотов (Slot Hierarchy). Будет создан новый

слот. Также, как и при создании класса, ему присваивается стандартное имя, в нашем случае tutorial_Slot_0 (имя будет автоматически
выделено, после создания слота).

3. Перед переименованием слота, убедитесь, что стандартное имя выделено в редакторе слота. Наберите новое имя слота (в нашем случае *name*). Рекомендованные правила наименования, таковы, что имя слота должно быть написано в нижнем регистре, при этом разные слова разделяются подчеркиванием. Такое наименование (классы с большой буквы, слоты с маленькой) помогает отличить классы от слотов в созданной онтологии.

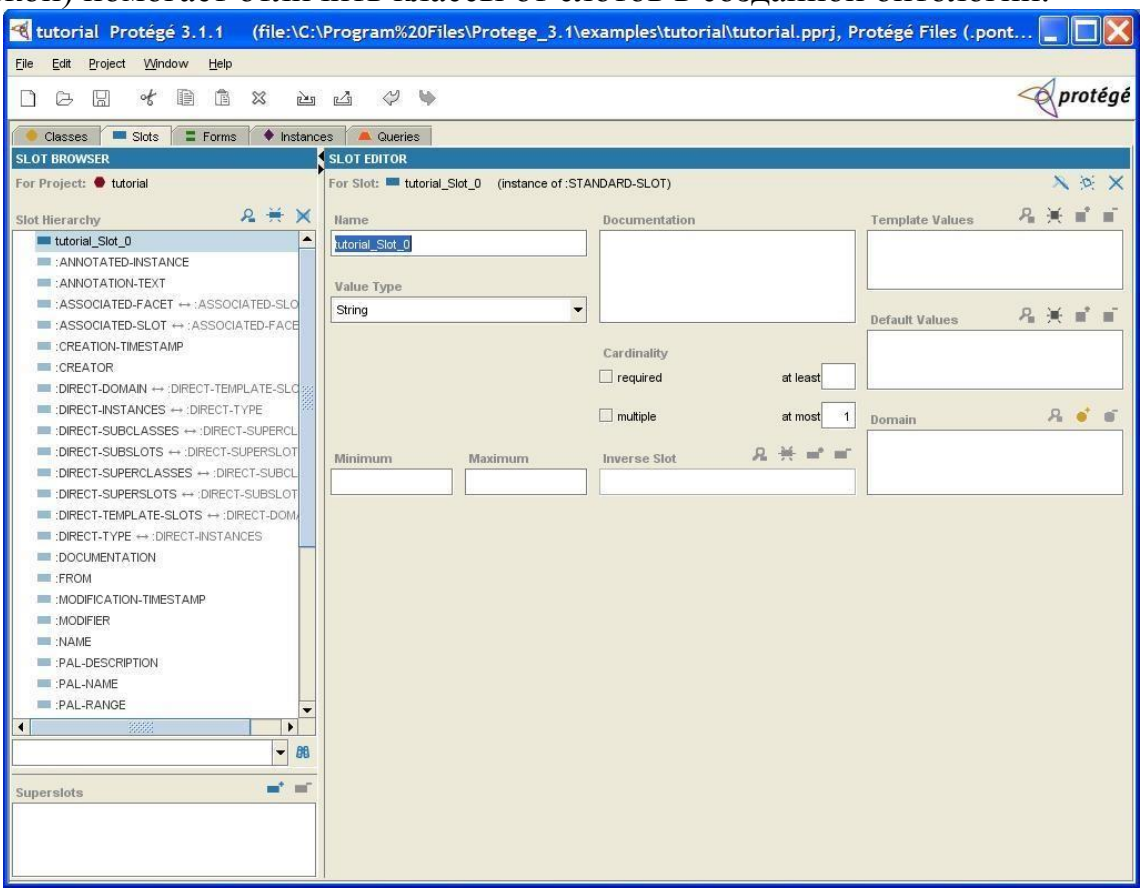

Рисунок 15.1 – Добавление слота

4. Заметьте, что слот имеет по умолчанию тип значения String (строка). Тип накладывает ограничения на то, какие значения может принимать слот. Строковый слот, к примеру, может принимать в качестве значений алфавитно-цифровые строки (включая пробелы).

Для этого простого слота мы не будем менять никаких аспектов/граней (facets) в редакторе слотов.

Связывание слота с классом. Для того чтобы задействовать общий атрибут name (имя) в нашей онтологии, необходимо привязать его к классу. Например, каждому из экземпляров подклассов класса «Автор» дать имя. Вернемся к закладке классов и откроем на редактирование класс Автор (Author). Любой из атрибутов, который вы создаете или связываете с классом, будет отображаться в редакторе классов, справа от навигатора классов. Мы уже использовали редактор классов для смены имени нового класса, а также для

изменения роли класса Автор. Будем использовать редактор классов для просмотра и именования слотов. Для того чтобы связать слот *name* с классом:

1. Щелкните на закладке классов.
2. Выделите класс Автор в панели отображения иерархии классов (Class Hierarchy). Посмотрите на редактор классов (справа), в этой области отображается поле имя, роль класса, а также документация и ограничения (constraints). Под этими полями расположена, панель шаблонов слотов (Template slots), которая занимает всю оставшуюся нижнюю часть редактора классов. Эта область показывает слоты, связанные с классом. На текущий момент она пуста.

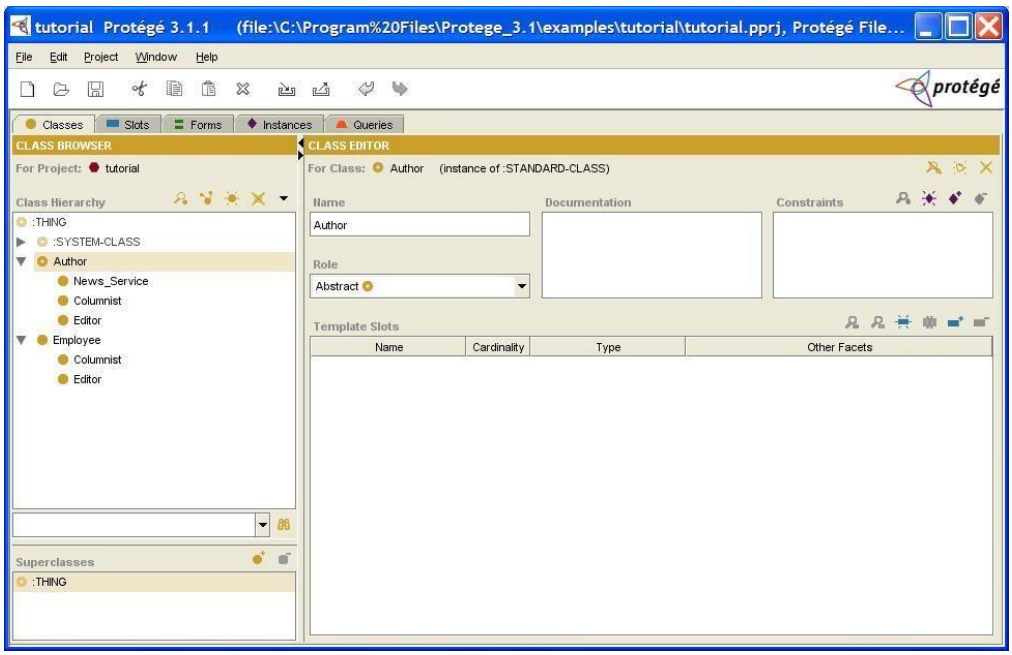

Рисунок 15.2 – Связывание слота с классом

3. Для добавления слотов к классу, нажмите кнопку **Add Slot** .

Кнопки управления слотами находятся в верхнем правом углу панели шаблонов слотов (Template slots).

Рисунок 15.3 – Панель инструментов шаблонов слотов

4. После того как вы нажмете кнопку, появится диалог выбора слота, в котором будет отображен список всех доступных слотов в вашем проекте (в алфавитном порядке, за исключением системных классов Protégé, которые будут видны в самом низу списка).

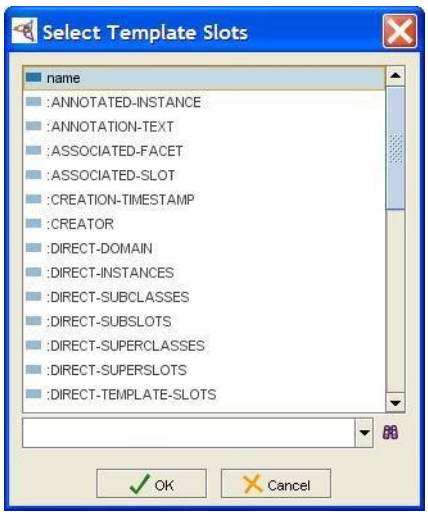

Рисунок 15.4 – Список доступных слотов

5. Выберите name и нажмите **OK**.

