ВИКТОР ПЕСТРИКОВ
АРТУР МАСЛОБОЕВ

DELPHI

НА ПРИМЕРАХ

ОСНОВНЫЕ КОМПОНЕНТЫ

МЕТОДИКА НАПИСАНИЯ И ОТЛАДКИ ПРОГРАММ

ГРАФИЧЕСКИЕ ВОЗМОЖНОСТИ DELPHI

70 ПРИМЕРОВ С ПОДРОБНЫМИ РЕШЕНИЯМИ

bhv

Виктор Пестриков
Артур Маслобоев

DELPHI

НА ПРИМЕРАХ

Санкт-Петербург
«БХВ-Петербург»
2005

УДК 681.3.06
ББК 32.973.26-018.2
П28

Пестриков В. М., Маслобоев А. Н.

П28 Delphi на примерах. — СПб.: БХВ-Петербург, 2005. — 496 с.: ил.

ISBN 5-94157-713-3

Изложены основы программирования в среде Delphi, начиная с составления программ в Turbo Pascal 7.0 и Object Pascal. Особое внимание уделено программам для решения задач из области высшей математики. Рассмотрены все этапы создания проекта в Delphi, начиная с разработки интерфейса и заканчивая особенностями работы с уже написанной программой. Приведены готовые проекты, которые должны помочь обучающемуся при выполнении самостоятельных заданий, помещенных в книге. Ко всем приведенным в книге заданиям имеются ответы и решения.

Для начинающих программистов, учащихся и студентов

УДК 681.3.06
ББК 32.973.26-018.2

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Игорь Шишигин*
Зав. редакцией	*Григорий Добин*
Редактор	*Екатерина Капалыгина*
Компьютерная верстка	*Ольги Сергиенко*
Корректор	*Зинаида Дмитриева*
Дизайн обложки	*Игоря Цырульникова*
Зав. производством	*Николай Тверских*

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 24.05.05.
Формат $70\times100^{1}/_{16}$. Печать офсетная. Усл. печ. л. 39,99.
Тираж 4000 экз. Заказ №
"БХВ-Петербург", 194354, Санкт-Петербург, ул. Есенина, 5Б.

Санитарно-эпидемиологическое заключение на продукцию
№ 77.99.02.953.Д.006421.11.04 от 11.11.2004 г. выдано Федеральной службой
по надзору в сфере защиты прав потребителей и благополучия человека.

Отпечатано с готовых диапозитивов
в ГУП "Типография "Наука"
199034, Санкт-Петербург, 9 линия, 12

ISBN 5-94157-713-3

Оглавление

Предисловие

Настоящая книга посвящена основам программирования в среде Delphi, разработанной компанией Borland. Данная среда является одной из ведущих систем программирования, используемых для разработки современных программных продуктов, и в первую очередь приложений операционной системы Windows. Система программирования Delphi позволяет значительно упростить процесс создания Windows-приложенией (до ее появления основным средством для разработки Windows-приложений был весьма сложный для изучения язык C++), а также резко повысить производительность труда программиста. Поэтому не случайно, что систему Delphi относят к категории RAD-систем программирования. *RAD* — Rapid Application Development, быстрая разработка приложений.

Система Delphi базируется на использовании языка программирования Object Pascal, который является логическим продолжением и развитием классического языка программирования Паскаль. В связи с этим, говоря о системе Delphi, нельзя не сказать несколько слов о языке Паскаль, его особенностях, об истории его возникновения и развития.

Созданный в начале 70-х годов XX века признанным классиком программирования Никлаусом Виртом, язык Паскаль был назван в честь французского ученого Блеза Паскаля (1623—1662). Великий ученый, которого современники называли французским Архимедом, вошел в историю не только как автор научных трудов, охватывающих самые различные области человеческого знания (от философии до математики), но и как изобретатель арифметической машины — первого в мире механического счетного устройства.

Изобретение Блеза Паскаля положило начало тому процессу, который, в конечном счете, привел к появлению современной вычислительной техники, ставшей одним из определяющих факторов научно-технического прогресса. Без компьютеров ныне вообще немыслимо нормальное существование и развитие цивилизованного общества. Исходя из сказанного ранее ясно, что на-

звание языка было выбрано не случайно. Паскаль был задуман как образцовый язык, который должен определенным образом формировать мышление программистов, помочь им почувствовать законы программирования, его красоту.

Первоначально Паскаль разрабатывался, прежде всего, как язык, предназначенный для эффективного обучения программированию, и успешно справлялся с этой задачей. Например, в США в 1983 году Паскаль был объявлен официальным языком программирования для учащихся средних школ, которые намерены специализироваться в области вычислительной техники и программирования в американских университетах. Но с течением времени язык Паскаль вышел за чисто учебные рамки и стал равноправным и популярным языком программирования.

В 80-е годы прошлого века позиции Паскаля еще более упрочились в связи с появлением версий языка, предназначенных для персональных компьютеров. Язык стал использоваться не только как средство обучения студентов и школьников, но и широко стал применяться как рабочий инструмент пользователей. Возникло целое семейство языков Паскаль, и ведущее место в этом семействе занял язык Turbo Pascal, разработанный программистами американской фирмы Borland International. На протяжении ряда лет (1983—1992) фирмой Borland был создан ряд новых, более совершенных версий языка, и итогом этой работы стало создание мощной системы программирования, включающей универсальную интегрированную среду, в которую "погружен" язык. Эта среда значительно упрощала и облегчала процесс создания программ, и в то же время предоставляла пользователю ряд новых, дополнительных возможностей.

Хотя система программирования Turbo Pascal создавалась как приложение операционной системы MS-DOS, она успешно работает и на компьютерах, на которых установлена операционная система Windows, и продолжает широко использоваться как в учебных целях, так и для решения практических задач. Однако с течением времени возникла настоятельная потребность в разработке системы программирования, которая бы позволяла сочетать сохранение лучших традиций программирования на Паскале с использованием всех возможностей, предоставляемых наиболее популярной сегодня для персональных компьютеров операционной системой Windows.

Поэтому в 90-е годы XX века фирма Borland завершила линию, связанную с программированием для операционной системы MS-DOS и представленную Turbo Pascal, и приступила к работе над системой программирования, ориентированной на создание приложений операционной системы Windows. Новая система программирования получила название Delphi. Она широко распространилась во всем мире и успешно развивается, а ко времени выхода этой

книги фирмой Borland была создана уже девятая по счету версия данного программного продукта.

Свое название система программирования Delphi получила по имени существовавшего в древней Греции города Дельфы, где находился знаменитый храм бога Аполлона. Аполлон почитался в древнем мире как небесный покровитель наук и искусств, а в храме, построенном в его честь, находился знаменитый дельфийский оракул, у которого люди стремились не только узнать о своей личной судьбе, но и получить ответы на самые сокровенные вопросы бытия. В настоящее время система Delphi представляет собой мощное орудие, которое современные ученые и инженеры используют для познания и преобразования окружающего мира.

Систему программирования Delphi, подобно системе Turbo Pascal, часто называют *интегрированной* средой программирования. Слово "интегрированный" (от латинского integrare — восстанавливать, восполнять) означает в данном случае, что в системе объединены в одно целое различные средства, способствующие наиболее быстрой и эффективной разработке программы.

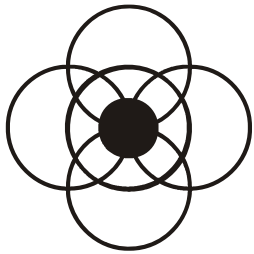
Характеризуя среду программирования Delphi, о ней также говорят как о *визуальной* и *событийно-ориентированной*. Первое означает, что пользователь наглядно (визуально) может увидеть в системе те заготовки, которые в дальнейшем будут использованы для создания экранных объектов в его программе, а затем сам сконструировать ее интерфейс (внешний вид) путем переноса этих заготовок на экранную форму. Второе же означает, что программист может выбрать из имеющегося в системе программирования списка те события, на которые должны реагировать экранные объекты, и запрограммировать эту реакцию нужным ему образом.

Наконец, еще одним важным достоинством системы программирования Delphi является ее универсальность. Дело в том, что многие современные языки и соответствующие системы программирования созданы для решения узкоспециальных задач. Так язык Cobol предназначен в первую очередь для создания программ в области экономики, язык Fortran — для инженерно-технических расчетов, языки Lisp и Prolog — для работы над системами искусственного интеллекта и т. д. Система же Delphi позволяет создавать профессиональные и эффективно работающие приложения, используемые в самых различных сферах человеческой деятельности. Поэтому время, затраченное будущим специалистом на изучение данной системы программирования, будет потрачено с пользой, вне зависимости от того, какую специализацию он изберет для себя в дальнейшем. Одну из основных целей при написании данной книги авторы видели в том, чтобы в доступной форме научить специалиста основам процесса программирования в системе Borland Delphi, а также сообщить ему необходимые базовые знания в этой области, которые в дальнейшем он сможет пополнять путем самообразования.

Все представленные в пособии программы на языке Паскаль реально работают. Любой желающий, мало-мальски разбирающийся в программировании, аккуратно набрав предложенные примеры программ, после их запуска, сможет получить требуемый результат и не разочаруется. Это же относится и к рассмотренным в книге проектам в системе Delphi, которые работают в любой версии данного программного продукта, начиная с пятой и заканчивая последней. На момент завершения работы над книгой последней версией была девятая.

В. М. Пестриков, СПбГАСЭ

А. Н. Маслобоев, СПбГТУ РП



ГЛАВА 1

От алгоритма к программе

Решение задач на персональном компьютере представляет собой диалог между человеком и электронной машиной на языке программирования. Этот диалог будет успешным только тогда, если правильно понята поставленная задача, разработан план ее решения, и она представлена в виде отлаженной программы на выбранном языке программирования.

1.1. Что такое алгоритм?

Ежедневно каждому человеку, как в своей профессиональной деятельности, так и в жизни, приходится решать ряд возникающих перед ним задач. Опыт показывает, что решение проблемы будет происходить быстрее, успешнее и эффективнее, если у человека есть готовый план или рецепт решения этой задачи. Такой план может быть составлен человеком самостоятельно на основе собственных знаний и жизненного опыта, или при условии, что у него есть возможность воспользоваться рецептом решения проблемы, составленным кем-то ранее, и проверенной на практике. В любом случае использование такого плана при четком и неукоснительном соблюдении его положений должно привести, в конечном счете, к решению задачи. Такая последовательность четких однозначных указаний, разработанных для решения поставленной задачи, называется *алгоритмом*.

Одним из наиболее простых примеров алгоритма являются правила перехода улицы пешеходом. Их можно свести к нескольким пунктам.

1. Перед началом перехода улицы посмотреть налево и убедиться, что в непосредственной близости не находится движущийся транспорт.
2. В случае если вблизи пешехода находится движущийся транспорт, подождать, пока он проедет. Если транспорта нет, начать переход улицы и дойти до середины проезжей части.

3. Дойдя до середины проезжей части, посмотреть направо для проверки наличия или отсутствия в непосредственной близости движущегося транспорта.
4. В случае если вблизи находится движущийся транспорт, подождать, пока он проедет. При отсутствии транспорта можно завершить переход улицы.

Естественно, что когда человек, проживший достаточно длительное время в большом городе, переходит улицу, он необязательно должен вспоминать все указанные ранее правила, т. к. они уже будут запечатлены у него в мозгу на подсознательном уровне. Но при решении какой-либо более сложной задачи даже на бытовом уровне, особенно на первых порах, без тщательного соблюдения всех инструкций, изложенных "на бумажке", часто не обойтись. Примером такого более сложного алгоритма является инструкция по пользованию банкоматом, с помощью которого в настоящее время получают зарплату сотрудники многих организаций и предприятий. Такая инструкция состоит из следующих пунктов:

1. Вставить кредитную карточку в банкомат определенным образом (как правило, логотипом банка, изображенным на карточке, от себя).
2. Набрать индивидуальный цифровой код владельца карточки, после чего нажать клавишу **Ввод**.
3. Выбрать язык, на котором пользователь дальше будет общаться с банкоматом (как правило, в нашей стране банкоматы запрограммированы на работу с двумя языками: русским и английским), подтвердив выбор нажатием соответствующей клавиши.
4. Выбрать операцию, которую пользователь будет производить с банкоматом, из списка, приведенного на экране банкомата (это может быть получение денег, получение сведений о нескольких последних операциях, произведенных со счетом пользователя, получение информации справочного характера о работе сети банкоматов и т. д.), нажав клавишу, соответствующую выбранной операции.
5. В случае если пользователю необходимо получить деньги со счета, ответить на вопрос, хочет ли он получить также и чек, в котором будет указан размер полученной суммы и остаток денег на счете.
6. Ввести сумму, которую пользователь хочет получить со счета и подтвердить ее правильность нажатием клавиши **Ввод**. В случае если снимаемая пользователем со счета сумма превышает ту сумму, которая находится в данный момент на счете пользователя, ввести другую, меньшую по величине сумму.
7. Подождать несколько секунд, пока банкомат отсчитывает требуемую сумму, а затем, после появления разрешающей надписи на экране банко-

мата, вынуть из банкомата кредитную карточку, а затем забрать деньги и чек.

Такой же набор инструкций может быть с успехом использован для решения задач из различных областей науки и техники. В качестве простого примера приведем алгоритм нахождения среднего арифметического ряда, состоящего из нескольких чисел. Такой алгоритм состоит всего из двух пунктов.

1. Сложить все числа ряда.
2. Разделить полученную сумму на число членов ряда. Полученное частное и будет средним арифметическим.

Все три приведенные примеры алгоритмов относятся к разным сферам человеческой деятельности, однако можно заметить, что у них имеются некоторые общие свойства, которыми должен обладать любой работоспособный алгоритм.

- *Понятность* — тот, кто выполняет алгоритм, должен понимать, как на основе алгоритма и имеющихся исходных данных можно получить искомый результат.
- *Дискретность* — процесс решения задачи можно разбить на несколько этапов (пунктов), каждый из которых представляет собой некоторое законченное действие.
- *Определенность* — каждый пункт алгоритма должен быть сформулирован четко и однозначно и не оставлять места для произвольного толкования.
- *Результативность* — с помощью алгоритма задача должна быть решена за конечное число шагов.
- *Массовость* — алгоритм должен быть пригоден для решения целого класса задач, если такие задачи отличаются друг от друга только различными исходными данными.
- *Правильность* — решение задачи, полученное посредством использования алгоритма, должно соответствовать действительности.

Важной особенностью правильно составленного алгоритма является то, что он может выполняться не только человеком, но и любым устройством, которое в состоянии механически, не размышляя, выполнять указанные в алгоритме действия. Часто человека или подобное устройство называют *исполнителем алгоритма*. В качестве исполнителя, например, может выступать персональный компьютер. При этом, однако, нужно иметь в виду, что алгоритм, по которому работает компьютер, должен быть задан на понятном ему (компьютеру) языке, что требует иной записи алгоритма, нежели та, которая была использована в приведенных примерах. В данных примерах применялась словесная форма записи алгоритма, помимо которой используются еще две формы записи: графическая и в виде программы, написанной на каком-либо из языков программирования.

Графическая форма записи удобна для более компактного и наглядного представления различных элементов, из которых складывается алгоритм. При графическом представлении алгоритм изображается в виде последовательности связанных между собой блоков, каждый из которых соответствует выполнению одного или нескольких действий и изображается в виде определенной геометрической фигуры. Такой способ представления алгоритма называется *блок-схемой* или *графической схемой*, что более правильно, т. к. соответствует названию, употребляемому в государственном стандарте. Наиболее часто употребляемые блоки, входящие в состав графических схем, приведены на рис. 1.1.

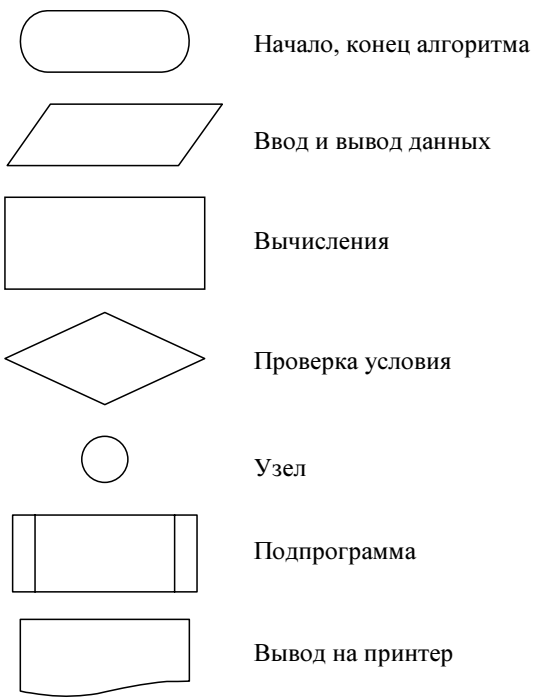


Рис. 1.1. Изображения основных блоков, используемых при составлении графических схем

Поясним назначение этих блоков. Блоки "начало" и "конец" используются для обозначения начала и конца алгоритма. Внутри обозначающей их фигуры пишется слово "начало" или "конец". Ввод и вывод соответственно используются для ввода исходных величин, необходимых для решения задачи, или вывода полученных результатов. Соответствующие величины указываются внутри параллелограмма, обозначающего ввод или вывод. Обработка данных (как правило, это какие-либо вычисления) изображается в виде прямоугольника. Внутри прямоугольника записывается содержание этих вычислений.

Проверка определенного условия изображается в виде ромба. Само условие записывается внутри ромба. В результате проверки условия осуществляется выбор одного из двух возможных путей дальнейшего выполнения алгоритма.

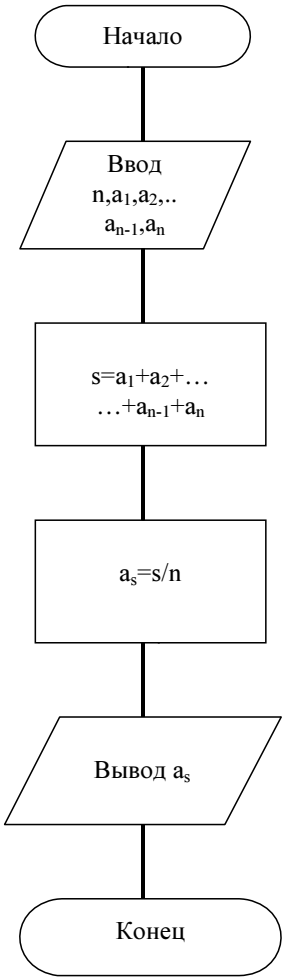


Рис. 1.2. Графическая схема нахождения среднего арифметического

Запишем в виде графической схемы процесс вычисления среднего арифметического (рис. 1.2). После блока, означающего начало алгоритма, находится блок ввода исходных данных, к которым в этом случае относятся n — число членов ряда и значения членов этого ряда a_1, a_2 и т. д. вплоть до последнего члена ряда a_n. Эти блоки, как и все входящие в состав графической схемы, объединяются друг с другом прямыми линиями. Следующий прямоугольный

блок представляет собой процесс вычисления — нахождение величины s, являющейся суммой членов ряда. Следующий блок также относится к разряду блоков обработки информации. На этот раз в нем производится операция деления, которая позволяет найти величину a_s — искомое среднее арифметическое членов ряда. В предпоследнем блоке осуществляется вывод полученного результата. Последний блок представляет собой конец алгоритма, о чем оповещает соответствующая надпись внутри него.

Приведенный способ записи является, как уже говорилось, более строгим и лаконичным, чем словесное описание алгоритма, и нередко используется на этапе подготовки к решению задачи на компьютере, но непосредственно в компьютер для получения искомого результата графическую схему вводить нельзя, так же как и словесный рецепт. Единственный способ описания алгоритма, который непосредственно понятен центральному процессору компьютера (центральный процессор — это компьютерное устройство, координирующее работу всех прочих устройств компьютера и выполняющее элементарные арифметические и логические операции) и с которым он может в дальнейшем работать, является *компьютерная программа*, записанная на одном из языков программирования. Об этих понятиях и пойдет речь в следующем разделе книги.

1.2. Программа и языки программирования

Итак, если попытаться дать в самом общем виде определение компьютерной программе, то можно сказать, что *программа* — это алгоритм, понятный компьютеру. Только язык, на котором написаны программы, понятен компьютеру, а при отсутствии программ компьютер из полезного, а во многих случаях незаменимого для современного человека устройства, превращается в абсолютно бесполезный набор микросхем и проводов. Процесс обработки информации, осуществляемый компьютером, требует наличия как *технических средств* (называемых часто англоязычным словосочетанием *hardware* или русскоязычным жаргонным словом "железо"), так и *программного обеспечения* (называемого *software* или "софт"). *Программированием* называется процесс создания компьютерной программы.

Единственным языком, который непосредственно понимает процессор компьютера, является *язык машинных кодов*, который представляет собой последовательность двоичных чисел, изображаемых нулями и единицами. Эта особенность языка машинных кодов объясняется причинами технического характера. Оказалось, что технически гораздо легче построить компьютер на базе элементов, каждый из которых может находиться в одном из двух устойчивых состояний, одно из которых соответствует нулю, а другое единице. Когда в 40-е годы XX века были созданы первые компьютеры, то машинный

код был единственным языком, на котором составлялись компьютерные программы. Естественно, что составление программ на таком языке является для человека крайне скучным и утомительным процессом, чреватым к тому же многочисленными ошибками. Еще одним существенным недостатком машинных кодов является то, что они привязаны к компьютерам определенной модели, конструкции. Созданные для компьютера одной конструкции программы в машинных кодах нельзя переносить без соответствующей переработки на компьютеры другой конструкции.

Уже через несколько лет после появления первых компьютеров для облегчения и упрощения работы программистов был создан *язык ассемблера*, в котором команды машинных кодов заменялись более понятными для человека. Такими командами в основном были сокращенные слова английского языка. Эти более удобные для запоминания команды называют *мнемоническими* или *мнемониками языка ассемблера*. Язык ассемблера до сих пор употребляется для составления программ в тех случаях, когда предъявляются повышенные требования к компактности программ. Программы, написанные на языке ассемблера, требуют минимального объема памяти и времени выполнения. Однако этот язык сохранил ряд существенных недостатков своего предшественника — языка машинных кодов. Это, во-первых, достаточно жесткая привязка языка к компьютерам определенной конструкции. Во-вторых, как и в случае с машинными кодами, язык ассемблера требует очень высокой квалификации программиста, знания не только нюансов программирования, но и знания тонкостей устройства компьютера, особенно в том, что касается архитектуры его центрального процессора. Поэтому язык ассемблера называют машинно-ориентированным языком. По отношению к машинным кодам и языку ассемблера употребляют термин *"языки низкого уровня"*, т. к. они максимально приближены к уровню технических устройств компьютера.

Следующим шагом в развитии программирования стало создание *алгоритмических языков высокого уровня*. Такие языки еще удобнее для понимания, чем команды ассемблера, т. к. они включают в себя обычные слова человеческого языка и общепринятые обозначения математических операций. Команды языков высокого уровня (называемые также операторами) зачастую напоминают фразы, составленные на английском языке. Надо сказать, что именно английский язык, являющийся в настоящее время языком международного общения, своего рода латынью XX—XXI веков, стал основой, на которой было создано подавляющее большинство языков программирования высокого уровня. Кроме того, языки высокого уровня стали машинно-независимыми, т. е. программы, написанные на таких языках, можно без труда переносить с одного компьютера на другой. Первым широко распространенным во всем мире языком программирования высокого уровня стал язык Фортран, созданный в первую очередь для решения различных научно-технических

задач. Название этого языка представляет сокращение английского словосочетания formula translation, что означает "перевод формул", т. е. имеется в виду перевод различных математических формул с обычного человеческого языка на язык, понятный компьютеру. За ним последовали такие языки высокого уровня, как Кобол, Алгол, Паскаль, Бейсик, Си и многие, многие другие.

Говоря о достоинствах языков программирования высокого уровня, надо иметь в виду, что процессор компьютера не может воспринимать эти языки непосредственно. Для того чтобы компьютер мог выполнять программы, написанные на этих языках, ему требуется перевод их в машинные коды. Такой перевод осуществляется с помощью специальных программ, называемых *трансляторами*. Трансляторы делятся на две основные категории. В одних случаях транслятор непосредственно переводит строку за строкой команды языка высокого уровня в машинные коды, которые немедленно выполняются. Такой способ трансляции называется *интерпретацией* и используется, например, для перевода в машинные коды программ, составленных на языке Бейсик. Для большинства других языков высокого уровня, включая язык Паскаль, используется другой способ трансляции, который заключается в том, что программа обрабатывается целиком, а затем на ее основе создается модуль в машинных кодах. Такой способ обработки программ называется *компиляцией*.

Каждый из этих способов трансляции имеет свои достоинства и недостатки. Составление программ на интерпретируемых языках часто бывает легче для программиста, т. к. многие ошибки в программе выявляются уже на стадии ввода ее текста в компьютер. После ввода очередной строки текста программы интерпретатор сразу выдает сообщение об ошибках (естественно, при наличии таковых). Однако за удобство программирования на интерпретируемом языке приходится расплачиваться быстродействием составленных на этом языке программ. Такая программа работает в несколько десятков раз медленнее, чем программа, переведенная в машинные коды посредством компилятора. Последнее обстоятельство, как правило, оказывается более важным, поэтому в большинстве случаев для создания программ, особенно на профессиональном уровне, используются компилирующие языки.

1.3. Этапы работы над программой. Система программирования

Из сказанного в предыдущем разделе понятно, что составление программ даже с помощью современных средств и языков программирования не является неким одномоментным процессом, а включает в себя ряд последовательных этапов.

Перечислим основные этапы:

1. **Постановка задачи.** Для того чтобы правильно составить программу, программисту нужно предварительно решить для самого себя, чего он хочет добиться, составляя программу, достаточно четко и конкретно сформулировать задачу, решаемую с помощью программы, определить, какие исходные данные необходимы для ее решения и какой требуется получить результат.
2. **Нахождение оптимального метода решения задачи и создание соответствующего алгоритма.** Для того чтобы поставленную задачу можно было успешно решить, программист должен вначале хотя бы в общих чертах составить план решения задачи, наметить наиболее надежный и эффективный путь ее решения. Часто бывает полезно перед тем, как начать составлять программу на компьютере, изобразить на бумаге ее алгоритм в виде графической схемы либо изложить в словесном виде основные пункты ее решения.
3. **Запись текста программы на языке программирования.** Этот процесс часто называют также кодированием программы, т. е. происходит перевод алгоритма в код, понятный компьютеру либо непосредственно (в случае языков низкого уровня), либо через переводчика — программу-транслятор (в случае языков высокого уровня). Для того чтобы можно было набирать текст программы, необходимо наличие на компьютере специальной программы — текстового редактора. *Текстовым редактором* называется программа, с помощью которой пользователь может осуществлять операции ввода и редактирования текста. Под редактированием текста понимается его правка, исправление обнаруженных в тексте ошибок. Наличие возможностей правки является обязательным элементом текстового редактора, т. к., во-первых, набор текста практически любой программы, особенно у неопытных пользователей на начальном этапе обучения программированию, не обходится без опечаток. Во-вторых, даже опытный программист часто в ходе составления программы осуществляет оптимизацию программы, т. е. делает уже работающую программу более качественной и эффективно работающей, что требует удаления определенных фрагментов программы и замены их новыми.

   Текст программы, уже набранный и откорректированный, находится до определенного момента в оперативной памяти компьютера. Если программа не является "одноразовой", необходимой для получения только один раз некоторых результатов (а такими, как правило, бывают только программы, написанные для тренировки, в учебных целях), то ее следует сохранить на жестком диске компьютера в виде отдельного файла. *Файлом* называется область на диске компьютера, имеющая свое собственное имя и служащая для хранения программ и данных. В противном случае

(если не сохранить программу в файле) сразу после выключения компьютера содержимое его оперативной памяти очистится, а вместе с ним пропадет и набранный пользователем текст программы.

4. **Проверка программы.** Данный этап начинается с того, что программа запускается на трансляцию. При этом программа-транслятор проверяет исходный текст программы (называемый также *исходным кодом*) на правильность с точки зрения орфографии и синтаксиса того языка программирования, на котором составлен текст данной программы. При обнаружении подобных ошибок транслятор выдает об этом соответствующее сообщение. В этом случае программа заведомо не будет работать, поэтому в случае выдачи сообщений об ошибках пользователь должен внести в программу соответствующие коррективы, а затем снова запустить ее на трансляцию. В том случае, если в тексте программы транслятором не было обнаружено явных ошибок, на основе исходного кода программы транслятор создает перевод текста программы в машинные коды, называемый *объектным модулем*.

   Однако объектный модуль еще не является готовой к выполнению на компьютере программой. Дело в том, что практически ни одна современная компьютерная программа не обходится без использования некоторых вспомогательных программ, разработанных ранее другими программистами. Такие вспомогательные программы называются *стандартными подпрограммами* и объединяются в специальные *библиотеки*. Подключение же этих библиотек производится с помощью специальной программы, называемой *редактором связей*. После подключения необходимых стандартных подпрограмм и создается готовая к выполнению программа, называемая *исполняемым модулем*.

5. **Отладка программы.** Нередко встречается ситуация, когда пользователь составил программу, в которой отсутствуют грамматические ошибки, но тем не менее программа не выдает желаемого результата, либо выдает результат в неправильном виде. Такая ситуация говорит о том, что в программе имеются семантические (смысловые) ошибки. Для исправления таких ошибок требуется не просто формальное знание правил языка программирования, как на предыдущем этапе, а наличие четкой логики мышления, знания основных принципов программирования и наиболее эффективных методов выявления ошибок, определенный опыт в составлении компьютерных программ. Однако и эти ошибки при наличии у пользователя терпения и желания довести работу до логического конца являются в конечном счете устранимыми, и тогда итогом работы над программой становится работоспособный окончательный вариант программы.

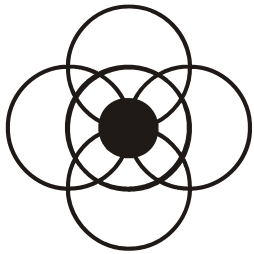
Если мы охватим мысленно все описанные этапы работы над программой, то станет ясно, что на первых двух этапах основным инструментом для работы

над программой является мозг программиста, позволяющий должным образом поставить задачу и найти способ ее решения. Однако на последующих этапах разработки программы имеется много рутинной работы, которую в значительной своей части способен выполнять не человек, а компьютер. Поэтому на основе опыта программирования на языках высокого уровня возникла идея дополнить транслятор рядом вспомогательных программ, облегчающих и упрощающих в значительной степени процесс создания и отладки программ, интегрировав эти программные средства в единый программный комплекс. Такой комплекс называется *интегрированной системой* или *средой программирования*. Данный комплекс должен включать в себя несколько основных элементов.

- *Встроенный текстовый редактор* — использование такого редактора гораздо удобнее, нежели подключение какого-либо внешнего. Ему можно, в частности, поручить такую задачу, как отмена одного или нескольких последних действий пользователя в том случае, если они были ошибочными.
- *Компилятор* — это основное ядро системы программирования. Компилятор может создавать как исполняемые модули в рамках самой системы программирования, так и модули, которые, будучи созданы в данной системе, могут работать независимо от нее.
- *Редактор связей* подключает требуемые стандартные библиотеки.
- *Программа-отладчик* выдает сообщения об обнаруженных ошибках, причем сообщает не только о самом факте ошибки, но и указывает характер сделанной ошибки, а в большинстве случаев также называет номер строки, в которой ошибка была обнаружена, или автоматически устанавливает на нее курсор.
- *Справочная система* содержит различную информацию как об основных структурах языка и правилах их использования с примерами программ, так и необходимую информацию о самой системе программирования.

Как уже было сказано в предисловии, подобная система программирования была разработана для языка Паскаль и получила название Turbo Pascal. Приставка Turbo означает "ускорение" и объясняется тем, что компиляция программы в данной системе происходила быстрее, чем в других аналогичных системах. Данная система позволяет успешно создавать достаточно сложные и в то же время эффективно работающие программы как в текстовом режиме работы компьютера, так и в графическом.

ГЛАВА 2



Подготовка к программированию на Паскале

Когда выбран язык для программирования поставленной задачи, возникает резонный вопрос, где взять этот программный продукт? Можно, например, систему программирования Turbo Pascal 7.0 купить на оптическом диске в компьютерном магазине, а можно, подключившись к Интернету, бесплатно ее "скачать" с соответствующего сайта. После того как программный продукт загружен и установлен, можно приступать к решению задач, предварительно изучив основы составления программ на этом языке.

2.1. Где взять Turbo Pascal

У человека, естественно, который ранее не занимался программированием, но хочет приступить к изучению языка программирования, возникает законный вопрос: "Откуда взять этот самый язык программирования?"

Для того чтобы ответить на данный вопрос, необходимо, прежде всего, сказать, что язык программирования неотъемлем от системы программирования. *Система программирования* представляет собой набор взаимосвязанных файлов, который может храниться либо на каком-то из дисковых носителей информации (компакт-диск или дискета), либо находиться на каком-то сайте в Интернете. Задача пользователя заключается в том, чтобы перенести этот набор файлов на жесткий диск (винчестер своего компьютера). Рассмотрим возможные варианты действий применительно к системе программирования Turbo Pascal 7.0.

Система программирования Turbo Pascal 7.0 невелика по объему (она занимает на диске около 2,5 Мбайт), поэтому, как правило, она записывается на имеющиеся в продаже компакт-диски не сама по себе, а вместе с другими программными комплексами. Часто на таких дисках можно обнаружить так называемые Rip-версии программных продуктов (в том числе и Turbo Pascal).

Rip-версия представляет собой сделанную пиратским способом копию того или иного программного продукта. При этом компьютерные пираты нещадно удаляют ненужные на их взгляд файлы, содержащиеся в программном комплексе (приставка Rip представляет собой не что иное, как сокращение английской фразы rest in peace, что в переводе означает "покойся с миром"). Делается это с очевидной целью — на одном и том же компакт-диске разместить как можно больше различных программ. Однако такая "забота" о пользователе может быть чревата тем, что программный комплекс будет неработоспособен. Возможен и другой вариант: в принципе программа будет запускаться и работать, но из-за отсутствия некоторых файлов пользователь сможет воспользоваться далеко не всеми ее возможностями.

