ФГОС

И. А. Калинин, Н. Н. Самылкина
П. В. Бочаров

ИНФОРМАТИКА

УГЛУБЛЕННЫЙ УРОВЕНЬ

Задачник-практикум
для 10 – 11 классов

Москва
БИНОМ. Лаборатория знаний

УДК 004.9
ББК 32.97
К17

Калинин И. А.

К17 Информатика. Углубленный уровень : задачник-практикум для 10–11 классов / И. А. Калинин, Н. Н. Самылкина, П. В. Бочаров. — М. : БИНОМ. Лаборатория знаний, 2015. — 248 с. : ил.

ISBN 978-5-9963-1722-6

Задачник-практикум входит в состав УМК по информатике углубленного уровня для старшей школы (10–11 классы) И. А. Калинина, Н. Н. Самылкиной и включает в себя практические работы по имитационному моделированию в среде AnyLogic, обработке статистических данных в электронных таблицах, обработке текста с использованием регулярных выражений, программированию в среде PascalABC.net, трехмерному моделированию в Google Sketchup, основам звукорежиссуры, защите данных в сетях и основам криптографии.

Практические работы оформлены в виде проектов к завершенной предметной линии учебников информатики углубленного уровня авторов. К практикуму предусмотрено электронное приложение.

УДК 004.9
ББК 32.97

Учебное издание

Калинин Илья Александрович
Самылкина Надежда Николаевна
Бочаров Павел Вячеславович

ИНФОРМАТИКА.
УГЛУБЛЕННЫЙ УРОВЕНЬ
Задачник-практикум для 10–11 классов

Редактор *Т. Г. Хохлова*. Художественный редактор *Н. А. Новак*
Технический редактор *Е. В. Денюкова*. Корректор *Е. Н. Клитина*
Компьютерная верстка: *Е. А. Голубова*

Подписано в печать 17.07.14. Формат 70×100/16.
Усл. печ. л. 20,15. Тираж 800 экз. Заказ

Издательство «БИНОМ. Лаборатория знаний»
125167, Москва, проезд Аэропорта, д. 3
Телефон: (499) 157-5272, e-mail: binom@Lbz.ru
http://www.Lbz.ru, http://e-umk.Lbz.ru, http://metodist.Lbz.ru

ISBN 978-5-9963-1722-6

Содержание

Введение

Практикум реализован в виде крупных проектов к отдельным главам учебников И. А. Калинина, Н. Н. Самылкиной по информатике углубленного уровня для старшей школы (10–11 классы). Это позволяет обеспечить модульную структуру всего изучаемого материала и реализовать возможность работы по индивидуальному учебному плану. Проектная работа предусмотрена ФГОС и должна быть интегрирована в учебный процесс. Это особая форма организации деятельности обучающихся в рамках одного или нескольких учебных предметов. Она позволяет сформировать навыки самостоятельного применения приобретенных знаний и способов действий при решении различных задач прикладного характера. Предусматривается обязательная защита результатов выполнения индивидуального проекта. Поэтому достаточно крупные вычислительные задачи и практические задания, результаты которых необходимо объяснить, мы объединили в отдельные проекты. К выполнению проекта следует приступать после изучения всей главы учебника, поскольку он опирается на весь содержательный материал главы. Результаты выполнения заданий, объединенных в проект, демонстрируются и объясняются обучающимися. Это может быть тематический семинар, конференция или индивидуальная защита только перед учителем. Формат определяется исходя из потребностей участников образовательного процесса.

Первый проект (к главе 3 учебника 10 класса) посвящен имитационному моделированию в среде AnyLogic. Здесь рассматриваются актуальные проблемные задачи по всем основным подходам в имитационном моделировании: агентное моделирование, дискретно-событийное моделирование и модели системной динамики. Содержание дополняет задачи, разобранные в учебнике.

Общая цель выполнения данного проекта — освоение современного инструмента имитационного моделирования и демонстрация возможностей метода. Конкретные цели и задачи ставятся совместно учителем и обучающимися в зависимости от того, какие модели планируется исследовать. Можно исследовать их все либо выбрать модели из предложенных в практикуме задач, а также реализовать разобранные задачи из учебника в дополнение к задачам из практикума.

Для практикума одна из ведущих мировых компаний-разработчиков средств имитационного моделирования, компания XJ Technologies (AnyLogic), предоставляет специально адаптированную к условиям школьного курса версию среды AnyLogic, которая позволяет создавать, демонстрировать и исследовать широкий спектр моделей из самых разных областей практической деятельности. Использование этой среды позволяет не только теоретически обсудить важность и возможности методов моделирования, но и продемонстрировать их важность и возможности для решения практических задач на актуальную тематику. Программу имитационного моделирования AnyLogic вместе с лицензионным соглашением на ее использование можно скачать по адресу: http://metodist.lbz.ru/authors/informatika/8/.

Второй проект «Алгоритмы и программы», относящийся к главе 4, отличается от остальных тем, что этот материал можно изучать параллельно с изучением тем этой главы: «Структуры данных» и «Типовые алгоритмы». При изучении программирования предполагается, что школьники уже владеют первичными навыками составления алгоритмов и программ, предусмотренными стандартом основного общего образования. Для учащихся классов углубленного уровня уже не актуален методический прием, опирающийся на графическое изображение алгоритмической конструкции (блок-схема) для перехода к анализу реального алгоритма. По теме предусматривается развитие уже известного материала через изучение теоретических основ создания и оценки алгоритмов; рассматривается проблема алгоритмической неразрешимости и дается ряд эффективных решений для задач, которые используются впоследствии при изучении информационных технологий. Содержание работы в первую очередь ориентировано на формирование представления об основных подходах к хранению и обработке данных при разработке программ, их вводу (как с клавиатуры, так и из файла) и выводу; демонстрируется работа с основными структурами данных и базовыми алгоритмами сортировки и поиска. В дальнейшем навыки программирования станут основой решения других проектных задач в области информационных технологий, как и было заявлено авторами. В этом проекте подробно рассмотрены задания из учебника; непосредственно из практикума они могут быть реализованы в виде программного кода. Для контроля сформированности необходимых навыков программирования приводятся задания для самостоятельной отработки.

Третий проект «Обработка статистических данных» (к главе 5) можно выполнить независимо от изучения довольно сложного теоретического материала учебника. Необходимая теория интегрирована в процесс обсчета статистических данных, позволяющих определить качество используемого тестового материала в ходе

проведения, например, государственной итоговой аттестации выпускников школ. Обучающиеся получат доказательства того, что в основу тестовых процедур заложена научно обоснованная методика, используемая во всех странах. Самый ценный результат выполнения данного проекта для обучающегося — это способность к самостоятельной информационно-познавательной деятельности, включая умение ориентироваться в различных источниках информации, критически оценивать и интерпретировать информацию, получаемую из различных источников. Именно этот результат можно считать общей целью данного проекта. Используемый инструмент для расчетов — табличный процессор.

Четвертый проект «Технология обработки текста» (к главе 6) может быть двухсоставным. Первая его часть может включать традиционную работу по подготовке текста по выбранной тематике со сложным форматированием. Объем и сложность подготовки оригинал-макета издания зависят от его типа (буклет, реферат, брошюра, плакат и пр.). Предполагается, что это межпредметная проектная работа. Задание из учебника уточняется учителем. Возможно, что такая работа окажется слишком простой для обучающихся, поэтому ее можно пропустить или предложить избирательно.

Вторая составляющая этого проекта более интересна и обязательна для выполнения. Общей целью работы является использование регулярных выражений для анализа текста. До тех пор, пока текст невелик, его можно проанализировать без применения техники, прочитав и обдумав. Но как только текст становится большим (особенно если он разбит на множество фрагментов), его анализ становится как минимум затруднительным — на чтение и обдумывание нужно немало времени.

Если же к большому объему текста добавляются сжатые сроки обработки, то чистый «ручной» анализ становится невозможным (или слишком дорогим) и появляется необходимость автоматизировать его обработку.

Анализ текста на естественном языке — многогранная задача. Различные методы и средства анализа разработаны для многих узких областей — это задачи поиска, перевода, выявления закономерностей и т. д. Мы продемонстрируем несколько специализированных методов автоматической обработки текста, которые в разных формах применяются очень широко.

Для выполнения заданий используется редактор Expresso, основанный на использовании механизма регулярных выражений среды .Net. Один из существенных плюсов такого рода средств — возможность отслеживать исполнение выражения, результаты его применения к тексту, создавать библиотеки готовых шаблонов.

Следующие три проекта относятся к 11 классу.

Проект «Графика и визуализация» к главе 1 состоит из двух самостоятельных частей. Использовать их можно как совместно, так и по отдельности.

В первой части рассматривается графика на плоскости. Приведены задания по темам, рассмотренным в учебнике: использование цветовых моделей, графические фильтры, реализация графических примитивов и геометрические преобразования фигур на плоскости. Для реализации используется программная среда PascalABC.NET, использованная ранее, и стандартные функции Microsoft.NET. Задания позволяют изучить базовые алгоритмы обработки графической информации и выявить основные проблемы, возникающие при этом.

Вторая часть проекта посвящена трехмерному моделированию. Для освоения основ работы с трехмерными моделями реальных объектов используется свободно распространяемая программа Google Sketchup версии 8 (возможно использование и более современных версий — без изменения содержания). Эта программа предназначена для так называемого «эскизного» моделирования: в ней нельзя создать действительно сложную модель или фотореалистичное изображение (тем более анимированное), зато удобно использовать для подготовки модели здания, предмета мебели, посуды и т. п. Именно эти модели рассматриваются в работе.

В проекте «Обработка звукового файла» к главе 2 используется третья версия продукта. Общей цель данного проекта — знакомство с основными понятиями музыкальной звукорежиссуры и основными приемами обработки звука на компьютере. Работа будет интересна многим старшеклассникам в основном из-за творческой составляющей и актуальности этой деятельности.

Проект «Защита данных в сетях» к главе 5 посвящен защите данных в сетях и основам криптографии. Задачи, рассматриваемые в рамках этой части практикума, предназначены, во-первых, для изучения фактического порядка функционирования сетевых компонентов и, во-вторых, для ознакомления с проблематикой защиты данных в сетях. Именно это можно считать общей целью выполнения этого проекта.

Приведенные задания требуют подготовки и использования виртуальной машины, на которой и отрабатываются практические действия по защите персональной машины. Рассматриваются средства обновления системы, установки и настройки межсетевого экрана, средства работы с сертификатами и шифрованием. Все практические работы связаны с непосредственной демонстрацией того, что бывает при отсутствии средств защиты.

Желаем успехов!

10 класс

- Проект «Моделирование» к главе 3 учебника
- Проект «Алгоритмы и программы» к главе 4 учебника
- Проект «Обработка статистических данных» к главе 5 учебника
- Проект «Технология обработки текста» к главе 6 учебника

Проект

«Моделирование»

Имитационное моделирование

Как вам уже известно из учебника, имитационное моделирование — мощнейший метод исследования самых разных проблем, в первую очередь — проблем чисто практических. Чтобы освоить его основные виды и познакомиться с его возможностями, мы построим несколько моделей с использованием среды AnyLogic. Российская компания The AnyLogic Company, производитель этой среды, любезно предоставила школьникам специальную версию, которой более чем достаточно для выполнения практических работ. Перед началом работы кратко опишем интерфейс среды.

Основное рабочее окно среды выглядит примерно так (рис. 1).

С левой стороны вверху — дерево, в котором показаны открытые сейчас модели (на иллюстрации — моделей две, открыта Epidemic), а в модели — объекты, из которых она состоит. В открытой сейчас модели мы видим основное пространство — Main, параметры и стартовую страницу модели Simulation, на которой можно будет размещать настройки перед прогонами.

Среда позволяет моделировать самые разные процессы через взаимодействие *объектов* — программных моделей предметов, явлений и связей. Каждая такая модель-объект включает в себя данные-описания (свойства) и программы-действия.

Наша цель при создании модели реального процесса — собрать комплекс объектов, который будет вести себя так же, как моделируемая нами действительность, конечно, в той мере, в какой наша модель вообще может это делать (напомним, что модель — *упрощенное* подобие действительности). Собрать комплекс мы можем, разместив объекты в рабочей зоне и связав их явно, указав, например, траекторию перехода объектов, записав действие-программу, вычислить значение параметра и т. д. Практически все наши описания подготовки моделей — это описание того, из каких объектов мы собираем модель, как их соединяем и какие значения свойств и действий придаем. Перед запуском модели среда переведет наш комплекс объектов в исполняемый код и (если это удастся) запустит.

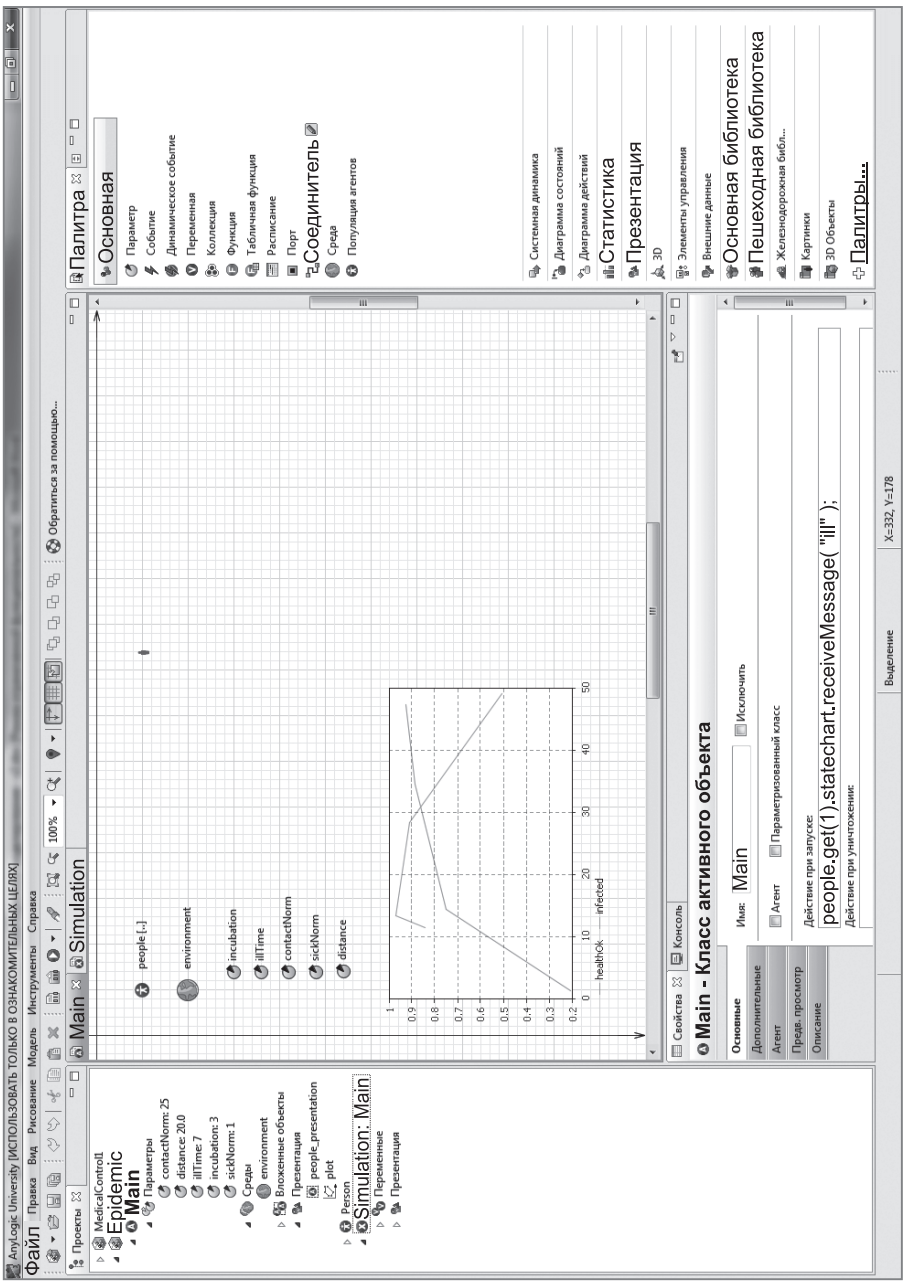

Рис. 1. Основное рабочее окно среды Anylogic

Основу языка программирования, на котором мы фактически описываем объекты в среде, составляет язык программирования Java. Его синтаксис, как и описание объектов, описан в разделе помощи к среде, которой мы очень рекомендуем пользоваться во время работы. Из языков, на которые мы опираемся при изучении информатики, ближе всех к нему язык С. Несмотря на это, среда построена так, что писать как таковые программы нам фактически не придется, поскольку там пошагово описано и создание моделей-примеров.

Возвращаясь к изображению основного рабочего окна системы, отметим, что объекты размещаются на общем поле — в графическом редакторе. Объекты можно добавлять, перетаскивая их из палитр наборов объектов в правой части (сейчас открыта палитра **Основная**). Свойства и методы объектов можно описывать, задавая им значения в нижней центральной части рабочего окна.

Начнем с наиболее иллюстративного вида имитационных моделей — агентных моделей.

Задача 1. *Изучение движения учащихся через турникеты с помощью агентной модели*

В школе № 0 планируется повысить уровень обеспечения безопасности. В первую очередь специалисты рекомендовали ограничить проход в школу посторонних. Для этого на входе установят два турникета и будут проверять всех входящих — требовать предъявления специальной карты.

У директора школы есть подозрение, что применение такого подхода приведет к существенным затруднениям перед началом занятий.

Как подтвердить или проверить эти предположения?

Решение

Создадим модель, изображающую первый этаж школы с турникетами и агентами-пешеходами, единственная задача которых — пройти через турникет.

При этом мы учтем, что проход через турникет (а точнее, проверка) занимает некоторое время, и время от времени будет попадаться человек без документов (например, без карточки), которого нужно возвращать.

Время прохода может меняться (человек проверяет документы медленнее, чем автомат — карточки), вероятность появления «неправильного» документа — тоже.

Чтобы модель выглядела реалистично, найдем на сайте авторской мастерской (http://metodist.lbz.ru/authors/informatika/8/) в разделе «Электронное приложение» типовой проект школы и загрузим план первого этажа.

Далее запустим среду и командой **Файл/Создать** создадим пустую модель (рис. 2):

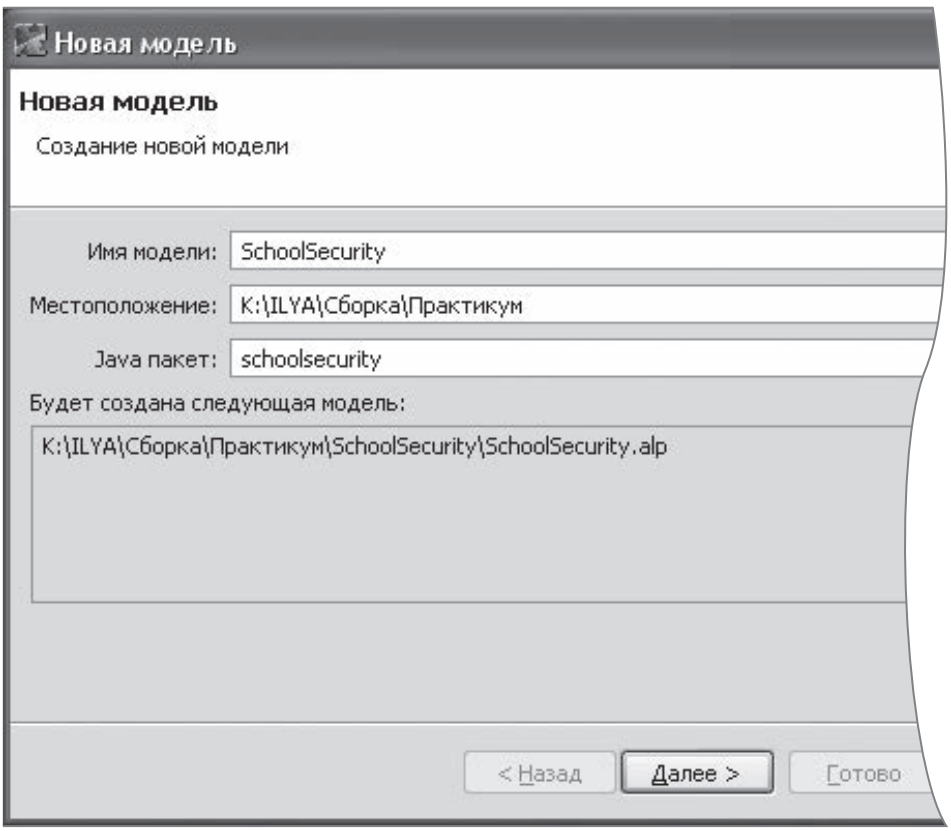

Рис. 2. Создание пустой модели

На втором шаге укажем, что модель создаем «с нуля», не используя заготовки.

План первого этажа очистим от лишних деталей и разместим на рабочем листе: из набора компонентов **Презентация** перетащим на рабочее поле объект **Изображение** и добавим в его параметрах картинку плана этажа (рис. 3).

Вставленную картинку «растянем».

Теперь прямо поверх плана отметим стены (поскольку на картинке система не может автоматически распознать их). Для этого построим ломаные вдоль стен. Обратите, пожалуйста, внимание:

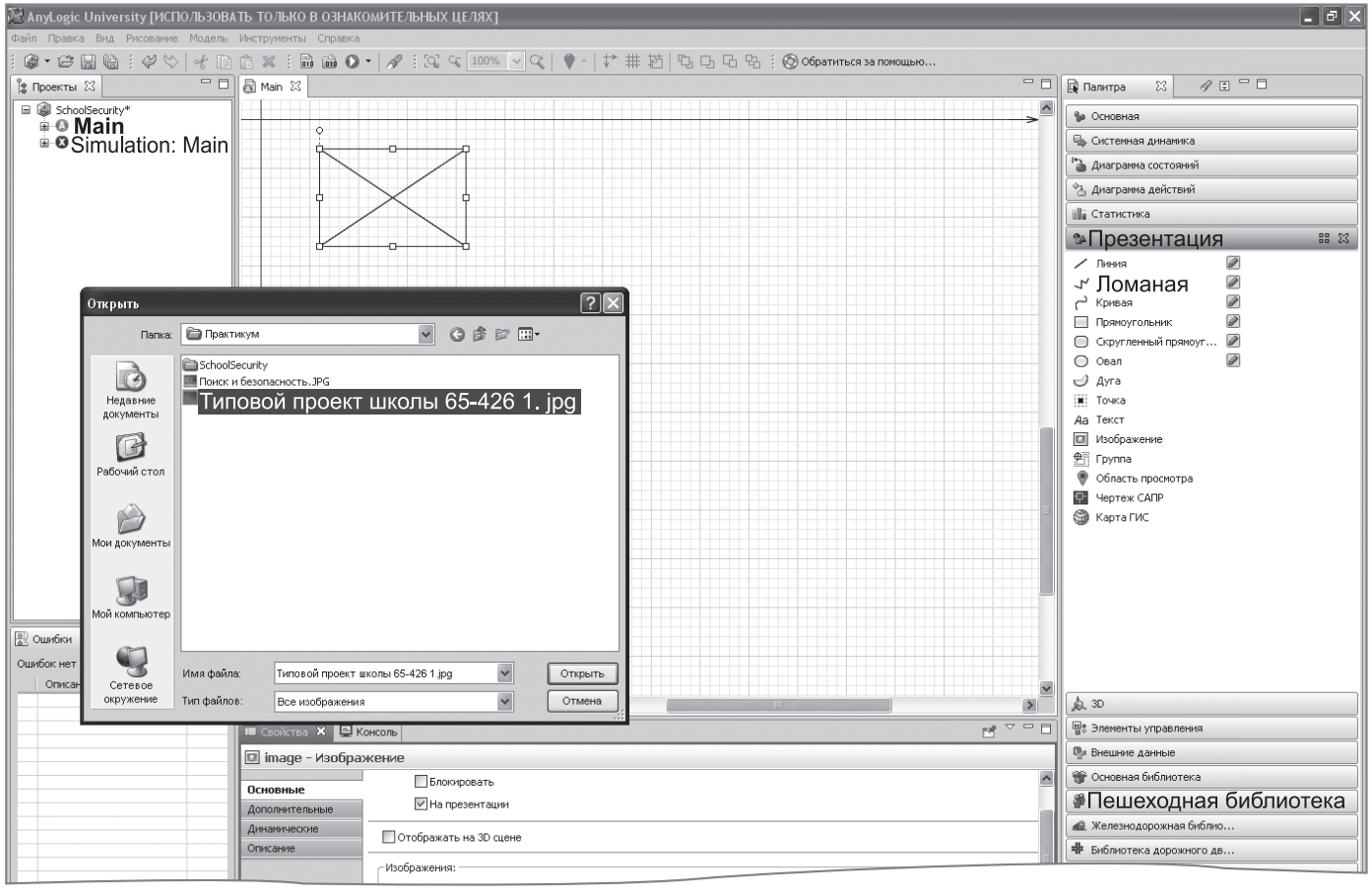

Рис. 3. Создание плана первого этажа

чтобы начать рисование ломаной, нужно дважды щелкнуть по значку **Карандаш** справа от объекта **Ломаная** в наборе компонентов **Презентация**.

Чтобы сделать нашу ломаную «стенами», нужно объединить все элементы в группу. Для этого выберите их, щелкнув на них мышью с нажатой клавишей **Shift**, нажмите правую клавишу мыши и выберите в меню **Группировка → Создать группу**. Появившуюся группу назовем **Walls**.

Создадим стартовую линию – «источник» пешеходов, расположив ее перед крыльцом, — с точки зрения модели, нам не важно, откуда люди приходят к школе. Чтобы ее нарисовать, перетащим объект-линию, подгоним размеры и зададим линии имя: enter.

Внутри школы отметим пару линий-выходов. На самом деле это, конечно, не выходы, а просто подъемы на 2-й этаж, но, опять же с точки зрения модели, важно только то, что люди уходят с этажа. Схема, безусловно, очень упрощена — в обычной ситуации людям надо еще как минимум раздеться.

Линии назовем exit1 и exit2, примерно так, как показано на иллюстрации (рис. 4).

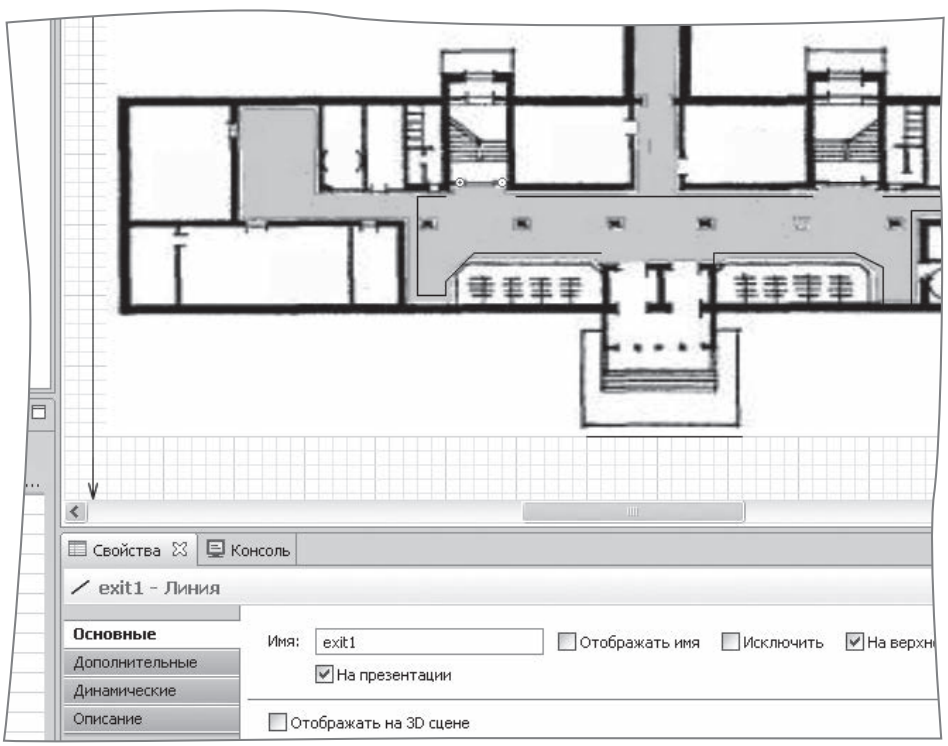

Рис. 4. Построение стен, линий входа и выходов

Пока что на нашей модели ничего не происходит, поскольку мы еще не задали движения пешеходов.

Построим общую логическую схему движения из готовых блоков. Блоки нужно перетащить по одному из раздела **Пешеходная библиотека** и расположить примерно так, как показано на рис. 5.

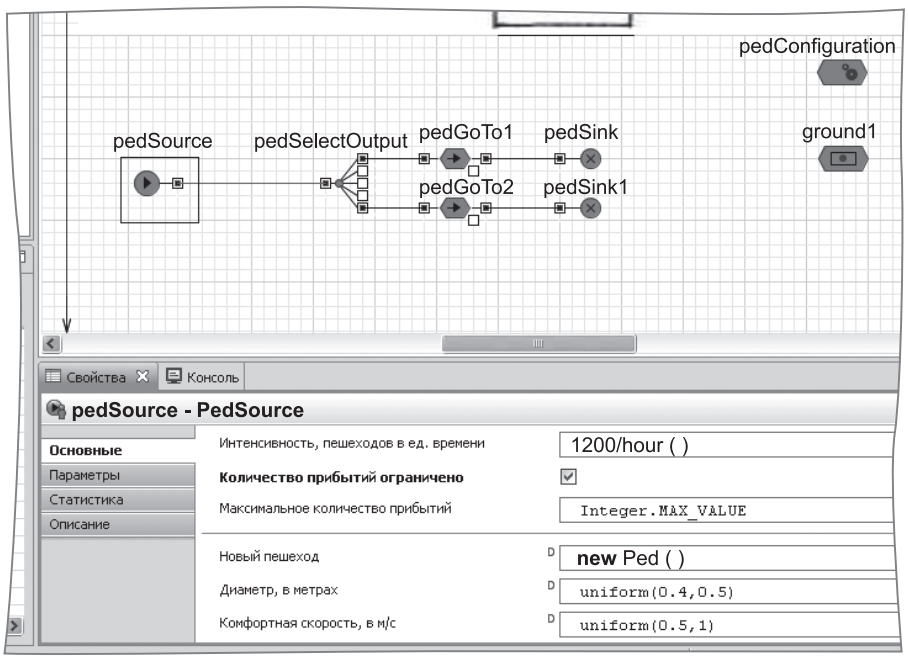

Рис. 5. Построение общей логической схемы движения

Блоки будут такие:

pedSource — источник пешеходов;
pedSelectOutput — выбор выхода;
pedGoTo — движение пешехода;
pedSink — выход, изъятие пешехода из модели.

Блоков pedGoTo будет два, поскольку у нас два выхода. Сейчас для нас неважно, что происходит за пределами турникетов, поэтому мы не учитываем, что люди могут от каждого входа пойти к любому выходу.

Блоки нужно связать между собой. Для этого щелкнем два раза на прямоугольнике — обозначении порта (точки приема соединителей), протянем линию-соединитель до соответствующего порта-приемника и там щелкнем второй раз. На рисунке 5 мы вытянули соединитель от pedSource до pedSelectOutput.

Кроме этого нам понадобятся на модели еще два объекта без соединений:

pedConfiguration — конфигурация;
pedGround — объект-этаж.

В параметрах этажа нужно указать:

- имя этажа. Для этого в поле **Имя** напишем «ground1»;
- объект-стены. В разделе свойств в поле **Стены** впишем название группы линий со стенами: Walls.

Зададим параметры источника пешеходов (pedSource):

- Ограничим количество пешеходов. Для этого поставим галочку **Количество прибытий ограничено** и укажем в поле **Максимальное количество прибытий** значение 600.
- Укажем объект — **этаж**. В поле **Этаж** напишем «ground1».
- Место появления. В соответствующем поле напишем имя линии — enter.
- Укажем частоту появления новых пешеходов, исходя из следующих параметров: в школе учится 600 человек и попасть внутрь они должны за полчаса. Таким образом, параметр **Интенсивность** должен иметь значение 1200 человек в час.

Укажем параметры «разветвления» pedSelectOutput — коэффициенты предпочтения, т. е. вероятность выбора каждого направления. В нашем случае они равны — по 0,5 первому и пятому исходам (остальные мы не использовали).

В каждом элементе движения pedGoTo нужно указать цель, т. е. линию выхода exit1 и exit2.

Запустим модель и, не меняя скорости, посмотрим, сколько времени займет проход пешеходов. Стоит заметить, что время в модели — «виртуальное». Фактически одна единица времени — это одна минута, т. е. нам не нужно наблюдать за прогоном полчаса, достаточно 30 с.

Во время прогона мы выясним несколько мелких огрехов.

- Пешеходы проходят через стенку между выходами и через колонны, поскольку мы их не учли в границах. Дорисуем недостающие линии и добавим их в группу: нажмем клавишу **Shift**, щелкнем на каждой новой линии, в контекстном меню вызовем команду **Группировка → Добавить в группу**. После этого мы выберем группу Walls для добавления — на рисунке появится изображение мишени с подписью, на котором надо будет щелкнуть.
- В некоторых случаях пешеход «не видит» свободного прохода к своему выходу, и это вызывает ошибку. Сдвинем линии выходов чуть внутрь.

В нашей модели скорость движения пешеходов различна, поэтому для уверенности сделаем несколько прогонов (рис. 6).

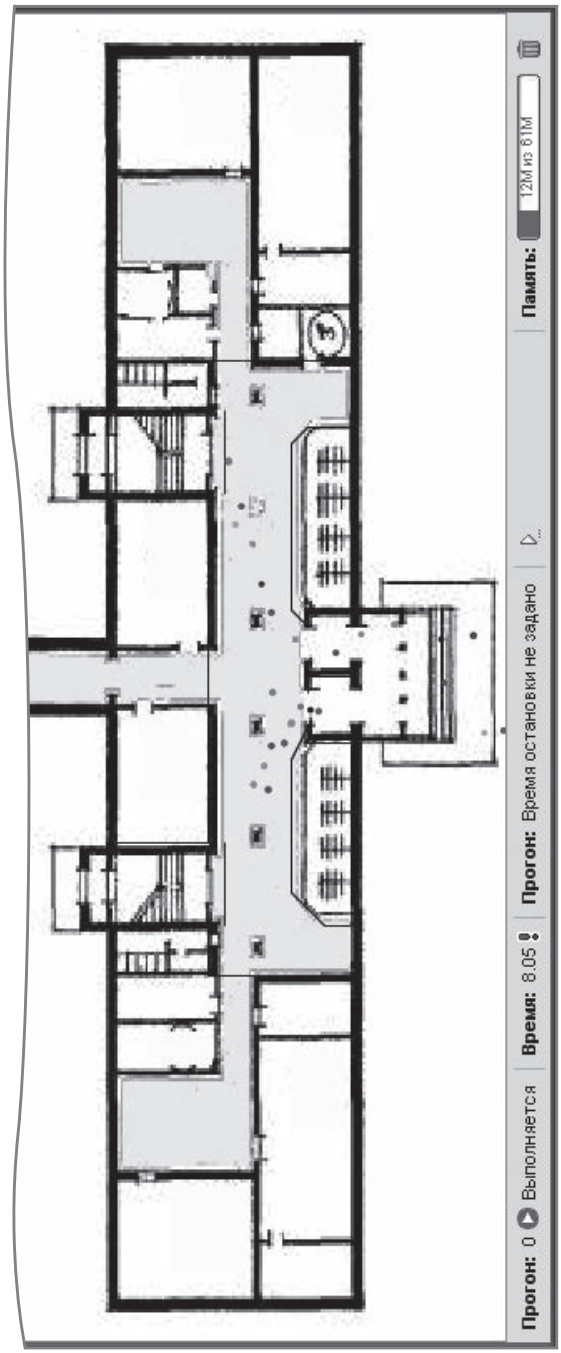

Рис. 6. Прогон модели

Результат будет вполне предсказуемым — никаких препятствий нет, 600 человек попадают внутрь примерно за 30 мин (т. е. по дороге нет затруднений — время зависит только от частоты появления и скорости движения).

Дополним нашу модель — поставим два турникета (рис. 7). Нарисуем линии gate1 и gate2, обозначим двумя прямоугольниками зоны ожидания обработки и предусмотрим линию denied1 — как цели для движения в случае отказа (например, забыта или отказала карта).

Время ожидания изменяется от 0,5 до 1 с.

Для обработки ожидания нужно добавить два объекта—зоны pedArea (см. рис. 7). Линия gate1 на рисунке выделена.

В схеме разобьем путь на три этапа: до турникета, ожидание пропуска и проход либо на этаж, либо на выход. Все объекты нужно связать, заполнив параметры: укажем соответствующий турникет как цель движения в каждом объекте pedGoTo, укажем время ожидания проверки — от 0,5 до 1 с в параметре «время задержки» и соответствующий выход exit1 или exit2 для выхода с этажа.

В новых ветвлениях мы уже укажем неравное разделение: будем считать, что 95% посетителей имеют действительные карты (первый выход из каждого нового ветвления имеет коэффициент

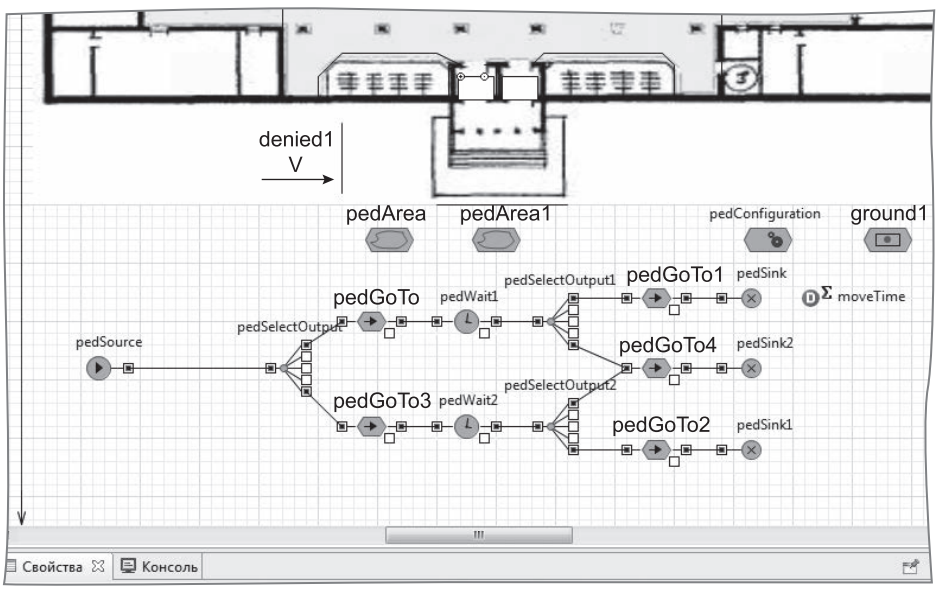

Рис. 7. Добавление зон pedArea

выбора 0,95), а 5% (0,05) – нет. Во всех случаях неудачной попытки (неверной карты) мы отправим людей на линию denied1 (см. рис. 7).

Снова запустим модель и обнаружим, что время выросло, и не просто выросло, оно выросло катастрофически. За 32 мин в школу прошел только 241 ученик и 4 вышли, все остальные толпятся на входе, причем 4 человека просто пытаются выйти с неработающей картой. Приходится отметить, что директор был абсолютно прав.

Заметим, что многое при проверке такой модели будет зависеть от того, как именно работает алгоритм действий агента, в нашем случае пешехода. В предыдущих версиях этой библиотеки пешеходы иногда проходили через турникет по два человека и меньше толкались на входе. В результате толпа не возникала, и за 32 мин все успевали пройти внутрь. Точное же соблюдение правил прохода привело к результатам, показанным на рис. 8.

Усовершенствуем нашу модель, чтобы больше не оценивать время «на глаз». Для этого воспользуемся средствами сбора статистики и возможностью модифицировать класс пешехода.

Мы видели, что во время работы модели некоторые пешеходы могут «заблудиться» и застрять на этаже. Поскольку на реальный

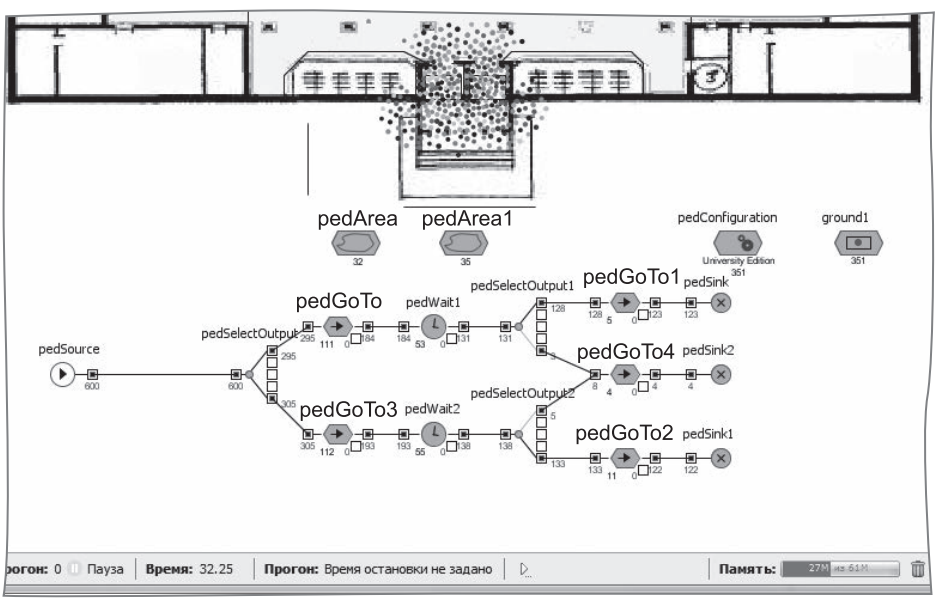

Рис. 8. Результат точного соблюдения правил прохода

результат они повлиять не могут, их можно отбросить. Параметр, который нас на самом деле интересует, — это время, которое потребуется пешеходу, чтобы пройти через этаж. Изначально время входа и выхода у пешехода не фиксируется, поэтому создадим нового пешехода, отмечающего время.

Для этого:

- на названии модели нажмем правую кнопку, в меню выберем пункт **Создать → Новый Java класс**;
- в появившемся окне укажем имя класса schoolPed и базовый класс Ped (рис. 9);

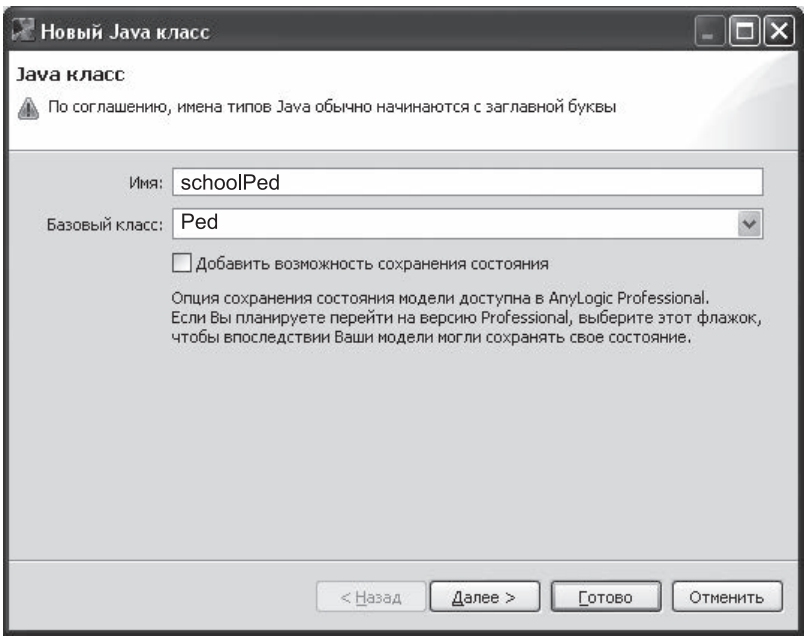

Рис. 9. Создание классов schoolPed и Ped

- в следующем окне (рис. 10) добавим параметр startMove — там будет храниться время входа. Тип этого параметра — double, поскольку именно так обрабатывается модельное время;
- галочки **Создать конструктор** и **Создать метод toString()** уберем — мы будем записывать время сами.

Теперь укажем в объекте-источнике:

- класс пешехода — schoolPed (вместо просто Ped, который был по умолчанию). Без этого мы не сумеем обратиться к новому параметру (в базовом классе его нет);
- в пункте **Новый пешеход** вместо `new Ped()` укажем `new schoolPed()`;

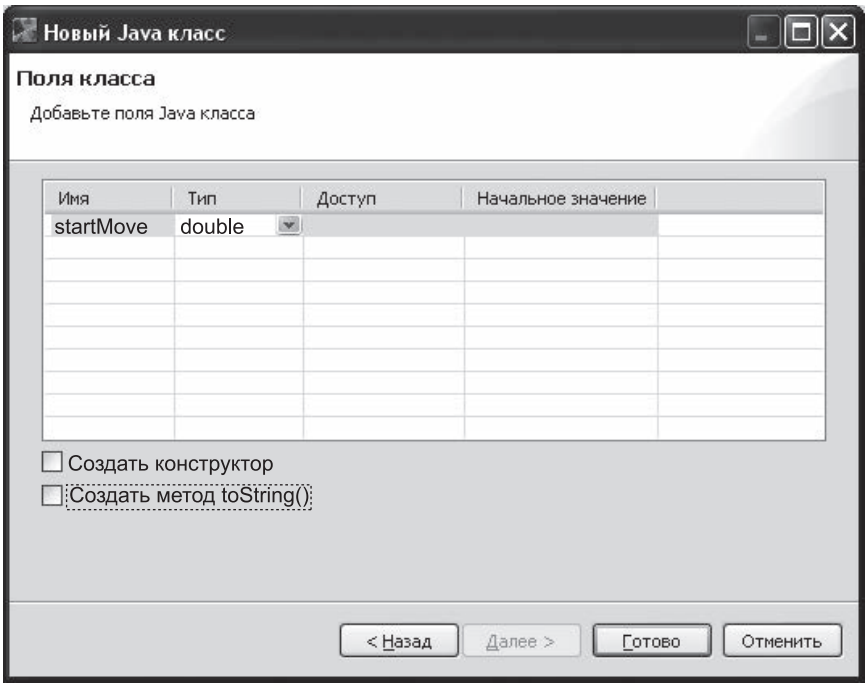

Рис. 10. Добавление параметра startMove

- в пункте **Действие при выходе** запишем в добавленный нами параметр время: `((schoolPed)ped).startMove = time();`
- поскольку в случае отказа пешеходы у нас идут в одно и то же место, модифицируем схему — исключим лишние объекты;
- добавим объект сбора статистики: из палитры **Статистика** перетащим объект **Статистика** и назовем его moveTime;
- сменим класс пешехода на всех объектах pedSink (так же, как на объекте-источнике pedSource);
- на всех pedSink укажем действие при входе, добавляющее данные к статистике: `moveTime.add(time()-((schoolPed)ped).startMove);` как видно, мы добавляем к статистике время, вычислив его как разницу между текущим временем и временем входа.

> Обратите внимание! Пожалуйста, учтите, что если вы не смените класс пешехода, это приведет к ошибке при попытке исполнения.

Теперь при выполнении прогонов можно не наблюдать за счетчиком, а просто отслеживать накопленные данные: минимальное, максимальное и среднее время прохода школьников.

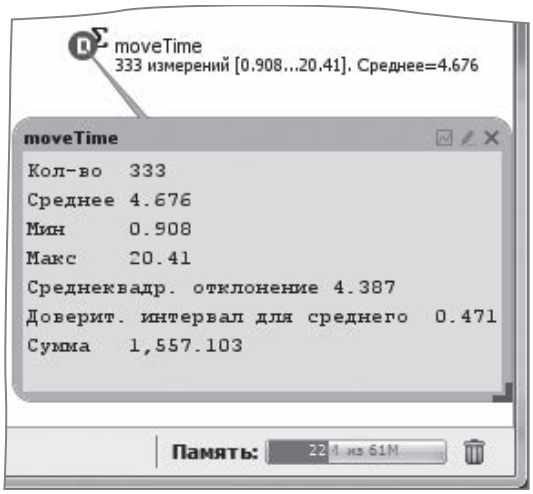

Рис. 11. Результат вычисления времени прохода школьников

Увидеть результаты можно, просто щелкнув мышью на статистике (рис. 11).

В этом прогоне среднее время прохода составляет примерно 4,7 мин, максимальное — 20 мин (!!!). В целом трудно признать, что турникет работает удовлетворительно.

Задания

1. Как изменится время, если карта будет обрабатываться от 1 до 2 с?
2. Сколько потребуется времени на проход, если детям придется еще и переодеваться (задержка от 2 до 5 мин в раздевалке)?
3. Модель предполагает, что люди приходят в течение всех 30 мин равномерно. Тем не менее очевидно, что на последних 10 мин плотность будет значительно выше. Исследуйте поведение модели с повышенной частотой появления людей.

Задача 2. *Простейшая модель распространения эпидемии*

Одна из ежегодных проблем крупного города — эпидемии гриппа. Требуется создать модель, которая позволит изучить действенность некоторых типовых мер по борьбе с эпидемиями — вакцинации, карантина и т. д.

Для решения этой задачи снова воспользуемся агентной моделью. Моделировать пространство целого города и поведение людей

не имеет смысла, поскольку для нас в этой модели неважно, как люди перемещаются по городу. Нам важно только, как количественно распространяется заболевание, а для этого «гонять» человечков не надо.

При этом агент фактически будет просто последовательно менять свое состояние: здоров, заражен, болен и выздоровел. Для простоты будем считать, что выздоровевший больше не заболеет. Кроме того будем считать, что человек может заболеть двумя путями: «случайно» (т. е. без непосредственного контакта с заболевшим) и при непосредственном контакте с зараженным или больным. Так что, для нас существенными параметрами болезни будут:

1) вероятность заражения «из воздуха»;
2) инкубационный период;
3) среднее время течения заболевания.

Это, конечно, очень упрощенная модель[1], но для проверки основных предположений она вполне годится.

Ключевой вопрос, который мы будем исследовать с помощью нашей модели, — это вопрос о параметрах, влияющих на скорость и широту распространения болезни. Опять-таки, упрощая, можем считать ключевой точкой каждого эксперимента момент времени, в который больных больше всего.

Для создания модели воспользуемся мастером. Обратимся к команде **Файл → Создать → Модель**, на первом шаге укажем название **Epidemic**, на втором шаге — что за основу берем агентную модель (рис. 12).

На следующем шаге укажем, что нам понадобится 500 агентов, далее — что они будут распределены случайно, размер пространства — 400*400 (т. е. ничего не будем менять).

На шестом шаге укажем, что объекты будут контактировать на расстоянии до 10 единиц (рис. 13).

Больше ничего добавлять не будем и создадим модель — нажмем кнопку **Готово**. Получим примерно следующее.

Мы создали модель, в которой при запуске будет создано 500 агентов на пространстве 400 на 400 единиц. Каждый агент будет «связываться» с соседями на расстоянии 10 единиц (рис. 14). По всему пространству они будут распределяться случайно, но происходить с ними пока ничего не будет — мы никаких действий им не приписали.

Наша модель, напомним, устроена таким образом, что перемещения агентов нас не интересуют, и фактически агенты будут

---

[1] Пример более аккуратной реализации подхода см. на сайте по адресу: http://ipi-cpl.ru/about/articles/kondratyev_epidemic_ntv_2010

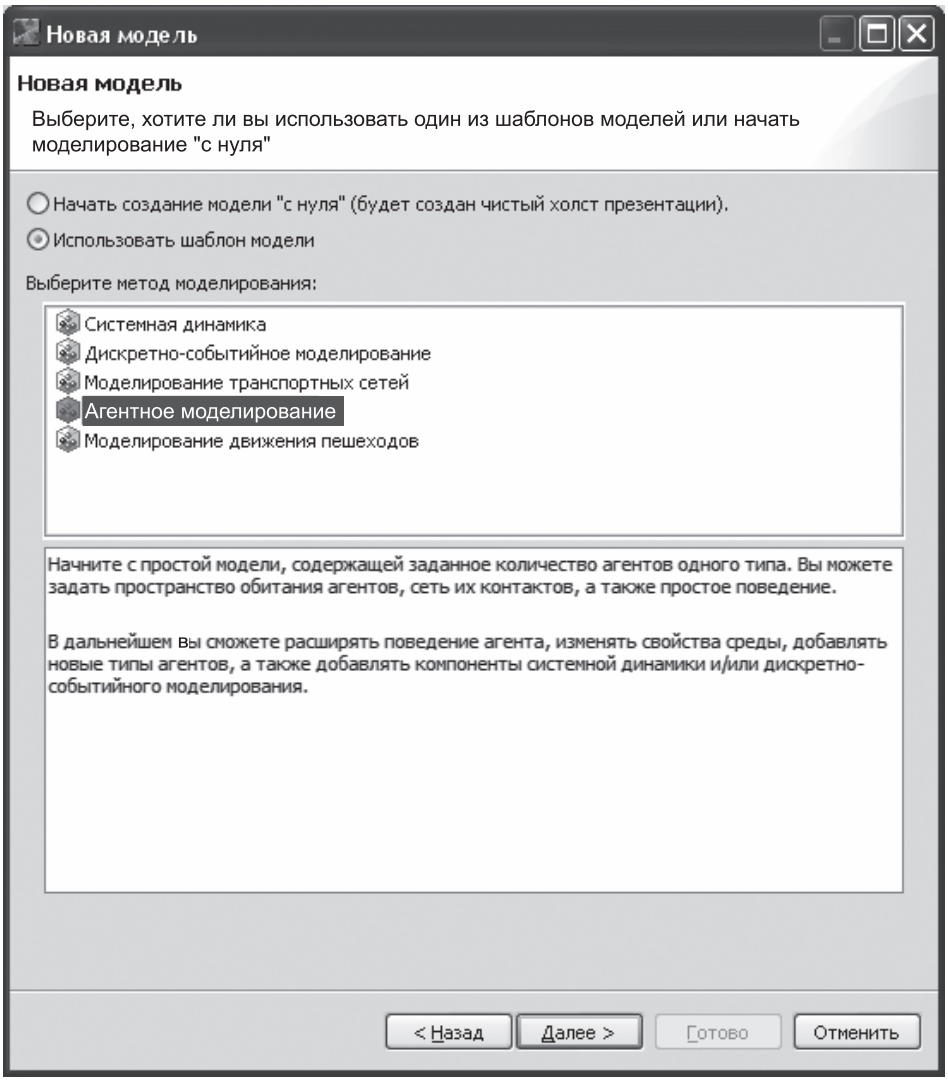

Рис. 12. Выбор агентной модели

просто менять свое состояние. Чтобы определить, какие состояния и когда будут возникать, мы должны открыть объект **Person** в графическом редакторе (см. рис. 14) и создать диаграмму смены состояний StateChart (рис. 15).

Наша диаграмма будет показывать последовательность состояний во время заболевания: сначала человек здоров, потом заражен (но еще не знает об этом и свободно перемещается), болеет и наконец выздоравливает.

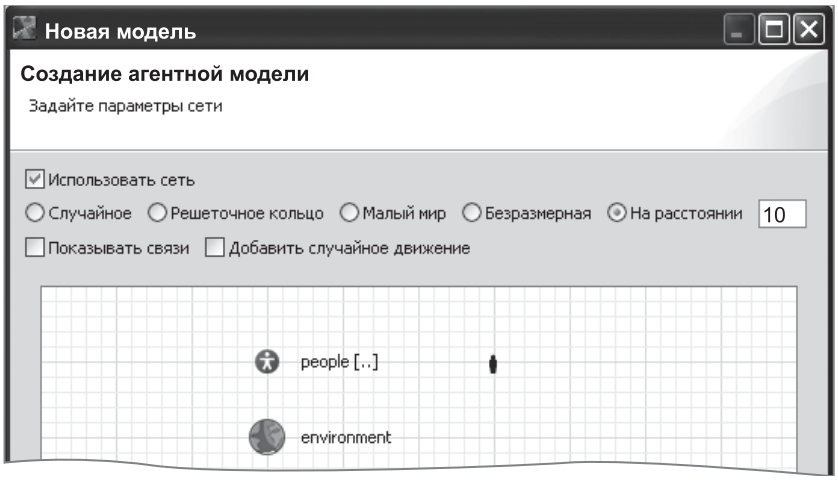

Рис. 13. Выбор расстояния между объектами

Откроем палитру **Диаграмма состояний** и перетащим на рабочее пространство объекта блоки. Блокам дадим названия: healthOk (Здоров), infected (Заражен), sick (Болен) и cured (Вылечен) (см. рис. 15). Чтобы их было легче различать, сменим цвет заливки.

Для простоты предположим, что:

1) выздоравливают все[2];
2) время вылечивания не меняется;
3) выздоровевший повторно не болеет.

Чтобы изучать влияние параметров на модель, из палитры **Основная (Main)** перетащим два объекта-параметра:

1) incubation (инкубационный период), зададим ему значение целого типа 5;
2) illTime (время болезни), зададим ему значение 10.

Блоки свяжем между собой соединителем-стрелкой из палитры **Диаграмма состояний**.

Получится примерно следующее (см. рис. 15).

Стоит заметить, что цвет блоков на состояние знака-человечка в рабочей модели не повлияет. Чтобы следить за происходящим, добавим на каждом состоянии команду, меняющую цвет в анимации (рис. 16).

Напишем для каждого блока действие при входе в него.

Для первого блока: *person.setFillColor(cyan);* для остальных блоков аналогично.

---

[2] К сожалению, в реальном мире даже для эпидемии гриппа это не так.

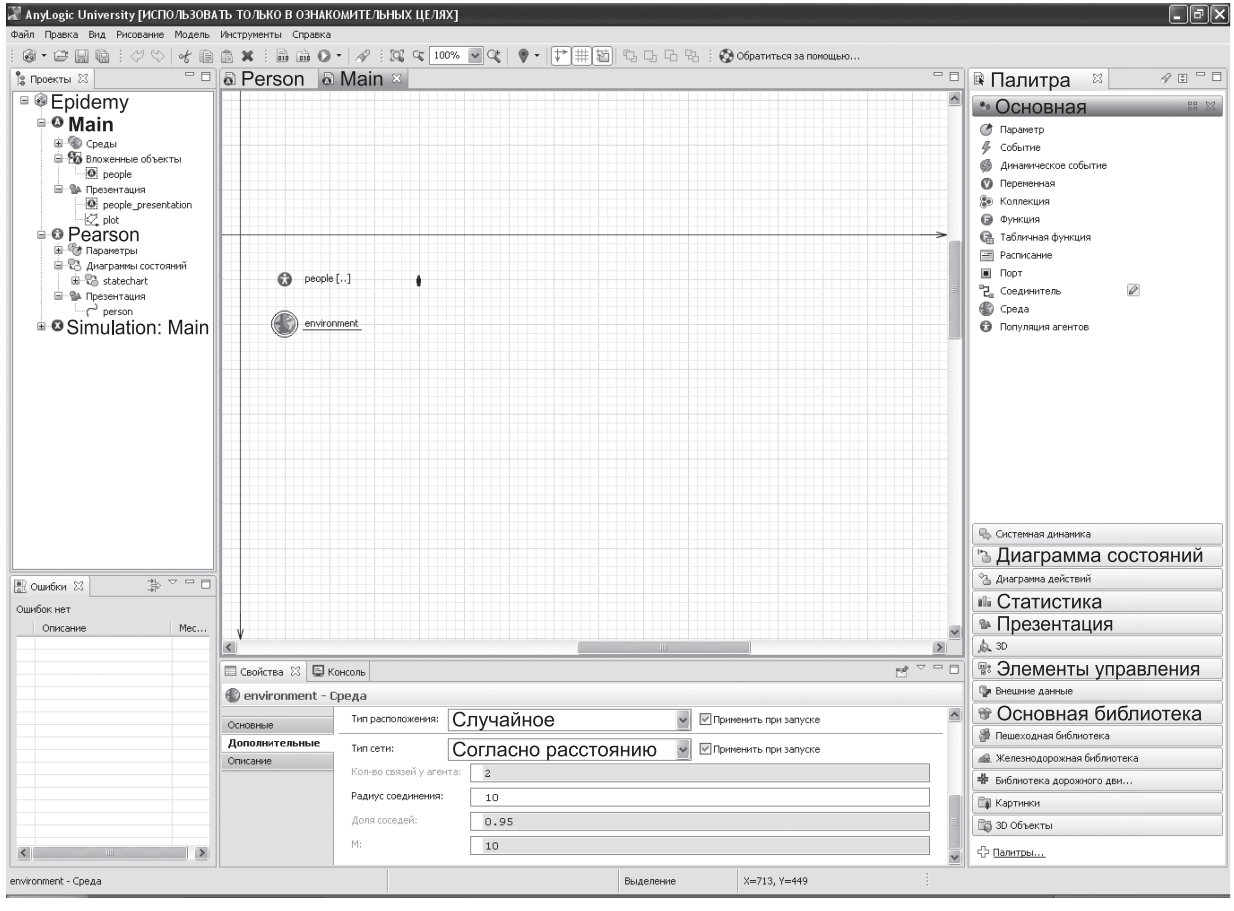

Рис. 14. Модель, в которой при запуске будет создано 500 агентов на пространстве 400 на 400 единиц

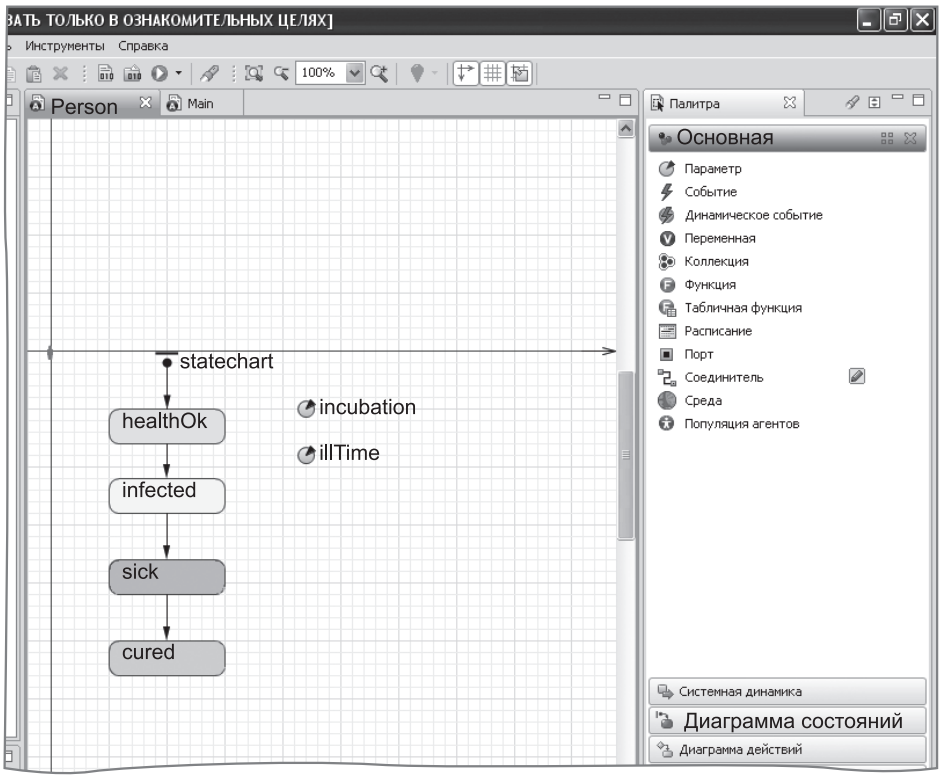

Рис. 15. Диаграмма смены состояний StateChart

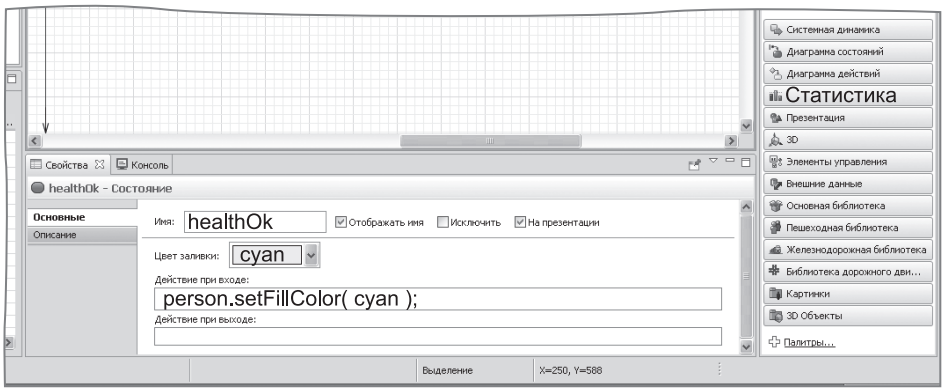

Рис. 16. Добавление команды, меняющей цвет в анимации

Смена состояния будет происходить по стрелкам-связям. Стрелка от «Здоров» к «Заражен» — первичное заболевание происходит из-за заражения случайно (без прямого контакта), т. е. просто с некоторой заданной интенсивностью. Зададим для стрелки частоту «0.0311» (рис. 17).

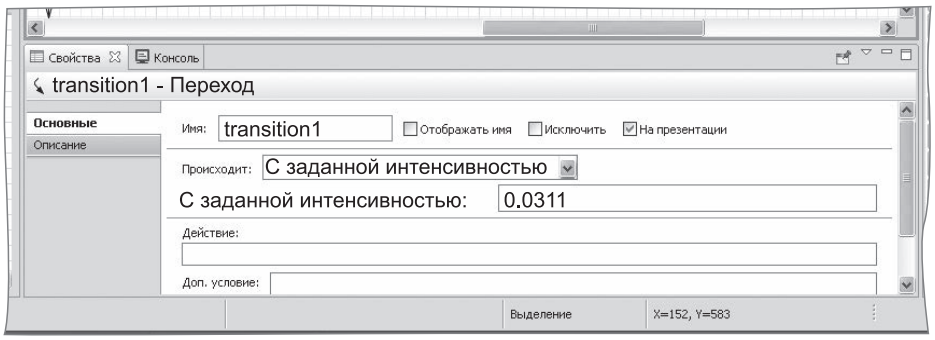

Рис. 17. Задание интенсивности заражения

Второй переход — от состояния «Заражен» к состоянию «Заболел» — будет определяться уже только временем инкубационного периода. Выберем для стрелки параметр По тайм-ауту, но вместо конкретного значения сошлемся на имя параметра (рис. 18).

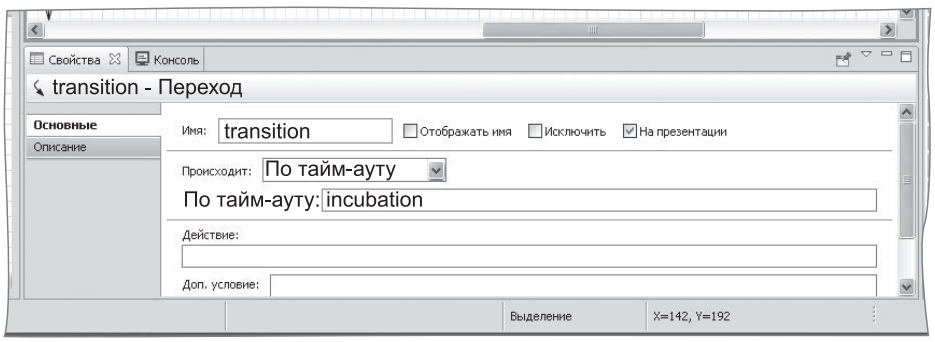

Рис. 18. Выбор параметра По тайм-ауту

Так же поступим и для последнего перехода.

Обратите внимание! Пока наша модель никак не зависит от количества контактов и близости людей друг к другу, т. е. единственный способ «заразиться» — это попасть под распределение.

Вернемся к общему экрану (закладка **Main**) (см. рис. 14).

Чрезвычайно неудобно следить за распространением эпидемии «вручную». Воспользуемся для этих наблюдений средствами автоматического сбора статистики. Включим сбор статистики у агентов, выберем объект people и переключимся на закладку **Статистика** (см. рис. 14). Изначально там ничего нет, но у нас есть возможность добавить счетчики. Сначала подсчитаем количество здоровых людей:

- добавим функцию сбора статистики;
- назовем ее healthOkCount;
- подсчитаем количество.

Условие подсчета будет состоять из двух: человек находится в состоянии либо «здоров», либо «выздоровел». Вот как выглядит условие «Здоров»: `item.statechart.isStateActive(item.healthOk)`. Соединим его с условием «выздоровел» с помощью связки «или». Названия статусов мы задали сами. Получим примерно вот что:

```
item.statechart.isStateActive(item.healthOk)
|| item.statechart.isStateActive(item.cured)
```

Аналогично добавим счетчик infectedCount с условием «заражен» или «болен» (рис. 19).

Теперь разместим средство просмотра — временной график:

- перетащим необходимый объект из палитры **Статистика** на рабочее пространство (см. рис. 16 — выделено);
- на закладке **Основные** этого объекта добавим элемент данных (нажмем кнопку);
- назовем элемент healthOk (рис. 20).

Получать значения будем из нашего объекта people с помощью заданной нами функции сбора статистики `people.healthOkCount()`.

Точно так же добавим второй элемент данных — infected.

Запустим модель и увидим примерно такую картину (рис. 21).

Максимум заболевших приходится на десятый день (удивительно, но это время инкубационного периода и периода явной болезни, после которого люди начинают выздоравливать). Пока что мы моделируем фактически незаразную болезнь, поскольку заболеваемость от количества собственно заболевших или зараженных не зависит никак.

Доработаем нашу модель, учтя распространение инфекции от заболевших. Сделаем это с помощью механизма рассылки сообщений.