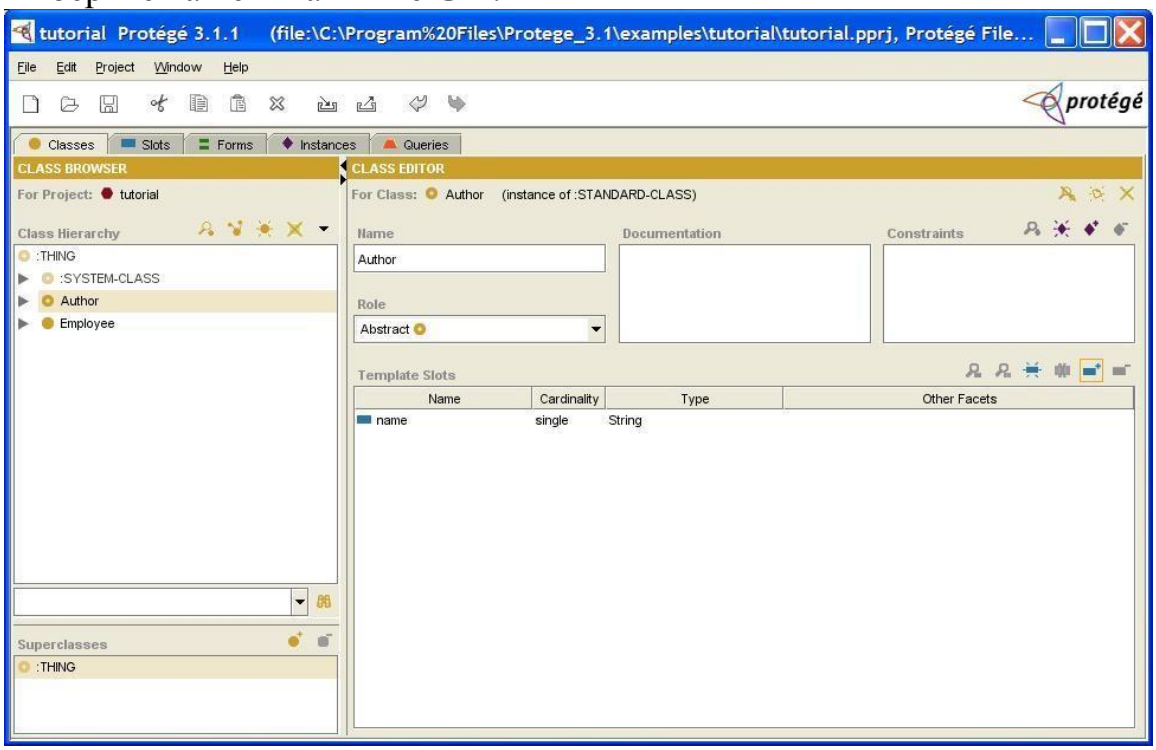

Рисунок 15.5 – Новый слот в списке шаблонов

Если вы посмотрите теперь на панель шаблонов слотов (Template slots), то увидите, что слот *name* был добавлен в список, и вместе с ним отображаются его свойства, в нашем случае это мощность (количество элементов типа) и сам тип (строка, String).

Создание слота из закладки классов. Переключение между закладками классов и слотов может показаться утомительным. И так как слоты есть свойства класса, их можно создавать проще, непосредственно с закладки классов.
Попытаемся создать слот для класса Employee, для этого:

1. Выберите класс Employee в панели иерархии классов.

2. Нажмите кнопку создать слот (Create Slot), в правом углу панели шаблонов слотов, будет вызвано окно добавления слота:

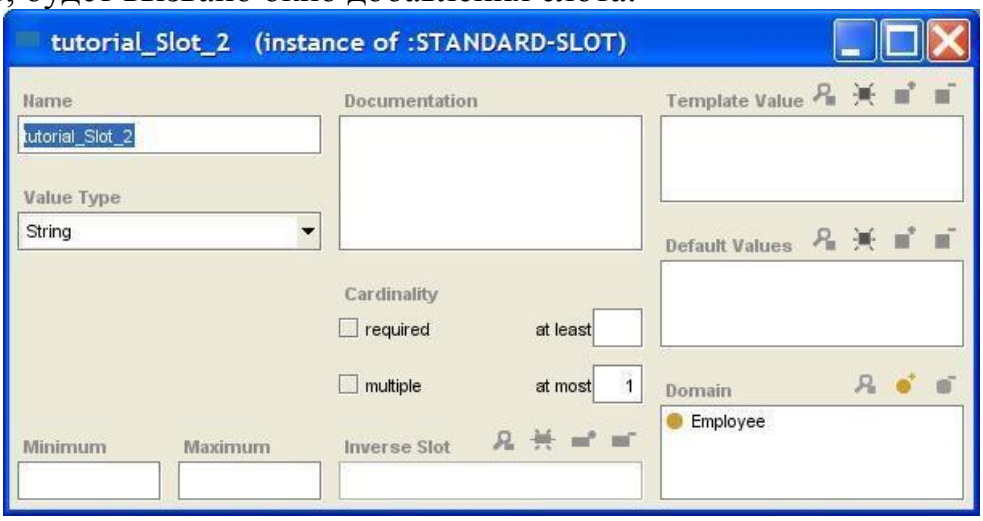

Рисунок 15.6 – Окно создания слота

3. Наберите salary (зарплата) в поле имя (Name), нажмите ввод.

4. Вернитесь в главное окно (при этом не обязательно закрывать окно редактирования, т. е. можно оставить его открытым и вернуться туда позже для редактирования свойств слота). Заметьте, что теперь новый слот показывается в панели шаблонов слотов, когда выбран класс Работник (Employee).

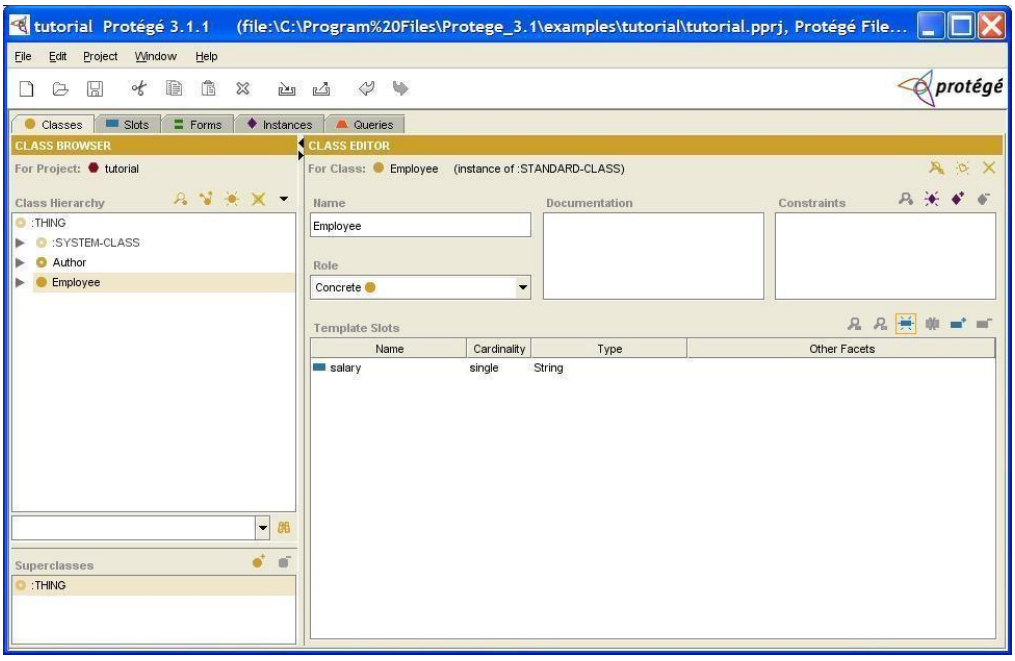

Рисунок 15.7 – Новый слот в списке шаблонов

Слоты и наследование. Мы не должны добавлять слот name (имя) к любому классу, где мы хотим его видеть. В смысле того, что любой подкласс класса автоматически наследует все слоты базового класса. К примеру, если вы выберите класс Служба новостей (News_Service), то увидите, что:

– Слот **name** (имя) уже связан с этим классом через механизм наследования.

– При этом иконка для слота отличается от той, которая использовалась для класса Автор (Author), а именно, для наследованных слотов используется иконка .