Поэтому авторы книги рекомендуют начинающим пользователям переписать Turbo Pascal с одного из сайтов, где имеется нормальная, работоспособная версия данного продукта. Таким сайтом является, например, **Turbo Pascal**, расположенный по адресу **http://borlpasc.narod.ru**. Этот сайт, помимо самого Turbo Pascal, содержит еще много другой интересной и полезной для начинающего программиста информации, но подробнее об этом мы расскажем в другой главе данной книги, посвященной языку Turbo Pascal во Всемирной паутине. Сейчас же нас интересует в первую очередь процесс загрузки Turbo Pascal, о чем мы и расскажем далее.

Процедура загрузки Turbo Pascal рассматривается исходя из того, что у пользователя на компьютере установлена операционная система Windows и браузер Internet Explorer. Итак, набрав в адресной строке вашего браузера адрес **http://borlpasc.narod.ru** и нажав клавишу <Enter>, вы оказываетесь на главной странице сайта, которая показана на рис. 2.1.

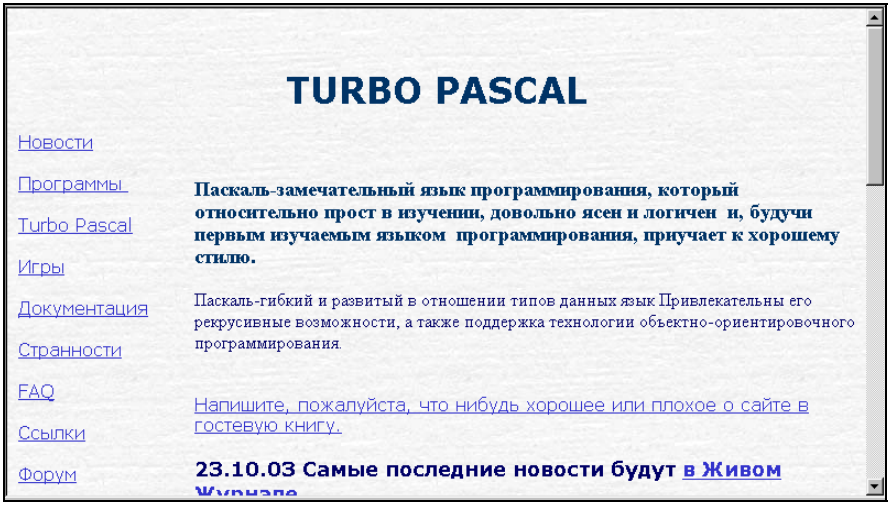


Рис. 2.1. Главная страница сайта **borlpasc.narod.ru**

Далее на главной странице нужно щелкнуть мышью гиперссылку, которая так и называется **Turbo Pascal**. В результате вы оказываетесь в разделе сайта, где содержатся различные версии языка Паскаль (рис. 2.2).

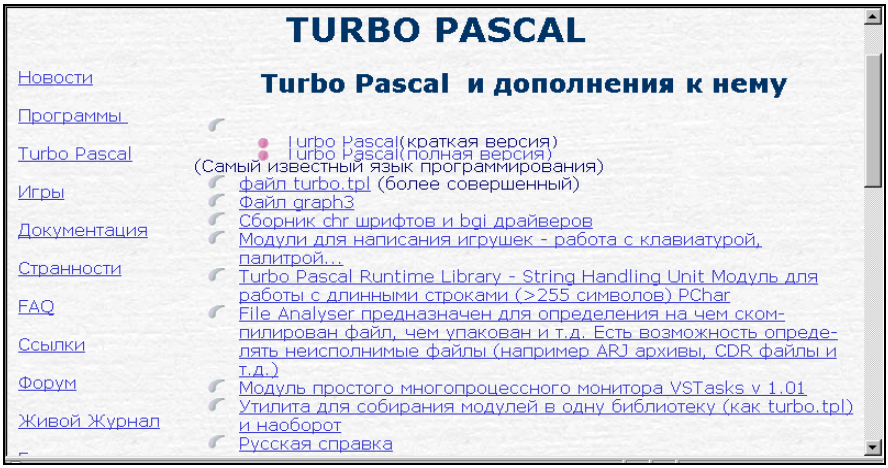


Рис. 2.2. Страница сайта **borlpasc.narod.ru**, содержащая различные версии языка Паскаль

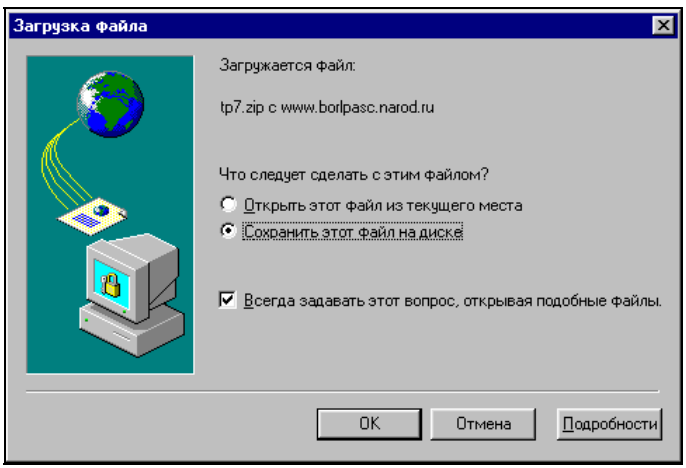


Рис. 2.3. Окно загрузки файла tp7.zip

Для наших целей лучше всего подойдет полная версия языка Turbo Pascal, поэтому щелкаем соответствующую гиперссылку. После щелчка на гиперссылке открывается диалоговое окно загрузки файла, содержащего систему Turbo Pascal (рис. 2.3). Этот файл называется tp7.zip и представляет собой архив, в котором в запакованном виде хранятся файлы системы программи-

рования. В окне загрузки файла tp7.zip пользователь должен указать, нужно ли открывать файл, находясь в Интернете, или сохранить его на диске. В данном случае выбираем второй вариант, установленный в окне по умолчанию, что подтверждаем щелчком на экранной кнопке **OK**.

После щелчка появляется дополнительное диалоговое окно, в котором пользователь может указать, в какой именно папке нужно сохранить загружаемый файл (рис. 2.4). Эту папку пользователь может выбрать самостоятельно, перемещаясь по файловой структуре компьютера. Для навигации по файловой структуре можно использовать экранную кнопку для перехода в расположенную выше папку или кнопку **Открыть** для открытия вложенной папки. Пользователь может создать и специальную папку, в которую будет записан файл tp7.zip щелчком на экранной кнопке . Когда папка для сохранения файла выбрана или создана, подтверждаем наши действия щелчком на кнопке **Сохранить**.

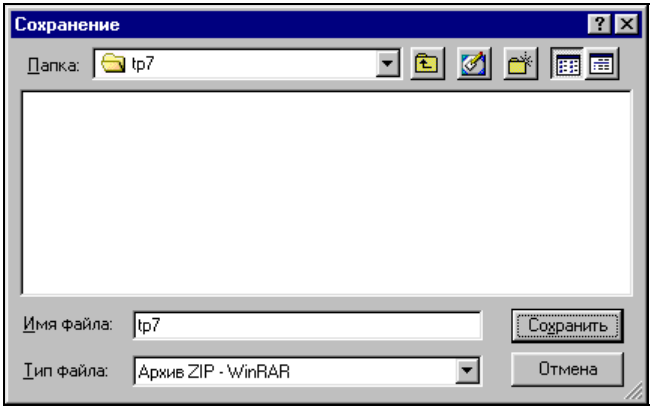


Рис. 2.4. Окно выбора папки, в которой будет сохранен файл tp7.zip

После щелчка на этой кнопке появляется информационное окно, которое показывает ход загрузки файла tp7.zip (рис. 2.5). В данном окне показывается количество записанной на жесткий диск компьютера информации в килобайтах и в процентах от общего объема загружаемого файла. Так как файл tp7.zip невелик по объему (1,09 Мбайт), то даже при плохом качестве связи загрузка данного файла занимает, как правило, не более 10—15 минут.

По окончании процесса загрузки файла на экране компьютера появляется окно, сообщающее о том, что процесс передачи данных завершен, а также показывающее, сколько времени было затрачено на загрузку файла, объем данного файла и среднюю скорость, с которой производилась загрузка (рис. 2.6). После ознакомления с данной информацией нужно щелкнуть в окне на кнопке **Закрыть**.

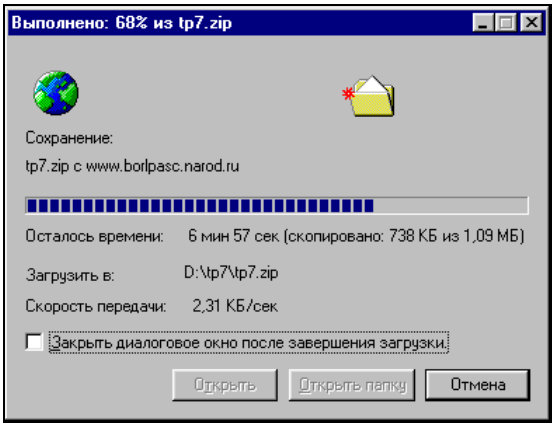


Рис. 2.5. Информационное окно, показывающее ход процесса загрузки файла tp7.zip

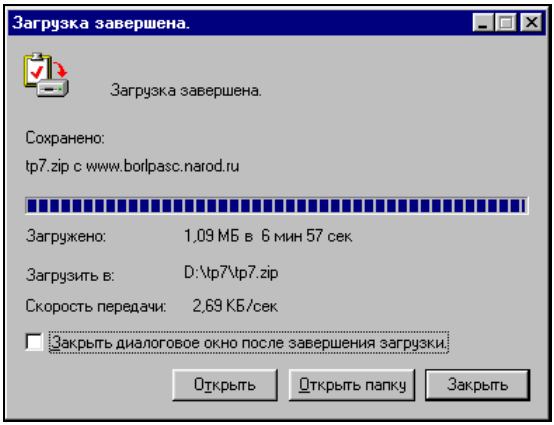


Рис. 2.6. Информационное окно, сообщающее о завершении процесса загрузки

2.2. Подготовка к запуску системы Turbo Pascal

Следующим этапом по созданию системы программирования Turbo Pascal на компьютере является распаковка архивного файла tp7.zip. Для этого нужно использовать соответствующую программу-архиватор. При этом необязательно, чтобы это была программа Winzip. Программа Winrar также понимает формат ZIP, в котором был записан архивный файл, и может быть с успехом применена для его распаковки. После разархивации файла на жестком диске компьютера появится папка (каталог) с именем TP7.

Этот каталог содержит ряд файлов системы программирования. Это файл turbo.exe — головной файл системы программирования, который запускает ее на выполнение, файл turbo.tpl, содержащий стандартные библиотечные модули, которые можно подключать по мере необходимости, файл turbo.tp, который содержит сведения о настройках системы программирования, файл turbo.tph, который содержит справку по языку Паскаль, а также другие файлы. Помимо файлов в каталоге могут содержаться и другие, вложенные каталоги (например, каталог BGI, который необходим для создания графических программ на языке Паскаль).

В принципе, этих файлов и каталогов вполне достаточно для того, чтобы приступить к программированию на языке Паскаль. Но желательно все же перед этим выполнить одну подготовительную операцию. Дело в том, что когда вы начнете создавать программы в системе Turbo Pascal, по умолчанию файлы программ будут записываться в основной каталог системы, в котором находятся указанные ранее системные файлы. При этом могут быть случайно повреждены или удалены системные файлы, что может привести к неработоспособности всей системы Turbo Pascal. Поэтому рекомендуется внутри каталога TP7 создать отдельный каталог, в который будут записываться файлы написанных вами программ. Назовем этот каталог prog.

Поскольку система Turbo Pascal является приложением операционной системы MS-DOS, то создание вложенного каталога можно производить как средствами этой операционной системы, так и средствами операционной системы Windows. Поэтому рассмотрим следующие варианты.

- Создание каталога в системе MS-DOS. Для этого необходимо перейти в каталог TP7 и написать в командной строке команду `md prog`, а затем для подтверждения команды нажать на клавиатуре компьютера клавишу <Enter>.
- Создание каталога в файловой оболочке Norton Commander (эта оболочка имеется на многих старых компьютерах, работающих под управлением системы MS-DOS). Необходимо вывести в активную панель Norton Commander содержимое каталога TP7 и нажать на клавиатуре компьютера функциональную клавишу <F7>. Затем в открывшемся диалоговом окне набрать имя создаваемого вложенного каталога (prog) и подтвердить его создание нажатием экранной кнопки **OK** (рис. 2.7).
- Создание вложенной папки в операционной системе Windows. Для этого нужно открыть окно папки TP7 и в строке меню щелчком мыши выбрать меню **Файл**, а в нем команду **Создать**. Справа от пункта **Создать** имеется треугольная стрелка, говорящая о том, что фактически эта команда представляет собой вложенное подменю, содержащее список объектов, которые можно создать с ее помощью. Если задержать указатель мыши (не

нажимая кнопку мыши) на пункте **Создать**, то через одну-две секунды вы увидите данный список. Для того чтобы выбрать нужный нам объект из списка, следует переместить указатель мыши на пункт списка **Папка** и щелкнуть его левой кнопкой мыши (рис. 2.8).

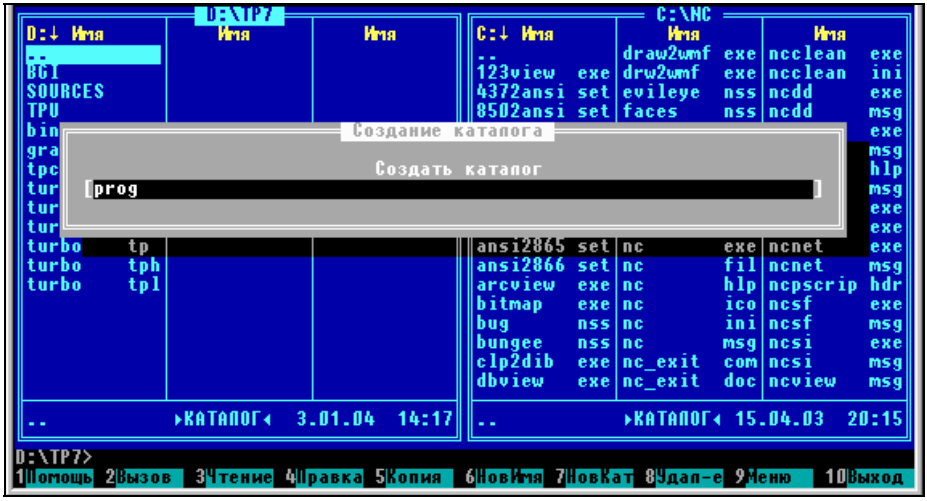


Рис. 2.7. Создание каталога для программ в файловой оболочке Norton Commander

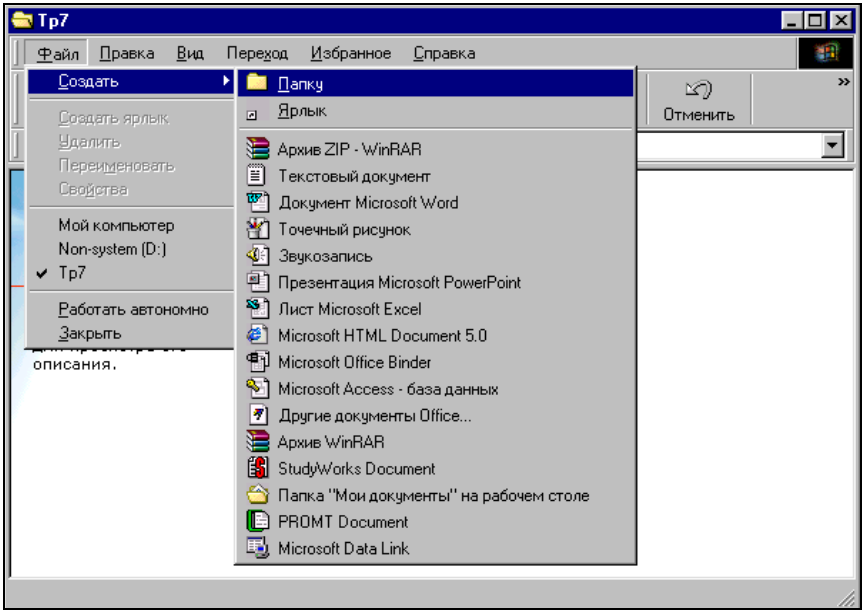


Рис. 2.8. Создание вложенной папки для программ в папке TP7
в операционной системе Windows

В результате указанных ранее действий в окне папки TP7 появится новый объект с именем **Новая папка**, причем поле с названием объекта будет выделено синим цветом. Для того чтобы переименовать эту новую папку, нужно сразу же после ее создания ввести с клавиатуры требуемое имя (в данном случае это имя prog). Ввод имени следует подтвердить нажатием клавиши <Enter>.

После создания папки prog любым из трех описанных ранее способов можно приступать к запуску системы программирования.

2.3. Запуск системы Turbo Pascal

Как уже было сказано в предыдущем разделе, система программирования Turbo Pascal представляет собой комплекс, содержащий ряд файлов, но в нем имеется один головной файл, запускающий систему на выполнение. Этот файл называется turbo.exe. Процедура запуска системы зависит от того, в какой операционной системе или программной оболочке работает пользователь, поэтому, как и ранее, рассмотрим следующие три варианта.

- Запуск Turbo Pascal из операционной системы MS-DOS. Для этого необходимо перейти в подкаталог, содержащий файл turbo.exe, набрать в командной строке команду `turbo` и затем нажать клавишу <Enter>.
- Запуск Turbo Pascal из программы-оболочки Norton Commander. Вывести в активную панель Norton Commander каталог, содержащий файл turbo.exe, установить на этот файл курсор и затем нажать клавишу <Enter> или дважды щелкнуть имя файла мышью (рис. 2.9).

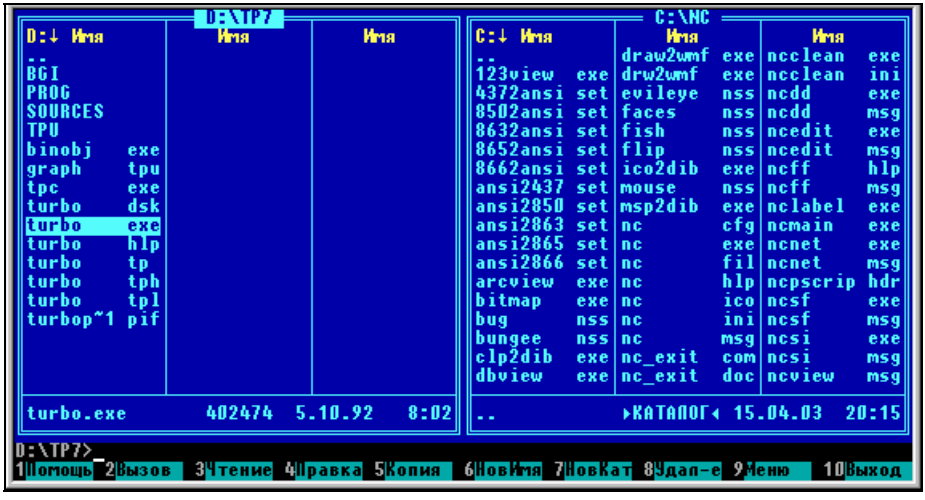


Рис. 2.9. Запуск системы программирования Turbo Pascal из файловой оболочки Norton Commander

- Запуск Turbo Pascal из операционной системы Windows. Как и большинство других операций в ОС Windows, запуск файла на выполнение можно осуществить несколькими способами:
  - открыть папку, содержащую файл turbo.exe, и запустить файл на выполнение двойным щелчком мыши;
  - если система программирования Turbo Pascal "прописана" в главном меню, открыть меню нажатием кнопки **Пуск**, найти в меню соответствующий пункт и один раз щелкнуть его мышью (рис. 2.10);
  - если на рабочем столе Windows для файла turbo.exe создан ярлык, для запуска файла достаточно дважды щелкнуть ярлык мышью.

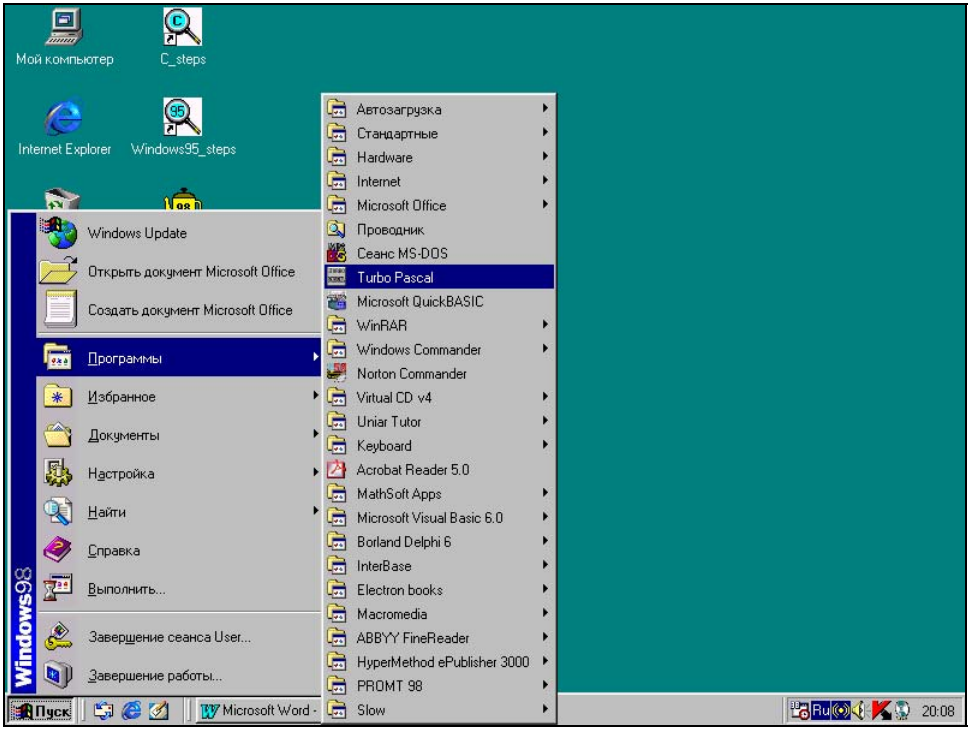


Рис. 2.10. Запуск системы программирования Turbo Pascal в операционной системе Windows через кнопку **Пуск**

Под управлением ОС Windows система программирования может работать либо в полноэкранном режиме, либо в оконном, занимая только часть экрана компьютера. Для того чтобы выбрать удобный для пользователя режим работы, следует отредактировать ярлык файла turbo.exe. Ярлык данного файла (как и других приложений ОС MS-DOS, работающих в ОС Windows) является не

просто ярлыком, а представляет собой программно-информационный файл, посредством редактирования которого можно настроить файл приложения (в данном случае это файл turbo.exe) оптимальным для пользователя образом.

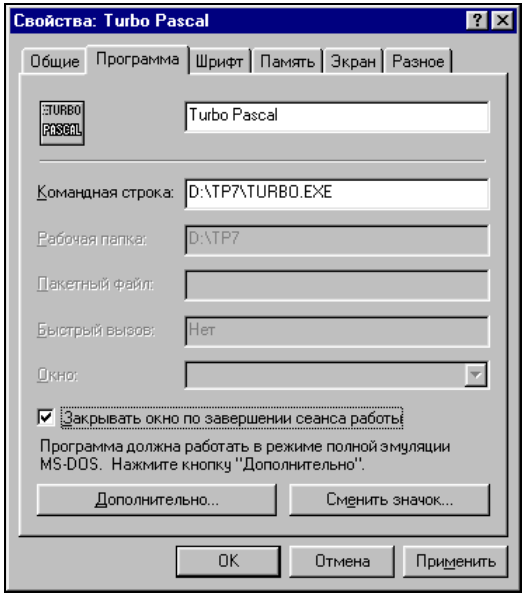


Рис. 2.11. Окно свойств программно-информационного файла для turbo.exe, вкладка **Программа**

Для редактирования ярлыка нужно щелкнуть его правой кнопкой мыши и в открывшемся контекстном меню выбрать пункт **Свойства**, щелкнув его мышью. В результате открывается диалоговое окно свойств ярлыка. Это окно содержит ряд вкладок: **Общие**, **Программа**, **Шрифт**, **Память**, **Экран**, **Разное**. Щелчком мыши выводим на передний план вкладку **Программа** (рис. 2.11). На вкладке установим флажок **Закрывать окно по завершении сеанса работы**. Тогда при выходе из системы программирования окно системы будет закрываться автоматически. Затем щелкнем на той же вкладке кнопку **Дополнительно**, после чего на экране компьютера появится еще одно окно для установки дополнительных настроек файла turbo.exe (рис. 2.12). В окне дополнительных настроек уберем флажок **Режим MS-DOS**, после чего можно будет работать в Turbo Pascal, не выходя из Windows. Подтвердим изменения щелчком на кнопке **OK**, после чего окно дополнительных настроек закроется. Теперь перейдем на вкладку **Экран** и установим переключатель в положение **Полноэкранный** или **Оконный** в зависимости от того, в каком режиме вам удобнее работать. Снова щелкаем мышью кнопку **OK** (теперь уже в основном окне свойств) для подтверждения сделанных изменений. По-

сле закрытия основного окна начальная настройка системы программирования произведена, и можно приступать к изучению самой системы.

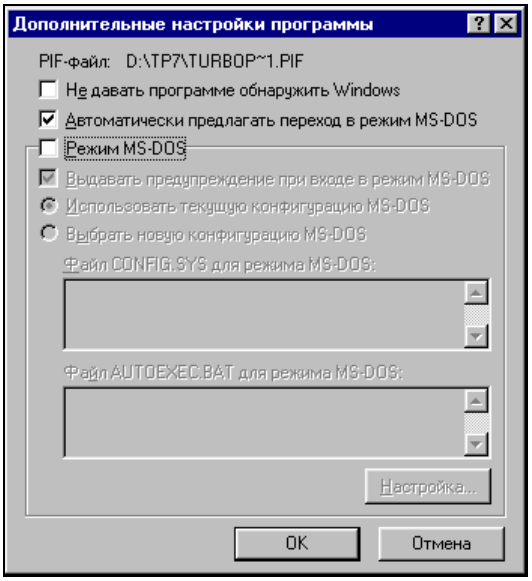


Рис. 2.12. Окно дополнительных настроек для файла turbo.exe

2.4. Экран системы Turbo Pascal

После успешного запуска системы программирования вы увидите на экране компьютера исходный экран системы. На экране системы можно выделить три основные части: верхнюю строку, основную часть экрана и нижнюю строку (рис. 2.13).

Верхняя строка исходного экрана системы программирования называется *строкой меню* и содержит десять разделов. Для того чтобы активизировать строку меню (т. е. привести ее в рабочее состояние), необходимо нажать клавишу <F10>. После этого одно из меню будет подсвечено (как правило, зеленым цветом). Перемещаться по строке меню можно клавишами <стрелка влево> и <стрелка вправо>.

После нажатия клавиши <Enter> подсвеченное меню будет раскрыто, т. е. появится ниспадающее меню. То же самое можно сделать, щелкнув строку меню мышью. После щелчка мышью или нажатия <Enter> в ниспадающем меню будут видны все пункты выделенного меню. Часть пунктов меню закрашена серым цветом. Эти пункты соответствуют командам, которые в данный момент времени недоступны. Те же команды, которые можно выпол-

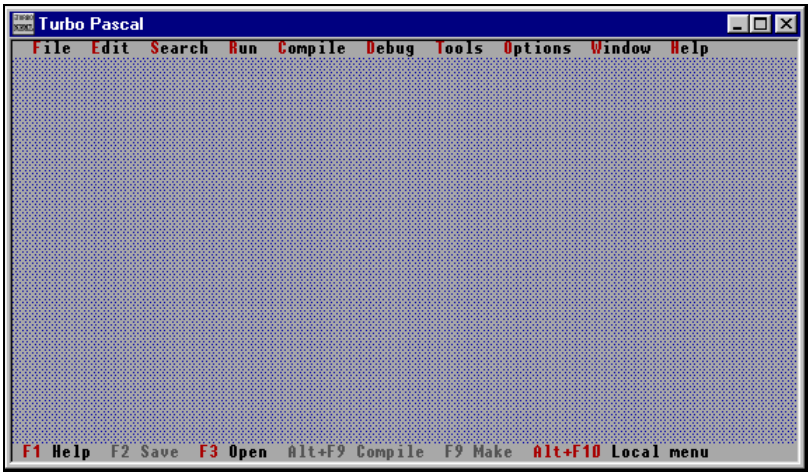


Рис. 2.13. Исходный экран системы программирования

нить, закрашены черным цветом. Справа от некоторых команд указываются названия клавиш или сочетаний клавиш, с помощью которых можно быстро выполнить данные команды, не открывая меню. Всего основное меню системы Turbo Pascal содержит десять разделов. Далее приведены названия разделов в том порядке, в котором они расположены в строке меню, и краткая информация по каждому из них:

- **File** — операции с файлами;
- **Edit** — редактирование исходного текста программы;
- **Search** — поиск и замена группы символов в тексте программы;
- **Run** — компиляция программы с запуском ее на выполнение;
- **Compile** — компиляция программы без запуска на выполнение;
- **Debug** — отладка программы (поиск ошибок в программе и просмотр результатов ее выполнения);
- **Tools** — работа с внешними по отношению к системе программирования программами (например, с ассемблером);
- **Options** — настройка параметров системы программирования;
- **Window** — операции с окнами, открытыми в текущем сеансе работы;
- **Help** — получение справочной информации по работе с системой программирования.

На рис. 2.14 изображен экран системы Turbo Pascal после активизации меню (раскрыто меню **File**). Данный раздел содержит команды, используемые для операций с файлами: создания новых файлов, открытия ранее созданных, со-

хранения изменений в файлах, их переименования, смены каталога, в который по умолчанию записываются файлы, а также для выхода из системы программирования. Например, выход из системы программирования по завершении сеанса работы в Turbo Pascal осуществляется командой **File | Exit**. Это означает, что для выполнения данной команды нужно произвести следующие действия: открыть меню **File**, стрелкой выбрать нужный пункт в разделе и затем нажать клавишу <Enter>, либо щелкнуть данный пункт мышью. После выполнения этой команды вы вернетесь в среду той операционной системы, либо файловой оболочки, из которой был запущен Turbo Pascal.

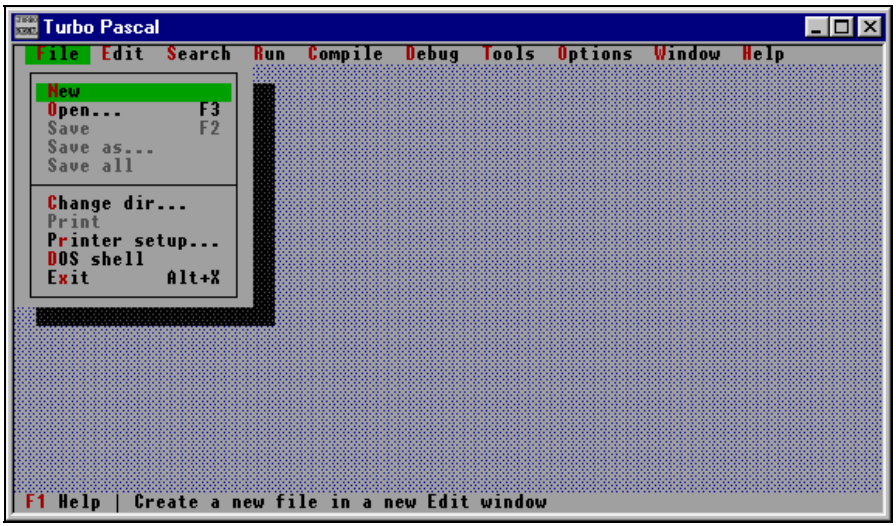


Рис. 2.14. Экран системы программирования с активизированной строкой меню

Под строкой меню расположена основная *рабочая область* системы программирования, в которой впоследствии мы будем вводить текст программы. В нижней части экрана находится *строка состояния* (у нее есть и другое название — *строка статуса*). В этой строке перечислены названия тех команд меню, которые чаще всего используются в текущем режиме работы, и соответствующих им клавиш или комбинаций клавиш. По мере того как вы будете осваивать систему программирования, вы выучите значения наиболее часто употребляемых клавиш и их сочетаний, что ускорит работу над программой. Названия тех из них, которые можно непосредственно использовать в данный момент, выделены красным цветом. Сразу после открытия системы программирования можно использовать клавиши <F1>, <F3> и комбинацию клавиш <Alt>+<F10>. При изменении режима работы системы программирования меняется и содержимое строки состояния.

2.5. Алфавит языка Паскаль

Перед тем как приступить к созданию первой программы на языке Паскаль, пользователю необходимо получить представление об основных компонентах, из которых складывается эта программа.

Любая программа на любом языке программирования записывается в виде символов. Символы эти не могут быть произвольными. Подобно естественным языкам, таким как русский, английский, французский и др., которые люди используют при общении друг с другом, языки программирования, на которых человек общается с компьютером, имеют свой алфавит. В данном случае *алфавит* — это набор букв, цифр и других символов, используемых при написании программ.

Алфавит языка Паскаль включает в себя:

- буквы латинского алфавита от A до Z. Буквы могут быть как прописными, так и строчными, т. к. компилятор языка Паскаль при обработке программ не делает различия между ними;
- цифры от 0 до 9;
- специальные символы. К специальным символам относятся знаки математических операций, знаки пунктуации и некоторые другие. Спецсимволы подразделяются на:
  - одиночные

    + - * / = < >

    [ ] , ( ) : ;

    ^ . @ { } $ #
  - парные

    <= >= := .. (* *) (. .)