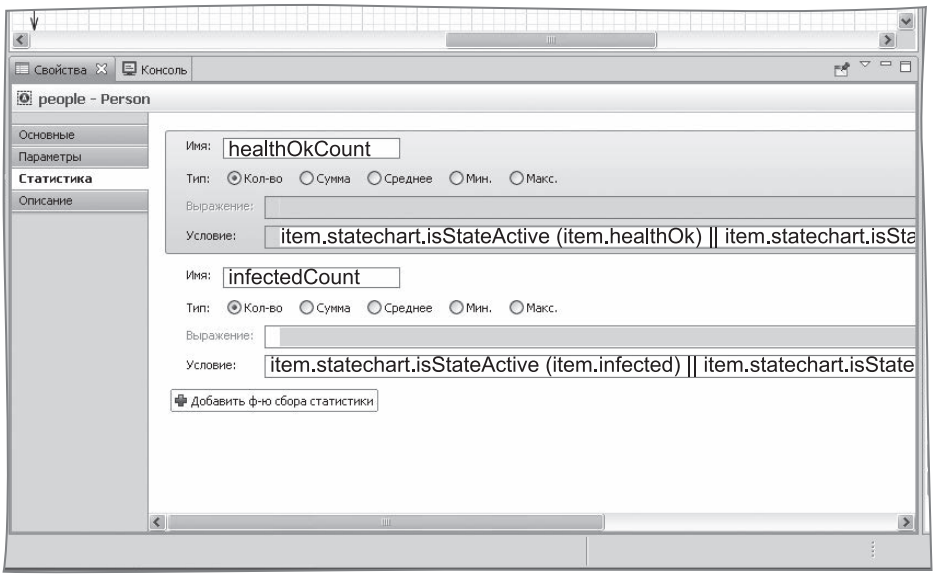

Рис. 19. Добавление счетчика **infectedCount** с условием «заражен» или «болен»

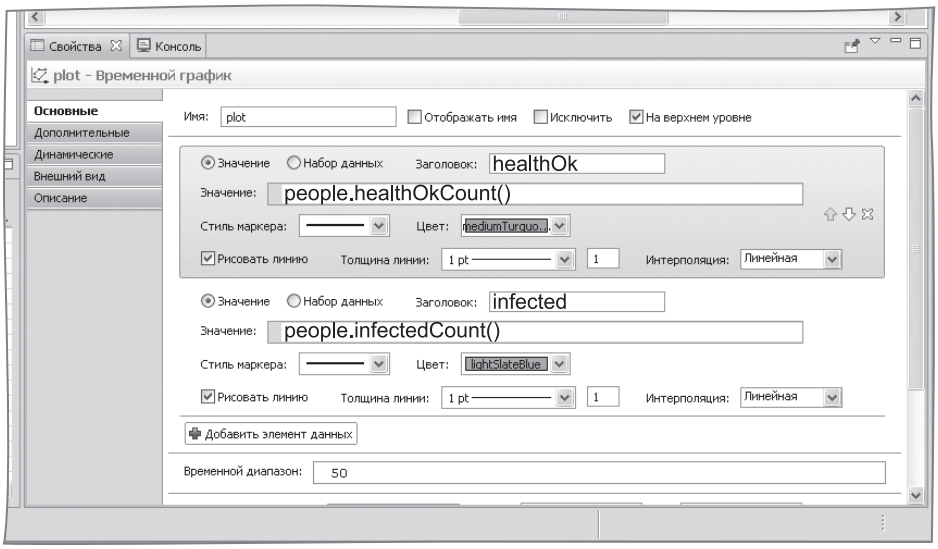

Рис. 20. Размещение средства просмотра — временно́го графика

В диаграмме добавим два внутренних (внутри блоков) перехода — внутри состояний infected и sick (рис. 22). У этих переходов с заданной интенсивностью будем рассылать сообщения «ill». Рассылка сообщений будет случайной, поскольку неизвестно с кем контактирует человек во время болезни.

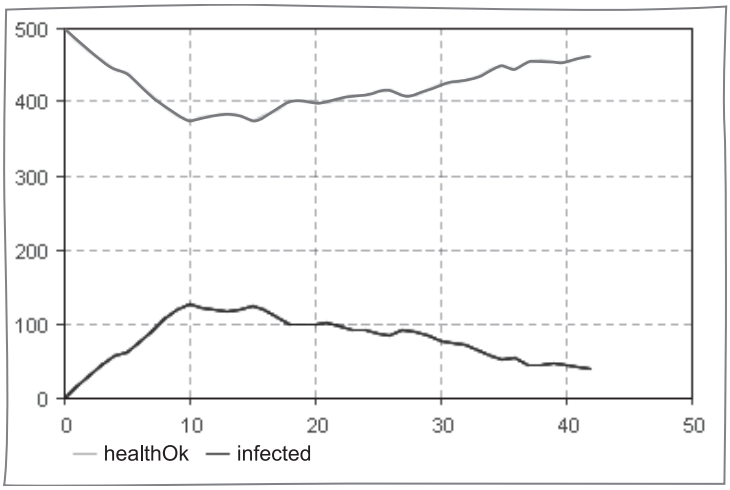

Рис. 21. Временной график без учета заражения

Для отработки сообщения добавим второй переход от состояния «Здоров» к состоянию «Заражен», который будет происходить при получении сообщения «ill» (рис. 23).

Сообщение будет приходить случайным людям, находящимся на расстоянии 10 единиц (которое мы задали при создании модели). Чтобы сообщения приходили агентам, зададим свойства класса-агента: выберем в дереве класс Person и зададим на странице агент — действие при получении сообщения (рис. 24).

Действие `statechart.receiveMessage(msg);` передает сообщение в диаграмму смены состояний.

Для начала предположим, что больные люди передвигаются по пространству точно так же, как здоровые и зараженные, т. е. поддерживают обычную частоту контактов (25 сообщений в единицу времени).

Выполнив прогон, увидим совсем другую картину (рис. 25).

К моменту начала снижения количества заболевших (10 дней) их количество становится равно количеству здоровых — 250 человек. В прошлый раз оно не успело подняться и до 200.

Введем ограничение: пусть заболевшие сидят в карантине, т. е. посылают только три сообщения «болеть» (рис. 26). Как видим, улучшение есть, но не принципиальное[3]. Если сделать несколько

[3] Авторам кажется, что это неплохо согласуется с историей распространения, например, чумных эпидемий и «успешностью», например, английских запираний.

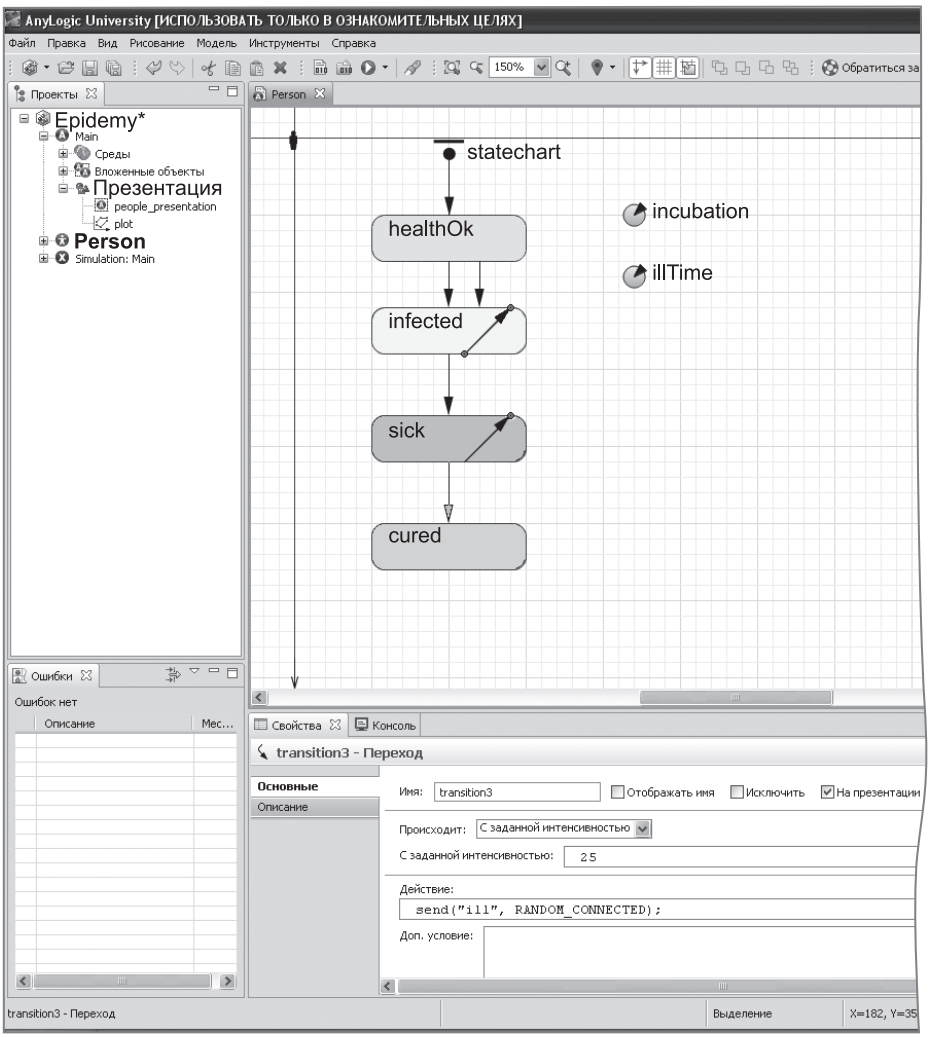

Рис. 22. Добавление переходов внутри состояний infected и sick

прогонов, то будет видно, что в обоих вариантах ситуация может быть и хуже, и лучше — в зависимости от распределения людей.

Можно предположить, что существенная разница возникнет в том случае, когда будет несколько «ареалов обитания» людей — тогда рост очага заболевания будет гораздо ниже. В пределах же уже зараженного поселения статистическая разница будет невелика.

Сделаем нашу модель более подходящей к картине распространения серьезной инфекции:

1) источником заражения так или иначе всегда является один человек-носитель. Уберем «фоновое» заражение;

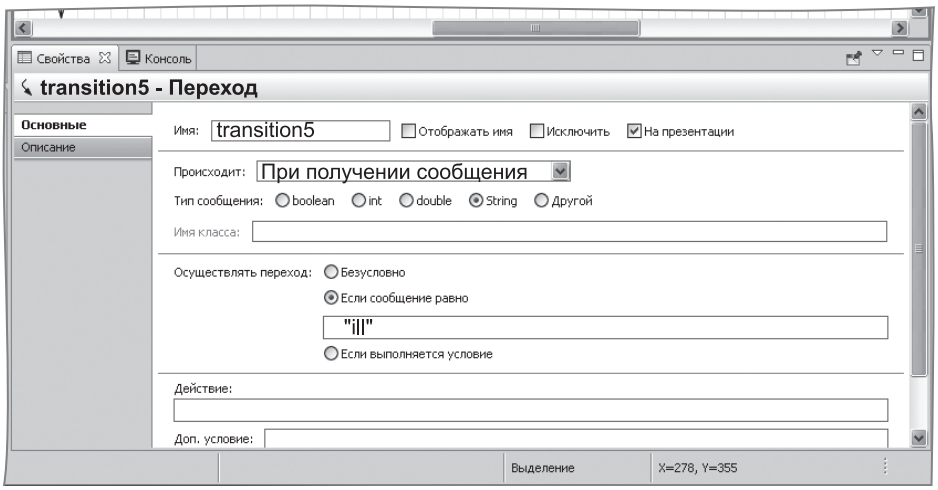

Рис. 23. Добавление второго перехода от состояния «Здоров» к состоянию «Заражен» при получении сообщения ill

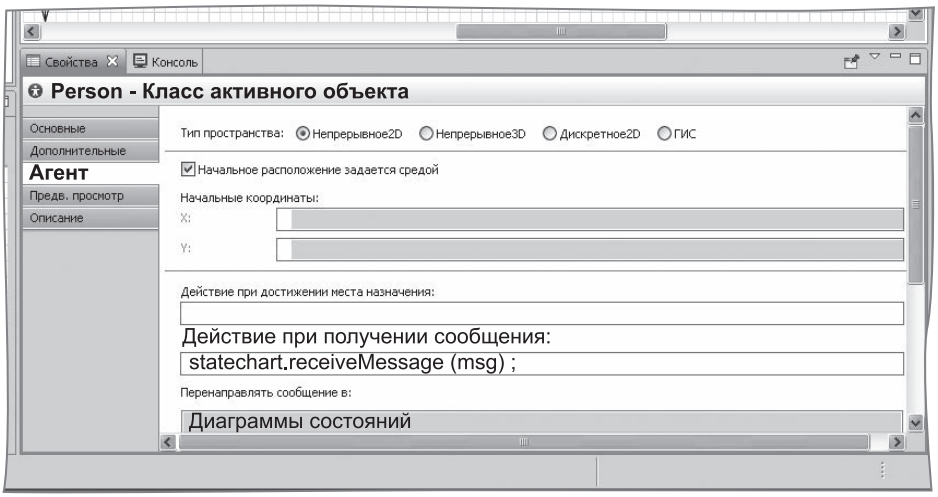

Рис. 24. Зададим на странице агент statechart.receiveMessage(msg) — действие при получении сообщения

2) если лечения нет, то в распространении эпидемии возникает понятие «смертность».

Чтобы исследовать поведение заболевания в нашей модели без необходимости каждый раз менять параметры вручную, вынесем эти параметры на стартовую страницу и будем их менять.

Первые изменения коснутся главной страницы (Main). Перенесем на нее параметры incubation и illTime и добавим новые параметры: contactNorm (со значением 25) и sickNorm (со значением 3) — количество контактов для «свободного» и «заболевшего»

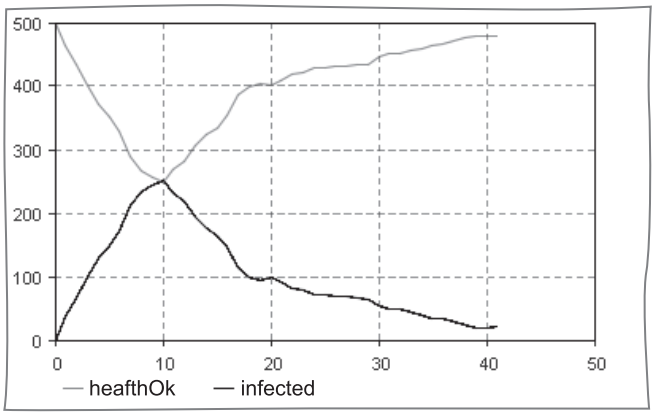

Рис. 25. Временной график с учетом заражения

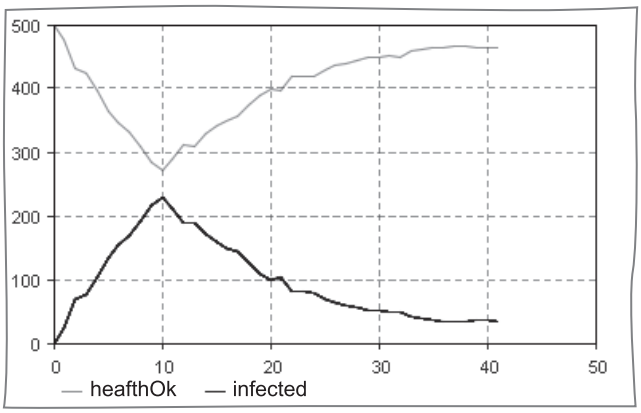

Рис. 26. Временной график при введении карантина

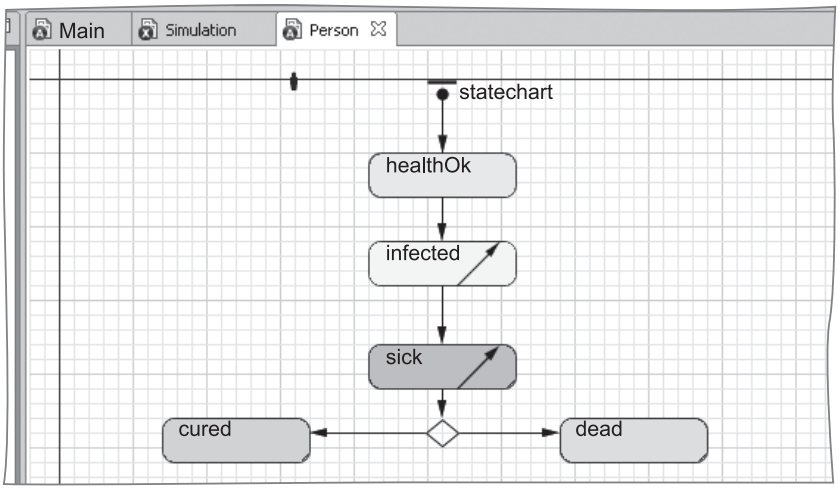

Рис. 27. Добавление элемента ветвления

человека, а также параметр distance — дистанция, на которой болезнь будет действовать.

Теперь внесем изменения в диаграмму смены состояний. Сначала изменим структуру (рис. 27).

1. Уберем первый переход от состояния «Здоров» к состоянию «Заражен».
2. Добавим после состояния «Болен» элемент ветвления и от него два исхода: «Выздоровел» и «Умер».

Условия перехода описываются в параметрах стрелок, выходящих из разветвления. Условие перехода к состоянию «здоров» будет таким: `uniform() < 0.98`. Оно означает, что случайное число, равновероятно попадающее в промежуток от 0 до 1, должно оказаться меньше 0,98, т. е. в 98 случаях из 100 человек выздоравливает.

Второе условие задавать не будем — под него попадут все остальные.

3. Там, где мы задавали значения частот и тайм-аутов, в соответствующих полях переходов следует сослаться на значения параметров из основного объекта. Делается это примерно так: `get_Main().contactNorm` — так ссылаются из подчиненного объекта на параметр в основном (рис. 28).

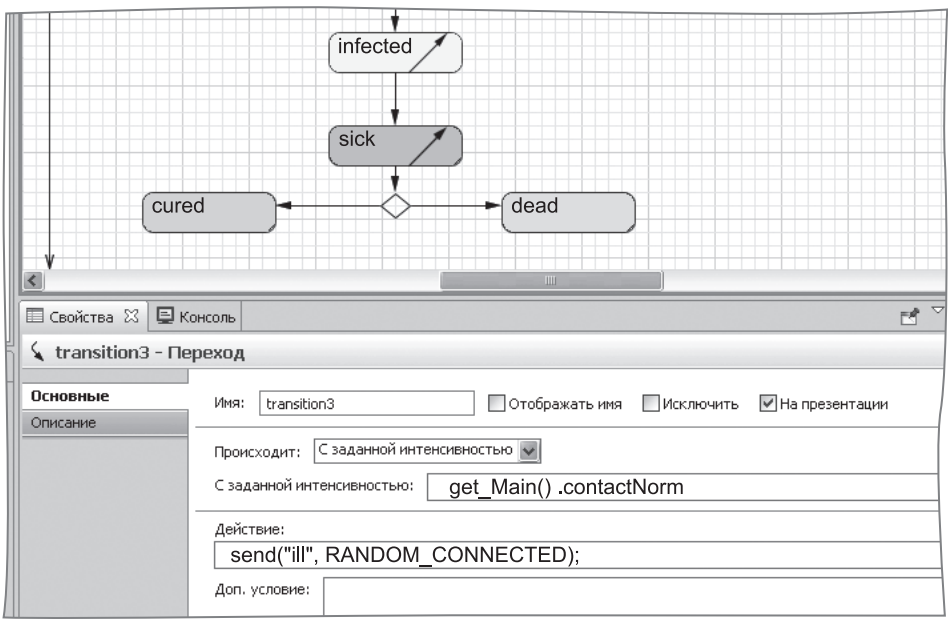

Рис. 28. Ссылка из подчиненного объекта на параметр в основном

Раньше заболевание начиналось с заразившихся «из воздуха», теперь у нас таких не осталось. Чтобы эпидемия стартовала, в свойствах главного объекта (Main) укажем действие — посылку сообщения «ill» первому по списку агенту. Он и будет нашей стартовой точкой. Для этого запишем в поле «Действие при запуске» следующее:

```
people.get(1).statechart.receiveMessage( "ill" );
```

И наконец, нужно обеспечить возможность задавать значения параметров на стартовой странице. Для этого откроем в графическом редакторе страницу **Simulation** (см. рис. 14) и добавим на нее несколько элементов:

1. Добавим переменные (не параметры, а именно переменные!) из палитры **Основная**. Назовем их так же, как параметры, только с припиской Value.
2. Добавим элементы ввода. Сами элементы добавим из палитры **Элементы управления**, а подписи к ним — из палитры **Презентация**. Текст подписей наберем как значение свойства **Текст**.
3. Свяжем элементы управления с переменными. Для этого поставим в их свойствах галочку **Связать с** и напишем имя соответствующих переменных.

Должно получиться примерно так, как изображено на рис. 29.

4. И последнее, свяжем параметры с переменными. Выберем в дереве объектов **Simulation** и на странице **Основные** зададим в строках появившихся параметров значения переменных (рис. 30).

> Элементы для сбора и отображения данных о смертности можете сделать сами, как было выше описано.

Теперь выполним несколько прогонов с разными значениями параметров. С параметрами, изначально заданными нами, картина, скорее всего, будет примерно такой, как на рис. 31.

Как видим, картина резко изменилась — заболела только небольшая группа людей. Причина очень проста — остальные не оказались в «радиусе заражения».

Если же радиус увеличить до 25, картина будет примерно такой, как на рис. 32. Как видим, количество заболевших выросло. При этом количество контактов, сроки заболевания и другие факторы на максимальное количество пораженных будут влиять мало. Основным фактором, как мы и предсказывали, оказывает-

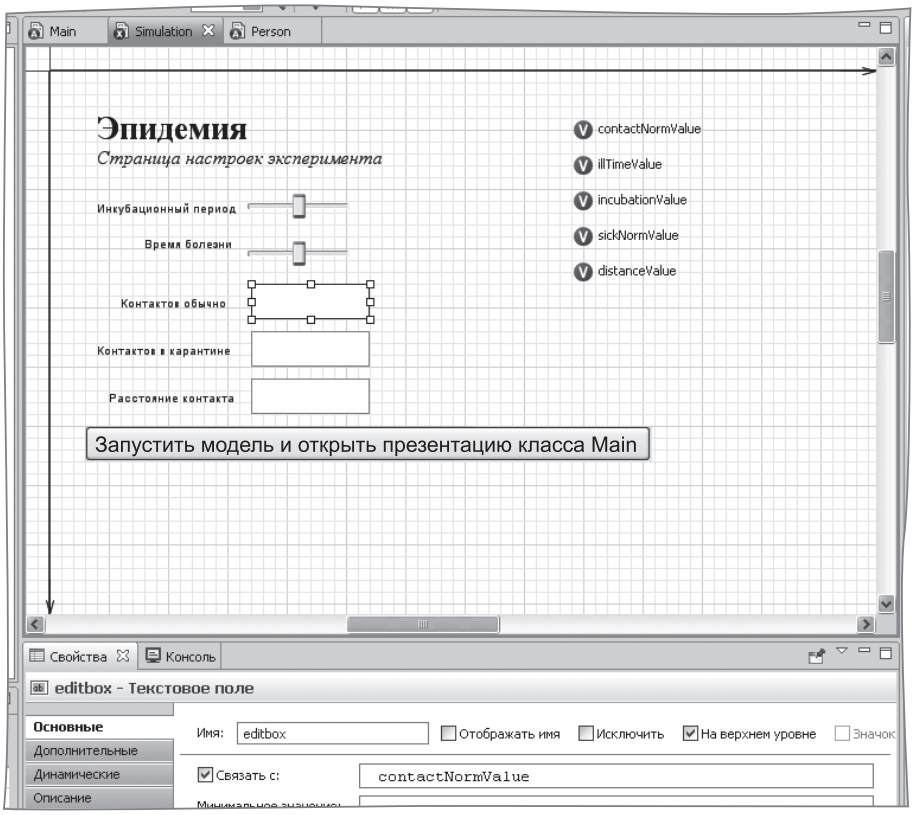

Рис. 29. Связывание элементов управления с переменными

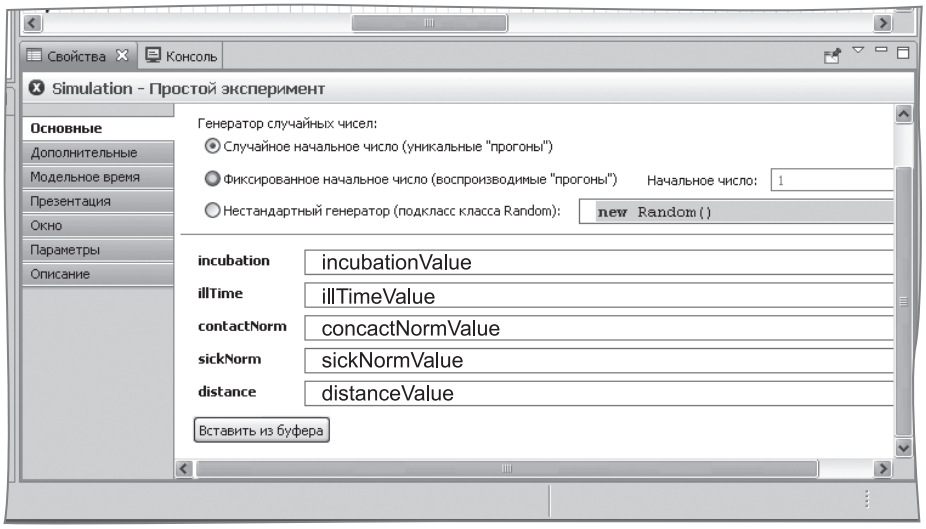

Рис. 30. Связывание параметров с переменными

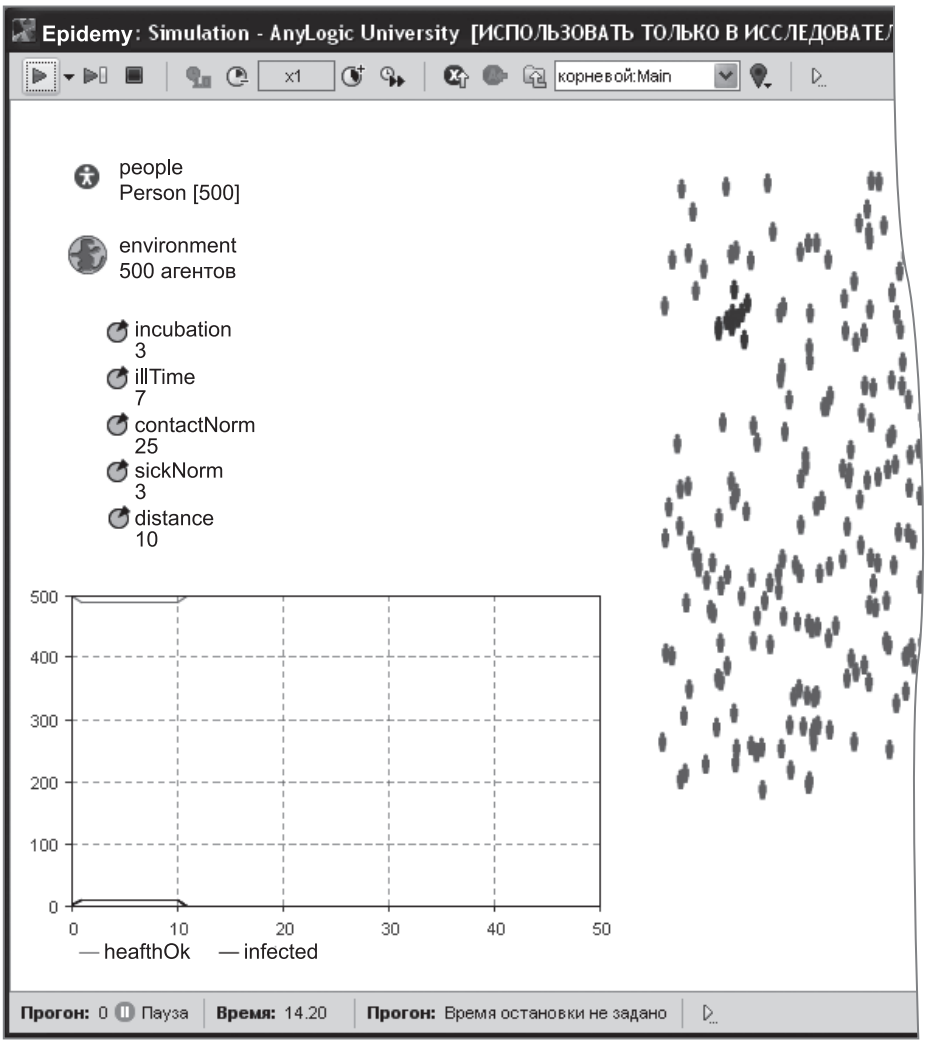

Рис. 31. Картина заболеваемости при радиусе заражения до 10 м

ся «площадь контактов», т. е. территория плотного расселения. Таким образом, ввод карантина — мера целесообразная, но только для целых районов (т. е. областей, не связанных с внешним миром). Внутри пораженного района уменьшить количество заболевших не удастся, поскольку большинство болезней имеют период, в течение которого болезнь еще не заметна, но уже заразна.

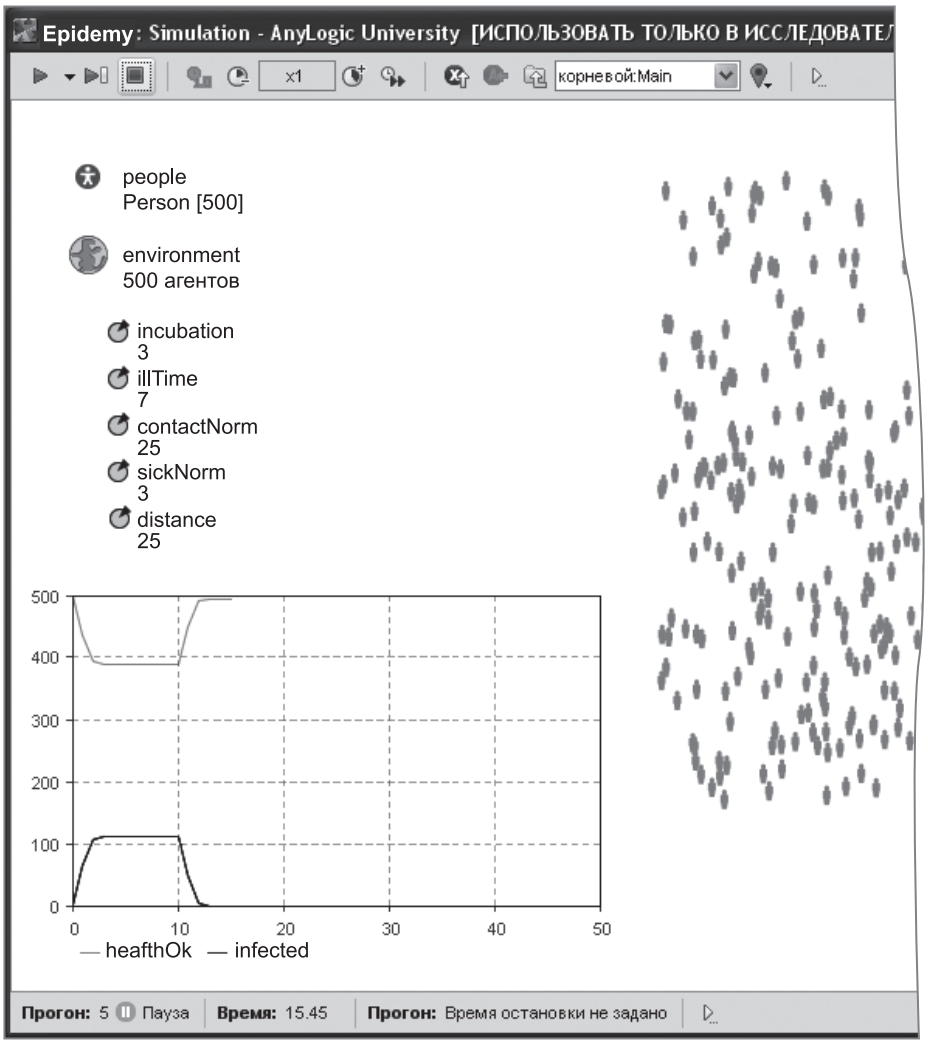

Рис. 32. Картина заболеваемости при увеличении радиуса заражения до 25 м

Задания

Самостоятельно усовершенствуйте модель, чтобы изучить влияние таких параметров, как:

1) изменение плотности расселения, например, в результате увеличения площади;
2) применение лечения, как варианта, при котором время болезни и смертность сокращаются;
3) если приобретенный иммунитет окажется временным;

4) в результате вакцинации, предположим, примерно 50% людей не заболеют (в течение какого-то промежутка времени). Как это повлияет на распространение болезни во время конкретной эпидемии? В долгосрочной перспективе?

Задача 3. *Дискретно-событийная модель работы медицинского учреждения*

Ежегодно работникам многих предприятий полагается проходить так называемый профессиональный медицинский осмотр (диспансеризацию). Процедура его построена примерно так: в назначенный день работник является в поликлинику, получает в регистратуре документы, получает от терапевта направление и должен пройти осмотр у нескольких врачей-специалистов. Люди выбирают последовательность прохождения специалистов сами, исходя из того, у кого очередь короче.

Общее время на проведение осмотра, выделенное лечебно-профилактическим учреждением, составляет 6 ч. Нам известно, сколько примерно времени тратится на осмотр и выдачу документов.

Требуется определить: сколько людей успеют пройти осмотр, не будут ли они где-то скапливаться при имеющихся нормах времени, каково среднее и максимальное время прохождения осмотра?

В модели, с помощью которой мы собираемся оценить эти параметры, не важна траектория движения людей, их взаимодействие между собой и т. д., важно только время, затрачиваемое на каждый этап обслуживания. При этом обслуживание для нас дискретно: нельзя же осматривать полтора человека.

Для моделирования воспользуемся специальным видом моделей, который называется *дискретно-событийным*. Основная единица в нем — заявка на обслуживание, которая будет обрабатываться некоторое время на каждом этапе.

Создадим чистую модель (**Файл → Создать → Модель**), на втором этапе выберем «с нуля» (рис. 33 и 34).

Для создания модели нам понадобится, как и в предыдущих случаях, разместить необходимые объекты-этапы на рабочем поле и связать их между собой. Для дискретно-событийной модели воспользуемся объектами с закладки **Основная библиотека** (см. рис. 14).

Обработка начинается с создания заявки; для этого перенесем из основной библиотеки объект **Source**. Каждый этап будет состоять из двух объектов: «Очередь» (Queue) и «Ожидание» (Delay).

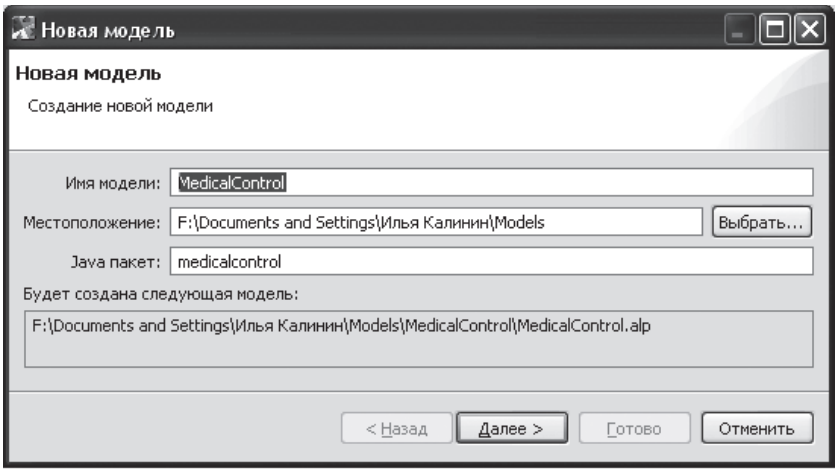

Рис. 33. Создание дискретно-событийной модели

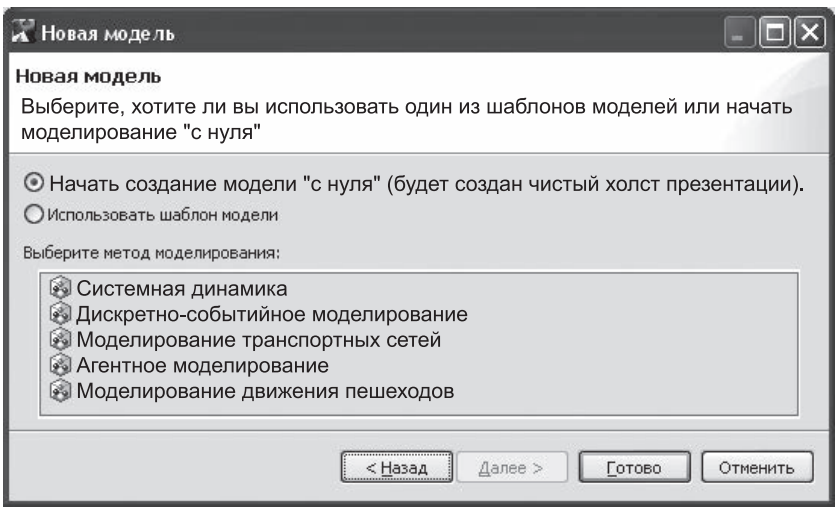

Рис. 34. Выбор пустого шаблона создания модели

Чтобы учесть выбор очереди к специалисту, нам придется вывести заявку из потока (с помощью объекта «Выход» — Exit) и потом для каждого специалиста принять заявку в объекте Enter. После обработки заявки уничтожаются в объекте Sink.

Там, где обработка идет линейно, свяжем объекты, как и в предыдущих моделях.

Для трех специалистов получится схема, похожая на рис. 35.

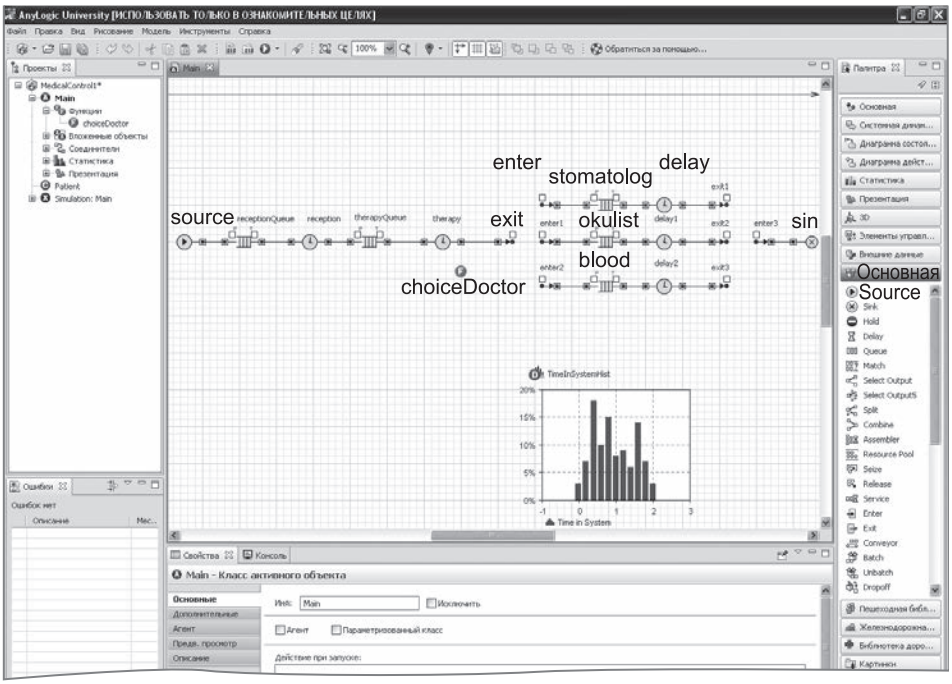

Рис. 35. Схема модели при приеме трех специалистов

Как и в предыдущем примере, нам понадобятся объект для сбора статистики и гистограмма для отображения данных. Основным параметром, статистику которого мы будем собирать специально, будет время, проведенное заявкой в системе, т. е. собственно время осмотра. Количество людей мы сможем узнать из чисел на объектах.

> Обратите внимание! Мы не связали очереди специалистов с основным потоком — заявки будут попадать туда с помощью специальной функции.

Для работы модели нам понадобится создать новый класс заявок, как в прошлом примере для пешеходов. Назовем его Patient. Класс создаем на основе класса Entity (заявка) и описываем четыре дополнительных переменных:

```
boolean stomatolog;
boolean okulist;
boolean blood;
double ArrivalTime;
```

В них мы учтем специалистов, которых человек прошел, и время входа в здание.

В конструкторе класса напишем начальные значения:

```
public Patient(double ArrivalTime){
  this.ArrivalTime = ArrivalTime;
  this.blood = false;
  this.okulist = false;
  this.stomatolog = false;
  }
```

Этот класс — Patient — нужно указать на всех объектах, через которые проходит заявка.

Кроме того, нам понадобится специальная функция выбора: в какую из очередей поставить заявку. Для этого из палитры **Основная** перетащим в рабочее окно компонент **Функция**, дадим ей название **choiceDoctor** и создадим один входной параметр entity, т. е. заявку из уже созданного нами класса Patient. Страница основных свойств функции будет выглядеть примерно, как на рис. 36.

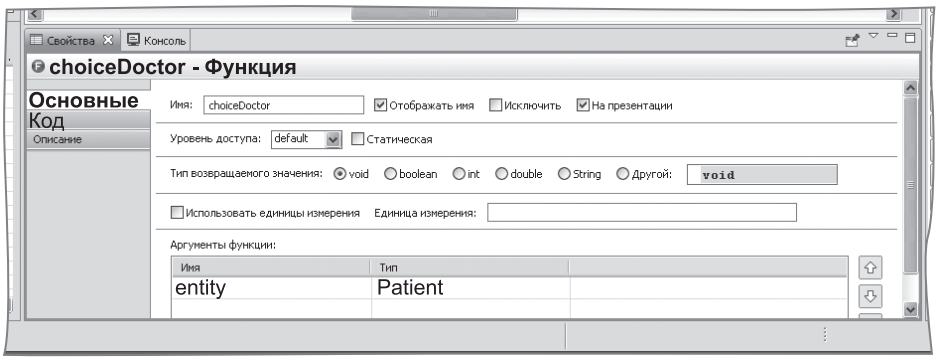

Рис. 36. Страница основных свойств функции

Конечно, функции необходим и программный код, который и будет реализацией. Код запишем в разделе «Код». Реализовать его можно по-разному, например так:

```
if ((entity.blood)&&(entity.stomatolog)&&
                    ((entity.okulist)))
        enter3.take(entity);
else{
   int bloodSize = ((!entity.blood)?( blood.
                       size()):(255));
```

```
int okulistSize = ((!entity.okulist)?( okulist.
                    size()):(255));
int stomatologSize = ((!entity.stomatolog)?
                     (stomatolog.size()):(255));
if ((bloodSize <= okulistSize)&&(bloodSize
              <= stomatologSize))
{
   entity.blood = true;
   enter2.take(entity);
} else
   if (okulistSize <= stomatologSize)
   {
         entity.okulist = true;
         enter1.take(entity);
   } else
   {
         entity.stomatolog = true;
         enter.take(entity);
   }
}
```

Логика работы этой функции такова:

1) если человек прошел всех специалистов — он свободен;
2) если нет, то выясняем размер очереди для каждого из них. Для тех специалистов, которые уже обработали заявку-обращение, делаем этот размер заведомо неприемлемым;
3) выбираем самую короткую очередь, ставим заявку в нее и делаем отметку, что человек этого специалиста прошел, потому пока заявка не выйдет из очереди, эти данные не используются.

Этой функцией мы будем пользоваться всякий раз, когда нужно выяснить, куда теперь направить заявку. Для этого во всех объектах exit укажем на странице свойств, что при выходе нужно вызвать действие: `choiceDoctor(entity)`.

Основная часть модели готова, но нужно задать еще время задержки. Очевидно, что время задержки на разных этапах будет разным для разных заявок. Это может быть обусловлено массой причин, но сами причины нам не важны. Важно, как распределится время обработки каждой заявки на каждом этапе. У нас нет полных статистических сведений, но есть средние оценки; будем считать время обработки заявки случайной величиной. Форма ее распределения нам неизвестна, поэтому воспользуемся простейшим подходом: допустим, что это «треугольник» с вершиной в области среднего значения.

Предположим, что время поиска и выдачи документов в регистратуре колеблется от 3 до 10 мин, а в среднем – 5 мин. Зададим время задержки так: triangular(3, 5, 10). Для начала у нас только одно окно выдачи.

Терапевт работает быстрее, время осмотра одного человека составляет от 2 до 5 мин, в среднем — 3 мин.

Стоматолог: от 5 до 15 мин, в среднем — 10 мин.

Окулист: от 5 до 10 мин, в среднем — 7 мин.

Забор крови: от 1 до 5 мин, в среднем — 3 мин.

Время на переходы игнорируем — предположим, что все происходит на одном этаже и человек видит все очереди.

Нетрудно посчитать, что формально максимальное время на полный осмотр составляет 45 мин. Теперь посмотрим, как это будет происходить в более точной модели.

Будем считать, что каждые 2 мин прибывает 1 пациент, т. е. в параметрах источника укажем скорость прибытия 0,5 единицы. Там же проверим, что не забыли указать класс заявки при создании (рис. 37).

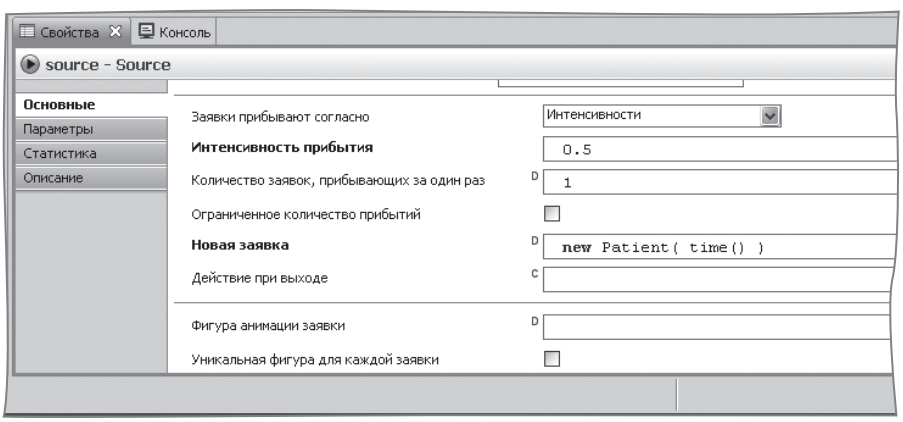

Рис. 37. Создание новой модели

Ограничим время работы модели: выберем ветку Simulation и в ее свойствах на закладке **Модельное время** укажем конечное время работы 480 мин.

Теперь можно запустить модель и сразу отметить несколько особенностей: у нас появляется очередь в регистратуру, причем она все время растет; к окулисту и стоматологу всегда стоит очередь, а на анализ крови ее нет.

Результаты исполнения таковы (рис. 38): примерно через 5,5 ч очередь в регистратуру переполняется. За это время осмотр смогли

пройти только 30 человек, причем солидная часть из них потратила на это более 3 ч.

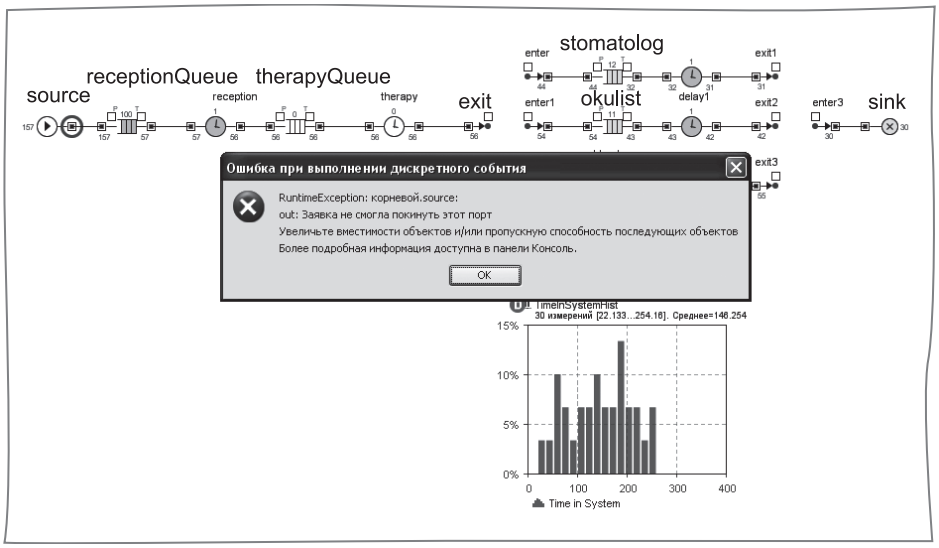

Рис. 38. Результаты прогона новой модели

Очевидно, что без снижения времени на осмотр и улучшения работы регистратуры такая система фактически не сможет выполнить свои задачи.

Задания

1. Почему на анализ крови практически не было очереди?
2. Как отразить в параметрах модели тот факт, что коридор, где ждут своей очереди на осмотр, — не безразмерный?
3. Что произойдет, если в очереди будут появляться заявки с длительным временем обслуживания, например любители поговорить о здоровье? Какие параметры нужно изменить, чтобы это выяснить?
4. Что произойдет в системе, если за счет внедрения электронной карточки врачи смогут уменьшить время на заполнение каждого документа на 2 мин?
5. Как повлияет на такую систему появление второго сотрудника в регистратуре? Вы можете реализовать это в модели с помощью объектов Service и ResourcePool (они заменят очередь и обслуживание в регистратуре) основной библиотеки. Пример использования — модель Bank из обучающих моделей AnyLogic.

Задача 4. *Системно-динамическое моделирование*

При организации работы компании сотовой связи в крупном городе один из существенных вопросов — сроки и условия наращивания мощности базовых станций сети. От этого показателя напрямую зависит качество обслуживания клиентов.

Если компания не предпринимает никаких особых усилий по привлечению клиентов, то их приток зависит от количества тех, кто уже обслуживается компанией. Сюда войдут и результаты стандартных затрат на рекламу (фиксированный процент выручки с клиента), и те, кто придет по рекомендации.

Модернизировать станции слишком часто нельзя, это приведет к большим расходам, а оборудование не успеет окупиться. Если мощность недостаточна, появляются недовольные клиенты. Клиенты, недовольные компанией, отказываются от ее услуг. Некоторое количество недовольных есть всегда, но если качество работы падает, их становится больше.

Выяснив, что мощность недостаточна, мы начинаем модернизацию, но она тоже потребует времени, а результат далеко не сразу окажет влияние на мнение (и динамику) клиентов.

Основным критерием успешности работы для нас будет отсутствие уменьшения общего количества клиентов, т. е. такая организация дела, при котором приток клиентов больше, чем потеря. Основная единица времени в этих подсчетах — неделя.

Вопрос: какое время задержки допустимо? Насколько нужно повышать мощность оборудования?

Структура сети, поведение конкретных клиентов, затраты на рекламу не могут оказывать принципиального влияния на ситуацию, поскольку *никак не влияют на взаимозависимости*. Нас фактически интересуют статистические параметры. Даже модернизация никогда не даст скачкообразного эффекта, потому что никогда не будет выполняться разом во всей сети.

Как уже рассказывалось в учебнике, для изучения влияния параметров на сложный процесс в крупных системах, как правило, нет смысла (а то и возможности) моделировать точное поведение каждого участника процесса.

Одним из наиболее мощных инструментов для решения задач такого рода является метод, получивший название «системно-динамическое моделирование». Его мы и применим: создадим новую чистую модель так же, как и в предыдущем примере.

Для создания модели воспользуемся палитрой **Системная динамика**. Основой нашей модели будет поток клиентов, который мы будем накапливать в компании.

- Поместим на рабочее поле модели объект «Накопитель» и назовем этот объект Clients. Изначально (по умолчанию) у нас будет 100 условных клиентов.
- Разместим два объекта «Поток»: приток клиентов и их потерю. Приток назовем NewClients, отток — LostClients.
- Конечную точку притока и конечную точку оттока поместим в накопитель так, чтобы соединитель принял вид, как на рис. 39.

Обратите внимание! Фактически мы предполагаем, что у нас есть бесконечный источник клиентов. В реальных условиях это, конечно, несколько не так.

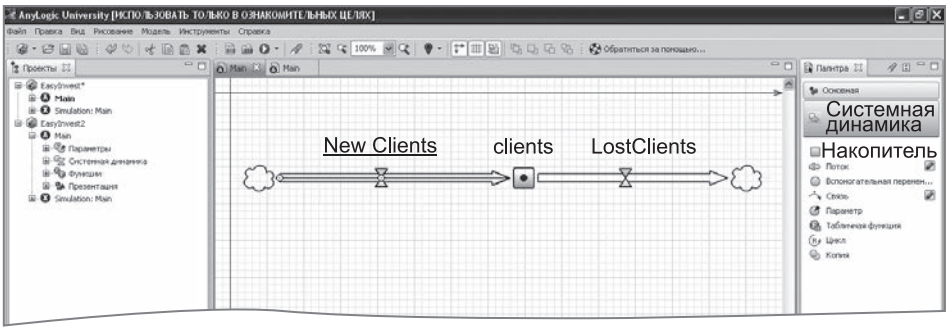

Рис. 39. Схема модели

На приток влияет реклама. Мы не изучаем никаких ее параметров, поэтому сведем все к одной вспомогательной переменной.

- Поместим на рабочее поле переменную, назовем ее *advertising* (рис. 40).

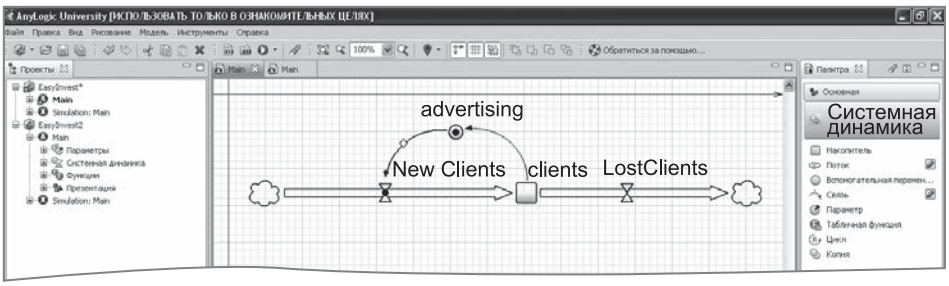

Рис. 40. Размещение переменной на рабочем поле

- Эту переменную свяжем с клиентской базой и притоком новых клиентов с помощью связей, которые тоже перетащим из палитры. Имена связям давать не нужно.

При этом возникнут два предупреждения: о том, что связь есть, но формулы для ее учета не заданы.

- Зададим значение этой переменной как формулу: clients/65 (рис. 41).

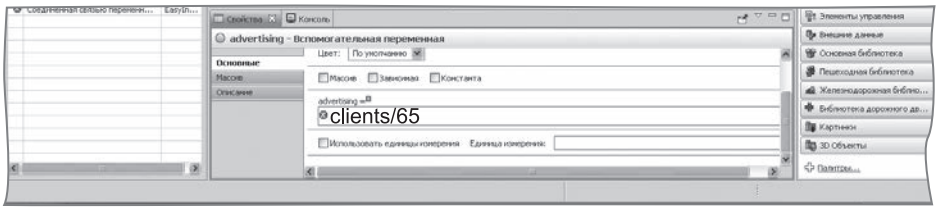

Рис. 41. Задаем значение переменной *advertising* как формулу clients/65

- Таким же способом зададим значение притока новых клиентов (рис. 42).

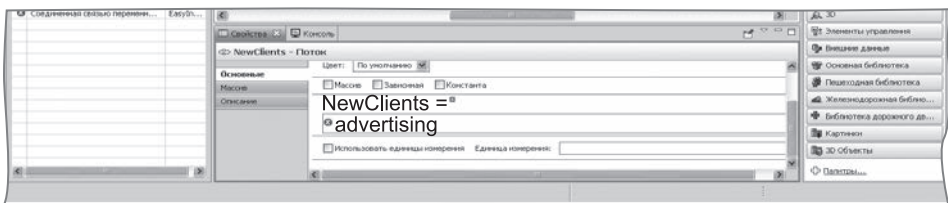

Рис. 42. Задаем значение притока новых клиентов

Сообщения об ошибках исчезнут.

Теперь перейдем к самому наращиванию мощности. Создадим вторую линию.

- Зададим имеющуюся мощность накопителем, который назовем capacity — вместимость. Изначальная мощность сети составит 60 условных клиентов.
- Мощность будет наращиваться потоком gradualCapacity из бесконечного источника, т. е. мощность мы наращиваем в зависимости только от наших решений.

Имеющаяся мощность по обработке обращений пользователей будет влиять на среднюю загрузку системы, которую мы вычислим как отношение мощности к количеству клиентов (clients/capacity).

- Добавим промежуточную переменную, свяжем ее с имеющимися параметрами и зададим формулу (рис. 43).

Далее опишем соотношение качества с нагрузкой. Качество обслуживания для клиентов — это нагрузка, но не текущая, а ранее работавшая, создавшая впечатление у клиентов. При этом нужно учесть, что зависимость эта нелинейная и задана некоторыми статистическими наблюдениями. Для этого создадим специальную функцию, в которой зададим зависимость не формулой, а набором значений.

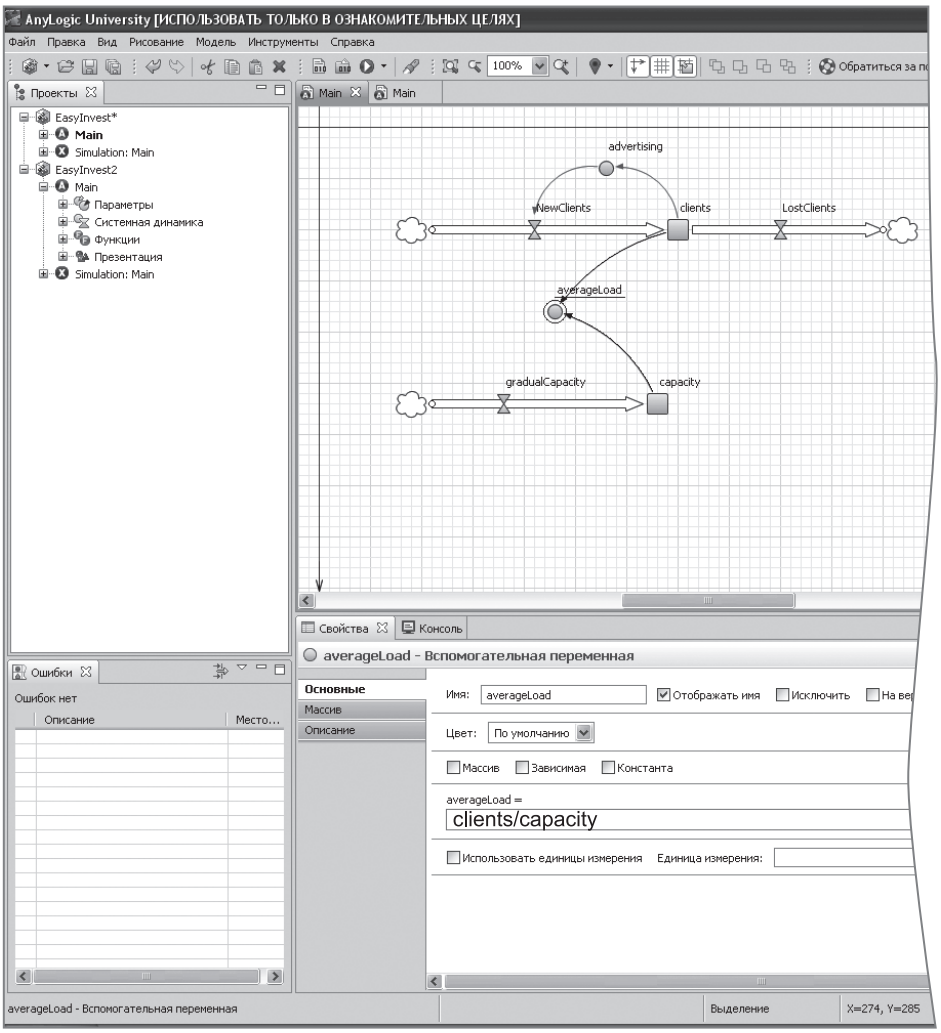

Рис. 43. Добавим промежуточную переменную

- Перетащим из палитры компонент «Табличная функция», назовем ее *qualityLoad* и зададим параметры (рис. 44).
- Создадим еще одну промежуточную переменную — *quality* и запишем формулу для этой переменной:
  `qualityLoad(delay(averageLoad,10))`.

Таким образом, оценка качества в виде количества «уходящих» клиентов будет даваться с задержкой на 10 единиц времени. Переменную мы свяжем с переменной средней загрузки и потоком уходящих клиентов. В «вентиле» уходящих клиентов просто напишем ее название.

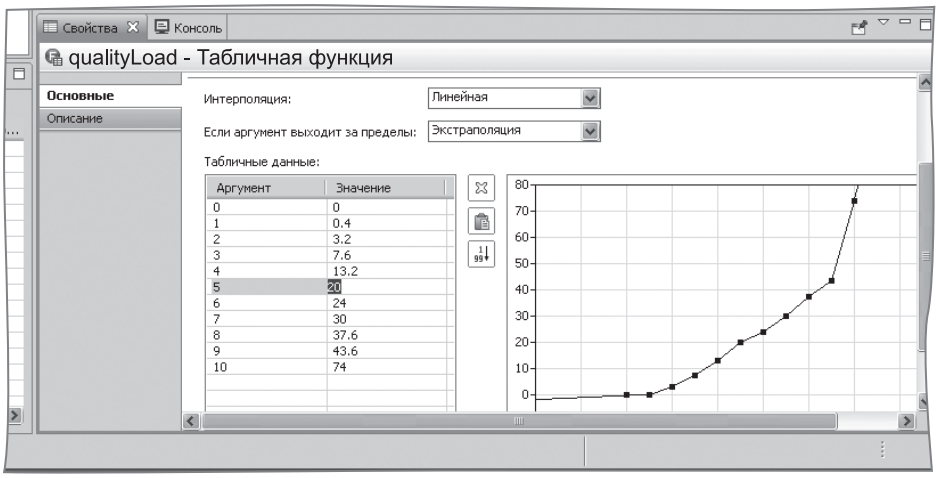

Рис. 44. Задание параметров табличной функции *qualityLoad*

- Для оценки и принятия решения о модернизации введем параметр *standard*, задающий нагрузку, при которой необходимо принять решение о модернизации (рис. 45). По умолчанию примем 1,7, т. е. примерно тогда, когда у нас за каждый цикл начнут уходить по 2 клиента.
- Параметр свяжем с текущей нагрузкой через промежуточную переменную *Capacity Defect* (недостаток мощности), вычислив ее как разницу между загрузкой и стандартом.

Влиять на увеличение пропускной способности она будет тоже с задержкой (рис. 45).

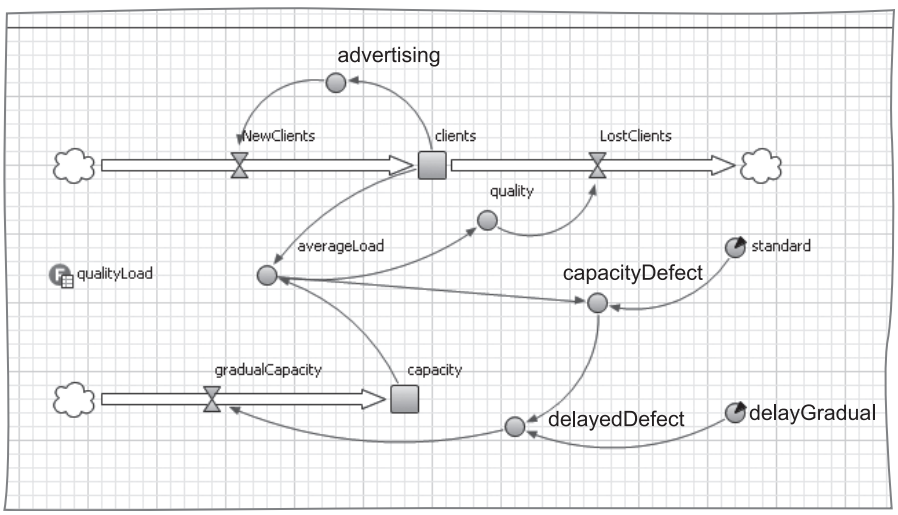

Рис. 45. Введение параметра *standard*

- Создадим параметр *delayGradual*. По умолчанию примем его за 30 целых единиц.
- Создадим промежуточную переменную для роста пропускной способности с задержкой *delayedDefect* и определим ее формулой: `delay(capacityDefect,delayGradual)`.
- Свяжем ее с текущим дефектом и потоком модернизации.

В результате получим схему, изображенную на рис. 45.

- На переменную *delayedDefect* мы сошлемся в «вентиле» потока gradualCapacity формулой `delayedDefect*5`, поскольку, очевидно, будем наращивать мощность с запасом.

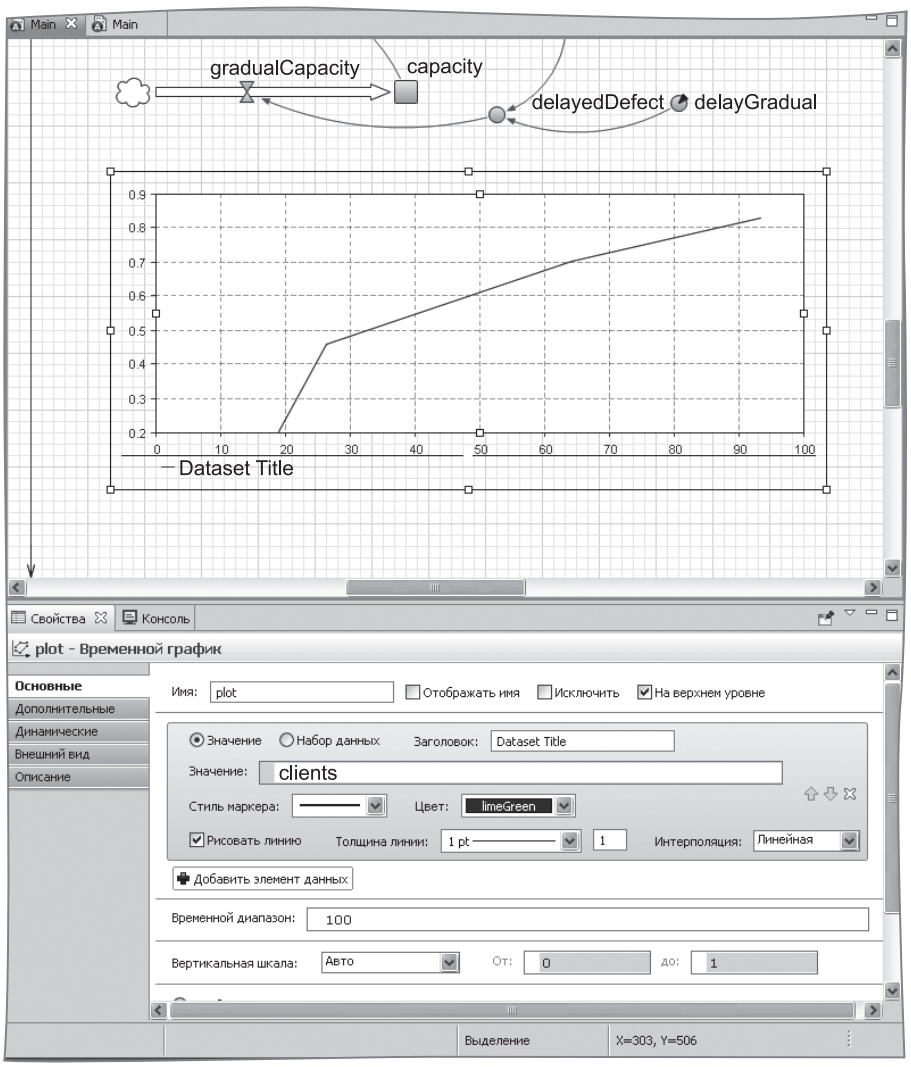

Рис. 46. Добавление на лист временно́го графика

Чтобы отслеживать результаты работы, добавим на лист временной график, на котором отразим изменение количества клиентов со временем. Для этого достаточно в графике добавить одну серию данных, связанную с переменной *clients* (рис. 46).

Как ведет себя система при наших значениях по умолчанию? Запустим ее и обнаружим неприятную деталь (рис. 47). Как только пропускная способность начинает расти, вентиль «увеличения» начинает ее уменьшать. Очевидно, портить нашу сеть мы не будем. Изменим формулу вычисления дефекта на следующую: `max(averageLoad-standard,0)`, т. е. не расти она может, но уменьшаться не должна. Повторим прогон (рис. 48).