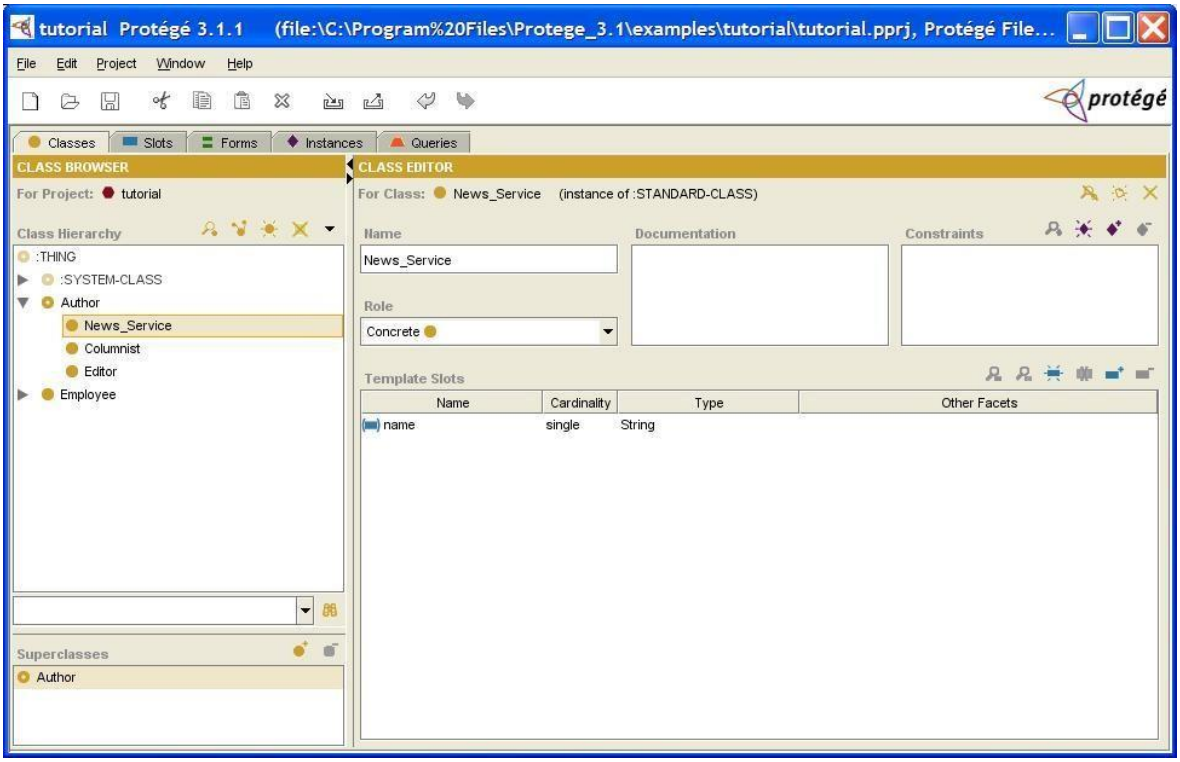

Рисунок 15.8 – Отображение наследованных слотов

Подклассы более чем с одним базовым классом наследуют слоты от всех базовых классов. К примеру, если Вы выберите класс Editor (редактор), то увидите, что он наследует слот name (имя) от Автора, и слот зарплата (salary) от Работника. Множественное наследование одна из основополагающих возможностей Protégé.

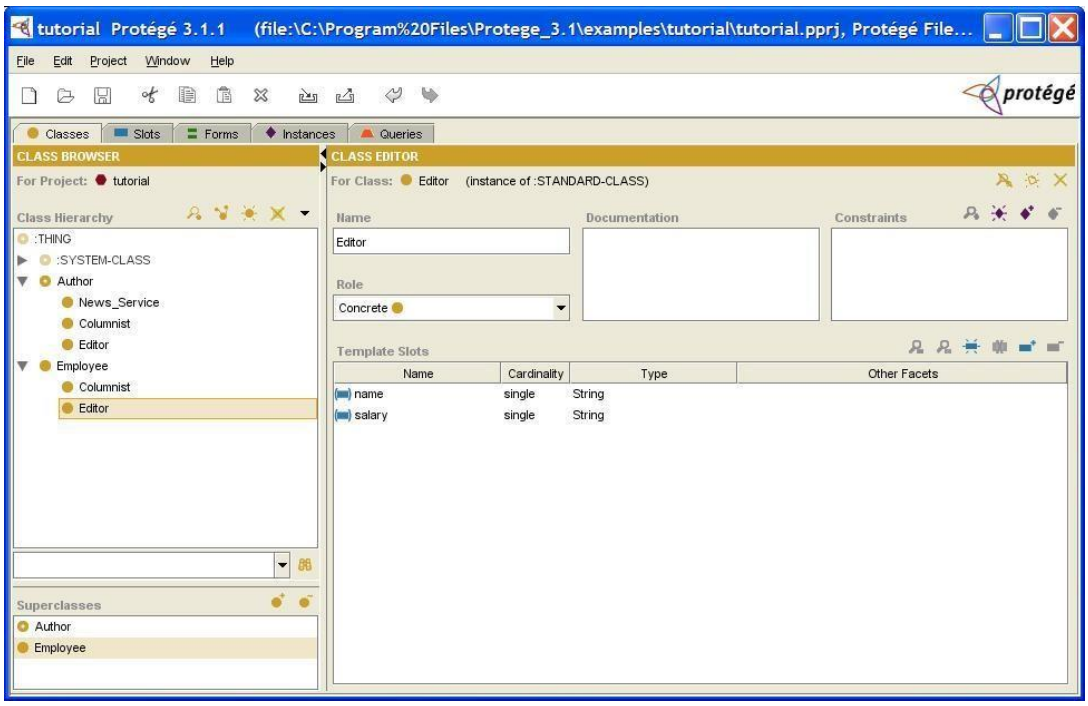

Рисунок 15.9 – Пример множественного наследования

Создание аспектов / граней (facets) слота. Слоты, которые были созданы на предыдущем шаге, очень простые. Однако, слоты сами по себе, тоже могут иметь свойства. К примеру, зарплата всегда является числом. Вы также можете использовать слоты для задания отношений между классами. Свойства слота, называемые аспектами / гранями (facets), могут быть созданы, как на закладке классов (используя диалог спецификации слота), так и на закладке слотов (используя редактор слота).

Создание аспектов слота «зарплата». Мы можем определить несколько аспектов для слота «зарплата», который был создан ранее.

1. Выберите класс «работник» (Employee) в панели иерархии классов.
2. Щелкните два раза на слоте «зарплата» в панели шаблонов слотов (Template slots), для того чтобы открыть форму выбора вида слота. Когда вы редактируете слот, Вы можете выбрать, будут ли изменения применяться к слоту и всем классам, связанным со слотом (вверх по иерархии до самого верхнего класса), или вы просто хотите, чтобы изменения коснулись текущего класса и всех его детей.

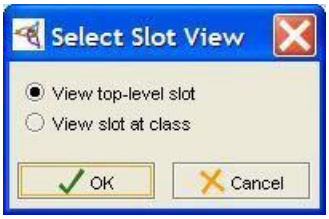

Рисунок 15.10 – Выбор формы отображения слота

3. В нашем случае, мы хотим просмотреть и отредактировать слот верхнего уровня. Потому убедитесь, что режим просмотра слотов верхнего

уровня (**View top-level slot**) выбран и нажмите **OK**. При этом изменение определения слота будет затрагивать всю онтологию.

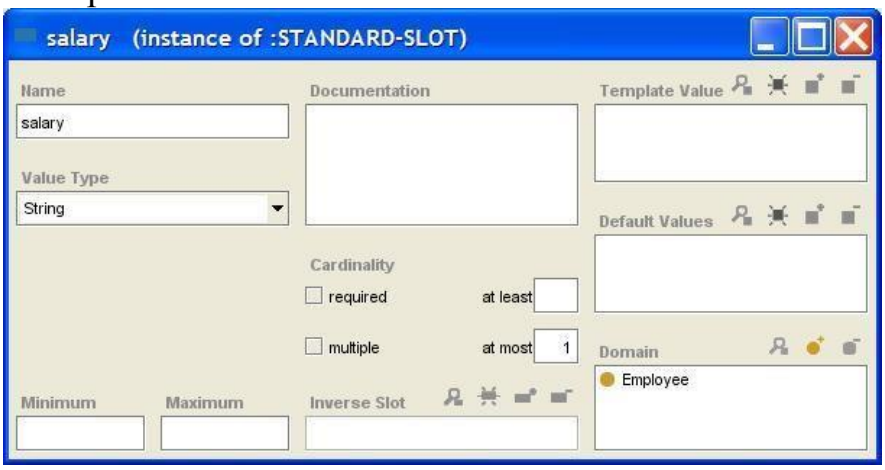

Рисунок 15.11 – Редактирование слота salary

4. В открывшейся форме редактирования слота, выберите *Float* из списка выбора типа значения (Value Type). Теперь при создании экземпляров, можно будет вводить для этого слота только правильные значения в формате с плавающей запятой.