Наряду с цифрами, буквами и специальными символами Паскаль содержит ряд *служебных слов*, значения которых заранее определены и не могут изменяться пользователем. К таким зарезервированным словам относятся:

```
asm  array  begin  case  const  constructor  destructor  div
do  downto  else  end  exports  file  for  function  goto
if  implementation  in inherited  inline  interface  label
library  mod  nil  not  object  of  or  packed  procedure
program  record repeat  set  shl  shr  string  then  to
type  unit  until  uses  var  while  with  xor
```

Из отдельных символов и служебных слов в Паскале складываются операторы. *Оператором* называется выражение, обозначающее и описывающее какую-либо операцию, осуществляемую в программе.

Использование букв русского алфавита в программе допускается только в *комментариях* (пояснениях к программе, не влияющих на ход ее выполнения) и в тексте, который выводится на экран компьютера.

2.6. Структура программы на языке Паскаль

Программа, как мы уже знаем, представляет собой последовательность действий, выполняемых в процессе решения поставленной задачи. Эти действия описываются в виде операторов языка Паскаль, являющихся основными элементами, из которых складывается программа. При этом операторы работают с различными величинами: постоянными (константами) и переменными.

Константой называется величина, которая в процессе выполнения программы остается неизменной. Константа может быть либо числом (числовая константа), либо некоторым произвольным набором символов (текстовая константа). В ряде случаев для удобства работы константам дают имена, состоящие из букв латинского алфавита и цифр.

Переменной называется величина, которая может изменяться в ходе выполнения программы. Каждая переменная должна иметь собственное имя, значение и тип. *Имя переменной* обозначается подобно имени константы латинскими буквами и цифрами, причем начинаться имя переменной обязательно должно с буквы.

Каждая отдельная переменная может принимать значения только определенного *типа*. Это могут быть, например, целые или вещественные числа. В первом случае переменная называется целочисленной, во втором — переменной вещественного типа или вещественной. *Значениями* переменной могут быть не только числа, но и отдельные символы. В таком случае переменная называется символьной. Если же значением переменной является не отдельный символ, а группа символов (такая группа называется в Паскале строкой), то соответствующая переменная будет именоваться строковой. Позднее мы познакомимся и с другими типами переменных.

В Паскале тип каждой используемой в программе переменной обязательно должен быть описан в соответствующем разделе программы. Тип переменной определяет не только область значений переменной, но и набор операций, которые можно производить над переменной. Такой набор является специфическим для каждого типа переменной. Например, над числовыми переменными можно производить операции умножения и деления, а для строковой переменной существует операция определения ее длины. Под каждую переменную в памяти компьютера отводится некоторая область, в которой хранится ее значение. Это значение сохраняется в неизменном виде до тех пор,

пока переменная не получит новое значение. Операция изменения значения переменной называется операцией *присваивания*.

Операторы, а также обрабатываемые ими величины, не могут располагаться в программе произвольным образом. Элементы программы должны находиться в определенных разделах программы. Программа на языке Паскаль состоит из разделов, каждый из которых начинается специальным ключевым словом, характерным для конкретного раздела:

- `program` — заголовок программы;
- `uses` — раздел подключения модулей — библиотек дополнительных процедур и функций;
- `label` — раздел описания меток безусловного перехода;
- `const` — раздел описания констант;
- `type` — раздел описания типов данных;
- `var` — раздел описания переменных;
- `function` — раздел описания функций;
- `procedure` — раздел описания процедур;
- `begin` — начало тела программы (основной части программы);
- `end` — конец программы.

Большинство перечисленных разделов не являются обязательными. Единственным обязательным разделом является тело программы, начинающееся со служебного слова `begin` и заканчивающееся служебным словом `end` с точкой на конце. Общее описание разделов программы приведено далее.

Заголовок программы имеет вид:

```
program название_программы
```

Название программы может быть произвольным, но для того чтобы легче было ориентироваться в имеющихся программах, рекомендуется давать программе название, совпадающее с именем файла, в котором она хранится. Обратите внимание, что в названии программы не должно быть пробелов; в случае необходимости пробел можно заменить знаком подчеркивания. Также в названии программы не должно быть точек, запятых, точек с запятыми — такие имена будут восприниматься системой программирования как ошибочные. В имени программы нельзя использовать также русские буквы, круглые скобки, вопросительный и восклицательный знаки. Желательно, чтобы имя программы было связано по смыслу с ее содержанием.

Со служебного слова `uses`, что в переводе означает "использует", начинается *раздел подключения дополнительных модулей*. При этом после служебного

слова `uses` следует перечисление этих модулей через запятую. В стандартную поставку Turbo Pascal входит несколько стандартных модулей: `Crt` — модуль для работы в текстовом режиме, графический модуль `Graph` и др. На применении модуля `Crt` мы в дальнейшем остановимся подробнее. Модули позволяют использовать в программе дополнительные команды и операторы.

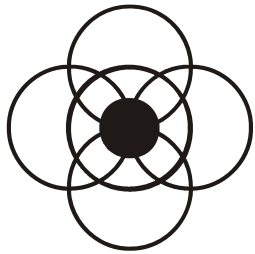
В *разделе объявлений* описываются константы и переменные. Описание констант начинается со служебного слова `const`, переменных — со слова `var`. После описания группы переменных, относящихся к одному типу, в программе ставится точка с запятой.

Наряду с указанными ранее в разделе могут присутствовать описания других элементов программы: функций — соответствующий раздел начинается со слова `function`, процедур — раздел начинается со слова `procedure`, меток — раздел начинается со слова `label`, типов данных — раздел начинается с `type`. Для чего используются все эти элементы и как их описывать, мы более подробно разберем в следующих главах книги. В самых простых программах, в которых не используются никакие из перечисленных ранее элементов, раздел описаний может отсутствовать.

Основную часть программы предваряет служебное слово `begin`. После него никаких знаков препинания не ставится.

В основной части программы находятся операторы, ответственные за ее выполнение. Операторы отделяются друг от друга точками с запятой, причем в одной строке может находиться несколько операторов, а если оператор не помещается в одну строку, то его можно переносить на следующую строку. После последнего в программе оператора точку с запятой ставить необязательно.

Программа завершается служебным словом `end`, после которого обязательно ставится точка.



ГЛАВА 3

Простейшие программы в Паскале

Усвоив основные компоненты, из которых складывается программа на языке Паскаль, можно приступать к составлению и отладке простейших программ. Такого типа программы являются, как правило, линейными. Они содержат операторы ввода, вывода и присваивания. В этих программах используется управляющая структура следования, т. е. один оператор в программе следует за другим, и он выполняется после того, как выполнил свои функции предыдущий. В этом разделе рассмотрены и другого типа программы, осуществляющие не только ввод или вывод информации, но и ее *обработку*. Простейшим примером такой обработки информации являются арифметические вычисления, в ходе которых по имеющимся исходным данным подсчитывается результат.

3.1. Первая программа на Паскале

Для того чтобы приступить к работе над программой, необходимо предварительно создать новый файл, в котором будет записан текст данной программы. Создание файла производится выполнением команды **File | New** (для этого нужно щелкнуть мышью пункт **New** в меню **File** или установить курсор на этот пункт стрелками клавиатуры и нажать клавишу <Enter>). После этого на экране появится окно вновь созданного файла, в котором можно вводить текст программы на языке Паскаль (рис. 3.1).

Можно настроить систему программирования таким образом, что окно для вновь создаваемого файла будет открываться автоматически сразу после запуска системы программирования. Для этого следует после открытия окна для нового файла записать изменения в файл turbo.tp. В файле turbo.tp, входящем в состав системы программирования, хранится информация о настройках системы. Запись изменений в файл настроек производится следую-

щим образом. Нужно щелкнуть мышью меню **Options**, а в открывшемся разделе команду **Save TURBO.TP** (рис. 3.2). При следующем запуске системы программирования сразу, без захода в меню, откроется окно для нового файла. Этот файл с текстом новой программы по умолчанию получает имя Noname00.pas. Следующие созданные вами файлы получат соответственно имена Noname01.pas, Noname02.pas и т. д.

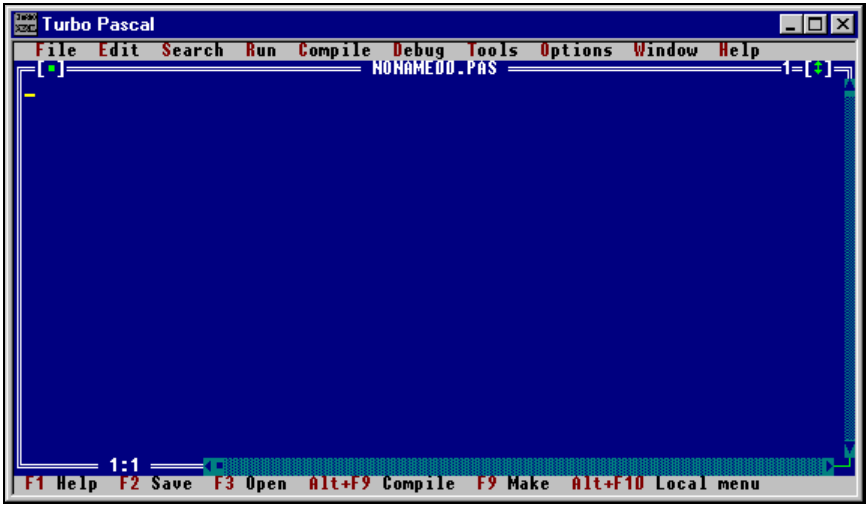


Рис. 3.1. Окно для ввода текста программы в системе Turbo Pascal

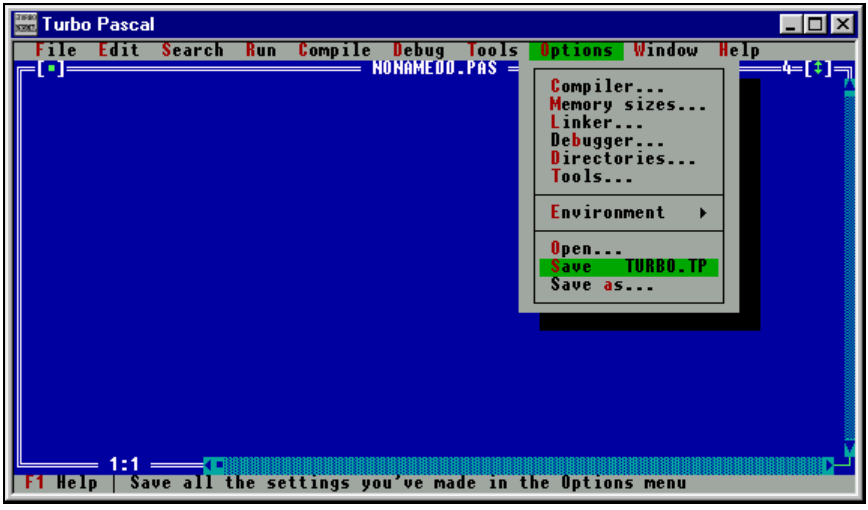


Рис. 3.2. Сохранение настроек системы программирования

Теперь, когда пользователь имеет общее представление о том, как должна выглядеть программа на языке Паскаль, можно приступить к созданию первой программы на этом языке. Первая программа должна вывести на экран компьютера сообщение. Текст этого сообщения будет следующим: "Моя первая программа на языке Паскаль".

Введем текст программы с клавиатуры компьютера по строкам (рис. 3.3, листинг 3.1).

Листинг 3.1. Первая программа на языке Паскаль

```
program pervprog;
begin
writeln ('Моя первая программа на языке Паскаль')
end.
```

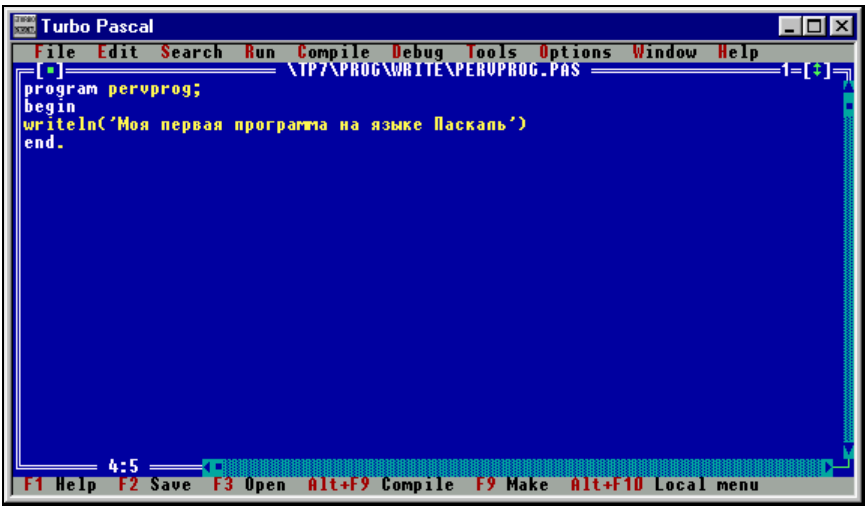


Рис. 3.3. Текст первой программы на Паскале

Место, куда пользователь вводит очередной символ, отмечается специальной меткой. Эта метка называется *курсором* и имеет вид мигающей горизонтальной черты. Ввод очередной строки завершается нажатием клавиши <Enter>. После нажатия этой клавиши курсор переходит на следующую строку. Если при наборе программы пользователь сделал опечатку, то неверный символ можно удалить, нажав клавишу <Backspace>, стирающую символ, находящийся слева от курсора. Для удаления того символа, на котором курсор установлен в данный момент, нужно нажать клавишу <Delete>. Для контроля правильности вводимого текста советуем обратить внимание на следующее:

ряд служебных слов (в частности слова program, begin и end) в случае их правильного написания выделятся на синем фоне экрана белым цветом в отличие от остального текста, имеющего желтый цвет.

Разберем подробно текст введенной нами программы. Сделать это будет несложно, т. к. программа занимает всего четыре строки.

Программа имеет заголовок, который обязательно должен начинаться со служебного слова program. После пробела вводится собственное имя программы (в данном случае имя pervprog). Раздел объявлений здесь отсутствует, т. к. в программе не задействована ни одна переменная, а используемая константа не имеет собственного имени. Перед основной частью программы обязательно ставится служебное слово begin (начало).

В самой же основной части содержится единственный оператор. Это оператор вывода, который состоит из служебного слова writeln и выводимой на экран компьютера информации, заключенной в скобки. Эта информация в данном случае состоит из одного элемента — текста, расположенного между апострофами (одиночными кавычками). Такие тексты в Паскале называются строковыми константами или просто *строками*. Строка может содержать любые символы (включая буквы русского алфавита), кроме апострофа. При выполнении оператора writeln эта строка выводится на экран компьютера, причем ограничивающие ее апострофы не выводятся, а затем курсор перемещается на следующую строку. Программа обязательно заканчивается служебным словом end (конец), после которого ставится точка — признак конца программы.

После того как текст программы набран, содержащий его файл нужно сохранить на жестком диске компьютера или на дискете. Сохранение файла включает в себя указание каталога (папки), в котором данный файл сохраняется, и имени, под которым сохраняется файл. Рассмотрим основные правила наименования файлов.

Во-первых, файлу нельзя оставлять даваемое по умолчанию системой программирования имя Noname (что в переводе с английского означает "безымянный"). Во время следующего сеанса программирования текст новой программы также запишется в файл с именем Noname, и текст предыдущей программы будет потерян. Поэтому нужно дать файлу какое-либо имя, отличное от Noname, причем желательно, чтобы это имя по смыслу было связано с содержанием файла. Например, файл, содержащий программу, складывающую два числа, можно назвать Summa2.pas. Имена файлов могут содержать буквы латинского алфавита и цифры. В именах файлов не следует использовать буквы кириллицы, пробелы, а также следующие символы:

* = + [ ] ; : , . < > / ?

Так как система Turbo Pascal является приложением операционной системы MS-DOS, то необходимо помнить еще об одном ограничении: имена файлов с программами на Паскале (как и все прочие имена файлов в MS-DOS) не могут содержать более 8 символов (не считая 3 символов, зарезервированных для расширения имени). Всего имя файла, таким образом, может содержать не более 11 символов. Такой способ наименования файлов часто называют "система 8+3". Файл, создаваемый в системе программирования Turbo Pascal, должен иметь расширение pas.

Сохранение файла на диске под новым именем осуществляется посредством команды **File | Save as** (Файл | Сохранить как). С помощью этого пункта меню задается также каталог, в котором будет сохранен данный файл. Обратите внимание, что после названия команды стоит многоточие. Многоточие говорит о том, что при выполнении данной команды открывается диалоговое окно. Внешний вид окна сохранения файла, которое открывается после щелчка мышью или нажатия клавиши <Enter> на команде **Save as**, показан на рис. 3.4.

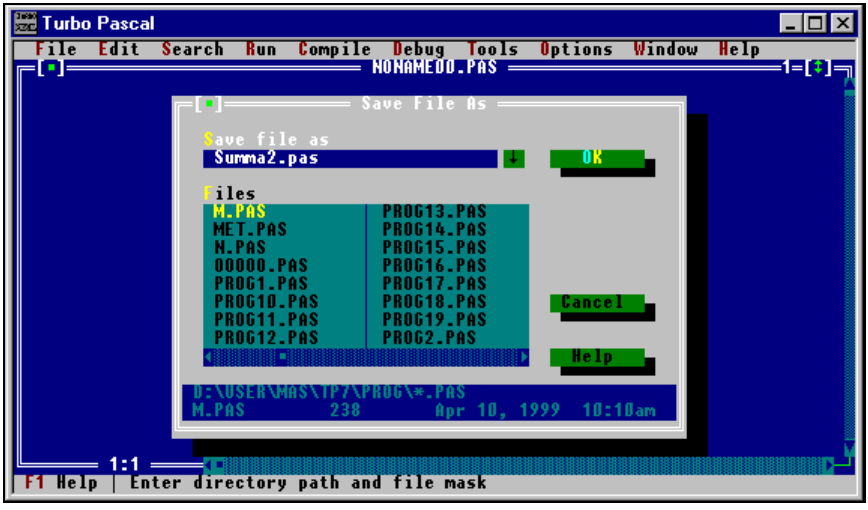


Рис. 3.4. Диалоговое окно сохранения файла

Диалоговое окно сохранения файла содержит несколько элементов.

- Текстовое поле, в которое можно вводить новое имя файла. Это поле находится в верхней части диалогового окна и выделено синим цветом. Само диалоговое окно имеет светло-серый цвет.
- Окно, содержащее список файлов с программами на Паскале, находящихся в том каталоге, куда по умолчанию записывается файл. Это окно занимает центральную часть диалогового окна и выделено бирюзовым цветом.

Если имена всех файлов, находящихся в каталоге, не помещаются в окне списка, то в нижней части этого окна появляется полоса прокрутки (полосы прокрутки часто также называют скроллерами), которая позволяет перемещаться по списку файлов путем перетаскивания находящегося на полосе бегунка, иначе называемого также лифтом. Полоса прокрутки выделена синим цветом, а бегунок имеет вид маленького квадратика бирюзового цвета.

- Стандартные экранные кнопки **OK** (подтверждение операции), **Cancel** (отмена операции) и **Help** (получение помощи). Экранные кнопки находятся в правой части диалогового окна и выделены зеленым цветом.
- Информационная строка, содержащая сведения о текущем каталоге и о выделенном файле (один файл в каталоге всегда выделен другим цветом). Строка расположена в нижней части диалогового окна и выделена синим цветом. Для каталога в информационной строке указывается путь к нему, начиная с имени диска, на котором он находится. Для файла, помимо его имени, указывается его размер в байтах, а также дата и время его создания или последнего изменения.

Перемещение между элементами диалогового окна можно производить с помощью клавиши <Tab> или мыши. Перемещение внутри элементов производится с помощью клавиш управления курсором (<стрелка влево>, <стрелка вправо>, <стрелка вверх>, <стрелка вниз>) или мышью.

Если каталог, заданный по умолчанию, устраивает пользователя, то ему необходимо после ввода имени сохраняемого файла только щелкнуть два раза мышью экранную кнопку **OK** или выделить эту кнопку клавишей <Tab> и нажать <Enter>, после чего диалоговое окно закроется и файл сохранится в текущем каталоге под указанным именем. Но, как уже говорилось в предыдущей главе, лучше сохранить файл с текстом программы в специальном каталоге prog. Для того чтобы изменить заданный по умолчанию каталог, нужно уметь перемещаться с помощью элементов диалогового окна по файловой структуре компьютера.

Для того чтобы перейти в каталог верхнего уровня (на один уровень вверх), необходимо выделить в текущем каталоге **..** (символ надкаталога) и нажать клавишу <Enter> или щелкнуть два раза мышью. Если символ надкаталога не виден в окне, то нужно прокручивать содержимое текущего каталога с помощью лифта до тех пор, пока вы его не обнаружите. Для перехода в подкаталог (на один уровень вниз) необходимо выделить в текущем каталоге имя этого подкаталога и нажать клавишу <Enter> или щелкнуть два раза мышью. После того как пользователь открыл нужный каталог, в текстовом поле в верхней части диалогового окна можно ввести имя сохраняемого файла.

Можно сразу ввести имя диска, на котором будет сохранен файл, каталога, в котором он будет сохранен, и самого файла непосредственно в текстовом поле. При этом не нужно переходить из каталога в каталог, как было описано ранее, но для начинающего пользователя это представляет определенные сложности, т. к. при этом нужно правильно указать путь к каталогу и файлу.

Если пользователь не будет сохранять данный файл (например, если он был создан только в учебных целях), то нужно щелкнуть кнопку **Cancel**.

После закрытия окна сохранения файла правильность выполненных действий можно проконтролировать по верхней части рамки рабочей области, в которой будет указано полное имя сохраненного файла, включающее имя каталога (каталогов), содержащих данный файл. Например, в случае успешного сохранения файла pervprog в каталоге prog, находящемся в каталоге TP7, который расположен на логическом диске D:, вверху рабочей области пользователь увидит следующее полное имя файла: **TP7\prog\pervprog.pas**.

Если в дальнейшем в ходе работы над программой пользователь будет корректировать ее текст, то изменения в тексте можно быстро сохранить, выполнив команду **File | Save**. При этом диалоговое окно открываться не будет, а все изменения в файле будут записаны на диск автоматически.

Когда ввод текста программы завершен, и программа записана в долговременную память компьютера, ее можно запускать на выполнение. Это действие можно произвести двумя способами. Первый из них заключается в том, что в начале с помощью команды **Compile** (компиляция), находящейся в одноименном разделе меню, создается исполняемый файл с расширением exe. Затем полученный файл можно запускать на выполнение. Этот способ предпочтителен в том случае, если мы будем использовать созданную программу за пределами системы программирования. Созданный таким образом исполняемый файл можно запускать штатными средствами операционных систем MS-DOS или Windows. Но в данном случае такая задача перед нами не стоит, поэтому воспользуемся для запуска программы другим, более удобным способом. Мы используем пункт **Run** в одноименном разделе меню. Эта команда осуществляет компиляцию файла и затем сразу запускает его на выполнение. Ту же самую операцию можно выполнить, нажав сочетание клавиш <Ctrl>+<F9>.

В том случае, если при компиляции в исходном тексте программы были обнаружены ошибки (при копировании текста готовой программы такими ошибками, как правило, являются опечатки), на экране появится красная строка с сообщением о номере и характере сделанной ошибки. Например, если вы забыли заключить текстовую строку в апострофы, то на экране будет следующее сообщение, которое означает просто "синтаксическая ошибка":

```
Error 5: Syntax error
```

Если же вы забыли поставить заключительную точку после служебного слова `end`, то на экране компьютера можно будет прочитать сообщение о неправильном конце файла:

```
Error 10: Unexpected end of file
```

После устранения ошибок произойдет следующее: экран компьютера "моргнет" и вернется в исходное состояние, т. е. результатов проделанной работы мы не увидем. Для просмотра результатов следует воспользоваться командой **Output** (Вывод) из раздела меню **Debug**. Слово **Debug** в переводе на русский язык означает "отладка" (буквально — "вылавливание насекомых"), но этот раздел содержит не только команды, связанные непосредственно с отладкой программы, но и команды просмотра полученных результатов. При выполнении команды **Output** на экране компьютера ниже текста программы появится дополнительное окно с результатами работы (рис. 3.5). В верхней правой части окна видна цифра 2 — порядковый номер этого окна. Номер 1 имеет окно с исходным текстом программы. Таким образом мы вывели требуемый текст на экран компьютера.

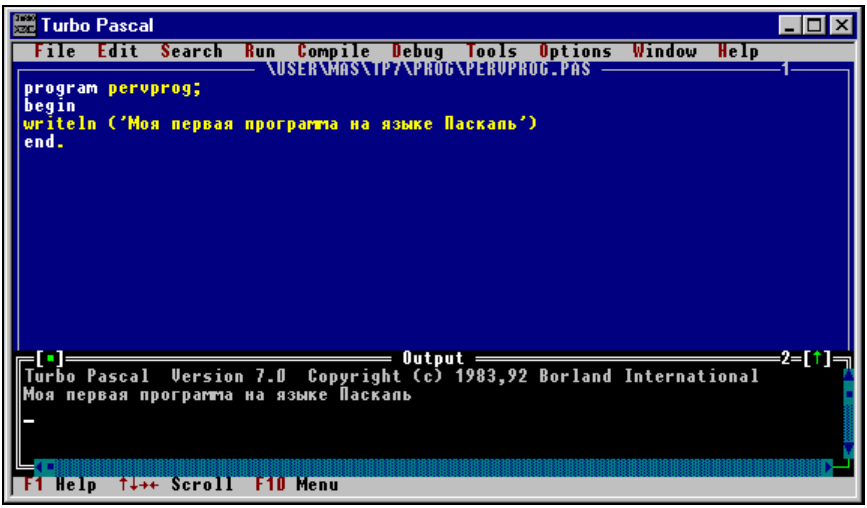


Рис. 3.5. Результаты работы первой программы

Если результаты необходимо просмотреть в полноэкранном режиме, а не в окне вывода, то необходимо выполнить команду **User screen** (экран пользователя) из того же меню **Debug** либо нажать сочетание клавиш <Alt>+<F5>. Можно развернуть во весь экран и окно, выводимое командой **Output**, щелкнув мышью стрелку в верхнем правом углу данного окна. Для того чтобы от экрана с результатами вернуться к экрану с исходным текстом программы, следует нажать любую алфавитно-цифровую клавишу.

Наряду с оператором `writeln` для вывода информации на экран компьютера можно использовать также другой оператор вывода `write`. Этот оператор действует подобно оператору `writeln`, но особенность оператора `write` заключается в том, что после его выполнения курсор не перемещается на следующую строку, а остается в той же строке, в которой выводилась информация. Различие между операторами `write` и `writeln` продемонстрируем на следующем примере. Пусть в программе нам необходимо вывести текст: "Мы изучаем программирование".

Если этот текст вывести тремя операторами `write`:

```
write('Мы');
write(' изучаем ');
write('программирование');
```

то при выполнении программы вся фраза будет выведена в одну строку:

```
Мы изучаем программирование
```

Если же тот же текст вывести тремя операторами `writeln`:

```
writeln('Мы');
writeln(' изучаем ');
writeln('программирование');
```

то каждое слово фразы будет начинаться с новой строки, и при выполнении программы на экране мы увидим следующее:

```
Мы
изучаем
программирование
```

Информация, заключенная в операторах `write` или `writeln` в скобках, называется *списком вывода*. Такой список может включать в себя один или несколько элементов, которые могут быть как константами, подобно тексту в нашей программе, так и переменными. Элементы списка вывода внутри скобок отделяются друг от друга запятыми и выводятся при выполнении оператора на экран в той последовательности, в которой они перечислены в самом операторе.

3.2. Цветовое оформление результатов выполнения программы

Окно с результатами работы программы необязательно должно быть черно-белым. Текст может выводиться различными цветами. Цветным может быть и фон, на котором выводится текст. Для этого необходимо использовать мо-

дуль Crt, входящий в состав системы программирования. *Модулем* называется блок, входящий в состав стандартной библиотеки системы Turbo Pascal и обеспечивающий дополнительные возможности системы при создании программ. По умолчанию при запуске системы в оперативную память загружается только модуль System. Для подключения других модулей необходимо дать специальную команду. В частности, для подключения модуля Crt первой командой программы, находящейся сразу после заголовка, должна быть команда Uses Crt. Цвет символов задается с помощью команды TextColor. После служебного слова TextColor в скобках указывается непосредственно цвет символов. Всего в Turbo Pascal используется 16 стандартных цветов. Вот их названия:

Black — черный;
Blue — синий;
Green — зеленый;
Cyan — бирюзовый;
Red — красный;
Magenta — фиолетовый;
Brown — коричневый;
LightGray — светло-серый;
DarkGray — темно-серый;
LightBlue — голубой;
LightGreen — светло-зеленый;
LightCyan — светло-бирюзовый;
LightRed — светло-красный;
LightMagenta — светло-фиолетовый;
Yellow — желтый;
White — белый.

Для задания цвета фона используется команда TextBackground. Формат ее аналогичен команде TextColor, но эта команда позволяет использовать только 8 цветов:

```
Black  Red  Blue  Magenta  Green  Brown  Cyan  LightGray
```

Цвета в командах TextColor и TextBackground можно обозначать не только указанными ранее словами, но и числами. В некоторых случаях использование чисел даже удобнее. Для кодирования цветов используются следующие числа:

1 — черный;
2 — синий;
2 — зеленый;
3 — бирюзовый;
4 — красный;
5 — фиолетовый;
6 — коричневый;
7 — светло-серый;
8 — темно-серый;
9 — голубой;
10 — светло-зеленый;
11 — светло-бирюзовый;
12 — светло-красный;
13 — светло-фиолетовый;
14 — желтый;
15 — белый;
128 — мерцание.

Поясним последний числовой код. Он не используется самостоятельно, а является дополнительным, т. е. употребляется вместе с каким-либо числовым значением цвета для создания эффекта мерцания. Для этого число 128 добавляется к основному числовому значению. Поясним сказанное следующими примерами: если в тексте программы имеется команда

```
Textcolor(lightgreen);
```

то следующий за ней текст будет выводиться светло-зеленым цветом. Аналогичный результат обеспечит и команда

```
Textcolor(10);
```

если же мы используем команду

```
Textcolor(10+128);
```

то текст будет выводиться светло-зеленым цветом и при этом будет мерцать.

Составим программу, которая выводит на светло-сером фоне следующий текст (рис. 3.6):

```
Эта программа представляет собой пример
использования цветовой палитры
системы программирования
Turbo Pascal 7.0
```

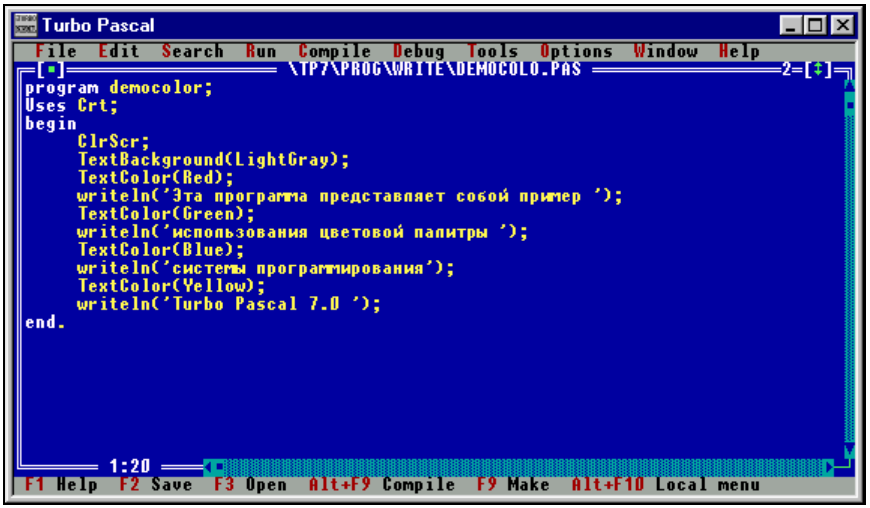
Первая строка этого текста будет выведена красным цветом, вторая — зеленым, третья — синим, а четвертая — желтым. Обратите внимание, что операторы в программе отделяются друг от друга точкой с запятой. В начале программы для очистки экрана дается команда `ClrScr`. Эта команда работает только в том случае, если к программе подключен модуль `Crt`. Далее командой `TextBackground` задается фон текста. Перед выводом очередной строки текста командой `TextColor` предварительно указывается ее цвет, а затем сам этот текст выводится оператором `writeln`. В листинге 3.2 приводится текст данной программы.

Листинг 3.2. Вывод текста различными цветами

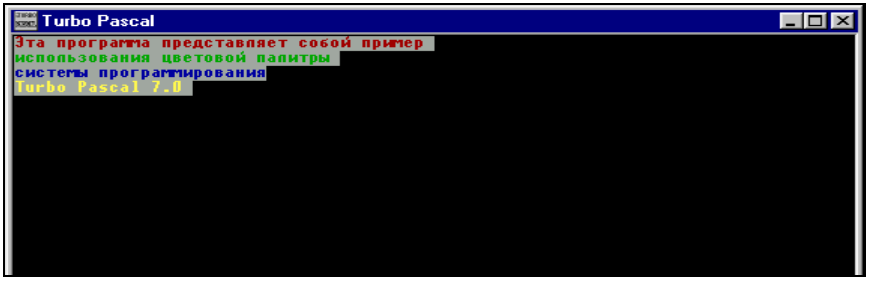
```
program democolor;
Uses Crt;
begin
     ClrScr;
     TextBackground(LightGray);
     TextColor(Red);
     writeln('Эта программа представляет собой пример ');
     TextColor(Green);
     writeln('использования цветовой палитры ');
```

```
     TextColor(Blue);
     writeln('системы программирования');
     TextColor(Yellow);
     writeln('Turbo Pascal 7.0 ');
end.
```

Отметим еще одну особенность данной программы. Если в первой программе все ее строки вводились, начиная с первой позиции в строке, то в этой программе перед рядом строк стоит несколько пробелов, и соответственно текст в этих строках сдвигается вправо. Такая группа пробелов называется *отступом* и используется для того, чтобы пользователь более четко и наглядно представлял себе структуру программы. В данной программе с отступом написаны операторы, входящие в основную часть программы. В программах, приводимых в последующих главах книги, с помощью отступов мы будем выделять отдельные блоки и программные структуры внутри основной части программы, что в значительной степени облегчит понимание смысла программ.