Изначально все неплохо: за 2 месяца количество клиентов возрастает примерно на 17%. Но потом происходит следующее: через 10 недель клиенты обнаруживают, что сети на них явно не хватает, и начинают «голосовать ногами». Несмотря на то что через 30 недель мы начинаем модернизацию, ситуация не улучшается до

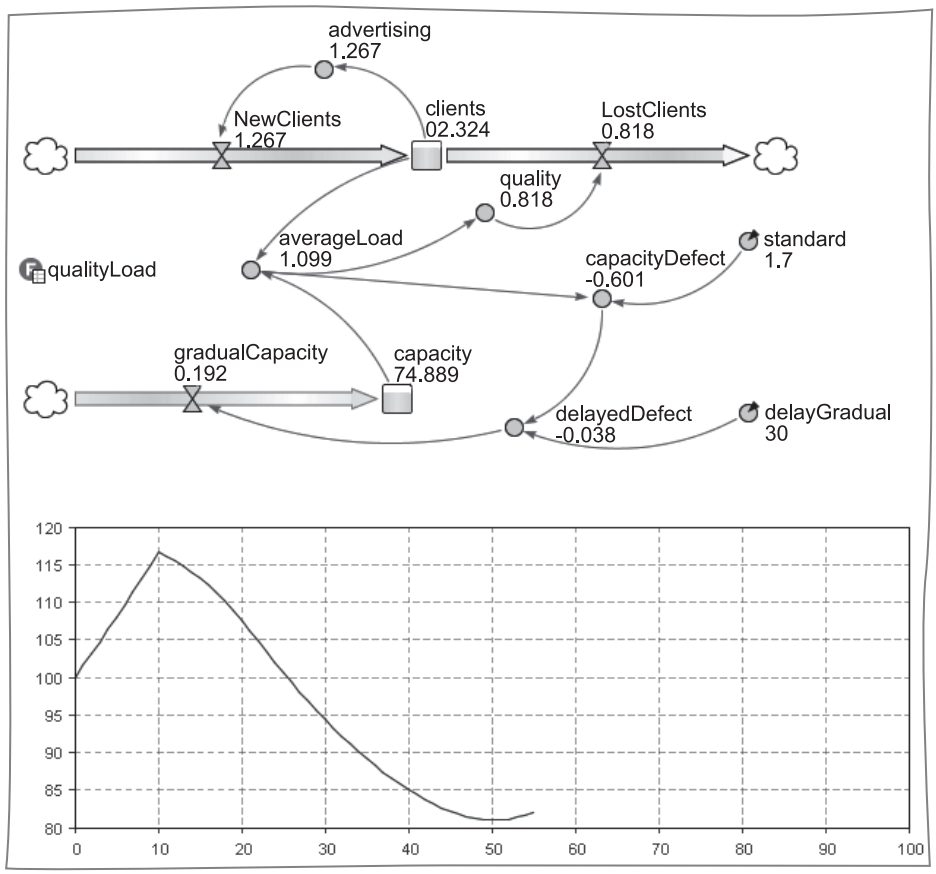

Рис. 47. Поведение системы при значениях параметров по умолчанию

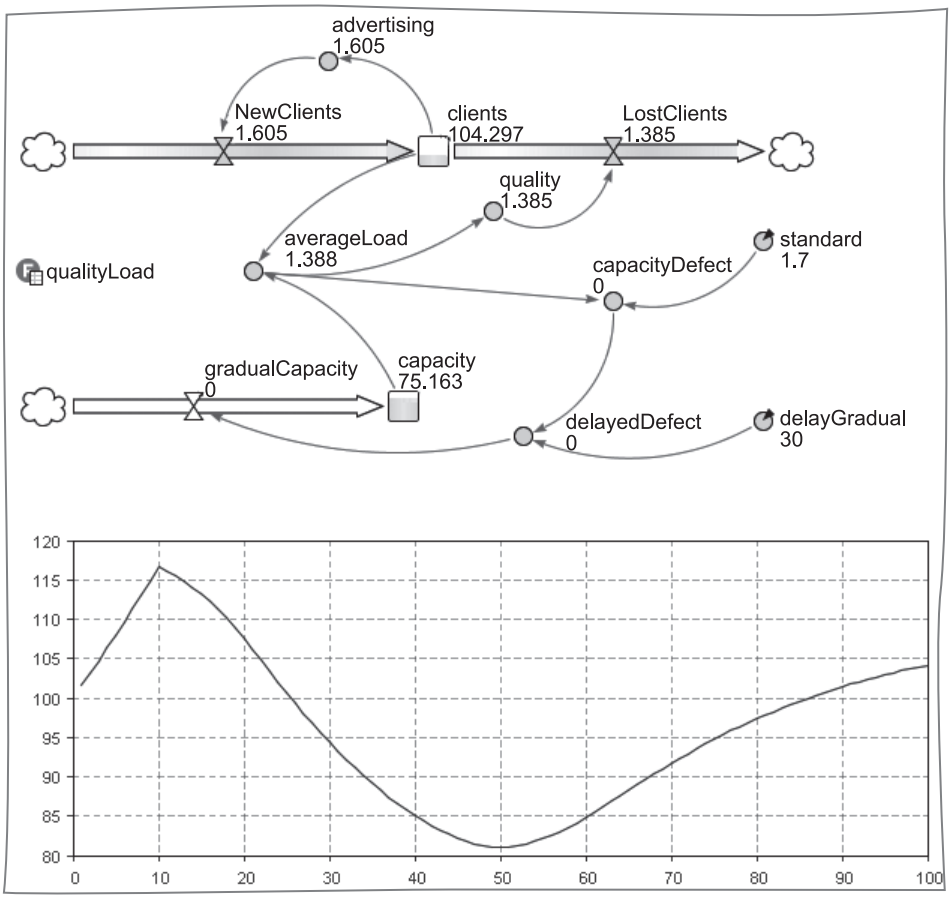

Рис. 48. Результат прогона при изменении формулы вычисления дефекта

50-й недели — к этому моменту мы теряем 17% от изначального количества клиентов. Ситуация исправляется только к 85-й неделе, т. е. более чем через год.

Очевидно, что с таким подходом к своей сети компания вряд ли обеспечит себе прибыль.

Задания

1. Изучите влияние имеющихся параметров на систему, добейтесь улучшения ситуации.
2. Почему мы не предлагаем варьировать в модели «рекламную мощность» и время задержки реакции клиентов?
3. Какой еще параметр можно добавить для управления системой, не меняя ее основной структуры?

Проект

«Алгоритмы и программы»

Знакомство со средой программирования

Как уже говорилось в учебнике, в подавляющем большинстве случаев разработка программ выполняется с помощью интегрированной среды — специального средства, в котором объединены (как минимум):

1. Текстовый редактор. В современных системах текстовый редактор не только позволяет подготовить текст программы, но и существенно помогает при подготовке, выделяя ключевые слова, выдавая подсказки, обеспечивая работу с несколькими файлами и т. д.
2. Транслятор: компилятор или интерпретатор, который переводит готовый текст в машинный код.
3. Отладчик. Средства, позволяющие проверить исполнение программы по шагам, проверить значения переменных во время исполнения, выявить ошибки и неудачно реализованные части.
4. Компоновщик и библиотеки функций, позволяющие решать типовые задачи (например, взаимодействия с внешней средой, чаще всего с операционной системой).

В современной среде могут присутствовать и другие инструменты, но перечисленные компоненты есть обязательно.

Из огромного количества существующих сред нужную выбирают по самым разным критериям. В нашем случае требуется среда:

1) использующая язык Pascal (или основанный на нем);
2) свободно распространяемая;
3) работающая под Windows и обеспечивающая доступ к ее средствам;
4) с документацией на русском языке.

Какие-то серьезные средства управления крупными проектами, подготовка специфических компонентов и прочие дополнительные возможности нам не потребуются.

Воспользуемся для решения имеющихся задач средой PascalABC.NET. Эта среда предназначена для обучения программированию и свободно доступна по адресу http://pascalabc.net/.

Немаловажным дополнением является то, что он опирается и активно использует платформу .Net от Microsoft, что позволяет получить доступ к самым разным средствам: от управления операционной системой до работы с различными графическими средствами (например, 3D-графикой).

Познакомимся со средой и заодно повторим базовые приемы и средства программирования.

Интерфейс среды

Установка PascalABC.NET не представляет из себя ничего необычного, поэтому останавливаться на ней мы не будем.

После запуска вы увидите окно, как на рис. 49.

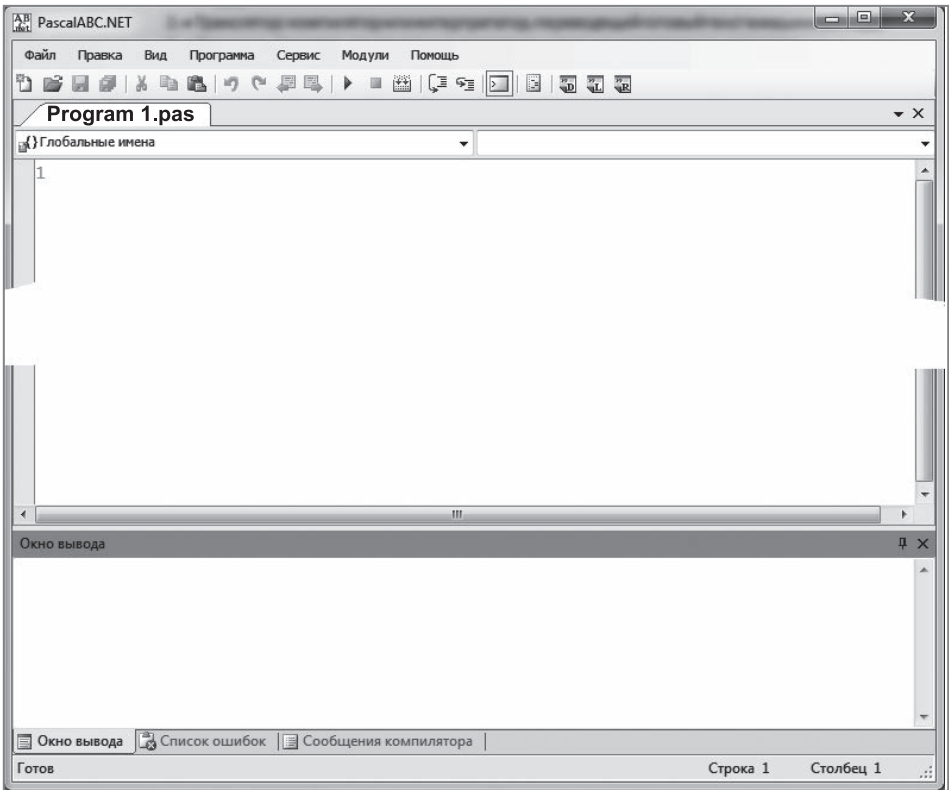

Рис. 49. Интерфейс среды PascalABC.NET

Интерфейс достаточно прост: в верхней части основное место занимает текстовый редактор, который позволяет одновременно работать с несколькими файлами. Выбрать файл можно наверху с помощью закладки. Сейчас открыт один файл (пустой) под названием Program1.pas.

Наберем в окне сверху первую программу (рис. 50). Учитывая уровень ее сложности, текст подробно разбирать не будем.

Program 1.pas*

{} Глобальные имена

```
1  program quadraticEquation;
2
3  var
4      a,b,c,discr,x1,x2 : double;
5
6  begin
7     write('Введите к-т A:');
8     readln(a);
9     write('Введите к-т B:');
10    readln(b);
11    write('Введите к-т C:');
12    readln(c);
13    discr := b*b - 4*a*c;
14    if discr < 0 then
15    begin
16       x1 := (-b-sqrt(discr))/(2*a);
17       x2 := (-b+sqrt(discr))/(2*a);
18       write ( 'Первый корень:',x1,' Второй корень:'x2 );
19    else
20       writeln ('Нет действительных корней');
21 end.
```

Рис. 50. Текст первой программы

Ошибки в 14-й, 18-й и 19-й строках допущены намеренно.

Попробуем запустить эту программу. Для этого либо на панели нажмем кнопку **Выполнить**, либо в меню **Программа** выберем команду **Выполнить**. Если бы ошибок не было, программа после компиляции начала бы исполняться, но они есть (рис. 51).

Из этой ошибки следует, что у нас нарушена структура строки: действительно, пропущена запятая. Исправляем, пытаемся запустить снова (рис. 52).

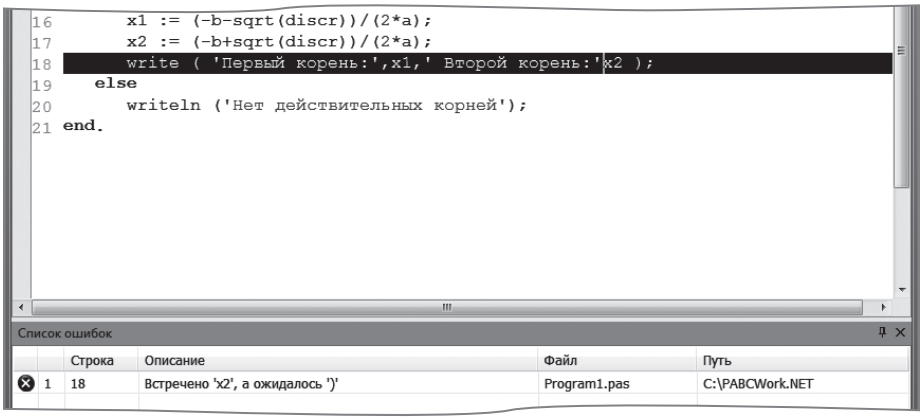

Рис. 51. Ошибка в программе в результате нарушения структуры строки

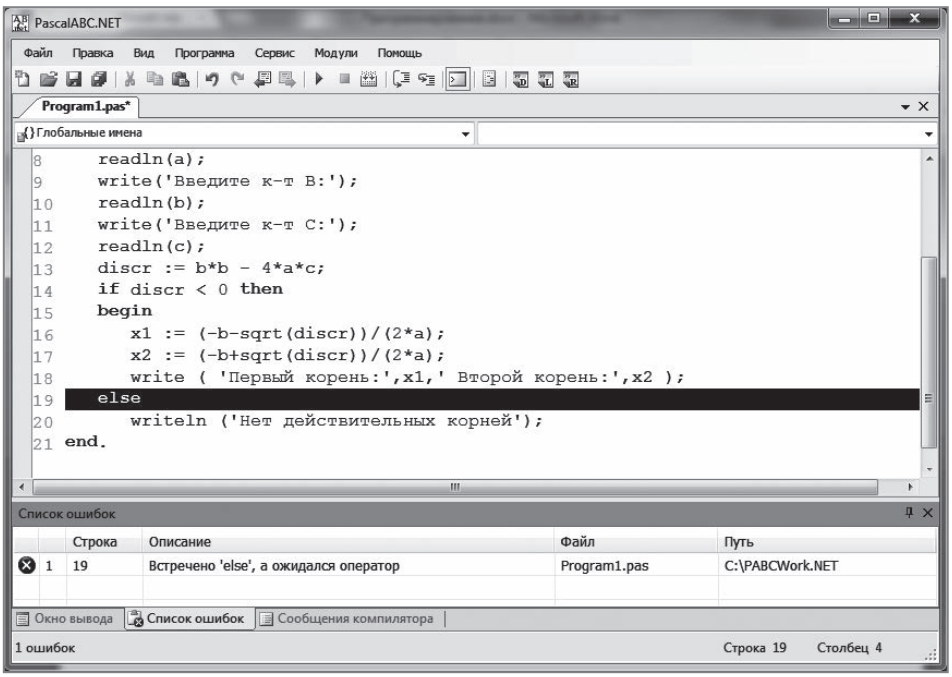

Рис. 52. Ошибка в программе в результате нарушения структуры программы

Теперь следующая ошибка: нарушена структура самой программы. Мы забыли закрыть операторные скобки: begin есть, end — нет. Поскольку программа написана «лесенкой», мы сразу видим нарушенный блок. Исправляем, запускаем (рис. 53).

Программа исполняется и запрашивает у нас данные. Мы работаем в учебной среде, поэтому не видим классического экрана-

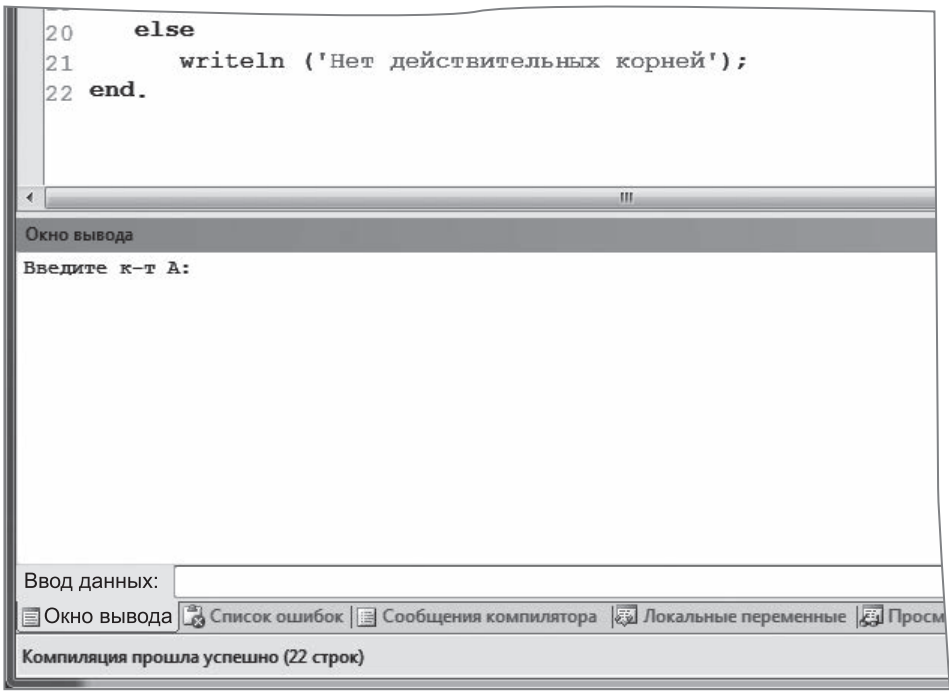

Рис. 53. Программа исполняется и запрашивает данные

консоли и данные вводим в нижней часть закладки «Окно вывода» (рис. 54).

Вводим данные, получаем результат, но, к сожалению, неверный.

> Обратите внимание! Мы не могли запустить программу до исправления всех ошибок. Классические правила работы компилятора Pascal таковы, что ошибки мы должны устранять одну за другой, поскольку нельзя увидеть весь список допущенных ошибок.

Разумеется, компилятор и компоновщик могут выявить только формальные, синтаксические ошибки. Логические ошибки, допущенные вами, в подавляющем большинстве случаев никто, кроме грамотного программиста, выявить и устранить не сможет. Это и произошло: в строке 14 есть ошибка — перепутан знак сравнения. На самом деле условие должно выглядеть как `discr >= 0`.

Тем не менее среда имеет в своем составе инструменты, предназначенные для выявления таких ошибок. Это средства отладки программ: пошаговый отладчик, точки останова и средства просмотра значений переменных. Разберем эти средства.

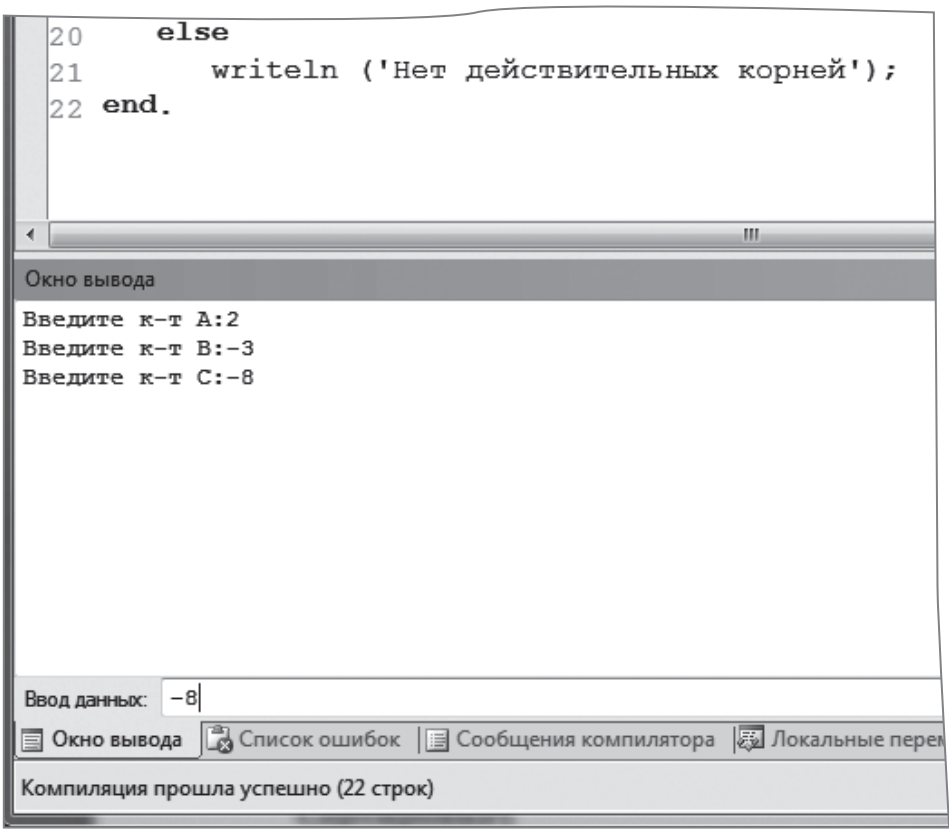

Рис. 54. Ввод данных

Начнем с основного: *пошагового отладчика*. Это средство позволяет нам проверить исполнение программы по шагам, контролируя значения переменных и последовательность исполнения.

Чтобы запустить такую отладку, нужно вместо запуска программы (клавишей F9) выполнить шаг (клавиша F8 или F7). Клавиша выбирается в зависимости от того, хотите вы попасть внутрь подпрограммы — процедуры или функции (F7) или нет (F8). Рабочее окно в таком режиме изображено на рис. 55.

Оператор, ожидающий исполнения, выделен на экране желтым. При нажатии на F8 он выполняется, и система переходит к следующему шагу; при нажатии на F7 мы попадаем внутрь функции newNode и можем проверить ее работу.

Исполняя программу по шагам, можно проследить, как она исполняется, например выяснить, правильно ли работают условия, циклы, функции. Так можно выявить конкретные места возникновения ошибок. Пройдя таким образом до места проверки дискриминанта, мы обнаружили бы, что условие срабатывает не так,

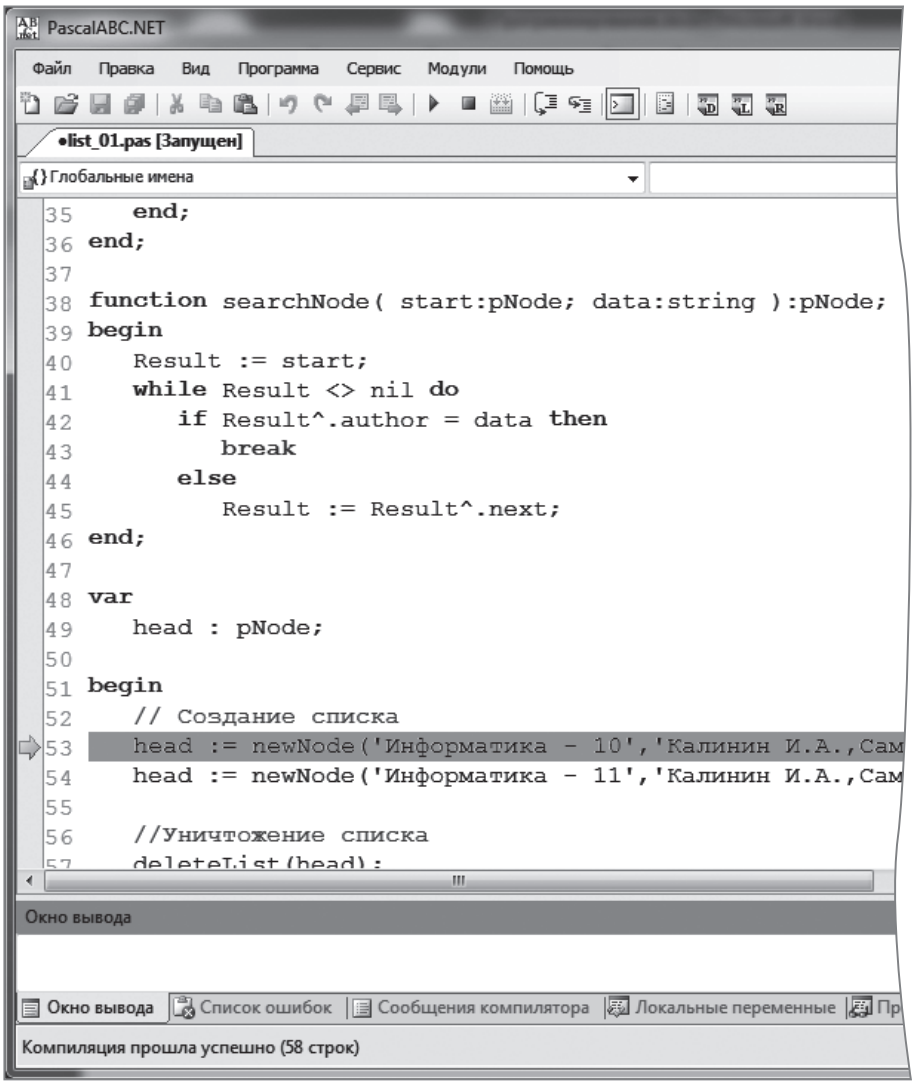

Рис. 55. Выбор клавиши для выполнения шага

как надо. Стоит обратить внимание, что выявить это можно при заранее выбранном наборе исходных данных.

Конечно, если программа велика, проходить ее всю до места отладки очень затруднительно. Чтобы не делать этого вручную, можно поставить точку прерывания исполнения. Для этого нужно на той строке, с которой вы хотите начать пошаговую проверку, при редактировании щелкнуть на полях (рис. 56).

Теперь можно просто запустить программу. Когда она дойдет до этого места, исполнение остановится и система будет ждать вашей команды.

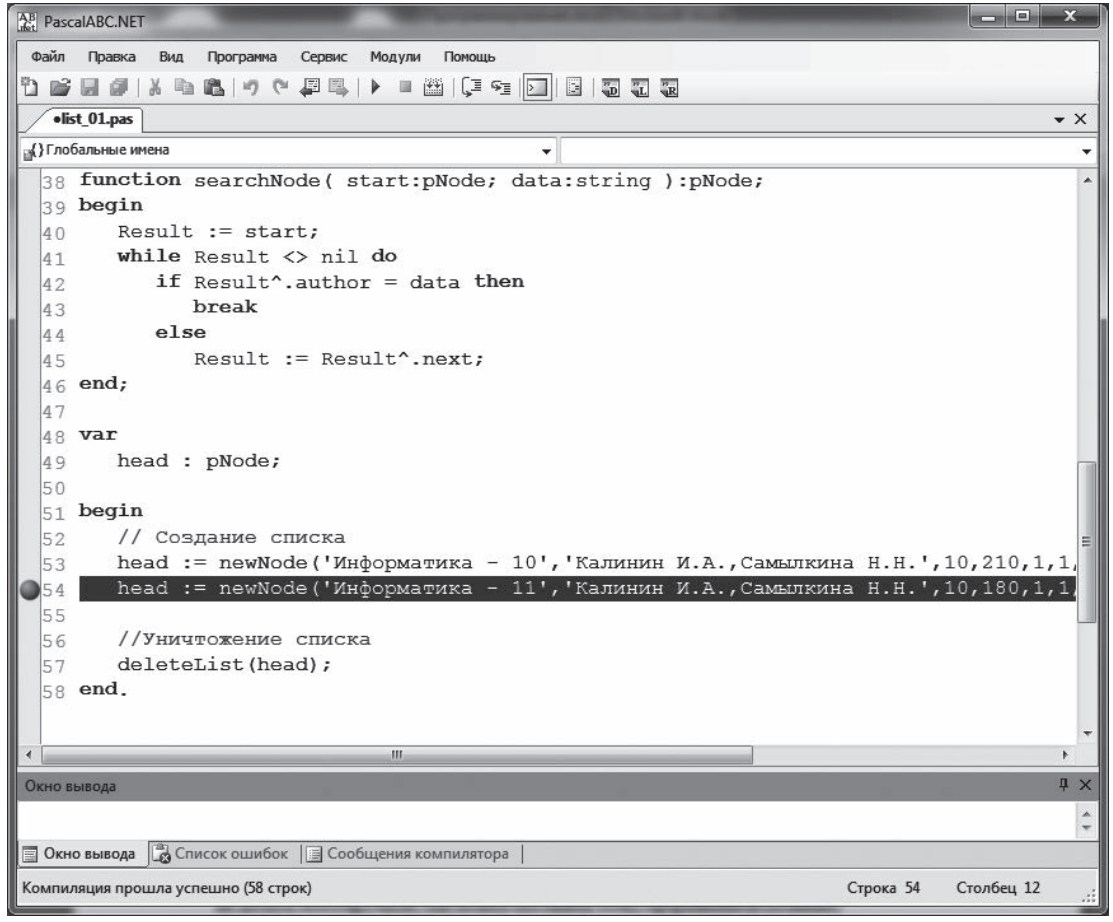

Рис. 56. Указание строки, с которой нужно начать пошаговую проверку

Структуры данных

Если количество строк программы больше 10, то, скорее всего, вам пришлось задуматься о том, как хранить данные для работы. Если данных мало, достаточно объявить несколько переменных.

Если данных много, причем заранее неизвестно сколько, задача существенно усложняется. Чтобы использовать циклические структуры (не описывать же все переборы сравнением всех переменных!), ускорить работу по поиску и добавлению данных, оптимизировать использование памяти, применяют конструкции, получившие название ***структур данных***.

Структура данных — способ организации связанного набора данных в памяти, позволяющий их хранить и обрабатывать.

Из сказанного следует, что любая структура реализуется с помощью программного кода, который позволит:

1) добавлять новые данные, т. е. записывать их в память, сохраняя общую логику;
2) получать доступ к уже имеющимся данным, например, для изменения;
3) искать нужные данные для каких-то операций;
4) удалять ненужные данные.

Задачи эти, конечно, связаны. Например, чтобы изменить данные, их сначала надо найти. Универсального способа хранить данные не существует, в каждой программе вы все равно будете выбирать оптимальный способ, исходя из задачи, имеющихся ресурсов, средств и знаний. Тем не менее в подавляющем большинстве случаев для построения своего способа используют типовые средства и решения, которые мы и разберем.

Массив

Простейшая структура данных, которая вам уже, возможно, известна, — ***массив***. В языке Pascal или C — это множество пронумерованных однотипных элементов. У массива масса достоинств: он прост в объявлении, понятен, обеспечивает доступ к любому своему элементу, сохраняется и считывается с диска.

В языке Pascal массив объявляется с помощью ключевого слова array:

```
var
   temperature : array[1..100] of double;
```

Этот способ прост, но ограничивает наши возможности и не является рекомендованным. Лучше для всех сложных конструкций объявить специальный тип данных, а потом использовать его в секции переменных:

```
type
   tempBase : array[1..100] of double;
var
   temperature : tempBase;
```

Такой конструкцией мы объявим массив под названием *temperature* из 100 переменных типа *double*, пронумерованных от 1 до 100. К каждой отдельной переменной (элементу массива) можно получить доступ по ее номеру-индексу:

```
temperature[10] := 5.3;
```

Массив очень удобен для организации циклических операций ввода, вывода, обработки.

Однако массив поможет вам, только если вы точно знаете, сколько данных у вас будет (хотя бы для того, чтобы оценить размер массива). Вторая его неприятная особенность: он должен храниться в памяти одним большим разделом, разбить его на отдельные части нельзя, и увеличение его, если и возможно, будет занимать много времени. В массив не всегда просто добавить или удалить элемент.

Современные языки часто позволяют реализовать так называемый динамический массив, т. е. массив, размеры которого можно изменять во время работы программы. Стоит учитывать, что такая операция может занять немало времени.

Чуть более сложная конструкция, объединяющая несколько переменных, — запись (она же структура). Запись позволяет объединить разнотипные элементы и обращаться к ним по именам.

Чтобы пользоваться записями, сначала объявляют тип:

```
Type
  Book = record
   title : string;
   author: string;
   count : word;
   price : longword;
   place1 : word;
   place2 : word;
   place3 : word;
  end;
```

а потом, в секции `Var`, переменную этого типа:

```
var
   myBook  :  Book;
```

К элементам записи можно обращаться таким образом:

```
   myBook.author  :=  'М.Дример';
```

Типы массива и записи вполне можно комбинировать, создавая, например, массив записей (таблицу). Эту возможность мы рассмотрим в рамках решения задачи-примера. Цель этого примера не столько изучить структуру данных как таковую, сколько показать, какие операции нам вообще потребуются.

Задача 1

Постановка задачи: требуется написать программу хранения и поиска данных о книгах в книжном магазине. О каждой книге нам известны: название, автор, количество, цена, номер зала, номер шкафа и номер полки. Известно, что памяти достаточно, а книг не может быть больше 1000 наименований.

Решение:

Сначала создадим структуру для хранения данных. Элемент данных (книга) описывается не одним числом или строкой, а целым набором отдельных переменных. Мы объединяем их в запись, описав сложный тип в разделе Type:

```
Type
  book = record
   title : string;
   author: string;
   count : word;
   price : longword;
   place1 : word;
   place2 : word;
   place3 : word;
  end;
  bookStore : array[1..1000] of book;
```

После этого можем объявить массив записей:

```
Var
   store : bookStore;
   storeCount : word;
```

Обратите внимание! Кроме самого массива нам требуется переменная, с помощью которой мы будем определять первую свободную запись массива. Чтобы не усложнять хранение, будем считать, что это счетчик количества книг. Это усложнит удаление записей, но упростит объяснения.

Добавление новой записи пояснений не требует: вносим данные и переходим к следующей свободной записи:

```
procedure newArrayNode (title : string; author: string;
                count : word; price : longword;
                place1 : word; place2 : word;
                place3 : word, var store : bookStore);
begin
      store[storeCount].title := title;
      store[storeCount].author:= author;
      store[storeCount].count:= count;
      store[storeCount].price:= price;
      store[storeCount].place1:= place1;
      store[storeCount].place2:= place2;
      store[storeCount].place3:= place3;
      inc(storeCount);
end;
```

Обратите внимание! Массив мы передаем с ключевым словом var, чтобы изменять его, не теряя изменений при возврате.

Процедура печати записи:

```
procedure itemArrayPrint (pos : word; var store :
                  bookStore);
begin
   write('|',store[pos].title, '|');
   write(store[pos].author, '|');
   write(store[pos].count, '|');
   write(store[pos].price, '|');
   write(store[pos].place1, '|');
   write(store[pos].place2, '|');
   writeln(store[pos].place3, '|');
end;
```

Поиск записей, соответствующих условию, — это просто циклический перебор:

```
function findRecord(author:string;
                    startSearchPos:word;
                    var store : bookStore):word;
var
   current : word;
begin
```

```
    current := startSearchPos;
    while (store[current].author <> author)
          and (current < storeCount)
    do
          inc(current);
end;
```

Здесь мы предполагаем, что фактическое количество записей в массиве хранится в глобально доступной переменной. При необходимости (и это было бы правильнее) мы можем передавать и эту величину.

Если нам понадобится найти все записи (например, для печати), можно несколько раз вызвать эту функцию, меняя стартовую позицию. Например:

```
procedure printAuthorBooks(author : string;
                           var store : bookStore);
var
    pos : word;
begin
    pos := findRecord(author, 1,store);
    while pos < storeCount do
    begin
        itemArrayPrint(pos);
        pos := findRecord(author, pos+1,store)
    end;
end;
```

Удаление записи из массива потребует перестановки всех записей от удаляемой «вверх», т. е. к началу:

```
procedure deleteArrayItem(del : word; var store :
                          bookStore);
var
  pos : word;
begin
   for pos := del to storeCount-2 do
   begin
        store[pos].title := store[pos+1].title;
        store[pos].author:= store[pos+1].author;
        store[pos].count:= store[pos+1].count;
        store[pos].price:= store[pos+1].price;
        store[pos].place1:= store[pos+1].place1;
        store[pos].place2:= store[pos+1].place2;
```

```
            store[pos].place3:= store[pos+1].place3;
        end;
        dec(storeCount);
    end;
```

Обратите внимание! Любая операция, требующая поиска, имеет сложность как минимум O(N), т. е. линейно растущую с длиной массива. А если эта операция еще и повторяется, зависимость становится и более «тяжелой». Пример такой операции — вставка элемента с предварительной проверкой.

Работа с файлами

Каждый раз вводить данные заново (особенно во время отладки) — не самый лучший способ. Как всем известно, если нужно сохранить данные на достаточно долгий срок, используются файлы.

Для реализации работы с файлами в языке Pascal предусмотрен набор функций, который мы сейчас используем для реализации функций записи и чтения данных. Pascal имеет средства для работы с файлами двух основных типов: двоичных и текстовых.

В *текстовых* файлах данные хранятся в виде текста (именно текста — строго букв и цифр, например число 1 будет записано байтом со значением 49) и разделяются на строки — парой символов № 10 и № 13 (подробнее см. в гл. 6 учебника).

Двоичные данные хранят любые данные так, как они хранятся в памяти компьютера. В свою очередь двоичные файлы могут быть *типизированными* (т. е. указано, из какого типа элементов состоит файл) и *бестиповыми* — в виде потока байтов, которые программа должна анализировать самостоятельно.

Мы для работы будем использовать самый удобный тип — типизированный двоичный файл, состоящий из записей объявленного нами типа.

Перед началом работы нужно объявить переменную, которая будет указывать на файл. При ее описании следует указать, какого типа данные хранятся в файле. Переменная описывается так:

```
Var
    bookStoreData : file of book;
```

Перед началом работы с файлом необходимо связать переменную с конкретным файлом на диске:

```
Assign(bookStoreData,'books.dat');
```

Файл будет открыт в том же каталоге, где будет запущена программа.

Далее файл необходимо открыть на запись:

```
Rewrite(bookStoreData);
```

После этого процедурой write можно записывать в файл данные:

```
write(bookStoreData, store[pos]);
```

Процедура сохранения данных на диске очень сильно напоминает все остальные процедуры — это просто перебор записей:

```
procedure saveArray( fname : string; var store :
                     bookStore);
var
   pos : word;
   bookStoreData : file of book;
begin
   Assign(bookStoreData,fname);
   Rewrite (bookStoreData);
   for pos := 0 to storeCount-1 do
        write(bookStoreData, store[pos]);
    Close(bookStoreData);
end;
```

Чтение данных из файла имеет ряд отличий. Во-первых, надо открыть файл не на запись, а на чтение путем замены процедуры Rewrite на Reset. Эта процедура откроет файл и поставит указатель чтения на начало файла.

Во-вторых, мы не знаем, сколько записей в файле, а поэтому читаем до конца. Конец мы определим с помощью логической функции eof (End Of File — конец файла).

Само чтение выполняется процедурой read.

```
procedure loadArray( fname : string; var store :
                     bookStore);
var
   pos : word;
   bookStoreData : file of book;
begin
   Assign(bookStoreData,fname);
   Reset (bookStoreData);
   pos := 0;
   while not eof(bookStoreData) do
```

```
    begin
          read(bookStoreData, store[pos]);
          inc(pos);
    end;
   Close(bookStoreData);
end;
```

Многие из этих функция требуют доработки, например:

1) при печати никак не учитывается, что данные имеют разную длину в разных записях — в результате не получится таблица;
2) при добавлении записи мы не проверяем, можно ли это сделать, ведь массив всего на 1000 единиц;
3) нет процедуры вставки данных (а не добавления);
4) нет ввода и вывода из файла;
5) собственно основной программы «ведения» этой миниатюрной базы данных мы так и не написали.

Так что, как видите, пространство для доработки — предостаточное.

Сортировка и поиск

Одна из проблем, которую обязательно приходится решать при работе с большими объемами данных, — организация поиска. От его эффективности напрямую зависит выполнение любых операций, связанных с выбором элемента.

До сих пор мы использовали только один способ поиска и отбора элементов — полный перебор. Этот способ прост для понимания, но приводит к огромным затратам времени, а иногда и памяти. Мы можем его ускорить, но при одном условии: если массив будет отсортирован.

Начнем с самой операции сортировки. В учебнике мы рассмотрели несколько способов сортировки массива. Реализуем самый быстрый из них Quick Sort:

```
function Partition(var A:sortArray; l,r : integer):
             integer;
var
  i,j,x,b : integer;
begin
  x := A[ l + (r-l)div 2];
  i := l-1;
  j := r+1;
```

```
    repeat
       repeat
          dec(j);
       until (A[j] <= x);
       repeat
                inc(i);
         until (A[i] >= x);
         if i < j then
         begin
                b := a[i];
                a[i] := a[j];
                a[j] := b;
          end;
    until i >= j;
    Partition := j;
end;

procedure QuickSort(var A:sortArray; l,r : integer);
var
   q : integer;
begin
   If r > l then
   begin
        q := Partition(A,l,r);
        QuickSort(a,l,q);
        QuickSort(a,q+1,r);
   end;
end;
```

Этот алгоритм сортировки мы подробно рассмотрели в учебнике, дадим только несколько комментариев.

Во-первых, в этом примере мы использовали условный тип sortArray, на месте которого должен быть ваш тип массива.

Во-вторых, при организации сравнения и перестановки элементов нам придется учесть то, что это записи. Не во всех случаях возможна простая перестановка присваиванием, поэтому стоит проверить, как это работает.

Теперь можно оптимизировать поиск. Идея бинарного поиска очень проста. Если у нас есть отсортированный (для определенности — по неубыванию) массив, то меньший элемент обязательно слева от большего.

Сравним искомый ключ с элементом в середине массива. Если ключ меньше, он обязательно слева, — будем проверять левую половину. Если больше, — обязательно справа, и проверять надо правую часть. Если он равен середине, мы нашли искомое.

На каждом таком шаге мы уменьшаем количество элементов для проверки в 2 раза. Будем повторять операцию выбора левой или правой части, пока либо найдем искомый элемент, либо выясним, что его нет. Поскольку алгоритм в учебнике уже разобрали, остается его только записать:

```
function binarySearch (key:integer; A:sortArray) :
                integer;
var
  middle,left,right : integer;
begin
  left:=1;
  right:=100;
  while left < right do
  begin
    middle := left + (right-left) div 2;
    if a[middle] > key then
      left := middle +1
    else
      if a[middle] <> key
        right := middle -1
      else
        break;
  end;
  if a[middle] = key then
     result := middle
  else
    result := -1;
end;
```

Ключевых условий применения этого алгоритма в общем два:

1) мы должны иметь возможность обратиться к каждому элементу последовательности по его номеру; это не всегда возможно, как мы увидим чуть позже;
2) последовательность должна быть отсортирована, причем направление сортировки будет очевидно влиять и на знаки сравнения в алгоритме.

Список

Более сложная, но и более гибкая структура хранения — список. Его ключевое отличие от массива — линейная связь элементов, т. е. прямого доступа к любому элементу у нас нет, но есть воз-

можность перейти к следующему по порядку и так добраться до нужного. Схематически это изображено на рис. 57.

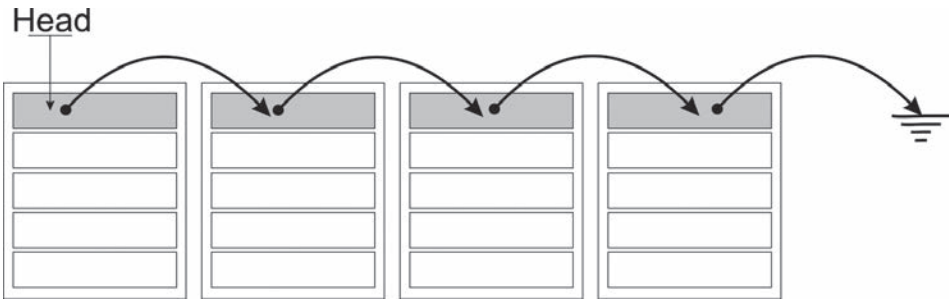

Рис. 57. Схематическое изображение списка

Задача 2

Постановка задачи: все тот же склад книжного магазина, но с условиями: неизвестно количество книг, объем памяти ограничен, заказчик недоволен временем вставки и удаления записей, особенно в самое начало массива.

Решение:

Сначала создадим структуру для хранения данных. Она очень похожа на предыдущую, но содержит добавление — ссылку на «следующий» элемент (специальный тип pNode).

```
type
  pNode = ^ListNode;
 ListNode = record
  next : pNode;
  bookData : book;
 end;
```

Обратите внимание! Как и в случае с массивом, для сохранения логики обработки мы используем тип book — запись о книге. Помимо прочего, это позволит нам хранить данные в файле, не меняя его формата.

Добавление записи:

```
function itemListNew (title : string; author: string;
                      count : word; price : longword;
                      place1 : word; place2 : word;
```

```
                        place3 : word; next :
                        PNode):pNode;
begin
   new (Result);
   Result^.title := title;
   Result^.author := author;
   Result^.count := count;
   Result^.price := price;
   Result^.place1 := place1;
   Result^.place2 := place2;
   Result^.place3 := place3;
   Result^.next := next;
end;
```

Добавление записи в список не сложнее, чем добавление записи в массив: сначала отводим память на новый элемент, вносим туда данные, а потом связываем его с предыдущим элементом. В основной программе нам потребуется хранить указатель только на «голову» списка.

Вот пример кода такой программы:

```
var
  head : pNode;

begin
   // Создание списка
  head := itemListNew ('Информатика - 10',
     'Калинин И.А.,Самылкина Н.Н.',10,210,1,1,1,nil);
  head := itemListNew ('Информатика - 11',
    'Калинин И.А.,Самылкина Н.Н.',10,180,1,1,1,head);
```

В этом примере мы создаем список, присваивая указатель на новый элемент головному указателю списка. Чтобы не терять уже добавленные элементы, передаем предыдущий указатель в качестве следующего.

Печать записи:

```
procedure itemListPrint ( item : pNode );
begin
   write('|', item^.title, '|');
   write(item^.author, '|');
   write(item^.count, '|');
   write(item^.price, '|');
   write(item^.place1, '|');
   write(item^.place2, '|');
```

```
   writeln(item^.place3, '|');
end;
```

Поиск записи, соответствующей условию:

```
function findListItem( author:string; startSearch:
                       pNode):pNode;
var
   current : pNode;
begin
   current := startSearch;
   while ( current <> nil) and (current^.author <>
                                author) do
         current := current^.next;
end;
```

В этом коде мы начинаем с какого-то стартового узла и переходим к следующему узлу, пока не найдем искомое или не дойдем до последнего элемента. Если искомое найти не удалось, то вместо указателя на элемент функция вернет значение nil.

Если требуется найти все вхождения (а не только одно первое), поступим так же, как и с массивом, т. е. будем перебирать, меняя начальный элемент, пока не дойдем до конца списка.

Удаление записи:

```
function itemListDelete(item : pNode) : pNode;
begin
   result := item^.next;
   Dispose ( item);
end;
```

На что стоит обратить внимание, так это на общий подход к обходу списка: мы начинаем с какого-то элемента и переходим к следующему, пока не сочтем нужным прерваться или не дойдем до конца.

Так, мы можем:

- вывести список (или некоторые его элементы, отобрав их по условию);
- удалить элементы;
- найти позицию и вставить элемент в список;
- сохранить элементы в файле;
- посчитать количество элементов и т. д.

Однако каждый раз писать проход по списку для разных целей представляется не самым оптимальным решением. Современные языки программирования позволяют решить эту задачу изящнее: с помощью *указателя на функцию*.

Сначала объявим нужный тип:

```
type
  iterate = function( item : pItem): pItem;
```

Напишем универсальную функцию перебора:

```
procedure iteratorList( start : pNode;
                        iterateFunction : iterate);
var
   current : pNode;
begin
   current := start;
   repeat
      current:=iterateFunction(current);
   until current = nil;
end;
```

Теперь можно написать функцию, например, печати всех элементов по условию:

```
function printListItem (item : pNode) : pNode;
begin
   if item^.author='Самылкина' then
      writeln(item^.author,',',item^.title,',',
            item^.price);
   result := item^.next;
end;
```

Вызов для перебора выглядит так:

```
  iteratorList(head,printListItem);
```

Задания

1. Напишите функцию, печатающую все элементы в виде таблицы.
2. Перепишите функцию освобождения списка с помощью функции-итератора.
3. Напишите функцию подсчета количества элементов в списке.
4. Напишите функции записи и чтения списка из файла по аналогии с такими функциями из раздела массивов.

5. Предложите доработку функции-итератора, которая позволит выполнять запись и чтение с ее помощью.

Бинарное дерево поиска

Рассмотренный нами алгоритм поиска применять к списку очень невыгодно из-за больших накладных расходов и при сортировке, и при переходах между элементами. Но существуют другие способы организации эффективного поиска, которые нам стоит рассмотреть. В их основе лежат *деревья*.

Простейший вариант дерева — бинарное дерево, разобранное нами в учебнике. Его основное достоинство — отсутствие необходимости вычислять позицию нужного элемента. То есть нам не нужна возможность перейти к любому элементу структуры — мы можем «дойти» до нее последовательно.

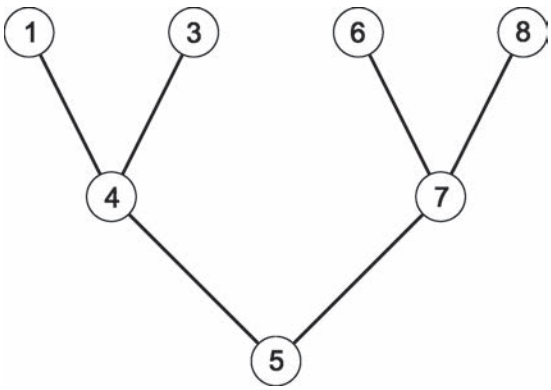

Рис. 58. Структура бинарного поискового дерева для набора чисел 1, 3, 4, 5, 6, 7, 8

На рисунке 58 схематически изображена структура бинарного поискового дерева для набора чисел 1, 3, 4, 5, 6, 7, 8. Стоит заметить, что, если мы будем добавлять в «дерево» число 2, оно окажется правой ветвью к узлу 1.

Структура элемента дерева похожа на структуру элемента списка:

```
type
  pNode = ^TreeNode;
  treeNode = record
    left, right: pNode;
    key: word;
end;
```

Элемент-узел дерева имеет три обязательные части: ключ (собственно данные) и минимум два указателя на другие узлы. Во время поиска, в зависимости от значения поискового ключа, можно идти либо по «левой», либо по «правой» ветви. Чтобы не путаться, будем считать, что элементы с меньшими значениями будут находиться слева, а с большими — справа.

По определению дерева, на каждый узел может сослаться как на «следующий» только один другой узел — во избежание появления циклов, т. е. замкнутых путей.

Создание элемента такого дерева выполняется функцией, очень похожей на добавление элемента в список:

```
function nodeTreeNew (key : word; left,right :
                      PNode):pNode;
begin
  new (Result);
   Result^.key := key;
   Result^.left := left;
   Result^.right := right;
end;
```

Но добавить правильно такой элемент в дерево можно, только найдя его место. Для этого нам потребуется функция поиска элемента в дереве:

```
function searchTree( root : pNode; key:word) : pNode;
var
  current : pNode;
begin
   current := root;
   while (( current <> nil )and(current^.key <> key) do
   begin
         result := current;
         if ( key > current^.key) then
               current := current^.rihgt
         else
               current := current^.left;
   end;
   if current <> nil then
         result := current;
end;
```

Функция вернет нам указатель на ближайший похожий элемент, если такой ключ уже есть в дереве, то на его узел.

Функция добавления ключа будет выглядеть так:

```
function insertTreeNode (key:word; root : pNode ):pNode;
var
   parentNode : pNode;
begin
   parentNode := searchTree(root,key);
   if ( parentNode^.key = key) then
         result := parentNode
   else
   begin
         if (parentNode^.key > key) then
         parentNode^.left := nodeTreeNew(key,nil,nil)
         else
         parentNode^.right := nodeTreeNew(key,nil,nil);
   end;
end;
```

Если ключ уже есть в дереве, мы ничего не добавляем, а возвращаем ссылку на него. Если его там еще нет, создаем его и присоединяем либо слева (если он меньше), либо справа.

Обход дерева. Печать элементов дерева, подсчет и другие операции, как и в случае со списком, удобно реализовывать с помощью указателей на функции.

Основной функцией для реализации у нас будет обход дерева. Выполнять его (в отличие от списка) можно по-разному:

1) сначала левое поддерево, потом правое, потом обработка самого узла;
2) сначала правое поддерево, потом обработка узла, потом левое;
3) сначала обработка узла, потом левое дерево, потом правое и т.д.

Для каких-то задач это неважно, а для каких-то очень важно. Напишем для примера первый вариант:

```
type
   iterate = function( item : pItem): pItem;

function iterateTree ( root : pNode;
                  iterateFunction : iterate):pNode;
var
   current : pNode;
begin
   current := start;
   repeat
```

```
    if current^.left <> nil then
          iterateTree(current^.left,iterateFunction);
    if current^.right <> nil then
          iterateTree(current^.right,iterateFunction);
      iterateFunction(current);
    until true;
end;
```

Обратите внимание! Мы использовали одно из свойств дерева — его часть это тоже дерево (по отношению к исходному — поддерево). Отразилось это в использовании приема, который называется *рекурсией*: мы вызвали функцию внутри ее самой. Стоит помнить, что прием этот очень мощный, но и опасный: программа может попасть в бесконечный цикл и очень быстро исчерпать ресурсы. В случае с деревом нас от этого защищает (при правильной реализации алгоритма) условие отсутствия внутренних циклов.

Задания

1. Напишите функцию удаления дерева, используя функцию-обходчик.
2. Напишите функцию подсчета количества элементов в дереве.
3. Напишите функцию подсчета узлов, у которых заполнена только одна ветвь.
4. Напишите итератор, с помощью которого можно вставлять элемент в дерево.

Хэширование

Деревья — эффективный способ организации поиска, но, как мы видели, он имеет целый ряд недостатков:

- есть вероятность, что мы не получим никакой выгоды, если дерево окажется несбалансированным;
- операция добавления с ростом дерева становится все длиннее, поскольку все больше элементов надо будет проверить;
- при достаточно большом объеме данных дерево станет очень большим, и поиск все равно окажется довольно продолжительным.

Один из вариантов решения такой задачи описан в учебнике — это использование хэш-функции.

Идея проста: напишем программу, которая по значению ключа рассчитает число «хэш». Число рассчитывается так, чтобы значения поискового ключа распределились среди всех возможных зна-

чений хэша равномерно. Тогда при поиске вместо сравнений мы сразу рассчитаем хэш и будем вести поиск только среди ключей с таким же значением хэша.

Реализуем такую задачу: у нас есть список слов русского языка. Требуется написать программу, которая быстро проверит, есть заданное слово в этом списке или нет.

Для решения используем хэш-функцию, которая описана в учебнике для строк. В первом варианте нашей программы проверим, действительно ли хэш-функция «разравнивает» ключи.

Зададим для функции количество хэш-значений — достаточно большое простое число 577:

```
program hashedSearch;
const
   NumHashLen = 577;
```

Для простоты поисковые ключи с одинаковым хэшем будем пока добавлять в список. Список реализуем точно так же, как уже рассматривали, поэтому здесь приведем только саму структуру:

```
type
  pHashNode = ^HashNode;
  HashNode = record
    val : string[40];
    next : ^HashNode;
  end;
```

Функцию вычисления хэш-значения по ключу (в нашем случае — строке) реализуем прямо по алгоритму, предложенному в учебнике:

```
function hashBKDR(S:string) : word; // Хэш-функция,
            предложенная Б.Керниганом и Д.Ритчи
var
   i : byte;
   h  : unit64;
begin
  h := 0;
  for i := 1 to length(s) do
    h := h*61 + ord(s[i]);
// 61 - Ближайшее простое число к количеству
// возможных символов русского и английского алфавитов
  hashBKDR := h mod NumHashLen;
end;
```

В основной программе заведем два массива: массив списков и массив количества значений. Для каждого прочитанного слова вычислим хэш, и он станет позицией-номером в массивах.

Данные (т. е. слова) мы прочитаем из текстового файла[4], в котором каждое слово записано на отдельной строке.

```
var
  SearchArray : array[0..NumHashLen] of pHashNode;
  CountArray : array[0..NumHashLen] of longword;
  txt : PABCSystem.Text;
  hs  :  word;
  n : pHashNode;
  s : string;
begin
   for hs:=0 to NumHashLen do
// Инициализация пустых массивов
   begin
     CountArray[hs] := 0;
     SearchArray[hs]:= nil;
   end;

   Assign(txt,'1grams-3.txt');
   Reset(txt);
   while not Eof(txt) do
   begin
     ReadLn(txt,s);// Читаем по одному слову
     if length(s) > 1 then
     begin
      hs := hashBKDR(s); // Вычисляем хэш
        // добавляем элемент в его список
      SearchArray[hs] := addNode( s, SearchArray[hs] );
        // увеличиваем количество таких элементов
     inc(CountArray[hs]);
    end;
  end;
  close(txt);
  // Вывод количества элементов для каждого
     значения хэш-функции
  For hs:=0 to NumHashLen do
  begin
      Writeln(hs,' - ',CountArray[hs]);
```

[4] Файл получен с сайта корпуса русского языка по адресу: http://www.ruscorpora.ru

```
        while SearchArray[hs] <> nil do
        begin
          n := SearchArray[hs]^.next;
          Dispose(SearchArray[hs]);
          SearchArray[hs] := n;
        end;
    end;
end.
```

Запустив эту программу, мы выясним, что все слова распределились по 577 значениям хэша примерно одинаково. Таким образом, за один шаг можно уменьшить количество элементов для поиска не в 2 раза (как в бинарном дереве), а в 577 раз.

К сожалению, второй шаг так просто не сделаешь — такой же хэш для этих ключей рассчитывать бесполезно. Но мы можем заменить хэш-функцию и рассчитать другие значения. Например, целый ряд таких функций приведен и проанализирован здесь: http://www.vak.ru/doku.php/proj/hash/efficiency

> Внимание! Для изучающих язык С рекомендуется использовать издание: Д. Г. Хохлов «Методы программирования на языке С: практикум в 2 частях». — М.: БИНОМ, 2014.
> http://www.lbz.ru/books

Проект

«Обработка статистических данных»

Расчет основных характеристик качества тестового инструментария

Статистические методы обработки данных очень широко распространены. Среди всего множества их применений можно указать как минимум одно, которое непосредственно касается лично вас, — это обработка результатов контрольных тестов. На статистике построена вся специальная дисциплина, посвященная созданию и применению тестов, — тестология. Именно с помощью этого инструментария готовятся и обрабатываются материалы ЕГЭ, так что вам будет полезно знать, как именно это делается.

В предлагаемом проекте вам предстоит проанализировать качество контрольных тестов по результатам их выполнения группой учащихся. При подготовке контрольного теста проверку на качество проходят как отдельные задания, так и тест в целом.

Мы не будем готовить тест, а проанализируем результаты его применения.

Первое, что нужно сделать, — оценить качество заданий, из которых тест составляется. Основные показатели качества тестовых заданий вводятся классической теорией тестов. К ним относят такие статистические характеристики, как трудность и дискриминативность (дифференцирующая способность).

Трудность определяется как доля учащихся, справившихся с заданием, и имеет точное определение: характеристика тестового задания, выраженная процентом верно выполнивших задание от количества испытуемых репрезентативной выборки. Поэтому для оценки качества самих заданий необходимо опробовать составленные тестовые задания на репрезентативной выборке учащихся. В обычных школьных условиях обеспечить репрезентативность невозможно, это задача специалистов. Но для обычных тематических тестов в этом нет необходимости. Достаточно обеспечить необходимый объем выборки. Следует помнить, что чем

больше выборка, тем достовернее данные. Заслуживают внимания статистические данные по заданию, которое выполняли не менее 250 тестируемых.

Необходимо правильно истолковывать понятие трудности. Рассмотрим ее значение на элементарном примере. Из 100 учащихся первое задание выполнили 30 учащихся, а второе — 60. Это означает, что второе задание менее трудное и его следует поставить в начале теста (если это тест для текущей проверки усвоения материала, т. е. обучающий). Некоторые специалисты пользуются обратной величиной (доля тех, кто с заданием не справился), называемой *индекс трудности*. В ЕГЭ используется просто процент выполнения от всех приступивших к выполнению. Значение трудности — величина условная, поскольку зависит от выборки. Для сильных и слабых групп это значение будет меняться. Значения трудности меньше 20 и больше 80 считают критическими, и задания с такими значениями трудности в тест стараются не включать.

Вторая характеристика качества заданий называется *дискриминативность,* или дифференцирующая способность. Эта характеристика определяется как способность отделять испытуемых с высоким общим баллом по тесту от тех, кто получил низкий балл, т. е. насколько точно задание дает возможность провести различие по определенному измеряемому признаку между экзаменуемыми с хорошей и не очень хорошей подготовкой. Задание, на которое одинаково хорошо могут ответить все экзаменуемые, не обладает хорошей дифференцирующей способностью, поскольку не дает никакой информации об относительных уровнях результатов. Самый простой и наглядный способ вычисления дискриминативности – это применение метода крайних групп (метод 27), когда при расчете учитываются результаты учащихся, наиболее и наименее успешно справившихся с тестом. При этом определяют:

$N_{\text{сильн}}$ — общее количество испытуемых в сильной группе;

$N_{1\text{сильн}}$ — количество учащихся в сильной группе, верно выполнивших задание;

$N_{\text{слаб}}$ — общее количество испытуемых в слабой группе;

$N_{1\text{слаб}}$ — количество учащихся в слабой группе, верно выполнивших данное задание.

Дискриминативность вычисляют как разность долей испытуемых из сильной (27%) и слабой (27%) групп, правильно выполнивших задание:

$$D = \frac{N_{1\text{сильн}}}{N_{\text{сильн}}} - \frac{N_{1\text{слаб}}}{N_{\text{слаб}}}.$$

Значение дискриминативности может изменяться от −1 до +1. Задание со значением, близким к 1, правильно разделяет учащихся и говорит о том, что большинство сильных учащихся справились с заданием, а слабым это не удалось. Нулевое значение говорит о том, что доли справившихся с заданием в сильной и слабой группах равны и задание нуждается в корректировке.

Причин низкой дискриминативности задания может быть много, например:

- излишняя сложность или запутанность формулировки;
- неоднозначно понимаемое условие;
- подсказка в условии;
- опора на память, а не на мыслительные навыки при выполнении задания;
- наличие двух или более правильных ответов;
- наличие «терминологической или логической ловушки» в условии или ответах.

Параметр не может указать на конкретную ошибку, но зафиксирует факт ее наличия.

На основании значения дискриминативности можно решить, что делать с каждым конкретным заданием: задание с отрицательным значением чаще всего требуется удалить либо существенно переработать. В тест должны попасть задания со значением дискриминативности выше 0,2.

Существуют и другие способы вычисления дискриминативности, которые используются в практике измерений. Рассмотренный нами способ — самый простой и вполне надежный.

После первичной апробации теста (на малой выборке испытуемых) его разработчик должен организовать выполнение этого теста большой представительной (репрезентативной) выборкой. Это делается для того, чтобы определить, как часто встречается тот или иной тестовый балл.

Накопив эти данные, можно оценивать уже не отдельные задания, а тест в целом и результаты его применения. Рассмотрим некоторые понятия и расчет основных статистических характеристик по тесту в целом (меры центральной тенденции, мода, дисперсия, стандартное отклонение, коэффициенты корреляции).

На основе анализа матрицы результатов тестирования поэтапно получим основные статистические характеристики по тесту в целом и интерпретируем их. Затем эту же матрицу используем для расчета дискриминативности каждого задания. Для упрощения работы в качестве наглядности будем использовать небольшую выборку 20 тестируемых и дихотомический способ оценивания выполнения каждого задания (1 — правильно выполнено,

0 — неправильно выполнено). Вначале сформируем таблицу — так называемую *матрицу результатов тестирования* (табл. 1). По вертикали матрицы располагаются профили (линейная матрица из 0 и 1) ответов тестируемых на каждое задание теста, по горизонтали — результаты выполнения каждым тестируемым заданий теста.

Введем обозначения:

i — номер тестируемого;

j — номер задания;

x_{ij} — результат выполнения i-м тестируемым j-го задания,

$$x_{ij} = \begin{cases} 1 \text{ при правильном ответе } i\text{-го тестируемого на } j\text{-е задание;} \\ 0 \text{ при неправильном ответе } i\text{-го тестируемого на } j\text{-е задание.} \end{cases}$$

R_j — количество правильных ответов для 20 заданий.

Первое, что мы сделаем, — очистим данные. Из таблицы 1 удаляются строки 11 и 12, поскольку они не несут необходимой информации. По профилям этих учеников получается, что данный тест не пригоден для истинной оценки уровня подготовленности этих тестируемых. Для 11-го этот тест слишком сложный, а для 12-го — слишком легкий. В реальной педагогической практике такая ситуация тоже может возникнуть. Безусловно, из обработки такие результаты исключаются.

Хороший тест для итогового контроля знаний и умений учащихся обладает приблизительно 70%-й точностью измерения результатов, находящихся в центре распределения. Результаты 3–5% учащихся, приходящиеся на края распределения, отбрасываются, так как отражают уровень подготовленности учащихся с очень большой ошибкой измерения. Удаляемые в таблице строки — это и есть крайние значения.

Далее простым суммированием считаются индивидуальные баллы тестируемых X_i и количество правильных ответов R_j.

Для графической интерпретации результатов тестирования необходимо упорядочить матрицу, располагая числа R_j в порядке убывания (по возможности), а значения X_i в порядке возрастания значений сверху вниз (табл. 2).

Для графического представления вначале оформим результаты в виде ранжированного ряда (табл. 3), а затем построим частотное распределение (табл. 4) в виде гистограммы (рис. 59). Сумма всех частот в группе должна равняться числу тестируемых учеников.

Таблица 1. Матрица результатов тестирования

Номер тестируемого i	Номер задания j										Индивидуальный балл X_i
	1	2	3	4	5	6	7	8	9	10	
1	1	1	1	1	1	1	0	0	0	0	**6**
2	1	1	0	0	0	0	0	0	0	0	**2**
3	0	0	0	0	0	0	0	1	0	0	**1**
4	1	1	0	1	1	1	1	1	1	1	**9**
5	1	0	1	1	1	1	0	0	0	0	**5**
6	1	1	1	0	0	0	0	1	0	0	**4**
7	1	1	1	1	0	1	0	0	0	0	**5**
8	1	1	1	1	0	0	0	0	0	0	**4**
9	1	1	1	1	1	1	1	1	1	0	**9**
10	1	1	1	1	1	0	1	0	0	0	**6**
11	0	0	0	0	0	0	0	0	0	0	**0 (удалить)**
12	1	1	1	1	1	1	1	1	1	1	**10 (удалить)**
13	0	1	1	0	0	0	0	0	1	1	**4**
14	1	0	1	1	0	1	1	1	1	0	**7**
15	0	1	1	0	1	1	1	0	0	0	**5**
16	1	0	1	1	1	1	0	0	0	0	**5**
17	1	1	1	1	1	1	1	0	1	0	**8**
18	1	1	1	1	1	0	1	0	0	0	**6**
19	1	0	0	0	0	0	0	0	0	0	**1**
20	1	1	1	0	1	0	0	1	0	0	**5**
$\mathbf{R}_j$	16	14	15	12	11	10	8	7	6	3	

Получилась гистограмма — последовательность столбцов, каждый из которых опирается на единичный интервал и имеет высоту, пропорциональную частоте наблюдаемых баллов. Гистограмма будет в большей мере похожа на нормальное распределение, если ее строить для сгруппированных данных (обычное количество таких групп 12–15). В нашем примере выборка очень маленькая, поэтому ограничимся тремя группами (табл. 5, рис. 60).

Таблица 2. Упорядоченная матрица результатов тестирования

Номер тестируемого i	**Номер задания j**										**Индивидуальный балл X_i**
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	
19	1	0	0	0	0	0	0	0	0	0	1
3	0	0	0	0	0	0	0	1	0	0	1
2	1	1	0	0	0	0	0	0	0	0	2
8	1	1	1	1	0	0	0	0	0	0	4
6	1	1	1	0	0	0	0	1	0	0	4
13	0	1	1	0	0	0	0	0	1	1	4
7	1	1	1	1	0	1	0	0	0	0	5
20	1	1	1	0	1	0	0	1	0	0	5
5	1	0	1	1	1	1	0	0	0	0	5
16	1	0	1	1	1	1	0	0	0	0	5
15	0	1	1	0	1	1	1	0	0	0	5
1	1	1	1	1	1	1	0	0	0	0	6
10	1	1	1	1	1	0	1	0	0	0	6
18	1	1	1	1	1	0	1	0	0	0	6
14	1	0	1	1	0	1	1	1	1	0	7
17	1	1	1	1	1	1	1	0	1	0	8
9	1	1	1	1	1	1	1	1	1	0	9
4	1	1	0	1	1	1	1	1	1	1	9
Число правильных ответов R_j для 20 заданий	15	13	14	11	10	9	7	6	5	2	

Таблица 3. Ранжированный ряд

Ранг	1	1	2	3	3	3	4	4	4	4	4	5	5	5	6	7	8	8
Номер	**19**	**3**	**2**	**8**	**6**	**13**	**7**	**20**	**5**	**16**	**15**	**1**	**10**	**18**	**14**	**17**	**9**	**4**
Балл	1	1	2	4	4	4	5	5	5	5	5	6	6	6	7	8	9	9

Таблица 4. Частотное распределение

Балл	1	2	4	5	6	7	8	9
Частота	2	1	3	5	3	1	1	2

Таблица 5. Сгруппированное частотное распределение

Интервал баллов	**Частота**
1–3	3
4–6	11
7–9	4

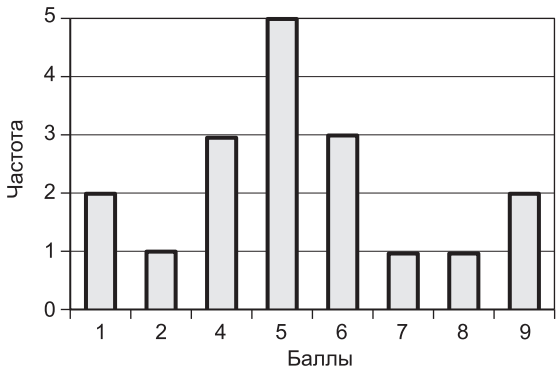

Рис. 59. Гистограмма частотного распределения баллов

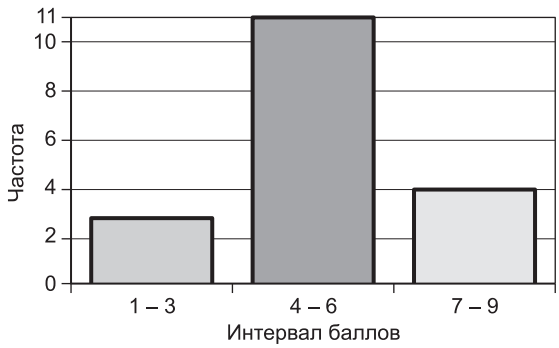

Рис. 60. Гистограмма частотного распределения сгруппированных баллов

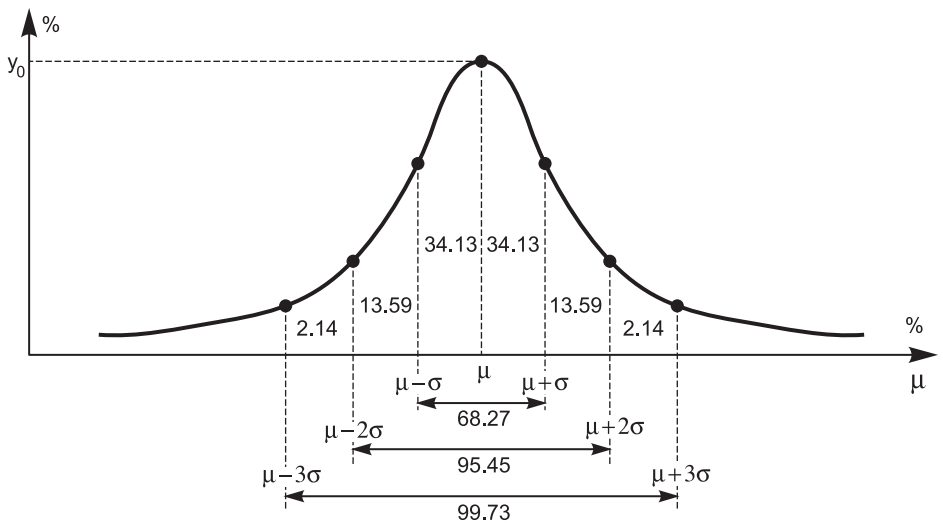

Рис. 61. Картина нормального распределения в идеальном варианте

На этой гистограмме вырисовывается картина нормального распределения, которое в идеальном варианте должно выглядеть, примерно как на рис. 61.