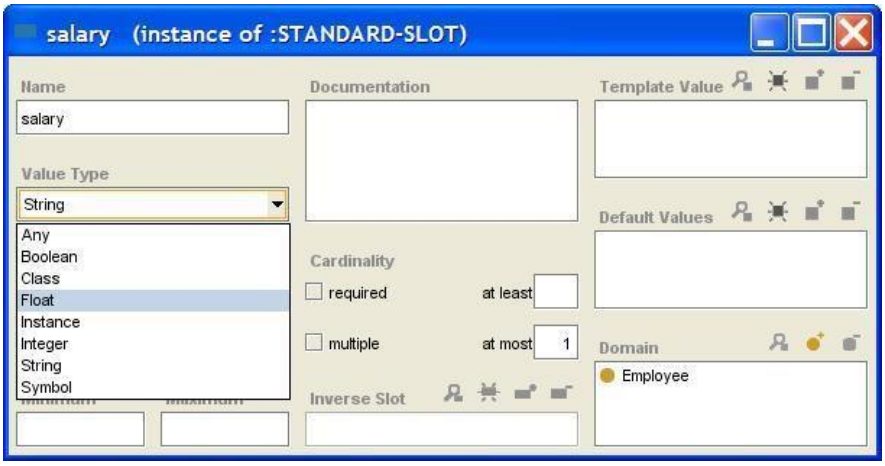

Рисунок 15.12 – Выбор типа значения

5. Введите 0 (нуль) в поле *Minimum* (минимальное значение). Таким образом, мы можем быть уверены, что теперь любое значение для поля «зарплата» будет не отрицательным.

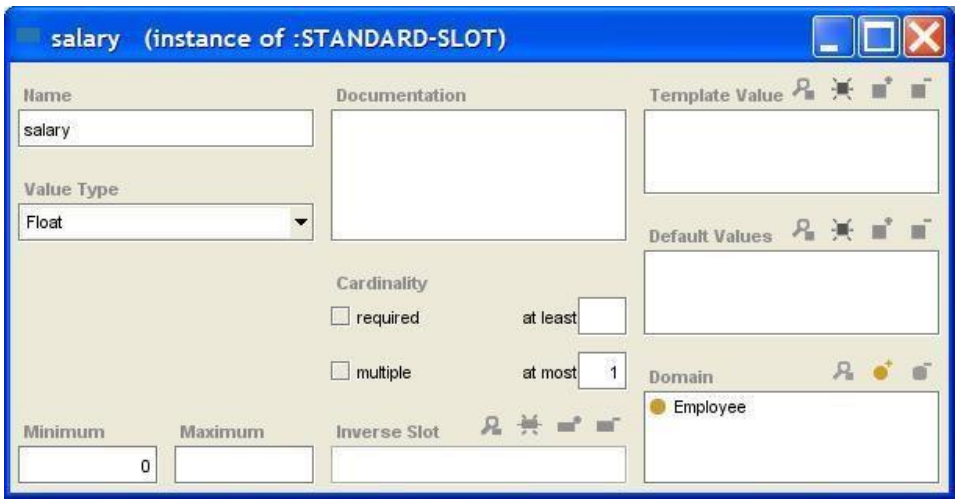

Рисунок 15.13 – Ввод минимального значения

6. Закройте, диалог редактирования слота, и Вы сможете увидеть, что описание слота в панели шаблонов слотов изменилось. В колонке тип теперь указан *Float*, а минимальное значение равно «0» появилось в колонке *Other facets* (другие аспекты).

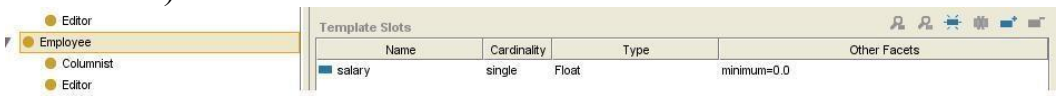

Рисунок 15.14 – Обновленное описание слота

Содержание отчета и его форма

В отчете к лабораторной работе должно быть указано название работы; перечислены этапы выполнения лабораторной работы и краткая характеристика работ на каждом этапе решения задачи, описанной в примере.

Вопросы для защиты работы

1. Для решения каких задач предназначена система Protégé?
2. Перечислите этапы создания проекта в системе Protégé.
3. Дайте определение понятия «класс» для системы Protégé.
4. Как создать класс в системе Protégé?
5. Дайте определение понятия «слот».
6. Как создать слот в системе Protégé?

ЛИТЕРАТУРА И ИСТОЧНИКИ

1. Джонс М. Т. Программирование искусственного интеллекта в приложениях. – М.: ДМК Пресс. – 2011. – 312 с. – Доступно: http://e.lanbook.com/books/element.php?pl1_cid=25&pl1_id=1244 – электроннобиблиотечная система «Лань».
2. Новиков Ф. А. Искусственный интеллект: представление знаний и методы поиска решений: Учеб. пособие. – СПб.: Изд-во Политехн. ун-та, 2010. – 240 с.
3. Путькина, Л. В. Интеллектуальные информационные системы / Л. В. Путькина, Т. Г. Пискунова. – СПб.: Изд-во СПбГУП, 2008.
4. Золотов, С. И. Интеллектуальные информационные системы: учеб. пособие / С. И. Золотов. – Воронеж: Научная книга, 2007. – 140 с.
5. Смолин Д. В. Введение в искусственный интеллект: конспект лекций. – М.: Физматлит. – 2007. – 264 с. – Доступно: http://e.lanbook.com/books/element.php?pl1_cid=25&pl1_id=2325 – электроннобиблиотечная система «Лань».
6. Муромцев Д. И. Онтологический инжиниринг знаний в системе Protégé. – СПб: СПбГУ ИТМО, 2007. – 62 с.
7. Башмаков А. И., Башмаков И. А. Интеллектуальные информационные технологии: учеб. пособие. – М.: Изд-во МГТУ им. Н. Э. Баумана, 2005. – 304 с.
8. Андрейчиков, А. В. Интеллектуальные информационные системы: учебник / А. В. Андрейчиков, О. Н. Андрейчикова. – М.: Финансы и статистика, 2004. – 424 с.
9. Братко И. Алгоритмы искусственного интеллекта на языке PROLOG, 3-е издание: Пер. с англ. – М.: Издательский дом «Вильямс», 2004. – 640 с.
10. http://www.intuit.ru – Национальный открытый университет «ИНТУИТ».
11. http://www.window.edu.ru – Единое окно доступа к образовательным ресурсам.
12. http://edu.consultant.ru – Образовательный сайт системы Консультант+

.

ПРИЛОЖЕНИЕ А

Клавиши главного меню

Вызов меню Files	Alt-F
Вызов редактора Edit	Alt-E
Вызов меню Compile	Alt-C
Вызов меню Options	Alt-O
Вызов меню Setup	Alt-S
Сохранение файла, находящегося в окне редактирования	F2
Загрузка файла	F3
Масштабирование текущего окна	F5
Циклическая смена окон	F6
Изменение размера окон	Shift-F10
Вызов DOS	Alt-D
Выход из Пролога	Alt-X

Командные клавиши компиляции

Компилирование программы из окна редактирования	Alt-C
Компилирование программы в память	F9
Компилирование программы в .OBJ	Shift-F9
Компилирование программы в .EXE	Ctrl-F9

Командные клавиши режима трассировки

Трассировать следующий шаг	F10
Вызов меню Trace	Alt-T

Командные клавиши режима запуска программы

Компилирование, компоновка и запуск исходного текста	Alt-R
Возврат в окно редактирования из данного режима	Ctrl-E
Повторный ввод цели	F8

Командные клавиши редактора

Вызов помощи	F1
Копирование из вспомогательного редактора (Xcopy)	F7
Вызов вспомогательного редактора (Xedit)	F8
Вызов главного меню	F10

Перемещения в окне редактирования

Курсор вправо на один символ	стрелка вправо	Ctrl-D
Курсор влево на один символ	стрелка влево	Ctrl-S
Курсор вверх на одну строку	стрелка вверх	Ctrl-E
Курсор вниз на одну строку	стрелка вниз	Ctrl-X
Скроллинг на одну строку вверх		Ctrl-W
Скроллинг на одну строку вниз		Ctrl-Z
Курсор вправо на одно слово	Ctrl-стрелка вправо	Ctrl-F
Курсор влево на одно слово	Ctrl-стрелка влево	Ctrl-A
Курсор в правый конец строки	End	Ctrl-Q D
Курсор в левый конец строки	Home	Ctrl-Q S
Курсор в верхнюю строку окна		Ctrl-Home
Курсор в нижнюю строку окна		Ctrl-End
Курсор на один экран вверх	PgUp	Ctrl-R
Курсор на один экран вниз	PgDn	Ctrl-C
Курсор в начало файла	Ctrl-PgUp	Ctrl-Q R
Курсор в конец файла	Ctrl-PgDn	Ctrl-Q C