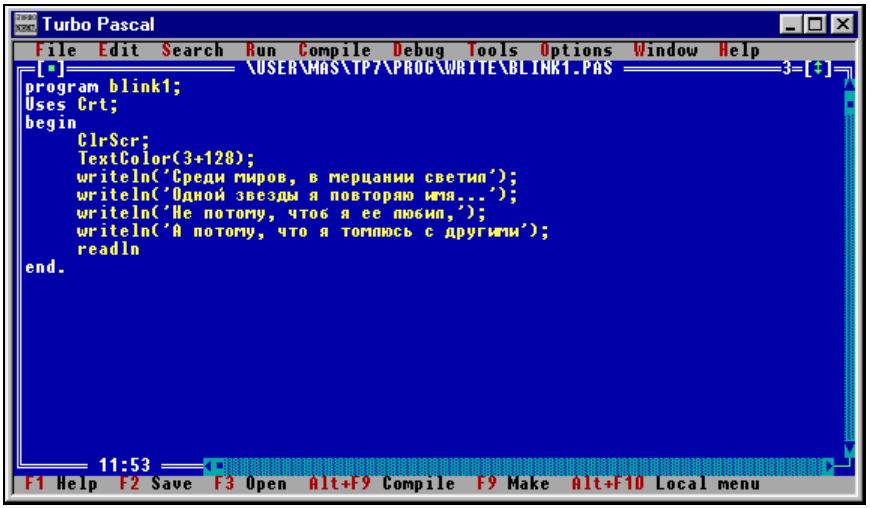
а



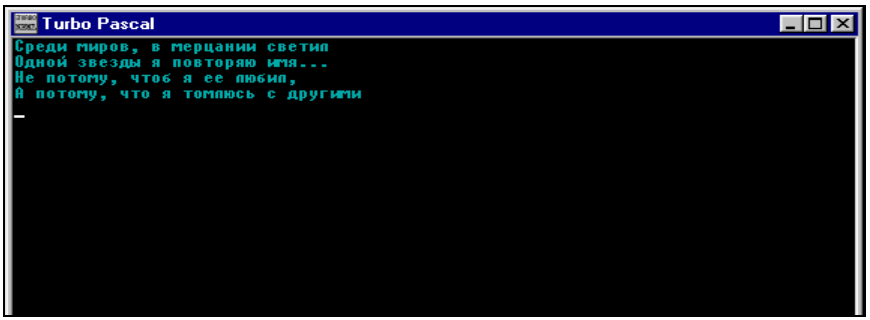
б

Рис. 3.6. Программа, демонстрирующая возможности модуля `Crt` *(а)*, и результат ее работы *(б)*

В программе на рис. 3.6 использовались словесные наименования цветов выводимых строк, а в следующей программе для обозначения цветов мы воспользуемся числовыми кодами (рис. 3.7). Эта программа выводит на экран компьютера четверостишие поэта И. Анненского на черном фоне бирюзовым цветом, причем текст четверостишия будет мерцать. Бирюзовый цвет текста четверостишия закодирован цифрой 3, к которой добавлено слагаемое 128, обеспечивающее эффект мерцания. В листинге 3.3 приводится текст программы.



а



б

Рис. 3.7. Программа, выводящая мерцающий текст *(а)*, и результат ее работы *(б)*

Листинг 3.3. Вывод текста с эффектом мерцания

```
program blink1;
Uses Crt;
```

```
begin
     ClrScr;
     TextColor(3+128);
     writeln('Среди миров, в мерцании светил');
     writeln('Одной звезды я повторяю имя...');
     writeln('Не потому, чтоб я ее любил,');
     writeln('А потому, что я томлюсь с другими');
     readln
end.
```

Отметим еще одно отличие данной программы от двух предыдущих. Первые две программы, в которых отсутствует эффект мерцания, будут нормально работать и в полноэкранном, и в оконном режимах. Программа же, изображенная на рис. 3.7, может создавать эффект мерцания только в полноэкранном режиме. Естественно, что эффект мерцания не виден на иллюстрации в книге, но вы увидите его, когда наберете текст данной программы на компьютере и запустите ее на выполнение.

Помимо указанных ранее возможностей модуль `Crt` позволяет также устанавливать курсор в указанное программистом место на экране, а также создавать различные аудиоэффекты.

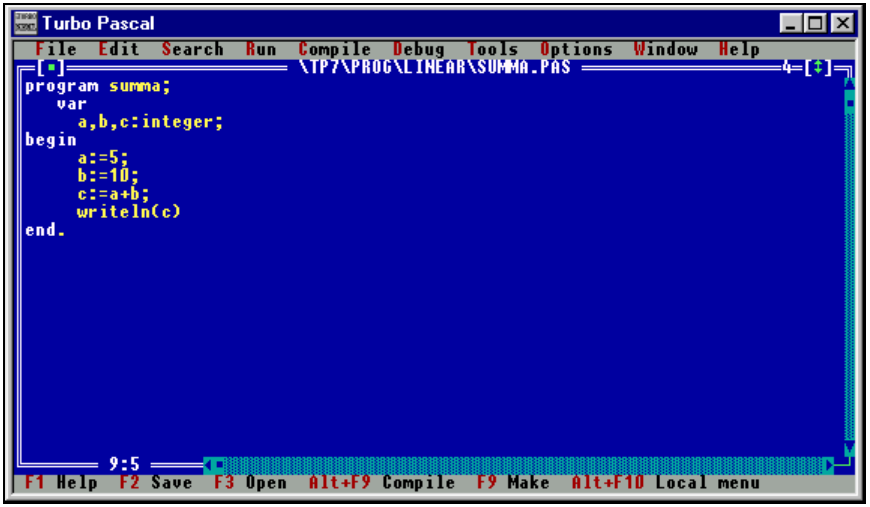
3.3. Использование переменных в программе

Программы, рассмотренные в предыдущем разделе, осуществляли вывод информации на экран компьютера. Теперь мы перейдем к задачам, в которых осуществляется не только ввод или вывод информации, но и ее *обработка*. Простейшим примером такой обработки информации являются арифметические вычисления, в ходе которых по имеющимся исходным данным подсчитывается результат. Составим программу, которая складывает два числа и выводит их сумму на экран компьютера (рис. 3.8, листинг 3.4). Для того чтобы сложить эти числа, необходимо в начале *присвоить* их каким-либо переменным (например, `a` и `b`), а потом описать эти переменные. Затем следует присвоить результат арифметической операции сложения третьей переменной (`c`) и вывести его на экран компьютера.

Листинг 3.4. Сложение двух чисел и вывод результата на экран

```
program summa;
   var
     a,b,c:integer;
```

```
begin
      a:=5;
      b:=10;
      c:=a+b;
      writeln(c)
end.
```



а

б

Рис. 3.8. Программа, складывающая два числа *(а)*, и результат ее работы *(б)*

Заметим, что в заголовке этой программы не указана команда подключения модуля `Crt`. Вообще говоря, операцию очистки экрана рекомендуется всегда производить в начале работы программы, т. к. в противном случае после запуска на выполнение текущей программы на экране могут оставаться "хвосты" (результаты работы предыдущей программы), что затруднит анализ полученных результатов работы текущей программы. В случае же, если мы не

забудем подключить модуль Crt, а в основной части программы первой командой будет команда очистки экрана ClrScr, то вывод результатов будет производиться на "чистый лист", что конечно гораздо удобней для пользователя.

Затем идет раздел описания переменных, который обязательно должен начинаться со служебного слова var (сокращение от английского *variable* — переменная). После служебного слова через запятую перечисляются сами переменные, используемые в программе, и после двоеточия указывается тип этих переменных. Обратите внимание на то, что в этом разделе в обязательном порядке должны быть описаны *все* используемые в данной программе переменные. Если мы забудем описать какую-либо из использованных в программе переменных, то при запуске этой программы на компиляцию мы увидим следующее сообщение:

```
Error 3: Unknown identifier
```

и программа не будет работать до тех пор, пока вы не исправите ошибку, описав недостающую переменную в разделе описаний после слова var.

В нашем случае для упрощения задачи будем складывать только целые числа, поэтому все переменные будут относиться к одному типу — integer (т. е. целые). В разделе описания должны быть описаны три переменные, используемые в программе: a, b и c. Затем в основной части программы переменным a и b должны быть присвоены конкретные числовые значения с помощью оператора присваивания, т. к. по умолчанию начальные значения переменных будут равны нулю.

Общий вид *оператора присваивания* в языке Паскаль:

имя_переменной:=*значение*;

где *значение* — присваиваемая величина, которая может быть числом, переменной или арифметическим выражением. Число присваивается непосредственно, а значение арифметического выражения в процессе выполнения программы вычисляется и присваивается стоящей в левой части оператора переменной.

После выполнения операции присваивания в ячейки компьютерной памяти, отведенные под переменную, будет занесено значение, записанное справа от знака равенства. Это значение будет оставаться в неизменном виде вплоть до выполнения в программе следующей операции присваивания.

В операторе присваивания после имени переменной обязательно должно стоять двоеточие. Если пропустить в записи оператора двоеточие, то получится не оператор присваивания, а операция сравнения. Отличие этих двух записей можно наглядно проиллюстрировать на следующем примере:

- запись `x=x+5` с математической точки зрения является неправильной;
- запись `x:=x+5` в Паскале является вполне допустимой. Эта запись означает, что переменной `x` присваивается новое значение, которое больше чем старое значение этой же переменной на 5. Например, если до выполнения операции присваивания значение `x` было равно 3, то после ее выполнения значение `x` станет равным 8.

В нашей программе мы присваиваем переменным `a` и `b` конкретные числовые значения — 5 и 10. Далее мы присвоим переменной `c` значение выражения `a+b`, т. е. в переменной `c` будет содержаться теперь значение суммы двух чисел. Теперь осталось вывести это значение на экран компьютера, что мы сделаем с помощью уже знакомого нам оператора `writeln`. Однако следует учесть, что в этом случае оператор `writeln` используется иначе, чем при выводе текста. Во-первых, имя переменной, в отличие от текстовой строки, не заключается в апострофы (хотя скобки нужно обязательно ставить и в этом случае). Во-вторых, при выполнении оператора вывода на экране мы увидим не имя переменной, а ее текущее значение. Оператор вывода в программе будет записан следующим образом:

```
writeln(c);
```

В результате же выполнения оператора на экран будет выведена сумма, содержащаяся в переменной `c` — число 15.

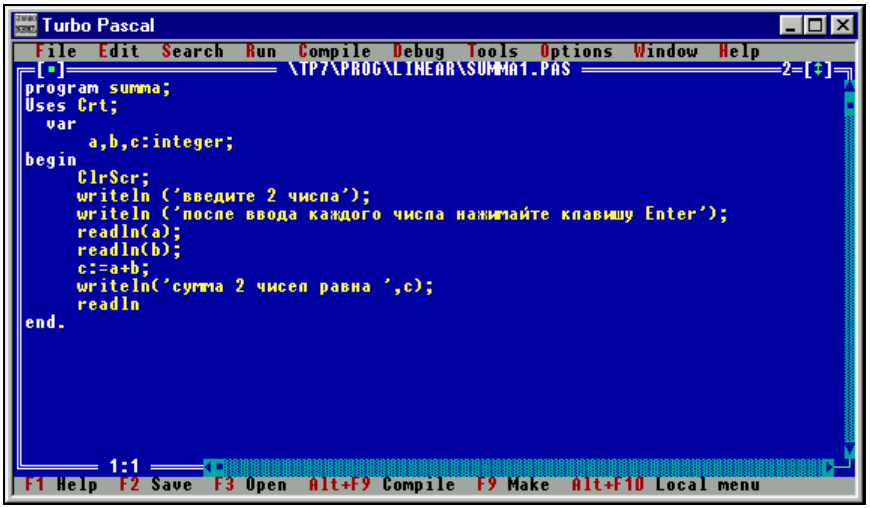
Эта программа (как и предыдущие) состоит из операторов, которые выполняются последовательно друг за другом. Такие программы называются *программами линейной структуры*.

Теперь попробуем усовершенствовать программу таким образом, чтобы она складывала любые два произвольных числа, которые пользователь введет с клавиатуры компьютера (рис. 3.9). Приведем текст усовершенствованной программы в листинге 3.5.

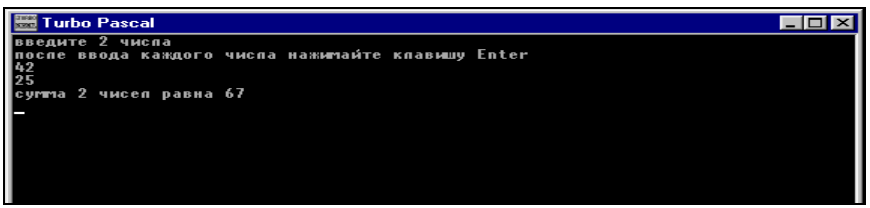
Листинг 3.5. Усовершенствованная программа сложения двух чисел

```
program summa;
Uses Crt;
  var
      a,b,c:integer;
begin
     ClrScr;
     writeln ('введите 2 числа');
     writeln ('после ввода каждого числа нажимайте клавишу Enter');
     readln(a);
     readln(b);
```

```
     c:=a+b;
     writeln('сумма 2 чисел равна ',c);
     readln
end.
```



а



б

Рис. 3.9. Усовершенствованная программа сложения двух чисел *(а)* и результат ее работы *(б)*

В этом новом варианте программы наряду с уже знакомыми нам операторами используется оператор ввода `readln`. Общий вид оператора `readln` похож на оператор `writeln`: он состоит из служебного слова `readln` и списка ввода, заключенного в скобки. Правда в отличие от списка вывода операторов `write` и `writeln` в списке ввода могут содержаться только имена переменных. В данной программе использованы два оператора `readln`, каждый из которых в списке ввода содержит по одной переменной:

```
readln(a);
readln(b);
```

С помощью операторов ввода переменным `a` и `b` будут присвоены численные значения. Это означает, что при выполнении программы в соответствующие

переменным `a` и `b` области памяти будут занесены те числа, которые пользователь ввел с клавиатуры. Теперь программу можно запускать на выполнение. После запуска программы появится пользовательский экран с мерцающим на нем курсором. Далее, можно вводить числовые значения переменных `a` и `b`. Для ввода каждого из двух чисел необходимо набрать его на клавиатуре, а затем нажать клавишу <Enter>. При этом нужно помнить о том, что вводимые числа по типу должны соответствовать переменным, т. е. быть целыми. При попытке ввести дробное число программа просто прекратит свою работу. После ввода второго числа на экране появится результат работы программы — сумма двух чисел.

Такая программа будет вполне работоспособна, но неудобна для пользователя, т. к. после ее запуска неопытному пользователю при виде "черного" экрана не понятно, что же нужно дальше делать. Поэтому сделаем в программе следующее дополнение. Перед операторами ввода поставим операторы вывода, которые будут выводить информацию, подсказывающую пользователю дальнейший ход его действий. Например, таким образом:

```
writeln ('введите 2 числа');
writeln ('после ввода каждого числа нажимайте клавишу Enter');
```

Пользователь будет таким образом знать, что именно ему предстоит делать даже в том случае, если он первый раз работает за компьютером. Теперь программа не только правильно работает, но и обеспечивает для пользователя *дружественный интерфейс*, т. е. удобный способ общения человека с программой.

В конец программы внесены еще 2 дополнения. В список вывода оператора `writeln` вставлен еще один элемент — текст, поясняющий полученный результат. Список вывода, таким образом, будет состоять из двух элементов, причем текст будет выводиться в неизменном виде, а для переменной будет выведено не ее имя, а ее числовое значение. Кроме этого, после оператора вывода добавлен еще один "пустой" оператор ввода `readln`. Этот оператор приостанавливает окончание работы программы до тех пор, пока не будет нажата клавиша <Enter>, т. е. у пользователя появляется возможность сразу ознакомиться с результатами выполнения программы, которые теперь не исчезают с экрана. Не нужно будет после запуска программы на выполнение специально открывать меню **Debug** и пользоваться командами **User screen** или **Output**, чтобы увидеть, что же получилось в итоге работы программы.

В описанной ранее программе использовалась только одна арифметическая операция — сложение. Но в языке Паскаль, естественно, используются и другие арифметические операции (табл. 3.1).

Таблица 3.1. *Арифметические операции в Паскале*

Операция	Описание
+	Сложение
−	Вычитание
*	Умножение
/	Деление
$x^n = \exp(n * \ln(x))$	Возведение в степень
div	Целочисленное деление
mod	Нахождение остатка от целочисленного деления

Первые четыре операции выполняются так же, как и в обычной арифметике. Единственное, на что следует обратить внимание: знаки умножения и деления отличаются по написанию (вместо принятых в математике знаков × или · используется *, вместо : используется /). Кроме того, знак умножения между сомножителями нельзя опускать, как это часто делается в математике. Запись ab, которая в математике может обозначать произведение двух чисел a и b, компилятором языка Паскаль будет воспринята как имя переменной, состоящее из двух букв, что приведет к ошибке в работе программы. Заметим, что в языке Паскаль нет стандартной функции возведения числа в степень, поэтому для этой операции пользуются формулой, приведенной в табл. 3.1. Однако в программе выражения с возведением во вторую, третью, четвертую и пятую степени чаще всего записывают в виде умножения самих на себя, столько раз, каков показатель степени.Что же касается целочисленного деления, то оно выполняется как обычное, но затем отбрасывается дробная часть получившегося при делении числа. Операции div и mod выполняются только над переменными и константами целого типа.

Пример использования действий div и mod:

- значение выражения 91 div 8 равно 11;
- значение выражения 91 mod 8 равно 3.

При вычислении значения арифметического выражения вначале выполняются действия, обладающие более высоким *приоритетом*. Слово "приоритет" в переводе с латыни означает первенство и, применительно к программированию, имеется в виду первенство при выполнении математических операций. Умножение, деление, целочисленное деление и нахождение остатка как раз и относятся к действиям с высоким приоритетом. Сложение и вычитание имеют более низкий приоритет. Если в выражении имеются действия с одинаковым приоритетом, то они выполняются слева направо по ходу выражения.

Если необходимо изменить порядок действий в выражении, то следует использовать круглые скобки. Действия, заключенные в скобки, обладают наиболее высоким приоритетом, т. е. выполняются в первую очередь.

Для лучшего понимания того, как действуют приоритеты, приведем следующий пример. Пусть в программе требуется вычислить значение арифметического выражения, записанного таким образом:

```
5 + 3*(7+2) - 40 div 4
```

Прежде всего, вычисляется выражение, находящееся в скобках — его значение равно 9. Затем слева направо выполняются следующие действия: умножение выражения в скобках на 3 (получаем 27) и целочисленное деление (получаем 10). На завершающем этапе складываем 5 и 27 и вычитаем из суммы 10. В итоге получаем конечный результат — 22.

Говоря об арифметических выражениях, необходимо отметить также следующее: хорошо знакомый нам оператор `writeln` может выводить не только тексты и значения переменных, но и значения арифметических выражений, в которые могут входить переменные и константы. Эти выражения в операторе `writeln` записываются подобно именам переменных в скобках, но без кавычек. Компьютер сперва подсчитает значение выражения, а затем покажет получившийся результат на экране. Например, если мы в какой-либо программе присвоим переменным `x` и `y` значения 20 и 30, а затем оператором `writeln` выведем на экран компьютера выражение `x+y`:

```
x:=20;
y:=30;
writeln(x+y);
```

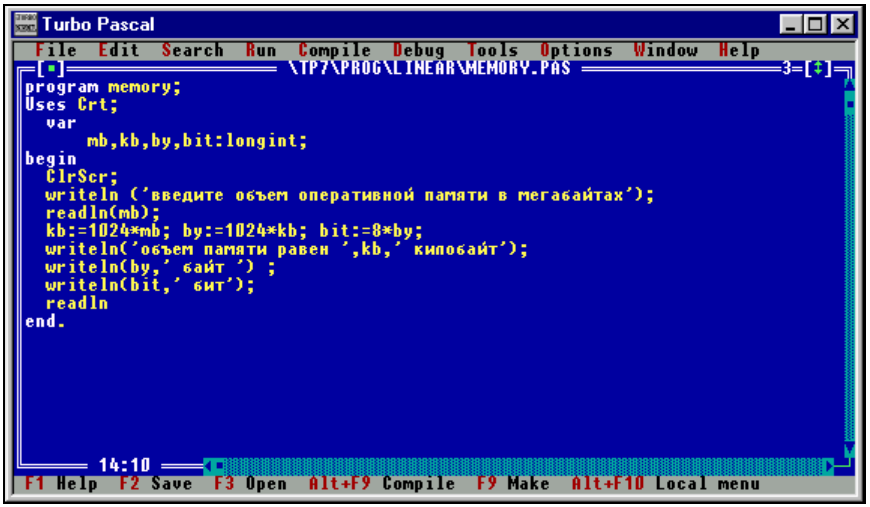
то в результате выполнения этого фрагмента программы компьютер сперва сложит значения двух переменных, а затем выведет получившуюся сумму на экран компьютера, т. е. в данном случае на экране мы увидим число 50.

До сих пор мы работали со сравнительно небольшими числами. Теперь составим программу, которая оперирует более масштабными величинами. Эта программа пересчитывает объем оперативной памяти компьютера, выраженный в мегабайтах, в килобайтах, байтах и битах (рис. 3.10). Между этими единицами измерения информации существуют следующие соотношения:

- 1 мегабайт = 1024 килобайта;
- 1 килобайт = 1024 байта;
- 1 байт = 8 бит.

В программе использованы 4 переменные: `mb` — для объема в мегабайтах, `kb` — для килобайт, `by` — для байт, `bit` — для бит. При составлении данной программы можно столкнуться со следующей проблемой: результаты работы

программы будут выражены достаточно большими числами, которые нельзя обработать с помощью переменных типа `integer`. Этот тип используется для представления чисел в диапазоне от –32 768 до 32 767. Поэтому для описания переменных используем другой целочисленный тип данных — `longint`, с помощью которого можно работать с числами в диапазоне от –2 147 483 648 до 2 147 483 647. В листинге 3.6 приводится текст программы.



а

б

Рис. 3.10. Программа, подсчитывающая объем оперативной памяти компьютера *(а)*, и результат ее работы *(б)*

Листинг 3.6. Программа с использованием больших целых чисел

```
program memory;
Uses Crt;
  var
     mb,kb,by,bit:longint;
```

```
begin
  ClrScr;
  writeln ('введите объем оперативной памяти в мегабайтах');
  readln(mb);
  kb:=1024*mb;
  by:=1024*kb;
  bit:=8*by;
  writeln('объем памяти равен ',kb,' килобайт');
  writeln(by,' байт ');
  writeln(bit,' бит');
  readln
end.
```

Исходная информация, необходимая для вычислений — объем памяти в мегабайтах — вводится с помощью оператора `readln`, как и в предыдущей программе, а все последующие результаты вычисляются в соответствии с приведенными соотношениями и затем выводятся на экран компьютера операторами `writeln`. В данной программе список вывода каждого из операторов `writeln` содержит не один, а два или три элемента. Каждый из этих элементов обрабатывается в соответствии с его видом. Текст, заключенный в апострофы, выводится так, как он есть, без изменений, а для переменных выводятся их текущие значения.

Для решения ряда задач, в которых используются только целые числа, удобно бывает использовать действия `div` и `mod`. В качестве примера составим программу, обслуживающую работу банкомата (листинг 3.7, рис. 3.11). Банкомат должен выдавать клиенту требуемую сумму, но при этом расходовать наименьшее количество купюр. Желательно также, чтобы на экране банкомата выводилось сообщение о том, какими именно купюрами и в каком количестве выдавалась запрошенная клиентом сумма. В банкомате имеются купюры по 1000, 500, 100 и 50 рублей. Тогда алгоритм решения данной задачи сводится к следующему. Вначале программа предлагает клиенту ввести требуемую сумму. (Так как банкомат не содержит мелочи, то попросим пользователя ввести сумму, кратную 50.) Затем эта сумма делится на 1000. Результат целочисленного деления и будет минимальным количеством банкнот по 1000 рублей. Остаток от деления разделим на 500 и получим соответственно минимальное требуемое количество банкнот по 500 рублей. Аналогичным образом найдем количество банкнот по 100 рублей. Наконец, получившийся остаток при делении на 100 разделим на 50 и получим количество банкнот по 50 рублей.

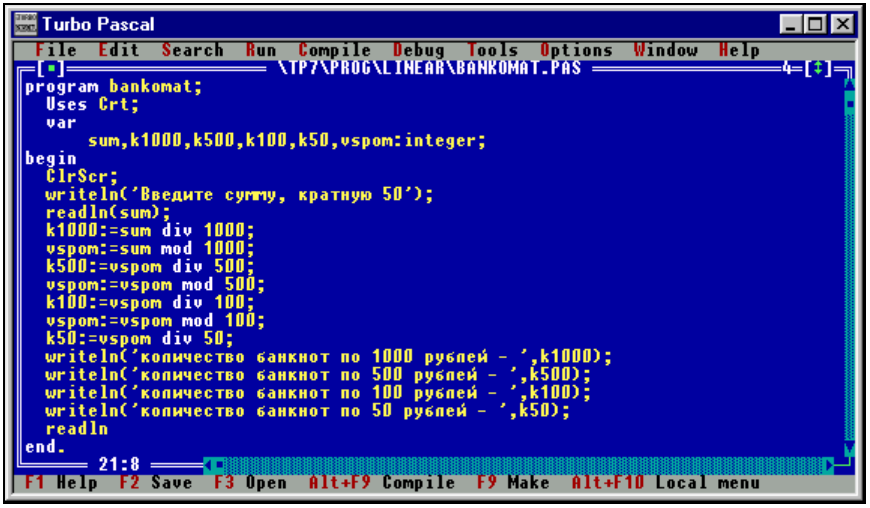
Если, например, нам нужно определить, каким минимальным количеством купюр можно выдать сумму 4750 рублей, то сперва делим эту сумму на 1000 и находим остаток от деления:

4750 `div` 1000 = 4; 4750 `mod` 1000 = 750

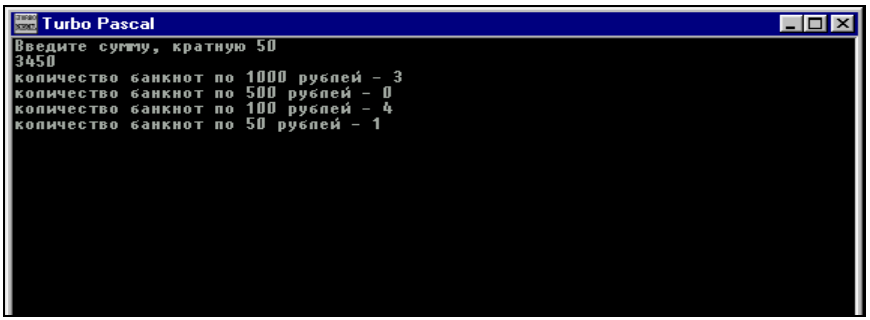
теперь нам известно, что минимальное количество купюр по 1000 рублей равно 4, а оставшаяся сумма составляет 750 рублей. Эту сумму делим уже на 500:

750 `div` 500 = 1; 750 `mod` 500 = 250

следовательно, нам понадобится как минимум 1 купюра по 500 рублей, а сумма, которую нужно будет выдать 50- и 100-рублевыми купюрами, составит 250 рублей. Используя аналогичные вычисления, мы получим, что эту сумму можно выдать одной 50-рублевой и двумя 100-рублевыми купюрами.



а



б

Рис. 3.11. Программа "Банкомат" *(а)* и результат ее работы *(б)*

Листинг 3.7. Программа "Банкомат"

```
program bankomat;
  Uses Crt;
```

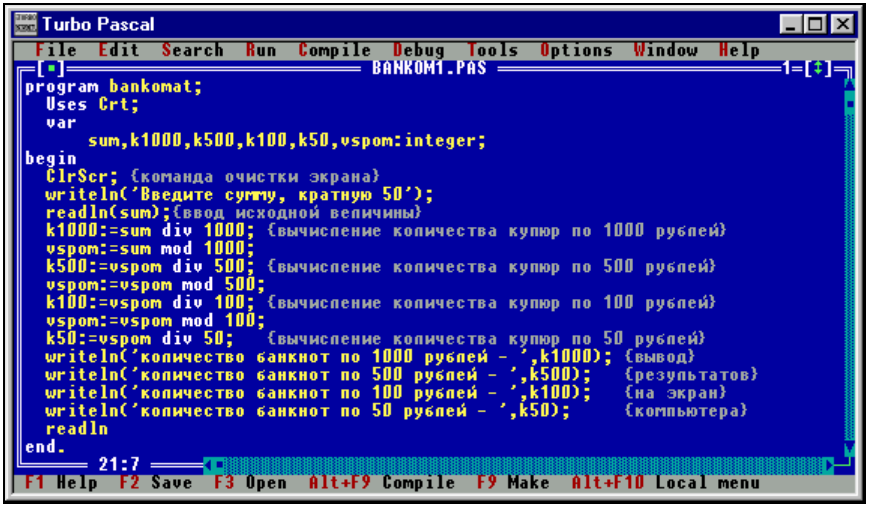
```
  var
      sum,k1000,k500,k100,k50,vspom:integer;
begin
  ClrScr;
  writeln('Введите сумму, кратную 50');
  readln(sum);
  k1000:=sum div 1000;
  vspom:=sum mod 1000;
  k500:=vspom div 500;
  vspom:=vspom mod 500;
  k100:=vspom div 100;
  vspom:=vspom mod 100;
  k50:=vspom div 50;
  writeln('количество банкнот по 1000 рублей - ',k1000);
  writeln('количество банкнот по 500 рублей - ',k500);
  writeln('количество банкнот по 100 рублей - ',k100);
  writeln('количество банкнот по 50 рублей - ',k50);
  readln
end.
```

При составлении программы используются следующие переменные: `sum` — вводимая клиентом сумма, `k1000` — количество купюр по 1000 рублей; `k500` — количество купюр по 500 рублей, `k100` — количество купюр по 100 рублей, `k50` — количество купюр по 50 рублей, `vspom` — вспомогательная переменная, в которой содержится остаток, получающийся при делении. Программа с помощью описанных ранее операций находит значения переменных `k1000`, `k500`, `k100` и `k50`, а затем выводит их на экран компьютера. При этом переменная `vspom` в процессе выполнения программы несколько раз меняет значение. В начале ей присваивается значение остатка от деления на 1000, затем на 500 и, наконец, на 100.

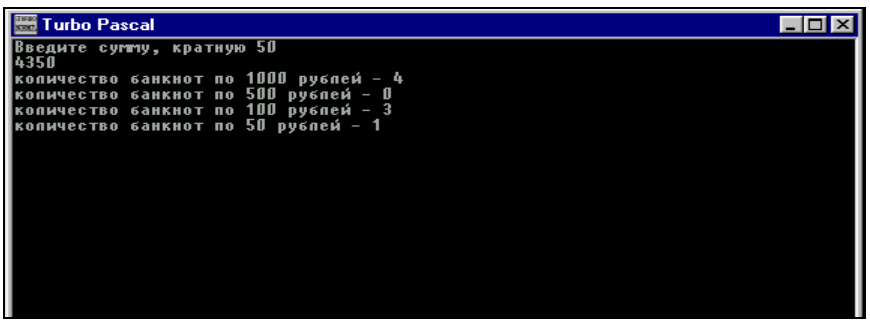
Для лучшего понимания данной программы снабдим ее комментариями. Мы уже упоминали о том, что комментарии представляют собой пояснения к тексту программы, а теперь рассмотрим их использование более подробно. Это необходимо сделать ввиду того, что комментарии являются неотъемлемой частью любой достаточно большой и сложной программы. При их отсутствии в такой программе бывает непросто разобраться даже опытному программисту.

Правила использования комментариев в программе достаточно либеральны. Они могут располагаться в любом месте программы. Комментарии могут находиться на той же строке, что и основной текст программы, либо занимать отдельную строку или несколько строчек в программе. Вне зависимости от места и способа расположения комментариев при запуске программы на выполнение компилятор игнорирует их, что показывает рис. 3.12. Хотя в про-

грамму "Банкомат" добавлено определенное количество комментариев, результат работы программы при использовании тех же исходных данных, что и ранее, остается неизменным. Комментарии таким образом можно добавлять в программу там, где это удобно программисту, но при этом нужно помнить о том, что любой комментарий должен быть обязательно ограничен фигурными скобками или символами (* и *) с обеих сторон. При просмотре текста ранее составленной программы комментарии легко определить визуально, т. к. в отличие от основного текста программы они выделены серым цветом. В листинге 3.8 приведен текст программы, дополненный комментариями.



а



б

Рис. 3.12. Программа "Банкомат" с комментариями *(а)* и результат ее работы *(б)*

Листинг 3.8. Программа "Банкомат" с комментариями

```
program bankomat;
  Uses Crt;
```

```
  var
      sum,k1000,k500,k100,k50,vspom:integer;
begin
  ClrScr; {команда очистки экрана}
  writeln('Введите сумму, кратную 50');
  readln(sum);{ввод исходной величины}
  k1000:=sum div 1000; {вычисление количества купюр по 1000 рублей}
  vspom:=sum mod 1000;
  k500:=vspom div 500; {вычисление количества купюр по 500 рублей}
  vspom:=vspom mod 500;
  k100:=vspom div 100; {вычисление количества купюр по 100 рублей}
  vspom:=vspom mod 100;
  k50:=vspom div 50;   {вычисление количества купюр по 50 рублей}
  writeln('количество банкнот по 1000 рублей - ',k1000); {вывод}
  writeln('количество банкнот по 500 рублей - ',k500);   {результатов}
  writeln('количество банкнот по 100 рублей - ',k100);   {на экран}
  writeln('количество банкнот по 50 рублей - ',k50);     {компьютера}
  readln
end.
```

3.4. Работа с вещественными величинами

Естественно, что задачи, решаемые с помощью Turbo Pascal, не ограничиваются такими, в которых при расчетах используются только целые числа. Для того чтобы можно было использовать переменные, которые могут принимать вещественные значения (т. е. значения в виде десятичных дробей), следует в разделе описания переменных описать их с помощью типа `real` (вещественный).