Визуально по полученной кривой распределения можно оценить асимметрию (обычно она рассчитывается, и на этом основании делается вывод о нормальности распределения). Нулевая асимметрия говорит о том, что тест сбалансирован по трудности заданий (см. рис. 61). Асимметрия распределения положительна, когда в тесте слишком много легких заданий (рис. 62), и отрицательна при избытке в тесте трудных заданий (рис. 63).

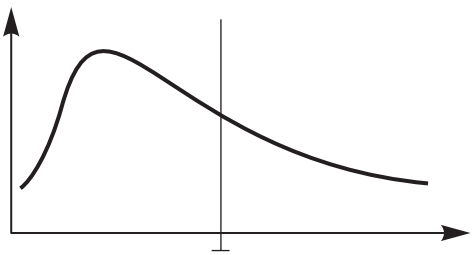

Рис. 62. Положительная асимметрия распределения

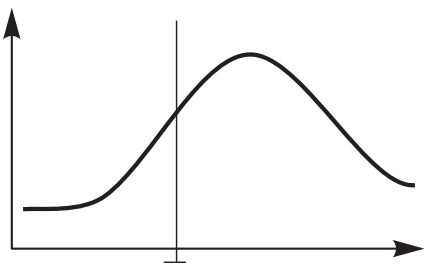

Рис. 63. Отрицательная асимметрия распределения

Определим некоторые другие характеристики.

Мода (μ) – наиболее часто встречающееся значение среди результатов выполнения теста. Для нашего случая модой является значение 5, поскольку оно встретилось чаще, чем другие значения (5 раз). Если получаются два значения моды, то распределение называют *бимодальным*. Нормальное распределение результатов должно быть унимодальным и симметричным. Бимодальное распределение говорит о неудачно построенном тесте, требующем внимательного анализа других характеристик для выявления причин неудачного построения. На рисунке 61 представлено нормальное распределение результатов для абстрактных данных, это идеальный вид кривой, достаточно редко встречающийся на практике. Но если распределение приближается к такой картине, то с определенной допустимой ошибкой измерения говорят о распределении по нормальному закону.

Хороший тест обеспечивает нормальное распределение индивидуальных баллов репрезентативной выборки тестируемых, если среднее значение баллов ($\bar{X}$) находится в центре распределения, а остальные концентрируются вокруг: примерно 70% в центре, остальные сходят до минимума по краям. *Смещение среднего значения влево говорит о слишком трудной подборке заданий те-*

ста, а смещение вправо — о слишком легкой подборке (см. выше асимметрию). Вычисляется *среднее* выборочное значение ($\overline{X}$) просто, поскольку это и есть среднее арифметическое индивидуальных баллов тестируемого:

$$\overline{X} = \frac{\sum_{i=1}^{N} X_i}{N}.$$

Среднее значение индивидуального балла важно не само по себе, а для анализа других описательных характеристик, позволяющих сравнивать различные распределения по тестам. Можно выявить различия в качестве тестов, сравнивая несколько распределений с одинаковым средним значением. Необходимо оценить, как разбросаны эмпирические данные вокруг среднего значения, сгруппированы тесно или, наоборот, сильно удалены. Для этого используют такие характеристики, как дисперсия и стандартное отклонение.

Дисперсия отражает меру неоднородности результатов по тесту и вычисляется по формуле:

$$S_x^2 = \frac{\sum_{i=1}^{n} (X_i - \overline{X})^2}{N-1}.$$

Можно подсчитать значение дисперсии вручную. Но на практике среднее значение чаще всего число дробное, подсчет вручную делать утомительно. Поэтому лучше написать небольшой фрагмент программы, которая будет проводить автоматический подсчет, если данная характеристика понадобится для дальнейшей обработки результатов:

```
program Dispersion;

const N = 18;

var
   balls: array[1..N] of word;
   m,d : real;
   i : Word;
   s : longint;
begin
   s:=0;
   for i:=1 to N do
   begin
```

```
        write('Введите балл ',i,'-го ученика:');
        readln(balls[i]);
        s := s + balls[i];
    end;
    m := s/N;
    d:=0;
    for i:=1 to N do
        d := d+(balls[i]-m)*(balls[i]-m);
    d := d/(N-1);
    Writeln('Дисперсия:',d);
end.
```

Результат работы для наших данных:

Дисперсия: 5.39869281045752.

В наших условиях его можно округлить до двух знаков: 5.40.

Если дисперсия — не единственная статистическая характеристика, которую вы вычисляете, может оказаться удобнее спланировать расчет в электронной таблице. Для Microsoft Excel достаточно ввести данные единым диапазоном и рассчитать параметры, вставив формулы.

Для нашего случая:

1	1	2	4	4	4	5	5	5	5	6	6	6	7	8	9	9	=СРЗНАЧ(A1:R1)	=ДИСП(A1:R1)

Результат вычисления, конечно, будет точно таким же.

Использование электронной таблицы может оказаться менее удобным, если объем данных будет большим, или вычисляемые характеристики не предусмотрены в стандартном пакете функций, или ресурсы ограничены.

Низкая дисперсия говорит о слабом разделении тестируемых по уровню подготовки, а излишне высокое значение — об искаженной картине распределения, а значит, проблемах в тесте.

Для анализа чаще используют *стандартное отклонение*, которое определяют как корень квадратный из дисперсии. Это значение, увеличенное в 3 раза, сравнивают со средним выборочным значением. Считается, что они должны быть приблизительно равны:

$$S_x = \sqrt{S_x^2}; \qquad \overline{X} = 3 \cdot S_x.$$

Существуют еще несколько характеристик для детального анализа результатов тестирования. Наибольшее внимание обращают на корреляцию (связь между исследуемыми объектами). Необходимо не только установить наличие связи, но и выбрать вид

и форму показателя для оценки этой связи. Это одна из характеристик, обеспечивающих *валидность* теста — соответствие теста целям контроля.

Для определения связи между различными наборами данных применяют *коэффициент корреляции Пирсона*. В случае необходимости определения связи между заданиями в одном тесте используют *преобразованный* коэффициент Пирсона, называемый «*коэффициент* φ»:

$$\varphi_{jl} = \frac{p_{jl} - p_j p_l}{\sqrt{p_j q_j \cdot p_l q_l}},$$

где p_{jl} — доля тестируемых, верно выполнивших оба задания, т.е. получивших по 1 баллу за оба задания; p_j – доля тестируемых, правильно выполнивших *j-е* задание; $q_j = 1 - p_j$; p_l – доля тестируемых, правильно выполнивших *l-е* задание; $q_l = 1 - p_l$.

Результаты подсчета коэффициентов корреляции между результатами по отдельным заданиям теста сводят в таблицу для удобства интерпретации. Коэффициенты корреляции для итоговых тестов должны быть в пределах [0; 0,3]. Поскольку итоговый предметный тест гомогенный, корреляция должна быть невысокой положительной. Высокое значение говорит о зависимости заданий друг от друга, что недопустимо в итоговом тесте. Отрицательные значения говорят об отсутствии предметной чистоты содержания теста. Такие задания, как правило, удаляются из теста. Для тематических тестов корреляция будет достаточно высокой, как и полагается. Результаты выполнения заданий тематического теста слабо варьируются, поскольку отражают исходное содержание.

Такой коэффициент корреляции используется в случае, если тест содержит задания одного типа, значит, и распределение задается в одной шкале. На практике в итоговом тесте используют задания разных типов. В случае, когда один набор значений распределения задается в дихотомической шкале, а другой — в интервальной, используют коэффициент бисериальной корреляции. В практической работе используют коэффициент точечно-бисериальной корреляции. Он проще в расчетах и обладает существенным преимуществом: его расчетное значение не выходит за границы интервала [−1; +1] в отличие от значений коэффициента бисериальной корреляции:

$$\left(r_{pbis}\right)_j = \frac{\left(\overline{X_1}\right)_j - \left(\overline{X_0}\right)_j}{S_x} \sqrt{\frac{(N_1)_j \cdot (N_0)_j}{N(N-1)}},$$

где $\left(\overline{X_1}\right)_j$ — среднее значение индивидуальных баллов тестируемых, выполнивших верно *j*-е задание теста; $\left(\overline{X_0}\right)_j$ — среднее значение индивидуальных баллов тестируемых, выполнивших *j*-е задание теста неверно; S_x – стандартное отклонение по множеству значений индивидуальных баллов; N – общее число тестируемых; $(N_1)_j$ – число тестируемых, верно выполнивших *j*-е задание теста; $(N_0)_j$ – число тестируемых, неверно выполнивших *j*-е задание теста.

Расчет коэффициента точечно-бисериальной корреляции в Excel будет выглядеть примерно, как на рис. 64.

Расчет выполняется в три этапа.

1. Рассчитывается суммарный балл для каждого ученика (например, =СУММ(B3:K3) — для первого ученика и т. д.) и стандартное отклонение (=СТАНДОТКЛОН(L3:L20) — ячейка L22) с помощью встроенных функций.

2. Рассчитывается средний балл выполнивших и не выполнивших каждое задание. Формула для подсчета среднего балла выполнивших задание выглядит в этом примере для первого задания так: =СУММЕСЛИ(B3:B20;1;$L$3:$L$20)/СЧЁТЕСЛИ(B3:B20;1), т. е. мы суммируем с условием и подсчитываем количество с условием. Обратите внимание, один из диапазонов зафиксирован.

3. Рассчитываем коэффициент корреляции. Для первого столбца — по такой формуле: =(B21-B22)*КОРЕНЬ(СЧЁТЕСЛИ(B3:B20;1)*СЧЁТЕСЛИ(B3:B20;0)/(18*17))/$L$22. Скопируем формулу для остальных столбцов.

Анализ значений точечно-бисериальной корреляции позволяет сделать вывод о валидности заданий, т. е. насколько задание пригодно для измерения в соответствии с общей целью построения теста. Если цель итогового тестирования — дифференцировать учащихся по уровню подготовки, то валидные задания должны четко разделять сильно и слабо подготовленных в тестируемой группе. Значение коэффициента должно приближаться к 0,5. Рекомендуется оценить разность $\left(\overline{X_1}\right)_j - \left(\overline{X_0}\right)_j$. Чем выше значение этой разности, тем лучше работает задание на дифференциацию учащихся.

Задания

Задача 1. Рассмотрим расчет дискриминативности (дифференцирующей способности) для 10 заданий на примере имеющихся результатов выполнения теста 18 учащимися. Напоминаем, что это слишком малый объем выборки для корректных выводов.

	A	B	C	D	E	F	G	H	I	J	K	L
1		Задание										
2	Ученик	1	2	3	4	5	6	7	8	9	10	Балл
3	1	1	0	0	0	0	0	0	0	0	0	1
4	2	0	0	0	0	0	0	0	1	0	0	1
5	3	1	1	0	0	0	0	0	0	0	0	2
6	4	1	1	1	1	0	0	0	0	0	0	4
7	5	1	1	1	0	0	0	0	1	0	0	4
8	6	0	1	1	0	0	0	0	0	1	1	4
9	7	1	1	1	1	0	1	0	0	0	0	5
10	8	1	1	1	0	1	0	0	1	0	0	5
11	9	1	0	1	1	1	1	0	0	0	0	5
12	10	1	0	1	1	1	1	0	0	0	0	5
13	11	0	1	1	0	1	1	1	0	0	0	5
14	12	1	1	1	1	1	1	0	0	0	0	6
15	13	1	1	1	1	1	0	1	0	0	0	6
16	14	1	1	1	1	1	0	1	0	0	0	6
17	15	1	0	1	1	0	1	1	1	1	0	7
18	16	1	1	1	1	1	1	1	0	1	0	8
19	17	1	1	1	1	1	1	1	1	1	0	9
20	18	1	1	0	1	1	1	1	1	1	1	9
21	Средний балл - выполнивших	5,47	5,62	5,64	6,36	6,4	6,56	7,14	5,83	7,4	6,5	
22	Средний балл - невыполнивших	3,33	3,8	3,25	3,14	3,5	3,67	3,82	4,75	4,23	4,94	2,323509
23												
24	К-т бисериальной корреляции:	0,35	0,36	0,44	0,7	0,64	0,64	0,72	0,23	0,63	0,22	
25												

Лист1 Лист2 Лист3

Рис. 64. Расчет коэффициента точечно-бисериальной корреляции в Excel

К сильной группе $N_{\text{сильн}}$ (см. табл. 3) можно отнести четырех учащихся, получивших 7, 8 и 9 баллов (27% от начала рейтинга), к слабой группе $N_{\text{слаб}}$ относятся учащиеся с 1, 2 и 4 баллами, таких — шесть (27% от конца рейтинга).

Воспользуемся приведенной ранее формулой $D = \frac{N_{1\text{сильн}}}{N_{\text{сильн}}} - \frac{N_{1\text{слаб}}}{N_{\text{слаб}}}$.

$D_1 = 4/4 - 4/6 = 1 - 0{,}66 = 0{,}34;$
$D_2 = 3/4 - 4/6 = 0{,}75 - 0{,}66 = 0{,}09;$
$D_3 = 3/4 - 3/6 = 0{,}75 - 0{,}5 = 0{,}25;$
$D_4 = 1 - 1/6 = 0{,}83;$
$D_5 = 3/4 - 0 = 0{,}75;$
$D_6, D_7 = 1 - 0 = 1;$
$D_8 = 3/4 - 2/6 = 0{,}75 - 0{,}33 = 0{,}42;$

$D_9 = 1 - 1/6 = 0{,}84;$
$D_{10} = 1/4 - 1/6 = 0{,}25 - 0{,}16 = 0{,}09.$

В итоговый тест могут быть включены задания с дискриминативностью более 0,2. Задания с нулевым или отрицательным значением дискриминативности следует удалить из теста, поскольку их формулировки некорректны. Чем ближе значение дискриминативности к единице, тем лучше задание разделяет тестируемых по уровню подготовленности.

В нашем тесте задания 2 и 10 следует существенно переработать.

Все перечисленные характеристики позволяют оценить качество отдельных заданий и весь тест, но не влияют на оценку тестируемого. Хотелось бы взять в качестве оценки тестируемого набранные баллы, но это делать нельзя, потому что в этом случае нельзя сравнивать данные разных тестов. Сравнивать же их необходимо, иначе как можно будет, например, провести конкурс аттестатов? Вариантов итогового теста должно быть очень много — как собрать полностью эквивалентные варианты? Это тоже непростая проблема, в настоящее время не разрешенная.

Поэтому нам нужен инструмент, который позволяет превратить «сырой» балл в «итоговый». Таким инструментом является *специальная шкала* — набранный «сырой» балл должен быть пересчитан в специальные абсолютные единицы. Конечно, в текущих тестах — обучающих или проверочных — никакого смысла в построении единой шкалы нет, все равно результаты сравниваться не будут, а вот в итоговых тестах, например в ЕГЭ, такой механизм необходим. Процедура перевода первичных, или «сырых», баллов в тестовый балл называется *шкалирование*.

Задача 2. Воспользуйтесь результатами выполнения теста (табл. 6) и рассчитайте основные характеристики заданий и теста в целом (трудность, дискриминативность, моду, дисперсию, стандартное отклонение, коэффициенты корреляции между результатами выполнения заданий и результатами отдельных учеников).

Постройте распределение и сделайте обоснованный вывод о качестве тестового материала. Для удобства проведения расчетов воспользуйтесь электронной таблицей.

Задача 3. Проанализируйте предложенное распределение участников экзамена по информатике по полученным первичным баллам в ходе ЕГЭ 2012 года (рис. 65) и ответьте на следующие вопросы:

Таблица 6

Номера вопросов																			
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	1	1
1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	0	0	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	1	1
0	1	1	0	1	1	1	1	1	1	1	0	1	1	1	0	0	0	0	1
0	1	1	0	1	0	1	1	1	1	1	1	1	1	1	0	1	1	0	1
1	0	1	0	1	1	0	0	1	1	0	0	1	1	0	0	0	1	0	1
0	0	1	1	1	0	1	0	1	1	0	1	1	1	0	0	1	1	0	0
1	0	0	1	1	1	1	0	1	1	0	0	1	0	0	0	0	1	0	0
1	1	1	1	0	1	0	0	1	1	1	1	1	1	0	0	0	0	1	0
1	1	0	1	1	0	1	1	1	1	1	1	1	1	0	0	0	0	1	0
0	0	1	1	0	0	0	1	1	1	1	0	1	1	0	0	0	0	1	1
0	1	0	1	1	0	1	1	1	0	1	1	1	1	0	0	1	0	1	1
1	0	1	1	0	1	1	1	1	0	1	1	1	1	0	0	0	0	1	1
1	1	1	1	1	0	0	1	1	0	1	1	1	1	0	0	0	0	1	1
1	1	1	1	1	1	1	1	1	1	1	0	1	1	0	0	0	0	1	1
1	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1
0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0
0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Примечание: 1 — правильный ответ, 0 — неправильный.

Таблица 6

Номера вопросов															Кол-во правильных ответов
21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	34
1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	32
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	32
1	1	1	1	1	1	1	1	1	1	1	0	1	0	1	30
1	1	1	1	1	0	1	1	1	0	0	1	1	1	0	24
1	1	1	1	1	1	1	1	1	0	1	1	0	1	1	28
1	1	1	1	1	1	1	1	1	0	0	0	1	1	1	22
0	1	1	1	1	0	1	1	1	1	1	1	0	1	1	23
0	0	1	1	1	0	1	1	1	1	1	1	0	1	1	20
0	0	1	1	1	0	1	1	1	0	1	1	1	1	0	22
0	0	1	1	1	1	1	1	1	0	0	0	1	1	1	23
0	0	1	1	1	1	1	1	1	0	1	1	1	0	0	20
1	1	1	1	1	0	1	1	1	0	1	1	1	0	0	24
1	1	1	1	1	0	1	1	1	0	1	1	1	1	1	26
1	1	1	1	1	0	1	1	1	0	0	1	1	1	1	25
1	1	1	1	1	1	1	1	1	1	0	1	0	1	1	28
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	32
0	0	0	0	0	1	0	0	1	0	0	0	1	0	0	5
0	0	0	0	0	1	0	0	1	0	0	0	1	0	0	5
0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	2
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

1. Можно ли считать данное распределение нормальным?
2. Какие выводы можно сделать по качеству тестового материала?
3. Можно ли вычленить основную проблему, которую необходимо разрешить?

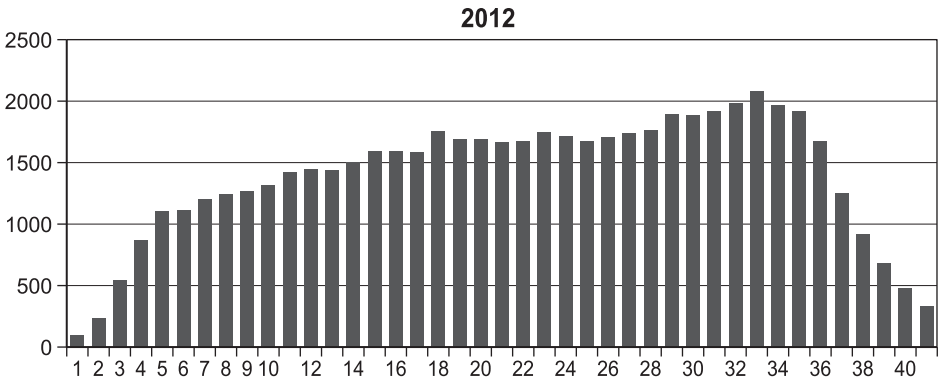

Рис. 65. Распределение участников экзамена по полученным первичным баллам

Проект

«Технология обработки текста»

Выделение последовательностей по шаблону

В этом проекте мы решим несколько практических задач, которые основаны на автоматической обработке текста. Поскольку задачи обработки текста очень разнообразны, мы не можем утверждать, что показанный нами инструментарий – самый лучший и единственно верный. Тем не менее у нас есть основания утверждать, что он достаточно мощный, чтобы решать с его помощью массу полезных задач.

Основным средством, которое мы будем использовать для обработки текста, будет уже знакомая вам реализация языка Pascal — PascalABC.Net. Напомним, что получить его можно по адресу http://pascalabc.net/ssyilki-dlya-skachivaniya.html.

Помимо рабочей среды языка, нам потребуется программа подготовки и тестирования регулярных выражений Expresso. Программа эта бесплатна, и ее можно скачать по адресу http://www.ultrapico.com/ExpressoDownload.htm.

В этом разделе мы будем опираться на возможности уже не столько самой системы, сколько на возможности дополнительных библиотек как стандартных для платформы .Net, так и дополнительно полученных. Для решения задач, связанных со сложной обработкой и анализом текста, нам потребуется библиотека реализации грамматического словаря: http://solarix.ru. Демонстрационный вариант этого комплекса бесплатен, и его возможностей нам более чем достаточно.

Задача поиска в тексте части, соответствующей шаблону (или для проверки, соответствует ли строка шаблону), возникает очень часто и в самых разных ситуациях. Основным способом ее решения, как мы уже говорили в учебнике, является механизм регулярных выражений.

Основным средством, которое мы будем использовать для создания и проверки регулярных выражений, будет программа Expresso. Начнем с краткого разбора ее интерфейса (рис. 66).

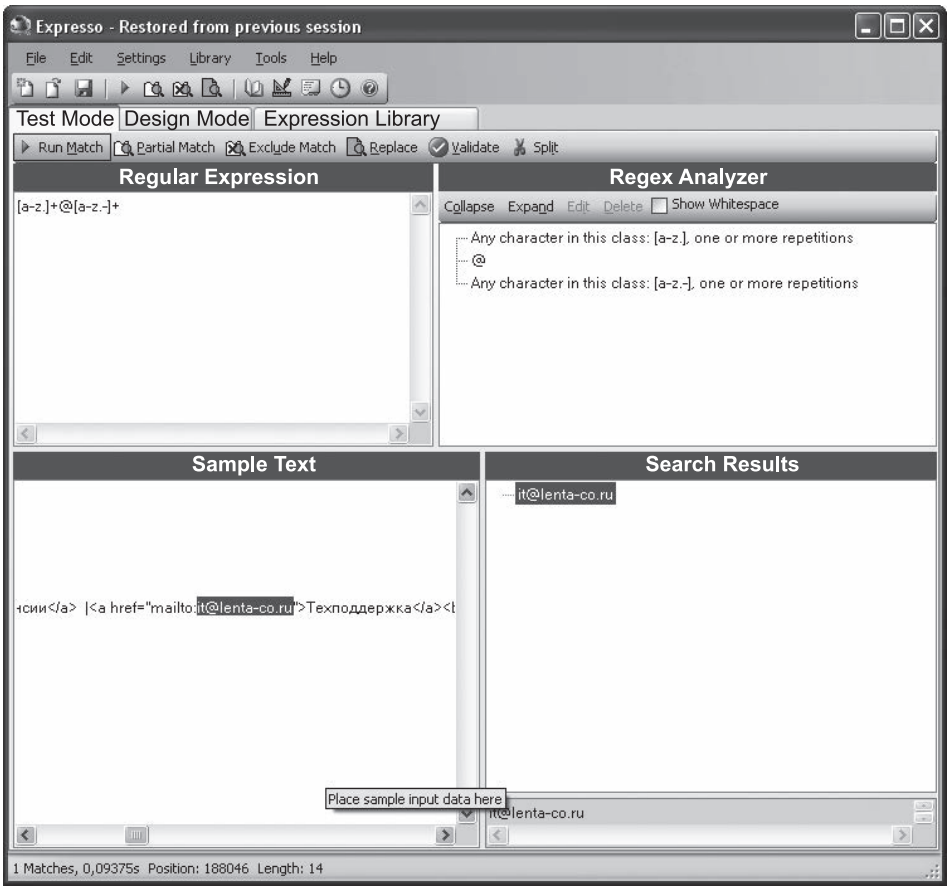

Рис. 66. Интерфейс программы Expresso

Основное рабочее окно программы — набор из трех страниц: **Test Mode** (режим проверки), **Design Mode** (режим создания) и **Expression Library** (библиотека выражений). В основном мы будем работать на первой в режиме тестирования.

На странице **Test Mode** есть управляющая панель и несколько разделов:

- **Regular Expression** — само регулярное выражение. В этом месте мы можем написать выражение просто как текст.
- **Regex Analyzer** — анализатор выражения. В этом окне выражение будет разобрано на части так, как его будет интерпретировать механизм. Здесь можно проверить синтаксис. Если выражение сложное, то будет показано дерево для проверки вложения частей.
- **Sample Text** — пример текста. В этот раздел загружается текст, к которому и применяется регулярное выражение. Текст мож-

но просто скопировать сюда. Тогда он будет считаться подстрокой для поиска.

- **Search Result** — раздел, в котором отображается результат применения — поиска по регулярному выражению. Будут показаны все найденные вхождения, и если вхождение выбрать, в примере текста будет показано, где именно оно найдено. Если выражение включает в себя несколько групп, то результат можно развернуть опять-таки как дерево.

Решим несколько задач, опираясь на сведения о синтаксисе регулярных выражений, о котором говорили в учебнике. Во всех задачах требуется написать выражение, проверить его, выполнить и оценить результат.

Задача 1

Выбрать из текста все целые числа.

Для начала выясним, из чего может состоять целое число. Очевидно, что в основном из цифр. Кроме цифр перед числом могут стоять знаки «−» или «+» (а могут и не стоять). К тому же последовательность цифр может стоять в начале или в конце отдельного слова, но такая конструкция (например, «Иванов-2-й») числом не будет.

Используем для решения задачи выражения «символьные классы» и «символы повторения»:

```
[+-]?\d+
```

Выражение очень простое: первый его символ это либо «+», либо «−», что мы и записали в его классе. Он может возникнуть один раз, а может не возникнуть, поэтому выбран квантификатор ?, т. е. необязательный элемент.

Следом идет набор цифр (символьный класс \d), состоящий минимум из одной цифры.

Применив это выражение для поиска, выясняем, что оно действительно находит все правильно написанные числа, но ситуацию «Иванов-2-й» обрабатывает неверно – выделяет из этой подстроки число 2.

Чтобы исключить такие ситуации, воспользуемся так называемой *незахватывающей проверкой* — конструкцией, которая проверяет часть строки на соответствие шаблону, но не выделяет эту часть строки из результата. Синтаксис этой конструкции таков: (?:<выражение>) — мы проверяем, что <выражение> есть, но не включаем его в результат. Получится примерно вот что:

```
(?:\s)[+-]?\d+(?:\s),
```

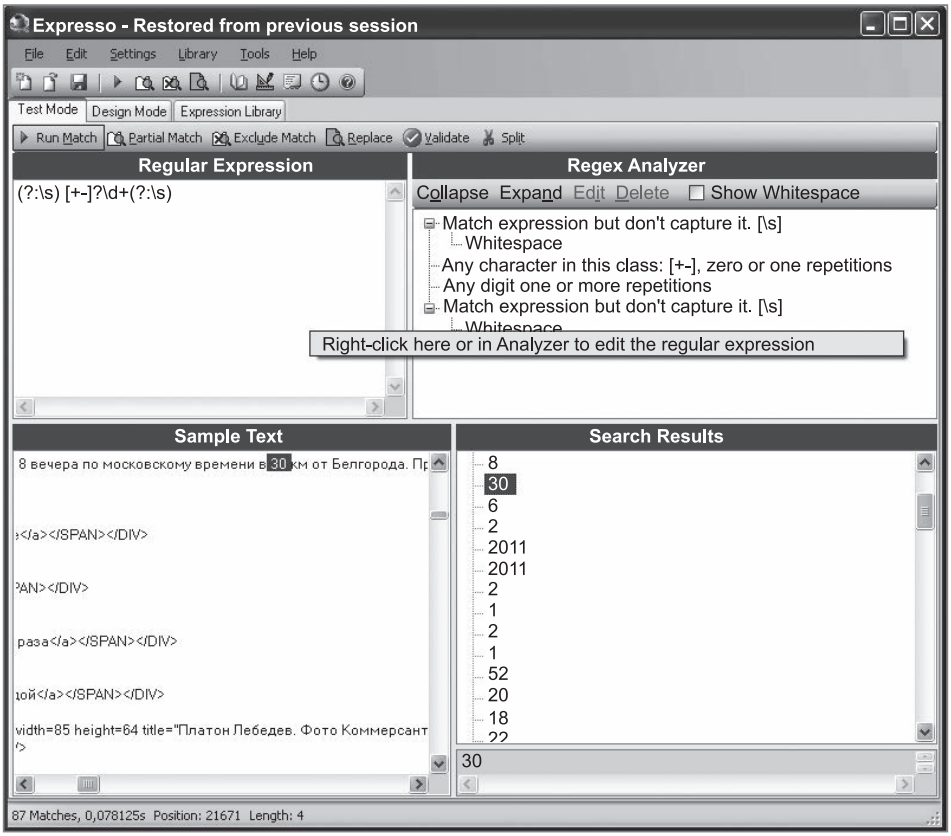

Рис. 67. Результат тестирования текста HTML-кода страницы

то есть мы потребовали, чтобы перед и после нашего числа стояли пробельные элементы (класс \s).

На рисунке 67 изображен результат тестирования, который мы получили, записав в качестве примера HTML-код страницы в виде текста.

Обратите внимание! В разделе Regex Analyzer подробно разобрано, в каком порядке и как анализируется текст в этом регулярном выражении.

Внимательно посмотрев, что именно выделено, мы поймем, что нашли не только нужное, но и пробелы слева и справа от найденного числа. Чтобы выделить только целое число, придется заключить нужную часть в круглые скобки. Тогда первая группа внутри найденного совпадения как раз и будет нужным нам числом:

```
(?:\s)([+-]?\d+)(?:\s)
```

Теперь рядом с каждым совпадением появится знак развертывания ветви, в которой и будет нужная группа.

Задания

1. Напишите выражение для поиска чисел в составе слов.
2. Напишите выражение для поиска чисел в формате числа с плавающей точкой.
3. Напишите выражение для поиска телефонных номеров.

Задача 2

Выбрать из текста все упоминания весов в метрической системе (например, 5 килограмм, 10 тонн и т. д.). Для упрощения задачи считать, что все веса – целые, а меры написаны без сокращений.

Определим структуру искомого текста: он состоит из числа и меры, которая за ним записана. Как искать число, мы разобрали в предыдущей задаче.

В метрической системе мер меры длины и веса образуются из основного слова и десятичной приставки. Перечислим некоторые десятичные приставки:

Кратное	Название	Кратное (СИ)	Название	Кратное	Название	Кратное (СИ)	Название
10^0	тонна	10^6	мегаграмм			10^5	(нет)
10^1		10^7	(нет)	10^{-1}	децитонна (центнер)	10^4	(нет)
10^2		10^8	(нет)	10^{-2}	сантитонна	10^3	килограмм
10^3	килотонна	10^9	гигаграмм	10^{-3}	миллитонна	10^0	грамм
10^6	мегатонна	10^{12}	тераграмм	10^{-6}	микротонна	10^{-3}	миллиграмм

Чтобы реализовать эту структуру, используем выбор из нескольких вариантов:

```
(?:\s)(\d+)(?:\s)+(((?:кило|мега|гига|тера|санти|милли|микро)
                    ?(?:грамм|тонн))|(г|кг|т|кт))
```

Напомним, что знак вопроса после группы задает возможное (но не обязательное) появление.

Как видите, кроме десятичных приставок мы добавили сокращения. На них правила записи не распространяются, поэтому придется их перечислить.

Применив это выражение, выясняем, что оно действительно выбирает необходимое, но не только (рис. 68).

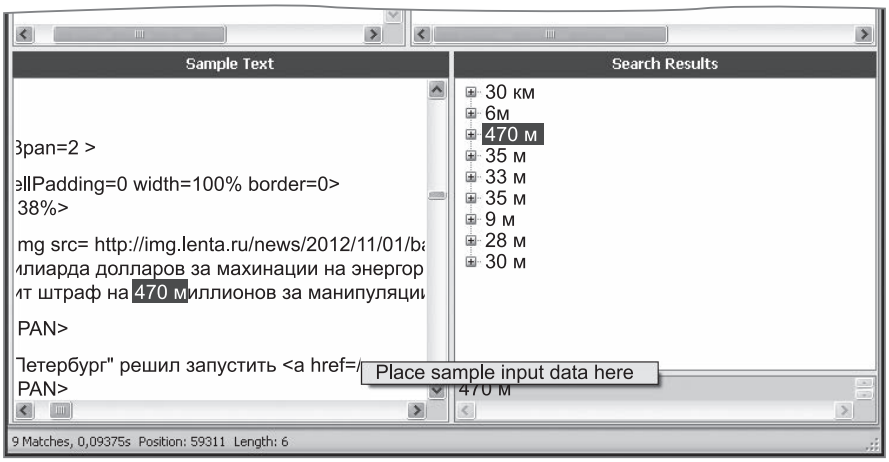

Рис. 68. Ошибочный результат выборки

«470 м» воспринято как мера длины, но речь в тексте идет о размере штрафа (см. рис. 68). Поэтому придется добавить в конец дополнительное условие — расширенное множество символов:

```
(?:\s)(\d+)(?:\s)+(((?:кило|мега|гига|тера|санти|милли|микро)
?(?:грамм|тонн))|(г|кг|т|кт))[ .,:;!?{}()\\[\]]+
```

Результат выборки показан на рис. 69.

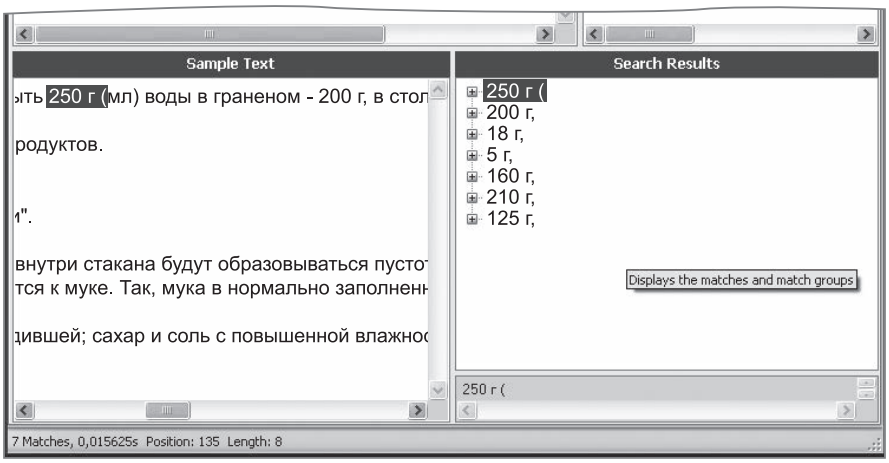

Рис. 69. Результат выборки при расширенном множестве символов

Задания

1. Внесите изменения в выражение так, чтобы из найденного можно было выделить ТОЛЬКО количество и меру веса.
2. Напишите аналогичное выражение для выбора мер расстояния.

Задача 3

Выбрать из текста все адреса электронной почты.

Как обычно, первое, что мы делаем, это выясняем, из каких символов состоит адрес. Согласно стандарту, адрес может состоять из двух частей: имени пользователя и имени почтового отделения. Имена в адресе разделяются знаком @, который является обязательным. Имена и слева, и справа от @ могут состоять из символов латинского алфавита, цифр, точки, знака тире и подчеркивания. Регистр букв имеет значение. Стандарт допускает использование и ряда других символов, но на нашу работу это повлияет мало.

Наше выражение будет состоять из трех частей:

1) множества символов — названия ящика, причем название непустое — минимум один символ из этого множества должен присутствовать;
2) знака @;
3) множества символов — названия почтового отделения.

Множество символов запишем так: [A-Za-z0-9.-_]. Поскольку оно встречается минимум один раз, зададим повтор знаком «+». В итоге выйдет примерно следующее:

[A-Za-z0-9.-_]+@[A-Za-z0-9.-_]+

Для проверки скопируем реальный текст в окно примера и применим выражение. Текст мы взяли из html-кода страницы сервера новостей и, применив, увидели примерно вот что (см. окно **Search Results** на рис. 70).

Как видим, адреса выделены неправильно. Причина — появились лишние символы-разделители. Откуда они взялись? Проанализируем запись множества и обнаружим, что добавили лишний диапазон .-_. Мы имели в виду просто символ -, но он имеет в записи множеств специальный смысл. Поэтому перед его упоминанием ставим знак \.

Теперь результат будет верен, он станет примерно таким, как на первой иллюстрации. А выражение будет выглядеть так:

[A-Za-z0123456789.\-_]+@[A-Za-z0-9.\-_]+

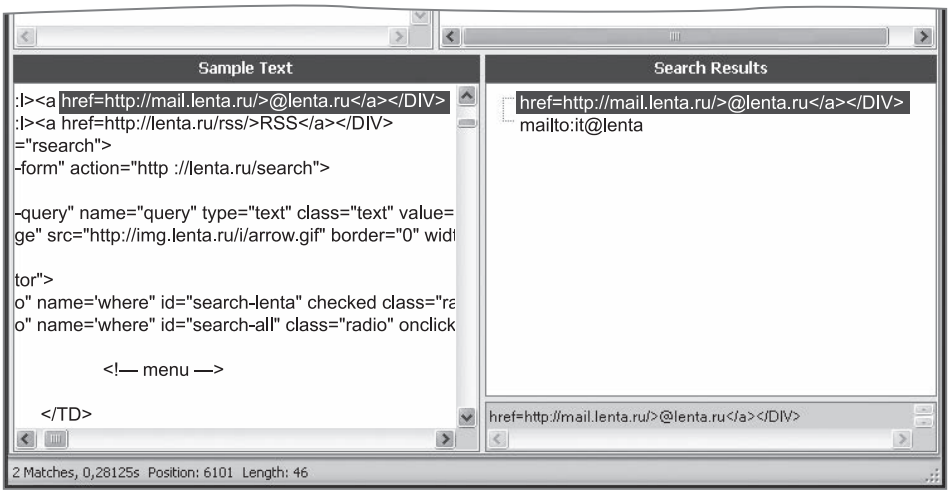

Рис. 70. Выборка адресов электронной почты из HTML-кода страницы сервера новостей

Тем не менее этого недостаточно. Название почтового ящика не может начинаться с точки и содержать две точки подряд. После знака-разделителя тоже не может стоять точка.

Решим эту задачу иначе — разобьем последовательность на группы:

[A-Za-z0123456789\-_]+(?:\.[A-Za-z0123456789\-_])*@ [A-Za-z0123456789_]+(?:\.[A-Za-z0123456789\-_])+

> Обратите внимание! В обоих случаях в начале идет группа без точки, а потом некоторое количество групп, которые начинаются с точки. Если в первой части таких групп может и не быть, то во второй они есть обязательно.

Задания

1. Напишите регулярное выражение для выборки из текста дат в формате ДД/ММ/ГГ или ДД/ММ/ГГГГ. В выражении учтите, что в датах не может быть номера дня больше 31 и месяцев всего 12.
2. Вы получили текст, в котором все даты записаны в виде ГГГГ-ММ-ДД (например, 2013-12-31 – 31 декабря 2013 года). Напишите регулярное выражение, преобразующее все эти даты к привычному нам виду: ДД.ММ.ГГГГ (т. е. 31.12.2013). Для проверки воспользуйтесь режимом Design Mode.
3. Вы получили текстовый файл, в котором описываются виды цветов. Каждый цветок описан набором строк (название, латинское название, внешний вид, место происхождения, кли-

мат), пустых строк в описаниях нет. Между описаниями цветов — пустая строка. Напишите регулярное выражение, преобразующее этот набор в текстовую таблицу, где на каждый цветок отведена одна строка, а атрибуты перечислены через точку с запятой.

4. Напишите регулярное выражение для обратного преобразования.
5. Вы получили текстовый файл протокола обращений пользователей к ресурсам среды Интернет, в котором построчно перечислены атрибуты каждого обращения через запятую. Порядок полей совпадает, названия полей заданы в первой строке. Подготовьте регулярное выражение, позволяющее отфильтровать только 1, 4 и 7-й столбцы и перечисляющее значения через точку с запятой.
6. Выделите с помощью регулярного выражения все имена собственные из географического текста (например, реферата).
7. Подготовьте набор регулярных выражений для выявления и устранения несоответствия текстов правилам набора, приведенным в учебнике.

Использование регулярных выражений при подготовке программ

Теперь рассмотрим, каким образом мы сможем использовать механизм регулярных выражений в своих программах. Стандарт Паскаля не предусматривает библиотек для работы с ними, но в учебной среде нам доступна библиотека, входящая в комплект платформы .Net. Подключив ее в модуле Uses, получим возможность использовать объект Regex и необходимый тип Match — совпадение.

```
program RegularExperssionsMatch;
uses System.Text.RegularExpressions;

var
  s: String;
  txt: Text;
  m: Match;

begin
  Assign(txt, 'k:\demo.txt');
  Reset(txt);

  while not Eof(txt) do
  begin
    ReadLn(txt, s);
```

```
    foreach m in Regex.Matches(s,
                    '[A-Za-z0123456789\-_]+(?:\.
                     [A-Za-z0123456789\-_])*
                     @[A-Za-z0123456789_]+
                    (?:\.[A-Za-z0123456789\-_])+') do
    Writeln('Найдено:', m.Value);
  end;
  Close(txt);
end.
```

Обратите внимание на конструкцию *foreach… in…* — с ее помощью мы получаем возможность перебрать все элементы во множестве-коллекции. Эта синтаксическая конструкция отсутствует в классическом языке Паскаль, но есть в расширенном (Extended Pascal, ISO 10206:1990).

Задания

1. Напишите программу поиска заданного слова в текстовом файле. Считайте, что текст записывается без использования переносов.
2. Напишите программу для поиска двух идущих подряд заданных слов. Учтите, что между словами может быть несколько разделителей, например два и более пробелов.
3. Напишите программу, подсчитывающую количество упоминаний в текстовом файле семьи Ивановых (или любого из ее членов). Считайте, что вся семья носит одну фамилию.

Помимо поиска совпадений, регулярное выражение можно использовать для разделения строки на части. В этом случае регулярное выражение будет использовать для поиска последовательности-разделители.

Рассмотрим пример, в котором с помощью последовательностей-разделителей строки делятся на слова, т. е. получаются последовательности символов между последовательностями разделителей.

```
Program RegularExperssionsSplit;
uses System.Text.RegularExpressions;

var
 s,s0 : String;
 txt: Text;

begin
  Assign(txt, 'k:\demo.txt');
  Reset(txt);
```

```
  while not Eof(txt) do
  begin
    ReadLn(txt, s);
    foreach s0 in Regex.Split(s, '[ ,.!?{}()\
             [\]/*+\-]+') do
      Writeln(s0);
  end;

  Close(txt);

end.
```

Задания

1. Напишите программу подсчета общего количества слов в текстовом файле. Условьтесь, что текст записывается без использования переносов, числа не считаются словами.
2. Напишите программу обработки текстового файла с переносами, которая уберет переносы, т. е. восстановит разбитые слова и результат сохранит в файл, добавив к имени .new. Учтите, что знак, обозначающий перенос, может применяться и в других случаях.

Еще один метод, который нам может понадобиться, — *замены*. Здесь требуется второе выражение, которое определит, что должно оказаться на месте найденного фрагмента.

Приведем пример, в котором по всему текстовому файлу квадратные скобки заменяются на угловые.

```
Program RegularExperssionsReplace;
uses System.Text.RegularExpressions;

var
 s,s0 : String;
 txt: Text;

begin
  Assign(txt, 'k:\demo.txt');
  Reset(txt);

  while not Eof(txt) do
  begin
    ReadLn(txt, s);
    s0 := Regex.Replace(s, '\[.+\]', '<$0>')
```

```
      Writeln(s0);
    end;

    Close(txt);

end.
```

Задания

1. Вы получили текст, в котором все даты записаны в виде ГГГГ-ММ-ДД (например, 2013-12-31 – 31 декабря 2013 года), или ДД-ММ-ГГ, или ДД-ММ-ГГГГ. Напишите программу, которая преобразует эти даты к привычному нам виду: ДД.ММ.ГГГГ (т. е. 31.12.2013) и результат сохранит в новый текстовый файл.
2. Вы получили текст, в котором записаны номера телефонов в разных формах: +7 (495) 123-4567, 123-4567, 1234567, 499-1234567. Напишите программу, которая преобразует все эти варианты к общему стандарту +7 (495) 123-4567 и результат сохранит в новый текстовый файл. По умолчанию (если не указано) предполагаем, что номера российские, код города — на ваше усмотрение.

Частотный анализ

Идея частотного анализа очень проста: если имеется набор каких-то элементов (букв, слов, предложений, яблок — чего угодно), то простой способ оценить этот набор — посчитать, сколько в этом наборе элементов каждого вида (каждой буквы, слова, размера яблок). Узнав, какую долю, т. е. насколько часто встречается элемент, мы можем делать какие-то выводы.

Анализ частоты символов в тексте

Первая группа задач, которую мы будем решать, связана с кодировками и алфавитом.

Дано: текст на русском языке в форме файла, в кодировке Windows 1251, длина заранее не известна.

Требуется: составить частотную таблицу упоминаемости символов в этом тексте.

Решение

Нетрудно увидеть, что основная часть этой задачи состоит в подсчете количества упоминаний каждого символа в тексте. Вспомним определение таблицы кодировки и особенности конкретной кодировки Windows-1251: в ней всего 256 символов с номерами от 0 до 255 включительно.

Создадим массив из 256 беззнаковых (не может быть отрицательного количества) целых чисел. Будем считывать по одному символу и увеличивать значение в соответствующей (т. е. с номером, равным коду символа) ячейке массива. Единственное, что нам придется сделать, — преобразовать символ в его код.

Заготовка программы будет выглядеть примерно так:

```
program frequency;

var
  txt : Text;
  c : char;
  countLetters : array[0..255] of Word;
  i : Word;

begin
  Assign(txt,'demo.txt');
  Reset(txt);

  for i:=0 to 255 do
    countLetters[i] := 0;

  while not Eof(txt) do
  begin
    read(txt,c);
    inc(countLetters[ ord(c) ]);
  end;

  for i:=0 to 255 do
    WriteLn( i, ' - ',countLetters[i]);

  Close(txt);
end.
```

Возьмем какой-нибудь достаточно большой текстовый файл, сохраним его под именем demo.txt и посмотрим, что получится.

Проанализировав результаты, мы выясняем, что:

1) довольно много символов не относится к языку, а часть — даже не буквы (особенно символы «возврат каретки» и «перевод строки», «пробел» и т. п.). Очевидно, они не должны влиять на частоту;
2) если объем текста достаточно велик (около 3 Мб), появляется довольно большая вероятность получить ошибку «Переполнение»;

3) поскольку нам придется считать общее количество символов, то переменная должна допускать бóльшие значения, чем обычный тип Word.

При подготовке программы нужно ввести фильтрацию символов. Самый простой способ, если мы работаем с русским языком, — учитывать только символы расширенного набора, т. е. с кодами больше 255.

В итоге программа будет примерно такой:

```
program frequency;

var
  txt : Text;
  c : char;
  countLetters : array[0..255] of LongWord;
  i : Word;
  sigma : LongWord;

begin
  Assign(txt,'k:\demo.txt');
  Reset(txt);

  for i:=0 to 255 do
    countLetters[i] := 0;

  while not Eof(txt) do
  begin
    read(txt,c);
    if ord(c) > 127 then
      inc(countLetters[ ord(c)]);
  end;

  sigma := 0;
  for i:=0 to 255 do
    sigma := sigma + countLetters[i];

  for i:=128 to 255 do
    WriteLn( chr(i), '(',i, ')',
             ' - ',(countLetters[i]/sigma):8:7 );

end.
```

Вот результат обработки программой текста I тома романа «Война и мир»:

Буква	Частота	Буква	Частота
о (238)	0.1123347	В (194)	0.0020581
а (224)	0.0822470	А (192)	0.0019786
е (229)	0.0789770	П (207)	0.0019214
и (232)	0.0653834	К (202)	0.0017662
н (237)	0.0635027	О (206)	0.0015796
т (242)	0.0557268	Б (193)	0.0015168
с (241)	0.0524678	М (204)	0.0011122
л (235)	0.0501234	Д (196)	0.0010235
р (240)	0.0445219	С (209)	0.0008868
в (226)	0.0437774	Р (208)	0.0008591
к (234)	0.0346491	Т (210)	0.0008406
д (228)	0.0292490	И (200)	0.0008018
м (236)	0.0283326	Я (223)	0.0005616
у (243)	0.0283308	ъ (250)	0.0005247
п (239)	0.0236641	Г (195)	0.0004896
я (255)	0.0224928	Э (221)	0.0004360
г (227)	0.0201539	Ч (215)	0.0003991
ь (252)	0.0193780	Е (197)	0.0003399
ы (251)	0.0188958	Л (203)	0.0002697
з (231)	0.0175231	З (199)	0.0002051
б (225)	0.0156794	У (211)	0.0002014
ч (247)	0.0131742	Ж (198)	0.0001496
й (233)	0.0114783	Ш (216)	0.0001275
ж (230)	0.0099319	Ф (212)	0.0001256
ш (248)	0.0092742	Х (213)	0.0001127
х (245)	0.0083801	Ц (214)	0.0000388
ю (254)	0.0064421	Ю (222)	0.0000148
ц (246)	0.0039979	Ь (220)	0.0000074
щ (249)	0.0027897	Щ (217)	0.0000037
э (253)	0.0025735	Й (201)	0.0000018
ф (244)	0.0021024	Ъ (218)	0.0000000
Н (205)	0.0020913	Ы (219)	0.0000000

Предлагаем несколько вопросов и предложений для совершенствования программы.

1. Проанализируйте вывод программы. Стал ли результат ее работы более коротким из-за ограничений (русский язык)?
2. Можно ли сократить объем занимаемой памяти?
3. Стоит ли вместо отдельного цикла с подсчетом количества русских букв добавлять единицу к сумме сразу в основном цикле?
4. Что еще нужно учесть, чтобы получить настоящую частотную таблицу, т. е. таблицу, в которой частота подсчитана именно *для буквы*, а не *печатного символа*?
5. Предложите вариант усовершенствования, который сможет работать с кодировкой Unicode.

Очевидно, что частоты встречаемости символов в больших русских текстах хоть немного, но будут различаться. Но вот порядок символов по частоте упоминаемости будет примерно одинаковым.

Задания

Используя эту программу, ответьте на следующие вопросы.

1. Подсчитайте частоты для большого объема текстов разного вида: для официальных документов и литературных произведений. Совпадают ли они? Сохраняется ли порядок символов по частоте встречаемости?
2. Как с помощью частот встречаемости различать таблицы кодировок текста: CP866, Windows-1251, KOI-8?
3. Напишите программу преобразования текста между разными таблицами кодировки:
   1) Win1251 <-> KOI-8;
   2) Win1251 <-> CP866;
   3) CP866 <-> KOI-8.

Анализ частоты упоминаемости слов в тексте

Дано: текст на русском языке в форме файла, в кодировке Windows 1251, длина заранее не известна.

Требуется: составить частотную таблицу упоминаемости слов в этом тексте.

Решение

При решении этой задачи нужно учесть несколько особенностей:

1) мы не знаем, сколько будет слов в тексте;
2) в отличие от символов слово в тексте нужно выделить, потому что это не единый объект, а набор объектов;
3) слово может занимать в предложении первое или не первое место и потому может быть написано по-разному;

4) слово может быть разделено на части, если текст заранее отформатирован устаревшими методами.

На самом деле, трудностей больше, но решать их мы будем позже.

Выделять слова из текста будем с помощью регулярного выражения. Чтобы его построить, вспомним, что такое «слово». Нас интересует строго формальное определение, которое позволит нам выделить слово из потока символов. С этой точки зрения слово — это последовательность взятых из текста символов, но не разделителей.

Для создания программы воспользуемся методом деления строки на части по регулярному выражению и библиотекой работы с коллекциями из .Net для организации словаря:

```
Program Frequency;

uses System.Text.RegularExpressions,System.
     Collections.Generic;

var
 s,s0 : String;
 txt : Text;
 tx  : System.Collections.Generic.Dictionary<string,
       integer>;
 m: Match;

begin
  tx := new
   System.Collections.Generic.Dictionary<string,
    integer>;

  Assign(txt,'k:\demo.txt');
  Reset(txt);

  while not Eof(txt) do
  begin
    ReadLn(txt,s);
    foreach s0 in Regex.Split(s,
              '[ ,.!?(){}+*\-;:»''\[\]\\]+') do
      if s0 <> '' then
         if tx.ContainsKey(s0) then
           tx[s0] := tx[s0]+1
        else
           tx.Add(s0,1);
```

```
    end;
    close(txt);

    Assign(txt,'k:\frequency.csv');
    Rewrite(txt);

    WriteLn( txt,'Word;Count;' );
    foreach s0 in tx.Keys do
      WriteLn( txt,s0, ';', tx[s0],';' );
    Close(txt);
end.
```

Запустив эту программу, проанализируем текстовый файл и получим результаты примерно в таком виде:

Автор	2
автор	8
автора	3
авторе	1
авторитет	9
авторитета	2
авторитетами	1
авторитете	1
авторитетная	1
авторитетнее	1
авторитетной	1
авторитетную	1
авторитетом	1
авторитету	3
авторитеты	1
авторов	2
авторских	1
авторскую	1
авторство	1
автору	3
авторы	3
Авторы	1

Ага	16
агитации	2
агрегата	1
агрегаты	1
агронома	1
Административная	1
административная	1
административной	1
административную	2
административные	1
административный	1
администратор	2
администратора	1
администрацией	2
администрации	1
администрировании	1
администрировать	1

Изначально результат будет не отсортирован, но это сделать несложно.

На что нужно обратить внимание, прежде чем воспользоваться результатами для анализа? На несколько особенностей:

1) учитывается регистр, поэтому «Административная» и «административная» — это два разных слова. Очевидно, что с точки зрения извлечения смысла, это не так;
2) самые частые слова не несут никакого значения для анализа текста. Сложно предположить, что нам важна частота слов «Ага» или слова «И»;
3) мы не учитываем формы слов, поэтому, например, видим шесть вариантов одного и того же по смыслу слова.

Первая проблема решается довольно просто: перед разбиением нужно преобразовать всю строку к единому регистру — верхнему или нижнему. Это реализует метод самой строки toLower (или toUpper — логически разницы никакой нет). Добавим в программу

```
s := s.ToLower;
```

Надеемся, читатель в состоянии определить сам, куда это поместить.

Вторая проблема решается несколько труднее: нельзя просто выкинуть слово, потому что оно часто встречается в этом конкретном тексте. Чтобы понять, какие слова не характеризуют конкретный текст, нужно проанализировать большое количество эталонных текстов — выделить самые частые слова. К счастью, за нас это уже сделано: воспользуемся статистической информацией с сайта корпуса русского языка: раздел «Частоты». Самые частые 100 словоформ будут нашим стоп-листом:
http://ruscorpora.ru/1grams.top.html

Для реализации нужно сохранить их в файл, а в программе предусмотреть отдельную коллекцию стоп-слов. В эту коллекцию загрузить слова из файла, проверяя перед добавлением — есть там уже это слово или нет. Реализацию этого авторы предоставляют читателям в качестве самостоятельного упражнения.

Третья задача намного сложнее. Для того чтобы учесть, что одно и то же (с точки зрения статистического анализа) слово встречается в разных формах, нужно привести эти формы к одной универсальной последовательности символов.

Существуют два основных подхода. Первый основан на специальном преобразовании, которое называется *стеммингом*. Этот метод исключает некоторые элементы слов и создает в результате не слово, но характеризующий его набор символов. Метод был разработан и применяется в основном для организации поиска на английском языке.

Для русского языка стемминг тоже существует, но дает худший результат, поскольку словообразование в русском языке сложнее. К тому же этот метод не подходит для нашей задачи — нам нужно будет легко узнавать слова.

Во втором методе словоформы преобразуются к единой эталонной форме — *лемме*. Например, для существительного лемма — это слово в единственном числе и именительном падеже. Это довольно сложная задача, поэтому мы не будем самостоятельно разрабатывать для нее решение, а воспользуемся готовым — демонстрационной версией лингвистической библиотеки Solarix. Приведем пример подключения и использования этой библиотеки в PascalABC:

```
{$reference lemmatizator_fx.dll}
uses SolarixLemmatizatorEngineNET,System;

var
    a : IntPtr;
    colLem : Integer;
    lem : Text.StringBuilder;
begin

  lem := Text.StringBuilder.Create(10);
     a   :=   LemmatizatorEngine.sol_LoadLemmatizatorW
              ('lemmatizer.db', 1);
  colLem := LemmatizatorEngine.sol_GetLemmaW
            (a,'примерная',lem,10 );

  writeLn(colLem,' ',lem.ToString);
end.
```

Программа начинается с указания транслятору подключить библиотеку динамической компоновки (lemmatizator_fx.dll). Для организации работы нужно положить в каталог с самой программой несколько дополнительных файлов: lemmatizator_fx.dll, lemmatizator.dll и lemmatizer.db.

В разделе подключения модулей мы указываем, что будем использовать функции из этой библиотеки (формально говоря, мы подключаем пространство имен): uses SolarixLemmatizatorEngineNET, System.

Последовательность работы такова.

1. Инициализировать механизм, указав основной словарь для его работы. Идентификатор-указатель механизма будем хранить в переменной *a*:

```
a := LemmatizatorEngine.sol_LoadLemmatizatorW
    ('lemmatizer.db',1)
```

Второй параметр указывает, что, если слово не удалось найти в словаре, можно использовать функцию-стеммер.

2. Создать специальную строку для хранения результата длиной 10 символов (для примера этого достаточно):
   ```
   lem := Text.StringBuilder.Create(10);
   ```
3. Выполнить поиск леммы для слова «примерная»: передать указатель на механизм, само слово, строку для результата и его возможную длину.
4. Перед выдачей результат преобразовать снова в обычную строку, вызвав метод toString.

Таким образом, мы получаем возможность создать программу, которая может подготовить данные для контент-анализа.

Контент-анализ — один из самых популярных и мощных методов анализа текстов (в самых разных областях гуманитарного знания). Он построен на том, что смысл, который вложен в текст, неизбежно отразится на словах, из которых текст состоит. Самый простой случай применения этого метода — подсчитать количество отдельных слов. Более сложные и эффективные методы анализа потребуют не только подсчетов, но и определения его морфологических признаков: является ли это слово глаголом, прилагательным, существительным и т. д.

Подсчитаем встречаемость каждого слова в тексте. Более важные будут встречаться чаще. Для этого нужно:

1) разбить текст на отдельные слова;
2) отфильтровать слишком короткие или незначащие слова (связки);
3) преобразовать все слова к нормальной форме;
4) подсчитать их количество;
5) сохранить результаты в виде, годном для дальнейшей обработки.

Вот один из несложных вариантов такой программы:

```
{$reference lemmatizator_fx.dll}

Program Frequency;

uses System.Text.RegularExpressions,
     System.Collections.Generic,
     SolarixLemmatizatorEngineNET,System;
```

```
var
 s,s0,s1      : String;
 txt          : PABCSystem.Text;
 tx,stop      : System.Collections.Generic.Dictionary
                          <string, integer>;
 a            : IntPtr;
 lem          : Text.StringBuilder;

begin
    tx  :=  new  System.Collections.Generic.Dictionary
                     <string, integer>;
   stop  :=  new  System.Collections.Generic.Dictionary
                     <string, integer>;
  lem := Text.StringBuilder.Create(30);
    a  :=  LemmatizatorEngine.sol_LoadLemmatizatorW(
             'lemmatizer.db',0);

  Assign(txt,'k:\stop.txt');
  Reset(txt);
  while not Eof(txt) do
  begin
    ReadLn(txt,s);
    if s <> '' then
      if not stop.ContainsKey(s) then
         stop.Add(s.ToUpper,1);
  end;
  close(txt);

  Assign(txt,'k:\message2012.txt');
  Reset(txt);
  while not Eof(txt) do
  begin
    ReadLn(txt,s);
    s := s.ToUpper;
    foreach s0 in Regex.Split(s,
       '[ ,.!?(){}+*\-;:»\[\]\\&1-9]+') do
      if length (s0) > 1 then
      begin
          if not stop.ContainsKey(s0) then
          begin
            LemmatizatorEngine.sol_GetLemmaW
              (a, s0,lem,30 );
            s1 := lem.ToString;
```

```
            if not stop.ContainsKey(s1) then
               if tx.ContainsKey(s1) then
               tx[s1] := tx[s1]+1
            else
               tx.Add(s1,1);
         end;
      end;
   end;
   close(txt);

   Assign(txt,'k:\frequency.csv');
   Rewrite(txt);

   WriteLn( txt,'Word;Count;' );
   foreach s0 in tx.Keys do
     WriteLn( txt,s0, ';', tx[s0],';' );
   Close(txt);
end.
```

Такой способ анализа требует достаточно длинного текста, иначе результат будет очень неустойчивым.

На следующей странице представлен примерный результат такой обработки (первые 50 лемм) текста послания Президента РФ Федеральному собранию в конце 2012 года[5].

Конечно, теперь этот список необходимо дополнительно обработать, например отнести слова к конкретным областям деятельности. Но уже сейчас на основании этого анализа можно сказать, каким темам было уделено много внимания, а каким — меньше, а на основании этого — какой смысл вкладывался в этот текст.

[5] Источник текста: http://президент.рф/выступления/17118

Word	Count		Word	Count
ДОЛЖНЫЙ	87		ЗАДАЧА	22
РОССИЯ	59		ЛЮДИ	22
НУЖНЫЙ	59		ОБЩЕСТВО	22
СТРАНА	55		ПРАВИТЕЛЬСТВО	22
СВОЙ	48		ЗНАТЬ	21
НАШ	44		УВАЖАЕМЫЙ	21
РАЗВИТИЕ	42		ВСЕГО	20
ГОД	41		СЕЙЧАС	19
ТАКОЙ	39		СТАТЬ	19
НОВЫЙ	36		ПРОЦЕНТ	19
РАБОТА	34		СИСТЕМА	19
РОССИЙСКИЙ	31		ЭКОНОМИКА	18
ВСЁ	29		МИР	18
ОБ	27		КОЛЛЕГА	18
ВАЖНЫЙ	27		ВОЗМОЖНОСТЬ	18
ГОСУДАРСТВО	26		СЧИТАТЬ	18
ХОТЕТЬ	25		ЕЩЁ	17
ГРАЖДАНИН	25		НАШИТЬ	17
ГОВОРИТЬ	25		ПЕРВЫЙ	17
ТОТ	23		НАЦИОНАЛЬНЫЙ	17
РЕГИОН	23		ВНИМАНИЕ	17
РАБОТАТЬ	23		ПРОСИТЬ	17
СФЕРА	22		КОНЕЧНЫЙ	16
ВОПРОС	22		МНОГО	16
СЕГОДНЯ	22		СКАЗАТЬ	16

11 класс

- Проект «Графика и визуализация» к главе 1 учебника
- Проект «Обработка звукового файла» к главе 2 учебника
- Проект «Защита данных в сетях» к главе 5 учебника

Проект

«Графика и визуализация»

Часть 1. Графика на плоскости (2D)

Решая любые задачи, связанные с визуализацией текста, фотографии, схемы и т. п., мы оказываемся перед необходимостью закрасить (или перекрасить) точки-пиксели на плоскости. Плоскостью может быть экран компьютера или лист бумаги, а точки могут светиться или просто наноситься краской — сути это не изменит.

Как вы знаете из учебника, чтобы представить и обработать набор таких точек, мы описываем цвет каждой из них (какой-либо цветовой моделью), а чтобы определить положение, задаем координатную сетку, узлы которой и образуют прямоугольный набор — матрицу пикселей.

Каким же образом формируется такая матрица, т. е. какие алгоритмы мы используем, чтобы тем или иным способом закрасить точки? Ответ на этот вопрос в общем виде есть в учебнике, а в здесь мы рассмотрим практическую реализацию.

Для решения задач воспользуемся уже знакомой средой программирования — PascalABC.Net. В рамках этой среды мы получим доступ к библиотекам функций, которые выполнят за нас техническую часть задачи: закрашивание точки в некоторый цвет, чтение графического файла и т. п. Конкретные функции и средства мы будем рассматривать по мере необходимости.

Первое, что нам понадобится, — исходное изображение. Получение, сжатие и декодирование изображения — сложные задачи, на решение которых здесь у нас нет времени. Воспользуемся готовыми средствами. В составе нашей среды есть библиотеки работы с изображениями, основанные на средствах Windows.

Практически во всех задачах нам понадобится создать изображение (т. е. матрицу пикселей) и показать его на экране. Вот как это делается в нашей среде:

```
uses ABCObjects,GraphABC; // Графическая библиотека

var
```

```
    // Специальный объект класса Picture
    //для работы с изображением
    p : Picture;
begin
// Создание объекта из файла
 p := new Picture( 'example.jpg');
 p.Draw(0,0); // Отрисовка изображения в рабочем окне
 while true do  ; // Ждем закрытия окна
end.
```

Дальнейшая работа с изображением зависит от вида задачи, которая перед нами будет стоять. Рассмотрим некоторые из возможностей.

Преобразование цвета

Начнем с задач, связанных с изменением общих характеристик растрового изображения. Одна из них — изменение яркости изображения. Мы решим эту задачу, используя переход между двумя цветовыми моделями.

Яркость в основной нашей рабочей модели (RGB) не является характеристикой точки, но входит как координата в модель HLS. Поэтому общий подход будет таким: пересчитать цвет из модели RGB в модель HLS, в этой модели изменить яркость и пересчитать обратно.

Процедуры преобразования не сложны, но описываются достаточно объемно. В них используются следующие параметры:

- R,G,B — значения компонентов цвета, преобразованные в диапазон [0,1];
- MAX, MIN — максимальное и минимальное значения из диапазона R,G и B;
- H — тон (от 0 до 360);
- L — яркость (от 0 до 1);
- S — насыщенность (от 0 до 1).

$C = \text{MAX} - \text{MIN}$, тогда

$$H' = \begin{cases} undefined & \text{при MAX} = \text{MIN}, \\ \dfrac{G-B}{C} \bmod 6, & \text{MAX} = R, \\ \dfrac{B-R}{C} + 2, & \text{MAX} = G, \\ \dfrac{R-G}{C} + 4, & \text{MAX} = B. \end{cases}$$

$$H = 60° \cdot H',$$

$$L = \frac{\text{MAX} - \text{MIN}}{2},$$

$$S = \begin{cases} 0 \text{ при C} = 0, \\ \dfrac{C}{1 - |2L - 1|}. \end{cases}$$

Обратное преобразование выполняется через цепочку вре́менных переменных:

$$Q = \begin{cases} L \cdot (1.0 + S) & \text{при } L < 0.5, \\ L + S - (L \cdot S) & \text{при } L \geq 0.5. \end{cases}$$

$$P = 2.0 \cdot \text{L} - \text{Q},$$

$$H_k = \frac{H}{360} \quad \text{для приведения к интервалу [0,1],}$$

$$T_R = H_k + \frac{1}{3},$$

$$T_G = H_k,$$

$$T_S = H_k - \frac{1}{3}.$$

После этого для каждого цвета (обозначим его C) выполняем следующие действия:

$$T_c = \begin{cases} T_c & \text{при } T_c \text{ от 0 до 1}, \\ T_c - 1 & \text{при } T_c > 1, \\ T_c + 1 & \text{при } T_c < 0. \end{cases}$$

$$C = \begin{cases} P + (Q - P) \cdot 6.0 \cdot T_c & \text{при } T_c < \dfrac{1}{6}, \\ Q & \text{при } \dfrac{1}{6} \leq T_c < \dfrac{1}{2}, \\ P + (Q - P) \cdot 6.0 \cdot \left(\dfrac{2}{3} - T_c \right) & \text{при } \dfrac{1}{2} \leq T_c < \dfrac{2}{3}, \\ P & \text{при } T_c \geq \dfrac{2}{3}. \end{cases}$$

Теперь нужно преобразовать полученное значение компонента к нашему диапазону — от 0 до 255.

Для работы с изображением нам понадобятся средства для доступа к отдельным пикселям изображения. Для этого у объекта Picture есть методы GetPixel и setPixel — получения и записи соответственно отдельного пикселя по его координатам.