Стирание текста

Стирание слова под курсором	Ctrl-T
Стирание строки под курсором	Ctrl-Y
Стирание от начала строки до курсора	Ctrl-Q T
Стирание от курсора до конца строки	Ctrl-Q Y

Манипулирование текстовыми блоками

Отметить начало блока	Ctrl-K B
Отменить конец блока	Ctrl-K K
Скопировать (вставить) выделенный блок	Ctrl-F5 Ctrl-K C
Перенести (вставить) выделенный блок	Ctrl-K V
Стереть выделенный блок	Ctrl-K Y
Восстановить стертый блок	Ctrl-F7 Ctrl-K U
Записать выделенный блок в дисковый файл	Ctrl-K W
Записать выделенный блок на принтер	Ctrl-K P

ПРИЛОЖЕНИЕ Б

Атрибуты цвета предиката

makewindow(WNo, ScrAttr, FrameAttr, Head, Row, Col, Height, Width)

Таблица Б.1 – Значения атрибутов цвета для монохромных мониторов

Цвет текста	Цвет фона	ScrAttr	Примечание
Черный	Черный	0	Пустой экран

Белый	Черный	7	Позитивное изображение
Черный	Белый	112	Негативное изображение

Таблица Б.2 – Значения атрибутов цвета для цветного графического адаптера

Цвет текста	Значение атрибута	Цвет фона	Значение атрибута
Черный	0	Черный	0
Синий	1	Синий	16
Зеленый	2	Зеленый	32
Голубой	3	Голубой	48
Красный	4	Красный	64
Фиолетовый	5	Фиолетовый	80
Коричневый	6	Коричневый	96
Белый	7	Белый	112
Серый	8		
Светло-синий	9		
Светло-зеленый	10		
Светло-голубой	11		
Светло-красный	12		
Светло-голубой	13		
Желтый	14		
Интенсивно-белый	15		

Существуют три необязательных атрибута, задание которых определяет вывод символов с подчеркиванием, изображение с высоким разрешением и вывод мерцающих символов. Вывод символов с подчеркиванием реализуется, если к значению аргумента ScrAttr добавить 1. Для получения изображения с высоким разрешением, нужно к основному значению атрибута экрана добавить 8. Символы будут мерцающими, если к значению атрибута экрана добавить 128.

Чтобы вычислить значение ScrAttr для различных комбинаций цветов, прежде всего, выберите необходимый цвет текста и цвет фона. Затем сложите соответствующие значения атрибутов. Если необходимо, чтобы символы мерцали, прибавьте к результирующему значению 128.
Вычисленное значение используется как второй аргумент в предикате ***makewindow***, т. е. ScrAttr.
Чтобы создать окно с белыми символами на черном фоне, сложите 7 (белый текст) и 0 (черный фон), результат будет 7.
Аргумент FrameAttr предиката ***makewindow*** есть целое число, значение которого определяет рамку окна. Если значение атрибута – 0, окно не имеет видимой границы. Другие значения определяют рамку окна с параметрами, указанными в таблице Б.3.
Часть значений атрибута рамки окна задает ее цвет. Это делается аналогично заданию значения атрибута экрана. Если задается мерцающая граница, то она всегда будет белой, с мерцающей тонкой линией (в середине границы), имеющей указанный цвет.
Аргумент Head задает метку окна. Например, меткой окна может быть «Главное Меню», «Окно Вывода». Строка, задаваемая в качестве атрибута Head, будет размещена в центре верхней линии рамки окна. Значение метки окна также может быть не определено, что соответствует отсутствию метки. В этом случае вводится аргумент, состоящий из двух последовательных знаков кавычки.

Таблица Б.3 – Значение атрибута FrameAttr

Значение атрибута	Вид рамки окна
0	Нет рамки
1	Синяя рамка
2	Зеленая рамка
3	Светло-синяя рамка
4	Красная рамка
5	Фиолетовая рамка
6	Желтая рамка
7	Белая рамка
8	Коричневая рамка
-1	Мерцающая белая рамка
-2	Мерцающая желтая рамка

-3	Мерцающая фиолетовая рамка
-4	Мерцающая красная рамка
-5	Мерцающая светло-синяя рамка
-6	Мерцающая светло-зеленая рамка
-7	Мерцающая синяя рамка
-8	Мерцающая серая рамка

ПРИЛОЖЕНИЕ В
Техническое и программное обеспечение

Для выполнения лабораторных работ необходим персональный компьютер со следующими характеристиками:

1) процессор Intel Pentium с тактовой частотой 800 МГц и выше,
2) оперативная память – не менее 64 Мбайт,
3) свободное дисковое пространство – не менее 500 Мбайт,
4) устройство для чтения компакт-дисков,
5) монитор типа Super VGA (число цветов – 256) с диагональю не менее 15".

Программное обеспечение – операционная система Microsoft Windows XP или выше; интегрированный пакет прикладных программ MS Office.

Лабораторные работы 1÷12 выполняются с помощью языка программирования Turbo Prolog / GNU Prolog / Visual Prolog (по выбору студента).

Лабораторные работы 13÷15 выполняются с помощью платформонезависимой среды Protégé v. 3.2 или выше.

ПРИЛОЖЕНИЕ Г
Указания по технике безопасности

При выполнении лабораторных работ студентами должны соблюдаться требования по технике безопасности, регламентированные в компьютерном классе: самостоятельно не производить ремонт персонального компьютера, установку и удаление программного обеспечения; в случае неисправности персонального компьютера сообщить об этом обслуживающему персоналу лаборатории (оператору, администратору); соблюдать правила техники безопасности при работе с электрооборудованием; не касаться электрических розеток металлическими предметами; рабочее место пользователя должно содержаться в чистоте; не разрешается в непосредственной близости от персонального компьютера принимать пищу, распивать напитки.

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Пятигорский институт (филиал) СКФУ

МЕТОДИЧЕСКИЕ УКАЗАНИЯ ДЛЯ СТУДЕНТОВ ПО ОРГАНИЗАЦИИ И ПРОВЕДЕНИЮ САМОСТОЯТЕЛЬНОЙ РАБОТЫ ПО ДИСЦИПЛИНЕ ЯЗЫКИ ПРЕДСТАВЛЕНИЯ ЗНАНИЙ

Направление подготовки	**09.04.02 Информационные системы и технологии**
Направленность (профиль)	**«Технологии работы с данными и знаниями, анализ информации»**
Квалификация выпускника	Магистр

Пятигорск, 2025

СОДЕРЖАНИЕ

1. ЦЕЛЬ И ЗАДАЧИ ОСВОЕНИЯ ДИСЦИПЛИНЫ

Цель данной дисциплины – дать систематический обзор современных моделей представления знаний, изучить и освоить принципы построения экспертных систем, рассмотреть перспективные направления развития систем искусственного интеллекта и принятия решений.

1.1. Цели освоения дисциплины.

В результате изучения данной дисциплины студент должен знать:

- модели представления знаний;
- принципы построения экспертных систем;
- современные системы искусственного интеллекта и принятия решений;

и уметь:

- разрабатывать программные реализации экспертных систем на ЭВМ;
- применять различные модели представления знаний при реализации экспертных систем на ЭВМ.

1.2. Задачи изложения и изучения дисциплины.

При изучении данной дисциплины в процессе чтения лекций преподаватель излагает студентам существующие модели представления знаний, принципы построения экспертных систем и перспективные направления развития систем искусственного интеллекта и принятия решений. В процессе самостоятельной работы студент на основе конспектов лекций, и рекомендованной литературы производит усвоение знаний. Контроль знаний осуществляется преподавателем по результатам контрольных работ. На основе полученных знаний и методических указаний по выполнению лабораторных работ студентом под руководством преподавателя проводится выполнение лабораторных работ.