Вещественные числа в Паскале можно записывать двумя способами — с фиксированной точкой и с плавающей точкой. Запись *с фиксированной точкой* похожа на обычную запись десятичной дроби в математике, только целая часть от дробной отделяется не запятой, а точкой. Например, число 10,23 с фиксированной точкой записывается как 10.23.

Другим способом представления вещественных чисел является запись *с плавающей точкой*. В этом случае число представляется как произведение двух сомножителей $m \cdot 10^n$, где `m` — мантисса числа, а `n` — порядок числа. Мантисса — это целое число или вещественное число с фиксированной точкой, которое может принимать значение в диапазоне от 1 до 10. Показатель может быть только целым числом. Именно в таком виде по умолчанию выводятся на экран компьютера вещественные числа, причем запись с плавающей точкой будет выглядеть следующим образом — `mEn`, где `E` заменяет собой основание

степени — число 10. Перед показателем указывается его знак. Число 125.73 (далее в пособии мы будем использовать именно такую запись десятичных дробей) с плавающей точкой будет записано как 1.2573000000E+02, т. е. $1.2573 \cdot 10^2$, а число 0.00441 будет записано как 4.4100000000E–03, т. е. $4.41 \cdot 10^{-3}$. Представление чисел с плавающей точкой, как правило, используется для записи либо очень больших, либо очень маленьких чисел, т. к. и в том и в другом случае при записи числа в обычном виде оно будет воспроизводиться с большим количеством нулей, что затрудняет восприятие данного числа. Но для представления результатов в большинстве задач все же более наглядным и удобным является вид числа с фиксированной точкой.

В качестве примера использования вещественных переменных рассмотрим задачу вычисления объема параллелепипеда, где его высота, длина и ширина — вещественные числа (рис. 3.13). В программе, написанной для решения данной задачи, используются четыре переменные. Все они описаны как вещественные. Это `l` — длина параллелепипеда, `w` — его ширина, `h` — высота и `v` — искомый объем. Исходные данные вводятся с помощью операторов `readln`. Затем вычисляется произведение этих трех величин и присваивается переменной `w`, значение которой выводится оператором `writeln`. При вводе в компьютер исходных данных не забываем отделять целую часть числа от дробной точкой. В листинге 3.9 приводится текст данной программы.

Листинг 3.9. Вычисление объема параллелепипеда

```
program paral;
Uses Crt;
var l,w,h,v: real;
begin
ClrScr;
writeln('Введите длину параллелепипеда');
readln (l);
writeln('Введите ширину параллелепипеда');
readln (w);
writeln('Введите высоту параллелепипеда');
readln (h);
v:=l*w*h;
writeln ('Объем параллелепипеда равен ',v);
readln
end.
```

При длине параллелепипеда, равной 4.2, ширине — 2.5 и высоте — 1.8, получим следующий результат: 1.8900000000E+01.

Как и следовало ожидать, результат выведен с плавающей точкой. Хотелось бы видеть результат вычислений в более привычной нам форме. На этот случай в языке Паскаль предусмотрена возможность *форматированного вывода*, т. е. программист может сам определять внешний вид выводимого значения вещественной переменной. Для этого в операторе вывода после имени переменной нужно поставить двоеточие. После двоеточия указывается общее количество символов, отводимое под значение данной переменной, далее ставится еще одно двоеточие, а затем указывается количество цифр в дробной части числа. Не нужно забывать при этом, что одну позицию нужно оставить для точки. Например, если мы хотим, чтобы в программе выводимое на экран значение некоторой переменной `t`, относящейся к вещественному типу, занимало 8 позиций, из которых 3 отводится под дробную часть, 1 — под десятичную точку, а 4 — под целую часть, то мы запишем наши пожелания в операторе вывода следующим образом:

```
writeln(t:8:3);
```

Следует заметить, что если фактически в вещественном числе (значении форматируемой переменной) содержится больше цифр после точки, чем указано в операторе вывода после второго двоеточия, то "лишние" цифры отбрасываются, а если число цифр после точки меньше, чем указанное, то недостающие цифры заменяются нулями.

Исходя из сказанного ранее, в нашей программе оператор вывода можно изменить следующим образом:

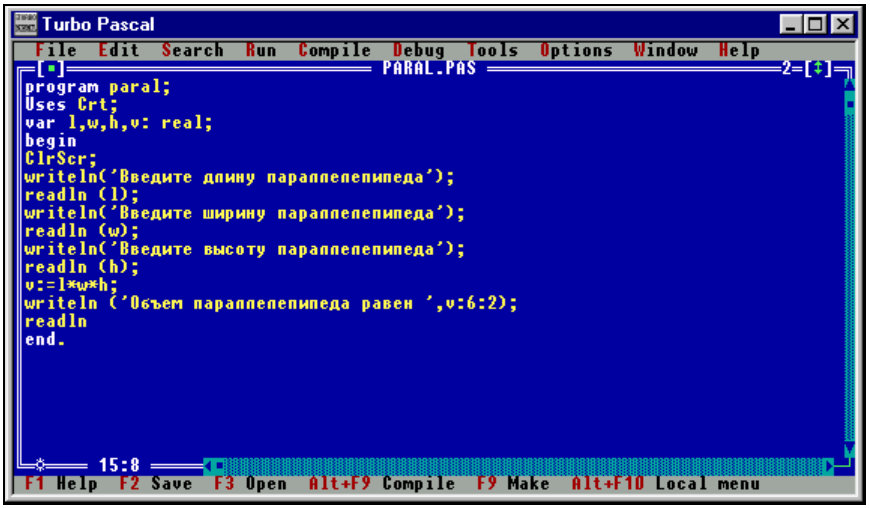
```
writeln ('Объем параллелепипеда равен', v:6:2)
```

т. е. под значение переменной будет использовано 6 символов, в том числе 3 символа для целой части, 1 — для точки и 2 — для дробной части. Тогда при тех же исходных данных результат будет выведен в следующем виде:

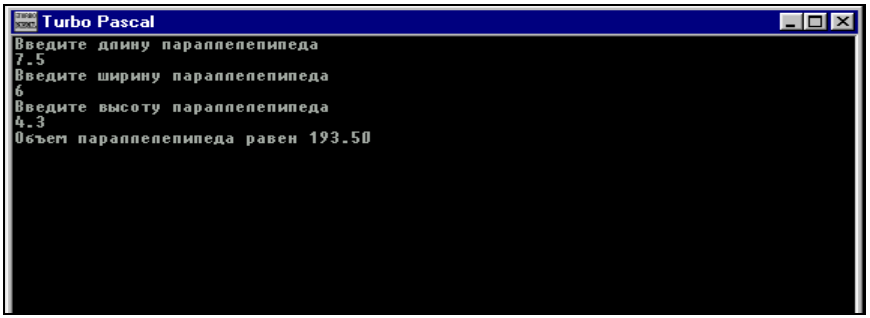
```
Объем параллелепипеда равен 18.90
```

т. е. результат будет представлен в виде, удобном для восприятия человеком (рис. 3.13).

В программе из листинга 3.9 мы использовали только переменные вещественного типа, но в дальнейшем нам придется составлять программы, в которых будут использоваться переменные как вещественного типа, так и целого, причем эти переменные будут взаимодействовать друг с другом. Взаимодействие это может проявляться и в том, что в программах будут встречаться математические выражения, в которых имеются и целые, и вещественные переменные и константы. Естественно, что возникает вопрос: к какому типу будет относиться результат вычислений с участием величин разного типа. Это важно еще и потому, что если мы, например, присвоим целой переменной вещественное значение, то при компиляции программы будет выдано



а



б

Рис. 3.13. Программа вычисления объема параллелепипеда *(а)* и результат ее работы *(б)*

сообщение о несоответствии используемых типов. На поставленный вопрос отвечает специальный набор правил языка Паскаль.

- Если над двумя величинами целого типа производятся операции сложения, вычитания или умножения, то результатом данной операции будет величина целого типа.
- Если операции сложения, вычитания или умножения производятся над двумя вещественными величинами, то результатом операции будет вещественная величина.
- Если операции сложения, вычитания или умножения производятся над двумя величинами, одна из которых является целой, а другая вещественной, то результатом будет вещественная величина.
- Если над любыми числовыми величинами (целочисленными или вещественными) производится операция деления (не целочисленного), то ее результатом будет вещественная величина.

На последнее правило следует обратить особое внимание, т. к. даже если в операции деления участвуют две целые величины, то результатом их будет величина вещественная. Проиллюстрируем это следующим простым примером. Пусть нам нужно найти частное от деления двух целых чисел — 5 разделить на 4. Результатом будет число 1.25, т. е. величина явно вещественная. И это понятно. Но если 25 разделить на 5, то получится также вещественное число 5.0 (а не просто 5, как могло бы показаться). Поэтому если, к примеру, в программе у нас имеются некие две переменные `k` и `l`, причем обе они относятся к целому типу, но в программе может встретиться операция деления одной переменной на другую, то для хранения результата этой операции следует использовать переменную `r`, которая должна быть описана как вещественная.

Однако в Паскале присутствует такое явление, как совместимость типов. В соответствии с ним переменным одного типа можно присваивать значения, относящиеся к другому типу. При этом должно соблюдаться следующее правило: тип присваиваемого значения должен являться подмножеством возможных значений переменной. Например, переменной вещественного типа можно присвоить целочисленное значение, поскольку множество целых чисел является подмножеством вещественных чисел. Но целочисленной переменной нельзя присвоить вещественное значение, т. к. множество вещественных чисел не является подмножеством целых чисел.

В Паскале имеются специальные средства (функции), позволяющие преобразовывать одни типы данных в другие: `trunc`, `round`, `chr` и `ord`. Эти функции будут подробно рассмотрены в следующих главах книги.

Перед тем как завершить данную главу, отметим еще одно обстоятельство, связанное с работой в среде программирования Turbo Pascal. Часто у пользователя, работающего в системе, возникает необходимость не только записать на диск созданный им файл, но и открыть один из ранее созданных. Такое желание может быть обусловлено двумя причинами.

Во-первых, пользователь может использовать текст написанных ранее им или кем-либо другим программ в качестве примера и наглядного пособия для создания новых программ. Во-вторых, пользователь может неоднократно обращаться к одной из уже записанных на жесткий диск программ в целях исправления имеющихся в ней ошибок, улучшения внешнего вида и надежности работы программы.

Для выполнения указанных операций используется команда **Open** из меню **File**. После выполнения этой команды открывается диалоговое окно, которое по своей структуре сходно с тем окном, которое используется для сохранения файла на жестком диске по команде **Save as**. Оба окна состоят из одних и тех же элементов, аналогичным образом производится навигация (перемещение

по файловой структуре) в обоих окнах. После обнаружения и выделения искомого файла его открытие производится щелчком на экранной кнопке **Open** (рис. 3.14).

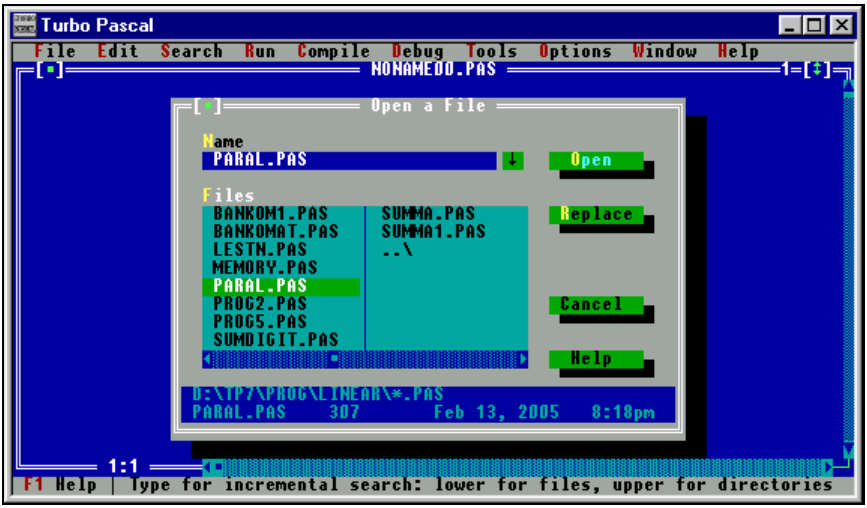


Рис. 3.14. Диалоговое окно открытия ранее созданного файла

Задания для самостоятельной работы

Задание 1. Составить программу, которая бы выводила на экран компьютера на светло-сером фоне фиолетовыми буквами в пять строк следующий текст:

```
Существуют следующие основные виды программного обеспечения
персональных компьютеров:
1) Системные программы
2) Прикладные программы
3) Инструментальные системы или системы программирования
```

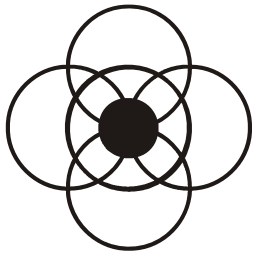
Задание 2. Составить программу, которая бы выводила на экран компьютера треугольник зеленого цвета, составленный из символов * (звездочка). Вершину треугольника должна составлять одна звездочка, на следующей строке должно быть три символа *, на третьей — пять и т. д. вплоть до основания треугольника, составленного из пятнадцати звездочек.

Задание 3. Составить программу, которая бы определяла разницу между двумя моментами времени, выраженными в часах и минутах. Первый момент времени предшествует второму. Сама разница также должна быть выражена

в часах и минутах. Величина как первого, так и второго моментов времени должна быть введена в компьютер с клавиатуры.

Задание 4. Составить программу, которая бы по известному радиусу окружности вычисляла бы длину окружности и площадь круга. Величина радиуса вводится в компьютер с клавиатуры.

Задание 5. Покупатель приобрел на рынке `a` килограммов картофеля, `b` килограммов огурцов и `c` килограммов помидоров. Составить программу, которая бы позволила подсчитать общую сумму покупки. Цена 1 килограмма каждого из приобретенных овощей (в рублях и копейках) и приобретенное их количество (в килограммах) для каждого вида овощей вводятся в компьютер с клавиатуры.



ГЛАВА 4

Ветвление в Паскале

В жизни человеку часто приходится сталкиваться с ситуациями, когда в зависимости от складывающейся обстановки ему нужно выбирать тот или иной способ действия. При решении поставленной перед программистом задачи также нередко возникает ситуация, когда нужно выбирать один из двух или нескольких возможных вариантов дальнейшего хода программы. В этом случае в программе используется структура, называемая *ветвлением*. В отличие от программ линейной структуры, в которых все операторы выполняются последовательно друг за другом, в программах с ветвлением в зависимости от значений вводимых исходных данных может работать одна из двух или нескольких ветвей, вариантов программы.

Такая структура реализуется в программе посредством условных операторов. В заголовке условного оператора содержится некоторое условие, выполнение или невыполнение которого и определяет дальнейшую работу программы. Так как программист заранее не знает, какой из вариантов будет выполняться, выбор должен производить сам компьютер. В качестве условий, истинность или ложность которых проверяется, чаще всего используются знакомые читателю по школьному курсу математики операции сравнения различных величин. В языке Паскаль используется 6 таких операций:

- `<` — меньше;
- `>` — больше;
- `=` — равно;
- `<=` — меньше или равно;
- `>=` — больше или равно;
- `<>` — не равно.

Написание последних 3 операций отличается от принятого в математике, где соответственно используются обозначения $\leq$, $\geq$ и $\neq$, но т. к. клавиш с по-

добными обозначениями нет на клавиатуре компьютера, то используются указанные ранее сочетания из двух символов.

Начнем мы рассмотрение условных операторов с более простого случая, когда в случае истинности проверяемого соотношения выполняется один вариант действий, а в случае ложности другой вариант. Такой выбор дальнейшего хода действий реализуется с помощью условного оператора `if`.

4.1. Условный оператор *if*

Общий вид оператора `if` следующий:

```
if условие
then вариант1
else вариант2;
```

служебные слова `if`, `then` и `else` в переводе с английского означают соответственно если, то, иначе. Оператор действует следующим образом: сначала проверяется, выполняется ли условие, находящееся после слова `if`. Если это условие выполняется (иначе говоря, является истинным), то осуществляется первый вариант действий, в противном случае, если условие не выполняется (является ложным) — второй, записанный после служебного слова `else`. В самом простом случае действие, осуществляемое в каждом из вариантов, состоит из одного оператора.

Классическим примером использования оператора `if` является программа, определяющая наибольшее из двух введенных чисел. Для наглядности решения данной задачи запишем предварительно алгоритм ее решения в виде графической схемы (рис. 4.1), а затем реализуем его в виде компьютерной программы.

Первоначально вводятся два числа `a` и `b` (ввод записан в верхнем параллелограмме), затем проверяется истинность условия `a>b` (проверка записана в ромбе). Если данное условие истинно, то на экран компьютера выводится число `a`, которое является наибольшим (выполняется операция, указанная в левом нижнем параллелограмме), если же условие ложно, то на экран выводится являющееся наибольшим число `b` (осуществляется операция, записанная в правом нижнем параллелограмме).

Теперь от графической схемы перейдем к программе, изображенной на рис. 4.2. В этой программе с помощью уже знакомых нам операторов `readln` с клавиатуры вводятся значения переменных `a` и `b`. Затем с помощью условного оператора `if` производится проверка условия `a>b` и выполняется либо оператор вывода `writeln`, записанный после служебного слова `then`, либо тот оператор вывода, который находится после служебного слова `else`. Обратите

внимание на то, что хотя конструкция `if...then...else` расположена на трех строчках, между этими строчками не ставится точка с запятой, т. к. вся эта конструкция представляет собой единый условный оператор, который в качестве своих элементов может включать в себя другие операторы (в данном случае операторы вывода). В листинге 4.1 приводится текст программы.

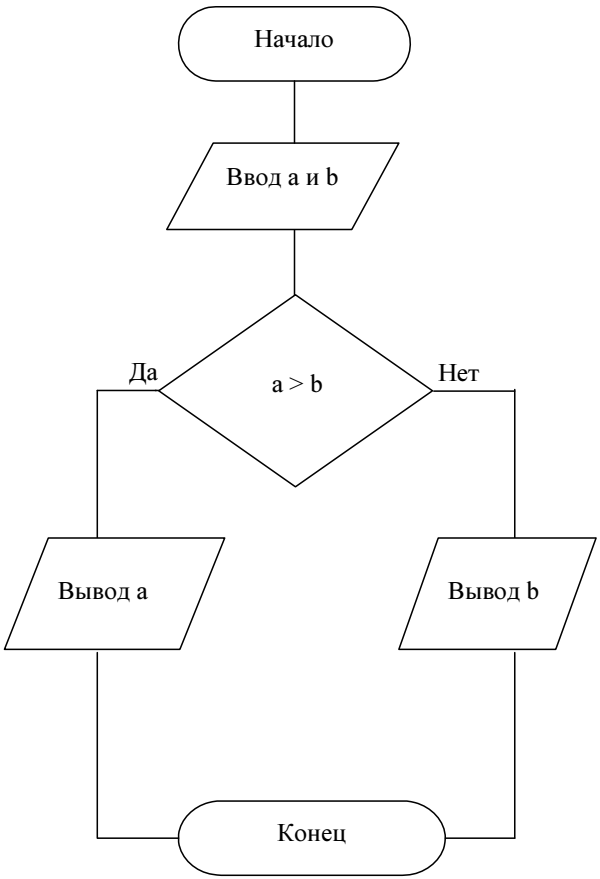
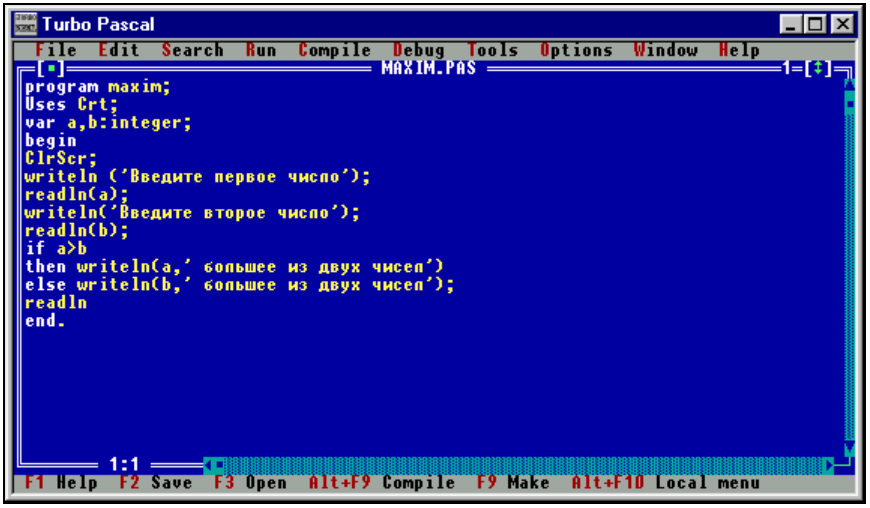


Рис. 4.1. Графическая схема определения наибольшего из двух чисел

Листинг 4.1. Определение наибольшего из двух чисел

```
program maxim;
Uses Crt;
var a,b:integer;
begin
 ClrScr;
```

```
 writeln ('Введите первое число');
 readln(a);
 writeln('Введите второе число');
 readln(b);
 if a>b
 then writeln(a,' большее из двух чисел')
 else writeln(b,' большее из двух чисел');
readln
end.
```



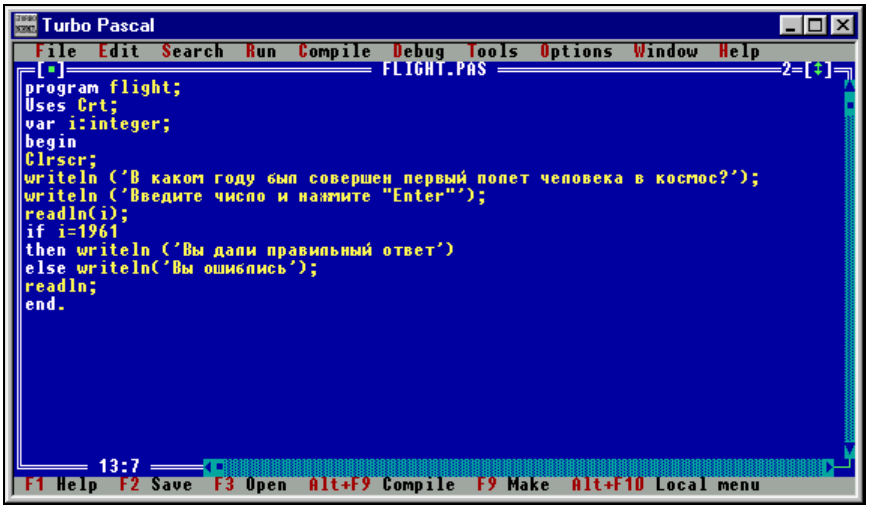
а

б

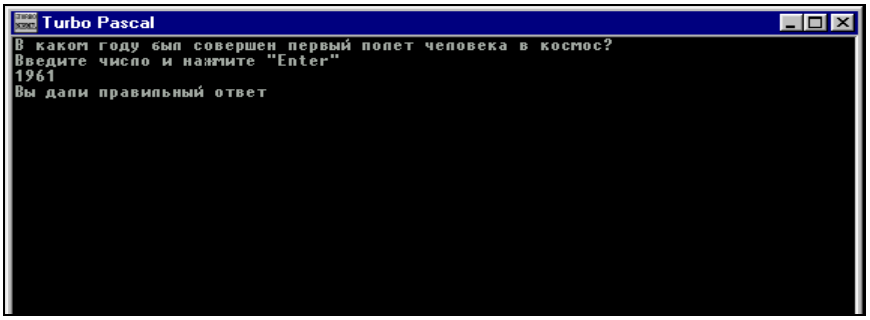
Рис. 4.2. Программа определения большего из двух чисел *(а)* и результат ее работы *(б)*

Условный оператор `if` удобно использовать для составления различных программ-тестов, проверяющих знания пользователя. Например, программа, приведенная на рис. 4.3 проверяет, знает ли пользователь дату первого полета человека в космос. В случае правильного ответа на экран выводится сообщение "Вы дали правильный ответ", в противном случае сообщение "Вы ошиб-

лись". В этой программе с помощью оператора `writeln` пользователю задается вопрос, ответ на который он должен дать, введя дату с клавиатуры. Введенное пользователем значение присваивается переменной `i`. Затем значение этой переменной сравнивается с правильным ответом — числом 1961. Если равенство выполняется, то оператором `writeln` выводится на экран сообщение о правильном ответе, если значение `i` не совпадает с правильной датой, выводится сообщение об ошибке. В листинге 4.2 приводится текст данной программы.



а



б

Рис. 4.3. Программа-тест *(а)* и результат ее работы *(б)*

Листинг 4.2. Пример простейшего теста

```
program flight;
  Uses Crt;
  var i:integer;
```

```
begin
  Clrscr;
  writeln ('В каком году был совершен первый полет человека в космос?');
  writeln ('Введите число и нажмите "Enter"');
  readln(i);
  if i=1961
  then writeln ('Вы дали правильный ответ')
  else writeln('Вы ошиблись');
  readln;
end.
```

Тот вид условного оператора, который был описан нами ранее, представляет собой полную форму условного оператора, но такая форма его записи не является единственно возможной. Наряду с ней в языке Паскаль используется и сокращенный условный оператор. Такой оператор имеет следующий общий вид:

```
if условие then действие;
```

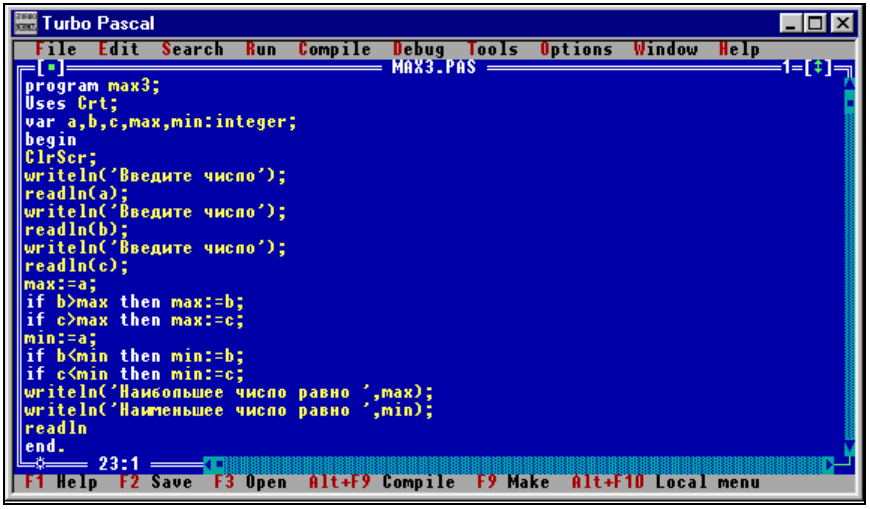
Сокращенный условный оператор работает следующим образом. Если условие, содержащееся после служебного слова `if`, истинно, то выполняется действие, записанное после `then`, а если условие ложно, то в условном операторе не выполняется никаких действий, и программа переходит к выполнению следующего оператора, расположенного вслед за данным условным оператором.

Как работают сокращенные условные операторы, разберем на примере следующей задачи: нам нужно отыскать наибольшее и наименьшее число на сей раз среди трех введенных с клавиатуры чисел. Исходный текст программы, которая решает данную задачу, приводится в листинге 4.3.

Листинг 4.3. Определение наибольшего и наименьшего числа из трех введенных чисел

```
program max3;
Uses Crt;
var a,b,c,max,min:integer;
begin
 ClrScr;
 writeln('Введите число');
 readln(a);
 writeln('Введите число');
 readln(b);
 writeln('Введите число');
```

```
 readln(c);
 max:=a;
   if b>max then max:=b;
   if c>max then max:=c;
 min:=a;
  if b<min then min:=b;
  if c<min then min:=c;
 writeln(' Наибольшее число',max);
 writeln(' Наименьшее число ',min);
 readln
end.
```



а

б

Рис. 4.4. Программа определения наибольшего и наименьшего числа из трех чисел *(а)* и результат ее работы *(б)*

В данной программе (рис. 4.4) операторами `readln` производится ввод исходных данных — трех чисел, которые присваиваются переменным `a`, `b` и `c`. Кроме переменных, содержащих сравниваемые числа, в программе используется еще одна переменная `max`. В начале переменной `max` присваивается значение, представляющее собой первое число, содержащееся в переменной `a`.

Затем переменная max сравнивается с переменной b с помощью сокращенного условного оператора if. Если значение b больше, чем значение max, то переменной max присваивается новое значение, равное b. Если же значение b меньше или равно значению max, то никаких действий сокращенный условный оператор больше не производит, и программа переходит к выполнению следующего оператора. В любом случае в итоге действия условного оператора мы в переменной max имеем наибольшее из двух чисел a и b.

Следующий оператор в программе также относится к числу сокращенных условных операторов. Этот оператор сравнивает имеющееся в переменной max наибольшее из двух чисел с третьим, содержащимся в переменной c. Если значение c больше чем max, то значение max изменяется: ей присваивается значение c. Если c меньше или равно max, то max остается без изменений. В итоге последовательного действия двух сокращенных условных операторов мы получим в переменной max искомую величину, которая является наибольшей из трех чисел. Остается только вывести значение переменной max оператором writeln на экран компьютера — это и будет решение поставленной задачи.

Аналогичные действия производятся для нахождения наименьшего значения (min).

Используя сокращенные условные операторы, мы усовершенствуем программу-тест, приведенную на рис. 4.3. Новый, усовершенствованный вариант этой программы позволит превратить тест в настоящий виртуальный экзамен на знание истории освоения космоса (рис. 4.5). В ходе сдачи экзамена пользователь должен ответить на 3 вопроса, которые последовательно появляются на экране компьютера. Программа будет не только информировать пользователя о правильности или неправильности ответа на очередной вопрос, но и выставит за экзамен итоговую оценку. Экзаменуемый, правильно ответивший на все вопросы, получает за экзамен 5 баллов. За каждый неправильный ответ оценка снижается на 1 балл. Поэтому минимальная оценка за экзамен (при неправильном ответе на все 3 вопроса) составит двойку.

Программу мы построим теперь следующим образом. Оценка экзаменуемого будет находиться в переменной n. В начале программы присвоим этой переменной значение 5. В дальнейшем значение этой переменной будет либо оставаться неизменным в случае правильных ответов, либо изменяться в сторону уменьшения при неправильных ответах. Каждый вопрос экзамена выводится на экран компьютера оператором writeln. Пользователь должен дать на него ответ, введя с клавиатуры спрашиваемую дату. Введенная дата оператором readln присваивается переменной i. Значение этой переменной сравнивается с правильной датой. Операция сравнения записывается в сокращенном условном операторе после слова if. Если введенный ответ не равен "эталонной" дате, записанной в условии, то значение переменной n уменьшается на

единицу, т. е. оценка снижается на 1 балл. Если же пользователь дал правильный ответ, то неравенство, записанное после `then`, выполняться не будет и значение `n`, т. е. оценка, не изменится. По завершении работы программы конечное значение `n`, т. е. итоговая оценка, будет выведено на экран компьютера. В листинге 4.4 приводится текст данной программы.

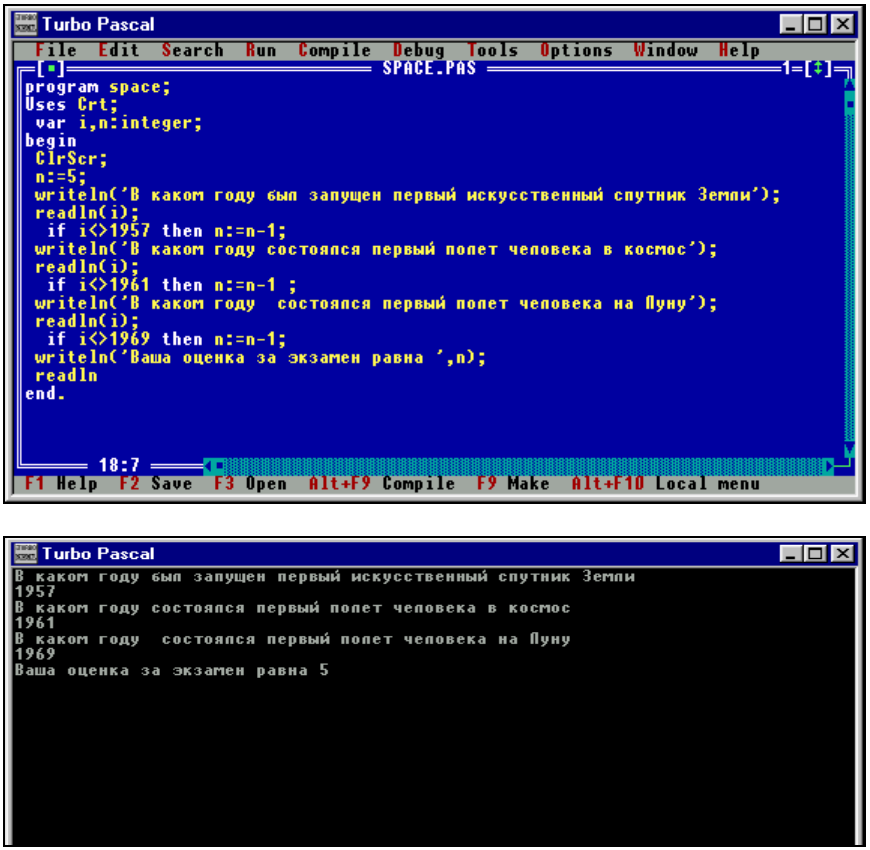


Рис. 4.5. Программа виртуального экзамена *(а)* и результат ее работы *(б)*

Листинг 4.4. Тест на знание истории освоения космоса

```
program space;
Uses Crt;
 var i,n:integer;
begin
 ClrScr;
 n:=5;
```

```
 writeln('В каком году был запущен первый искусственный спутник Земли');
 readln(i);
  if i<>1957 then n:=n-1;
 writeln('В каком году состоялся первый полет человека в космос');
 readln(i);
  if i<>1961 then n:=n-1;
 writeln('В каком году  состоялся первый полет человека на Луну');
 readln(i);
  if i<>1969 then n:=n-1;
 writeln('Ваша оценка за экзамен равна ',n);
 readln
end.
```

Во всех рассмотренных до сих пор примерах в каждой из ветвей условного оператора совершалось не более одного действия, производимого каким-либо одним оператором. Но нередко в ходе разработки программ возникает необходимость разместить в одной или обеих ветвях условного оператора не одно, а целый ряд действий. В таком случае в качестве вариантов действий, находящихся в ветвях условного оператора, используют не простые, а составные операторы, которые помещаются в программе после служебных слов `then` или `else`.