Метод Picture.GetPixel вернет объект Color со свойствами *R*, *G*, *B*, которые нам и нужны. Изменить эти свойства нельзя, поэтому новое значение мы «соберем» из компонентов с помощью функции *RGB* (*r*, *g*, *b*).

```
uses GraphABC;

const
 HLSMAX = 240; // Максимальные значения в HLS-модели
 RGBMAX = 255; // Максимальные значения в RGB-модели
 UNDEFINED = (HLSMAX*2) div 3;

type
  HSLColor = record // // описание точки в HLS-модели
     s,l,h : Single;
  end;
// Функция пересчета по приведенным формулам
function RGBtoHLS ( rgb : Color):HSLColor;
var
 cMax, cMin: single;
 r,g,b,c : single;
begin
  r := rgb.R/RGBMAX;
  g := rgb.g/RGBMAX;
  b := rgb.b/RGBMAX;
  cMax := Max( Max(r,g), b);
  cMin := min( min(r,g), b);
  c := cMax - cMin;

  Result.L := (cMax+cMin)/2;

 if (cMax = cMin) then
 begin
   Result.S := 0;
   Result.H := UNDEFINED;
 end
 else
 begin
     Result.S := c / ( 1-abs(2*Result.L-1)) ;
     if cMax = r then
```

```
      begin
         if g <b then
            result.h := ((g-b)/c +6)*60
         else
            result.h := ((g-b)/c)*60;
      end
      else
         if cMax = g then
            result.h :=  (((b-r)/c)+2)*60
         else
            result.h := (((r-g)/c)+4)*60;
 end;
end;
function convertC( tc,p,q : single):integer;
        // Вспомогательная функция
var
  t : single;
begin
  if tc > 1 then
     t := tc -1
  else
     if tc <0 then
        t:= tc + 1
     else
        t := tc;
  if 6*t < 1 then
    result := round((p+(q-p)*6*t)*RGBMAX)
  else
    if ((1 <= 6*t)and(2*t<1)) then
        result := round(q*RGBMAX)
    else
        if ((2*t>=1)and(3*t<2)) then
            result := round((p+(q-p)*6*(2/3-t))*RGBMAX)
        else
            result := round(p*RGBMAX);
end;
function HLStoRGB (hsl : HSLColor ) : Color ;
         // Пересчет из HLS-модели в RGB
var
  p,q,hk : single;
begin
 hk := hsl.h/360;
 if hsl.l < 0.5 then
```

```
  q := hsl.l*(1.0+hsl.s)
 else
  q := hsl.l+hsl.s-hsl.l*hsl.s;
 p := 2*hsl.l-q;

 Result := RGB(convertC(hk+0.333,p,q),
           convertC(hk,p,q),convertC(hk-0.333,p,q));
end;
procedure lightCorrect ( s,t : Picture; k : single );
          // Изменение яркости
var
  i,j : integer;
  pnt : Color;
  hls : HSLColor;
begin
  for j:=0 to s.Height-1 do // Переберем все пиксели
     for i:=0 to s.Width-1 do
     begin
       pnt := s.GetPixel(i,j); // Получили цвет в RGB
       hls := RGBtoHLS (pnt); // Пересчитали в HLS
       hls.l := hls.l*k; // Изменили яркость
       if hls.l > 1 then hls.l := 1;
       pnt:=HLStoRGB (hls); // Вернули назад в RGB
       t.SetPixel(i,j,pnt); // Поставили точку
                            в результате
     end;
end;

var // Основная программа
 p, r : Picture;
begin
  p := new Picture( 'example.jpg');
       // Исходное изображение
  r := new Picture( p.Width ,p.Height );
       // Результирующее
  lightCorrect(p,r,1.1); // Подняли яркость на 10%
  r.Draw(0,0); // Нарисуем его (рис. 71, 72)
  while true do
     ;

end.
```

Рис. 71. Результат применения: коэффициент 1.1

Рис. 72. Результат применения: коэффициент 2.0

Обработка растровых изображений. Фильтры

Множество растровых изображений мы получаем из реального мира: с помощью фотографий, видеосъемки, обработки значений датчиков и т. д. Полученное изображение, если и оказывается достаточного качества (что бывает далеко не всегда), часто требует изменений в зависимости от вашего замысла или предпочтений. Поэтому нужны средства, которые позволяют менять растровое изображение целиком. Основой очень значительной части таких средств являются ***фильтры***.

Фильтрация — это способ изменения изображения, при котором оно «пропускается» через преобразование и обрабатывается поточечно. Самый известный способ такого преобразования — применение *матрицы-свертки*, в которой цвет каждой точки является суммой цветов окружающих ее точек с некоторыми коэффициентами.

К каждой точке растра, находящейся в зоне действия фильтра, применяется некоторое действие для расчета ее нового цвета. Очень часто это действие основано на *весовой матрице,* которая называется *ядром преобразования*. Весовая матрица имеет нечетные длину и ширину. В ней указан вес (т. е. доля участия) пикселя в итоговом значении. Матрица накладывается на изображение так, чтобы ее центр (именно поэтому длина и ширина выбраны нечетными) совпал с изменяемым пикселем. Параметры пикселей изменяемой зоны, попавшие под матрицу, суммируются с весом, указанным в матрице. Результат нормируется (т. е. пересчитывается в доли от общего веса) и записывается в центральный пиксель. После этого матрица сдвигается на 1 пиксель, и все повторяется.

Вот простой фильтр «размытие» (англ. blur):

1	1	1
1	1	1
1	1	1

При наложении на изображение действовать это будет примерно так:

14	4	55	44	44	55
12	64	44	221	33	34
22	55	22	127	22	22
12	44	33	67	44	55
11	33	77	23	23	22
135	44	65	34	22	34

Цвет в центральной точке (темно-серая клетка) рассчитывается как сумма всех цветов окружающих точек (на экране это синий квадрат), деленная на суммарный вес, с округлением до целого: (64*1+44*1+221*1+55*1+22*1+127*1+44*1+33*1+67*1)/9 ≈ 75.

Такой расчет выполняется для каждого компонента цвета.

Теперь реализуем функцию наложения фильтра. С ее помощью мы будем получать объект-исходное изображение, объект-результат для изображения с наложенным фильтром и сам фильтр.

Изначально, для простоты и понятности, фильтр будет целочисленной матрицей 5*5 (размер мы оговорим константой). Напоминаем — этот размер должен быть нечетным.

Чтобы матрица поместилась целиком, будем накладывать ее, начиная с точки (2, 2). Перед наложением рассчитаем суммарный вес всех точек в матрице, чтобы выполнить нормирование. Результат применения — цвет точки — называется ***откликом фильтра***.

```
const
 filterSize = 5;
 half = 2;

type
 filter = array[0..filterSize-1,0..filterSize-1] of real;

procedure filterApply( s,t : Picture; filterMatrix :
                       filter);
var
 i,j,ip,jp : integer;
 dv : real;
 rr,rg,rb : real;
 pnt : Color;
begin
 dv := 0; // Рассчитаем делитель - для нормирования
 for i:=0 to filterSize-1 do
    for j:=0 to filterSize-1 do
       dv := dv + filterMatrix[i,j];
 if dv = 0 then dv := 1;
 for ip:=half to s.Height-half-1 do
    for jp:=half to s.Width-half-1 do
    begin
       rr :=0;
       rg :=0;
       rb :=0;
// Накладываем фильтр-свертку, захватывая пиксели вокруг
       for i:= -half to half do
         for j:= -half to half do
         begin
            pnt := s.GetPixel(jp+j,ip+i);
            rr := rr + pnt.R*filterMatrix[i+half,j+half];
            rg := rg + pnt.G*filterMatrix[i+half,j+half];
            rb := rb + pnt.B*filterMatrix[i+half,j+half];
```

```
            end;
          t.SetPixel(jp, ip, RGB(round(rr/dv), round(rg/dv),
                                 round(rb/dv)) );
      end;
  end;
```

В основной программе ничего особенного происходить не будет:

```
begin
 p := new Picture( 'example.jpg');
 r := new Picture( p.Width ,p.Height );
 filterApply(p,r,blur);
 r.Draw(0,0);
 while true do
    ;
end
```

Имя фильтра blur мы пока не описали — это и есть матрица свертки. Содержание матрицы очень простое:

1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1

то есть в этом фильтре цвет по всем компонентам будет смесью цветов окружающих точек. Опишем и применим его:
blur : filter := ((1,1,1,1,1),(1,1,1,1,1),(1,1,1,1,1),(1,1,1,1,1),(1,1,1,1,1)).

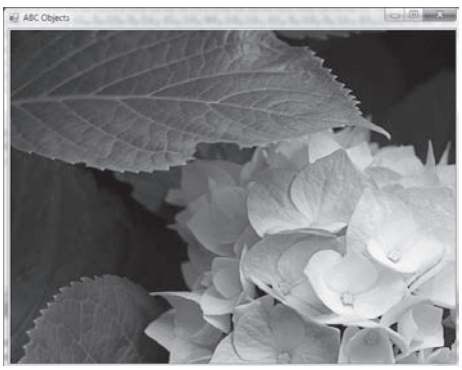

Рис. 73. Исходное изображение

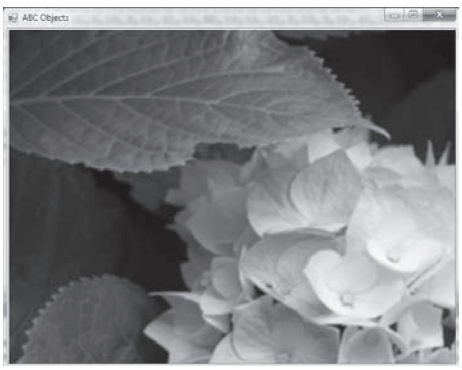

Рис. 74. Изображение, полученное в результате обработки фильтром

Результат работы фильтра можно оценить, сравнив изображения на рис. 73 и 74.

Вот еще несколько фильтров на пробу.

Мягкое размытие (Soft blur):

0	1	2	1	0
1	3	10	3	1
1	10	90	10	1
1	3	10	3	1
0	1	2	1	0

Резкость (Shavpen):

0	0	0	0	0
0	–0.25	–0.25	–0.25	0
0	–0.25	3	–0.25	0
0	–0.25	–0.25	–0.25	0
0	0	0	0	0

Результат их применения показан на рис. 75.

У приведенных выше фильтров может быть несколько вариантов, которые можно без труда найти в сети и сравнить результаты. Отметим, что «сила», с которой действует фильтр, зависит не от

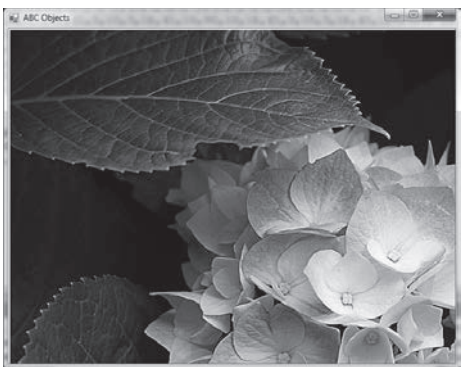

Рис. 75. Результат применения фильтров «мягкое размытие» и «резкость»

величины коэффициентов, а от фактических размеров матрицы. Чем больше пикселей задействовано, тем заметнее эффект.

Теперь доработаем нашу функцию так, чтобы увеличить возможности применения фильтров. Для этого будем передавать (а не рассчитывать) коэффициент нормирования и величину смещения — для компенсации.

Функция будет выглядеть примерно так:

```
const
   filterSize = 5;
   half = 2;

type
   filter = array[0..filterSize-1,0..filterSize-1] of real;

  procedure filterApply( s,t : Picture; filterMatrix :
                         filter; dv,off : integer);
var
   i,j,ip,jp : integer;
   rr,rg,rb : real;
   pnt : Color;
begin

   for ip:=half to s.Height-half-1 do
      for jp:=half to s.Width-half-1 do
      begin
         rr :=0;
         rg :=0;
         rb :=0;
         for i:= -half to half do
            for j:= -half to half do
            begin
               pnt := s.GetPixel(jp+j,ip+i);
               rr := rr + pnt.R*filterMatrix[j+half,
                                             i+half];
               rg := rg + pnt.G*filterMatrix[j+half,
                                             i+half];
               rb := rb + pnt.B*filterMatrix[j+half,
                                             i+half];
            end;
        t.SetPixel(jp, ip, RGB(test(round(rr/dv)+off),
                               test(round(rg/dv)+off),
```

```
                                        test(round(rb/dv)+off)) );
        end;
    end;
```

Обратите внимание, мы используем функцию test, задача которой — убирать значения меньше 0 и больше 255. Здесь эта функция очень проста: на все значения, меньшие 0, возвращает 0, на все, большие 255, — 255. Полагаем, читатели вполне в состоянии написать эту функцию сами.

Для повышения качества стоило бы не «обрубать» значения, а определить шкалу значений яркости и пересчитать все цвета пропорционально их положению на этой шкале.

С этими изменениями мы можем, например, повысить яркость изображения, просто добавив число ко всем компонентам цвета и применив нейтральный фильтр:

```
begin
      p := new Picture( 'example.jpg');
      r := new Picture( p.Width ,p.Height );
      filterApply(p,r,neutral,1,120);
      r.Draw(0,0);
      while true do
         ;

    end.
```

Еще один пример — *инверсия* цвета. Этот фильтр имеет только одно ненулевое значение — «1» в центре, а для инверсии цвета добавляет смещение «256».

Фильтров, приводящих к появлению самых разных эффектов, великое множество. Результаты фильтрации зависят от многих факторов: от вида матрицы, от того, как она накладывается, от способов создания отклика и т. д.

Возможностей для экспериментирования у вас масса. Вот несколько вариантов:

1) свертка накладывается на все пиксели, включая край, причем на край она накладывается частично — с пересчетом делителя нормирования;
2) перед наложением край изображения следует масштабировать так, чтобы размеры увеличились на величину свертки;
3) фильтр накладывается прямо на изображение — используются и уже обработанные, и еще не обработанные точки.

Задания для самостоятельной работы

1. Размытие по гауссиане может быть выполнено, в частности, с помощью такой свертки:

$$\begin{pmatrix} 1 & 2 & 3 & 2 & 1 \\ 2 & 4 & 5 & 4 & 2 \\ 3 & 5 & 6 & 5 & 3 \\ 2 & 4 & 5 & 4 & 2 \\ 1 & 2 & 3 & 2 & 1 \end{pmatrix}.$$

   Примените его и сопоставьте результат с «обычным» размытием.
2. Усреднение — один из практически важных фильтров. В этом фильтре все значения в матрице сортируются, и отклик фильтра — среднее значение. Реализуйте такой фильтр и проанализируйте его поведение для серых и цветных изображений.
3. Для получения эффекта «Акварельность» изображения применяют фильтр усреднения, а после него — фильтр резкости. Реализуйте этот комбинированный фильтр.

Графические примитивы. Растеризация

Следующая важная задача, которую решают при разработке средств машинной графики, — *отрисовка примитивов*. Практически все современные аппаратные средства в машинной графике — растровые, т. е. так или иначе опираются на координатную сетку пикселей. Но любой графический примитив, например отрезок, состоит (а точнее — проходит) из нескольких пикселей. Возникает вопрос: какие из них нужно закрасить?

Приведем пример одного из основных алгоритмов, используемых в машинной графике в этом случае, — алгоритма построения отрезка. На этом примере можно увидеть некоторые основные приемы решения таких задач.

Построение отрезка для нас означает заполнение тех точек растра, которые находятся на отрезке, соединяющем точки с заданными координатами. Очевидно, что точно вычислить эти координаты нельзя, поскольку не может быть «дробных» точек. Для простоты будем считать, что координаты определены именно в точках растра (т. е. их не нужно пересчитывать).

Самый распространенный алгоритм рисования отрезков получил название *алгоритма Брезенхема*. Основная идея этого алгоритма — не использовать вычисления с плавающей точкой, а просто «пройти» по одной из осей и определить для каждой координаты в пределах линии, нужно ли для нее «сдвинуть» значение второй координаты.

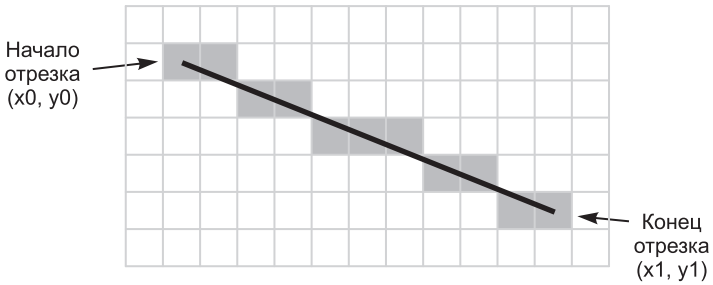

Рис. 76. Пример построения отрезка

Например, рассмотрим алгоритм для рисования линии на рис. 76, приняв координаты концов отрезка $x0$, $y0$ и $x1$, $y1$. Чтобы упростить понимание алгоритма, сначала запишем его на псевдокоде с использованием дробных чисел, считая, что исходное изображение — объект Image, а все пиксели в нем — внутренняя матрица Pixel.

```
Deltax = x1 – x0
Deltay = y1 – y0
error = 0
deltaErr = Deltay/Deltax
y = y0
for x = x0 to x1
   Image.Pixel[x,y] = 1
   error = error + deltaErr
   if error > 0.5
         y = y +1
         error = error – 1
```

Таким образом, мы вычислили коэффициент наклона и увеличивали y, когда прошли больше половины пикселя (т. е. накопили ошибку в переменной *error* больше, чем 0,5). Чтобы перейти к целым числам, просто умножим дробные числа на Deltax и на 2 (там, где использовался коэффициент 0,5):

```
Deltax = x1 – x0
Deltay = y1 – y0
error = 0
deltaErr = Deltay
y = y0
for x = x0 to x1
   Image.Pixel[x,y] = 1
```

```
error = error + deltaErr
if 2*error > Deltax
      y = y +1
      error = error – Deltax
```

Чтобы получить полный алгоритм, нужно учесть и другие случаи наклона отрезка. Обычно их учитывают, меняя местами координаты и направления сдвига точек. Аналогично можно реализовать алгоритмы построения окружностей и эллипсов с осями, параллельными осям координат.

Алгоритм можно использовать и для большего разнообразия линий, например задав шаблон в виде битовой карты (рисовать точку или не рисовать), получать пунктирные линии, использовать не просто точку, а фигуру сложной формы (кисть) и т. д.

Решим задачу, используя этот алгоритм: написать процедуру построения линии, нарисованной треугольниками или любыми фигурами, вписанными в размер 4*4 пикселя (рис. 77).

Типовые встроенные функции такого сделать не позволяют: даже рисование произвольной кистью приведет к появлению сплошной линии. Но мы знаем, как строится линия в принципе, — используем это для рисования линии не пикселями, а штампами 4*4. Сначала напишем процедуру рисования штампом:

```
type
   stamp = array[0..3,0..3] of byte;

procedure DoStamp(s:Picture; x,y : word; c : Color;
                  brush : stamp);
var
   i,j : byte;
```

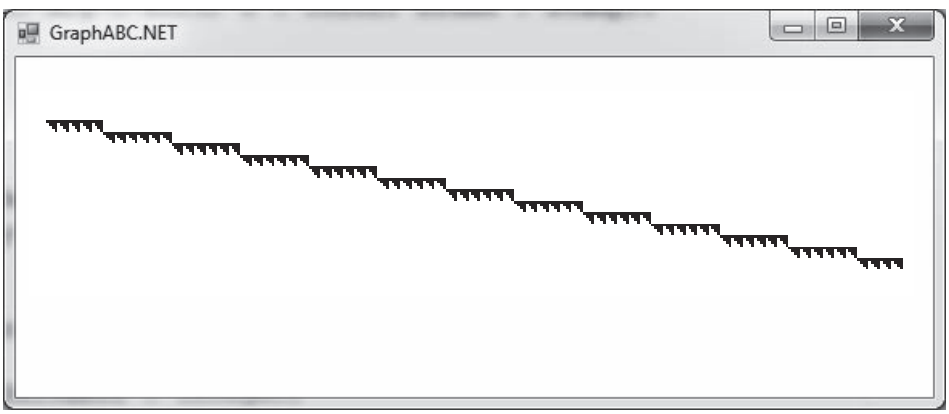

Рис. 77. Линия, нарисованная треугольниками

```
begin
   for i:=0 to 3 do
      for j:=0 to 3 do
         if brush[j,i]=1 then
           s.SetPixel(x+j,y+i,c);
end;
```

Как видим, ничего сложного — штамп будет задан массивом 4*4, в котором стоит 1 там, где нужна точка цвета «c».

Теперь рисование линии:

```
procedure MyLine(s : picture; x0,y0,x1,y1 : Word;
                 c : Color; brush : stamp);
var
   Deltax, Deltay, error, DeltaErr : integer;
   x ,y, stepx, stepy : integer;
begin
   if (x1 < x0 ) then
      stepx := -4
   else
      stepx := 4;
   if (y1 < y0 ) then
      stepy := -4
   else
      stepy := 4;

   Deltax := abs(x1 - x0);
   Deltay := abs(y1 - y0);
   error  := 0;
   deltaErr := Deltay;
   y := y0;
   x := x0;
   while x <> x1 do
   begin
      DoStamp(s,x,y,c,brush);
      error := error + deltaErr;
        if 2*error > Deltax then
         begin
             y := y +stepy;
             error := error - Deltax;
        end;
        x := x + stepx;
  end;
end;
```

Единственное существенное отличие реализации от алгоритма на псевдокоде — это учет того, что точки могут быть заданы в любом порядке. Это определит направление шага по каждой координате. Шаг равен 4, поскольку это размер штампа.

Основная программа, как и в случае с фильтром, проста:

```
var
  p : Picture;
  triangle : stamp := ( (1,0,0,0),(1,1,0,0),(1,1,1,0),
                        (1,1,1,1));
begin
  p := new Picture( 500,500);
  MyLine(p,400,100,100,50,rgb(20,0,0),triangle);
  p.Draw(0,0);
  while true do
      ;
end.
```

Как и в прошлом примере, простора для экспериментов здесь много: размер и форма штампа, использование фона, режимы наложения и т. д.

Рисование отрезков — далеко не единственный алгоритм Брезенхема. Аналогично строятся, например, эллипсы и дуги эллипсов.

Основы векторной графики. Преобразования координат

Исторически первые задачи, которые решались в компьютерной (машинной) графике, — это задачи создания изображения (графика, чертежа и т. д.). При подготовке таких рисунков очень часто возникает задача преобразования координат для отображения, преобразования объектов: масштабирования, поворота и т. д. Такие задачи решаются методами аналитической геометрии с помощью преобразования координат.

Одна из базовых задач, которую мы решим таким способом, — пересчет координат из математической системы координат в экранную. Как вы знаете, в математике 0 располагается в центре или слева внизу, оси направлены слева направо и снизу вверх. На экране же точка отсчета расположена слева вверху, а ось y направлена вниз. Придется пересчитать.

Предположим, что у нас есть область отображения, заданная координатами углов ($X0$, $Y0$), ($X1$, $Y1$). Нужно отобразить в ней объект (например, график функции), координаты которого укладываются в пределах ($x0$, $y0$), ($x1$, $y1$).

Тогда предварительно рассчитаем коэффициенты:
$kx = (X1 - X0)/(x1 - x0)$; $ky = (Y1 - Y0)/(y1 - y0)$.
Координаты точки на экране можно рассчитать так:

$$\begin{cases} X = \left[(x - x_0) \cdot kx + X_0\right]; \\ Y = \left[(y - y_0) \cdot ky + Y_0\right]. \end{cases}$$

Квадратными скобками мы обозначаем операцию получения целой части числа. Если важно соблюдать масштаб, то коэффициент выбирается один — наименьший.

Фигуру, заданную координатами точек (всех или только ключевых), можно легко преобразовывать. Сдвиг точки — это просто добавление чисел (координат вектора) к координатам:

$$\begin{cases} x' = x + a; \\ y' = y + b. \end{cases}$$

Увеличить или уменьшить фигуру (и не обязательно оставлять ее на месте) можно, просто умножив координаты ее точек на коэффициент. Если фигура должна сохранить логическое положение на «листе», то координаты предварительно придется пересчитать в систему координат с центром в середине фигуры ($x0$,$y0$), определяемым как среднее арифметическое координат:

$$\begin{cases} x' = x_0 + k * (x - x_0); \\ y' = y_0 + k * (y - y_0). \end{cases}$$

Наконец, поворот точки на угол α вокруг начала координат описывается системой уравнений:

$$\begin{cases} x' = x * \cos\alpha - y * \sin\alpha; \\ y' = x * \sin\alpha + y * \cos\alpha. \end{cases}$$

Если поворот нужно выполнить не вокруг начала координат, можно использовать перенос координат.

Важное свойство таких преобразований — сохранение формы фигуры, т. е. линия преобразуется в линию. Поэтому можно пересчитать только координаты начала и конца отрезка, а промежуточные точки рассчитывать заново не надо.

С помощью этой теории решим задачу синтеза достаточно сложного изображения — многолучевой звезды, имеющей семь лучей.

Начнем с постановки задачи. Звезда представляет собой симметричный относительно центра многоугольник, имеющий два радиуса: длинный и короткий. Радиусы чередуются через равные углы, соединяя вершины.

Пусть первая вершина лежит на оси ординат, тогда координаты остальных вершин можно вычислить поворотом на нужный угол.

Для реализации нам потребуется несколько вещей:

1) типы для хранения координат — математических и экранных;
2) функции для выполнения поворота точки на угол;
3) средства рисования ломаной линии по точкам.

Первая часть ничего сложного из себя не представляет:

```
type
   pointR = record
      x,y : real;
   end;
   pointS = record
      x,y : word;
   end;
   screenCalc = record
      kx,ky : real;
      x0,y0 : word;
      xm0,ym0 : real;
   end;
```

Здесь имеются две простые структуры с координатами и структура для хранения данных пересчета (по формулам, которые мы приводили).

Теперь можно написать функции преобразования:

```
// Расчет коэффициентов преобразования в экранные
   координаты
function calcScreen( x0,y0, x1,y1 : word;
                     xm0,ym0,xm1,ym1 : real )
                     :screenCalc;
begin
   Result.kx := (x1-x0)/(xm1-xm0);
   Result.ky := (y1-y0)/(ym1-ym0);
   Result.x0 := x0;
   Result.y0 := y0;
   Result.xm0 := xm0;
   Result.ym0 := ym0;
end;

// Перевод математических координат в экранные
function toScreen( math : pointR; screen :
                   screenCalc ) : pointS ;
var
```

```
   res : pointS;
begin
   res.x := round((math.x-screen.xm0)*screen.
                  kx+screen.x0);
   res.y := round((math.y-screen.ym0)*screen.
                  ky+screen.y0);
   toScreen := res;
end;

// Поворот точки на угол (в радианах)
function Rotate( math : pointR; angle : real ) :
                 PointR ;
var
   res : pointR;
begin
   res.x := math.x*cos(angle)-math.y*sin(angle);
   res.y := math.x * sin(angle) + math.y * cos(angle);
   rotate := res;
end;

// Масштабирование относительно центра координат
function Scale( math : pointR; k : real ) : PointR ;
var
   res : pointR;
begin
   res.x := math.x*k;
   res.y := math.y*k;
   Scale := res;
end;
```

Как видите, ничего сложного в этих функциях нет: все эти формулы вы встречали либо здесь, либо на уроках геометрии. Функция масштабирования нам понадобится позже.

С хранением и отрисовкой линии чуть сложнее. В общем виде мы не знаем, сколько точек будет в ломаной. Эту задачу мы решим с помощью списка, добавив в раздел type:

```
   PList = ^PointList;
   PointList = record
     point : PointR;
     next  : PList;
   end;
```

И в функции:

```
function addPoint( coord : pointR; next : PList ) :
                   PList;
begin
   new (Result);
   Result^.point := coord;
   Result^.next := next;
end;
```

И наконец, функция отрисовки линии:

```
procedure Polyline( p : Picture; vertex : PList;
                    screen :screenCalc );
var
   test : PList;
   p1,p2 : pointS;
begin
  // Построение линии
    p1 := toScreen(vertex^.point,screen );//Первая
                точка списка
    test := vertex^.next; //Со второй точки
    while test <> nil do // Пройдем по всему списку
    begin
      p2 := p1;
      p1 := toScreen(test^.point,screen );
      p.Line(p2.x,p2.y,p1.x,p1.y);
      test := test^.next;
    end;
    p2 := toScreen(vertex^.point,screen );
    p.Line(p2.x,p2.y,p1.x,p1.y); // Замкнем ломаную
end;
```

Обратите внимание! Перед собственно рисованием линии рассчитываются ее экранные координаты.

В секцию Const добавим определение количества точек-вершин:

```
const
  StarVertex = 14;
```

Теперь можно привести основную программу:

```
var
   star, test : PList;
```

```
    i : word;
    start1,start2 : pointR;
    screen : screenCalc;
    p : Picture;
begin

    start1.x := 10;
    start1.y := 0;
    start2.x := 5;
    start2.y := 0;

    star := addPoint(start1,nil);
    star := addPoint(Rotate(start2,2*Pi/
                    StarVertex),star);

   for i:=3 to StarVertex do
    if i mod 2 = 1 then
     star := addPoint(Rotate(start1,(2*Pi/
                      StarVertex)*(i-1)),star)
    else
     star := addPoint(Rotate(start2,(2*Pi/
                      StarVertex)*(i-1)),star);

    screen := calcScreen(0,0,500,500,-36,-36,36,36);
    p := new Picture( 500,500);
    PolyLine(p,star,screen);
    p.Draw(0,0);
    while true do
       ;
end.
```

Результат исполнения представлен на рис. 78.

Усложним задачу: построим 14 таких звезд, вложенных друг в друга. Решить такую задачу только средствами растровой графики трудно, а векторной — ничего сложного.

Изменим часть, посвященную собственно рисованию линии, — выполним после каждой отрисовки ее масштабирование:

```
for i:=1 to 14 do
   begin
    PolyLine(p,star,screen);
    test := star; // Пройдем по всему списку
    while test <> nil do
```

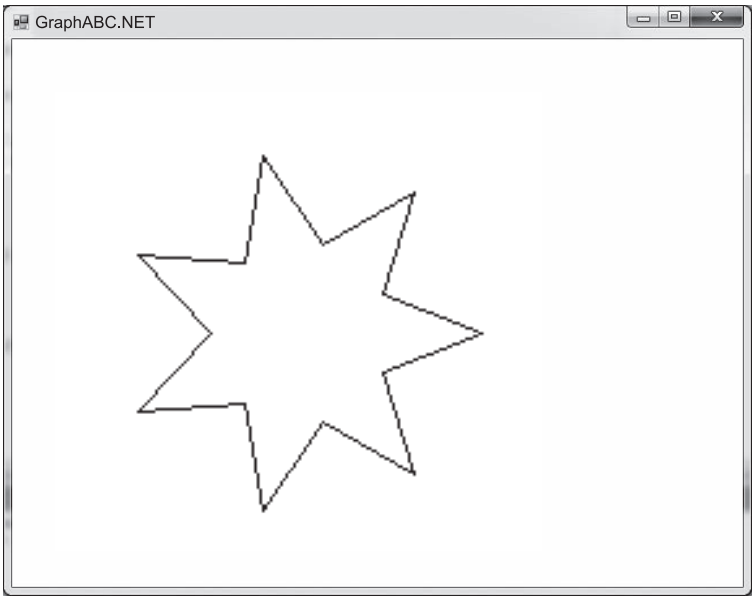

Рис. 78. Результат построения семилучевой звезды

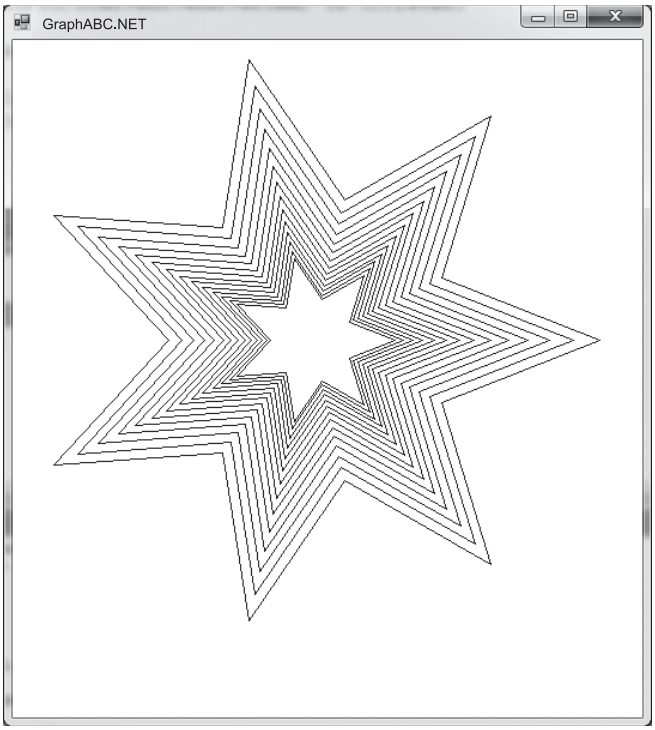

Рис. 79. Вложенные звезды

```
    begin
      test^.point := Scale(test^.point,1.1 );
      test := test^.next;
    end;
  end;
```

Результат исполнения показан на рис. 79.

Как видите, используя этот аппарат, можно строить сложные изображения, опираясь на простые средства, без потери качества изображения выполняя целый ряд преобразований. Стоит заметить, что эти возможности будут напрямую зависеть от вашей математической подготовки.

Задания для самостоятельной работы

1. Нарисуйте с помощью описанных средств фигуры-стрелки по кругу диаметром 10 математических единиц.
2. Реализуйте набор звезд, в котором будет плавно меняться цвет: каждая звезда рисуется одним своим цветом.
3. Реализуйте картинку из 10 вложенных квадратов. Вершины внутреннего квадрата — середины сторон наружного.

Часть 2. Основы трехмерного моделирования

В учебнике мы кратко рассказали о том, как с помощью компьютера создаются реалистично выглядящие изображения, передающие объем. В этой части практикума мы познакомимся с некоторыми приемами построения таких изображений, в основном с первым этапом создания трехмерного изображения — созданием ***модели***.

Мы будем использовать для этого бесплатно распространяемую программу Google Sketchup версии 8. Эта программа предназначена для так называемого «эскизного» моделирования, т. е. в ней изначально нельзя создать действительно сложную модель и действительно фотореалистичное изображение[1] (тем более анимированное). Зато она удобна для подготовки вполне убедительного эскиза-заготовки и освоения основ работы в трехмерном пространстве.

Построенное изображение будет напоминать архитектурный эскиз, что вполне подходит для модели здания, предмета мебели, посуды и т. п.

---

[1] Существует множество дополнительных модулей, предоставляющих эту возможность.

Для начала рассмотрим простой пример — схематичное построение модели дачного дома. Заранее договоримся, что это только примерная модель: у дома не будет подвала, крыльца, окна останутся незастекленными и т. п.

Основные инструменты и приемы

Первое, с чего мы начнем работу, — это выбор шаблона. С помощью шаблона определим два основных параметра пространства, в котором будем работать: способ измерения и вид проекции. Для начала выберем самый простой шаблон: единицей измерения в нем будут метры, а для преобразования трехмерного представления в двумерное представление на экране — перспективную проекцию (рис. 80). Такой шаблон подходит для моделирования зданий, что нам и требуется. Обратите внимание, возле начала координат на рис. 81 схематично обозначена фигурка человека для облегчения оценки размеров.

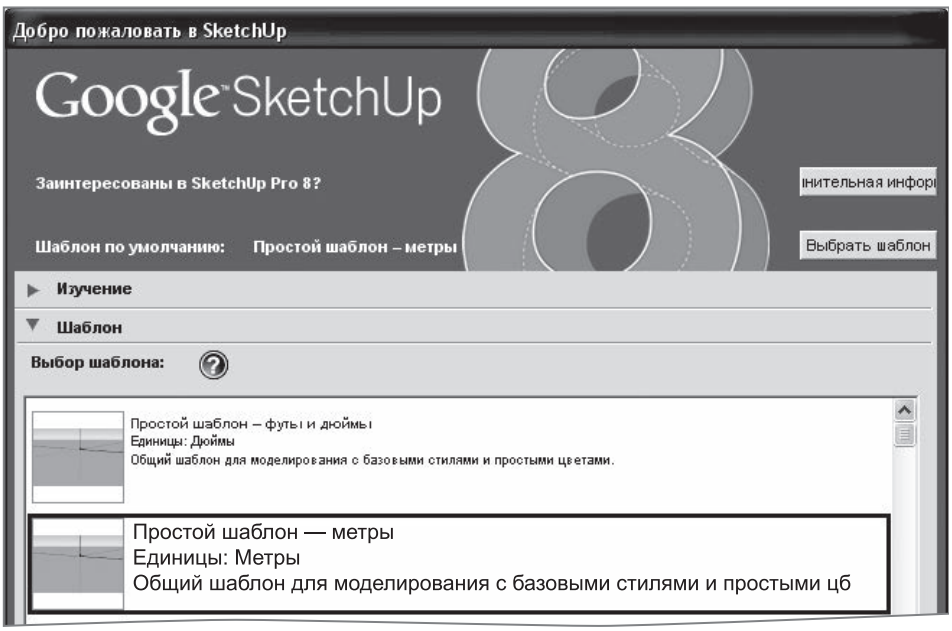

Рис. 80. Выбор шаблона

Все построения будут происходить в пространстве, определенном тремя осями. В математике их обозначают x, y и z, но, поскольку мы будем постоянно вращать оси, буквами для обозначения пользоваться неудобно. Проще ориентироваться по цветам: на

Рис. 81. Обозначение координатных осей: *1* — зеленая; *2* — красная; *3* — синяя

экране — зеленая и красная оси определяют «землю», а синяя — высоту. В нашем шаблоне дополнительно обозначено еще и небо.

Основные инструменты

Для работы используются инструменты (рис. 82), как обычно размещенные на панели слева. Панель может находиться и сверху и вообще быть отдельным окном, но изначально она слева. Инструмент можно выбрать на панели щелчком мыши, действовать он начнет после щелчка мышью в рабочей области.

В зависимости от назначения некоторые инструменты будут действовать, пока кнопка нажата, некоторые потребуют нескольких щелчков — для указания нескольких параметров.

Первое, с чем нужно познакомиться, — со способами «перемещения» в пространстве. От положения камеры (нашего глаза) будет зависеть и то, что мы увидим, и то, как будет пониматься программой наша попытка что-то построить.

К основным средствам перемещения относятся: **Орбита**, **Панорама** и **Масштаб**.

Поскольку менять положение камеры в пространстве часто приходится во время работы, а переключать инструмент быва-

Обозначение инструмента	Действие	Описание работы с инструментами
	Выбрать	Этот инструмент позволяет выбрать один или несколько объектов для последующих операций. Если вы выбираете один элемент – достаточно на нем щелкнуть, если несколько – надо растянуть рамку или щелкнуть на каждом с прижатой клавишей Ctrl
	Прямоугольник	Позволяет построить плоский прямоугольник
	Заливка	Позволяет задать внешний вид поверхности
	Линия	Отрезок между двумя точками
	Дуга	Дуга некоторого плоского овала по трем точкам: двум точкам опорного диаметра и радиусу
	Окружность	Плоская окружность внутри растянутого прямоугольника
	Переместить	Перемещение выбранного объекта
	Тяни/толкай	Создание объемного объекта "вытягиванием" плоского объекта вверх
	Повернуть	Поворот объекта
	Ведение	Создание объемного объекта "проводкой" плоского объекта вдоль траектории
	Масштабирование	Изменение размеров объекта
	Смещение	Создание нового объекта смещением существующего
	Рулетка	Измерения
	Орбита	"Облет" сцены
	Панорама	Смещение точки просмотра сцены
	Масштаб	Изменение масштаба просмотра

Рис. 82. Основные инструменты программы

ет крайне неудобно, для быстрого перехода к этим инструментам можно использовать третью кнопку мыши. Чаще всего это колесико, но не у всех манипуляторов есть такая возможность.

Во-первых, вращение колесика мыши дает приближение или удаление (инструмент ).

Во-вторых, если нажать и удерживать третью кнопку, будет действовать инструмент **Орбита**, т. е. можно будет облетать объект.

В-третьих, если при нажатой третьей кнопке нажать левую кнопку мыши, включится инструмент  **Панорама** и можно будет переместить точку обзора.

После того как вы отпустите колесико, система вернется к действующему инструменту.

«Полетайте» вокруг начала координат и добейтесь примерно такого положения, которое показано на рис. 83.

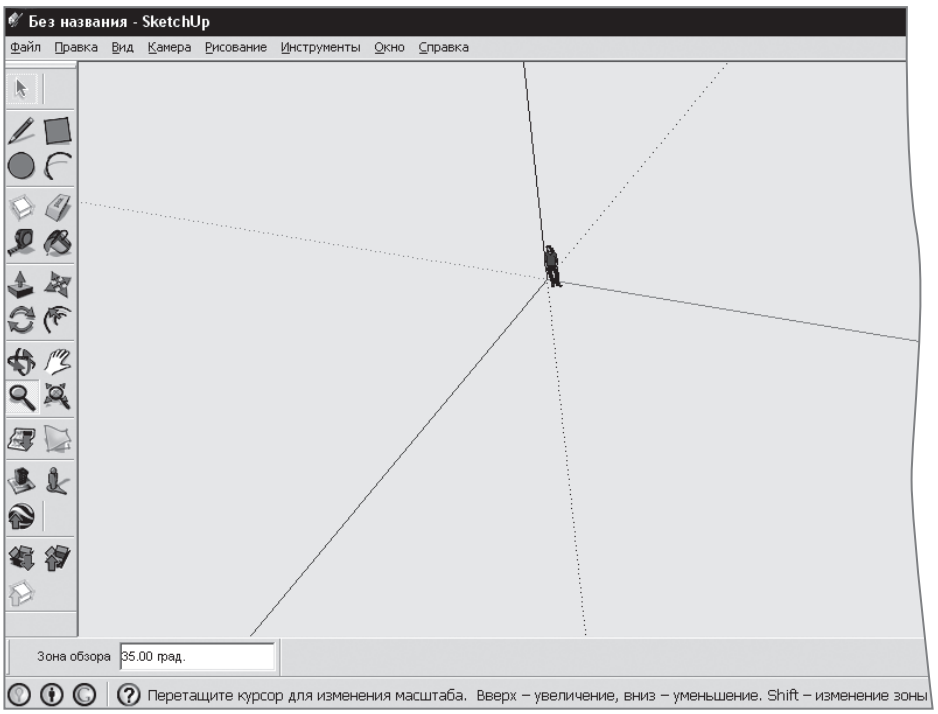

Рис. 83. Стартовое положение при начале построения

Базовые приемы построения

Начнем с примерной постройки плоского плана. Чтобы толщина стен в нашем доме была одинаковой и углы комнат соответствовали друг другу, нам понадобятся *вспомогательные линии*, т. е.

линии, не входящие в модель, но помогающие ее строить. Такие линии обозначаются пунктиром.

Построим их с помощью инструмента **Рулетка** от красной и зеленой осей. Для этого:

- наведем курсор **Рулетки** на ось так, чтобы там появился красный квадрат;
- нажмем кнопку и отведем линию от оси (рис. 84).

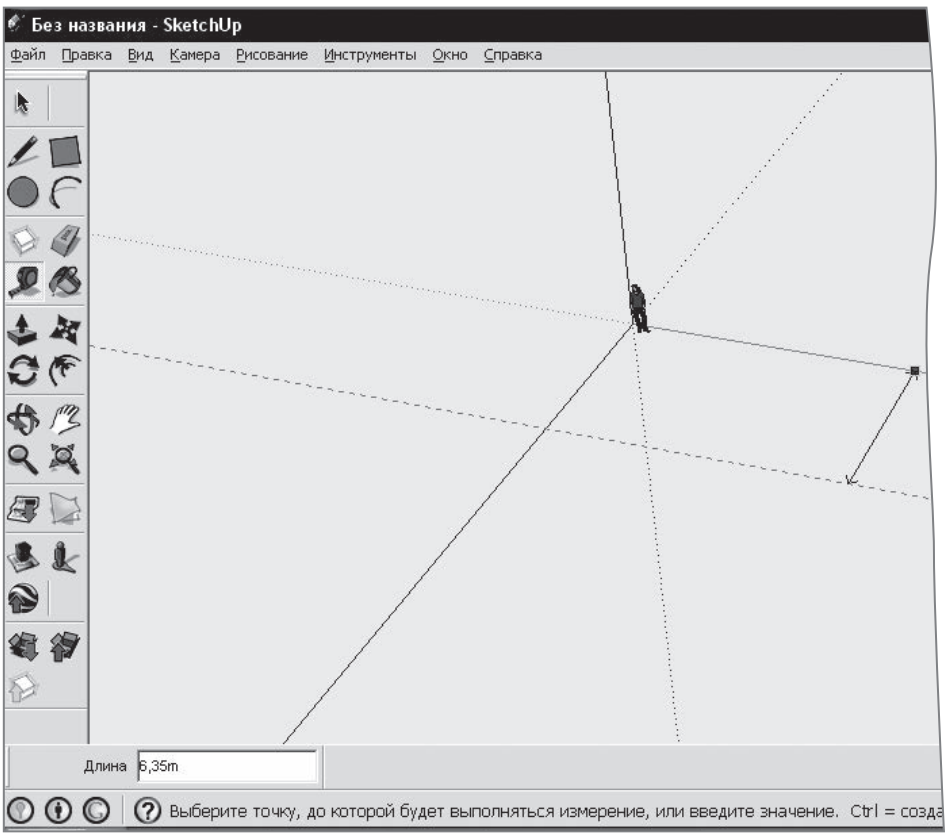

Рис. 84. Построение вспомогательной линии

Схема, конечно, весьма приблизительная, и жить в таком «доме» вряд ли удобно, но для демонстрации подойдет.

Инструментом **Рулетка** можно отмерить и расстояние на имеющемся объекте: от его края, оси, точки пересечения и т. д. При этом на ребре останется дополнительная точка, а не пунктирная прямая.

Результат должен быть примерно таким, как на рис. 85:

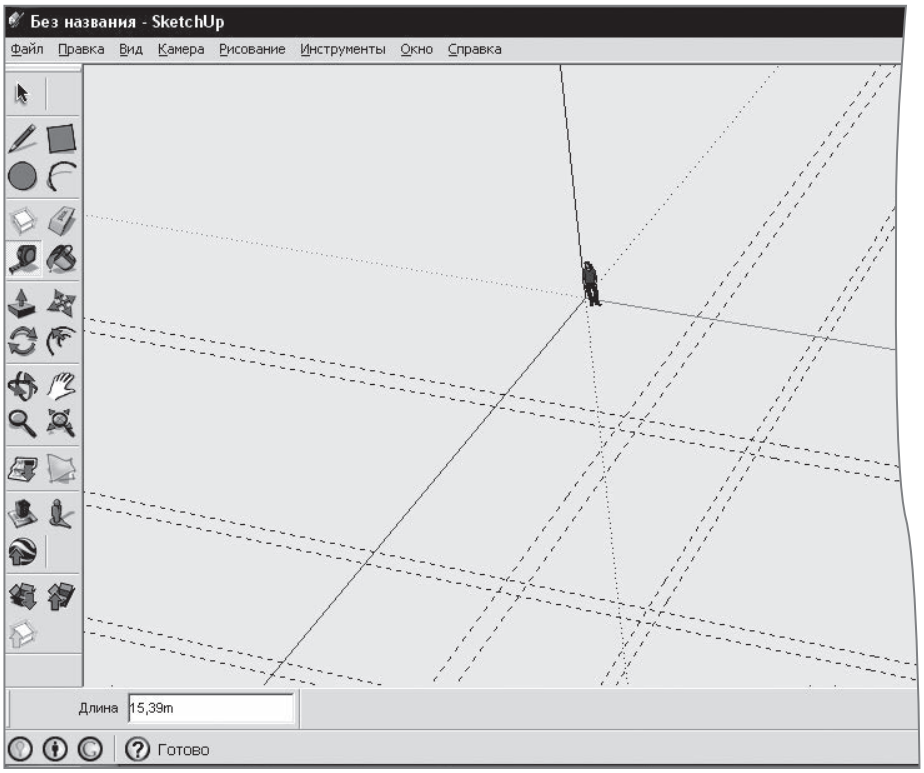

Рис. 85. Продолжение построения вспомогательных линий

Теперь воспользуемся инструментом  «Прямоугольник» и построим «комнаты». Для этого от крайнего левого пересечения вспомогательных линий до крайнего правого растянем прямоугольник — внешние стены (рис. 86).

В каждом внутреннем прямоугольнике между вспомогательными линиями построим отдельные прямоугольники-комнаты так, чтобы между ними осталось расстояние — так образуются стены.

При выполнении построений фигуры составляются из плоских многоугольников – граней[2]. Каждая грань имеет ребра (отрезки, которые ее ограничивают) и площадь. Эти элементы можно изменять и редактировать отдельно друг от друга. Грани и ребра возникают и при пересечении двух фигур.

---

2 Да и круги тоже. Количество отрезков можно менять.

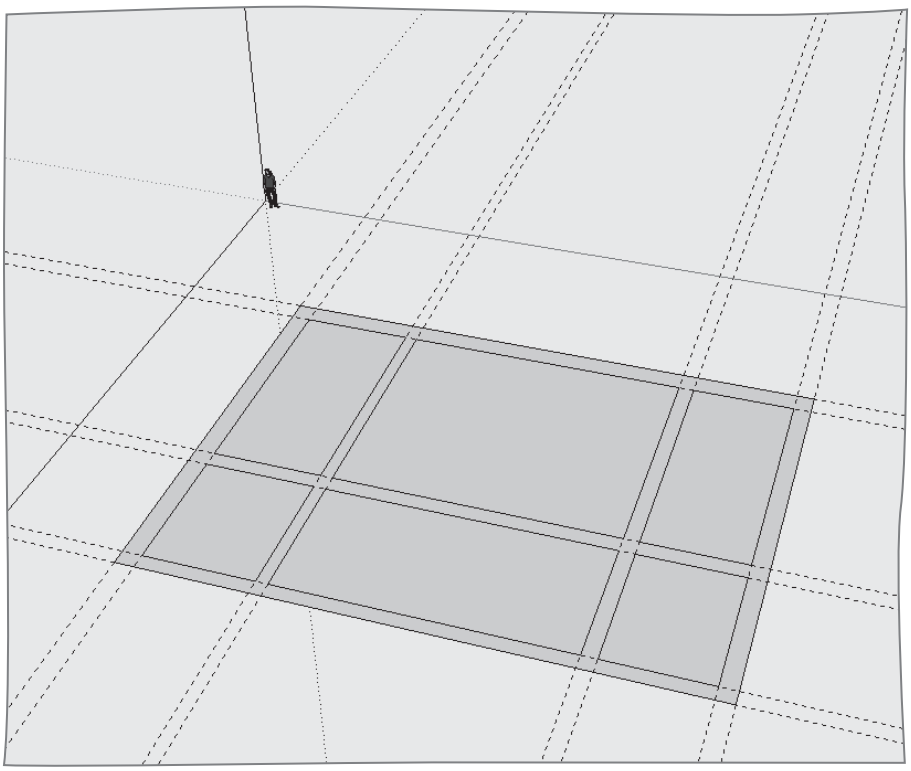

Рис. 86. Построение границ внешних стен

Обратите внимание! При построении инструменты «притягиваются» к вспомогательным линиям, ребрам, точкам пересечения, серединам ребер — это помогает ориентироваться при построении объемных фигур.

Итак, мы получили совершенно плоский план. «Облетев» его, вы увидите, что эти прямоугольники не имеют никакой толщины.

Удалим внутренние прямоугольники, оставив только стены: выберем их инструментом **Выбрать** и нажмем **Del** (рис. 87).

С помощью таких же прямоугольников обозначим дверные проемы[3]: подготовим места дверных проемов, удалив внутреннее пространство прямоугольников.

Теперь можно придать этому плоскому плану объем — воспользуемся инструментом **Тяни/толкай** и «вытолкнем» стены

[3] Авторы упрощают построение, поэтому двери будут иметь высоту стен, но не до абсурда: окна будут строиться по-другому.

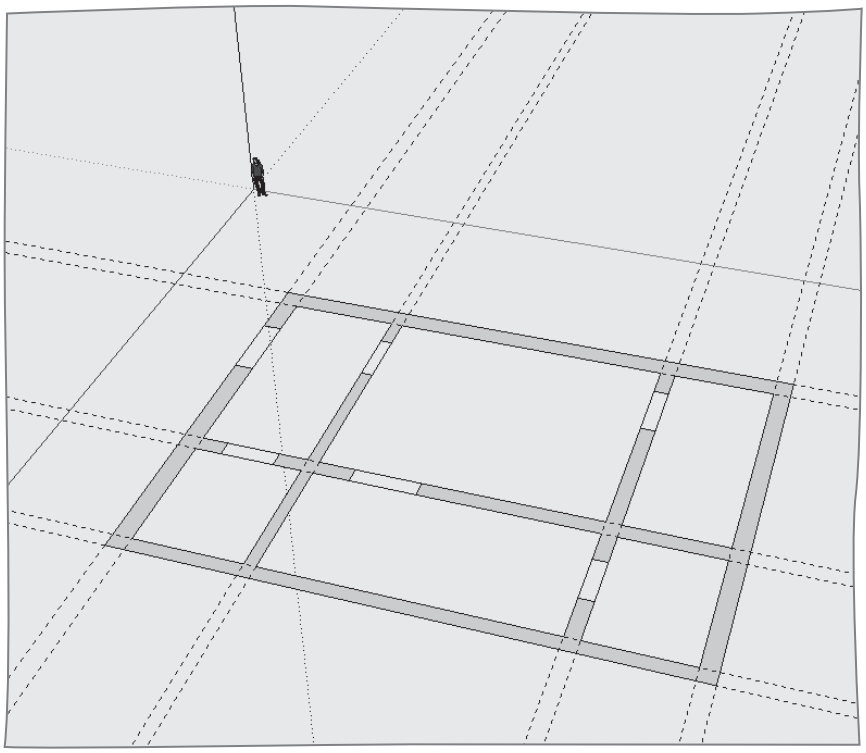

Рис. 87. Удаление внутренних прямоугольников

вверх. Этот инструмент создает объемную фигуру, протягивая плоскую фигуру перпендикулярно (не обязательно вверх).

Выбрав инструмент  **Тяни/толкай**, наведем его на стену. Нажмем левую клавишу мыши и, не отпуская ее, «вытянем» стены вверх примерно на 5 м (рис. 88). Расстояние (и другие параметры, когда мы ими пользуемся) показывается в поле ввода в нижней левой части экрана.

> Обратите внимание! Во время выталкивания в строке **Расстояние** внизу менялась высота выталкивания. Сразу после того, как мы вытянем стены, зададим им точную высоту: нажмите на клавиатуре цифру 5 и **Enter**.

Стены у нас есть. Перейдем к крыше. Разверните дом фасадом к себе и постройте осевую вспомогательную линию, на этот раз — от синей оси. Чтобы линия не оказалась вдали от стен (традиционно крыша связана со стенами), установите эту линию по уже имеющейся вспомогательной линии. Если хотите получить точное

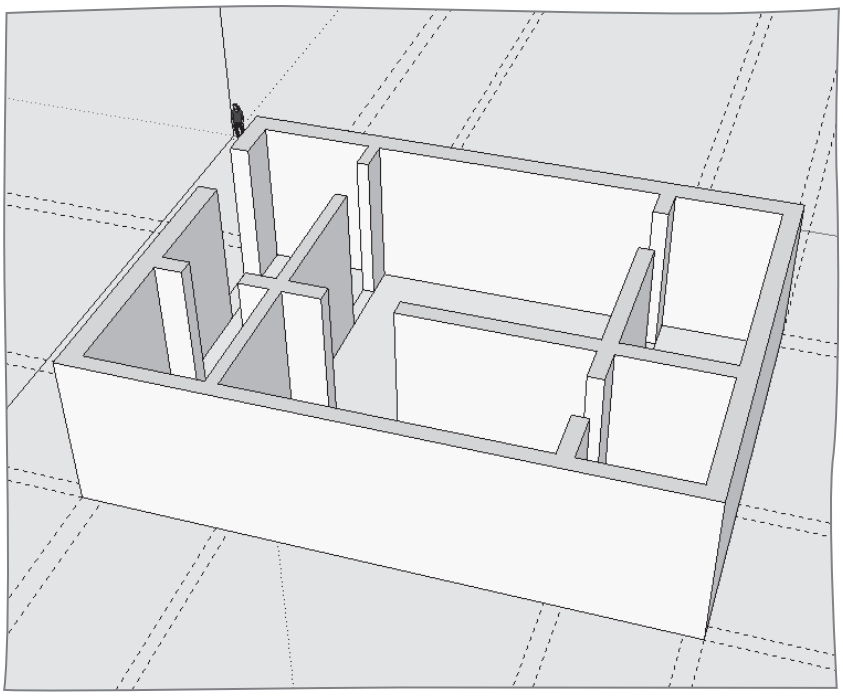

Рис. 88. Вытягивание стен вверх

положение, рассчитайте расстояние и введите его, как вводили высоту стен.

Теперь построим заготовку крыши (рис. 89). С помощью инструмента  **Линия** построим треугольник:

- выбрав инструмент **Линия**, нажмем клавишу мыши на левом верхнем углу стены и вытянем линию до вертикальной вспомогательной линии на желаемую высоту конька крыши;
- от получившейся точки вытянем линию до правого угла крыши;
- от правого угла протянем линию к левому так, чтобы под курсором появилась зеленая точка, которая покажет, что мы попали в начало.

Если все сделано верно, получится треугольник (см. рис. 89). При этом появятся замкнутые фигуры: «потолок» и «дверь». Дверь нужно удалить, а «потолок» можно оставить.

Выберем инструмент  **Смещение**:

- наведем его на треугольник, нажмем клавишу мыши и уменьшим его внутрь — появится уменьшенная копия;

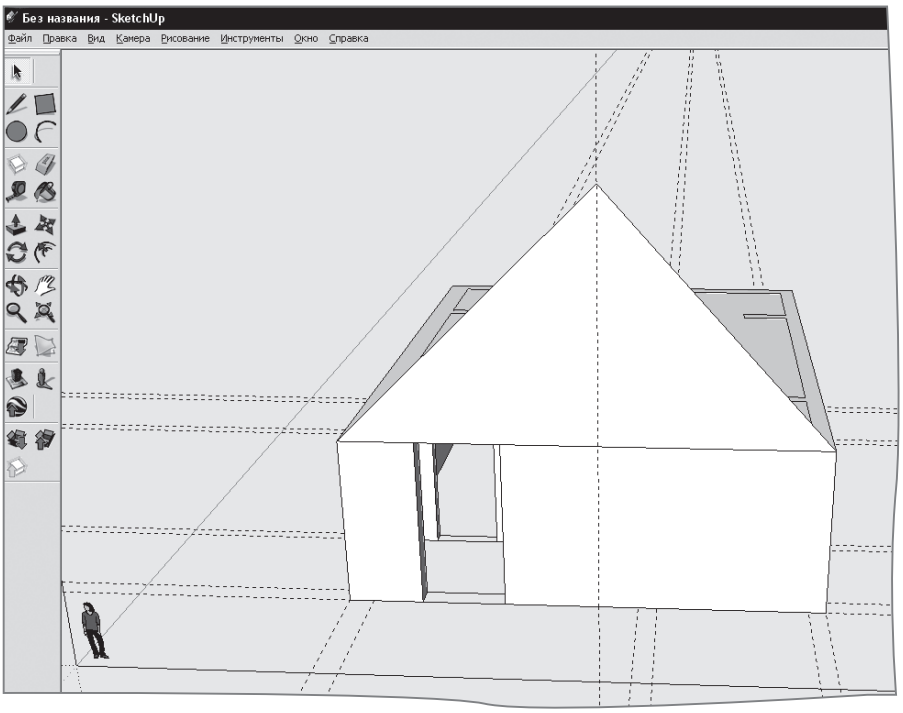

Рис. 89. Заготовка крыши

- внутренний треугольник уберем;
- результат «вытянем» до противоположной стены (рис. 90).

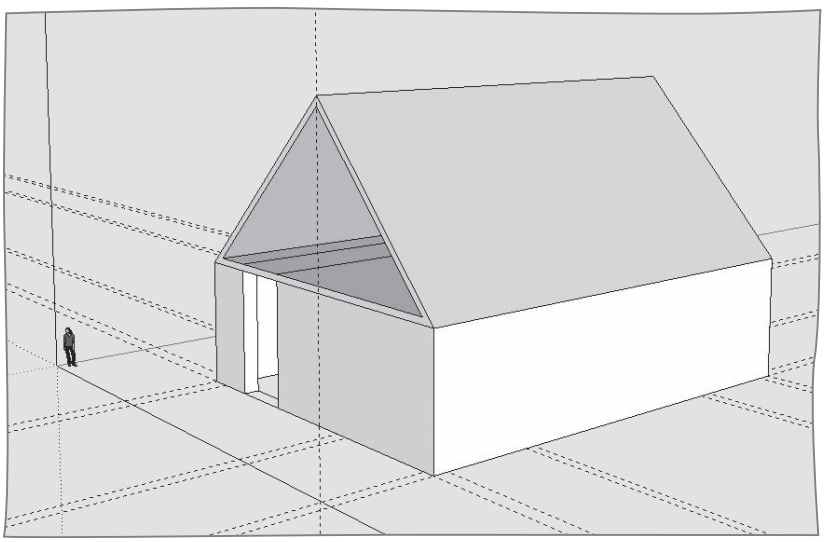

Рис. 90. Построение фронтона и скатов крыши

Сдвигать придется в несколько этапов — на границах комнат сдвиг будет останавливаться. Чуть позже мы увидим, что это удобная особенность.

Теперь построим окна. Чтобы окна оказались на одной высоте, построим вспомогательную линию. Для этого  **Рулеткой** отмерим 1,5 м от линии «земли».

Обратите внимание! Отмерять и строить линию надо на стене, не попадая на посторонние пересечения, иначе линия пройдет не там, где надо (программа неверно определит положение вашей линии в пространстве).

Построим прямоугольник, который и будет выступать в роли заготовки окна (рис. 91).

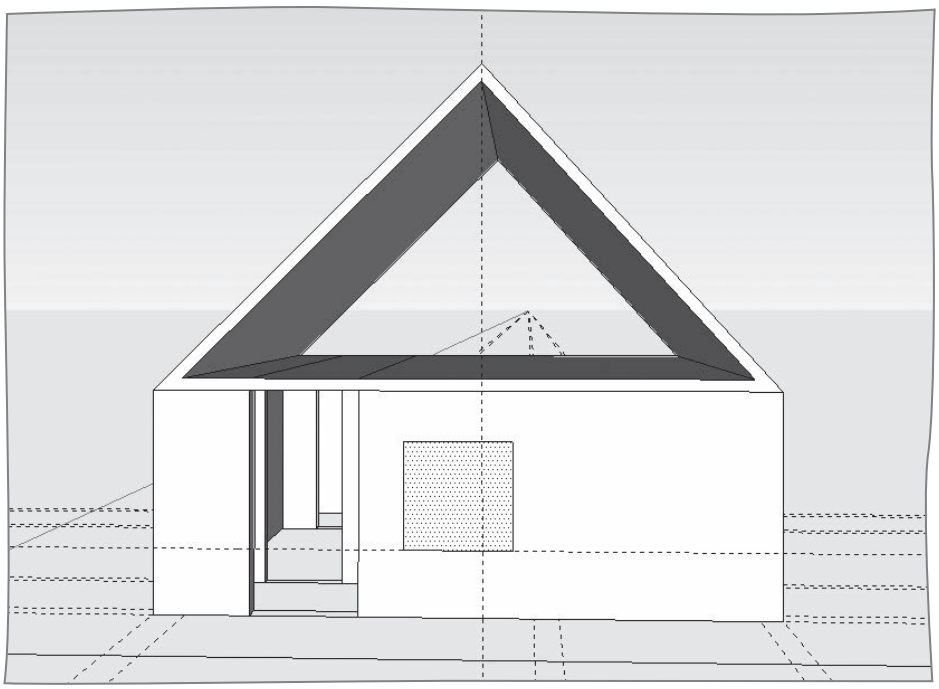

Рис. 91. Построение заготовки окна

Обратите внимание! Место выбрано так, чтобы окно не оказалось на внутренней стене, которую мы строили по вспомогательным линиям.

Чтобы все окна в доме были одинаковыми:

- выберем прямоугольник-заготовку и скопируем ее в буфер обмена (меню **Правка**, команда **Копировать**);
- перенесем камеру к другой стене и построим на ней такую же вспомогательную линию в 1,5 м от уровня земли;
- с помощью команды **Вставить** перенесем полученный прямоугольник на стену, разместив его нижнюю сторону на вспомогательной линии, и щелкнем клавишей мыши, чтобы его закрепить.

Второе окно создадим с помощью инструмента  **Переместить**.

- выберем прямоугольник этим инструментом;
- нажмем на клавиатуре клавишу **Ctrl** для создания копии (у курсора появится знак «+»);
- не отпуская **Ctrl**, сместим окно по вспомогательной линии.

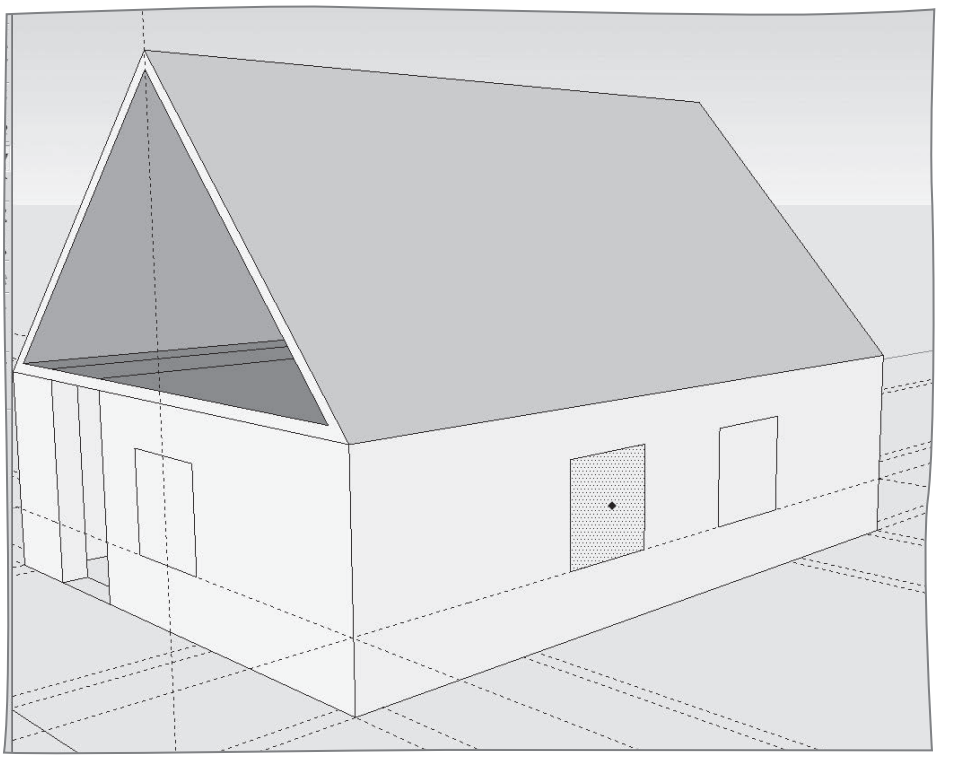

Рис. 92. Построение заготовок окон на другой стене

Аналогично построим заготовки и на других стенах (рис. 92). Чтобы из заготовок действительно получились оконные проемы,

снова воспользуемся инструментом **Тяни/толкай**. Вытолкнем оконный проем внутрь, пока не появится сообщение **На грани**. После этого клавишу мыши можно отпустить, и появится проем (рис. 93).

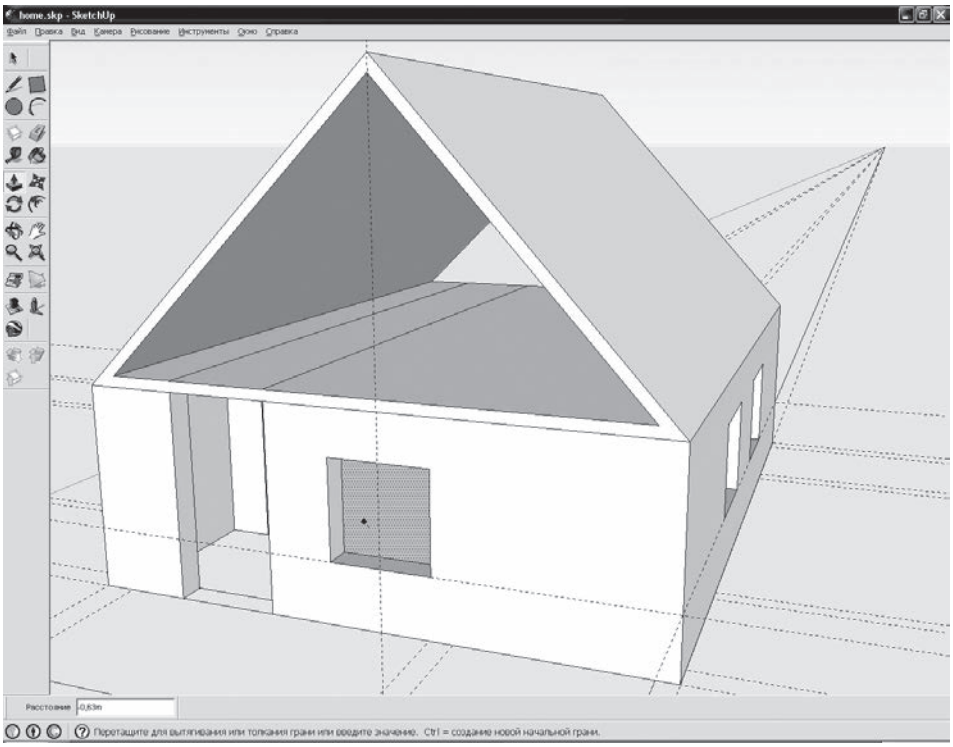

Рис. 93. Построение оконных проемов

Остальные окна сделаем таким же образом.