2. КОМПЕТЕНЦИИ ОБУЧАЮЩЕГОСЯ, ФОРМИРУЕМЫЕ В РЕЗУЛЬТАТЕ ИЗУЧЕНИЯ ДИСЦИПЛИНЫ

Код, формулировка компетенции	Код, формулировка индикатора	Планируемые результаты обучения по дисциплине (модулю), характеризующие этапы формирования компетенций, индикаторов
ПК-1 способен осуществлять управление,	**ИД-1 ПК-1** Осуществляет управление,	

развитием баз данных, включая развертывание, сопровождение, оптимизацию функционирования баз данных, являющихся частью различных информационных систем	развитием баз данных. **ИД-2 ПК-1** Обеспечивает развертывание, сопровождение и оптимизацию баз данных **ИД-3 ПК-1** Осуществляет документальное сопровождение управления базами данных	
ПК-5 способен разработать новые инструментарии и методы управления проектами в области ИТ	**ИД-1 ПК-5** Разрабатывает новые инструментарий для эффективного управления проектами. **ИД-2 ПК-5** Организовывает эффективное взаимодействие персонала при работе над проектом. **ИД-3 ПК-5** Создает новые методы управления проектом в IT-сфере.	Разрабатывает инструментальные средства и методы управления проектами в области ИТ.

3. ТЕХНОЛОГИЧЕСКАЯ КАРТА САМОСТОЯТЕЛЬНОЙ РАБОТЫ СТУДЕНТА

Для студентов заочной формы обучения:

Коды реализуемых компетенций, индикатора(ов)	Вид деятельности студентов	Средства и технологии оценки	Объем часов, в том числе		
			СРС	Контактная работа с преподавателем	Всего
2 семестр					
ПК-1 (ИД $1_{ПК-1}$, ИД $2_{ПК-1}$, ИД 3	Подготовка к лекциям	Собеседование	0,18	0,02	0,2

ПК-1), ПК-5 (ИД 1ПК-5, ИД 2 ПК-5, ИД 3 ПК-5)					
ПК-1 (ИД 1ПК-1, ИД 2 ПК-1, ИД 3 ПК-1), ПК-5 (ИД 1ПК-5, ИД 2 ПК-5, ИД 3 ПК-5)	Самостоятельное изучение литературы	Собеседование	49,14	5,46	54,6
ПК-1 (ИД 1ПК-1, ИД 2 ПК-1, ИД 3 ПК-1), ПК-5 (ИД 1ПК-5, ИД 2 ПК-5, ИД 3 ПК-5)	Подготовка и выполнение практических работ	Отчет письменный	1,08	0,12	1,2
ПК-1 (ИД 1ПК-1, ИД 2 ПК-1, ИД 3 ПК-1), ПК-5 (ИД 1ПК-5, ИД 2 ПК-5, ИД 3 ПК-5)	Выполнение контрольной работы	Контрольная работа	9	1	10
Итого за 2 семестр			**59,4**	**6,6**	**66**
Итого			**59,4**	**6,6**	**66**

4. СОДЕРЖАНИЕ САМОСТОЯТЕЛЬНОЙ РАБОТЫ

4.1. **Подготовка** к лекциям. Самостоятельное изучение литературы

Тема самостоятельного изучения. Области применения систем искусственного интеллекта, их специфика.

1. Области применения систем искусственного интеллекта.
2. Знакомство с оболочкой экспертной системы.
3. Освоение механизмов внесения и изменения продукционных правил в оболочке.
4. Пример реализации экспертной системы в оболочке.

Вопросы для самоподготовки.

1. Что такое система искусственного интеллекта?
2. Какими чертами должна обладать система, чтобы называться интеллектуальной?
3. В каких областях используются системы искусственного интеллекта?

Задание для самостоятельной работы. Ознакомьтесь и проведите анализ материала лекционного занятия №1 и соответствующей информации в рекомендуемой литературе. Подготовьте конспект ответов на вопросы для самоподготовки. Выберете предметную область, в которой вы хорошо разбираетесь и можете выступать в качестве эксперта. Данная предметная область послужит вам базой для разработки собственной экспертной системы.

Тема самостоятельного изучения. Системы продукций как средство формализации ЭС.

1. Понятие системы продукций.
2. Примеры реализации систем продукций.
3. Виды логического вывода в ЭС.
4. Пример построение дерева логического вывода.

Вопросы для самоподготовки.

1. Что такое продукция?
2. По какому принципу работает алгоритм продукции Поста?
3. Какие виды логического вида существуют?
4. Что такое дерево вывода в ЭС?

Задание для самостоятельной работы.

Ознакомьтесь и проведите анализ материала лекционного занятия №2 и соответствующей информации в рекомендуемой литературе. Подготовьте конспект ответов на вопросы для самоподготовки. Постройте дерево вывода для выбранной вами предметной области, предварительно определив набор целевых и листьевых вершин.

Тема самостоятельного изучения. Архитектура ЭС. Основные этапы и технология разработки ЭС. Архитектура статической экспертной системы.

1. Отличие архитектуры динамической ЭС от статической.
2. Обсуждение дерева решений для создания ЭС. Выборочно представление результата выполнения задания для самостоятельной работы несколькими обучающимися.
3. Заслушивание и обсуждение устных докладов, связанных с темами занятий 1,2,3.

Вопросы для самоподготовки.

1. Понятие статической экспертной системы?
2. В чем заключается отличие динамической ЭС от статической?
3. Перечислите основные компоненты статической ЭС?
4. Каково назначение решателя и интерпретатора в ЭС?

Задание для самостоятельной работы.

Ознакомьтесь и проведите анализ материала лекционного занятия №3 и соответствующей информации в рекомендуемой литературе. Подготовьте конспект ответов на вопросы для самоподготовки.

Тема самостоятельного изучения. Исчисления: отличия от алгоритмов, наиболее изученные исчисления.

1. Отличия исчислений от алгоритмов.
2. Основы исчисления предикатов.
3. Основы исчисления высказываний.

4. Разбор примера по созданию набора правил вывода на основе дерева решений.

Вопросы для самоподготовки.

1. В чем заключается особенность использования математического аппарата исчислений?
2. Основные идеи исчислений предикатов?
3. Какие задачи рекомендуют решать с помощью аппарата исчисления высказываний?

Задание для самостоятельной работы.

Ознакомьтесь и проведите анализ материала лекционного занятия №4 и соответствующей информации в рекомендуемой литературе. Подготовьте конспект ответов на вопросы для самоподготовки. На основе разработанного вами ранее дерева решений напишите набор правил вывода для вашей экспертной системы. Постарайтесь, чтобы правила были сложносоставными, т.е. в условной части использовалось более одного условия, а так же присутствовал часть «иначе». Набор правил вывода можно представить в электронном или распечатанном виде.

Тема самостоятельного изучения.. Примеры реализации систем продукций. Неформальная структура систем продукций.

1. Неформальная структура системы продукции.
2. Примеры реализации систем продукции.
3. Реализации разработанных правил вывода в оболочке экспертной системы.

Вопросы для самоподготовки.

1. Чем вызван неформальный характер структуры системы продукций?
2. Перечислите примеры реализации систем продукций?
3. Можно ли использовать в правилах вывода сложные условия?

Задание для самостоятельной работы.

Ознакомьтесь и проведите анализ материала лекционного занятия №5 и соответствующей информации в рекомендуемой литературе. Подготовьте конспект ответов на вопросы для самоподготовки.

Тема самостоятельного изучения. Формальная модель систем продукций.

1. Основные компоненты формальной модели систем продукций.
2. Примеры использования формальной модели систем продукций.
3. Наполнение в оболочке экспертной системы оперативной базы данных фактами и проверка выполнимости разработанных ранее правил вывода.

Вопросы для самоподготовки.

1. Для чего предназначена формальная модель систем продукции?
2. Перечислите основные составляющие формальной модели систем продукций?
3. Что такое листьевая вершина в дереве решения?

Задание для самостоятельной работы.

Ознакомьтесь и проведите анализ материала лекционного занятия №6 и соответствующей информации в рекомендуемой литературе. Подготовьте конспект ответов на вопросы для самоподготовки. Подготовьте пример прямого и обратного логического вывода на базе дерева решений. В качестве дерева решений возьмите ранее разработанную вами иерархическую структуру.

Тема самостоятельного изучения. Логические и эвристические модели представления знаний.

1. Основные логические модели представления знаний.
2. Основные эвристические модели представления знаний.
3. Заслушивание и обсуждение устных докладов, связанных с темами занятий 4-7.

Вопросы для самоподготовки.

1. Перечислите представителей логических моделей представления знаний?
2. Перечислите представителей эвристических моделей представления знаний?
3. Что такое фрейм и как он устроен?

Задание для самостоятельной работы.

Ознакомьтесь и проведите анализ материала лекционного занятия №7 и соответствующей информации в рекомендуемой литературе. Подготовьте конспект ответов на вопросы для самоподготовки.

Тема самостоятельного изучения. Представление разработанной экспертной системы.

1. Представление обучающимися разработанных экспертных систем и их обсуждение всеми участниками занятия (преподаватель и студенты).
2. Подведение итогов практических занятий по всему учебному курсу.

Вопросы для самоподготовки.