Составной оператор представляет собой группу операторов, размещенную между служебными словами `begin` и `end`. Слова `begin` и `end`, называемые в данном случае *операторными скобками*, обозначают здесь не начало и конец основной части программы, а начало и конец составного оператора. Между ними могут располагаться различные операторы: операторы ввода и вывода, присваивания, вложенные условные операторы (*вложенным* называется условный оператор, входящий в качестве составной части в другой условный оператор) и другие виды операторов. Количество простых операторов, входящих в составной, не ограничено.

Операторы, входящие в составной оператор, друг от друга отделяются точками с запятой. После слова `begin` и перед словом `end` точка с запятой не ставится. Не ставится в данном случае точка с запятой и после слова `end` (если только это слово не находится в конце всего условного оператора), т. к. `end` обозначает здесь лишь конец одного из вариантов, входящих в состав единого условного оператора. Таким образом, в общем виде структуру условного оператора с составными операторами в его ветвях можно представить следующим образом:

```
if условие then
   begin
   оператор_1; оператор_2; ... оператор_m
   end
```

```
else
   begin
   оператор_1; оператор_2; ... оператор_n
   end;
```

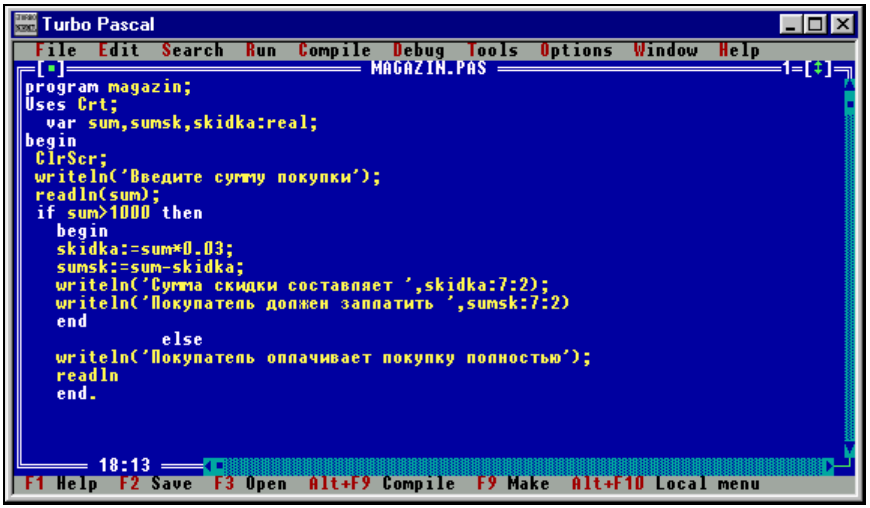
Составные операторы могут использоваться как в полных, так и в сокращенных условных операторах. Кроме условных операторов составные операторы могут использоваться и в других программных структурах, которые мы рассмотрим далее.

Составной оператор можно использовать для решения следующей задачи. Администрация одного магазина для привлечения большего числа клиентов ввела правило, согласно которому каждый покупатель, который приобрел товар на сумму более 1000 рублей, имеет право на трехпроцентную скидку со стоимости покупок. Требуется составить программу, облегчающую работу кассира данного магазина. Эта программа в случае, если стоимость покупки превышает указанную сумму, должна подсчитывать величину скидки и ту сумму, которую должен оплатить покупатель с учетом скидки. В случае же если стоимость покупок меньше 1000 рублей, программа должна выдавать сообщение о том, что покупка должна быть оплачена полностью.

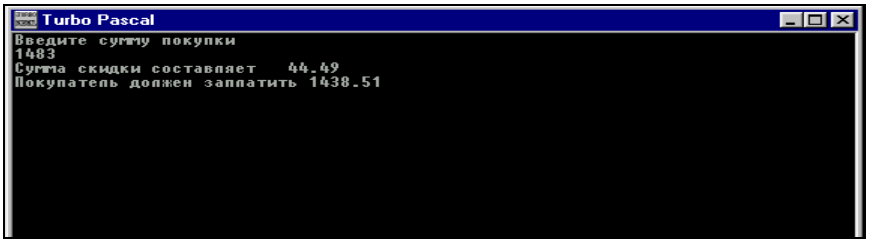
В программе, составленной для решения приведенной задачи, используются три переменных вещественного типа: `sum` — сумма покупки без учета скидок, `skidka` — величина трехпроцентной скидки, `sumsk` — стоимость покупки с учетом скидки. В начале программы с клавиатуры вводится сумма покупки. Затем в действие вступает условный оператор. Если величина переменной `sum` больше чем 1000, то будет работать тот вариант программы, который находится после служебного слова `then`. Этот вариант представляет собой составной оператор, состоящий в свою очередь из 4 простых операторов.

В первом из этих операторов находится величина скидки путем умножения значения `sum` на 0.03. Найденная величина присваивается переменной `skidka`. Во втором операторе определяется сумма покупки со скидкой путем вычитания величины скидки из начальной суммы. Эта величина присваивается переменной `sumsk`. В третьем и четвертом операторах производится вывод подсчитанных в предыдущих двух операторах величин на экран компьютера. Вывод этот осуществляется в отформатированном виде. Для значения каждой выводимой переменной предусмотрено два разряда в дробной части, соответствующие копейкам. Следовательно, вывод переменных будет осуществляться таким образом, что цифры перед точкой в каждой из сумм будут соответствовать рублям, а цифры, расположенные после точки, — копейкам.

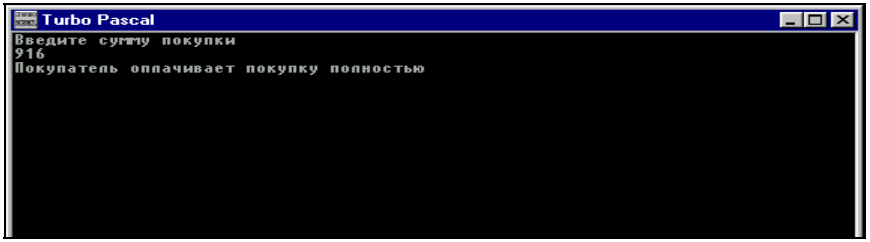
В том случае, если сумма покупки составляет меньше 1000 рублей, будет работать вариант программы, находящийся после служебного слова `else`. В этой ветви программы содержится только один простой оператор `writeln`,



а



б



в

Рис. 4.6. Программа, определяющая скидку при покупке товаров в магазине *(а)*, и результаты ее работы для различных исходных данных *(б, в)*

который выводит на экран компьютера сообщение о том, что покупатель должен оплатить покупку полностью. В листинге 4.5 приводится текст программы.

Листинг 4.5. Программа, определяющая сумму скидки на покупку

```
program magazin;
Uses Crt;
  var sum,sumsk,skidka:real;
```

```
begin
 ClrScr;
 writeln('Введите сумму покупки');
 readln(sum);
 if sum>1000 then
   begin
   skidka:=sum*0.03;
   sumsk:=sum-skidka;
   writeln('Сумма скидки составляет ',skidka:7:2);
   writeln('Покупатель должен заплатить ',sumsk:7:2)
   end
else
  writeln('Покупатель оплачивает покупку полностью');
readln
end.
```

На рис. 4.6 под текстом программы приведены результаты ее работы для обоих возможных случаев (рис. 4.6, *б* — сумма больше 1000 и рис. 4.6, *в* — сумма меньше 1000).

4.2. Логические переменные. Логические операции

Как уже известно читателю, результатом операции сравнения может быть величина, которая принимает одно из двух возможных значений: "истинно" (если указанное соотношение действительно выполняется) и "ложно" (если соотношение не выполняется). По-английски "истинно" и "ложно" пишется соответственно `true` и `false`. Константы и переменные, значениями которых могут быть только эти две величины (`true` или `false`), называются *логическими* или *булевскими* константами и переменными. Название это дано в честь английского математика XIX века Джона Буля, внесшего большой вклад в развитие математической логики. Для описания таких переменных в языке Паскаль существует специальный тип `boolean`. Пример описания логической переменной:

```
var logic:boolean;
```

Как понятно из определения логических переменных, им можно присваивать значения операций сравнения подобно тому, как мы присваиваем числовым переменным значения арифметических операций. Такая операция присваивания может выглядеть, например, следующим образом:

```
logic:=x>=15;
```

т. е. переменной логического типа `logic` мы присваиваем значение операции сравнения `x>=15`. В случае если указанное неравенство выполняется, значение переменной `logic` будет равно `true`. В противоположном случае ее значение будет равно `false`. Имя логической переменной, которой присвоено значение операции сравнения, можно подставлять в условный оператор вместо самой этой операции. Такой прием позволяет сделать программу более компактной в том случае, если одна и та же операция сравнения повторяется в программе несколько раз. Значения логической переменной можно выводить на экран компьютера оператором `writeln` подобно значениям числовых переменных. Можно присваивать значения `true` и `false` логическим переменным и напрямую:

```
logic:=true;
```

Именно такой способ присваивания значений логической переменной мы и используем в следующей программе. Эта программа проверяет, имеется ли в ряду из трех целых чисел хотя бы одно положительное. Эти числа, как обычно, мы будем вводить с клавиатуры, а сообщение о результатах проверки будет выведено на экран компьютера. В программе используются 3 целочисленные переменные `x`, `y` и `z` для вводимых числовых значений и одна переменная логического типа `flag`, назначение которой в данной программе мы рассмотрим далее. В отличие от всех предыдущих программ, в каждой из которых все переменные относились к одному типу, в данной программе имеются переменные, относящиеся к разным типам (целочисленному и логическому). Описание переменной или переменных, относящихся к одному типу, составляет отдельную группу, которую от следующей группы отделяет точка с запятой. При этом слово `var`, открывающее раздел описания переменных, ставится только один раз.

В основной части программы переменной `flag` присваивается начальное значение `false`. Вслед за оператором присваивания в программе идут 3 однотипных блока, в каждом из которых обрабатывается одно из вводимых чисел. Каждый такой блок состоит из оператора вывода, предлагающего пользователю ввести число, оператора ввода, присваивающего введенное значение одной из переменных целого типа, и сокращенного условного оператора, проверяющего, является ли введенное число положительным или отрицательным. В случае если число положительное, переменной `flag` присваивается значение `true`. В том случае, если число отрицательное, никаких действий не производится. Таким образом, если среди вводимых чисел окажется хотя бы одно положительное, то в результате выполнения этих условных операторов итоговое значение переменной `flag` изменится и станет равным `true`. Если же положительных чисел среди введенных нет, то значение `flag` останется неизменным, т. е. равным `false`.

Далее в программе расположен полный условный оператор, который проверяет итоговое значение переменной `flag`. Наличие в этом операторе переменной `flag` (как и любой логической переменной в операторе `if`) означает следующее: если значение этой переменной равно `true`, то выполняется ветвь оператора, расположенная после слова `then`, а если значение переменной — `false`, то выполняется та ветвь, которая расположена после `else`. В данной программе в одной ветви выводится сообщение о наличии среди введенных чисел хотя бы одного положительного, а в другой сообщение о том, что все введенные числа отрицательны. В листинге 4.6 приводится текст программы.

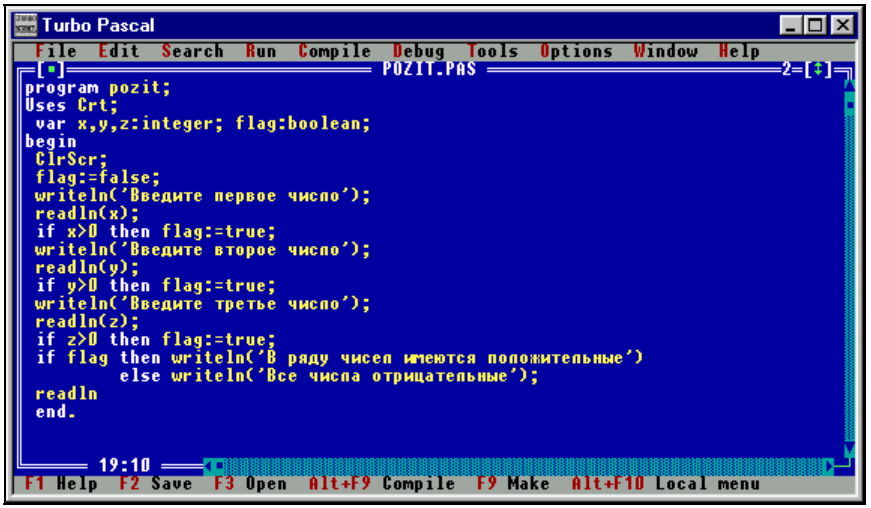
Листинг 4.6. Программа, определяющая, есть ли в ряду чисел хотя бы одно положительное

```
program pozit;
Uses Crt;
 var x,y,z:integer; flag:boolean;
begin
 ClrScr;
 flag:=false;
 writeln('Введите первое число');
 readln(x);
 if x>0 then flag:=true;
 writeln('Введите второе число');
 readln(y);
 if y>0 then flag:=true;
 writeln('Введите третье число');
 readln(z);
 if z>0 then flag:=true;
 if flag then writeln('В ряду чисел имеются положительные')
         else writeln('Все числа отрицательные');
 readln
 end.
```

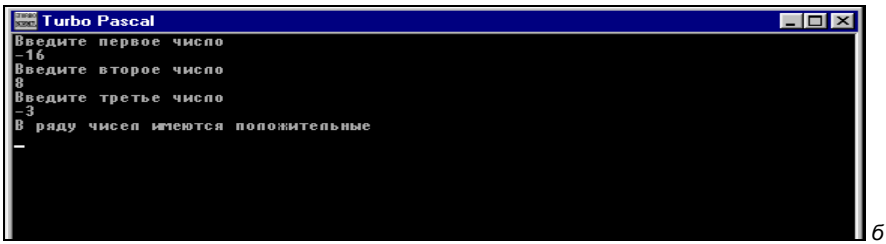
На рис. 4.7 кроме текста программы показаны результаты ее работы при различных исходных данных.

Над величинами логического типа можно производить логические операции подобно тому, как над данными целого или вещественного типа производятся арифметические операции. Результатом логических операций могут быть только логические величины, т. е. величины, имеющие значения `true` и `false`. Всего в Паскале используется 4 логических операции: `not` (Нет — логическое отрицание), `and` (И — логическое умножение), `or` (Или — логическое сложе-

ние), xor (Исключающее Или). Эти операции подразделяются на унарные, т. е. такие, которые производятся только над одной величиной, и бинарные, т. е. те, которые производятся сразу над двумя величинами. К первому типу операций относится not, ко второму — все остальные.



а



б

в

Рис. 4.7. Программа, определяющая, имеются ли в ряду чисел положительные *(а)*, и результаты ее работы при различных исходных данных *(б, в)*

При выполнении логических операций соблюдается, как и для арифметических операций, определенный приоритет. Наиболее высоким приоритетом обладает операция not, т. е. она выполняется в первую очередь. Далее выпол-

няется операция `and`. Самый низкий приоритет имеют операции `or` и `xor`. Если в выражении необходимо изменить порядок выполнения логических операций, то так же, как и в случае с арифметическими операциями, для этого используются скобки, потому что выражение, заключенное в скобки, обладает высшим приоритетом.

Подобно тому, как для числовых величин существуют таблицы сложения и умножения, для логических величин также существует таблица, в которой указаны результаты логических операций при различных исходных данных. Такая таблица называется *таблицей истинности*. В табл. 4.1 и 4.2 приведены таблицы истинности соответственно для унарных и бинарных операций.

Таблица 4.1. *Унарные операции*

X	not(X)
false	true
true	false

Из табл. 4.1 видно, что в результате операции `not`, производимой над любой величиной, ее значение изменяется на противоположное.

Если изучить результаты, показанные в табл. 4.2, то можно сделать вывод о том, что для операции `and` значение будет равно `true` только тогда, когда обе исходные величины `X` и `Y`, над которыми производится эта операция (такие величины называются операндами) имеют значение `true`. Во всех остальных случаях результатом операции будет `false`. Для операции `or` значение будет `true`, если хотя бы один из операндов (или `X` или `Y`) имеет значение `true`. Результат операции `or` будет равен `false` только тогда, когда оба операнда имеют значение `false`. Для операции `xor` значение будет `true`, если значения операндов не совпадают. Если же значения операндов совпадают (вне зависимости от того, будут ли эти значения равны `false` или `true`), то итог операции будет равен `false`.

Таблица 4.2. *Бинарные операции*

X	Y	X and Y	X or Y	X xor Y
false	false	false	false	false
false	true	false	true	true
true	false	false	true	true
true	true	true	true	false

Логические операции используют при составлении программ, в которых требуется проверить сразу несколько условий. Например, если некоторое действие должно выполняться при условии, что значение переменной a находится в диапазоне от 5 до 20, т. е. a должно быть больше или равно 5 и меньше или равно 20, то это условие можно записать следующим образом:

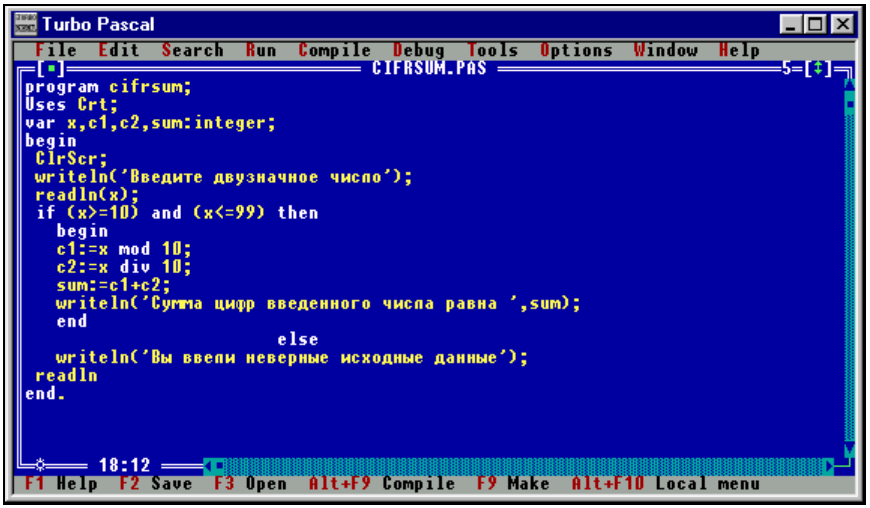
```
(a>=5) and (a<=20)
```

При этом группируемые операции сравнения заключаются в скобки, как в приведенном выражении.

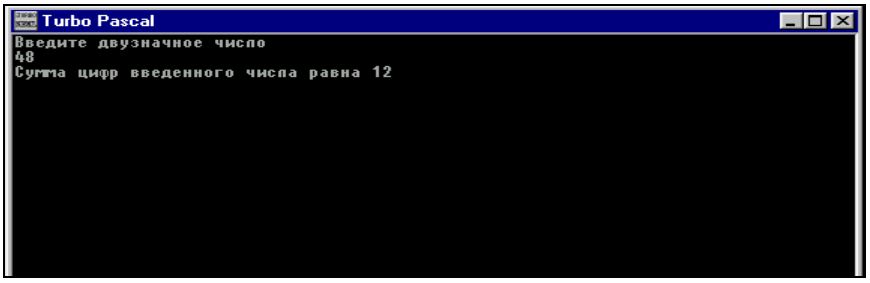
Рассмотрим использование логических операций на примере следующей задачи: составим программу, которая находит сумму цифр двузначного числа. Если же введенное число не является двузначным, то программа должна сообщить пользователю о его ошибке и прекратить на этом свою работу.

Основная часть программы начинается с того, что мы вводим некоторое целое двузначное число x. Далее программа должна проанализировать введенное число и в случае, если оно удовлетворяет условиям задачи, произвести необходимые вычисления, а в случае ошибочного ввода данных вывести сообщение об этом на экран компьютера. Данная задача в программе возложена на условный оператор if, в котором содержится сложное условие, состоящее из двух простых. Этими простыми условиями являются два неравенства, ограничивающие введенное число снизу (оно не может быть меньше 10) и сверху (число не может быть больше 99). Для того чтобы обеспечить проверку совместного выполнения этих двух неравенств, используется объединяющая их операция логического сложения and. Только в случае одновременного выполнения двух условий будет действовать составной оператор, расположенный после служебного слова then.

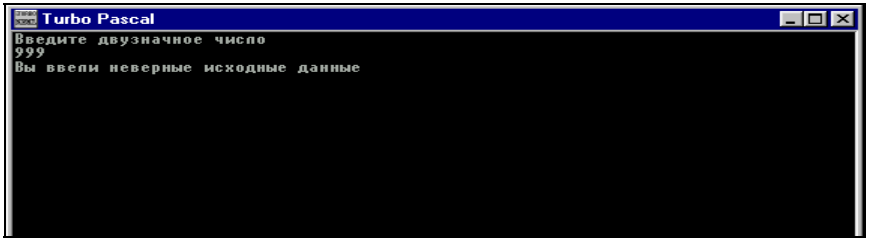
Этот составной оператор производит следующие действия. Вначале мы определяем цифры двузначного числа c1 и c2. Цифра младшего разряда двузначного числа c1 находится как остаток от деления числа на 10. Величину старшего разряда c2 найдем как результат целочисленного деления исходного числа x на 10. Затем останется сложить получившиеся цифры и результат их сложения — число sum — вывести оператором writeln на экран компьютера. В том же случае если число или меньше 10, или больше 99 (т. е. не выполняется хотя бы одно из поставленных в заголовке оператора if условий) работает ветвь оператора, расположенная после else, т. е. на экране появляется сообщение о неправильном вводе исходных данных. В листинге 4.7 приводится текст программы.



а



б



в

Рис. 4.8. Программа определения суммы цифр двузначного числа *(а)* и результаты ее работы при различных исходных данных *(б, в)*

Листинг 4.7. Программа, определяющая сумму цифр двузначного числа

```
program cifrsum;
Uses Crt;
var x,c1,c2,sum:integer;
begin
 ClrScr;
```

```
 writeln('Введите двузначное число');
 readln(x);
 if (x>=10) and (x<=99) then
   begin
   c1:=x mod 10;
   c2:=x div 10;
   sum:=c1+c2;
   writeln('Сумма цифр введенного числа равна ',sum);
   end
 else
   writeln('Вы ввели неверные исходные данные');
 readln
end.
```

На рис. 4.8 показаны результаты работы для обоих возможных вариантов. Таким образом, данная программа не только определяет требуемый результат, но и осуществляет защиту от ошибки при вводе исходных данных (в программировании существует жаргонное наименование такой защиты — "защита от дурака").

4.3. Оператор множественного выбора *case*

При составлении программ часто возникает потребность в структуре, которая обеспечивала бы возможность рассмотреть не два, а большее количество возможных вариантов дальнейших действий. В некоторых случаях для этого используется оператор `if`. В одну или обе ветви оператора вставляется еще по одному условному оператору `if`, который позволяет разбить каждый вариант на два подварианта, т. е., как говорят, используются вложенные условные операторы. В каждый из вложенных условных операторов можно вставить следующий вложенный оператор и т. д. Такой способ создания многовариантного ветвления делает, однако, структуру программы чересчур сложной, что, понятно, может привести к появлению в программе ошибок. Поэтому в языке Паскаль предусмотрена другая конструкция, которая позволяет осуществлять выбор из множества возможных вариантов и в то же время является более простой и наглядной, чем описанная ранее. Эта конструкция реализуется с помощью оператора `case`.

Общий вид оператора `case` следующий:

```
case селектор of
Значение_1: Вариант_1;
Значение_2: Вариант_2;
.....................
```

```
Значение_n: Вариант_n;
else  Вариант_n+1
end;
```

где `case`, `of`, `else` — служебные слова языка Паскаль. Словосочетание `case of` переводится на русский как "в случае если".

Селектором называется переменная целого или символьного типа. Переменная-селектор может принимать различные значения.

После заголовка в операторе идет перечень возможных значений переменной-селектора. Каждому из этих значений соответствует определенный вариант действий, который реализуется в том случае, если в программе селектор принимает это значение. Этот вариант указывается после двоеточия, отделяющего его от значения, и представляет собой простой или составной оператор.

Если переменная-селектор не принимает ни одно из перечисленных в операторе `case` значений, то выполняется альтернативный вариант, который указан после служебного слова `else`. Обратите внимание на то, что в отличие от условного оператора `if` в операторе `case` перед вариантом с `else` ставится точка с запятой. Эта часть оператора (`else` с соответствующим ему вариантом) не является обязательной — возможен сокращенный вариант оператора. Заканчивается оператор `case` служебным словом `end`, после которого ставится точка с запятой.

Разберем следующий фрагмент программы, в которой использован оператор `case`:

```
case k of
1: x:=x+5;
2: x:=x+10;
3: x:=x+20;
else x:=0
end;
```

В данном условном операторе в роли переменной-селектора выступает целочисленная переменная `k`. Если `k` принимает значение, равное 1 (значения переменной-селектора могут либо вводиться с клавиатуры оператором `read` или `readln`, либо присваиваться в программе оператором присваивания), то текущее значение другой целочисленной переменной `x` увеличивается на 5, если `k` принимает значение, равное 2, то текущее значение `x` увеличивается на 10, если `k` принимает значение 3, то значение `x` увеличивается на 20, если же переменная-селектор принимает любое другое значение, то переменная `x` обнуляется. Каждое из указанных после двоеточия действий производится про-

стым оператором присваивания. Составные операторы внутри данного оператора `case` отсутствуют.

Возможности, предоставляемые программисту оператором `case`, мы используем в программе, которая имитирует работу простейшего электронного калькулятора. По запросу пользователя программа выполняет одно из четырех основных арифметических действий над любыми двумя введенными им с клавиатуры целыми числами. В начале работы программы-калькулятора пользователю предлагается ввести с клавиатуры два целых числа, над которыми будет произведена одна из арифметических операций. Затем после ввода чисел операторами `readln` на экран компьютера операторами `writeln` выводится текст программного меню. Этот текст сообщает пользователю программы, нажатие какой клавиши соответствует выполнению какой арифметической операции. Например, нажатие клавиши с цифрой 1 производит сложение введенных чисел, 2 — вычитание и т. д.

Введенное пользователем значение считывается оператором `readln` и присваивается переменной `x`, которая в данной программе выступает в роли переменной-селектора. Далее в действие в программе вступает условный оператор `case`. В первых трех случаях (сложение, вычитание, умножение) для получения результата используется простой оператор `writeln`. Каждый из операторов `writeln` вначале выводит значения операндов, над которыми производится действия, а между ними указывается знак производимой арифметической операции. Затем на экран выводится знак равенства (т. к. этот знак и знак операции являются здесь фрагментами текста, то в списке вывода заключены в апострофы), а в конце указывается само вычисляемое выражение, которое подсчитывается компьютером и выводится на экран. В последнем, четвертом случае мы используем составной оператор. Причина заключается в том, что в этом случае результат операции будет числом вещественным. Мы сперва подсчитаем результат деления, присвоим его вещественной переменной `d`, а затем в списке вывода оператора `writeln` выведем значение переменной `d` в отформатированном виде, обеспечив таким образом точность выведенного вещественного числа-результата до третьего знака после запятой. В листинге 4.8 приводится текст программы.

Листинг 4.8. Программа, моделирующая работу калькулятора

```
program kalkulat;
  Uses Crt;
  var a,b,x:integer; d:real;
begin
  ClrScr;
  writeln('Введите первое число');
  readln(a);
```

```
  writeln('Введите второе число');
  readln(b);
  writeln('Выберите действие, производимое над числами');
  writeln('(введите соответствующую цифру)');
  writeln ('1 - сложение, 2 - вычитание, 3 - умножение, 4 - деление');
  readln(x);
  case x of
    1: writeln(a,'+',b,'=',a+b);
    2: writeln(a,'-',b,'=',a-b);
    3: writeln(a,'*',b,'=',a*b);
    4: begin d:=a/b; writeln(a,'/',b,'=',d:7:3) end
  end;
  readln
end.
```

Результаты работы программы приведены на рис. 4.9.

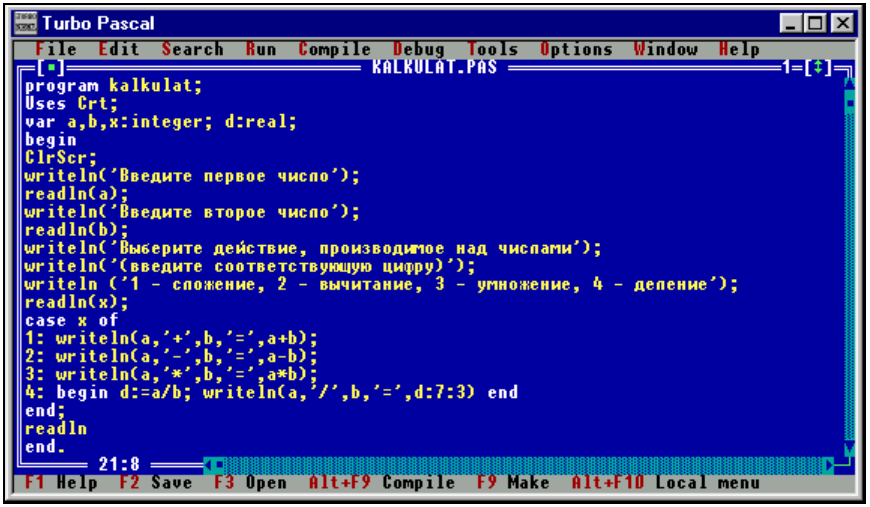


Рис. 4.9. Программа-калькулятор *(а)* и результат ее работы *(б)*

Рассмотренный нами вариант оператора case является несколько упрощенным. В нем каждому варианту действий соответствует только одно значение переменной-селектора. Возможна, однако, и другая форма того же оператора, когда каждому варианту действий соответствует какая-либо группа значений селектора. Эта группа значений, находящаяся в операторе перед двоеточием, может представлять собой как несколько отдельных значений, перечисленных через запятую, так и целый диапазон значений. В последнем случае в соответствующем разделе оператора указывается только начальное и конечное значение диапазона, а между ними ставятся две горизонтальные точки.

Поясним сказанное на следующем примере. Перед нами фрагмент программы с условным оператором case, в котором переменной y присваиваются различные значения, представляющие собой степени x. Какое именно значение будет присвоено переменной y, зависит от значения переменной-селектора n:

```
case n of:
3,5,7: y:=x*x;
10..20: y:=x*x*x;
30..40: y:=x*x*x*x;
else y:=x
end;
```

В случае если переменная n принимает значение, равное числам 3, 5 или 7, то значение y будет равно квадрату x. Если n примет любое целочисленное значение из диапазона от 10 до 20 (10, 11, 12 и т. д. до 20 включительно), то y будет присвоено значение, равное кубу x. Если же n примет любое значение из диапазона от 30 до 40, то y получит значение, равное четвертой степени x. Наконец, если n не примет ни одно из указанных ранее значений, то y будет равно первой степени x.

Приведенную ранее форму записи оператора case мы используем для решения следующей задачи: требуется составить программу, которая запрашивала бы пользователя о том, какой сейчас месяц, и по введенному пользователем номеру месяца определяла бы текущее время года. При этом сообщение о времени года должно выводиться на экран компьютера соответствующим сезону цветом. Если сейчас зима, то сообщение об этом должно выводиться белым цветом, для весны сообщение выводится зеленым цветом, для лета — красным цветом и для осени — желтым. В листинге 4.9 приводится текст программы.

Листинг 4.9. Программа, определяющая время года по введенному номеру месяца

```
program month;
Uses Crt;
```

```
var m:integer;
begin
ClrScr;
 writeln('Введите номер месяца');
 readln(m);
 case m of
 1,2,12:
   begin
   textcolor(LightGray);
   writeln('Пришла зима')
   end;
 3,4,5:
   begin
   textcolor(Green);
   writeln('Пришла весна')
   end;
 6,7,8:
   begin
   textcolor(Red);
   writeln('Наступило лето')
      end;
 9,10,11:
   begin
   textcolor(Yellow);
   writeln('Наступила осень')
   end;
  else writeln('Месяца с таким номером не существует')
 end;
readln
end.
```

В данной программе используется всего одна переменная — номер текущего месяца `m`, выступающий в роли переменной-селектора. В зависимости от вводимого пользователем значения переменной выполняется один из пяти возможных вариантов работы программы. В первом случае для зимних месяцев с номерами 1, 2 или 12 можно использовать один простой оператор `writeln`, который выводит сообщение о том, что сейчас зима. Так как для вывода информации в Turbo Pascal по умолчанию используется белый цвет, то специально указывать цвет сообщения в данном случае нет необходимости. Следующие три варианта работы программы аналогичны друг другу. Каждый из них (для месяцев с третьего по пятый, с шестого по восьмой или с девятого по одиннадцатый) представляет собой составной оператор, который включает в себя два простых. Первый из этих операторов устанавливает цвет сообще-

ния (соответственно зеленый, красный или желтый). Вторым оператором является оператор вывода, который и выводит на экран компьютера нужным цветом соответствующий текст. Наконец последний вариант работы программы соответствует тому случаю, когда пользователь случайно ошибся (например, указал несуществующий в григорианском календаре нулевой или тринадцатый месяц). В этом случае программа сообщает пользователю о совершенной им ошибке и на этом прекращает свою работу (рис. 4.10).