Добавим к нашему дому трубу для отвода дыма. Для этого:

- рядом с домом нарисуем круг — инструментом **Окружность** сначала отметим центр круга (первый щелчок мыши), а потом вытянем радиус размером 0,5 м;
- точно так же, как мы поступили при построении крыши, сместим круг внутрь ( ) и удалим середину;
- полученное кольцо[4] сдвинем вверх с помощью инструмента **Тяни/толкай**.

[4] Ничто не мешает вам сделать то же самое с прямоугольником и получить трубу, похожую на сложенную из кирпича.

Чтобы не мучиться с размещением трубы на крыше, подбирая ее положение в трехмерном пространстве, подготовим разметку (рис. 94): от конька или нижнего края крыши **Рулеткой** отводим первую вспомогательную линию, от торцевого края крыши — вторую, перпендикулярную. Размещаем их так, чтобы точка пересечения находилась примерно в месте расположения трубы.

Теперь выделим трубу инструментом **Выбрать** и перенесем ее инструментом **Переместить** на подготовленное перекрестье. Подберем высоту ее положения так, чтобы нижний край трубы касался потолка (да, при этом труба пройдет через крышу — это и является ее основным назначением). Если высоты трубы не хватит, снова потяните вверх верхнее кольцо инструментом **Тяни/толкай**.

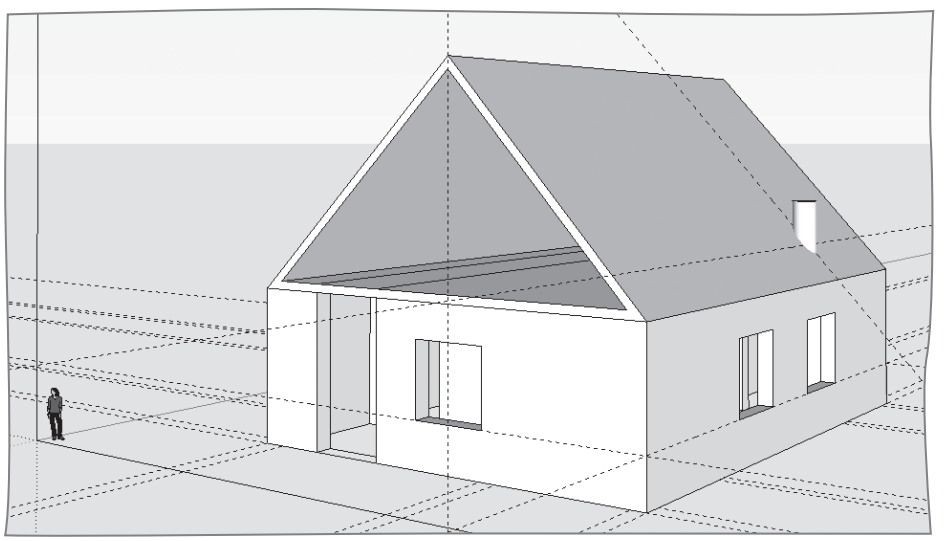

Рис. 94. Построение трубы

Чтобы привести дом к «жилому» виду, нарисуем с фронтальной стороны веранду под крышей. Для этого построим опорную конструкцию из брусьев (авторы выбрали толщину 25 см), построив вспомогательные линии от края крыши и стенку.

Единственно новым в построении будет способ, которым надо убирать лишнее от этой стенки (рис. 95):

- построим стенку как выдвинутый прямоугольник;
- выберем стенку и кусок крыши, как показано на рис. 95;
- нажмем правую кнопку мыши и выберем команду **Перекрытие граней — с моделью** (эта команда разделит стенку на части);

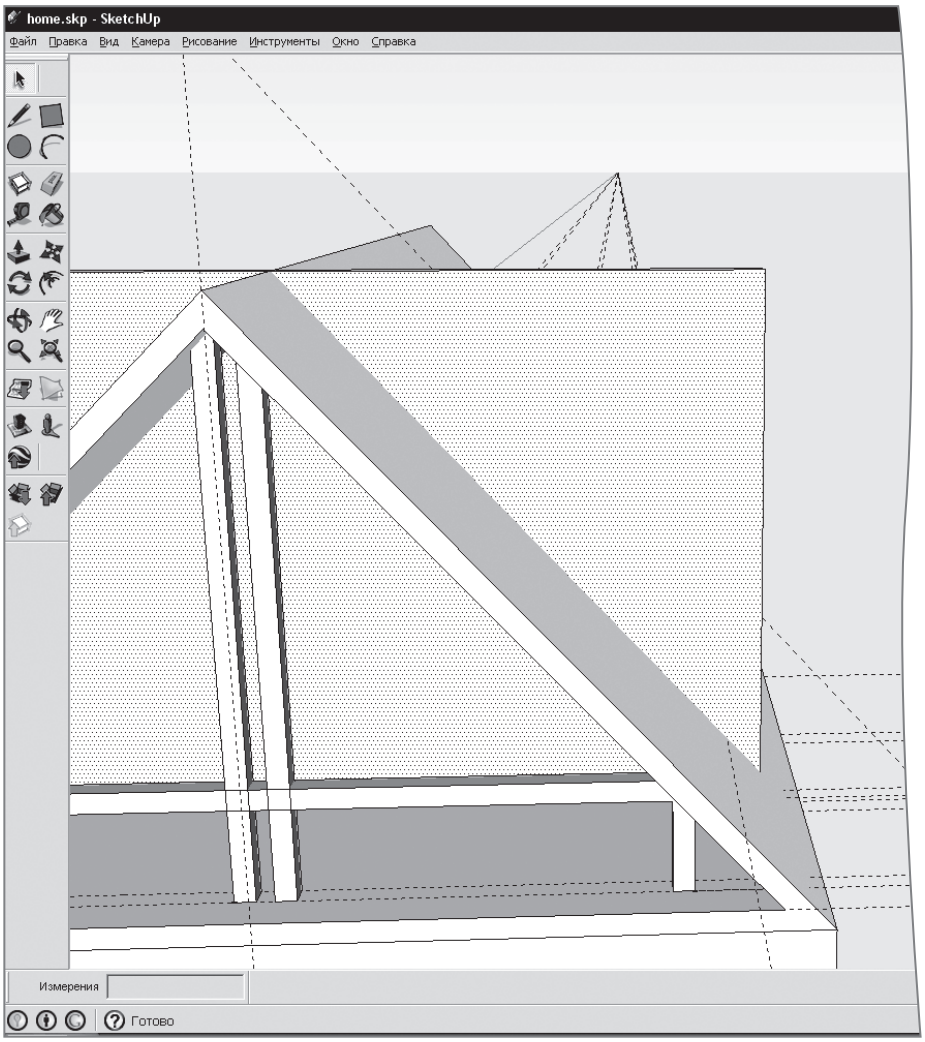

Рис. 95. Построение веранды с фронтальной стороны дома

- «выступающие» треугольники сдвинем до нулевой толщины инструментом **Тяни/толкай;**
- удалим лишние ребра — остатки от пересечения со стеной на крыше.

Аналогично могут быть построены рамы и стенка под крышей с другой стороны дома.

Следующий этап придания дому более реалистичного вида — задание текстур, описывающих материалы. Делается это с помощью инструмента **Заливка:**

- выберем инструмент  **Заливка** — появится окно выбора текстур;
- выберем раздел **Кровля**, а в нем — текстуру **Черепица**;
- щелкнем инструментом на плоскости крыши.

Зададим текстуры черепицы, дерева и камня на нашем доме, после чего включим режим **Тени** в меню **Вид** (рис. 96).

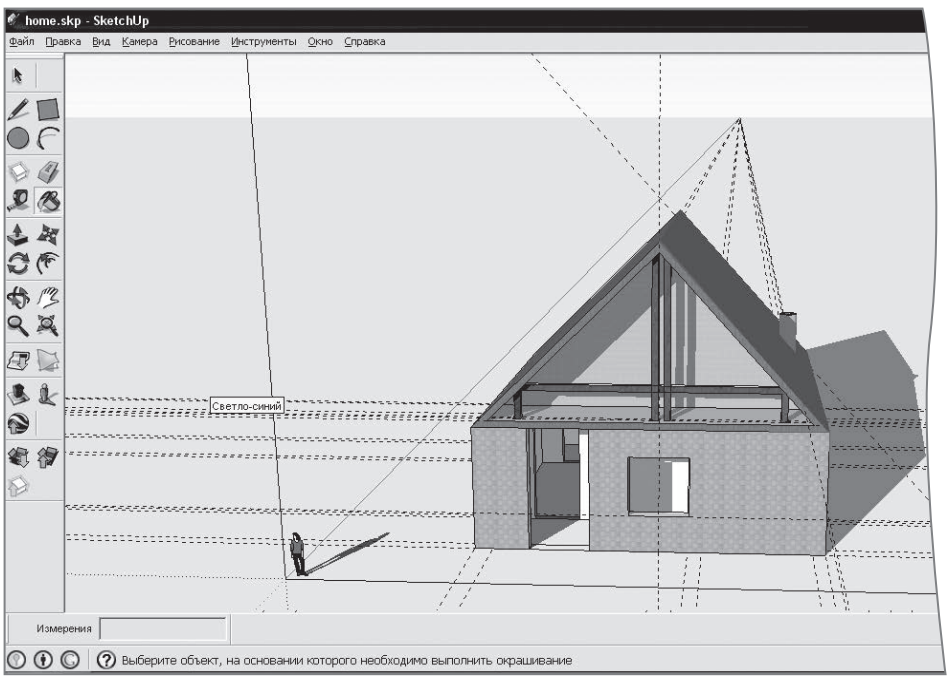

Рис. 96. Задание текстуры

Очевидно, что результат больше похож не на готовый, а на недостроенный дом (хотя бы потому, что пол и рамы в окнах отсутствуют, а на веранде нет двери[5]), но надеемся, что вы уже имеете представление о том, как можно создать изображение более высокого качества.

Задания для самостоятельной работы

1. Постройте модель стула, состоящую из параллелепипедов.
2. Постройте дом, аналогичный уже сделанному, но с учетом подвала, размеров окон и дверей.

---

[5] Автору довелось видеть внутренний балкон в здании, на который архитекторы-профессионалы в проекте забыли предусмотреть хотя бы один выход, — так что ничего оригинального в этом нет. Лестницу пришлось доделывать спешно перед сдачей.

Движение по траектории, компоненты

Следующая модель, которую мы подготовим, может стать неплохим дополнением к предыдущей — нарисуем стул[6]. Для проектирования выберем другой шаблон — «Проектирование изделий и деревообработка». Основной единицей измерения в нем будут миллиметры, «Небо» и «Земля» не обозначаются.

Чтобы воспользоваться некоторыми новыми инструментами, включим расширенную панель инструментов. Для этого:

- в меню **Вид/Панели инструментов** уберем галочку **Начальная**;
- там же поставим галочку **Расширенная**.

Начнем с сидения (рис. 97). За основу возьмем обычный прямоугольник. Построим его инструментом  **Прямоугольник** параллельно осям. Вытянем фигуру вверх инструментом  **Тяни/толкай**.

На верхней грани построим  дугу так, чтобы она почти касалась одной из коротких сторон. Для этого первую точку щелчком мыши поставим на одной стороне, вторую — на другой, и вытянем дугу в сторону середины отрезка.

Удалим получившийся внешний сегмент, «утопив» его, как обычно, инструментом  **Тяни/толкай**.

Чтобы прорисовать отдельно раму и отдельно обивку, инструментом  **Смещение** сделаем уменьшенную копию верхней поверхности. Получим примерно такой результат (см. рис. 97).

Теперь с помощью нового инструмента сделаем верхний край рамы полукруглым. Для этого:

- развернем камеру так, чтобы видеть прямоугольную сторону;
- нарисуем дугу примерно в толщину рамы к середине боковой стороны;
- выберем инструмент  **Ведение** и щелкнем сначала на получившемся сегменте, а потом последовательно наведем (только наведем!) курсор на каждое из ребер верхней части сидения, начав с угла, который выделили, пока не дойдем снова до начального угла; выбранные в качестве траектории ведения ребра будут становиться красными;
- доведя до угла, щелкнем мышью — сегмент будет удален по всей длине ребер.

6 Это описание фактически повторяет урок № 4 из видеоуроков Google по использованию Sketchup.

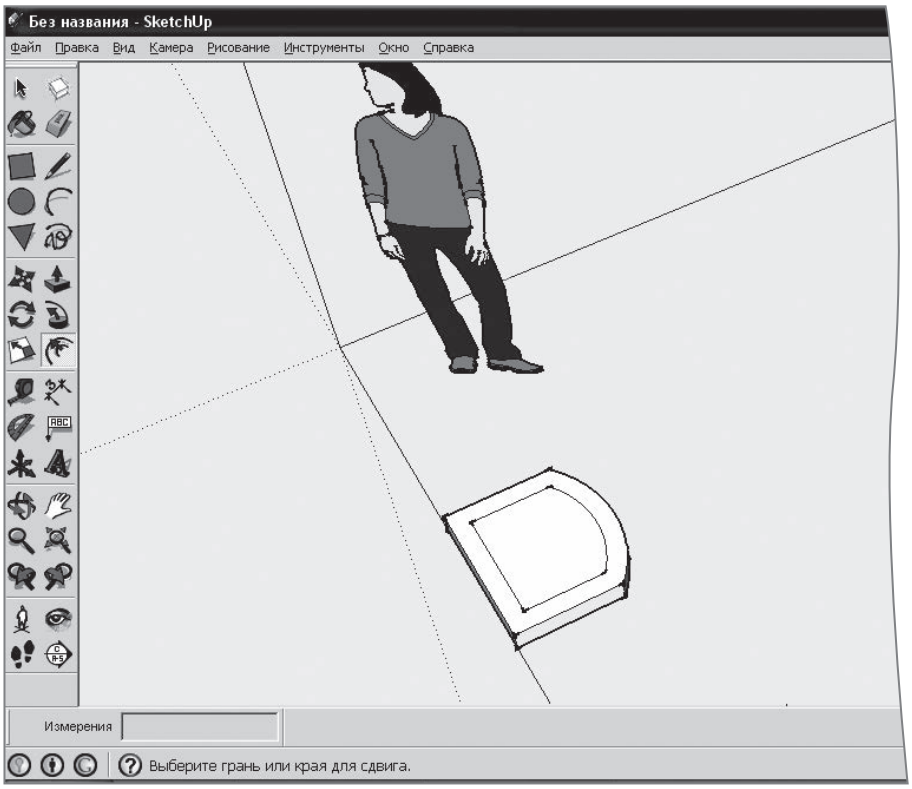

Рис. 97. Начало построения сиденья стула

Должно получиться примерно, как на рис. 98.

Теперь нарисуем ножки нашего стула. Одна из характерных особенностей ножек хорошего стула — их одинаковость, поэтому при построении ножек мы используем специальную возможность Sketchup — создание компонентов. Один и тот же компонент можно использовать в нескольких местах, а редактировать — только в одном. Для этого:

- развернем камеру на низ стула;
- построим квадрат — заготовку ножки — со стороной примерно 40 мм;
- выделим площадь квадрата и добавим (инструментом **Выбрать** с прижатой клавишей **Shift**) стороны;
- в контекстном меню (правая клавиша мыши) вызовем пункт **Создать компонент**;
- назовем его «Ножка стула» и нажмем кнопку **Создать** (рис. 99);

Обратите внимание! Компонент получил свою собственную систему координат.

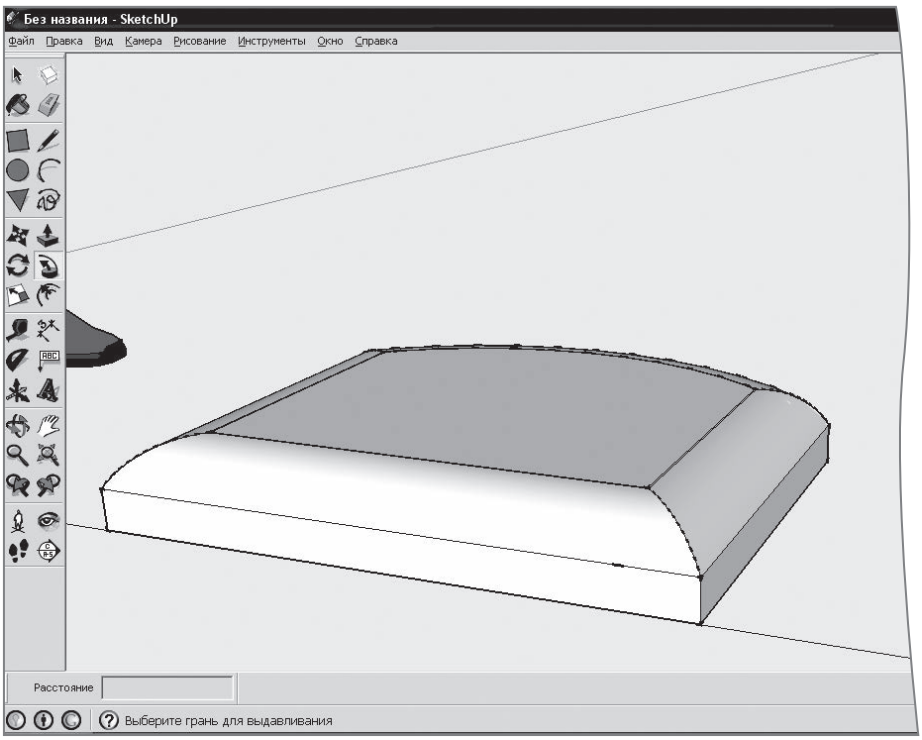

Рис. 98. Придание сиденью объема

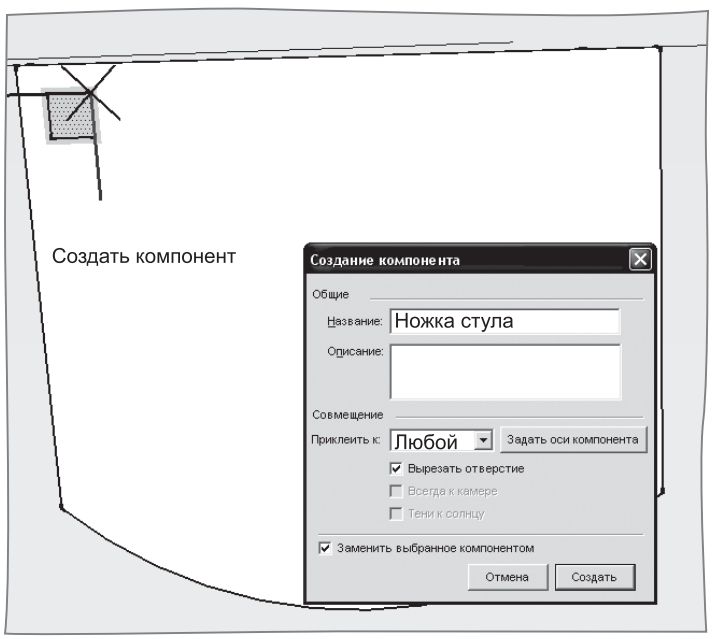

Рис. 99. Начало построения ножки стула

- создадим сдвигом копию (инструмент **Переместить** с прижатой клавишей **Ctrl**) ножки с другой стороны;

Обратите внимание! Если компонент сдвигается правильно, он сдвигается вдоль оси, и это отмечается красной или зеленой пунктирной линией.

- инструментом **Выбрать** добавляем исходный компонент (с прижатой клавишей **Shift**) и сдвигаем копию обеих ножек к оставшейся стороне;
- поворачиваем камеру и дважды щелкаем на любой удобной для редактирования ножке;
- инструментом **Тяни/толкай** сдвигаем ножку на 500 мм вниз.

Обратите внимание! Изменились все 4 ножки. Кроме того, стул затенен, мы редактируем только компонент.

Сделаем ножку фигурной. Для этого:

- нарисуем инструментами **Линия** и **Дуга** на ножке профиль, по которому будем «обрезать»;
- развернем камеру так, чтобы видеть весь квадрат основания ножки снизу;
- инструментом **Ведение** выделим профиль для обрезки и проведем вдоль каждой грани сечения;
- выйдем из редактирования компонента, щелкнув мышью снаружи.

В процессе работы это должно выглядеть примерно, как на рис. 100.

Теперь нарисуем спинку стула (рис. 101):

- на краю сидения нарисуем круг — заготовку опоры спинки;
- выделим круг и его окружность, создадим компонент;
- выберем компонент инструментом **Выбрать**;
- выберем инструмент **Поворот** и щелкнем этим инструментом в середине противоположной стороны — отметим центр поворота;
- этим же инструментом щелкнем в середине компонента — отметим радиус поворота;
- нажмем клавишу **Ctrl**, чтобы сделать копию, а не перенос во время поворота;
- повернем компонент примерно на 20 градусов.

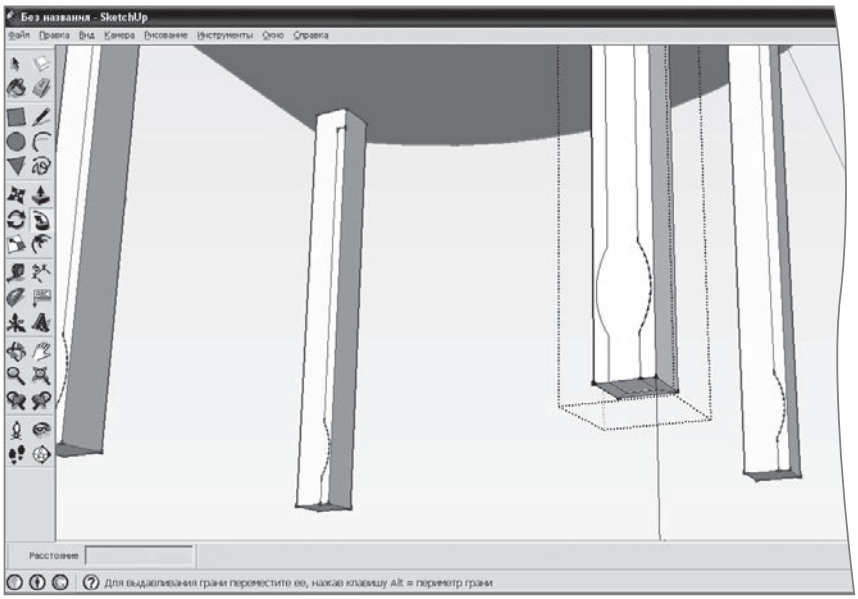

Рис. 100. Четыре ножки стула

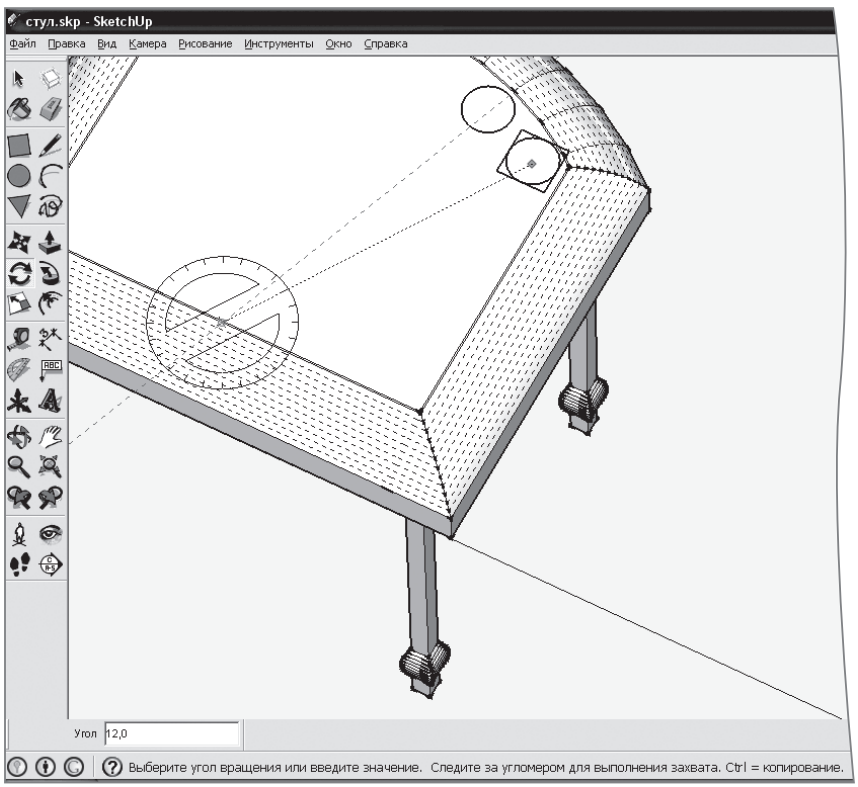

Рис. 101. Начало построения опор спинки стула

Выглядеть это будет примерно как на рис. 101:

- теперь отпустим клавишу мыши и наберем на клавиатуре «3x», после чего нажмем **Enter.** По этой команде будет выполнен тройной поворот;
- откроем компонент и вытянем опору спинки на 450 мм;
- инструментом **Переместить** при нажатой клавише **Ctrl** сдвинем дугу задней части сиденья на место перед опорами спинки;
- после этого инструментом **Переместить** при нажатой клавише **Ctrl** сдвинем получившуюся фигуру вверх на 420 мм (рис. 102);

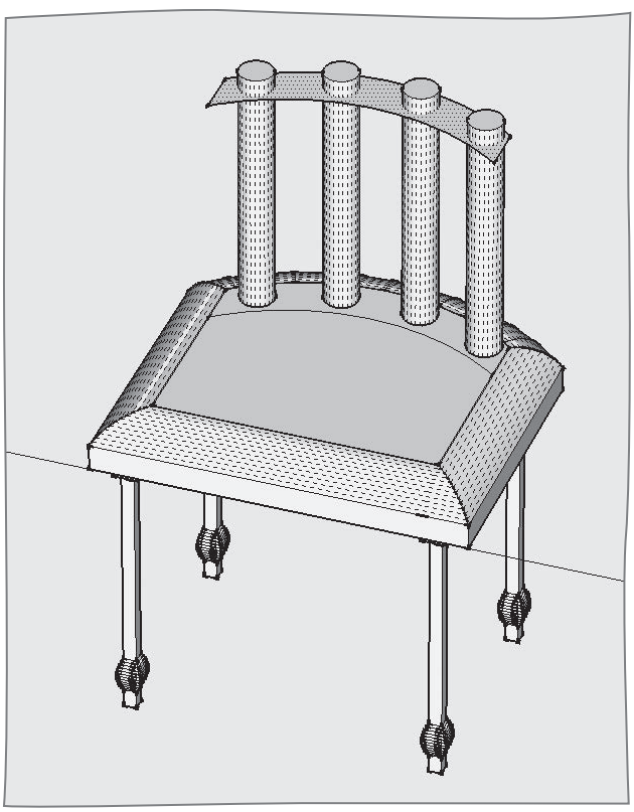

Рис. 102. Начало построения спинки стула

- инструментом **Тяни/толкай** выдвинем спинку, тем же инструментом сдвинем боковые части спинки к краям стула (рис. 103).

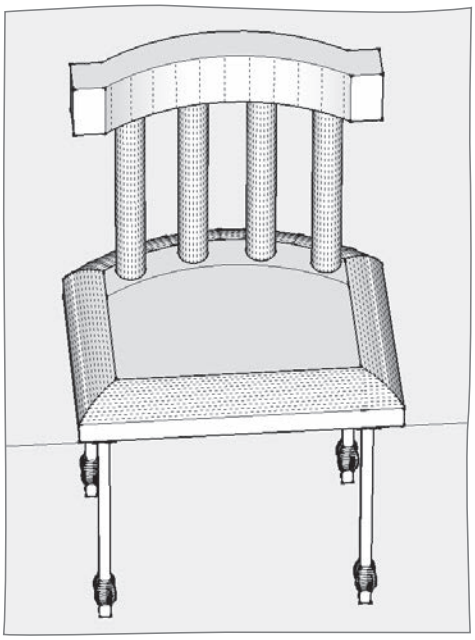

Рис. 103. Завершение построения спинки стула

После этого останется только задать текстуры. Итог будет выглядеть примерно, как на рис. 104.

Рис. 104. Задание текстуры стула

Фигуры вращения и сложные поверхности

Следующий пример-упражнение будет более сложным: мы построим модель заварочного чайника с носиком. Для этой работы нам потребуется расширить возможности рисования Sketchup. Форма носика, скорее всего, не укладывается в набор из прямых и дуг, поэтому будем рисовать и редактировать с помощью привычного инструмента — кривых Безье.

В стандартных возможностях такого нет, но Sketchup — расширяемая программа, и для нее существует масса дополнительных модулей[7]. Мы выберем модуль BezierSpline[8] (рис. 105). Это далеко не единственный вариант, и если вы найдете что-то более удобное, ничто не мешает использовать и другой вариант.

Для установки модуль нужно скачать и распаковать в каталог **Plugins** в каталоге программы. После этого среди панелей в меню **Вид → Панели инструментов** появится панель **BZ_toolbar**.

Кроме того, для рисования носика чайника нам потребуется включить компонент **Песочница** — он позволит нам строить слож-

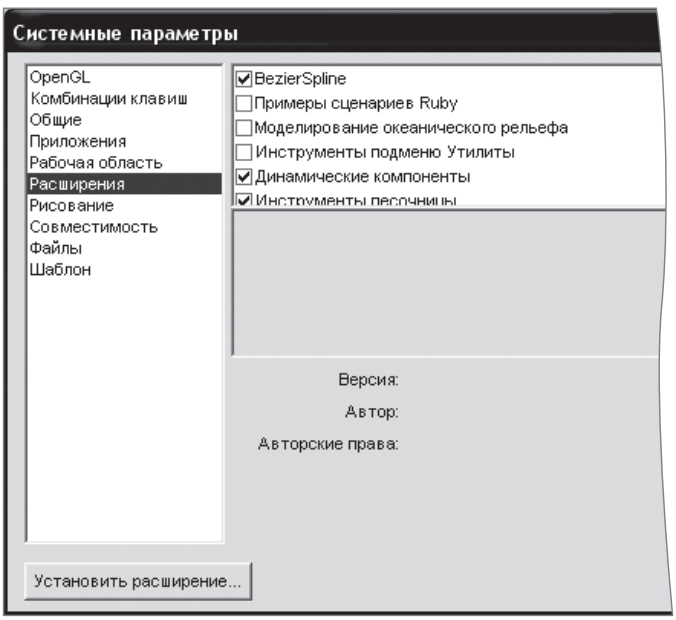

Рис. 105. Подключение дополнительных компонентов

7 Есть и модуль рисования чайника, но это «неспортивно».

8 http://forums.sketchucation.com/viewtopic.php?f=323&t=13563.

ные поверхности. В разделе **Окно → Параметры** выберите раздел **Расширения** и включите компонент **Песочница**.

Для работы используем шаблон. Начнем с подготовки «основной» части чайника. Емкость построим, как фигуру вращения, т. е. как результат проводки фигуры вдоль некоторого круга. Фигурой будет половина сечения чайника:

- сменим вид модели: в меню **Камера → Стандартные представления** выберем пункт **Спереди**;

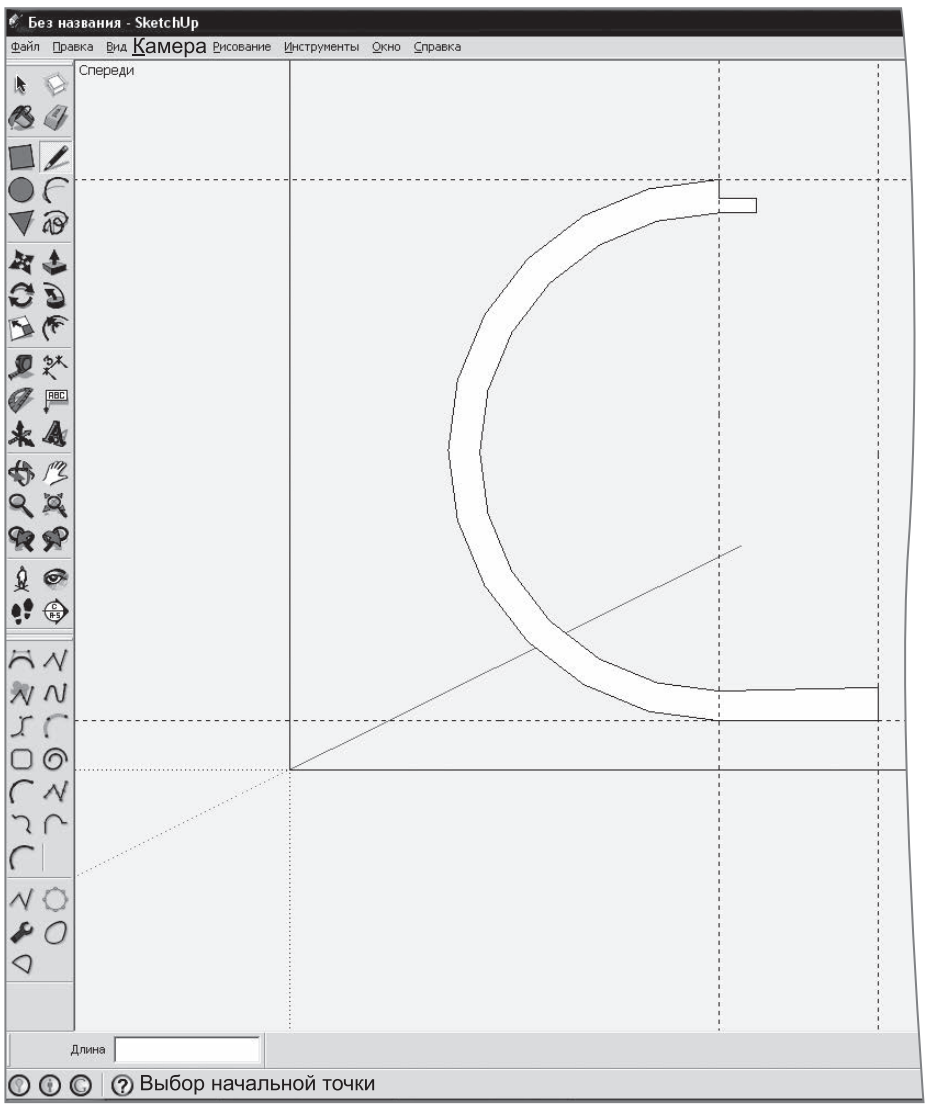

Рис. 106. Построение оси чайника

- построим вспомогательную линию — ось чайника (рис. 106);
- построим вторую вспомогательную линию — внутреннюю границу стенки чайника;
- построим дугу — внешнюю стенку чайника;
- внутри этой дуги построим вторую дугу поменьше, чтобы получилась стенка;
- обычными линиями достроим оставшиеся части стенки и дно, не забывая, что сверху чайник закрывается крышкой, — оставим для нее место.

Получится примерно, как на рис. 106.

Теперь построим саму фигуру вращения. Для этого:

- сменим вид модели: в меню **Камера** → **Стандартные представления** выберем пункт **Изометрическое**;
- инструментом  **Круг** на зелено-красной плоскости построим круг с центром на пересечении вспомогательной и координатной осей и радиусом до второй вспомогательной оси (рис. 107).

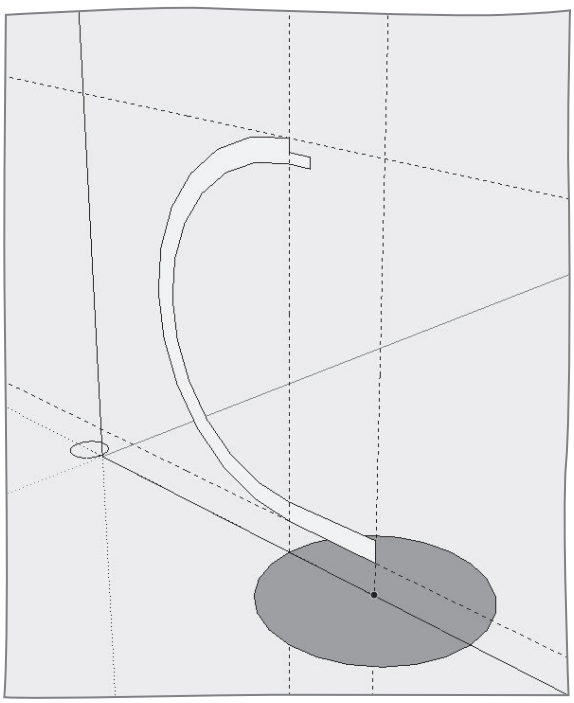

Рис. 107. Построение фигуры вращения

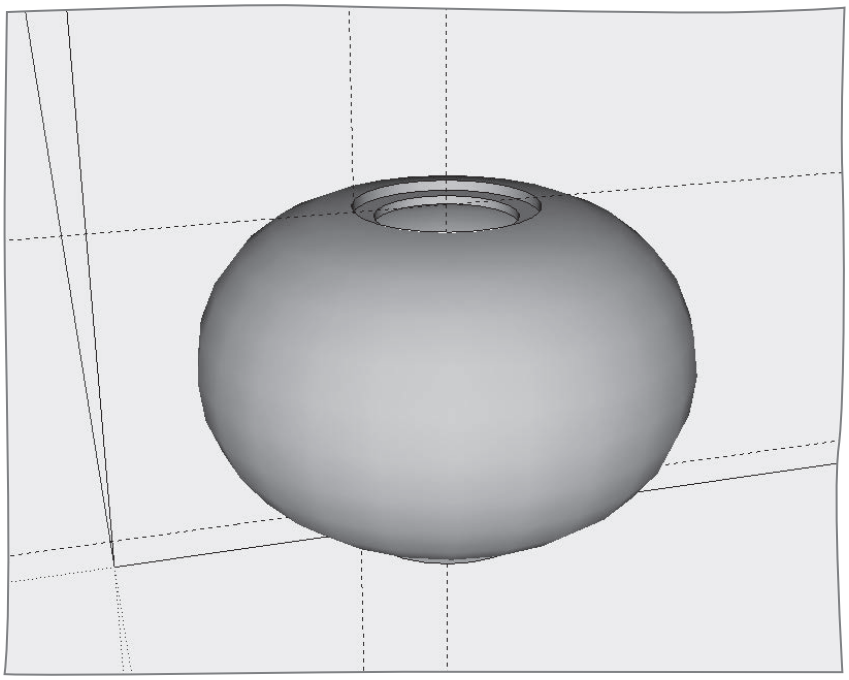

Рис. 108. Получение объемной фигуры

Инструментом  **Ведение** щелкнем сначала на площади сечения, а потом проведем по окружности, пока чайник не замкнется (рис. 108).

Чайник получился несколько «угловатый», потому что мы не повысили точность рисования дуг и окружности для ведения. В результате плоские грани заметны. При желании можно сделать место для крышки более натуральным, сгладив стыки в сечении.

Теперь нарисуем ручку. Развернем камеру к другой стороне чайника. Построим  **Рулеткой** на некотором отдалении от чайника несколько вспомогательных линий (см. рис. 107):

- вертикальную линию — для начальной и конечной точек ручки;
- горизонтальную линию через точку пересечения с красной осью;
- еще одну горизонтальную — от этой прямой вверх до опорной прямой «дна» чайника (эта линия будет над красно-зеленой плоскостью, поэтому при вытягивании стрелка должна быть синей);
- и наконец, вертикальную линию, на которой будут располагаться контрольные точки кривой.

Если этих опорных линий не будет, скорее всего, кривая окажется вне единой плоскости и нормальной удобной ручки не получится.

Теперь выберем инструмент построения кривой **Classic Bezier Curve** и построим кривую: первая ее точка располагается в нижней части крепления ручки, вторая — в верхней части, а третья и последующие снизу вверх таким образом, чтобы получилась оболочка ручки.

Контрольные точки после установки можно двигать. После того как нужная форма ручки найдена, щелкните клавишей мыши дважды. Должно получиться примерно, как на рис. 109.

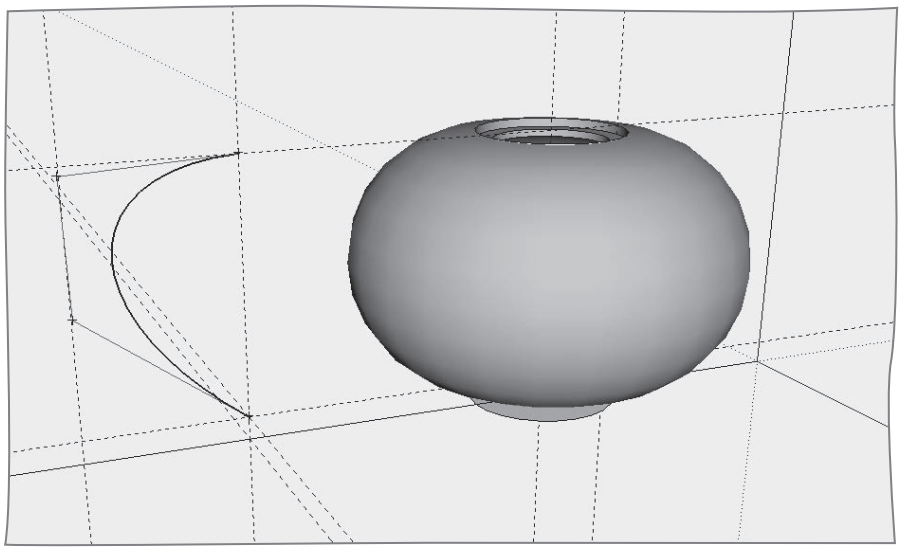

Рис. 109. Построение формы ручки чайника

Осталось немного — придать объем ручке:

- разворачиваем камеру ближе к ручке и рисуем небольшой круг в плоскости, перпендикулярной кривой, с центром в точке пересечения (в нижней точке ручки);
- выбираем инструмент **Ведение** и проводим этот круг вдоль ручки; если все сделано правильно, то даже «вести» не потребуется — ручка будет нарисована сразу;
- выберем ручку, вызовем инструмент **Масштабирование** и, двигая опорные точки, подгоним размер;

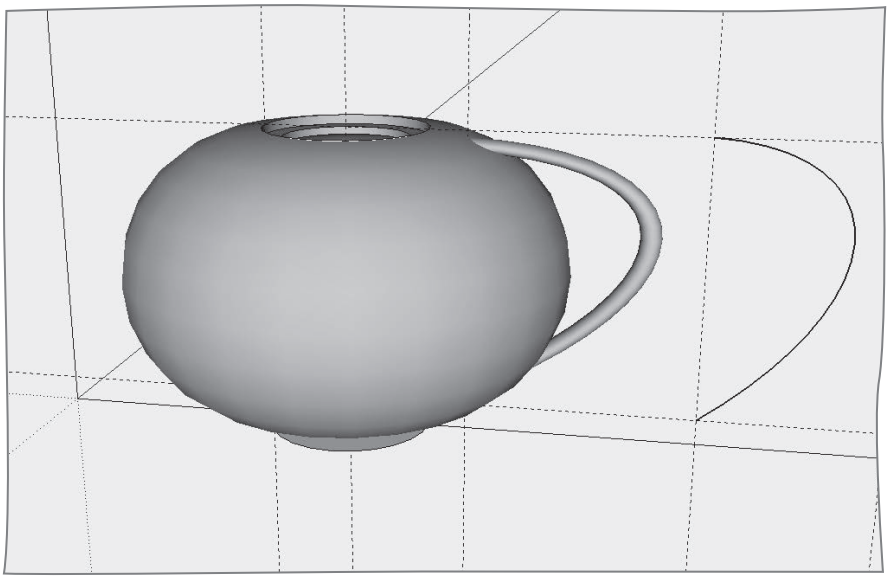

Рис. 110. Окончание построения ручки

- инструментами  **Переместить** и  **Масштабировать** подгоним положение ручки на чайнике (рис. 110).

Обратите внимание! Исходная кривая ручки и круг, по которому строился чайник, остались отдельными. Их можно удалить.

Теперь нарисуем носик чайника (рис. 111):

- построим опорные линии, как для ручки, с другой стороны чайника;
- с помощью инструмента  **Classic Bezier Curve** построим верхнюю и нижнюю линии носика;
- соединим точки отрезков, на которые разбивается кривая, новыми отрезками (рис. 112).
- повернем носик чайника так, чтобы он оказался в красно-зеленой плоскости.

Это действие связано с особенностями работы инструмента, которым мы будем строить поверхность. Для этого:

- выделим носик, т. е. все его линии; если что-то еще попадает в выделение — поверните камеру;

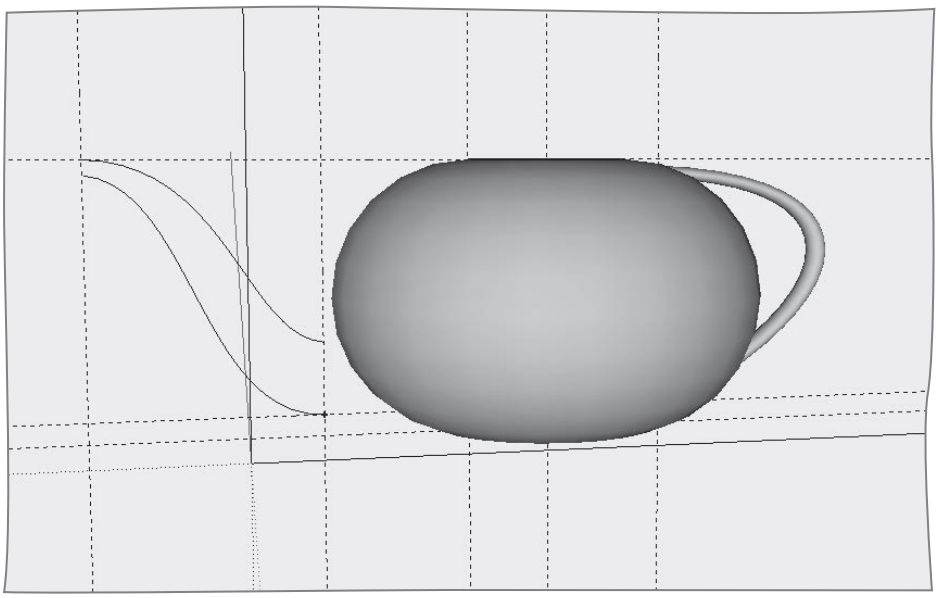

Рис. 111. Строим опорные линии для носика чайника

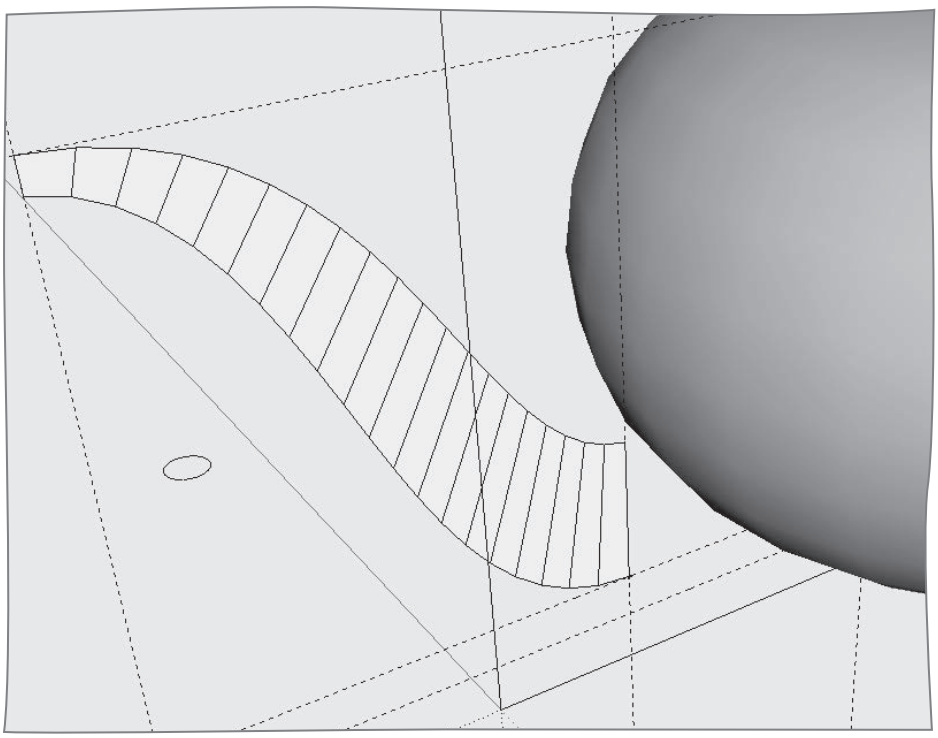

Рис. 112. Строим верхнюю и нижнюю линии носика

- разверните камеру так, чтобы видеть носик в плоскости со стороны его основания;
- выберите инструмент **Повернуть** и щелкните на центре поворота в середине первого отрезка — появится изображение транспортира;
- выберите вторую точку — в конце отрезка;
- поверните носик, поворачивая «транспортир».

Теперь приступим к построению поверхности.

На каждом диаметре носика построим полуокружность . Для этого выберем первую и вторую точки диаметра и запомним диаметр. Вытянем вверх дугу так, чтобы ее радиус был синим, т. е. строго вертикально вверх. После этого введем длину радиуса — половину запомненного диаметра. Чем больше таких дуг мы построим, тем более точным и гладким будет носик чайника.

> Обратите внимание! Очень желательно соблюдать порядок выделения точек на линиях (т. е. заранее определить, какая линия будет первой, а какая — второй). Иначе поверхность может выглядеть странно.

Выберем две соседних дуги и в меню «Рисование» выберем инструмент «Песочница», вариант «Из контуров». Можно выбрать и несколько дуг подряд, но появляется вероятность перепутать — и поверхность не получится.

Повторите эту операцию для всех пар дуг, чтобы получилась половинка носика. После построения поверхностей опорные дуги можно стереть.

> Обратите внимание! Если вы будете пропускать какие-то дуги на линиях носика, то контур будет выступать или уходить за объемную часть (рис. 113).

После того как первая половинка носика построена, построим вторую. Для этого выберите построенную половинку носика. При этом проверьте, что выбрана только она. Если выделены и вспомогательные линии или сам чайник, щелкните по ним при нажатой клавише **Shift**, чтобы исключить их из выбора.

Построим копию половинки: переключимся на инструмент **Переместить**, нажав клавишу **Ctrl**, сместим копию вниз, под уже имеющуюся;

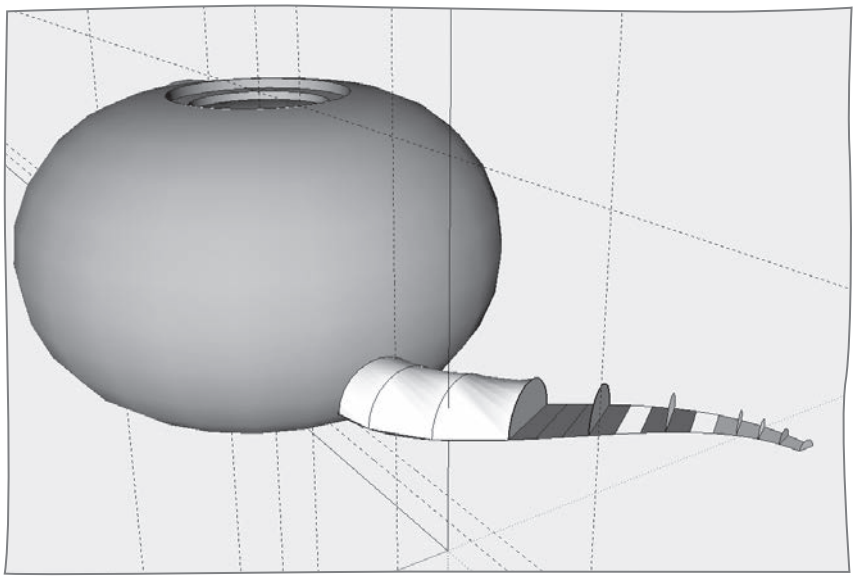

Рис. 113. Придание объема носику чайника

- переключимся на инструмент **Масштабирование** и за центральную точку верхней грани охватывающего параллелепипеда перетянем половинку вниз, чтобы она «перевернулась»;
- чтобы точно соблюсти масштаб, сразу введем число «−1»;
- полученную половинку инструментом **Переместить** поднимем вверх, к верхней половинке;
- выберите обе половинки разом, нажмите правую клавишу мыши и создайте новый компонент;
- переключитесь на инструмент **Поворот**, на задней грани носика разместите опорные точки и поверните носик так, чтобы он был правильно ориентирован, скорее всего, на 90 градусов;
- переключитесь на инструмент **Переместить** и перетащите носик на его место.

Получится фигура, сильно напоминающая чайник (рис. 114).

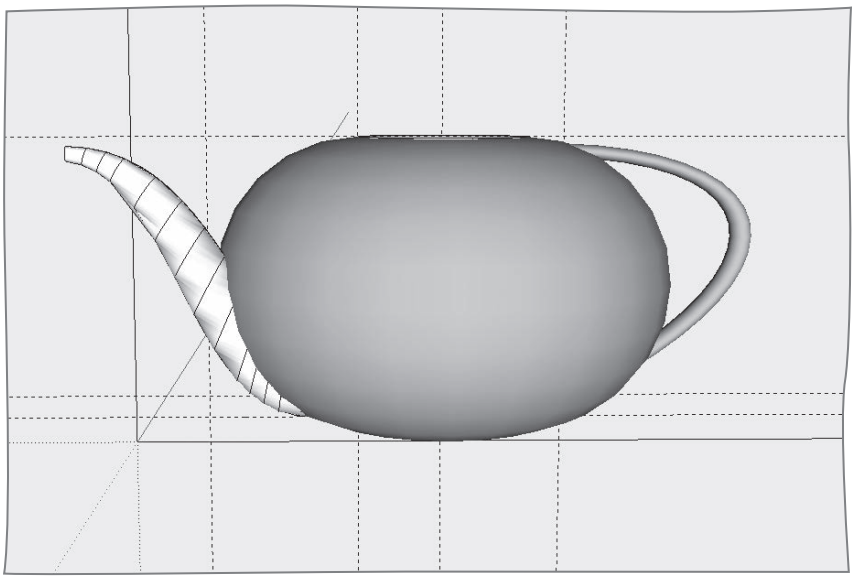

Рис. 114. Завершение рисования чайника

Проект

«Обработка звукового файла»

Программ обработки звука существует огромное количество. Некоторые из них позволяют изменить скорость воспроизведения звукового файла, другие — изменить тональность записанной музыки и многое другое. Универсальные программы, с помощью которых можно выполнить большинство операций с музыкальными файлами (фактически — последовательностями звуков), называются **музыкальными секвенсорами** и предназначаются для профессиональной звукорежиссерской работы.

К таким программам относятся Adobe Audition, Fruity Loops Studio, Cubase, Nuendo и др. На взгляд авторов, более простыми для освоения и работы являются продукты компании Steinberg — *Cubase* и *Nuendo*. Эти программы имеют схожий интерфейс и незначительные отличия. В данном практикуме мы будем осуществлять обработку звука в программе *Nuendo 3*.

Nuendo относится к условно бесплатному типу программного обеспечения: ею можно бесплатно пользоваться в течение нескольких сотен рабочих часов. Этого хватит для выполнения заданий практикума. Пробную версию можно бесплатно скачать с сайта разработчика (www.steinberg.net).

Процесс обработки записанной музыки

Процесс обработки музыки, записанной в виде отдельных файлов (дорожек или треков) записи партии музыкального инструмента, либо вокала, либо просто некоторых звуков, принято делить на три части.

1. Трекинг (Tracking) — процесс редактирования записанных звуковых файлов с помощью стандартных инструментов секвенсора (обрезка, склейка и пр.).
2. Сведéние (Mixing) — этап создания единого звукового трека. На этом этапе основная цель звукорежиссера — добиться, чтобы каждая из записанных дорожек была отчетливо слышна в общем звучании всех треков.

3. Мастеринг (Mastering) — финальный этап работы со сведенной звуковой дорожкой. На этом этапе основная задача звукорежиссера — добиться максимально качественного звучания в рамках стиля музыкального трека, а также устранить оставшиеся недостатки звука в целом.

Интерфейс программы Nuendo

Как уже говорилось, программа Nuendo является многофункциональным музыкальным секвенсором. Мы изучим лишь основные ее возможности.

1. Запустите программу.
2. При первом запуске программа предложит протестировать имеющееся звуковое оборудование — согласитесь.
3. Откроется окно программы (рис. 115).

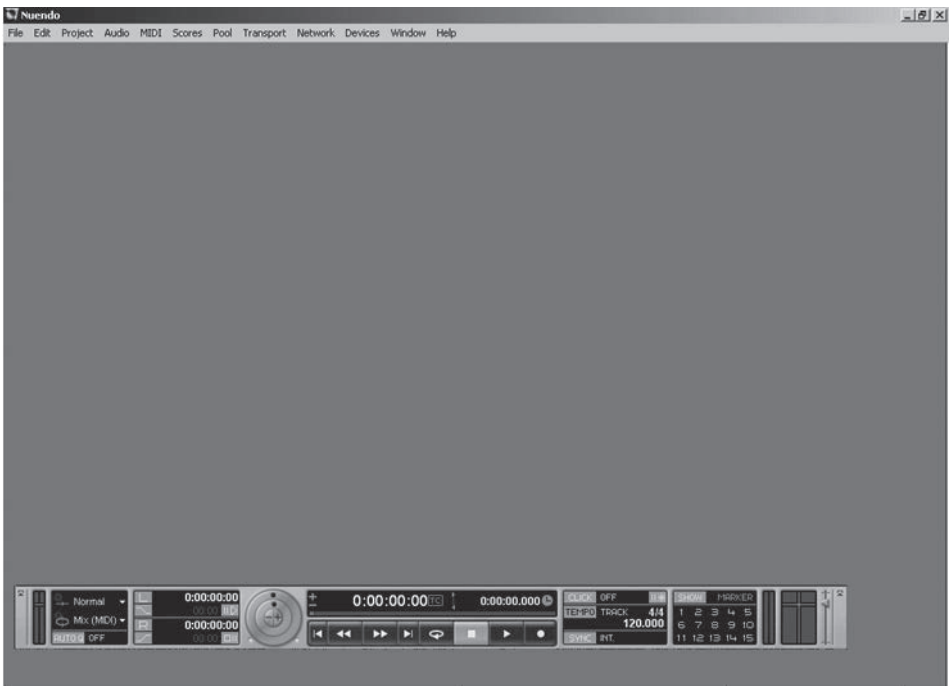

Рис. 115. Окно программы Nuendo

4. Изучите элементы меню программы, которые нам понадобятся (рис. 116).

Рис. 116. Элементы меню программы: File — основные операции: открыть, сохранить и пр.; Edit — основные операции по редактированию дорожек; Project — настройки текущего проекта; Pool — отображает папку, в которой хранятся аудиофайлы проекта; Devices — устройства (настройки звуковой карты и прочих устройств)

5. Проверьте, какой звуковой драйвер выбран в настройках программы. Если выбран один из стандартных драйверов производителя секвенсора, поменяйте его на драйвер своей звуковой карты. Если этого не сделать, программа, возможно, будет работать некорректно.

   Порядок выбора драйвера своей звуковой карты (рис. 117): **Меню → Devices → Device Setup**.

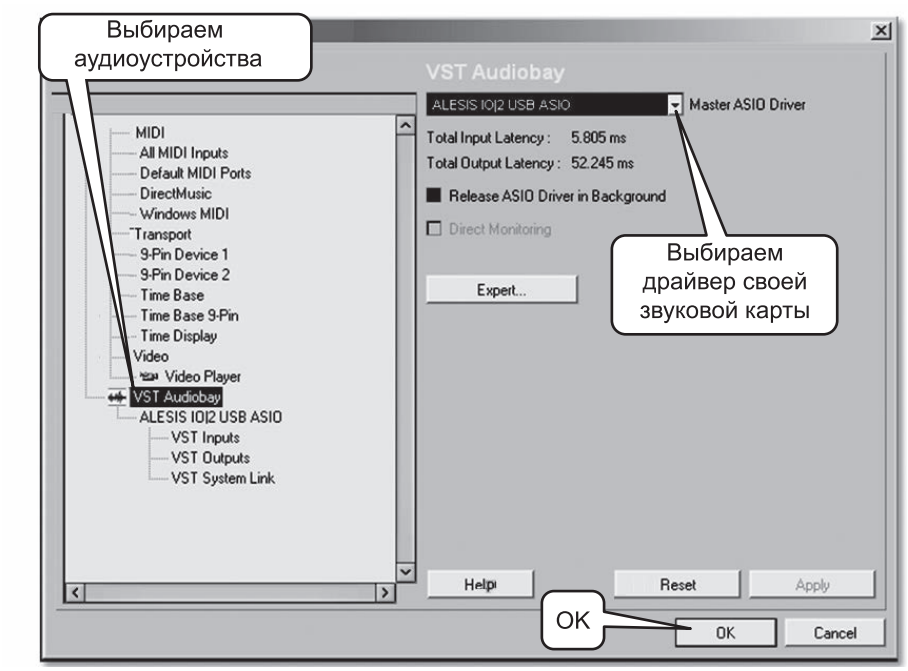

Рис. 117. Выбор драйвера звуковой карты

Если у вас возникают трудности с выбором драйвера, скачайте из Интернета и установите драйвер **ASIO4ALL**. Этот драйвер универсален и должен работать со всеми звуковыми картами.

Создание и настройка проекта

Для того чтобы начать обработку треков, требуется создать проект с определенными настройками и разместить треки в проекте.

1. Создайте новый проект (рис. 118):
   Меню → File → New Project.

Рис. 118. Создание нового проекта

2. Здесь нам предлагается выбрать ранее сохраненные настройки проекта. Поскольку мы создаем проект впервые, выбираем **Empty** (пустой).
3. Перед нами дерево каталогов нашего компьютера, где мы должны выбрать папку, в которой будет храниться наш проект. Чтобы не превращать все в свалку, лучше всего создать для проектов отдельную папку и помещать туда все папки с дальнейшими проектами. Итак, создадим такую папку и назовем ее, к примеру, **Projects**. В ней создадим папку **First Project**. Выбираем эту папку и нажимаем **OK**. В папке **First Project** автоматически будет создана папка **Audio**, в которой будут храниться все звуковые файлы проекта.

4. Перед вами появится окно проекта (рис. 119), в котором представлен снимок проекта с уже загруженными аудиодорожками. Изучите основные области окна проекта.

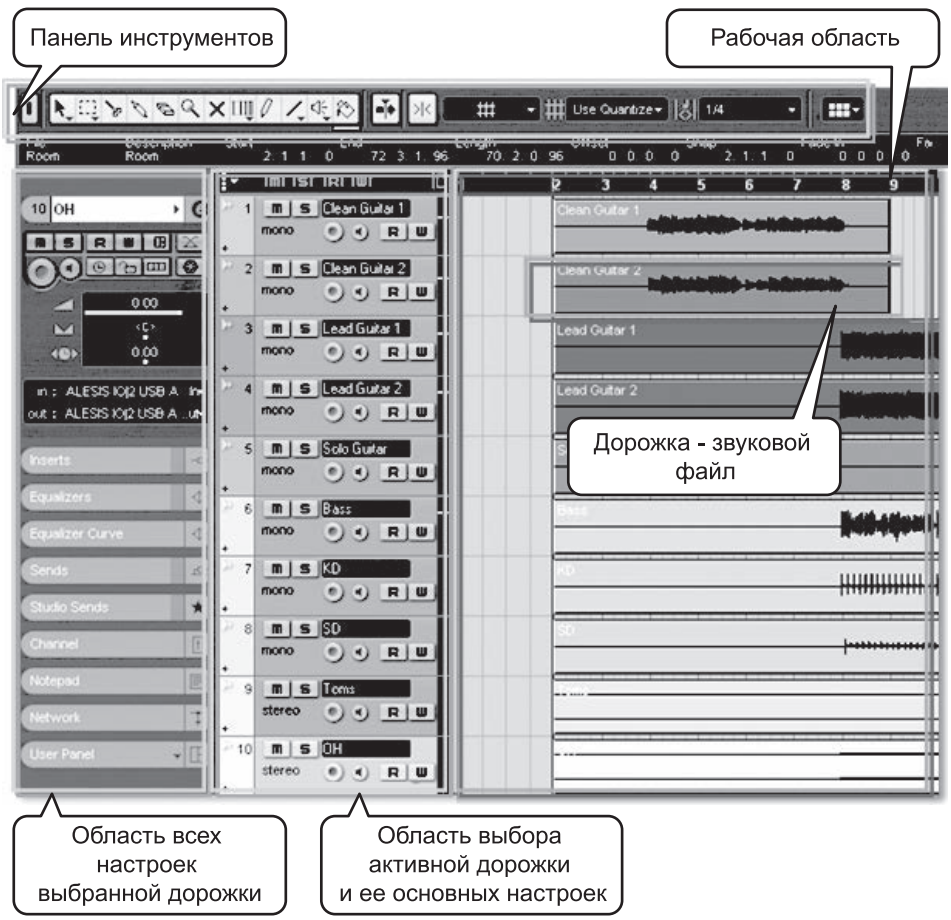

Рис. 119. Окно проекта

5. Откройте окно настроек проекта:
   Меню → Project → Project Setup.
6. Настройте параметры проекта (рис. 120):
   Sample Rate – **44.100 кГц** (частота отсчетов в секунду при кодировании звука);
   Record format – **24 бит** (глубина кодирования – количество бит для кодирования каждого отсчета);
   Record File Type – **Wave** (разрешение/тип аудиофайлов проекта).

Чем выше частота отсчетов и чем больше глубина кодирования, тем естественнее и качественнее будет компьютерный звук. Указанные настройки являются оптимальными, и в большинстве случаев используют именно их. Можно указать настройки и выше. В результате изменения в качестве звука вряд ли будут различимы, зато размер звукового файла значительно увеличится.

Рис. 120. Оцифровка звука: уровень одного отсчета может иметь одно из 2^{24} = 16 777 216 состояний; частота отсчетов составляет 44100 отсчетов в секунду или 44 100 Гц = 44,1 кГц

7. Сохраните ваш проект:
   Меню → File → Save As
   Укажите имя файла, например **Tracking and Mixing**. После сохранения в папке проекта появится файл проекта, с которым можно продолжить работу в любое время.

Добавление аудио в проект

1. Сейчас в нашем проекте нет ни одного звукового файла. Добавим их:
   Pool → Open Pool Window.

Правой кнопкой мыши нажимаем на пиктограмме **Audio → Import Medium.**

2. Через проводник найдем в папке практикума **Обработка звука** папку **Instruments** (рис. 121). В ней лежат записанные партии инструментов к песне, с которой мы будем работать.

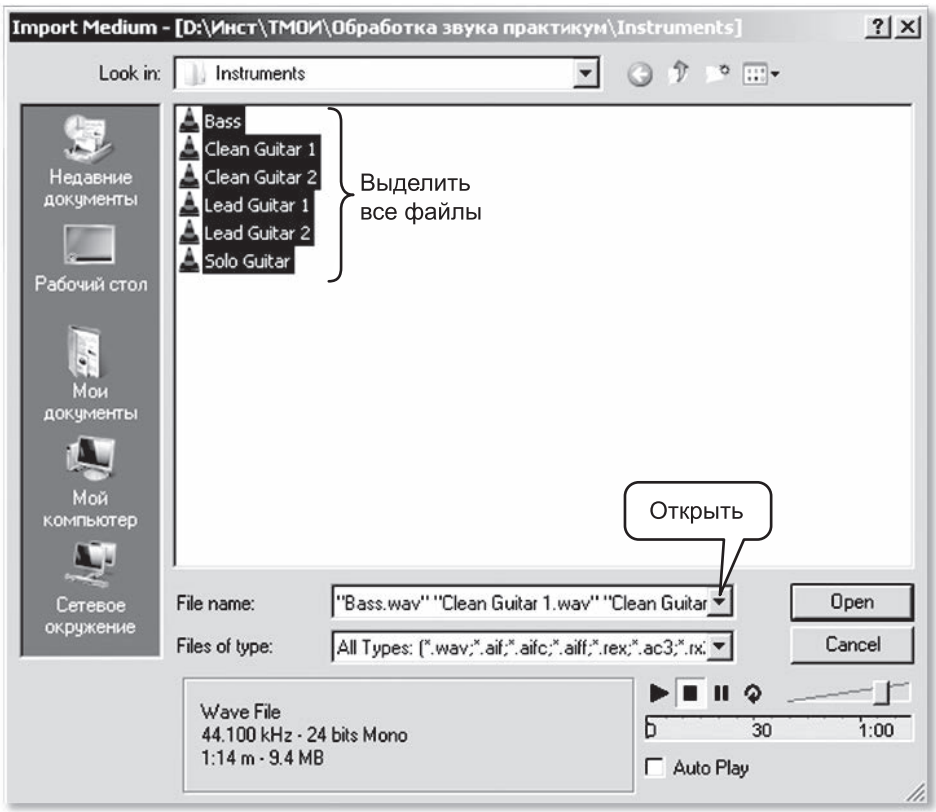

Рис. 121. Папка Instruments с записанными партиями инструментов к песне

3. В появившемся окне импорта отметьте указанные пункты (рис. 122).

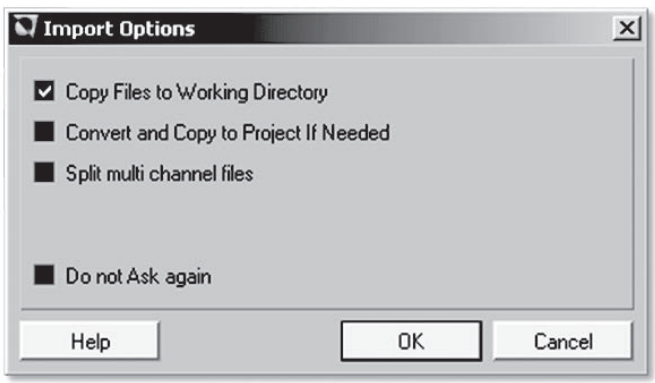

Рис. 122. Окно импорта

Нам нужно выбрать первый пункт — скопировать файлы в папку проекта. Если этого не сделать, никакие дальнейшие операции с файлами, кроме прослушивания, выполнять будет нельзя. Во втором пункте при копировании в папку проекта предлагается конвертировать копируемые файлы в файлы с расширением проекта и его параметрами, чтобы с ними можно было работать. Этот пункт можно не выбирать поскольку эти файлы уже в формате **Wave** и имеют те же параметры, что и наш проект.