1. Что вы почерпнули полезного из практической части учебного курса «представление знаний в ИС»?
2. Какие формы и модели представления знаний в информационных системах вы знаете?

3. В каких областях применяются интеллектуальные информационные системы, в частности, экспертные системы?

Задание для самостоятельной работы.

Ознакомьтесь и проведите анализ материала лекционного занятия №8 и соответствующей информации в рекомендуемой литературе. Подготовьте конспект ответов на вопросы для самоподготовки.

5. КРИТЕРИИ ОЦЕНИВАНИЯ КОМПЕТЕНЦИЙ

Оценка «отлично» выставляется студенту, если он продемонстрировал глубокие, исчерпывающие знания и творческие способности в понимании, изложении и использовании учебно-программного материала; логически последовательные, содержательные, полные, правильные и конкретные ответы на все поставленные вопросы и дополнительные вопросы преподавателя; свободное владение основной и дополнительной литературой, рекомендованной учебной программой.

Оценка «хорошо» выставляется студенту, если он продемонстрировал твердые и достаточно полные знания всего программного материала, правильное понимание сущности и взаимосвязи рассматриваемых процессов и явлений; последовательные, правильные, конкретные ответы на поставленные вопросы при свободном устранении замечаний по отдельным вопросам; достаточное владение литературой, рекомендованной учебной программой.

Оценка «удовлетворительно» выставляется студенту, если он продемонстрировал твердые знания и понимание основного программного материала; правильные, без грубых ошибок ответы на поставленные вопросы при устранении неточностей и несущественных ошибок в освещении отдельных положений при наводящих вопросах преподавателя; недостаточное владение литературой, рекомендованной учебной программой.

Оценка «неудовлетворительно» выставляется студенту, если он продемонстрировал неправильные ответы на основные вопросы, допущены грубые ошибки в ответах, непонимание сущности излагаемых вопросов; неуверенные и неточные ответы на дополнительные вопросы.

6. Организация контроля знаний студентов

Успешное освоение дисциплины основывается на систематической работе студентов. В процессе самостоятельной работы студенты в течение одного – двух дней прорабатывают материалы лекционных и лабораторных занятий по конспектам и рекомендованной основной литературе.

Конспекты дополняются материалами, полученными при проработке дополнительной литературы. При подготовке к письменной контрольной работе необходимо самостоятельно проработать задания из соответствующих глав рекомендуемой литературы.

Написание конспекта по темам самостоятельного изучения является одной из форм обучения студентов. Данная форма направлена на организацию и повышение уровня самостоятельной работы студентов.

Конспект, как форма обучения студентов - это краткий обзор максимального количества доступных публикаций по заданной теме, подготовка самого реферативного обзора и презентации по нему. При проведении обзора должна проводиться и исследовательская работа, но объем ее ограничен, так как

анализируется уже сделанные выводы и в связи с небольшим объемом данной формы работы. Преподавателю предоставляется сам конспект и презентация к нему. Сдача конспекта происходит в форме защиты-доклада с использованием подготовленной презентации.

При проверке конспектов дается анализ качества их ведения. Отмечаются допущенные ошибки, даются рекомендации по улучшению качества конспектирования изучаемого материала.

Тема и направленность контрольной работы представлена в методических рекомендациях. Контрольная работа составляется из типовых заданий, рассмотренных на занятиях. При выполнении контрольной работы студенты должны выполнить задания, показав при этом понимание теоретического материала и навыки решения практических задач. При выполнении контрольной работы студенты должны кроме основной и дополнительной рекомендованной литературы использовать и другие источники.

Формы контроля знаний студентов

Контроль и оценка знаний, умений и навыков студентов осуществляется на лабораторных занятиях, консультациях, при сдаче зачета. В ходе контроля знаний преподаватель оценивает понимание студентом содержания дисциплины «Вычислительные системы».

Контроль знаний студентов может осуществляться в следующих формах:

- текущий контроль знаний;
- итоговый контроль знаний.

Текущий контроль знаний студентов имеет целью: дать оценку работы каждого студента по усвоению им учебного материала, выявить недостатки в его подготовке и оказать практическую помощь в их устранении;

Основными формами текущего контроля знаний студентов являются:

- устный контрольный опрос;
- защита лабораторной работы;
- проверка конспектов лекций;
- проверка конспектов по темам, вынесенных на самостоятельное изучение.

Устный контрольный опрос студентов проводится на лекциях (и лабораторных занятиях). По его результатам преподаватель оценивает качество подготовки студента к занятию.

На лабораторных занятиях знания и практические навыки студентов оцениваются по 5-балльной системе.

Рекомендации по подготовке к экзамену

Экзамен, как итоговый контроль знаний студентов имеет целью проверить и оценить учебную работу студентов, уровень полученных знаний и практических навыков.

Экзамен проводится после защиты всех лабораторных работ в объеме учебной программы.

Рекомендации по работе с литературой и источниками

Изучение литературы и источников необходимо начинать с прочтения соответствующих глав учебных изданий, учебных пособий или литературы, рекомендованной в качестве основной или дополнительной по дисциплине «Вычислительные системы», которые прямо или косвенно относятся к изучаемой теме.

При изучении литературы и источников студенту рекомендуется вести краткий конспект. Однако не следует переписывать все содержание изучаемой темы, нужно выписывать лишь основные идеи и главные мысли. В отдельных случаях, когда встречаются важные определения, понятия, необходимый фактический материал и

примеры, статистическая информация, имеющие отношение к изучаемой теме, необходимо выписать их в виде цитат с полным указанием библиографических источников.

Конспектирование рекомендуемой литературы и источников необходимо вести с распределением собранных материалов по отдельным главам и параграфам согласно учебно-тематическому плану. Необходимо выписывать все выходные данные по используемой литературе и источникам.

Основой технологии интенсификации обучения являются учебно-иллюстрационные материалы (презентации) по дисциплине.

Работа с учебно-иллюстрационными материалами имеет следующие этапы.

1. Изучение теоретических основ учебного материала в аудитории: изложение преподавателем изучаемого материала студентам с объяснением;

2. Самостоятельная работа: индивидуальная работа студентов по конспекту;

3. Первое повторение - воспроизведение содержания заданной темы конспекта по памяти.

4. Устное проговаривание материала конспекта – необходимый этап внешне речевой деятельности при усвоении учебного материала.

5. Второе повторение – взаимоопрос и взаимопомощь студентов друг другу.

Применение учебно-иллюстрационных материалов позволяет обобщить сложный по содержанию материал, активизировать мыслительную деятельность студентов.

Необходимо помнить, что главное для студента в самостоятельной работе с рекомендуемой литературой и источниками - это формирование своего индивидуального стиля, который может стать основой в будущей профессиональной деятельности.

6. УЧЕБНО-МЕТОДИЧЕСКОЕ И ИНФОРМАЦИОННОЕ ОБЕСПЕЧЕНИЕ ДИСЦИПЛИНЫ

Рекомендуемая литература

Перечень основной литературы:

1. Советов Б.Я. Представление знаний в информационных системах: учебник. - 2-е изд. - М.: Академия, 2012-144с.

2. Громов Ю.Ю., Иванова О.Г., Серегин М.Ю., Дидрих В.Е., Мартемьянов Ю.Ф. Представление знаний в информационных системах: учебное пособие- Томбов: Изд-во ФГБОУ ВПО ТГТУ,2012-169с.

Перечень дополнительной литературы:

1. Фридман А.Я., Фридман О.В. Логические прикладные системы искусственного интеллекта. Учебное пособие. Апатиты: КФ ПетрГУ, 2004

2. Коробова И.Л., Артемов Г.В. Принятие решений в системах, основанных на знаниях: уч ебное пособие - Тамбов: Изд-во ФГБОУ ВПО ТГТУ,2012-81с.

3. Сосенская С.С. Представление знаний в информационной системе: учебная литература- М.: ТНТ, 2015-216с.

Перечень учебно-методического обеспечения самостоятельной работы обучающихся по дисциплине:

1. Методические указания по выполнению лабораторных работ по дисциплине «Представление знаний в информационных системах»;
2. Методические указания по выполнению контрольной работы по дисциплине «Представление знаний в информационных системах»;
3. Методические указания для студентов по организации самостоятельной работы по дисциплине «Представление знаний в информационных системах».

Перечень ресурсов информационно-телекоммуникационной сети

«Интернет», необходимых для освоения дисциплины:

1. http://www.intuit.ru – сайт дистанционного образования в области информационных технологий
2. http://www.iqlib.ru - интернет библиотека образовательных изданий, в которой собраны электронные учебники, справочные и учебные пособия;
3. http://www.biblioclub.ru - электронная библиотечная система «Университетская библиотека – online»: специализируется на учебных материалах для ВУЗов по научно-гуманитарной тематике, а так же содержит материалы по точным и естественным наукам.
4. http://www.iprbookshop.ru– электронно-библиотечная система IPRbooks.