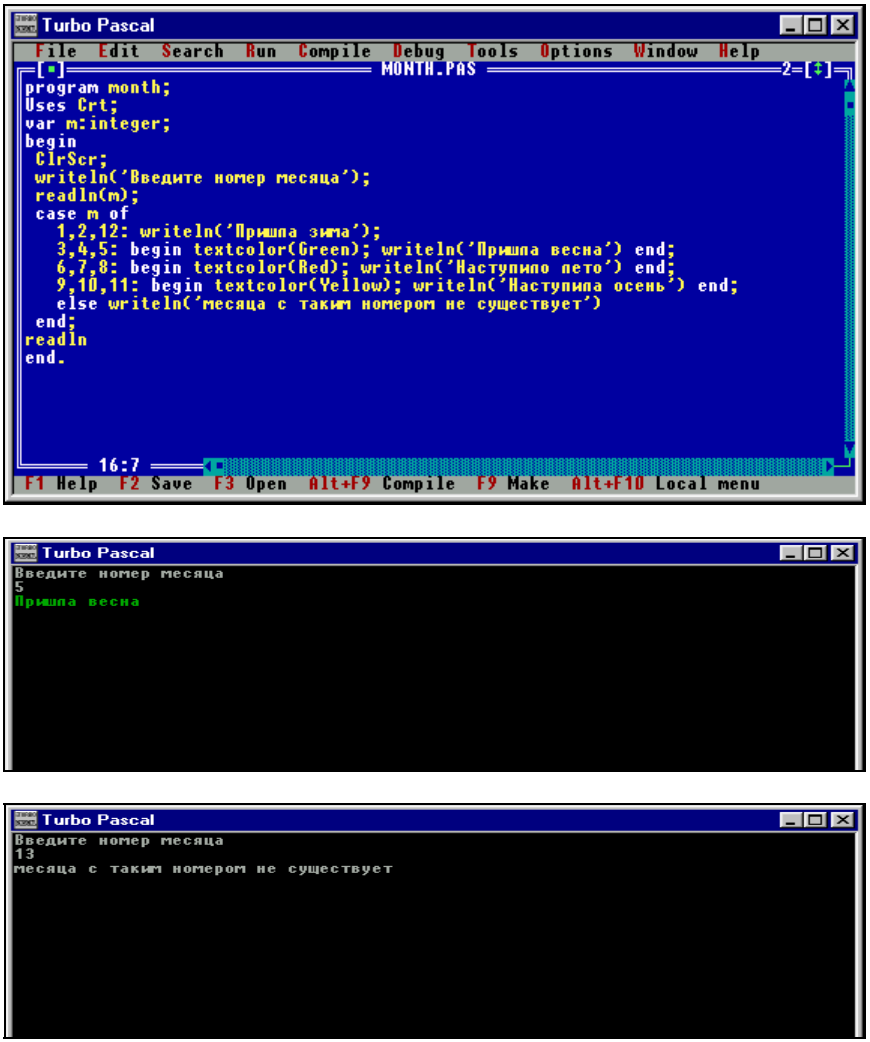


Рис. 4.10. Программа, определяющая время года по номеру месяца *(а)*, и результаты ее работы при различных исходных данных *(б, в)*

4.4. Безусловный оператор перехода *goto*

В программах, рассмотренных нами в предыдущих разделах книги, операторы либо выполнялись друг за другом последовательно (линейные программы), либо при наличии в программе условного оператора ход выполнения программы зависел от выполнения или невыполнения некоторого условия. Однако бывают и такие ситуации, когда требуется нарушить линейный ход выполнения программы без выполнения какого-либо предварительного условия. Такая ситуация возникает, например, если в определенном месте программы требуется вернуться к ее началу для того, чтобы выполнить заново находящиеся в начале программы операторы или, наоборот, перейти в конец программы для того, чтобы досрочно завершить ее выполнение.

Оператор, который позволяет производить такое действие, называется оператором *безусловного перехода*. Этот оператор может не только осуществлять переход в начало или конец программы, но и перемещаться практически в любое место программы и продолжать ее выполнение, начиная с любого нужного оператора (или, как говорят, передавать управление в любую точку программы). Естественно, что при выполнении оператора безусловного перехода компьютер должен знать, куда именно следует передавать управление. Для этого в начале строки, содержащей оператор, на который передается управление, ставится целое число, называемое меткой. Эта же метка указывается и в самом операторе безусловного перехода.

Общий вид оператора безусловного перехода следующий:

```
goto имя_метки;
```

где `goto` — служебное слово, означающее в переводе с английского "перейти к", *имя_метки* — имя метки, ранее описанное в разделе `label`.

Имя метки может быть произвольным, но оно должно подчиняться тем же правилам, что и имена переменных. В качестве меток также можно использовать целые числа без знака. Для удобства работы меткам следует давать осмысленные имена: `metka1`, `metka2` и т. д.

Строка, содержащая метку, должна выглядеть следующим образом:

```
metka: оператор;
```

т. е. строка, на которую производится переход, должна обязательно начинаться с имени метки, после которого ставится двоеточие. После двоеточия в строке пишутся оператор или операторы, как и в любой обычной строке программы. Дальнейшее выполнение программы осуществляется именно с этого места.

Используя оператор безусловного перехода и соответствующую ему метку, необходимо помнить о том, что подобно константам и переменным метка

должна быть предварительно описана в разделе описаний перед ее использованием в основной части программы. Описание метки должно находиться в самом начале раздела описаний перед описанием переменных и констант.

В общем виде описание метки выглядит следующим образом:

```
label имя_метки;
```

где `label` — служебное слово, означающее в переводе "метка", *имя_метки* — имя метки, удовлетворяющее описанным ранее условиям.

Говоря о метках, необходимо обратить внимание еще и на следующее обстоятельство. Если в программе содержится некий безусловный оператор, осуществляющий переход на начало программы, то каждый раз, как только программа будет доходить до этого оператора, выполнение программы будет происходить с начала до места нахождения безусловного оператора, после чего будет вновь возвращение на начало. Программа с безусловным оператором будет в этом случае выполняться без конца или, как говорят, "зацикливаться", не выдавая конечных результатов работы.

Поэтому, как правило, оператор `goto` не используют в программе сам по себе, а вставляют его в качестве составной части в другой оператор, в частности в условный. Безусловный оператор может содержаться в одной из ветвей условного оператора. В таком случае возвращение на начало программы будет производиться не вечно, а до тех пор, пока будет выполняться или не выполняться условие, находящееся в заголовке условного оператора, т. е. процесс выполнения программы станет конечным, зацикливание будет преодолено. В этом своем качестве оператор может быть полезен, например, для защиты программы от неправильного ввода данных.

В случае если исходные данные, введенные пользователем неверны, можно осуществить проверку исходных данных с помощью условного оператора и, если они неправильны, вернуться на начало программы посредством безусловного оператора. После возвращения на начало желательно, чтобы пользователь был проинформирован о том, в каком диапазоне должны находиться правильные введенные данные. Если пользователь ввел данные правильно, то ветвь условного оператора, содержащая оператор `goto`, не будет реализована, и программа продолжит свою работу до конца.

Как конкретно работает такая защита, мы разберем на следующем примере. Вернемся к уже рассмотренной нами программе (листинг 4.10), в которой мы предусмотрели "защиту от дурака" в ветви программы, находящейся после служебного слова `else`. Невнимательному пользователю сообщается о сделанной им ошибке (вводе несуществующего номера месяца), а затем программа завершается. Эта программа работает в принципе правильно, но ведет себя не очень корректно по отношению к пользователю, которому, для того

чтобы исправить допущенную ошибку, требуется заново запускать программу. Модифицируем программу таким образом, чтобы пользователь мог исправить свою ошибку, не выходя из программы.

Сделаем мы это следующим образом: изменим ветвь, находящуюся после `else`. В новой версии программы после `else` будет находиться не простой оператор вывода, а составной оператор, заключенный в операторные скобки `begin` и `end`. В составной оператор будут входить три простых оператора. Первый оператор — оператор вывода, идентичный тому, что был в предыдущей версии программы. Этот оператор сообщает о самом факте ввода неверных исходных данных.

Следующий оператор конкретизирует информацию. Он сообщает о том, в каких пределах может находиться вводимое данное — номер месяца. Наконец третий оператор представляет собой оператор безусловного перехода на вторую строчку основной части программы (сразу после очистки экрана), которая имеет метку 10. Закрывается составной оператор служебным словом `end`. За ним находится следующее служебное слово `end`, которое на сей раз закрывает весь условный оператор `case` целиком.

Строка, на которую осуществляется переход, как уже говорилось, имеет метку с номером 10. Этот номер метки ставится в начале строки вместе с двоеточием, после двоеточия находится оператор вывода, который предлагает пользователю вновь ввести исходное данное. Метка 10 указана в разделе описаний после служебного слова `label`. Теперь в том случае, если пользователь введет номер месяца, равный 0 или 13, его не будет "выбрасывать" из программы, а программа будет каждый раз при вводе неверных данных выдавать сообщение об ошибке и возвращать пользователя к моменту ввода данных. В листинге 4.10 приводится текст модифицированной программы.

Листинг 4.10. Модифицированная программа определения времени года по номеру месяца

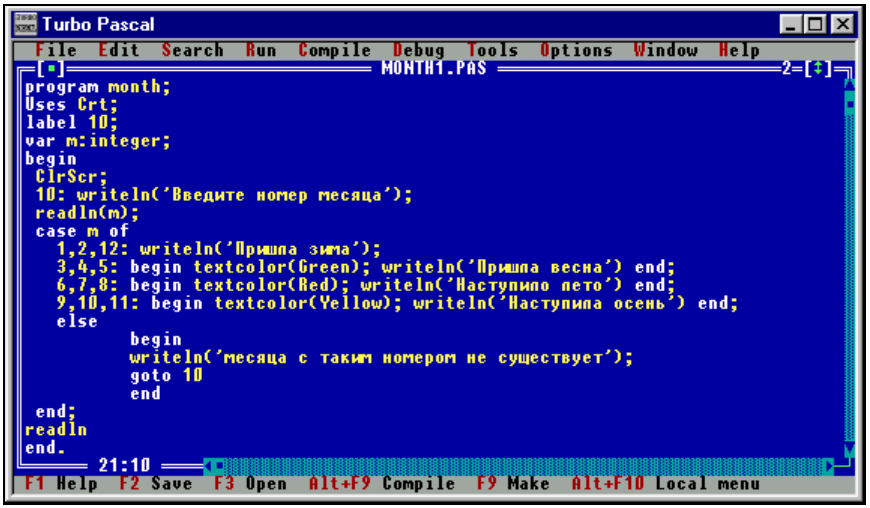
```
program month1;
Uses Crt;
label 10;
var m:integer;
begin
 ClrScr;
 10: writeln('Введите номер месяца');
 readln(m);
 case m of
 1,2,12:
   begin
   textcolor(LightGray);
```

```
   writeln('Пришла зима')
   end;
 3,4,5:
   begin
   textcolor(Green);
   writeln('Пришла весна')
   end;
 6,7,8:
   begin
   textcolor(Red);
   writeln('Наступило лето')
      end;
 9,10,11:
   begin
   textcolor(Yellow);
   writeln('Наступила осень')
   end
  else
    begin
    writeln('Месяца с таким номером не существует');
    goto 10
    end
 end;
readln
end.
```

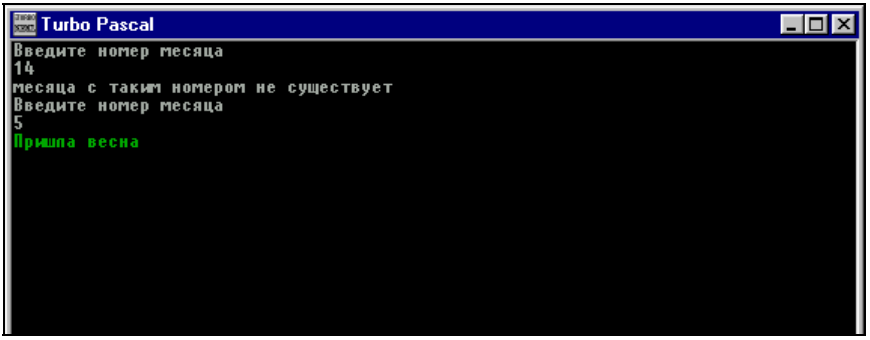
На рис. 4.11 изображены текст модифицированной программы определения времени года по номеру месяца и два возможных результата ее работы: для неправильного ввода данных и для правильного ввода.

В приведенной программе был использован один оператор безусловного перехода и одна метка, но в Паскале разрешается производить ссылки на одну и ту же строку из разных мест программы несколькими операторами безусловного перехода. Использовать же несколько раз одну и ту же метку для обозначения разных строк или разных операторов недопустимо.

Использование меток и операторов безусловного перехода в ряде случаев облегчает составление программ, однако не следует злоупотреблять их применением, т. к. при наличии в программе большого количества меток и операторов ее структура излишне запутывается и усложняется. В дальнейшем мы с вами убедимся, что можно избежать использования меток и операторов безусловного перехода за счет грамотного использования других стандартных операторов (в частности, операторов цикла). При этом не будет нарушаться и запутываться структура программы.



а



б

Рис. 4.11. Модифицированная программа определения времени года по номеру месяца *(а)* и результат ее работы *(б)*

Задания для самостоятельной работы

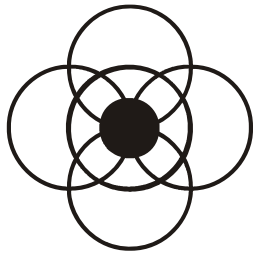
Задание 1. Составить программу, которая определяет, делится ли введенное с клавиатуры целое число на 5.

Задание 2. Составить программу, которая проверяет, является ли введенное с клавиатуры число одновременно положительным и четным.

Задание 3. Стоимость проезда одной зоны на пригородном поезде составляет 3 рубля за одну зону. Нумерация зон ведется, начиная с городского вокзала (1 зона). Составить программу, которая бы определяла стоимость проезда в один конец между станциями, расположенными в разных зонах. Номера зоны, в которой находится станция отправления и той зоны, в которой распо-

ложена конечная станция, вводятся в компьютер с клавиатуры. При составлении программы следует учесть, что пассажир может отправляться как из пригорода в город (из зоны с большим номером в зону с меньшим номером), так и из города в пригород (т. е. из зоны с меньшим номером в зону с большим номером).

Задание 4. Усовершенствовать программу, вычисляющую стоимость проезда на пригородном поезде, следующим образом. В начале работы программа должна выводить на экран компьютера меню, предлагающее пользователю следующие варианты: 1 — вычисление стоимости поездки в один конец, 2 — вычисление стоимости поездки в оба конца (стоимость такого билета равна удвоенной стоимости билета в один конец), 3 — вычисление стоимости месячного билета (эта стоимость равна стоимости поездки в один конец, умноженной на 15). Затем программа должна вычислять соответствующую стоимость.



ГЛАВА 5

Циклы в Паскале

В ходе создания программ нередко возникает ситуация, когда программисту нужно многократно использовать один и тот же оператор или группу операторов, которые аналогичны друг другу и незначительно отличаются лишь значениями некоторых переменных или вводимых исходных данных. Такая ситуация хорошо видна на следующем примере. Пусть нам необходимо разработать программу, создающую таблицу для перевода величины, выраженной в одних единицах измерения в другие.

Конкретизируем задачу следующим образом: нам требуется составить таблицу для перевода веса, выраженного в центнерах в килограммы. Созданная программой таблица должна охватывать диапазон от 1 до 50 центнеров. Для каждого целого значения, выраженного в центнерах (1, 2, 3 и т. д. вплоть до 50), процесс перевода этого веса в килограммы и создания соответствующей строки в таблице будет осуществляться с помощью двух операторов. Первый из этих операторов производит вычисление суммы в килограммах, а второй оператор выводит получившуюся величину на экран компьютера. Данная группа операторов будет выглядеть следующим образом:

```
kg:=c*100;
writeln(c,kg);
```

где `c` — вес, выраженный в центнерах, а `kg` — тот же вес, выраженный в килограммах, причем, как нетрудно понять, значение переменной `kg` напрямую зависит от значения `c`.

Для того чтобы выполнить поставленную задачу, можно конечно поступить следующим образом: повторить данную группу операторов в программе 50 раз, а перед каждой группой операторов присваивать переменной `c` новое значение либо с помощью оператора присваивания, либо используя оператор ввода `readln`. При этом начальным значением переменной `c` должна быть

единица, после выполнения приведенных ранее двух операторов переменная kg получит значение 50. Затем перед следующим выполнением группы из двух операторов c должно быть присвоено значение 2, а после их выполнения kg будет иметь значение 100. Последнее значение c будет равно 50, а kg соответственно 5000. Понятно, однако, что при решении данной задачи "в лоб" получившаяся программа будет чересчур громоздкой за счет многократного повторения одних и тех же действий.

Поэтому естественно, что возникает вопрос, нельзя ли использовать при решении этой и других подобных задач какую-либо структуру, которая бы позволяла облегчить и сделать менее трудоемким решение этой задачи путем автоматического повторения данного оператора или группы операторов с соответствующим изменением значений переменной или переменных. Такая структура в языке Паскаль существует и называется она *циклом*.

В общем виде цикл состоит из двух основных блоков:

- *Заголовок цикла.* В заголовке цикла содержится некоторое условие, которое определяет число повторений цикла.
- *Тело цикла.* Телом цикла называется простой или составной оператор (как читатель помнит, составным оператором называется группа операторов, заключенная в операторные скобки), который может многократно повторяться в ходе работы цикла. В теле цикла могут содержаться операторы различных типов — это операторы ввода и вывода, операторы присваивания, условные операторы, а также другие *операторы цикла*. Оператор цикла, находящийся в теле другого циклического оператора, называется вложенным.

Перед началом работы цикла производится проверка содержащегося в заголовке условия. Если условие выполняется, то происходит выполнение тела цикла, после чего программа вновь возвращается к заголовку цикла и осуществляется новая проверка условия и т. д. Если же условие в заголовке оказывается невыполненным, то работа цикла завершается и управление программой переходит к оператору, расположенному в программе вслед за циклом. При этом в правильно работающем цикле не должно происходить бесконечного повторения тела цикла, приводящего к зацикливанию программы. Поэтому условие, находящееся в заголовке, должно быть составлено таким образом, чтобы в какой-то момент в ходе выполнения программы оно становилось ложным, что приводило бы к завершению работы цикла. В нашем примере с программой перевода центнеров в килограммы таким изменением могло бы стать получение переменной c значения больше 50. В случае, когда c, например, получает значение 51, цикл должен прекращать свою работу.

Циклы могут использоваться для решения самых разнообразных задач, и соответственно структура самих циклов может несколько отличаться.

Всего в языке Паскаль используется 3 разновидности циклов.

- Цикл с заранее заданным числом повторений. Его также называют *циклом со счетчиком* (цикл `for...to`).
- *Цикл с предусловием* (цикл `while`).
- *Цикл с постусловием* (цикл `repeat...until`).

Каждый из этих видов цикла имеет свои особенности и область применения, которые мы и рассмотрим далее. Общим для всех этих операторов является то, что они включают в себя управляющие конструкции (в операторах `for...to` и `while` — это строка заголовка, а в операторе `repeat...until` — строка заголовка и завершающая строка) и тело цикла. Если количество повторений тела цикла заранее известно, то рекомендуется использовать оператор цикла `for...to`. В другом случае нужно использовать оператор `repeat...until` (если для решения поставленной задачи требуется, чтобы тело цикла выполнялось хотя бы один раз), либо оператор `while` (в этом случае перед выполнением тела цикла проверяется, есть ли вообще необходимость в его выполнении). Далее перейдем к более подробному рассмотрению каждого из этих видов цикла.

5.1. Цикл с заранее известным числом повторений

Общий вид данного оператора следующий:

```
for счетчик_цикла:=нач_значение to кон_значение do
тело_цикла;
```

где:

`for`, `to` и `do` — служебные слова (`for` в данном случае означает "для", `to` — "до", `do` — "делать, выполнять").

`счетчик_цикла` — управляющая переменная цикла, значение которой изменяется от начального до конечного. При этом начальное значение должно быть меньше конечного. После того как переменная-счетчик цикла примет конечное значение, тело цикла выполнится последний раз и цикл будет завершен. Переменная-счетчик обычно относится к целочисленному типу, допускаются и некоторые другие типы, которые мы рассмотрим позднее. Категорически нельзя использовать в качестве счетчика переменные вещественного типа.

`нач_значение`, *`кон_значение`* — начальное и конечное значения счетчика цикла. При этом конечное значение должно быть больше начального. Оба значения должны быть того же типа, что и счетчик цикла.

тело_цикла — обычно в качестве тела цикла выступает составной оператор, в который входит несколько других. Однако это может быть и один оператор. В последнем случае телом цикла считается первый оператор, расположенный после служебного слова do.

Если переменная-счетчик имеет тип integer, то в ходе работы цикла ее значение при каждом выполнении тела цикла увеличивается на единицу и таким образом принимает все целочисленные значения от начального до конечного. Следует обратить внимание на то, что между заголовком цикла и телом цикла не ставится точка с запятой.

Решение задачи по переводу центнеров в килограммы, рассмотренной нами в предыдущем разделе книги, будет тогда выглядеть следующим образом:

```
for c:=1 to 50 do
  begin
  kg:=c*100;
  writeln(c,kg)
  end;
```

В приведенном фрагменте программы роль счетчика цикла выполняет переменная c (вес в центнерах). Начальное значение переменной цикла равно 1. Для этого начального значения выполняются действия, указанные в составном операторе, т. е. будет вычислено значение kg, равное 50, а затем значения 1 и 50 будут выведены на экран компьютера. Далее будет подсчитано значение kg для c, равного 2, и числа 2 и 100 будут выведены на экран в следующей строке, аналогичные действия будут производиться до тех пор, пока не будет выведена последняя строка со значениями c и kg — 50 и 5000. Цикл прекратит свою работу, когда переменная c достигнет конечного значения — 50. Всего действия, указанные в теле цикла, будут повторены согласно приведенной формуле 50–1+1 раз, т. е. 50 раз. Таким образом, мы решили задачу, поставленную в начале данной главы, написав фрагмент программы из 5 строк. Нетрудно подсчитать, что если решать эту задачу без использования цикла, как мы делали раньше, то такой фрагмент программы состоял бы из более 100 операторов.

Для большей наглядности решение данной задачи представлено на рис. 5.1 в виде графической схемы. На схеме внутри шестиугольника заключены параметры цикла (название переменной цикла, ее начальное и конечное значение). Такой блок в графических схемах называют также "модификация". Тело цикла состоит из двух блоков. В первом блоке (блок "процесс") происходит вычисление очередного значения, выраженного в килограммах. Во втором блоке производится вывод полученных величин на экран компьютера, что изображается в виде параллелограмма. Замкнутый контур, образованный

стрелками, показывает, что в цикле повторяется одна и та же последовательность действий до тех пор, пока переменная цикла не достигнет конечного значения. После этого тело цикла выполняется в последний раз, а затем программа переходит к выполнению следующего оператора (стрелка, ведущая вниз).

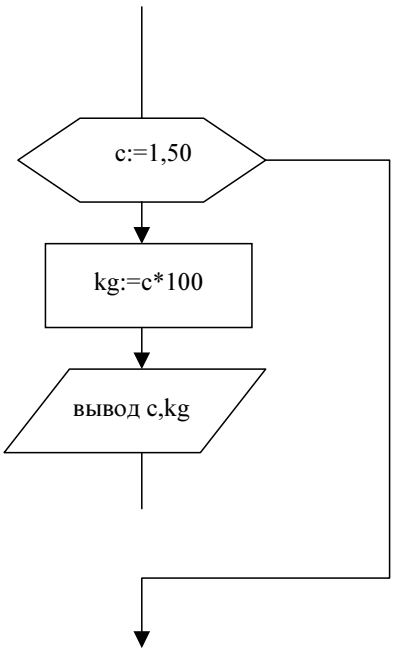


Рис. 5.1. Графическая схема фрагмента программы перевода веса из центнеров в килограммы

Приведенный вариант цикла `for` не является единственным. Цикл типа `for` может иметь и такой общий вид:

```
for счетчик_цикла:=нач_значение downto кон_значение do
тело_цикла;
```

В этом случае при каждом повторении тела цикла значение счетчика уменьшается на единицу, и, следовательно, конечное значение счетчика цикла должно быть меньше начального.

В качестве примера работы цикла типа `for` приведем программу, которая составляет таблицу для перевода расстояния, выраженного в милях, в километры. На экран компьютера должна быть выведена таблица для расстояний от 1 до `n` миль, где `n` — целое положительное число, вводимое с клавиатуры пользователем. 1 миля составляет 1,609 километра. Таблица, выводимая на экран

компьютера, должна иметь заголовок и состоять из двух столбцов. В левом столбце должны быть указаны расстояния в милях, а в правом — соответствующие им расстояния в километрах. В листинге 5.1 приводится текст данной программы.

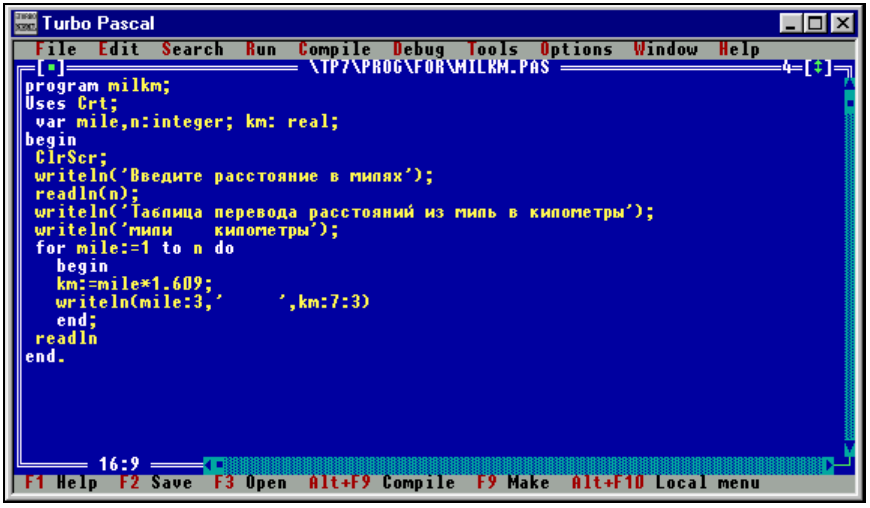
Листинг 5.1. Программа, переводящая расстояние, выраженное в милях, в километры

```
program milkm;
Uses Crt;
 var mile,n:integer; km: real;
begin
 ClrScr;
 writeln('Введите расстояние в милях');
 readln(n);
 writeln('Таблица перевода расстояний из миль в километры');
 writeln('мили    километры');
 for mile:=1 to n do
   begin
   km:=mile*1.609;
   writeln(mile:3,'     ',km:7:3)
   end;
 readln
end.
```

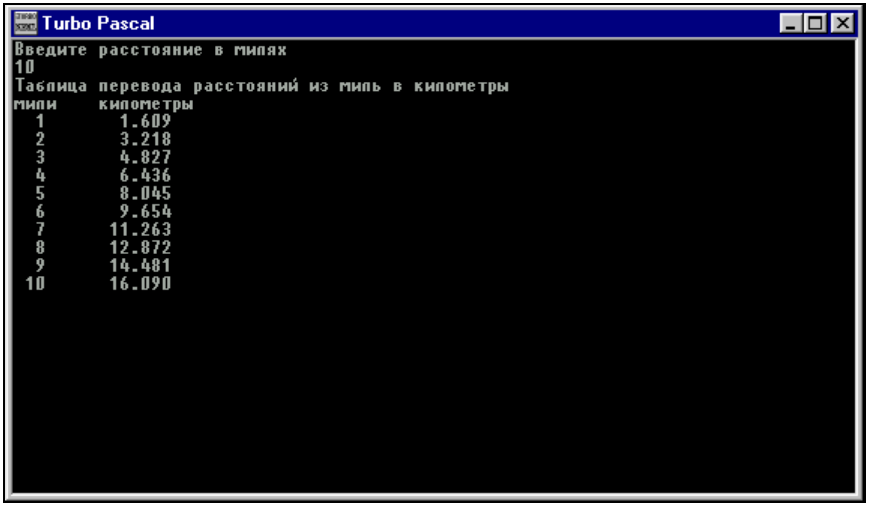
В данной программе для решения задачи используются три переменные. Это, во-первых, переменная целого типа `mile`, содержащая расстояния в милях, которая одновременно является счетчиком цикла. Во-вторых, это переменная `n`, содержащая максимальное число миль, которое нужно перевести в километры. Наконец, это переменная вещественного типа `km`, представляющая собой расстояние в километрах.

В программе с помощью оператора `readln` пользователь вводит величину `n`. Это требуется для определения параметров цикла. Начальное значение счетчика цикла `mile` равно 1, а конечное — `n`. Тело цикла представляет собой составной оператор, содержащий 2 оператора. Первый оператор является оператором присваивания, в котором для каждого значения `mile` (расстояние в милях) вычисляется соответствующее ему значение `km` (расстояние в километрах). Во втором операторе, входящем в состав тела цикла, операторе `writeln`, для обеих выводимых переменных используется форматный вывод. Разница заключается только в том, что в первом случае (для целочисленной переменной `mile`) указывается лишь общее количество выводимых на экран

символов (здесь 3), а во втором случае (для вещественной переменной km) указывается и общее количество символов и их количество в дробной части (здесь соответственно 7 и 3). Это делается для того, чтобы выровнять столбцы выводимых чисел. При n=10 получаем результаты работы программы, показанные на рис. 5.2.



а



б

Рис. 5.2. Программа для пересчета расстояний из миль в километры *(а)* и результаты ее работы *(б)*

5.2. Цикл с постусловием

Общий вид оператора следующий:

```
repeat
тело_цикла
until условие;
```

где `repeat` и `until` — служебные слова. `repeat` означает "повторять", `until` — "до тех пор". Цикл работает следующим образом: вначале выполняется оператор или группа операторов, входящая в тело цикла. Затем проверяется, выполняется ли условие, находящееся после `until`. Если условие истинно, то работа цикла на этом завершается, если условие ложно, то тело цикла выполняется снова, после чего выполняется очередная проверка и т. д. Таким образом, данный цикл будет работать до тех пор, пока условие, находящееся после слова `until`, ложно. Такой цикл называют циклом с постусловием (латинская приставка "пост" означает "после"). Подобное название цикла связано с тем, что проверка условия, определяющего работу цикла, производится уже после его выполнения. По определению цикла понятно, что даже если условие вообще никогда не выполняется, то тело цикла будет выполнено хотя бы один раз.

Цикл такого типа удобно использовать для решения следующей задачи: требуется определить средний рост учеников в классе. Данные о росте каждого ученика вводятся с клавиатуры. Сколько учащихся в классе заранее не известно, но известно, что признаком окончания ввода исходных данных является ввод нуля.

Для решения данной задачи требуется произвести следующие действия: во-первых, обеспечить ввод требуемых исходных данных, представляющих собой последовательность целых чисел (условимся, что рост мы измеряем с точностью до сантиметра). Далее нужно найти среднее арифметическое этой последовательности чисел. Для этого сначала нужно найти сумму членов последовательности.

Попробуем большую часть этих действий выполнить в ходе работы цикла. Перед началом работы программы опишем следующие переменные: `rost` — используется для хранения роста каждого отдельного ученика; `n` — общее число членов последовательности, которая будет на единицу больше, чем общее число учеников в классе, т. к. последний член последовательности — это, как мы помним, не чей-либо рост, а признак конца; `sum` — сумма членов последовательности; `sred` — искомый средний рост учеников в классе (эта величина должна быть вещественной, т. к. она является результатом деления).

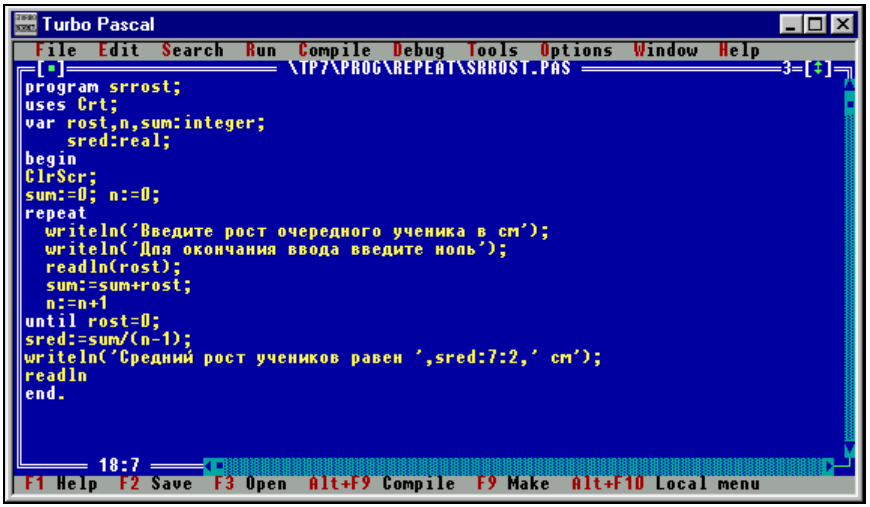
Перед началом работы цикла обнуляем переменные `sum` и `n`, т. к. число членов пока что должно быть равно нулю, а соответственно нулю должна быть равна и их сумма. Далее после служебного слова `repeat` идут операторы тела цикла. Первыми двумя операторами тела являются оператор вывода, содержащий приглашение на ввод роста очередного ученика, и оператор, напоминающий о том, что для окончания ввода требуется ввести нуль. Следующий оператор — оператор ввода — обеспечивает присваивание введенной с клавиатуры величины (роста данного ученика) переменной `rost`. Далее в цикле находится оператор присваивания, который увеличивает текущее значение переменной `sum` на величину, равную введенному росту. Таким образом, к концу работы цикла в переменной `sum` мы получим сумму ростов всех учеников класса (при последнем выполнении цикла будет введен еще и дополнительный нуль, но он, понятно, на общую сумму не повлияет). Но знания значения только переменной `sum` нам недостаточно для решения задачи, т. к. нам необходимо знать еще и число учеников. Для определения этого числа мы подсчитаем сначала число членов последовательности, которое будет лишь на единицу больше числа учеников. Для такого подсчета в цикле мы будем использовать счетчик `n`. Этот счетчик будет увеличиваться на единицу после ввода очередного члена последовательности.

В итоге по окончании работы цикла нам останется только определить средний рост, представляющий собой частное от деления итоговой суммы на число учеников класса. Первая величина нам уже известна, а вторую величину мы найдем, вычтя из итоговой величины `n` (общего числа членов последовательности) единицу. Частное от деления этих двух величин `sred` и будет искомой величиной, которую останется только вывести на экран компьютера оператором вывода. В листинге 5.2 приводится текст данной программы.