Третий пункт — разделение многоканальных файлов (стерео и пр.) нам тоже не нужен — мы будем работать пока только с монодорожками (рис. 123).

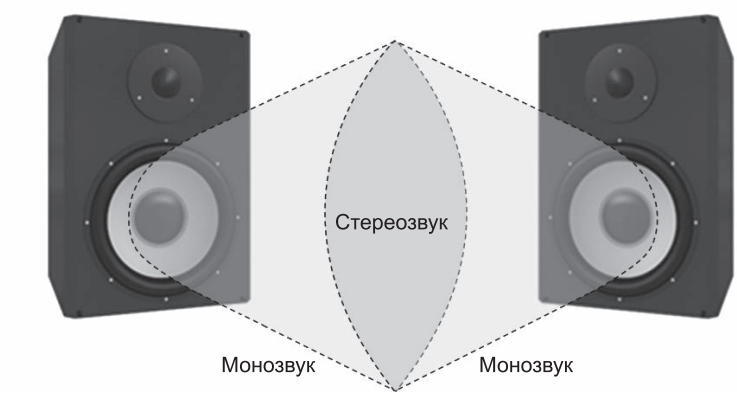

Рис. 123. Образование стереозвука

Нажимаем **ОК**.

В результате мы загрузили в проект все необходимые партии, т. е. дорожки, из которых он будет состоять.

4. Разместим загруженные дорожки в проекте.

Закройте **Pool Window** и нажмите правой кнопкой на пока еще пустую область настройки всех дорожек (см. рис. 119). Появится список всех типов треков, которые мы можем создать. Нам нужно добавить аудиотрек — нажимаем правой кнопкой мыши по пустой области выбора активной дорожки (рис. 124) → **Add Audio Track**.

Рис. 124. Добавление аудиотрека

В появившемся окне выбора конфигурации трека (сколько каналов будет в треке) выбираем **Mono** (см. рис. 124).

5. Вместо названия **Audio 01** напишите более понятное название появившегося трека — **Clean Guitar 1** (рис. 125).

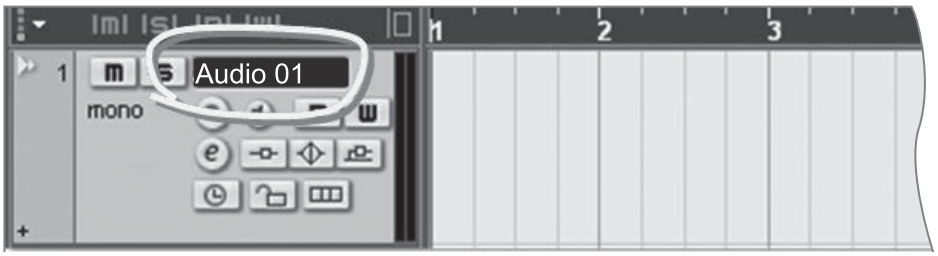

Рис. 125. Переименование аудиотрека

6. Переместим соответствующий звуковой файл из **Pool Window** в рабочую область на эту дорожку следующим образом:

открываем **Pool Window** и перетаскиваем мышкой файл **Clean Guitar 1** на рабочую область соответствующей дорожки. Должно получиться, как на рис. 126.

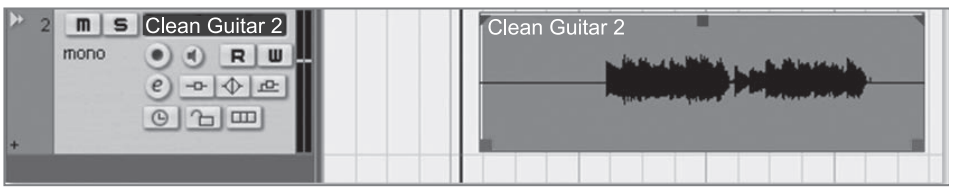

Рис. 126. Перемещение звукового файла из **Pool Window** в рабочую область дорожки

Сверху над изображением звукового файла у нас идут цифры 1, 2, ... — это такты сетки метронома проекта. При работе с музыкой удобнее всего работать именно с ней. Если ее нет, ее можно настроить в **Project Setup** (настройках проекта).

7. Поставьте курсор проигрывания перед дорожкой (рис. 127). Для этого кликните на поле сетки метронома в нижней его части (рис. 128).

Рис. 127. Установка курсора проигрывания перед дорожкой

Рис. 128. Позиционирование начала проигрывания

Нажмите клавишу **Пробел** для воспроизведения дорожки с места, где установлен курсор.

8. Теперь добавим еще один файл в поле проекта, чтобы можно было одновременно слушать две гитары. Для этого создайте еще одну звуковую дорожку, как мы это только что сделали, и добавьте в нее файл **Clean Guitar 2** (рис. 129).

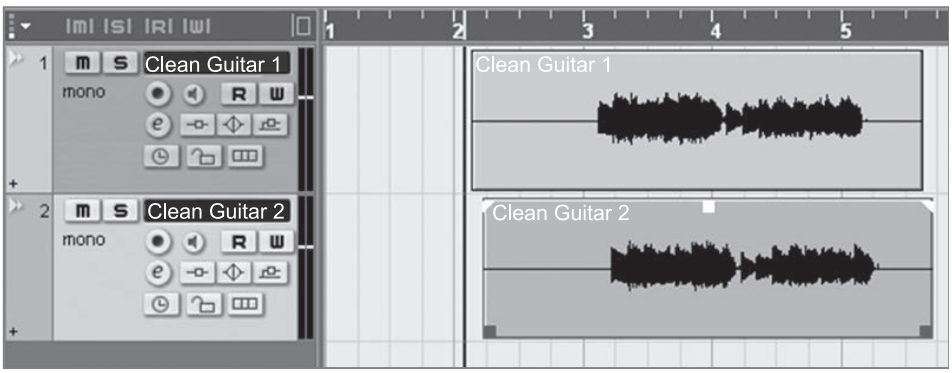

Рис. 129. Добавление файла **Clean Guitar 2**

Включите воспроизведение — две гитары звучат одновременно. Правда, мы их еще не выровняли, поэтому они могут немного «не попадать» друг в друга.

9. Чтобы не выравнивать дорожки вручную, в Nuendo есть возможность притягивать их к общей сетке метронома. Именно так на записи добиваются идеального попадания музыкантов в ритм. Чтобы это сделать, установите на панели инструментов настройки, указанные на рис. 130.

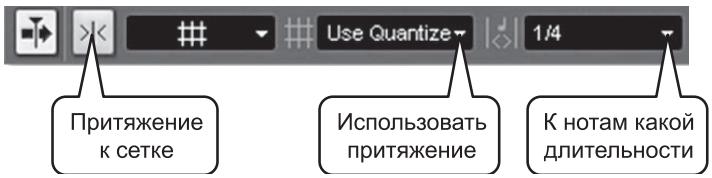

Рис. 130. Притяжение дорожек к общей сетке метронома

Если какого-то элемента нет на панели инструментов, то нажмите на ней правую кнопку мыши и отметьте галочкой необходимый инструмент (рис. 131).

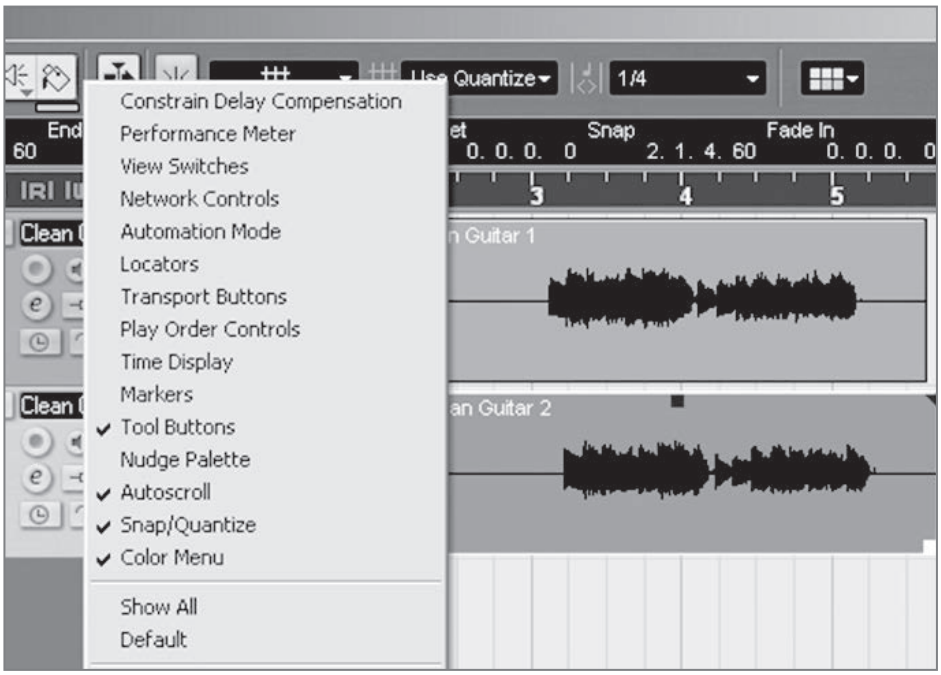

Рис. 131. Добавление необходимого инструмента

10. После включения притяжения к сетке выберите на панели инструмент **Перемещение**, чтобы двигать дорожки четко по четвертым (1/4) нотам сетки (рис. 132). Сделайте так, чтобы файлы начинали воспроизводиться точно в одном месте — с начала второго такта (рис. 133). Включите воспроизведение — теперь гитары должны «попадать» друг в друга.

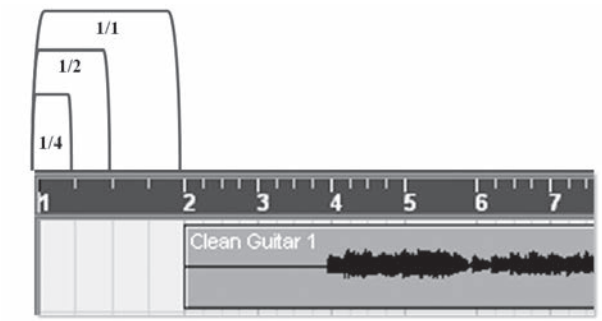

Рис. 132. Длительности нот

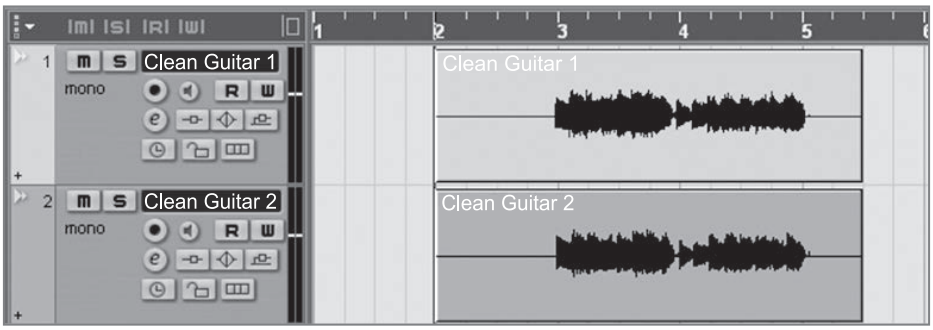

Рис. 133. Установка воспроизведения обеих дорожек с начала второго такта

11. Аналогичным образом добавьте в проект остальные файлы из **Pool Window**. Сделайте так, чтобы все дорожки начинали воспроизводиться с начала второго такта (рис. 134).

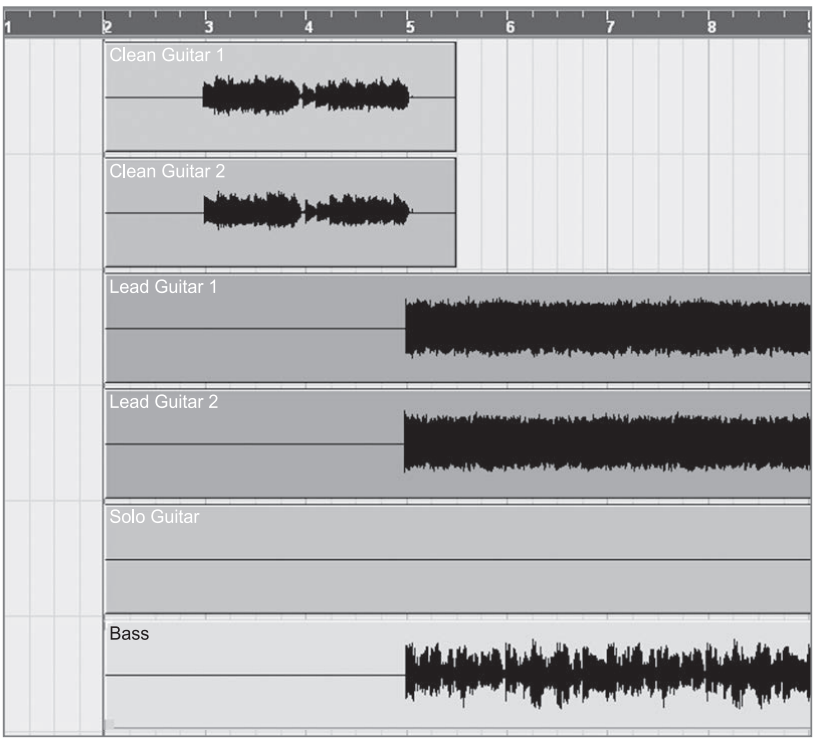

Рис. 134. Установка воспроизведения всех дорожек с начала второго такта

12. Воспроизведите и послушайте результат.

Трекинг

Трекинг — первый этап редактирования. Здесь задача звукорежиссера — сделать так, чтобы каждый трек звучал идеально для дальнейшего сведения.

1. Послушайте отдельно дорожку с бас-гитарой. Нажмите кнопку **S** **(Solo)** на дорожке **Bass**.
2. Воспроизведите, послушайте. Слышите щелчки при склейке кусков? Это абсолютно нормальное явление при склейке на записи, которое очень просто убрать с помощью Nuendo.
3. Щелчки находятся на сетке метронома. Так как в середине песни темп (скорость игры музыкантов) меняется, нужно загрузить темпотрек (темп, отображающийся в виде общей сетки метронома), данный для этой песни:
   Меню → File → Import → Tempo Track.

Выбираем файл **tempo_track** в папке практикума. Теперь все места склейки находятся на линиях сетки метронома (рис. 135).

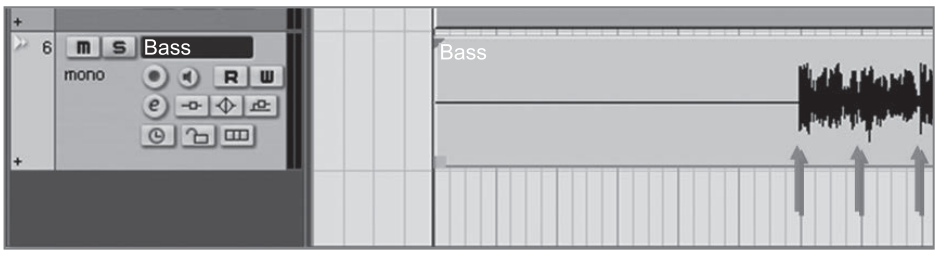

Рис. 135. Установка всех мест склейки на линиях сетки метронома

4. Теперь слушаем, где щелчки у баса. Между 11-м и 12-м тактами есть щелчок. Выберите инструмент **Ножницы** на панели инструментов и разрежьте файл между 11 и 12 тактами (притяжение к сетке должно быть включено, иначе разрезать дорожку точно по сетке сложно). Выделите две полученные дорожки инструментом  с нажатой клавишей **Shift** (рис. 136).

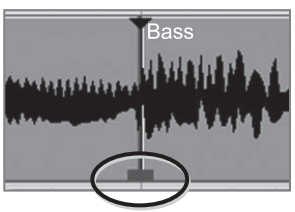

Рис. 136. Подготовка к удалению щелчков у баса

5. Сделайте склейку выделенных треков, нажав клавишу *X* на клавиатуре. Полученный результат изображен на рис. 137.

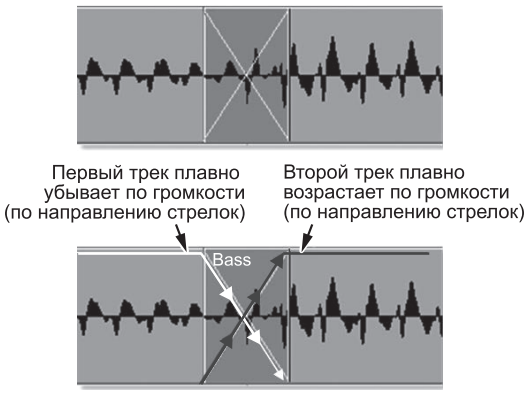

Рис. 137. Результат удаления щелчков у баса

Воспроизведите — щелчок должен убраться. Сделайте так с остальной частью басовой дорожки, где услышите щелчки (ищите на началах такта).

Выполненная операция называется **Crossfade** — плавный переход уровня сигнала.

6. Добавим в проект ударные инструменты. В **Pool Window** добавим для удобства отдельную папку барабанов Drums в папке **Audio**. Правой кнопкой нажмите на папке **Audio** → **Create Folder** — появится новая папка. Переименуйте ее в **Drums** (рис. 138). Добавьте туда все файлы из папки **Drums** в папке практикума (правой кнопкой мышки в **Pool Window** на папке **Drums**, дальше аналогично). Должно получиться, как показано на рис. 138. Справа всегда указывается информация о файлах. Заметьте, три из новых файлов имеют конфигурацию **Stereo**. Такую конфигурацию нужно выбрать для создаваемого трека для этого файла.

По аналогии с инструментами, поместите в проекте партии ударных инструментов. Приравняйте все партии к началу второго такта (рис. 139).

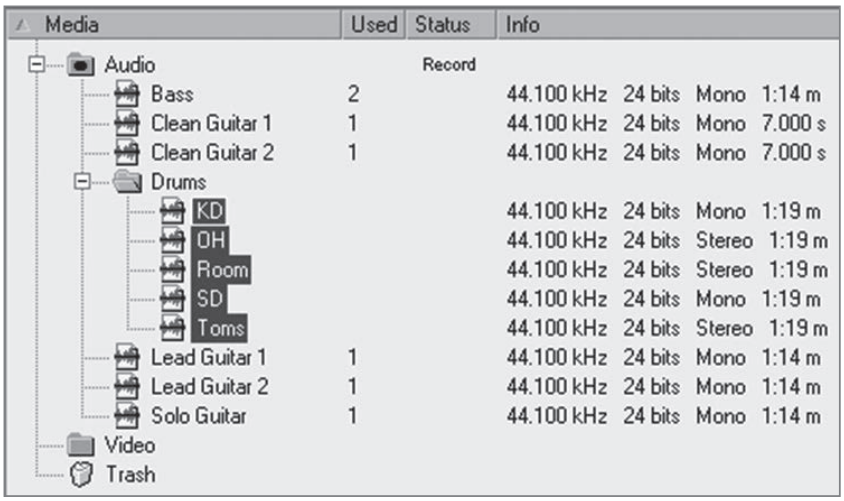

Рис. 138. Добавление в проект ударных инструментов

Рис. 139. Добавление в проект партий ударных инструментов

Сведение

Итак, у нас теперь каждая партия находится в проекте, все дорожки выровнены и склеены, можно приступать к сведению. Для начала прослушаем еще раз то, что у нас уже есть, без какой-либо обработки. Звучит не очень, правда? Громкости не настроены, что-то теряется, что-то слишком выбивается, все инструменты перекрывают друг друга. Ликвидировать все эти недоразумения и сделать общее звучание более выразительным можно на этапе сведения. Вот основные его шаги:

1) панорамирование;
2) эквализация;
3) компрессия;
4) дополнительные эффекты.

Разберемся подробнее с каждым из них.

Панорамирование

Панорамирование — это первый этап сведения, на котором каждой дорожке выделяется отдельная область в пространстве. У нас две колонки и из них звучит музыка, о каком пространстве идет речь?

Все не так просто. Стереозвучание, как вы, должно быть, знаете, существует не для того, чтобы звук из двух колонок был просто громче. Если вы, например, включите свою любимую группу на плеере и послушаете музыку через один наушник — вы все поймете.

Дело в том, что при сведении музыки, например, одной гитаре выделяется полностью правый канал (колонка, наушник), а другой — полностью левый. Таким образом, создается имитация объема, с разных сторон которого звучат разные музыкальные инструменты (создается панорама). У слушателя возникает ощущение, что справа от него играет один инструмент, а слева — другой. При этом инструменты, «разведенные» по панораме, уже не будут так перекрывать друг друга (рис. 140).

При прослушивании музыки часто можно услышать, как какой-то звук как бы перетекает из одного наушника в другой. Это тоже достигается за счет панорамы.

1. Итак, «разведем» гитары по панораме. Для этого откройте микшерный пульт (микшер) Nuendo:
   Меню → Devices → Mixer (или клавиша **F3**).

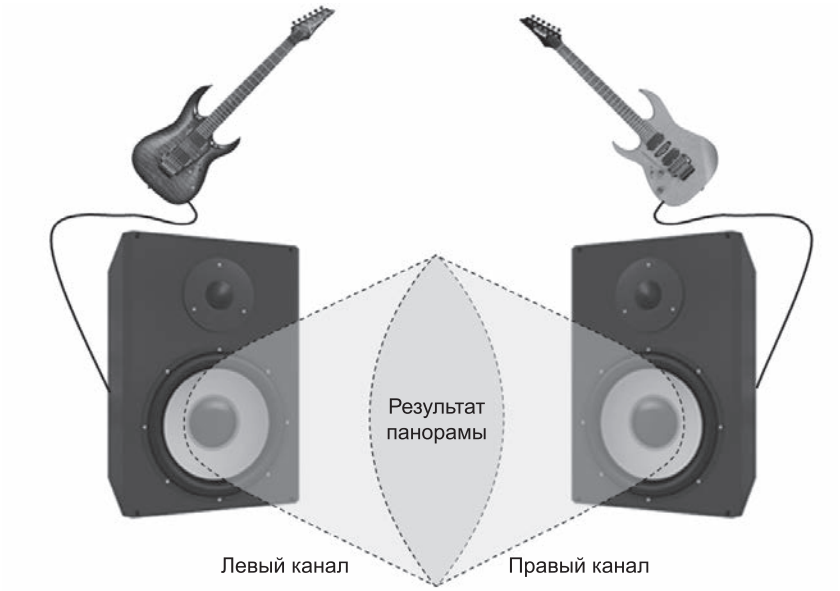

Рис. 140. «Разведение» инструментов по панораме

Нажмите на микшере кнопку **S** **(Solo)** на дорожках **Lead Guitar 1** и **Lead Guitar 2**, чтобы слышать только их (рис. 141).

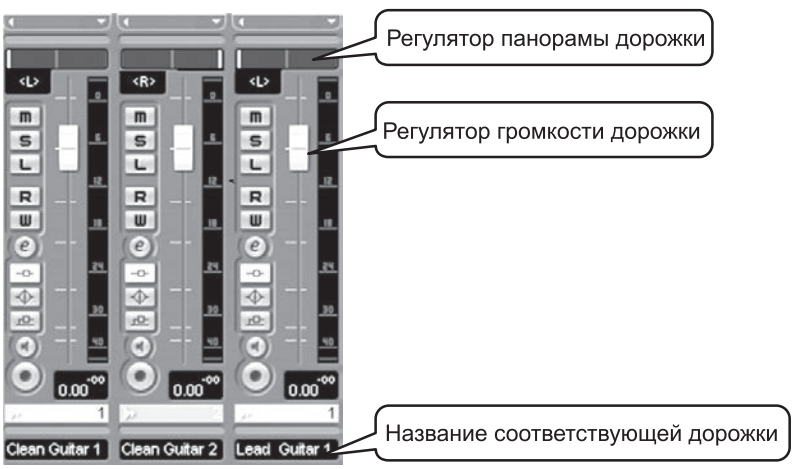

Рис. 141. «Разведение» гитар по панораме

С помощью регуляторов панорамы поместите дорожку **Lead Guitar 1** полностью в левый канал, а **Lead Guitar 2** — полностью в правый канал. Прослушайте результат. Улучшилось ли звучание?

Заметьте, что дорожка может находиться не полностью в каком-то одном канале, а, например, только на 80% в левом. Это значит, что 20% останутся в правом канале, и мы их будем немного слышать правым ухом. В результате у нас возникнет ощущение, что звук играет не слева, а слева спереди. Это особенно заметно, если слушать музыку в наушниках.

2. Аналогично «разведите» по панораме дорожки **Clean Guitar 1** и **Clean Guitar 2**. Прослушайте результат. Общее звучание должно стать лучше, гитары не будут наслаиваться друг на друга.

У остальных дорожек не меняйте панораму. Все монодорожки будут по умолчанию в центре панорамы, а стереодорожки уже разведены по панораме (послушайте отдельно стереодорожку **OH**).

3. Регуляторами громкости убавьте звук у дорожек гитар и баса так, чтобы они не заглушали барабаны и мы могли по возможности слышать каждую дорожку.

Эквализация

Эквализация — это второй и к тому же один из самых важных этапов сведения. Что же такое эквализация? Слово кажется сложным, но на самом деле все элементарно.

Звуки можно разделить по частоте на низкие, средние и высокие. Например, бас-гитара имеет низкое звучание — по большей части ее звук находится в низких частотах и занимает их почти целиком. Обычная гитара занимает большую часть средних частот и часть высоких. «Железо» барабанов (тарелки) — большую часть высоких.

Однако бас-гитара все равно немного звучит в области средних и в области высоких частот, перекрывая звучание обычных гитар и тарелок. Так и любой другой записанный инструмент все равно немного будет звучать за пределами своего основного частотного диапазона.

Эквализация помогает убавить громкость ненужных частот у звука и поднять громкость тех частот, на которых инструмент будет хорошо звучать вместе с остальными. Давайте попробуем.

Обработка эквализацией ведется с помощью расширений программы — так называемых *звуковых плагинов*, которые реализуют большинство эффектов при обработке. Наложим нужные эффекты на отдельные треки.

1. Откройте микшер, добавьте расширенный микшер и в нем выберите **Разрывы** (Inserts — поля для вставки звуковых плагинов) по всем дорожкам (рис. 142).

Рис. 142. Добавление расширенного микшера

2. Вставьте эквалайзер в первый **Разрыв** трека **Clean Guitar 1** (рис. 143).

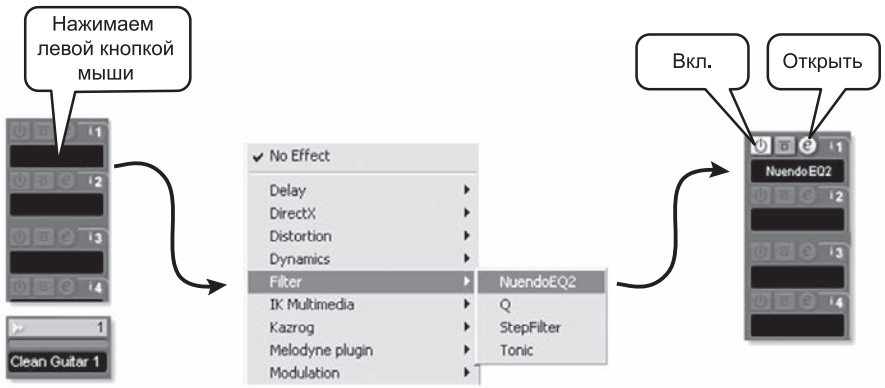

Рис. 143. Вставка эквалайзера в первый **Разрыв** трека **Clean Guitar 1**

3. Установите режим **Solo** только для дорожки **Clean Guitar 1**, чтобы услышать, как эквалайзер меняет звук (рис. 144).
4. Откройте и изучите окно эквалайзера (рис. 145).

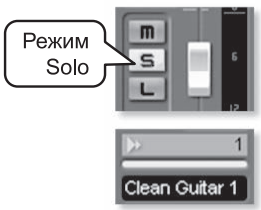

Рис. 144. Установка режима **Solo** только для дорожки **Clean Guitar 1**

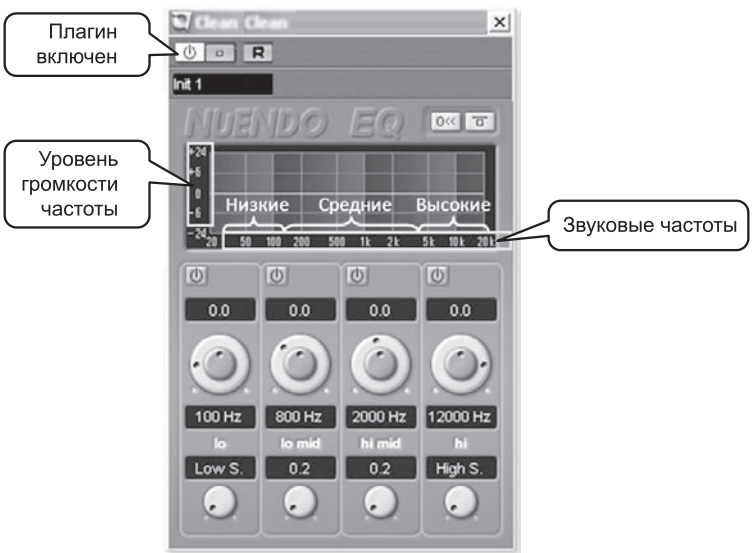

Рис. 145. Настройка эквалайзера

5. Изучите, как настроить эквалайзер (см. рис. 145).

Поэкспериментируйте с ручками, чтобы лучше разобраться, как изменение их положения влияет на кривую эквалайзера и на звук дорожки.

6. Частотный диапазон гитары обычно выше 150 Гц. Убавьте у дорожек **Clean Guitar 1**, **Clean Guitar 2, Lead Guitar 1**, **Lead Guitar 2** низкие частоты (рис. 146).
7. Сделайте эквализацию у других дорожек:
   7.1. **Solo Guitar** — убавьте низкие частоты до 200 Гц и «узко» поднимите частоту 5000 Гц (5 кГц) на 10 децибел (дБ);
   7.2. **Bass Guitar** — убавьте низкие частоты до 50 Гц и «узко» поднимите частоты 150 Гц и 5 кГц на 10 дБ, «узко» понизьте 300 Гц на 10 дБ;

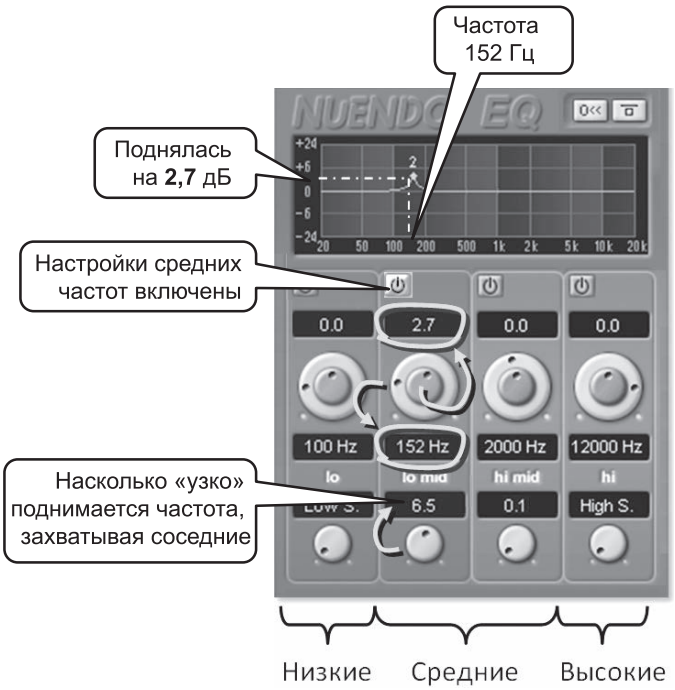

Рис. 146. Убавление низких частот у дорожек **Clean Guitar 1, Clean Guitar 2**, **Lead Guitar 1**, **Lead Guitar 2**

7.3. **KD** — убавьте низкие частоты до 60 Гц, «узко» поднимите частоты 80 Гц и 5 кГц на 10 дБ;
7.4. **SD** — убавьте низкие частоты до 100 Гц, «узко» поднимите частоты 200 Гц и 5 кГц на 10 дБ;
7.5. **Toms** — убавьте низкие частоты до 100 Гц, «узко» поднимите частоты 200 Гц и 5 кГц на 10 дБ;
7.6. **OH** — убавьте низкие частоты до 550 Гц;
7.7. **Room** — ничего не меняйте.

8. Настройки эквалайзера могут варьироваться по вкусу звукорежиссера, так как каждый проект уникален, а у разных инструментов немного различаются частотные диапазоны.

Определить точно, в каком частотном диапазоне звучит конкретный инструмент, могут помочь более многофункциональные эквалайзеры, которые при воспроизведении трека отображают уровень звучания инструмента в каждой частотной полосе.

Воспроизведите то, что у нас получилось. Для сравнения можно на время отключить все эквалайзеры, которые мы загрузили, и послушать, какой звук был раньше. При обработке звука очень полезно делать такие сравнения. Включите все эквалайзеры снова. Эквализация завершена.

Компрессия

Компрессия — это третий шаг в сведении композиции. Что такое компрессия? Сравните, например, дорожки **Lead Guitar 1** и **Bass** (рис. 147).

Рис. 147. Рисунок уровня громкости у дорожек **Lead Guitar** и **Bass**

Дорожка **Lead Guitar** по рисунку уровня громкости ровная на протяжении всего трека, а дорожка **Bass**, наоборот, динамична на протяжении всего трека.

Динамика в игре на музыкальном инструменте — естественное явление, так как музыкант не робот и не может играть все ноты с одинаковой громкостью. Однако в общем звучании такая динамика иногда очень сильно «режет слух». Поэтому мы ее уберем компрессором — плагином, который в заданной степени уменьшает громкость звуков, уровень громкости которых превышает заданный порог. На рисунке 148 представлен отрывок дорожки баса не компрессированный (*а*), компрессированный (*б*) и обработанный лимитером (*в*). Лимитер — это плагин, выполняющий полное «выравнивание» всех нот по громкости.

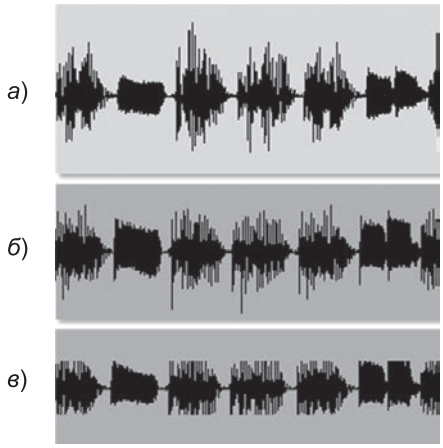

Рис. 148. Звук без обработки — явные пики громкости (*а*); компрессированный звук — менее явные пики громкости (*б*); звук, обработанный лимитером, — отсутствие пиков громкости (*в*)

1. Добавьте компрессор во второй разрыв дорожки **Bass** и изучите его окно (рис. 149).

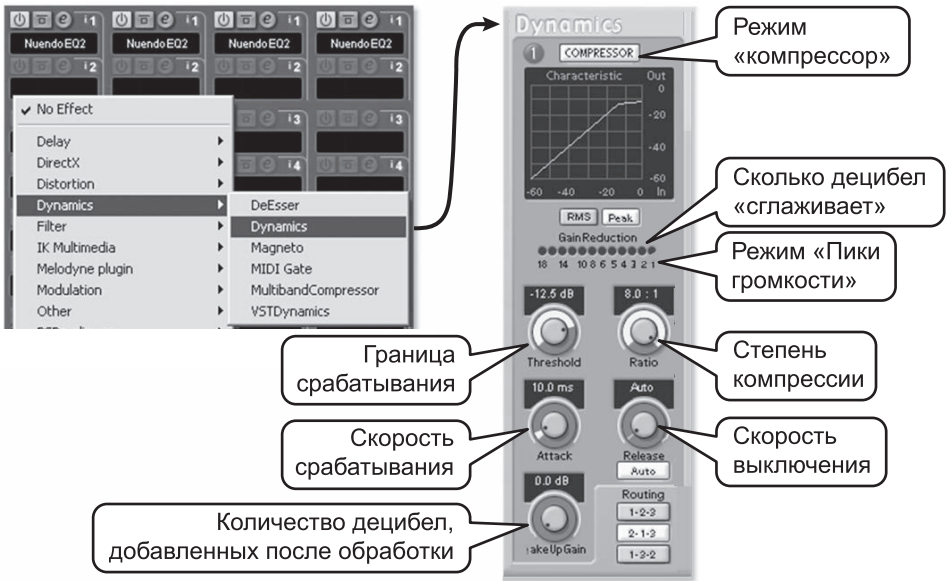

Рис. 149. Окно дорожки **Bass**

Настроим границу срабатывания *Threshold*, чтобы «сгладить» пики, т. е. уменьшить их громкость на 10 дБ (рис. 150).

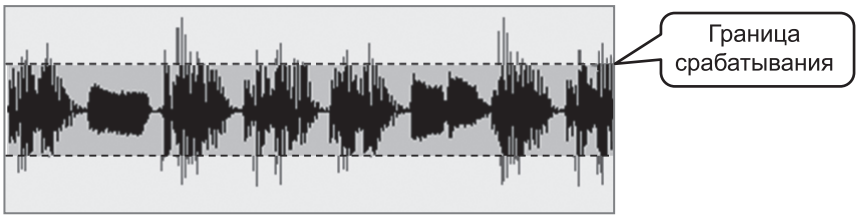

Рис. 150. Настройка границы срабатывания

Включите проигрывание дорожки баса. Меняйте границу срабатывания и следите на компрессоре за шкалой *Gain Reduction* (сколько децибел «сглаживается»). Выберите такую границу, чтобы показатель «сглаживания» *Gain Reduction* колебался в области 10 дБ.

2. Установите максимальную степень компрессии и минимальное значение скорости срабатывания *(Attack)*. Скорость выключения *(Release)* оставьте по умолчанию.

3. Поднимите выровненный звук на 10 «сглаженных» дБ *(Make Up Gain)*. Прослушайте результат. Отключите компрессор и сравните с предыдущим звучанием.
4. Включите режим **Лимитер** в окне плагина (рис. 151).

Сделаем наш бас идеально ровным по громкости: выставьте границу срабатывания –12 дБ. Таким образом, лимитер будет «срезать» все звуки, уровень громкости которых выше –12 дБ. Послушайте результат.

Рис. 151. Включение режима **Лимитер** в окне плагина

5. Сделайте компрессию дорожек, задав параметры:
   1. ***KD***: алгоритм *Peak; Trashold* (–32) дБ, *Ratio* — 8:1; *Attack* — 0,1; *Release* — Auto; *Make Up Gain* — 15 дБ.
   2. ***SD***: алгоритм *Peak; Trashold* (–40) дБ; *Ratio:* 8:1; *Attack* — *10; Release* — Auto; *Make Up Gain* — 10 дБ.
   3. ***Toms*** — так же, как для дорожки *SD*.

Компрессия имеет много других применений, например можно сделать несколько компрессий, и звук станет довольно специфичным. У каждого звукорежиссера есть свои секреты по использованию компрессоров.

Дополнительные эффекты

Обычно для того чтобы свести трек на хорошем уровне, профессиональному звукорежиссеру будет достаточно трех перечисленных выше этапов. Но, как правило, появляется желание добавить еще что-нибудь. Здесь уже все зависит от фантазии и знаний звукорежиссера. Расскажем о двух часто применяемых эффектах.

Мультиполосная компрессия. Вкратце, это компрессия определенных частотных диапазонов. Этот эффект может придать инструменту более яркое звучание. Сделаем это для гитар в нашем проекте.

1. Откройте микшер и установите дорожкам **Lead Guitar 1** и **Lead Guitar 2** режим **Solo**.
2. Во вторые разрывы треков **Lead Guitar 1** и **Lead Guitar 2** установите плагин **Multiband** (рис. 152).
3. Послушайте результат. Выключите мультиполосную компрес-

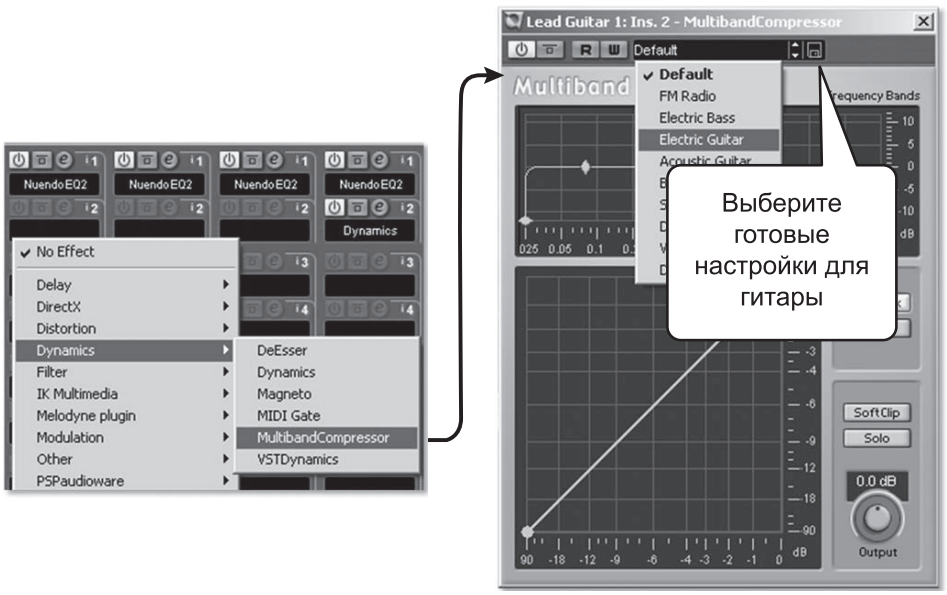

Рис. 152. Включение мультикомпрессии

сию и сравните с тем, что было ранее.

4. Примените мультиполосную компрессию к дорожкам: **Solo Guitar**, **Bass**, **KD**, **SD** и **Toms** с соответствующими готовыми настройками: *Electric Guitar*, *Electric Bass*, *Bass Drum* и *Snare Drum*.
5. Послушайте результат. Выключите мультиполосную компрессию и сравните с тем, что было ранее.

Реверберация. Реверберацией называют эффект, при котором звук прекращает свое звучание не сразу, а продолжает некоторое время отдаленно звучать в пространстве. Представьте, что вы находитесь в длинном коридоре с гладкими твердыми стенами. Издавая какие-либо звуки, вы можете слышать, как они распространяются вдоль коридора, отражаясь от стен. Это и есть эффект реверберации. Применение его к некоторым трекам улучшает их звучание. Обычно этот эффект применятся к вокалу, соло гитаре, к акустическим гитарам, но, как правило, не применяется к басу и другим дорожкам, которые «поддерживают» ритм композиции.

В нашем проекте вокала нет, поэтому сделаем реверберацию на дорожках **Clean Guitar 1** и **Clean Guitar 2**.

6. Во второй разрыв треков **Clean Guitar 1** и **Clean Guitar 2** поместите **Реверб** (рис. 153).

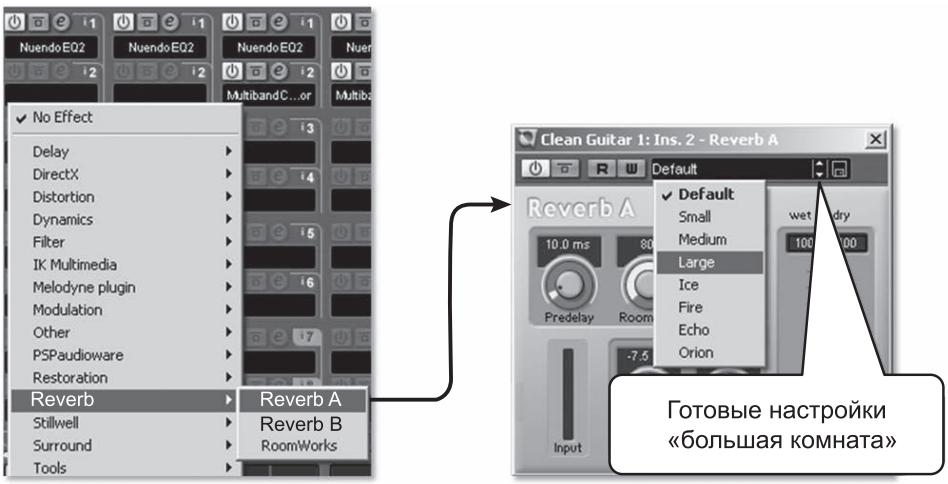

Рис. 153. Введение реверберации на дорожках **Clean Guitar 1** и **Clean Guitar 2**

7. Сделайте также реверберацию для дорожек:
   1) **SD** с готовыми настройками *Medium* (средняя комната);
   2) **Solo Guitar** с готовыми настройками *Small* (маленькая комната).

Мастеринг

Итак, осталась заключительная часть, — придать нашей композиции звучание, соответствующее выбранному стилю, и устранить оставшиеся общие дефекты. Сделать это можно в этом же проекте, а можно создать отдельный проект. Мы сделаем это в отдельном проекте.

Первое, что нужно сделать, — проверить, чтобы наш сведенный проект не зашкаливал по общей громкости и, как следствие этого, не было искажений в звуке. Откройте микшер и посмотрите дорожку общего звучания (мастер-шину).

Если горит красная кнопка **Clip**, это означает, что наш проект зашкаливает по громкости и ее нужно уменьшить (рис. 154). Делается это на этой же дорожке регулятором громкости. Сделайте так, чтобы общая громкость была около −6 дБ.

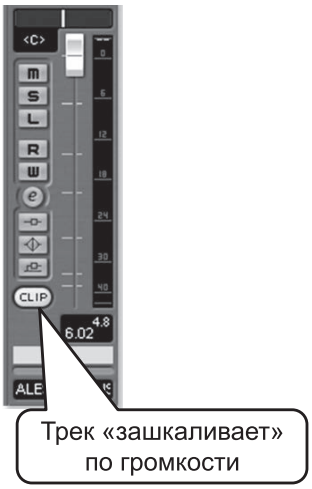

Рис. 154. Индикация чрезмерной громкости дорожки

Сохраните файл, полученный после сведения.

1. Для этого выделите область проекта, которую нужно сохранить. Подведите курсор к верхней части поля сетки метронома, пока не появится значок в виде карандаша. Поставьте курсор для выделения (рис. 155).

Рис. 155. Установка курсора в начало сохраняемой области проекта

Рис. 156. Растягивание области выделения

Растяните слева направо область выделения на столько тактов, сколько нам нужно — до конца песни (рис. 156).

Выделенная область должна быть окрашена в синий цвет. Область, окрашенная в красный цвет, означает неправильное выделение (справа налево). Чтобы было удобнее растягивать, можно уменьшить масштаб ползунком внизу справа (рис. 157).

Рис. 157. Уменьшение масштаба ползунком

2. После того как мы выделили сохраняемую область, можно сохранить сведенный файл:

   Меню → File → Export → Audio Mixdown.

   Выбираем папку нашего проекта и сохраняем сведенный файл (рис. 158):

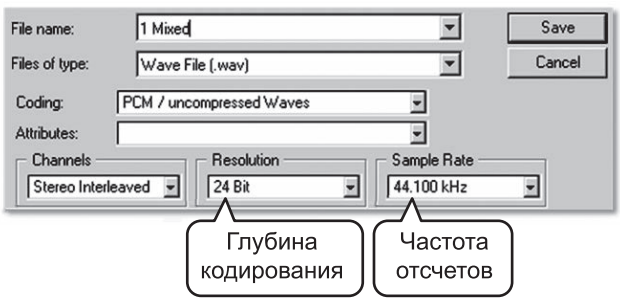

Рис. 158. Сохранение сведенного файла: Files of type — тип/разрешение файла; Coding — тип кодирования; Stereo Interleaved — стерео

3. Теперь приступаем к мастерингу. Создайте новый проект и назовите его, например, **Mastering**. В нем создайте одну аудиодорожку с конфигурацией **Стерео** и через **Pool Window** импортируйте сведенный нами трек на эту дорожку.
4. Откройте микшер и поместите на сведенную дорожку эквалайзер. Увеличьте звук в районе **5 кГц**. Поэкспериментируйте с другими частотами.
5. Посмотрите на рисунок дорожки в проекте. Снова видим динамику уровня громкости. С помощью компрессии выровните звук по громкости. Поместите компрессор после эквалайзера (порядок, кстати, имеет значение, именно в таком порядке будет происходить обработка звука).
6. Настройте компрессор: алгоритм — *Peak; Trashold* — (–22) дБ; *Ratio* — 4:1; *Attack* — 0,1; *Release* — Auto; *Make Up Gain* — 5 дБ.
7. Добавьте мультиполосную компрессию. Сделайте настройки по вкусу.
8. Отрегулируйте общую громкость проекта так, чтобы она не зашкаливала.
9. Сохраните полученный звуковой файл с расширениями **wave** и **mp3**.

Теперь можно закрыть Nuendo и насладиться своей работой, послушав файл на плеере. Сравните результат с исходной несведенной версией.

Проект

«Защита данных в сетях»

В нашей практической работе мы создадим несколько «модельных» ситуаций, чтобы показать, какие средства и как будут использоваться. Будем считать, что практически все угрозы безопасности — внешние, и рассмотрим построение системы защиты именно с точки зрения сети.

Подготовка рабочей среды

Из всего написанного ранее в учебнике видно, что средства защиты информации либо уже встроены в операционную систему (далее ОС), либо встраиваются во время установки. В противном случае обеспечить контроль за всеми выполняемыми операциями просто невозможно.

Выполняя практические работы, вам придется серьезно вмешиваться в работу ОС, что, в случае правильного планирования деятельности в компьютерном классе, возможно только для преподавателя. На домашнем компьютере это просто нежелательно хотя бы потому, что эксперименты могут закончиться не очень удачно.

Для решения таких задач удобно использовать так называемые виртуальные машины. Использование виртуальной машины — это способ выделить часть ресурсов своего компьютера и представить его как отдельный, независимый компьютер.

Способов такого выделения ресурсов (виртуализации) довольно много. Мы выберем тот, который позволяет создать максимально независимую виртуальную машину: со своей операционной системой, своими устройствами и т. д. Разумеется, без машины-хозяина гостевая машина работать не будет, но свобода выбора в этом случае будет максимальной.

На машине-хозяине нужно установить среду виртуализации, т. е. программы, которые позволяют предоставить ресурсы как отдельный компьютер.

Разумеется, виртуальная машина — не «настоящая». В любом случае она работает медленнее, не все реальные устройства смогут с ней взаимодействовать, ее работоспособность зависит от среды виртуализации. Однако и плюсов у такого решения много.

1. Виртуальную машину легко упаковать и сделать резервную копию. Если в процессе работы что-то будет испорчено, можно стереть файлы и восстановить готовую к работе систему. Это позволяет много экспериментировать или «размножить» готовую машину.
2. Виртуальную машину легко перенести на другой компьютер: не требуется ничего стирать, выполнять процедуру установки, устанавливать драйверы. Требуется только установить среду виртуализации.
3. Можно, если позволяют ресурсы, запустить и использовать несколько машин одновременно, например моделируя работу в сети.

Используя эти возможности, подготовим себе стенд для выполнения практических работ.

1. Скачайте с сайта компании Microsoft бесплатную среду виртуализации для экспериментов Microsoft Virtual PC. В зависимости от версий вашей операционной системы можно скачать либо Microsoft Virtual PC 2007, либо Microsoft Virtual PC for Windows 7.
2. Установите среду работы с виртуальными машинами, как любую другую программу.
3. Запустите среду Virtual PC (рис. 159). Сейчас там ничего нет, поскольку ни одной виртуальной машины мы пока не создали.

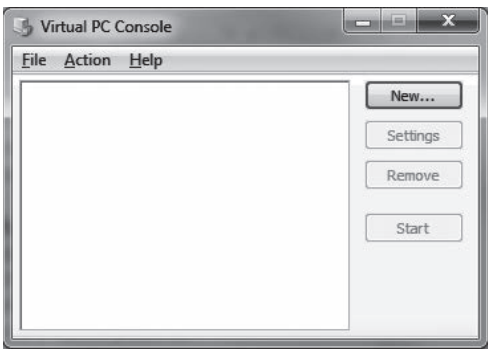

Рис. 159. Окно среды VirtualPC

4. Создаем виртуальную машину: нажмите кнопку **New...**. При создании требуется пройти несколько шагов (рис. 160).

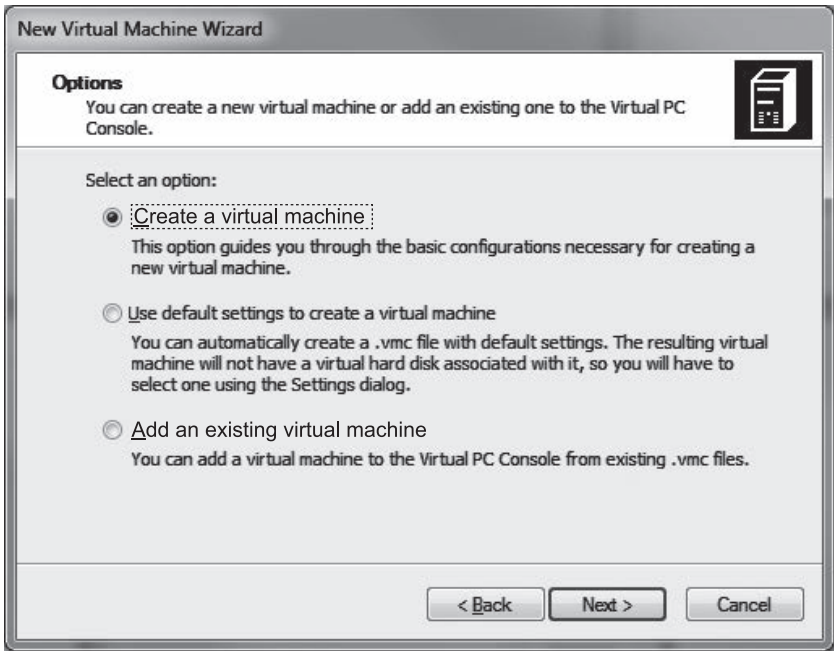

Рис. 160. Выбор варианта «Создание новой машины»

Если на компьютере-хозяине уже есть готовая виртуальная машина, выберите вариант **Add an existing virtual machine** и укажите место ее хранения. В этом случае можно сразу перейти к шагу 10 — проверке и настройке сетевых соединений.

Если нет, создаем новую машину, указывая все ее настройки. Выбираем первый пункт.

Задайте имя (рис. 161). На следующем экране нам предлагаются стандартные настройки (рис. 162). Операционная система в этом окне — просто способ выбрать рекомендуемый комплект настроек.

На следующем шаге согласитесь с размером выделяемой памяти — 128 Мб, а потом укажите новый виртуальный диск, фактически просто файл на диске (рис. 163). Согласитесь с его параметрами (рис. 164).

Если у вас недостаточно ресурсов, можно изменить эти величины в сторону уменьшения, но это сильно замедлит работу машины.

Подтвердите создание виртуальной машины (рис. 165) кнопкой **Finish**. Машина готова.

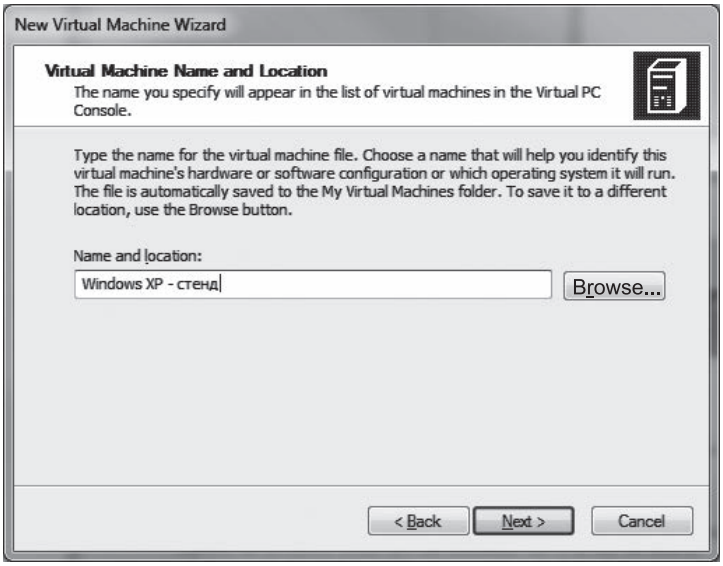

Рис. 161. Присвоение имени новой машине

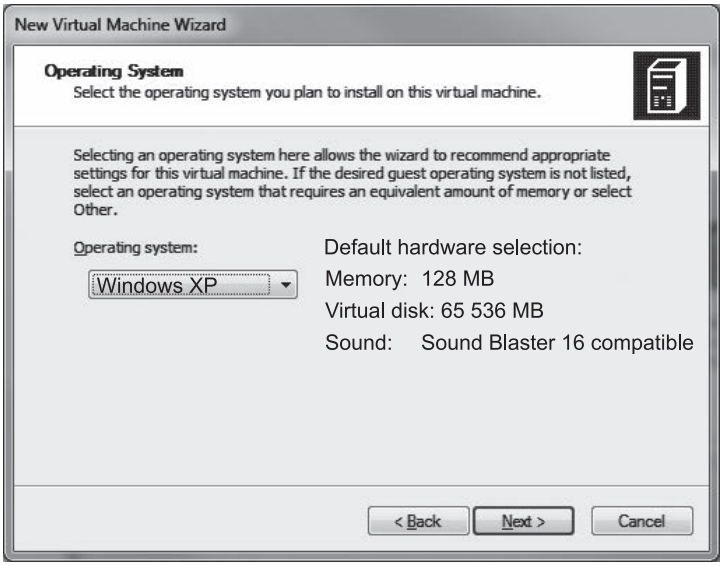

Рис. 162. Стандартные настройки

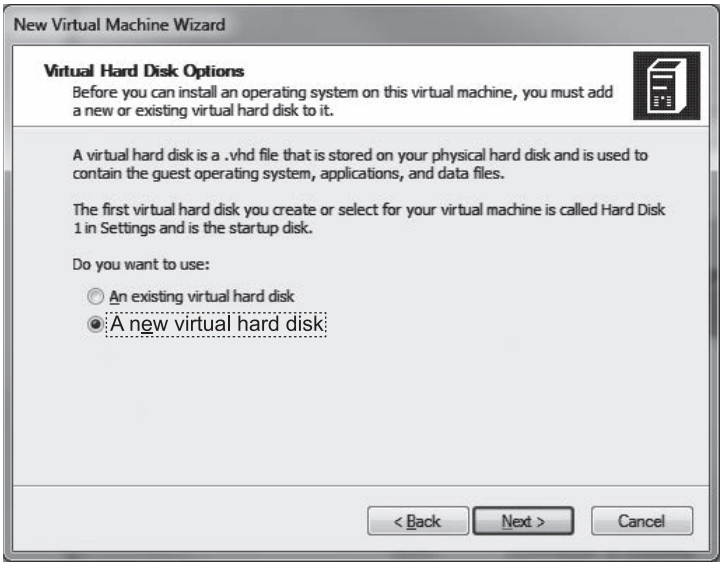

Рис. 163. Указание нового виртуального диска

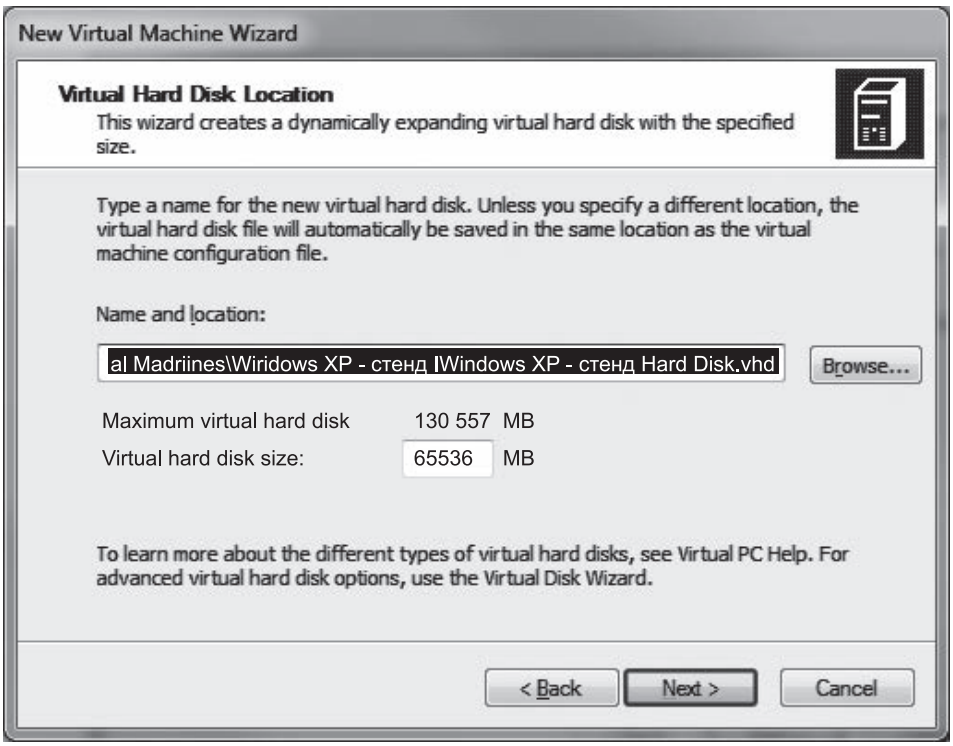

Рис. 164. Соглашаемся с параметрами нового виртуального диска

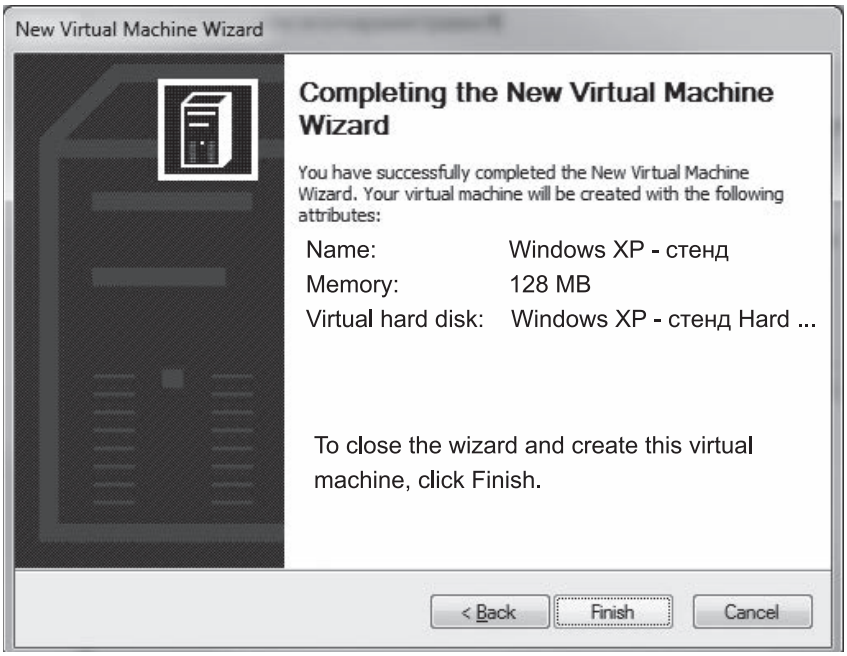

Рис. 165. Машина готова

5. В общем списке машин у вас появилась первая машина (рис. 166).

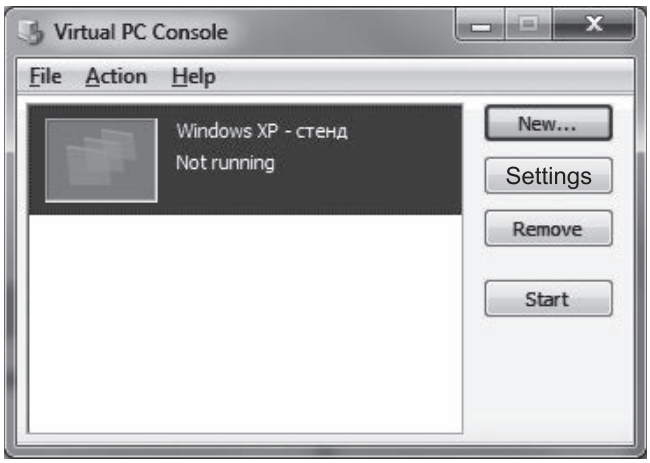

Рис. 166. В общем списке машин появилась первая машина

Как следует из текста на экране, она не запущена. Прежде чем ее запускать, проверьте некоторые настройки, нажав кнопку **Settings** (рис. 167).

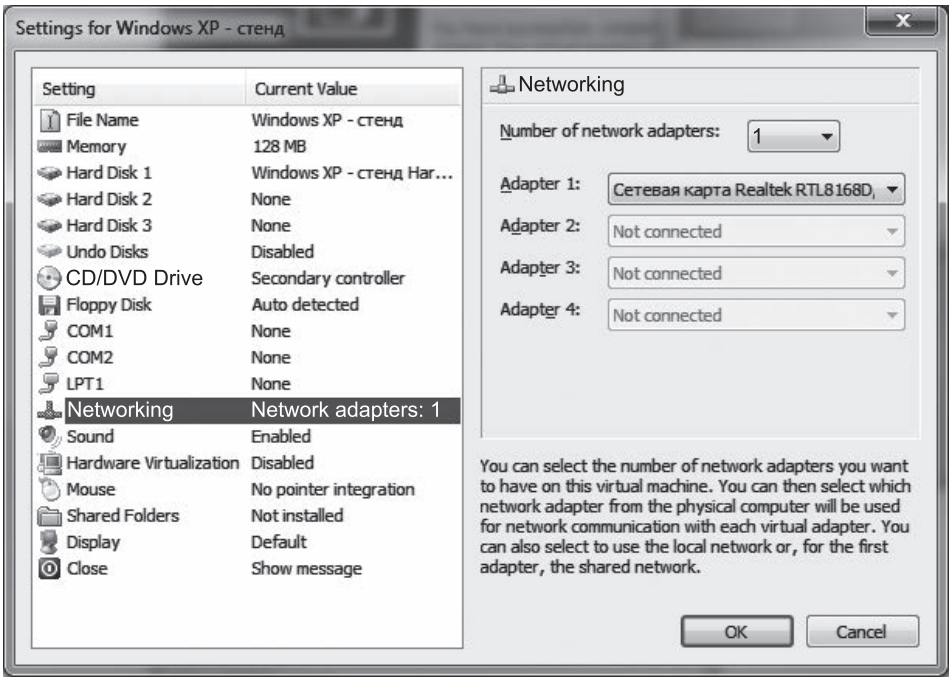

Рис. 167. Проверка некоторых настроек новой машины

В разделе **Networking** — обеспечение работы сети — вы увидите, что виртуальной машине выделена одна сетевая карта, причем она привязана к существующей, реальной карте вашего компьютера. Это важно, иначе мы не сумеем моделировать работу сети. Если на машине-хозяине нет сетевой карты, придется выбрать адаптер **Local** и настроить аналогично вторую виртуальную машину, чтобы можно было работать.

6. Запускаем машину (нажимаем кнопку **Start**) и видим, что она загружается «как настоящая» и ведет себя точно так же, как машина без ОС: пытается ее найти и сообщает об ошибках.
7. Теперь можно подключить CD-привод и установить ОС. Для этого в разделе CD можно выбрать либо подключение **(Захват)** реального дисковода для чтения оптических дисков, либо подключение файла с образом установочного диска ОС.
8. После этого машину можно перезагрузить: меню **Action**, команда **Reset**.
9. Установим средства взаимодействия виртуальной машины с машиной-хозяином. Для этого в окне запущенной виртуальной машины в меню Action выберите пункт **Install or Update Virtual Machine Additions**. Появится сообщение о том, что

будет вставлен диск и начнется установка — как на обычной машине.

Если ваша машина подключена к сети, то потребуется одна виртуальная машина, которую мы будем настраивать и на которой будем проверять результаты с основной машины.

Если подключения к сети нет, то виртуальных машин потребуется две: одна для работы, другая для проверки. Необязательно снова проходить всю процедуру установки. Достаточно выключить готовую машину, скопировать два файла (с расширениями vmc и vhd) в другой каталог и подключить вторую машину как существующую.

Потом ее придется переименовать (уже в настройках самой машины), но ставить все «с нуля» не придется.

10. Проверьте взаимодействие основной машины с виртуальной (или взаимодействие двух машин).

Запустите настроенную при подготовке виртуальную машину и узнайте ее IP-адрес. Для этого в контрольной панели вызовите настройку сетевых подключений, на подключении по локальной сети нажмите правую клавишу мыши и вызовите раздел **Состояние**. В результате на закладке **Поддержка** увидите IP-адрес (рис. 168).

11. Проверьте доступность виртуальной машины с помощью команды **ping** с основной машины (рис. 169).

Если вы установили более современную версию ОС, возможно, вы не сможете проверять доступность таким способом.

Для дальнейшей работы нам понадобятся следующие программы:

- Сканер уязвимостей PT-Checks (http://www.ptsecurity.ru/download/pt-check-09-001-ru.zip).
- Программы исправления безопасности, устраняющие уязвимости, которые может обнаружить сканер: их можно скачать с сайта Microsoft, выбрав свою ОС (ссылки и описание: http://www.securitylab.ru/news/368759.php).
- Свободно распространяемый антивирусный сканер CureIt. Скачать ее можно с сайта http://free.drweb.com/ Точное название файла программы дать не получится — это программа сканер; чтобы помешать вирусам блокировать ее применение, программа будет менять название при каждом скачивании.
- Сетевой анализатор Wireshark (http://www.wireshark.org/download.html).