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Пятигорский институт (филиал) СКФУ

МЕТОДИЧЕСКИЕ УКАЗАНИЯ ПО ВЫПОЛНЕНИЮ КОНТРОЛЬНОЙ РАБОТЫ ПО ДИСЦИПЛИНЕ ЯЗЫКИ ПРЕДСТАВЛЕНИЯ ЗНАНИЙ

Направление подготовки	**09.04.02 Информационные системы и технологии**
Направленность (профиль)	**«Технологии работы с данными и знаниями, анализ информации»**
Квалификация выпускника	Магистр

Пятигорск, 2024

СОДЕРЖАНИЕ

Введение

Методические указания содержат перечень вариантов заданий и требования к оформлению контрольных работ.

Теоретической основой подготовки специалиста являются знания в области вычислительной систем.

5. Цель, задачи и реализуемые компетенции

Методические указания составлены с учетом требований стандарта высшего образования по дисциплине: «Языки представления знаний». Целью освоения дисциплины «Языки представления знаний» является получение магистрантами знаний о принципах организации современных вычислительных систем, их структуре и функционировании.

6. Формулировка задания и его объем

Контрольная работа включат в себе вопросы по темам, которые нужно изучить самостоятельно и по ним подготовить отчет и презентации. Результаты выполнения контрольной работы предоставляются в электронном виде. Объем контрольной работы составляет 10-15 печатных листов формата А 4.

Варианты заданий выбираются из таблицы по последним двум цифрам зачетной книжки.

Послед цифра	Предпоследняя цифра									
	0	1	2	3	4	5	6	7	8	9
0	1,11	2,12	3,13	4,14	5,15	6,16	7,17	8,18	9,19	10,21
1	11,22	12,23	13,24	14,25	15,26	16,27	17,27	18,28	19,29	20,30
2	1,30	2,29	3,28	4,27	5,26	6,25	7,24	8,23	9,22	10,21
3	11,29	12,28	13,27	14,26	15,25	16,24	17,23	18,22	19, 21	20,10
4	19,31	18, 32	17, 33	16,34	15,35	14,18	13,19	12,20	11,21	10,22
5	9,35	8,34	7,33	6,32	5,31	4,30	3,29	2,28	1,27	30,26
6	30,11	29,10	28,9	27,8	26,7	25,6	24,5	23,4	22,3	21, 2
7	20,31	19, 30	18,29	17,28	16,27	15,26	14,25	13,24	12,23	11,22
8	10,29	9,31	8,33	7,35	6,25	5,23	4,21	3,19	2,17	1,15
9	1, 20	2,26	3,24	4,23	5,22	8,20	9,18	11,17	8,30	7, 31

7. Общие требования к написанию и оформлению работы

Контрольная работа выполняется и сдается в электронном виде на CD/CDRW носителе. На конверте необходимо указать название дисциплины, ФИО студента, факультет, номер группы, шифр зачетной книжки, № варианта задания, и список всех созданных в ходе выполнения задания файлов.

Приведенный в конце методических указаний список литературы может использоваться студентами при выполнении контрольной работы.

8. Варианты заданий для студентов заочной формы обучения

1. Типы знаний и данных
2. Признаки формально-логической модели
3. Элементы синтаксиса предикатов
4. Семантика предикатов первого порядка
5. Операция унификации
6. Принцип резолюции
7. Правило продукции
8. Прямая дедукция
9. Обратная дедукция
10. Представление фактов в Прологе
11. Представление правил в Прологе
12. Работа интерпретатора Пролога
13. Типы операций Пролога
14. Определение принадлежности элемента списку в Прологе
15. Выбор элемента списка в Прологе
16. Соединение двух списков в Прологе
17. Выбор наименьшего элемента списка в Прологе
18. Перевод одного списка в другой в Прологе
19. Сортировка списка в Прологе
20. Описание иерархической модели
21. Описание семантической модели
22. Характеристики Фреймовой модели
23. Признаки объектно-ориентированной модели
24. Структура реляционной модели
25. Типы индексирования
26. Виды связей реляционных таблиц
27. Элементы реляционной алгебры
28. Типы экспертных систем
29. Инженерия экспертных систем
30. Определение и операции с нечеткими множествами
31. Пример вывода на основе нечетких множеств
32. Применение нечетких множеств в экспертных системах

9. План-график выполнения задания

Дата получения задания	Дата предоставления выполненного задания

Установочная сессия.	Зимняя сессия за две недели до начала сессии.

10. Критерии оценивания работы

Оценка «отлично» выставляется студенту, если он продемонстрировал глубокие, исчерпывающие знания и творческие способности в понимании, изложении и использовании учебно-программного материала; логически последовательные, содержательные, полные, правильные и конкретные ответы на все поставленные вопросы и дополнительные вопросы преподавателя; свободное владение основной и дополнительной литературой, рекомендованной учебной программой.

Оценка «хорошо» выставляется студенту, если он продемонстрировал твердые и достаточно полные знания всего программного материала, правильное понимание сущности и взаимосвязи рассматриваемых процессов и явлений; последовательные, правильные, конкретные ответы на поставленные вопросы при свободном устранении замечаний по отдельным вопросам; достаточное владение литературой, рекомендованной учебной программой.

Оценка «удовлетворительно» выставляется студенту, если он продемонстрировал твердые знания и понимание основного программного материала; правильные, без грубых ошибок ответы на поставленные вопросы при устранении неточностей и несущественных ошибок в освещении отдельных положений при наводящих вопросах преподавателя; недостаточное владение литературой, рекомендованной учебной программой.

Оценка «неудовлетворительно» выставляется студенту, если он продемонстрировал неправильные ответы на основные вопросы, допущены грубые ошибки в ответах, непонимание сущности излагаемых вопросов; неуверенные и неточные ответы на дополнительные вопросы.

11. Порядок защиты работы

Защита контрольной работы проводится в виде научного дискурса с презентацией выполненных заданий, в соответствии с графиком защиты. После доклада студенту задаются вопросы, как преподавателем, так и студентами группы.

В процессе защиты своей работы студент делает доклад продолжительностью 7-10 минут. Доклад должен быть предварительно

подготовлен студентом. Лучшее впечатление производит доклад, в форме пересказа, без зачтения текста, которым следует пользоваться только для уточнения цифрового материала. Студент должен свободно ориентироваться в своей работе.

В выступлении необходимо корректно использовать демонстрационные материалы, которые усиливают доказательность выводов и облегчают восприятие доклада студента. Они оформляются в виде презентации в системе Power Point.

12. Учебно-методическое и информационное обеспечение дисциплины

8.1.1. Перечень основной литературы:

1. Советов Б.Я. Представление знаний в информационных системах: учебник. - 2-е изд. - М.: Академия, 2012-144с.
2. Громов Ю.Ю., Иванова О.Г., Серегин М.Ю., Дидрих В.Е., Мартемьянов Ю.Ф. Представление знаний в информационных системах: учебное пособие- Томбов: Изд-во ФГБОУ ВПО ТГТУ,2012-169с.

8.1.2. Перечень дополнительной литературы:

1. Фридман А.Я., Фридман О.В. Логические прикладные системы искусственного интеллекта. Учебное пособие. Апатиты: КФ ПетрГУ, 2004
2. Коробова И.Л., Артемов Г.В. Принятие решений в системах, основанных на знаниях: уч ебное пособие - Тамбов: Изд-во ФГБОУ ВПО ТГТУ,2012-81с.
3. Сосенская С.С. Представление знаний в информационной системе: учебная литература- М.: ТНТ, 2015-216с.

8.2.Перечень учебно-методического обеспечения самостоятельной работы обучающихся по дисциплине:

4. Методические указания по выполнению лабораторных работ по дисциплине «Представление знаний в информационных системах»;
5. Методические указания по выполнению контрольной работы по дисциплине «Представление знаний в информационных системах»;
6. Методические указания для студентов по организации самостоятельной работы по дисциплине «Представление знаний в информационных системах».

8.1.3. Перечень ресурсов информационно-телекоммуникационной сети «Интернет», необходимых для освоения дисциплины:

5. http://www.intuit.ru – сайт дистанционного образования в области информационных технологий
6. http://www.iqlib.ru - интернет библиотека образовательных изданий, в которой собраны электронные учебники, справочные и учебные пособия;
7. http://www.biblioclub.ru - электронная библиотечная система «Университетская библиотека – online»: специализируется на учебных материалах для ВУЗов по научно-гуманитарной тематике, а так же содержит материалы по точным и естественным наукам.
8. http://www.iprbookshop.ru– электронно-библиотечная система IPRbooks.