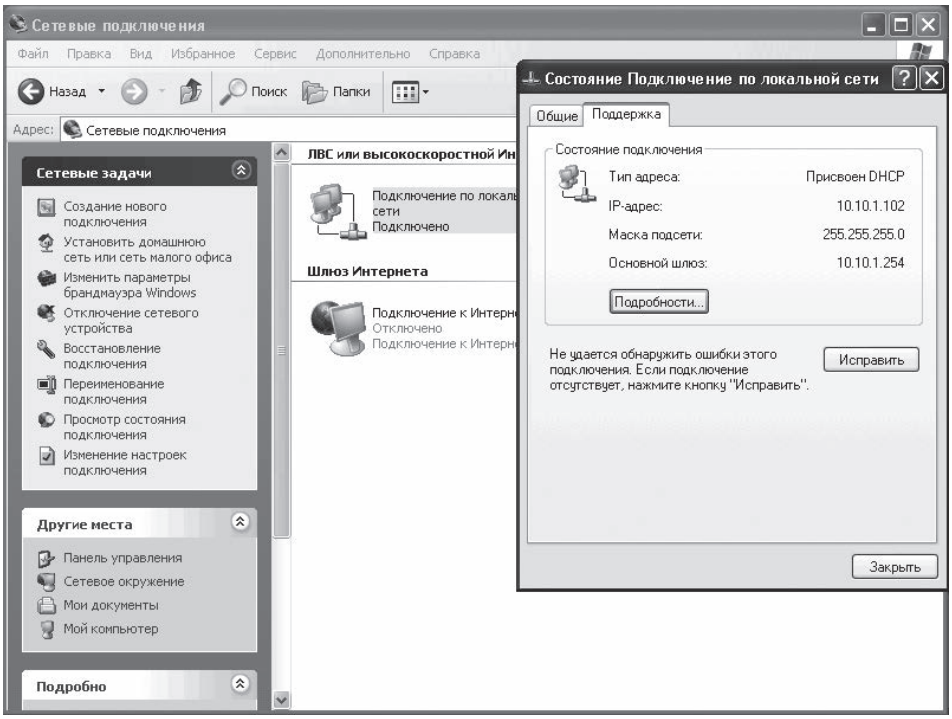

Рис. 168. IP-адрес виртуальной машины

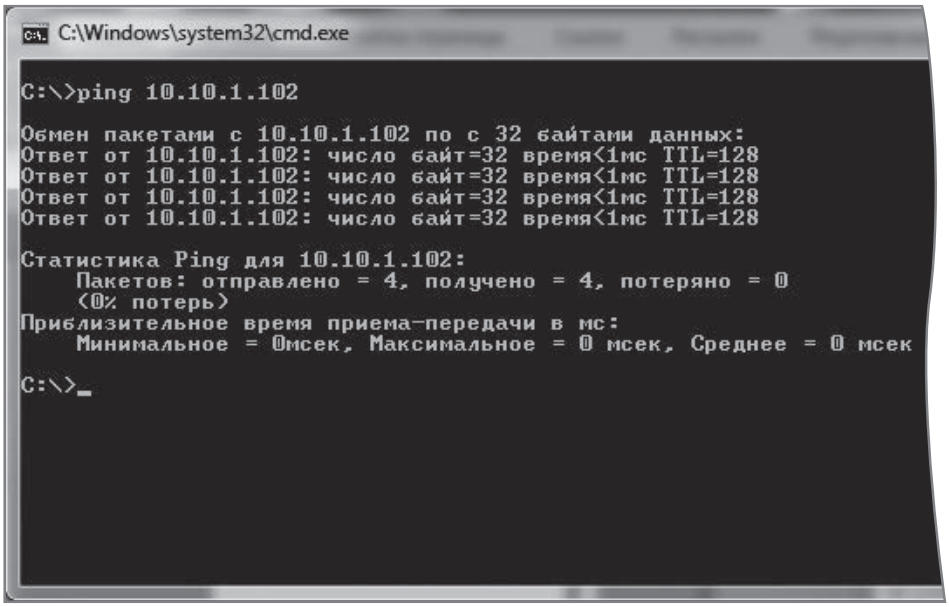

Рис. 169. Проверка доступности виртуальной машины

Средства защиты: сетевой и транспортный уровни

Теперь мы можем приступить к экспериментам. Они будут структурированы под 4-уровневую модель организации передачи данных в сети – модель DOD.

Пользователь, как правило, не может выбирать меры защиты на уровне доступа к среде передачи. Поэтому мы пропустим этот уровень и начнем экспериментировать на более высоком уровне.

Меры защиты, предпринимаемые на сетевом и транспортном уровнях, объединены в общий раздел по двум основным причинам:

1) протоколы именно этих двух уровней — наиболее общая часть стека TCP/IP, которая так или иначе используется на всех узлах среды Интернет;
2) основные программы, которые позволяют защитить информацию на этих уровнях, — комплексные и используют информацию этих двух уровней в совокупности.

Напомним, что сети-посредники не интересуются ни содержанием, ни тем более целью отправки фрагментов. Задача протоколов сетевого уровня — доставить отправленные данные по назначению, т. е. узлу-получателю.

Шлюз не обязан проверять «обратные» адреса и уж тем более определять, верные ли они.

Данных об узле отправления и узле получения недостаточно для принятия решения о безопасности данных, поскольку, как мы знаем, на конкретном узле может работать множество клиентов и серверов. Поэтому при анализе также используются данные транспортного уровня, позволяющие определить приложение и стадию обмена.

Логичным решением, повышающим уровень безопасности, является использование специальных программ-фильтров, определяющих, откуда и к каким программам можно обращаться.

Программы, выполняющие фильтрацию данных, называются *брандмауэрами* (они же «файрволы» от английского «Firewall», межсетевые экраны).

Современные брандмауэры — сложные и многофункциональные комплексы программ, задача которых — обеспечение безопасного взаимодействия сетей. Они содержат несколько компонентов, как правило следующие.

1. Пакетный фильтр — программа, определяющая на основе правил данных сетевого и транспортного уровней, пропускать ли пакет — в сеть или из сети.

2. Proxy — посредники для протоколов прикладного уровня. В отличие от пакетного фильтра, сервер-посредник проверяет также данные прикладного уровня, например URL-адреса. Конечно, такой посредник не универсален, поэтому, как правило, это веб-посредник.
3. VPN (Virtual Private Network) — средства организации защищенных каналов.
4. Модули обнаружения атак.

Межсетевые экраны — необходимый компонент защиты во всех современных сетях. Если узлы сети независимы и нет возможности определить общую политику доступа (например, в сети провайдера), применяются персональные межсетевые экраны, защищающие одну машину. В этом случае такие средства часто интегрируют с антивирусными средствами, добавляют средства контроля за поведением конкретных приложений и т. д.

Практическая работа № 1. *Применение межсетевого экрана*

1. Запустим на основной машине утилиту поиска уязвимостей **PT-Checks** (рис. 170).

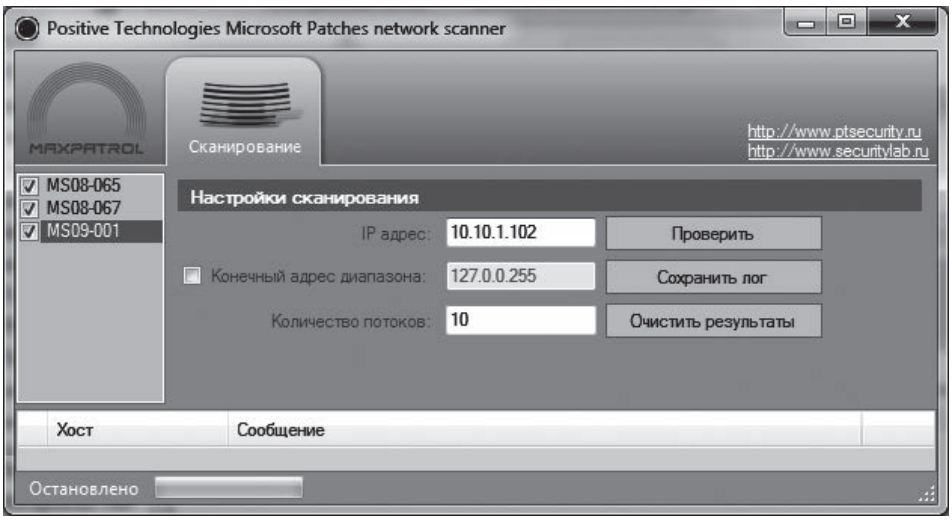

Рис. 170. Запуск на основной машине утилиты поиска уязвимостей PT-Checks

2. После запуска выберем все три уязвимости, доступные для проверки, и укажем адрес машины, на которой будем их искать. После проверки получили ответ (рис. 171).

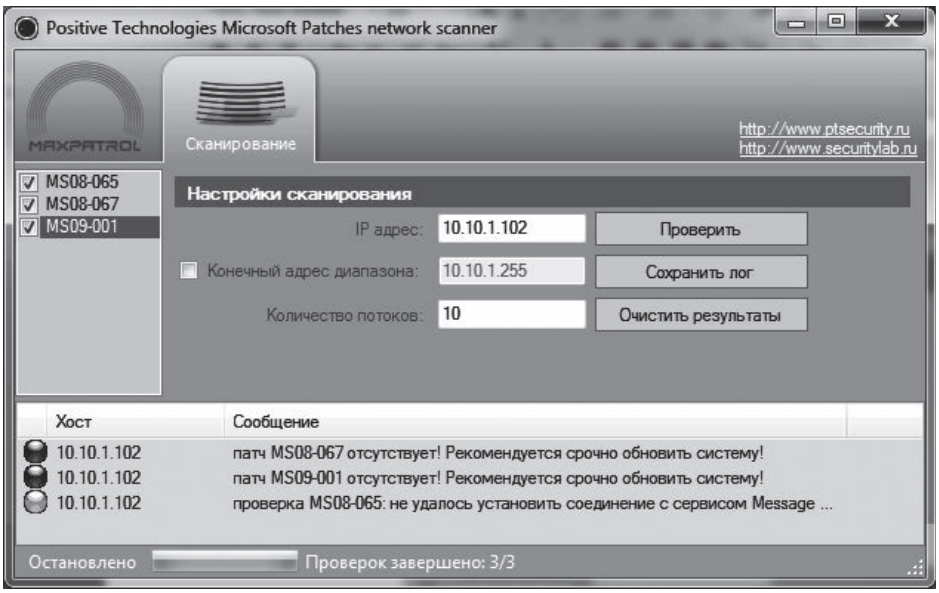

Рис. 171. Результат проверки уязвимостей

Как видим, две уязвимости из трех на виртуальной машине есть и могут быть использованы для удаленной (сканер ведь тоже обращался по сети) атаки.

3. Включим на виртуальной машине простейший межсетевой экран, который входит в комплект системы. Для этого: вызываем свойства сетевого адаптера, на закладке **Общие** вызываем **Свойства**, в свойствах на закладке **Дополнительно** вызываем окно настройки параметров брандмауэра и включаем его (рис. 172).
4. Снова запускаем проверку с теми же параметрами (рис. 173).

Как видим, уязвимости недоступны, следовательно, мы сделали далеко не все необходимое.

1. Использованный нами сканер — очень узконаправленный. Он обнаруживает только три уязвимости и не использует сложных методов обхода ограничений доступа. Таким образом, мы должны допускать, что есть и другие, неизвестные нам проблемы.
2. Уязвимости после включения никуда не делись. Мы только ограничили к ним доступ, и, если брандмауэр сделает исклю-

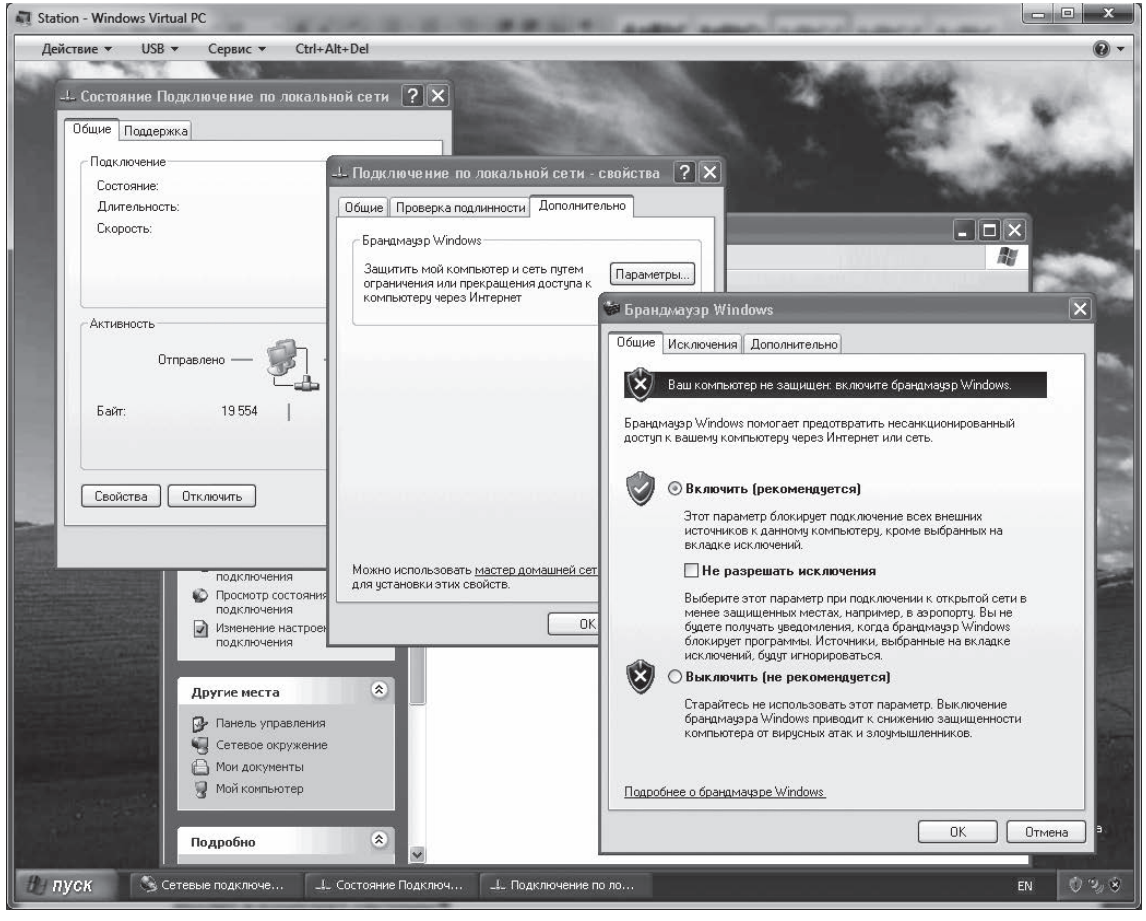

Рис. 172. Окно настройки параметров брандмауэра

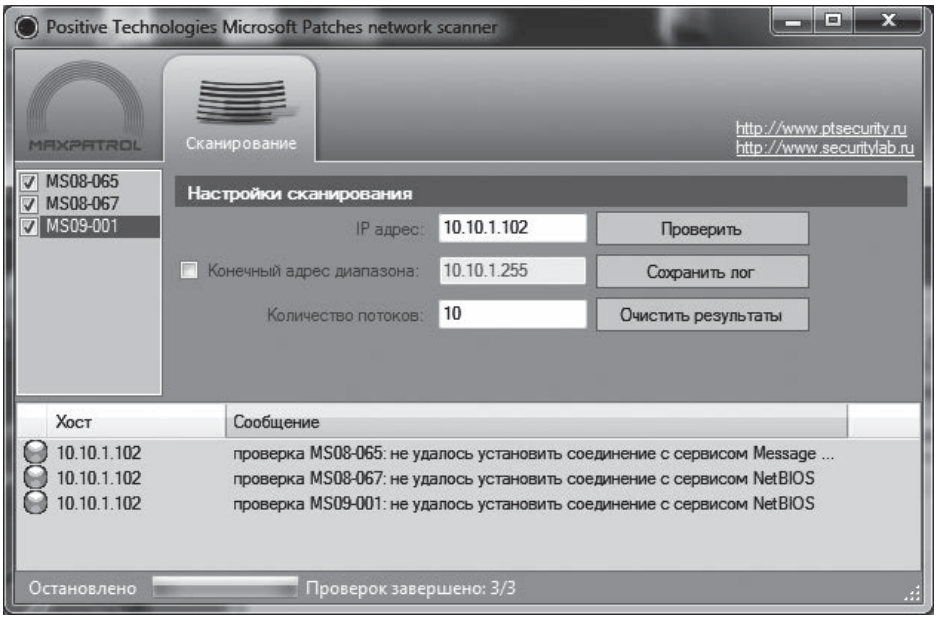

Рис. 173. Проверка уязвимостей с теми же параметрами

чения для каких-то узлов или сетей, атака оттуда станет возможной. Кроме того, между началом работы машины с сетью и запуском брандмауэра проходит некоторое время, вполне достаточное для поражения.

3. Ошибки и/или злонамеренные действия пользователя вообще не могут быть обнаружены средствами такого типа.

Практическая работа № 2. *Установка исправлений*

Следующий обязательный этап — установка обновлений, т. е. исправлений уязвимостей.

Самостоятельно сделайте следующее.

1. Установите на виртуальной машине обновления: WindowsXP-KB958644-x86-RUS.exe, WindowsXP-KB958687-x86-RUS.exe. Перезагрузите виртуальную машину, отключите брандмауэр, проверьте сканером уязвимости.
2. Установите более сложный и мощный брандмауэр Agnitum Outpost. Как он отреагирует на попытку проверить наличие уязвимостей?

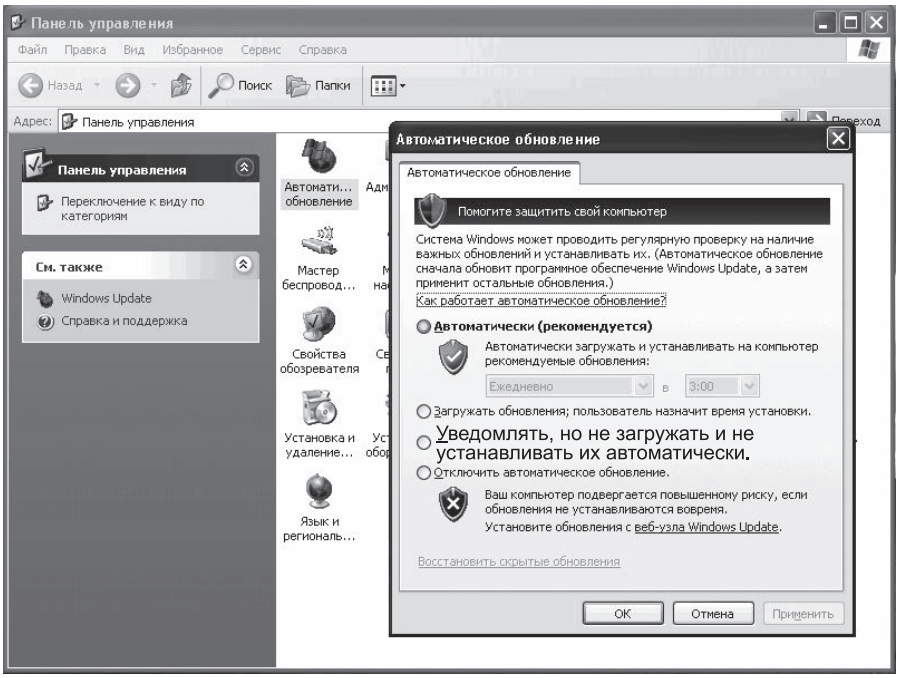

Рис. 174. Настройка работы автоматической системы обновления

Обновления к самым разным программам, устраняющие найденные ошибки, появляются постоянно. Чтобы не следить за этим самому, следует настроить работу автоматических систем обновления. Например, такая система есть в составе ОС Windows. При наличии у вас действующей лицензии система обновления будет сама скачивать и предлагать установку обновлений (рис. 174).

Настройте систему обновлений на своей виртуальной машине в режиме «Уведомлять, но не загружать обновления».

Проверить обновления можно также с помощью Internet Explorer (рис. 175).

Вызвав в меню **Сервис** пункт Windows Update, вы перейдете на специальный сайт. Вам предложат проверить наличие обновлений на вашей машине, а после проверки — список того, что может быть установлено. Поскольку на вашей машине обновления с момента формирования дистрибутива не вносились, список будет длинный. Скачивать и ставить обновления в условиях лабораторной работы не стоит, а вот в реальной ситуации — обязательно.

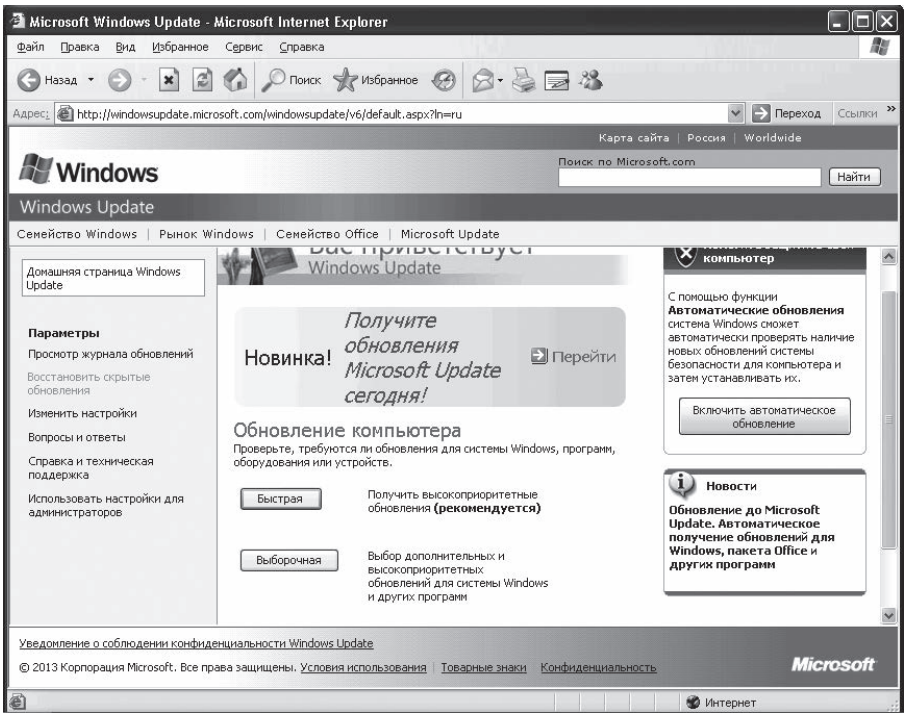

Рис. 175. Проверка обновлений с помощью Internet Explorer

Средства защиты — прикладной уровень

Поскольку самое большое количество разнообразных программ и сервисов реализовано именно на прикладном уровне, именно на этом уровне сосредоточено наибольшее количество уязвимостей и именно этот уровень чаще всего становится объектом атак[9].

На прикладном уровне обычный пользователь чаще всего сталкивается с двумя видами угроз: вредоносным программным обеспечением и социальной инженерией. Упрощенно можно сказать, что в первом случае мы сталкиваемся со специально подготовленными для нанесения вреда (а автору программы — получения пользы за ваш счет) программами, а во втором — выманиванием информации или навязыванием действий опять-таки для нанесения вреда.

Эти угрозы вполне могут совмещаться, например вас могут уговаривать запустить вредоносную программу.

Зачастую вредоносные программы называют вирусами, но это не вполне верно. Существует множество разновидностей таких

9 Или ошибок пользователей.

программ, и чаще сейчас встречаются не классические вирусы, а так называемые «троянские кони».

Основным, но не единственным средством борьбы с вредоносными программами являются антивирусные комплексы. Современные комплексы такого рода включают в себя:

1) ядро — основные средства поиска и борьбы с вирусами;
2) базу данных вирусов — базу данных фрагментов вирусов, по которым их можно опознавать;
3) антивирусный сканер — программу, которая по команде пользователя, используя ядро, проверяет (сканирует) заданные файлы;
4) антивирусный сторож — программа, постоянно находящаяся в памяти и проверяющая файлы в момент обращения к ним;
5) систему обновления всех компонентов;
6) программы проверки почтовых сообщений, ссылок на файлы в сети и т. п.

Особо надо отметить, что вредоносные программы появляются и совершенствуются постоянно, поэтому любой антивирусный комплекс должен постоянно обновляться, причем крайне желательно — каждый день.

Проиллюстрируем работу программы на примере поиска «тестового» вируса. Если быть точным, то ищется не вирус, а только сигнатура, — и никаких вредоносных действий не будет.

Практическая работа № 3. *Реакция антивирусной программы на появление предположительно вредоносного файла*

1. Установите программу DrWeb. Получите пробный ключ. Перезагрузите машину.
2. Найдите в каталоге DrWeb (скорее всего: C:\Program files\DrWeb) файл test.txt. Откройте его в текстовом редакторе «Блокнот»
3. Выделите и скопируйте в буфер обмена строку: X5O!P%@AP[4\PZX54(P^)7CC)7}$EICAR-STANDARD-ANTIVIRUS-TEST-FILE!$H+H*
4. Создайте новый файл[10] и вставьте туда этот текст.
5. Сохраните его под именем test.com

[10] Сохранить test.com нужно именно как текстовый файл, т. е. без каких-либо добавлений. С учетом печального опыта преподавания, напомним, файл Microsoft Word текстовым НЕ является.

Рис. 176. Старт проверки DrWeb CureIT

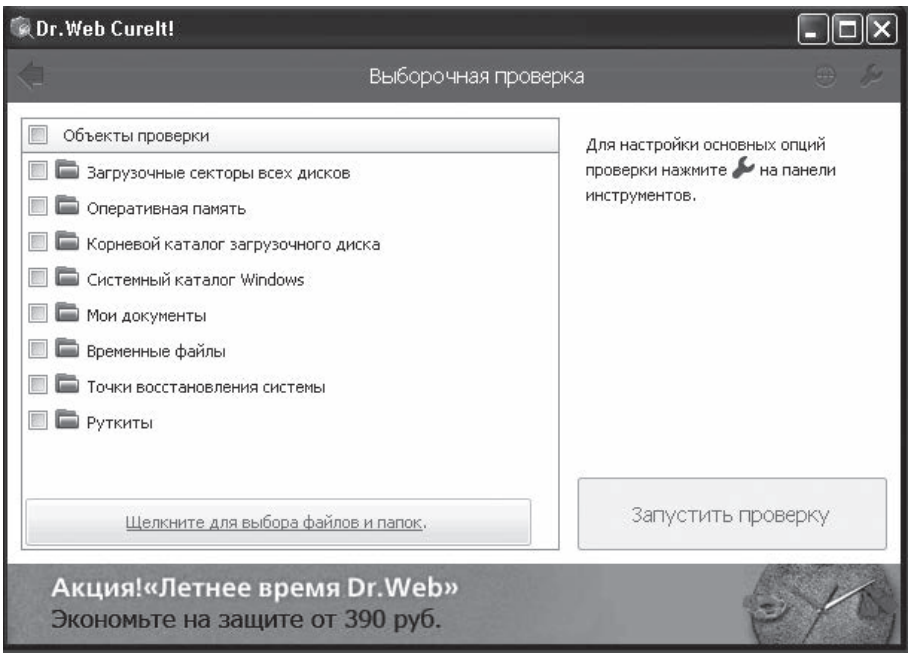

Рис. 177. Выбор объектов для проверки

6. Запустите программу CureIt. После формальных ответов на несколько запросов вы увидите примерно то, что изображено на рис. 176.
7. Перейдите по ссылке **Выбрать объекты для проверки** (рис. 177).

Перейдите по ссылке **Щелкните для выбора файлов и папок**, на которой находится каталог с тестовым примером, поставьте галочку рядом с каталогом и начните проверку. Проверка начнется после нажатия кнопки **Запустить проверку**.

8. После того как проверка закончится, выберите в списке обнаруженных вредоносных программ EICAR и выберите действие **Удалить** — поскольку лечить там нечего. Можно выбрать действие слева от списка, можно — вызвав контекстное меню у самого файла. Нажмите кнопку **Обезвредить** (рис. 178).

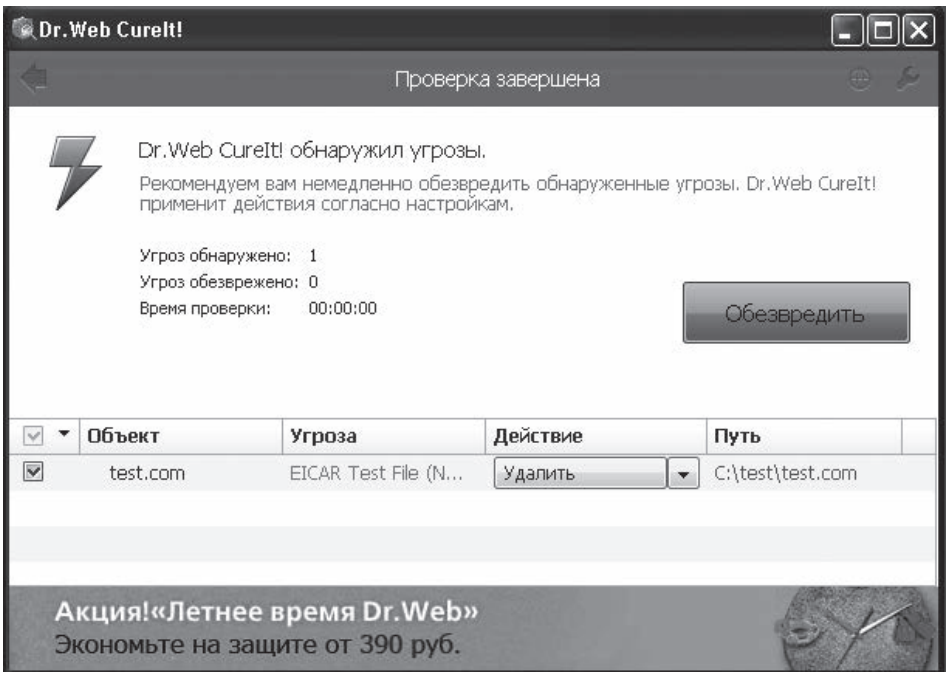

Рис. 178. Выбор действия

Защита пароля и разграничение доступа

На прикладном уровне для обеспечения защиты самых разных личных данных очень часто используются регистрационные записи пользователей. Регистрационная запись — это средство, позволяющее определить, кто и какие данные видит и что может с ними делать.

Чтобы пользователь не мог злонамеренно воспользоваться чужой записью, работа в самых разных системах начинается с аутентификации: установления соответствия между человеком и его записью. Чтобы выполнить ее, как правило, надо предъявить какую-то уникальную информацию, фактор, отличающий «правильного» пользователя от всех остальных.

Самый простой и распространенный способ аутентификации — однофакторная аутентификация по паролю. По причине этой простоты и распространенности, пароль — самая распространенная мишень для злоумышленников.

Простые угрозы для пароля: подбор или кража. Чем короче и проще пароль, тем больше вероятность того, что, попробовав наиболее распространенные и предсказуемые варианты (ваше имя или прозвище, например), злоумышленник его подберет.

Что делать, чтобы этого не произошло?

1. Не использовать короткие, простые и предсказуемые пароли. Как правило, рекомендуют использовать пароли не короче восьми символов, содержащие и цифры, и буквы, причем желательно в разных регистрах. Если затрудняетесь придумать пароль сами, можно воспользоваться генераторами паролей, например, по адресу: http://pasw.ru/
2. Регулярно (не реже одного раза в два месяца) менять пароль.
3. Не давать доступа к паролю посторонним: не кричать его через людное помещение, не записывать на листочках, не предоставлять доступа к замене пароля через простой вопрос.
4. Не пользоваться учетными данными в недоверенных местах (см, например, практическую работу № 4 «Перехват передаваемых данных»).

Перехват и защита от перехвата

Технология Ethernet и различные ее варианты — сейчас одна из самых популярных и активно используемых не только для офисных сетей, но и для домашних (точнее, домовых — охватывающих один или несколько жилых домов). Не вдаваясь в подробный анализ технических требований, скажем только, что причина ее популярности — сравнительно высокая скорость при недорогом оборудовании и несложном монтаже.

Для понимания возможных угроз при использовании таких сетей нужно представлять себе принцип их работы. Основной принцип обмена данными в сетях Ethernet получил называние CSMA/CD (Carrier Sense Multi Access/Collision Detect).

Проще всего представить себе работу такой сети как разговор в общей комнате. Во-первых, все прислушиваются к происходя-

щему (контролируют несущую волну). Во-вторых, каждый имеет равные возможности что-либо сказать (множественный доступ). В-третьих, каждый перед началом разговора прислушивается — не говорит ли кто-то еще? Если не говорит, то он пытается сказать что-то сам. Возможна ситуация, когда двое начинают говорить одновременно (определяют коллизию — столкновение). В этом случае они оба замолкают, каждый ждет случайный промежуток времени и снова пытается что-то сказать.

В современных сетях Ethernet самым популярным способом организации является использование топологии «звезда» на базе устройств-повторителей (Hub) или более сложных устройств-коммутаторов (Switch). Задача этого устройства — повторить сигнал, полученный от одного порта на все остальные или, если это более сложное устройство, передать данные только на тот узел, которому они предназначены. Таким образом, сохраняется «общая» комната.

Каждый узел при таком способе обмена должен иметь уникальный идентификатор, для того чтобы выделять из общего потока («разговора») свои пакеты[11]. Такой идентификатор имеет каждый адаптер, и называется он MAC-адрес (Multi Access Channel — канал множественного доступа). MAC-адрес (6 байтов) присваивается каждому адаптеру на заводе[12] и должен быть уникальным.

Очевидная угроза в таких сетях — перехват данных. Для того чтобы получить чужие данные, достаточно перевести свою сетевую карту в «неразборчивый» (promiscuity) режим. В этом случае можно получать и анализировать не только «свои» кадры, но и чужие. Кроме этой очевидной угрозы есть и менее очевидная – можно посылать кадры с чужого адреса.

Практическая работа № 4. *Перехват передаваемых данных*

1. Установите на своей виртуальной машине анализатор Wireshark (рис. 179). Учтите, что для ее установки и использования необходимо иметь административные права.
2. Запустите программу на перехват пакетов в течение 15 с.

---

[11] На этом уровне они называются «кадрами», поскольку на самом деле речь идет о некотором «участке» сигнала.

[12] MAC-адрес можно изменить, причем сделать это несложно. В некоторых сетях такие действия запрещены, поскольку на их основе пытаются контролировать доступ к сети или препятствовать атакам.

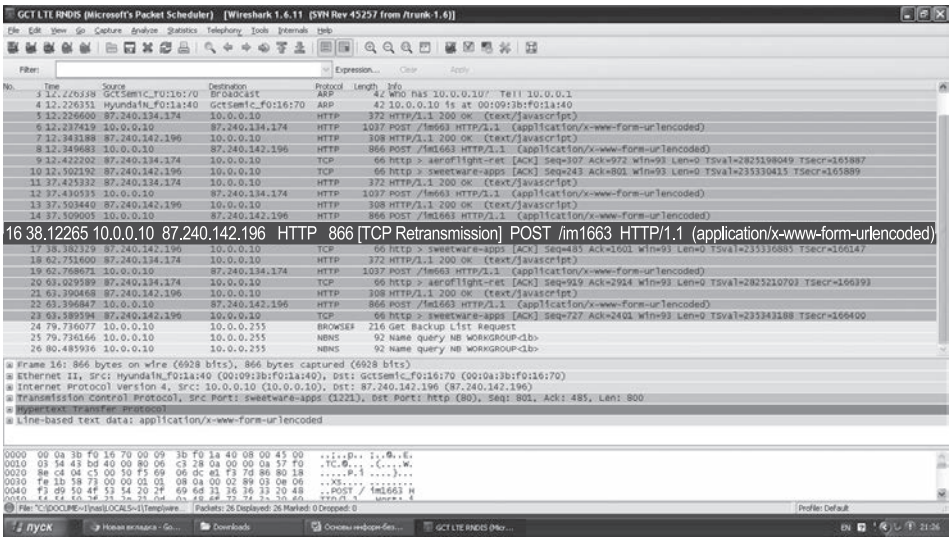

Рис. 179. Установка анализатора Wireshark на виртуальной машине

3. Обратите внимание на адреса отправителей и получателей. Если сеть функционирует на базе обычного концентратора, то, помимо пакетов, содержащих ваш адрес, там будут и другие пакеты. Изучите интерфейс программы, чтобы понимать, какие данные где отображаются.
4. Откройте на любой машине страницу форума http://www.rutracker.org (или любого другого). Проанализируйте HTML-код страницы, чтобы обнаружить имя поля для ввода пароля. Вас интересует значение атрибута Name.

Замечание. *Тег этого поля, скорее всего, будет иметь вид: <input type="password" name=… , хотя порядок атрибутов в теге может меняться. Но другого типа поля для ввода паролей язык HTML не предусматривает, а значит ухищрения могли бы привести к сокращению спектра программ, с которыми интерфейс работает.*

5. Задайте в программе-перехватчике условие: отбирать при перехвате только те пакеты, которые направлены с адреса вашей машины и содержат найденное имя поля.

Внимание! Способ, которым фильтруются пакеты, зависит от конкретного программного продукта. В комплексе WireShark возможны два решения: либо отбирать пакеты во время перехвата (фильтр перехвата), либо наложить фильтр на все перехваченные пакеты.

6. Запустите перехват и обратитесь к форуму.
7. Остановите перехват и найдите в перехваченных пакетах свой пароль (рис. 180).

> Внимание! Совершенно необязательно вводить реальное имя пользователя и реальный пароль. Для того чтобы убедиться в работоспособности перехвата, достаточно попытки входа.

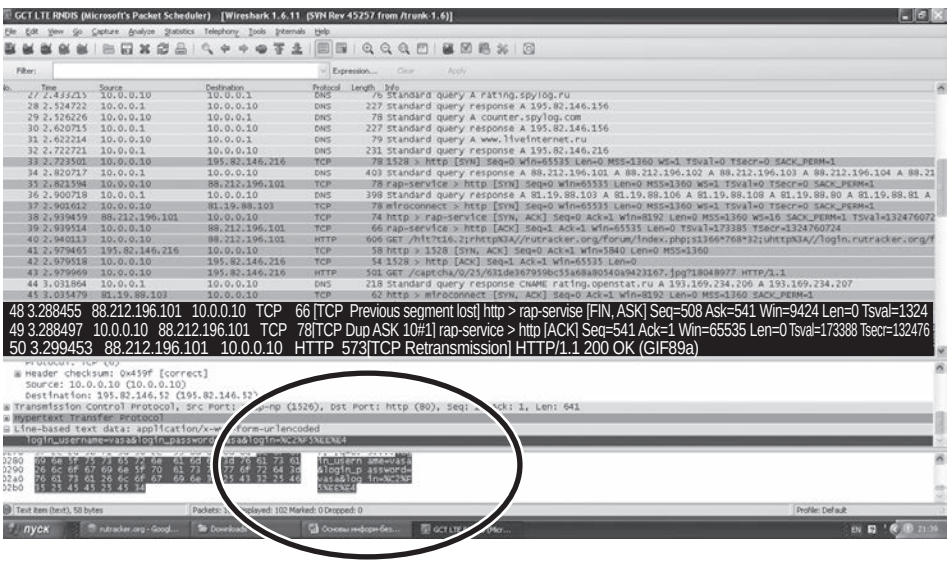

Рис. 180. Поиск своего пароля в перехваченных пакетах

Следует понимать, что перехват данных необязательно выполняется именно таким способом. Например, те же данные будут доступны сотрудникам провайдера (причем не только вашего), хозяевам сайта и хостинговой компании и т. д.

Возможность перехвата — это не только перспектива утечки, но и возможность подменить существенные данные и действовать от вашего имени.

Защитой от таких неприятностей является шифрование.

Шифрование

Вопрос, который у нас возникает практически сразу, как только мы вообще задумываемся о безопасности в сети, это: «А не получит ли мои данные кто-то чужой?» Чужим в этой ситуации оказывается кто угодно, причем даже необязательно злоумышленник.

Это все равно неприятно, а иногда и опаснее, чем просто недоброжелатель. Только что мы видели, что, если не предприняты специальные меры, доступа к каналу данных достаточно, чтобы получить самую чувствительную информацию.

Проблема безопасной передачи данных — очень старая, и, как мы уже писали в учебнике, основной способ ее решения — шифр, т. е. такое преобразование сообщения, при котором уполномоченный агент может получить полезную информацию, а все остальные — нет.

Необходимый элемент такого преобразования — ключ, т. е. такое заранее (до передачи) известное сообщение, которое требуется для преобразования нашего полезного сообщения, — и зашифровать и расшифровать.

Ключевой проблемой при организации такой работы в сети является то, что мы не можем просто передать сообщение-ключ, поскольку заранее не знаем своих абонентов и не имеем возможности с каждым из них встречаться безопасным способом.

При этом нам надо не только договориться о передаче данных, но и быть уверенным, что абонент не сможет отказаться от своих действий, т. е. сказать, например: «Ты сам этот договор сочинил, я его не видел!».

Таким образом, для того чтобы организовать защищенную передачу данных, необходимо:

1) согласовать алгоритмы шифрования;
2) разработать процедуру обмена ключами;
3) разработать процедуру удостоверения «личности».

Решение заключается в применении защищенного варианта протокола HTTP, получившего название HTTPS. Не описывая HTTPS подробно, скажем, что он позволяет двум абонентам согласовать ключ шифрования и передавать данные по защищенному каналу, т. е. в зашифрованном виде. Ключевой вопрос — как удостовериться, что абонент тот, за кого себя выдает? Для этого используется инфраструктура открытого ключа.

Ключевой компонент этой технологии — *система шифрования с открытым ключом*, в которой для шифрования используется один ключ, а для расшифровки — другой. Ключ для расшифровки считается публично доступным (открытым).

Рассмотрим, как организована работа этой системы с технической точки зрения.

Каждый участник обмена должен иметь сертификат — документ (хранится чаще всего в виде файла, но может быть просто записью в базе данных), в котором содержатся:

- имя субъекта обмена (человека, узла сети, программы, группы узлов и т. д.);
- открытый ключ субъекта;
- срок действия сертификата;
- разрешенные действия.

Этот сертификат передается другому участнику, чтобы удостоверить, что он тот, за кого себя выдает. Очевидный вопрос: а что мешает подделать такой сертификат? Ответ — электронно-цифровая подпись (ЭЦП). С логической точки зрения, ЭЦП устроена несложно: сформировав документ, мы создаем его хэш-код (для документа он, конечно, длиннее, чем для слова, и алгоритмы используются другие, но принцип примерно тот же), а потом шифруем этот хэш-код. Документом может быть HTML-страница, файл Microsoft Word и т. д.

Теперь проверить неизменность данных может любой участник — открытый ключ мы даем всем, а вот изменить — никто, кроме того, у кого закрытый ключ есть. Возникает вопрос: а доверять-то сертификату как, злоумышленник ведь тоже может его сформировать? Для этого мы используем удостоверяющие центры. Эти центры получают запросы на сертификаты и формируют их, подписывая своим ключом. Нам уже не надо «знать» всех участников, достаточно знать центры (точнее — их сертификаты).

Посмотрим, как это работает.

Практическая работа № 5. *Попытка перехвата данных по протоколу HTTPS*

1. Обратимся к сайту https://mail.ru (можно и к другому), как и в предыдущей работе, найдем там поля ввода имени и пароля.
2. Запустим программу перехвата, настроим фильтр и попробуем обратиться к сайту (точно так же, как в работе № 4).
3. Результат мы не показываем, потому что его не будет. Мы использовали для пересылки страницы mail.ru защищенный протокол https, и никакие данные в открытом виде не передавались.

> Внимание! При открытии страницы знак «замок» находится в адресной строке (рис. 181).

В разных браузерах он выглядит по-разному, но смысл один — это «закрытый» канал.

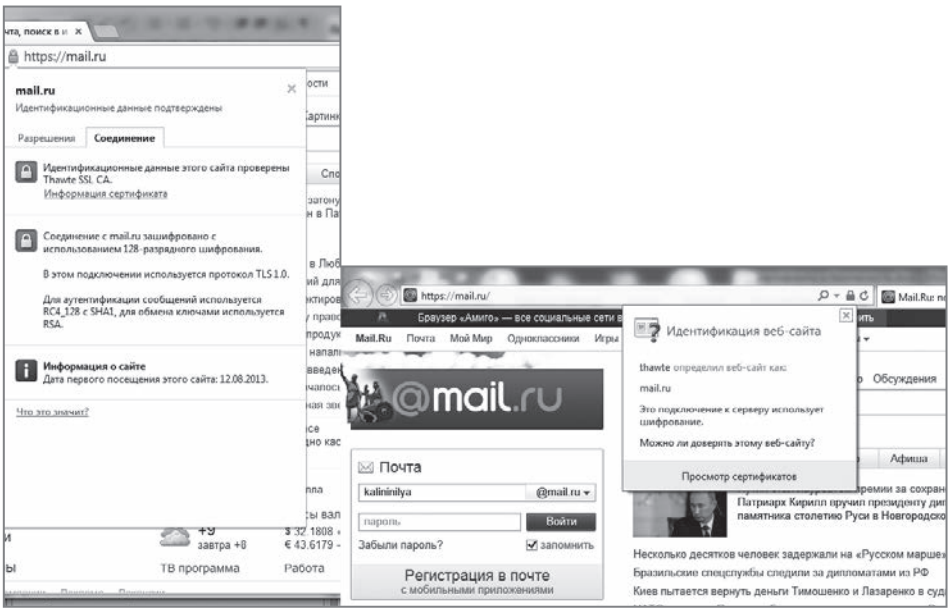

Рис. 181. Обозначение защищенного канала в разных браузерах

Щелкнем на замке и вызовем свойства сертификата (рис. 182). Видно, что сертификат заверен одним из центров Thawte, причем не напрямую, а цепочкой доверия.

Конец этой цепочки обязательно должен быть известен нашему браузеру как доверенный корневой центр. Посмотреть его можно в специальном хранилище (рис. 183).

4. Запускаем консоль управления **mmc**.
5. Добавляем оснастку **Сертификаты** (меню **Файл/Добавить оснастку**), указываем, что работать будем с сертификатами своей учетной записи пользователя (рис. 184).
6. Открываем ветку **Доверенные корневые сертификаты** и видим там корневой сертификат Thawte (рис. 185). Открываем его и видим его параметры (рис. 186).
7. Теперь посмотрим, как будет вести себя система, если сертификат НЕ ПРОЙДЕТ проверку. Обратимся к сайту https://uc-em.ru (Московский региональный удостоверяющий центр) и увидим сообщение о проблемах с сертификатом (рис. 187).

Что не так? Щелкнем два раза на замке, вызовем свойства сертификата (рис. 188):

- узел сам себе выдал сертификат;
- узел свое имя определил только для себя — как локальное;
- срок действия сертификата истек, но его не обновили.

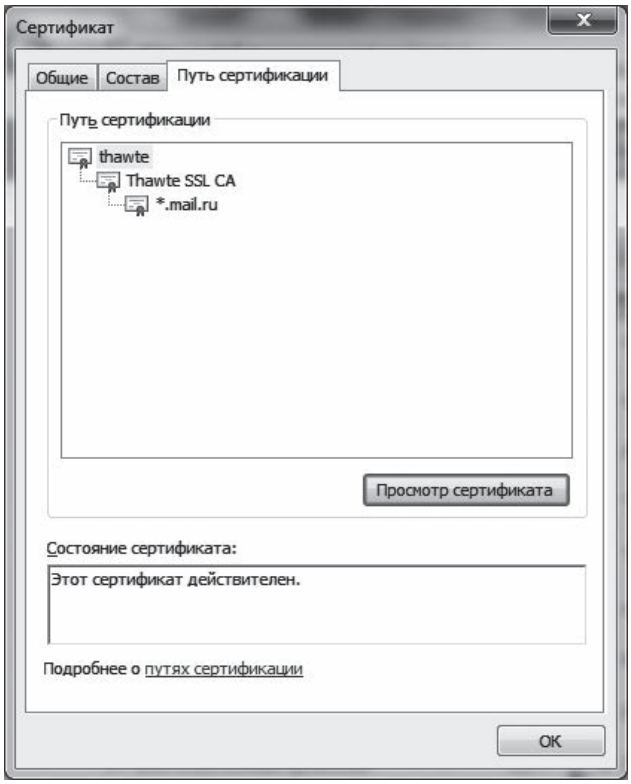

Рис. 182. Цепочка доверия в свойствах сертификата

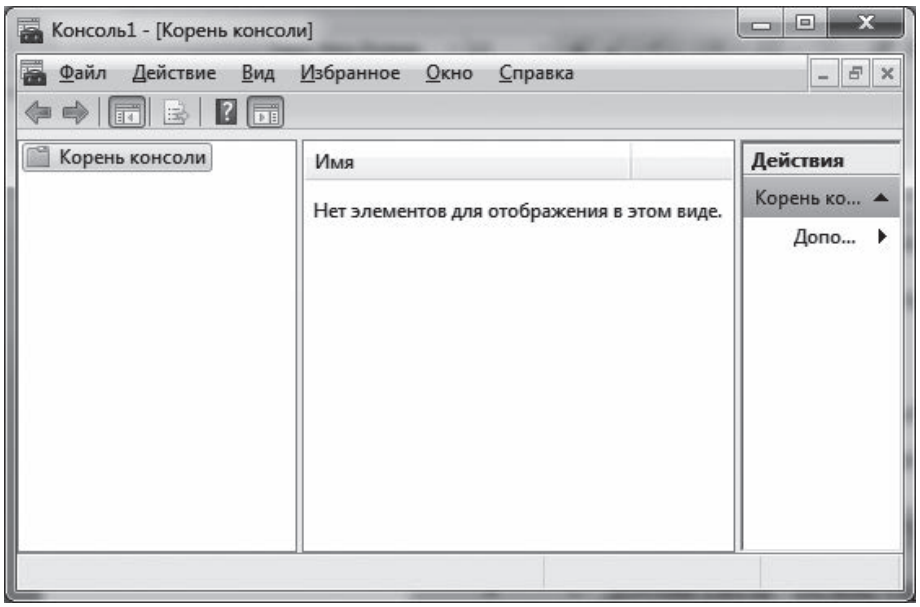

Рис. 183. Доверенный корневой центр

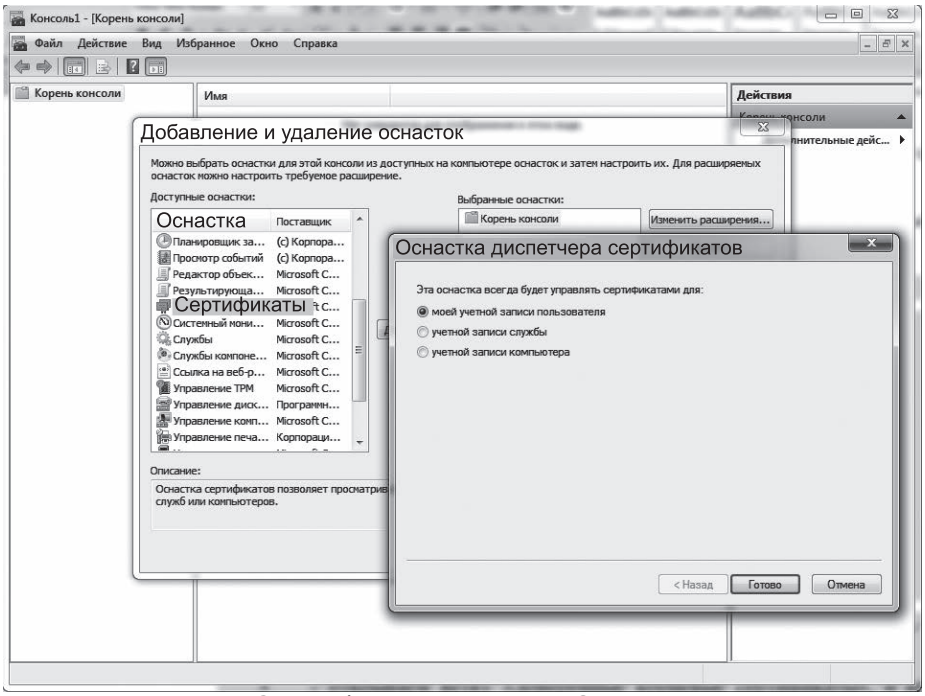

Рис. 184. Добавление оснастки **Сертификаты**

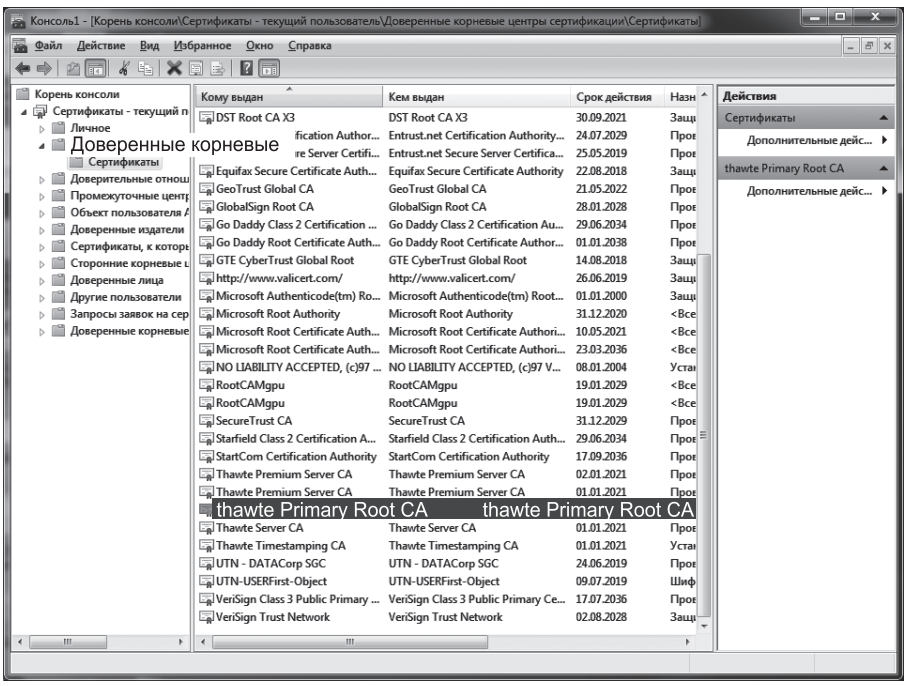

Рис. 185. Корневой сертификат Thawte в общем списке

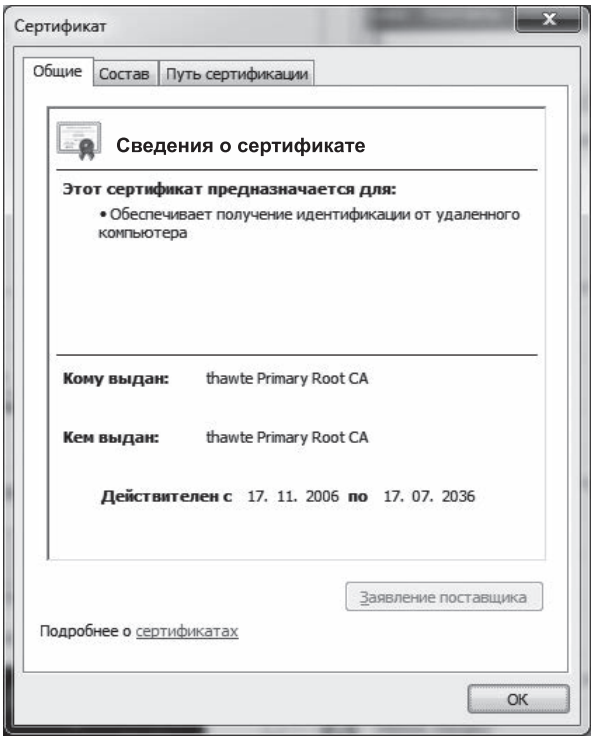

Рис. 186. Параметры корневого сертификата Thawte

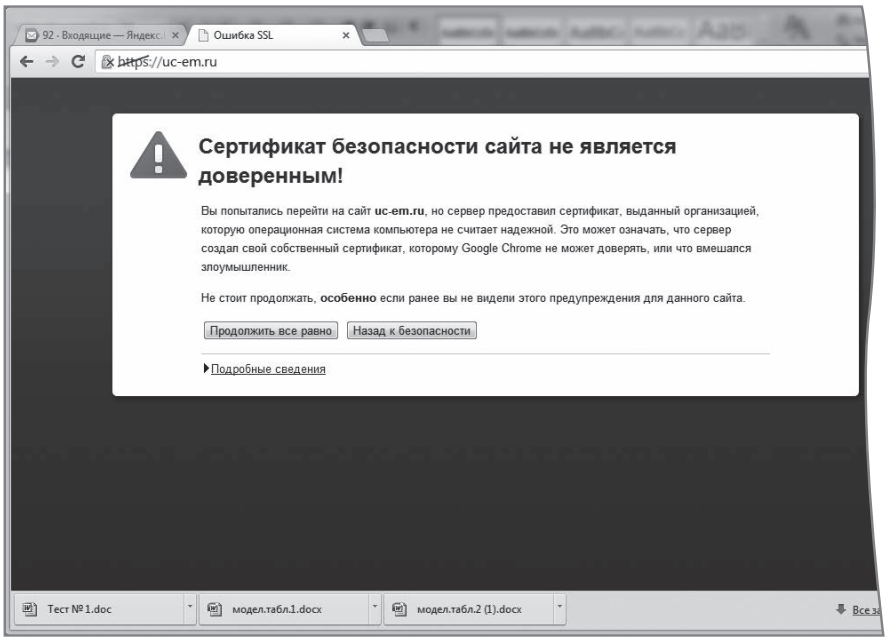

Рис. 187. Сообщение о проблемах с сертификатом

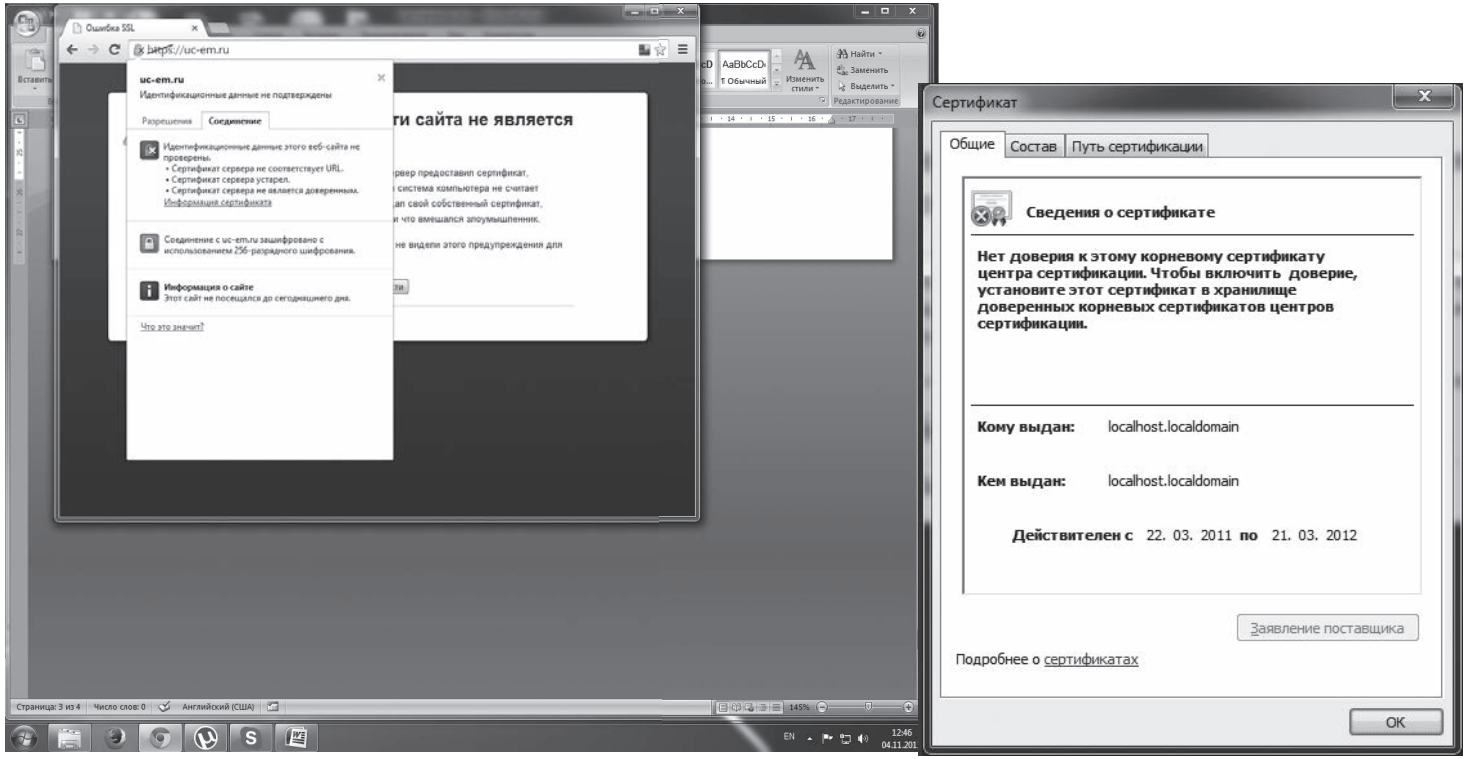

Рис. 188. Свойства сертификата

Очевидно, разработчики сайта не очень заботились о том, чтобы этот протокол у них работал полностью. Основные задачи организация, видимо, решает без участия сайта. Возможно (если бы нам требовалось, например, выполнить там оплату), стоит задуматься, продолжать ли работу — это вполне может быть кто угодно.

Откроем этот сайт по обычному протоколу HTTP и получим оттуда сертификат этого удостоверяющего центра — именно этот центр позволяет проверять все сертификаты, например во время торгов.

Первое, что нам сообщат при открытии сертификата, что он поврежден. Дело в том, что его хэш нельзя проверить — алгоритм расчета и шифрования изначально неизвестен. Юридической значимостью в РФ обладают только те ЭЦП, которые сформированы по стандарту ГОСТ Р 34.11/34.10—2001. В стандартную поставку Windows он входить не может.

Придется установить поддержку этого стандарта, т. е. программы, которые обеспечат его поддержку. Для личных некоммерческих целей мы можем воспользоваться продуктом ViPNet CSP[13] (http://www.infotecs.ru/products/catalog.php?SECTION_ID=&ELEMENT_ID=2096) — для скачивания и последующей активации достаточно указать свои данные на сайте.

В России создана целая система корневых удостоверяющих центров, и их список можно найти по адресу: http://www.reestr-pki.ru/tsl.html

Посмотрите их сертификаты, а сертификат UC-EM мы теперь (имея поддержку ЭЦП по ГОСТ) можем внести в список доверенных.

8. Скачиваем сертификат с сайта, открываем его (рис. 189). Обратите внимание: сертификат пока считается недоверенным.
9. Нажимаем кнопку **Установить сертификат** и окне выбора хранилища выбираем **Доверенные корневые центры** (рис. 190).
10. Подтверждаем установку (рис. 191).
11. При открытии сертификата сообщений об ошибке не будет, несмотря на то что центр сам себе выдал сертификат (рис. 192).

В каждом регионе РФ есть свой удостоверяющий центр, поэтому вы можете точно так же найти и поставить сертификат «своего» центра.

---

[13] Стоит обратить внимание, что заниматься разработкой криптокомпонентов и применять их в нашей стране имеют право только компании, получившие лицензию ФСБ, причем лицензируются и компании, и продукты. Удивительным образом это приводит к тому, что такие продукты бесплатно не предоставляются. Кто-то в любом случае оплачивает лицензию.

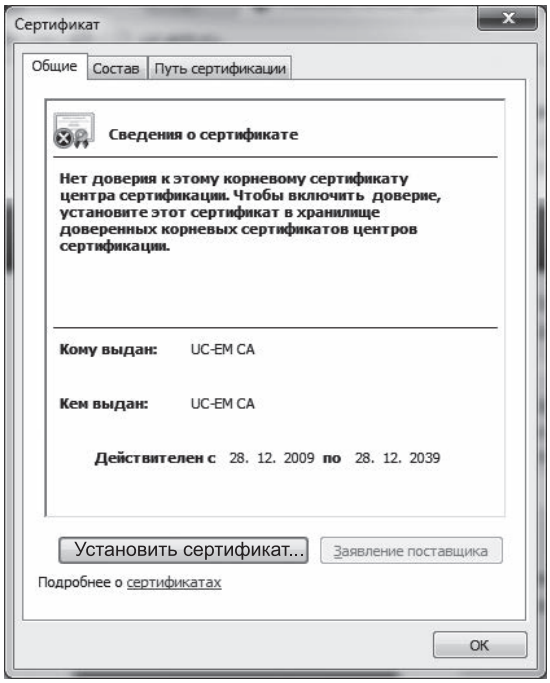

Рис. 189. Недоверенный сертификат

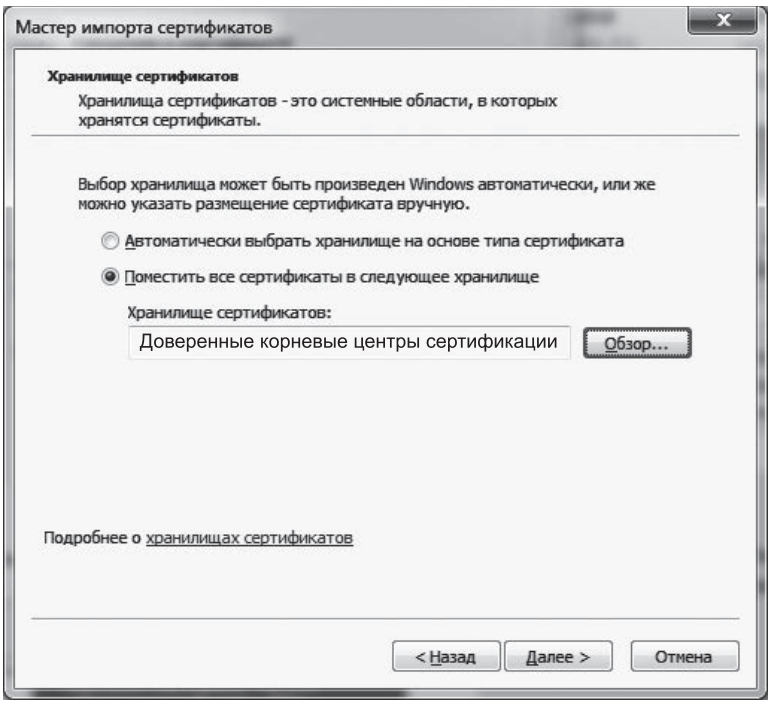

Рис. 190. Окно выбора хранилища

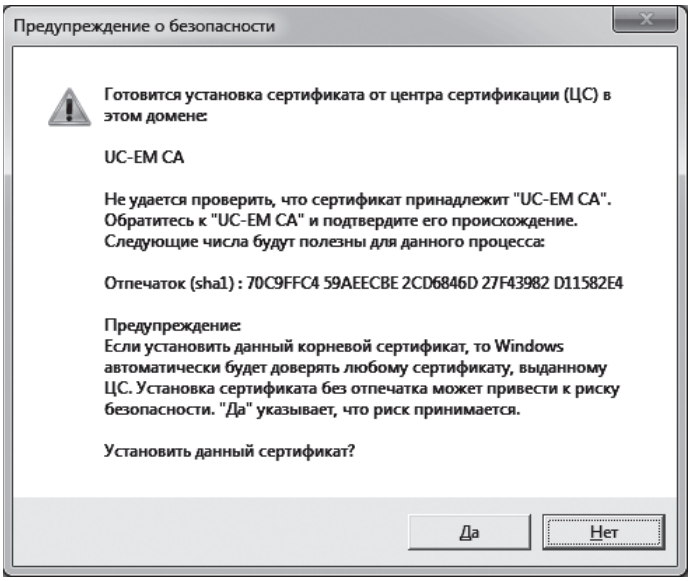

Рис. 191. Подтверждение установки сертификата

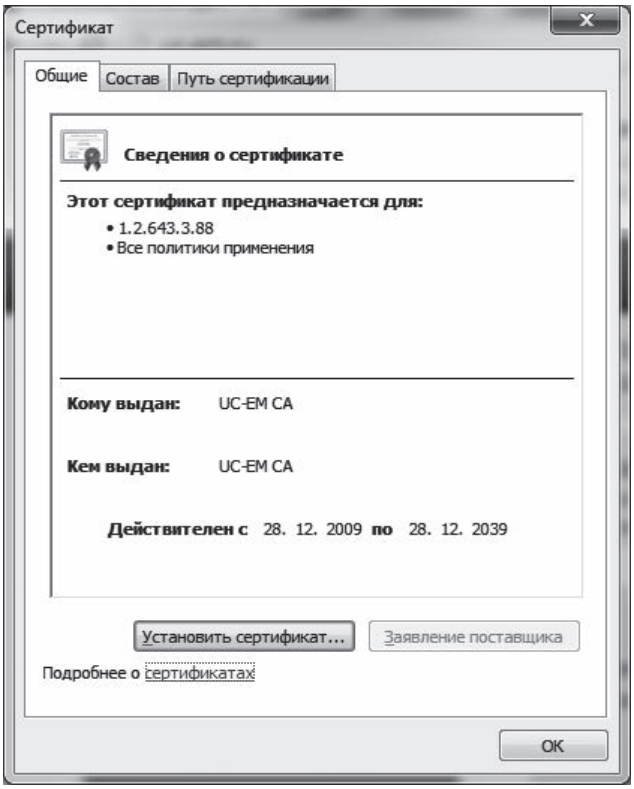

Рис. 192. Состояние проверки сертификата после установки доверенного корневого удостоверяющего центра

Для заметок