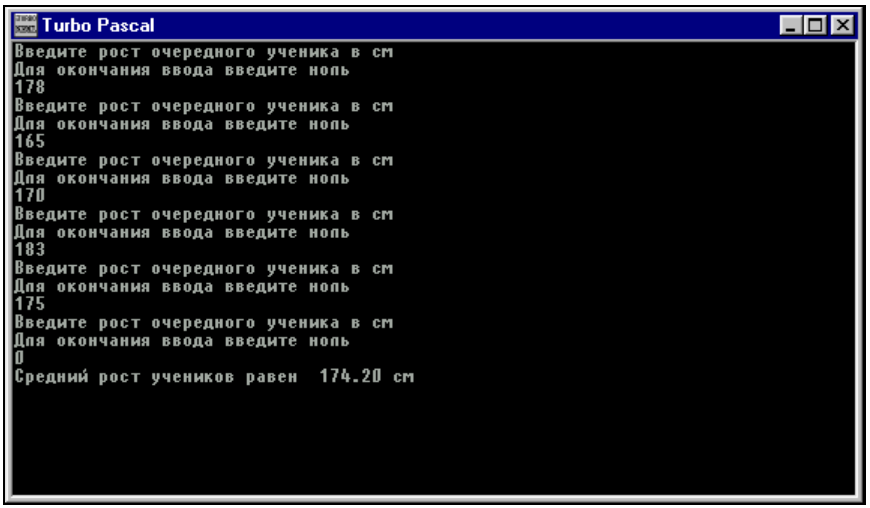
Листинг 5.2. Программа определения среднего роста учеников класса

```
program srrost;
Uses Crt;
 var rost,n,sum:integer;
     sred:real;
begin
 ClrScr;
 sum:=0;
 n:=0;
 repeat
   writeln('Введите рост очередного ученика в см');
   writeln('Для окончания ввода введите 0');
   readln(rost);
```

```
    sum:=sum+rost;
    n:=n+1
  until rost=0;
  sred:=sum/(n-1);
  writeln('Средний рост учеников равен ',sred:7:2,' см');
  readln
end.
```



а



б

Рис. 5.3. Программа определения среднего роста *(а)* и результат ее работы *(б)*

Как уже говорилось, эту разновидность оператора цикла целесообразно использовать в тех случаях, когда заранее нельзя предсказать сколько раз цикл будет повторяться. В частности, с такой ситуацией часто можно столкнуться при разработке различных компьютерных игр, когда те или иные действия игрока повторяются заранее неизвестное количество раз.

Создадим собственными силами компьютерную игру со следующими правилами. Компьютер задумал некоторое целое число в диапазоне от 1 до 20. Игрок должен угадать задуманное число. Всего игроку дается 5 попыток угадать число. Игра продолжается до тех пор, пока игрок не угадал число или не исчерпал имеющиеся у него попытки. Игрок выигрывает в том случае, если он угадал число с любой из 5 попыток, и проигравшим, если попытки закончились, а он так и не угадал число. При этом желательно, чтобы число, которое загадал компьютер, не повторялось от игре к игре, и каждый раз было другим.

Для того чтобы составить такую программу, нам нужно будет познакомиться с двумя новыми командами языка Паскаль, которые используются для работы с генератором случайных чисел. *Генератором случайных чисел* называется специальная подпрограмма, входящая в состав языка Паскаль, которая выдает случайным образом выбранные случайные числа, которые в зависимости от указанных параметров могут быть как целыми, так и дробными. Мы для нашей программы рассмотрим тот вариант работы генератора, когда он выдает целые числа. Для того чтобы использовать генератор случайных чисел, его следует предварительно запустить в работу или, как говорят, инициализировать. Инициализация генератора случайных чисел производится командой

```
Randomize;
```

Следующее, что требуется сделать, — это взять некоторое число из созданного ряда случайных чисел. Данное действие производится посредством команды

```
random(n)
```

где `n` — указываемое пользователем целое число или имя целочисленной переменной. Команда `random` будет выбирать числа, которые находятся в диапазоне от 0 до `n`–1. Полученное командой `random` число можно в дальнейшем вывести на экран компьютера оператором вывода или использовать в качестве составной части арифметического выражения. Если в программе будет, например, указана команда

```
random(10);
```

то данная команда выдаст число из ряда 0, 1, 2, ..., 9, т. е. первое число ряда равно 0, а последнее будет на единицу меньше, чем указанное в скобках. Ес-

ли же мы хотим, чтобы выводимое на экран компьютера посредством команды `random` число было натуральным однозначным числом (т. е. принимало значения 1, 2, ..., 9), то используем прием, показанный в следующем фрагменте программы:

```
Randomize;
writeln(random(9)+1);
```

В этом случае к выводимому значению будет добавлена единица. Поэтому минимальное из возможных выводимых чисел будет равно 1 (0+1), а максимальное будет равно 9 (8+1).

Текст программы, использующей генератор случайных чисел, приведен в листинге 5.3.

Листинг 5.3. Компьютерная игра по угадыванию задуманного компьютером числа

```
program igra;
Uses  Crt;
 var
    x,y,n:integer;
begin
 ClrScr;
 Randomize;
 n:=1; x:=random(19)+1;
 Textcolor(cyan);
 writeln('Компьютер задумал целое число (от 1 до 20)');
 writeln('попробуйте угадать это число (всего у Вас будет 5 попыток)');
 repeat
    Textcolor(green);
    writeln(n,' попытка ');
    writeln('Введите число и нажмите клавишу Enter');
    readln(y);
    n:=n+1
until (x=y) or (n>5);
if x=y then
    begin
    Textcolor(lightred);
    writeln('Поздравляю Вас с выигрышем')
    end
       else
    begin
    Textcolor(lightblue);
```

```
    writeln ('Вы проиграли. Задуманное компьютером число равно ',x)
    end;
 readln
end.
```

В данной программе после очистки экрана производится инициализация генератора случайных чисел. Затем командой `random` берется случайное число, которое присваивается целочисленной переменной `x`. Это и будет число, задуманное компьютером, но неизвестное заранее пользователю, которое он и должен угадать. В процессе угадывания числа компьютер должен фиксировать согласно условиям игры сделанное пользователем количество попыток. Это количество попыток будет содержать другая целочисленная переменная `n`. Так как попытки мы будем считать, начиная с первой, а не с нулевой, то мы присвоим переменной `n` начальное значение, равное единице.

Сам процесс угадывания числа будет производиться в цикле с постусловием. Тело цикла включает в себя следующие операции. В начале пользователю оператором вывода предлагается ввести предполагаемое значение `x`. Для справки указывается номер попытки, который будет равен текущему значению переменной `n`. Предполагаемое значение `x` вводится в компьютер оператором `readln` и присваивается переменной `y`. Последним оператором тела цикла является оператор присваивания, который увеличивает текущее значение переменной `n` на единицу, для того чтобы при следующем повторении тела цикла (если таковое будет иметь место) правильно фиксировался номер попытки.

Постусловие, которое определяет, нужно ли следующее повторение тела цикла, является сложным и состоит из двух простых условий. Необходимость повторения тела цикла отпадает, во-первых, в том случае, если пользователь угадал число, т. е. введенное им предполагаемое значение `y` совпало с действительным значением `x`. Равенство значений этих двух переменных и будет одним из возможных условий окончания цикла. Второй вариант окончания работы цикла заключается в том, что пользователь исчерпал имевшееся у него число попыток. В этом случае после завершения последней неудачной попытки (пятой по счету) переменной `n` будет присвоено новое значение, равное 6 (5+1). Таким образом, вторым условием окончания цикла будет превышение переменной `n` значения 5, т. е. `n>5`. Оба этих условия заключаются согласно правилам языка Паскаль в скобки и объединяются операцией `or` (логическое Или), т. е. работа цикла завершится в том случае, если будет выполняться или одно, или другое условие.

По завершении работы цикла остается выяснить, было ли все-таки угадано загаданное компьютером число, и вывести для пользователя соответствующее сообщение. Для такой проверки используется условный оператор `if`, в заголовке которого указывается проверяемое условие — это опять-таки ра-

венство значений переменных x и y. Если эти значения на данный момент равны, остается поздравить пользователя с победой и вывести сообщение об этом оператором вывода. Сообщение о победе выводится в ветви программы, находящейся после служебного слова then. Если же значения переменных не совпадают, то нужно сообщить пользователю, что он проиграл и проинформировать его о том, какое число было задумано на самом деле. Эту задачу выполняют операторы, находящиеся в условном операторе после служебного слова else.

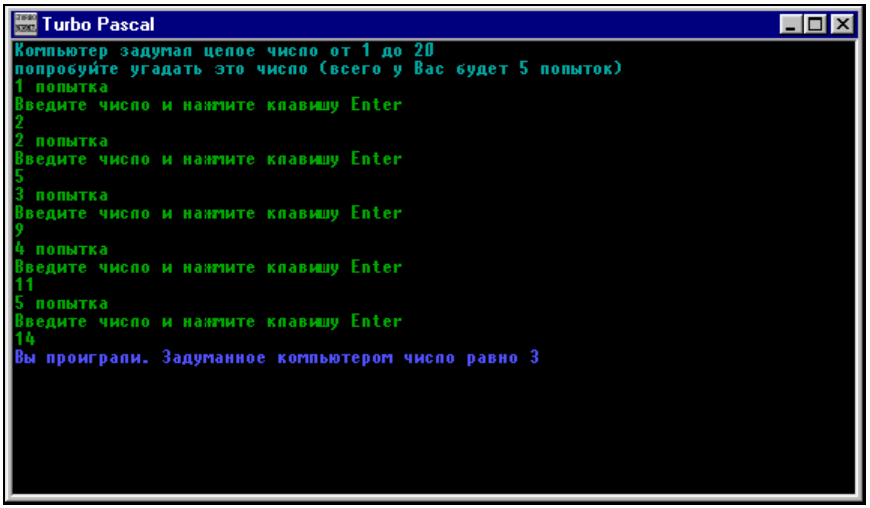
Для того чтобы процесс игры для пользователя был менее монотонным, мы "раскрасили" выводимую программой информацию в разные цвета с помощью операторов Textcolor. Так начальное сообщение об условиях игры выводится на экран бирюзовым цветом, информация о номере очередной попытки и приглашение на ввод пользователем очередного числа выводится зеленым цветом. В том случае, если пользователь выиграл, информация о победе выводится светло-красным цветом, а в случае проигрыша соответствующая информация и задуманное число — голубым цветом. На рисунках приведены два возможных варианта игры: в первом случае игру выиграл компьютер (рис. 5.4, *а*), во втором — пользователь (рис. 5.4, *б*).

Еще одним заслуживающим внимание применением цикла с постусловием является защита программы от неправильного ввода данных. В *главе 4* уже было показано, как подобную защиту можно производить с помощью оператора безусловного перехода, но более правильным считается использование оператора цикла, т. к. при этом не происходит чрезмерного усложнения структуры программы.

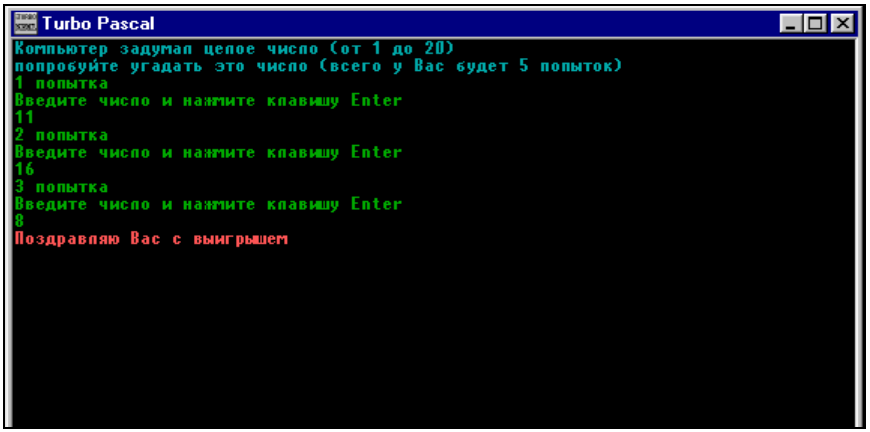
В качестве примера рассмотрим уже составленную ранее программу, моделирующую работу калькулятора, рассмотренную ранее в *разд. 4.3*. В первоначальном варианте этой программы отсутствует защита от неправильного ввода данных при выборе в меню одного из возможных вариантов действий. Напомним, что для выбора одного из действий, которое можно произвести над двумя введенными числами, в этой программе нужно ввести целое число от 1 до 4.

Для защиты программы от неверного ввода данных мы используем цикл с постусловием. Телом этого цикла будут операторы вывода, сообщающие пользователю о том, какие числа можно вводить для выбора вариантов дальнейших действий, и оператор ввода, который присваивает значение переменной x, являющейся переменной-селектором, используемой для выбора варианта.

```
writeln('Выберите действие, производимое над числами');
writeln('(введите соответствующую цифру)');
writeln ('1 - сложение, 2 - вычитание, 3 - умножение, 4 - деление');
readln(x)
```



а



б

Рис. 5.4. Возможные варианты игры "угадай число"

Это тело цикла будет повторяться до тех пора, пока не будет выполнено одно из условий, находящихся в завершающей строке цикла:

```
(x=1) or (x=2) or (x=3) or (x=4)
```

Тогда весь оператор цикла будет выглядеть следующим образом:

```
repeat
 writeln('Выберите действие, производимое над числами');
 writeln('(введите соответствующую цифру)');
 writeln ('1 - сложение, 2 - вычитание, 3 - умножение, 4 - деление');
 readln(x)
until (x=1) or (x=2) or (x=3) or (x=4);
```

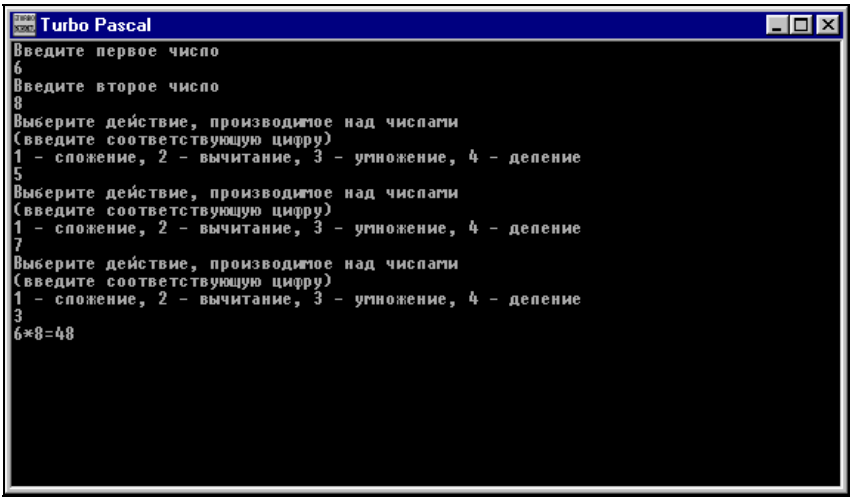


Рис. 5.5. Результаты работы программы-калькулятора с защитой от неправильного ввода данных

Теперь, если пользователь попробует ввести при выборе варианта в меню число 5 или 7, программа вновь будет возвращать его к вводу данных. Целиком же усовершенствованная программа будет выглядеть следующим образом — листинг 5.4.

Листинг 5.4. Программа-калькулятор с защитой от неправильного ввода данных

```
program kalk1;
Uses Crt;
var a,b,x:integer; d:real;
begin
ClrScr;
writeln('Введите первое число');
readln(a);
writeln('Введите второе число');
readln(b);
repeat
 writeln('Выберите действие, производимое над числами');
 writeln('(введите соответствующую цифру)');
 writeln ('1 - сложение, 2 - вычитание, 3 - умножение, 4 - деление');
 readln(x)
until (x=1) or (x=2) or (x=3) or (x=4);
case x of
1: writeln(a,'+',b,'=',a+b);
2: writeln(a,'-',b,'=',a-b);
```

```
3: writeln(a,'*',b,'=',a*b);
4: begin d:=a/b; writeln(a,'/',b,'=',d:7:3) end
end;
readln
end.
```

На рис. 5.5 показан результат работы программы-калькулятора при попытке ввести неправильные данные и полученный результат вычислений при правильном вводе.

5.3. Цикл с предусловием

Общий вид оператора `while` следующий:

```
while условие do
тело_цикла;
```

где `while` и `do` — служебные слова, `while` означает "пока", `do` — "делать, выполнять".

Цикл работает следующим образом: первоначально проверяется условие, находящееся после слова `while`, если данное условие является ложным, то работа цикла на этом завершается. Если условие является истинным, то выполняется оператор или группа операторов, входящая в тело цикла. Затем снова производится проверка условия, содержащегося в строке заголовка и т. д. Таким образом, тело цикла выполняется пока истинно условие, содержащееся в заголовке цикла. Цикл такого типа называют циклом с предусловием. Это объясняется тем, что перед первым же выполнением цикла производится проверка истинности находящегося в заголовке условия. Если условие изначально не выполняется, то тело цикла не будет выполнено ни разу.

В качестве примера использования цикла типа `while` приведем программу разложения целого положительного числа на простые множители. Напомним, что простым называется целое число (кроме единицы), которое делится только на единицу и само на себя. Составным или сложным называется число, которое делится еще и на другие числа. Любое составное число можно представить единственным образом в виде произведения простых чисел, которые и называются простыми множителями.

Наиболее простой (хотя и достаточно трудоемкий для человека, но не для компьютера) алгоритм разложения числа на множители заключается в следующем. Исходное число делится на все числа, начиная с 2. Если число нацело разделилось на делитель, то этот делитель является одним из искомых простых множителей. Тогда мы определяем частное от деления этих двух чисел и делим уже его на все числа, начиная с 2. Если же исходное число или

последнее получившееся частное не делится нацело на делитель, то значение делителя увеличиваем на единицу и уже это новое значение используем при дальнейшей работе. Так мы будем действовать до тех пор, пока значение делителя не станет равным частному или самому числу (в том случае, если простых множителей для исходного числа не было найдено). На этом поиск простых множителей завершается.

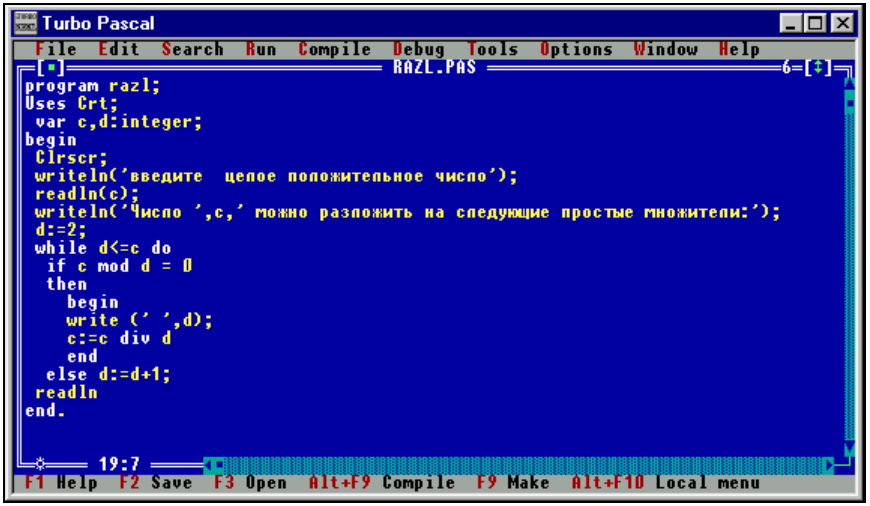
В начале программы оператором вывода пользователю предлагается ввести с клавиатуры исходное число `c`, которое вводится оператором `readln`. Затем следующий оператор вывода сообщает о том, что далее будет производиться вывод простых множителей. Для этого мы будем делить введенное число на некоторое число `d`. Если число `c` разделится на число `d` без остатка, то `d` будет одним из искомых множителей, на которые нужно разложить `c`. Деление без остатка подразумевает, что остаток от деления будет равен нулю, т. е. должно выполняться соотношение `c mod d = 0`. Начинать поиск делителей мы начнем с наименьшего возможного — числа 2. Поэтому переменной `d` мы присвоим начальное значение 2.

Поиск простых множителей мы будем производить в цикле. Этот цикл с предусловием будет работать до тех пор, пока значение делителя `d` будет меньше или равно делимому `c`. Как только `d` станет больше `c`, работа цикла прекратится, т. к. дальнейший поиск не имеет смысла. В теле цикла первым оператором будет условный оператор, проверяющий соотношение `c mod d = 0`. Если данное соотношение истинно, то текущее значение `d` является одним из искомых простых множителей, и тогда нужно будет выполнить две операции. Во-первых, надо вывести на экран компьютера текущее значение переменной `d`. Во-вторых, для того чтобы продолжить дальнейший поиск простых множителей, нужно найти частное от деления `c` на `d`. Это частное мы найдем посредством операции целочисленного деления `c div d`. Результат этой операции мы присвоим переменной `c`, и в дальнейшем будем работать с этим новым ее значением, ища для него возможные делители. Указанные ранее операции находятся в той ветви условного оператора, которая расположена после слова `then`. Если же `c` не разделилось на `d` без остатка, то работает ветвь условного оператора, которая расположена после `else`. В этом случае текущее значение делителя `d` увеличивается на единицу, и при новом выполнении тела цикла будет использоваться это новое значение делителя. В листинге 5.5 приводится текст данной программы.

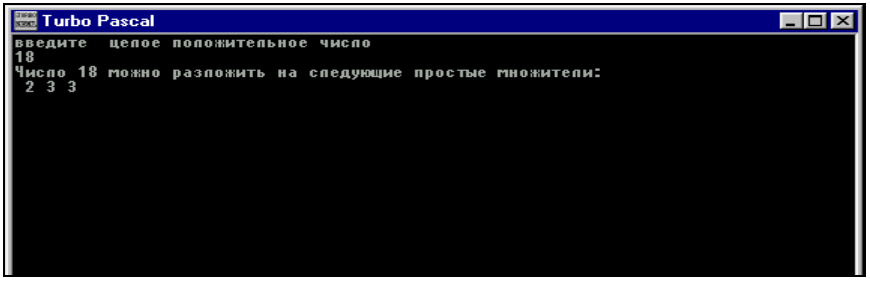
Листинг 5.5. Программа разложения числа на простые множители

```
program razl;
Uses Crt;
 var c,d:integer;
```

```
begin
 Clrscr;
 writeln('введите  целое положительное число');
 readln(c);
 writeln('Число ',c,' можно разложить на следующие простые множители:');
 d:=2;
 while d<=c do
  if c mod d = 0
  then begin write (' ',d); c:=c div d  end
  else d:=d+1;
 readln
end.
```



а



б

Рис. 5.6. Программа разложения числа на простые множители *(а)* и результаты ее работы *(б)*

Таким образом мы переберем все возможные значения простых множителей и выведем на экран те из них, которые действительно являются таковыми.

Если исходное число является простым, то в результате работы программы на экран будет выведен единственный результат — само исходное число. Если же число является составным, то простых множителей будет как минимум два. Результат работы программы показан также на рис. 5.6.

Задания для самостоятельной работы

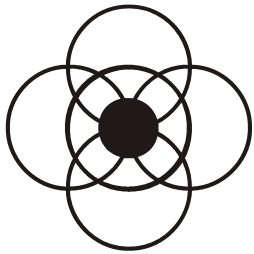
Задание 1. В университете формируется баскетбольная команда, членами которой могут стать студенты, рост которых составляет не менее 175 сантиметров. Составить программу, которая позволила бы тренеру команды определить, сколько потенциальных кандидатов в команду имеется в студенческой группе, насчитывающей 25 человек. Данные о росте каждого из студентов группы вводятся в компьютер с клавиатуры.

Задание 2. Клиент положил в банк сроком на 5 лет некоторую денежную сумму под определенный процент. Составить программу, которая бы определяла, какую величину составит сумма вклада по истечении срока хранения данного вклада в банке, если по условиям договора о вкладе, заключенного между клиентом и банком, в течение всего срока хранения вклада банковский процент не должен изменяться. Сумма вклада и процент вводятся в компьютер с клавиатуры.

Задание 3. Известна легенда о том, как одному могущественному индийскому радже некий мудрец оказал важную услугу. В награду за услугу раджа был готов выполнить любое желание мудреца. Мудрец попросил положить перед ним шахматную доску и на первую клетку доски положить одно зерно риса, на вторую клетку два зерна, на третью клетку четыре зерна и т. д., т. е. на каждую следующую клетку должно было быть положено вдвое больше зерен, чем на предыдущую, и таким образом должны были быть заполнены все 64 клетки шахматной доски. Как оказалось при подсчете, для выполнения просьбы мудреца не хватило бы не только всех запасов риса во владениях раджи, но и всего урожая риса на всем земном шаре. Составить программу, которая определила бы, на какой по счету клетке шахматной доски количество зерен риса должно было превысить один миллион, в случае если просьба мудреца была бы выполнена.

Задание 4. Определить, исходя из условий, поставленных в предыдущем задании, сколько клеток шахматной доски нужно было бы заполнить для того, чтобы суммарное число зерен на них превысило бы один миллиард.

ГЛАВА 6



Работа с текстом в Паскале

До сих пор мы рассматривали программы, в которых производилась обработка только числовых данных (программы из *разд. 3.1* не являются исключением, т. к. в них мы имели дело с текстовыми данными, но при этом производился их вывод, а не обработка). Но современный компьютер, как известно, представляет собой устройство, способное работать не только с числовой информацией, но и с другими ее видами (текстовой, графической, аудио и видеоинформацией). В языке Паскаль также предусмотрена возможность обработки текстовых данных. Для работы с ними используются типы `char` и `string`, которые соответственно применяются для работы с данными символьного и строкового типов.

6.1. Символьные величины

В программах, написанных на Паскале, широко используются символьные константы и переменные. Значениями как тех, так и других являются символы. Символами называются все буквы и цифры, пробел, знаки препинания, которые можно вводить с клавиатуры компьютера.

Как уже говорилось в начале данной книги, центральный процессор компьютера может работать только с информацией, представленной в численном виде. Любая вводимая в компьютер информация другого вида должна быть обязательно преобразована в числовую форму или, как говорят, закодирована с помощью соответствующих чисел. Так обстоит дело и при вводе в компьютер и дальнейшей обработке в нем текстовой информации. Фактически компьютер работает не с буквами, пробелами, точками и запятыми, а соответствующими им числовыми кодами. Когда пользователь при работе на компьютере использует готовые программы, такие как текстовые редакторы, электронные словари и переводчики, то ему не обязательно знать механизм

обработки текстовых данных внутри компьютера. Программисту же необходимо представлять себе взаимосвязь между символом, отображаемым на экране компьютера, и хранящимся в памяти компьютера его числовым кодом.

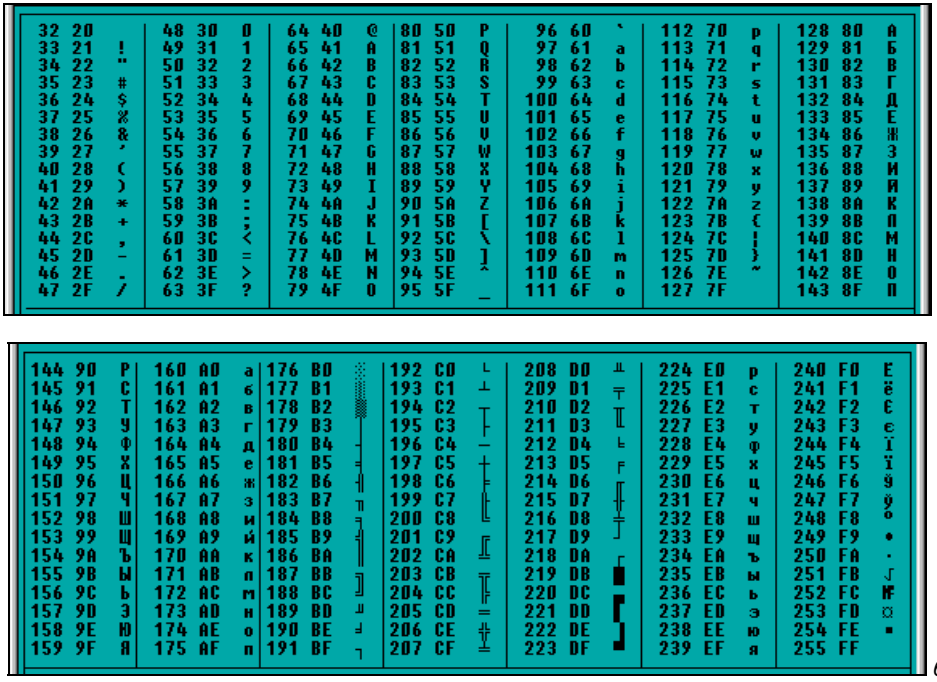
Понятно, что если каждый программист будет изобретать свои собственные числовые коды для используемых в компьютере символов, то применять созданные им программы для обработки текстов сможет только он сам. Поэтому возникла необходимость в создании единой, общепринятой системы представления символов числовыми кодами. Такая система была реализована в виде *кодовой таблицы ASCII*.

ASCII — это аббревиатура от American Standart Code for Information Interchange, что в переводе означает "американский стандартный код для обмена информацией". Это используемая во всем мире таблица для кодирования вводимой в компьютер текстовой информации. Данная таблица содержит буквы латинского алфавита (прописные и строчные), цифры от 0 до 9, символы арифметических операций и знаки препинания, а также различные специальные символы. Каждому из имеющихся в таблице символов соответствует свой числовой код. Например, прописной букве "N" латинского алфавита соответствует числовой код 78, строчной букве "n" — код 110 (строчные и прописные буквы имеют разные коды). Цифре "6" соответствует код 54, пробелу — код 32, запятой — код 44 и т. д. Кроме того, кодовая таблица содержит так называемые управляющие символы, которые при вводе не отображаются непосредственно на экране компьютера, но при этом выполняют определенные действия, например, производят перевод курсора на следующую строку при работе с текстом или выдают звуковой сигнал. Каждому из управляющих символов также соответствует числовой код. Всего эта таблица содержит 128 символов. Данная таблица является единой для всех IBM-PC совместимых компьютеров.

К основной таблице ASCII существует дополнение, также содержащее 128 символов. Эта вторая половина таблицы зарезервирована для символов национальных алфавитов и, соответственно, для разных стран выглядит поразному. В России во второй половине содержатся, естественно, прописные и строчные буквы кириллицы (русского алфавита). Помимо символов кириллицы в этой части таблицы содержатся так называемые псевдографические символы, которые не имеют соответствующих им клавиш на клавиатуре компьютера, но могут быть выведены на экран путем набора их числового кода при нажатой клавише <Alt>. Содержит эта часть таблицы и некоторые специальные символы, отсутствующие в первой, международной части. К их числу, например, относится радикал (√ — знак квадратного корня).

Следовательно, всего расширенная таблица, состоящая из международной и национальной части, содержит 256 символов. Обычно числовой код записывают в привычной всем нам десятичной системе счисления, но в ряде случаев

для его записи применяют и другую, шестнадцатеричную систему счисления, в которой помимо цифр от 0 до 9 используют также буквы латинского алфавита от "A" до "F", обозначающие числа от 10 до 15. На рис. 6.1 приведены символы, входящие в таблицу ASCII, и соответствующие каждому из этих символов десятичный и шестнадцатеричный числовой код.



а

б

Рис. 6.1. Символы кодовой таблицы ASCII
(*а* — первая половина таблицы, *б* — вторая половина)

В каждом из столбцов первая запись слева обозначает десятичный код символа, вторая запись — код этого же символа, записанный в шестнадцатеричной системе счисления, широко применяемой в программировании. Справа в столбце изображен сам символ.

Знание числовых кодов символов понадобится нам в дальнейшем при составлении различных программ по обработке текстов, но большинство символов, которые не являются псевдографическими и управляющими, можно использовать в программе непосредственно в виде символьных констант. Употребляемые в программе символьные константы записываются так:

```
'A'   'y'   '?'   'щ'
```

т. е. запись символьной константы состоит из символа, заключенного в апострофы.

Если в вашей программе задействована символьная переменная, она должна быть предварительно указана в разделе описаний подобно числовым переменным. Описание символьной переменной выглядит следующим образом:

```
var имя_переменной:char;
```

где `char` — служебное слово, обозначающее тип данной переменной. `char` — это сокращение от английского слова character, означающего в переводе "символ".

Например, если в программе в разделе описания переменных мы видим:

```
var i,n: integer;
      d: real;
      sim,bukv: char;
```

то в данной программе используются две переменных целого типа (`i` и `n`), одна — вещественного типа (`d`), и две — символьного типа (`sim` и `bukv`). Если переменной символьного типа присваивается какое-либо значение, то оно также должно быть заключено в апострофы. Например:

```
sim:='ш';
```

означает, что переменной `sim` было присвоено значение буквы "ш". Возможен в программе и такой оператор присваивания:

```
cfr:='7';
```

наличие в программе такой записи говорит о том, что переменной `cfr` присваивается значение символа "7" (а не числа 7, как было бы в случае с числовой переменной).

Подобно обычным числам символьные величины можно сравнивать. При этом фактически сравниваются не сами эти величины, а соответствующие им числовые коды. Например, можно записать такую операцию сравнения:

```
'л'<'н'
```

результат этой операции будет истинным (равен `true`), т. к. код символа "л" равен 171, а код символа "н" — 173. А если мы запишем следующее соотношение:

```
'к'>'т'
```

то такое соотношение будет ложным (`false`), т. к. код символа "к" равен 170, а символа "т" — 226. Можно проверять символьные величины также на равенство или неравенство друг другу. Знаки для операций сравнения символьных величин используются те же, что и для числовых величин.

В языке Паскаль для работы с символьными переменными используются специальные стандартные функции `chr` и `ord`, которые мы подробно рассмот-

рим в соответствующей главе книги. Пока же достаточно знать то, что стандартная функция по известной исходной величине, которая называется аргументом и заключается в скобки, находящиеся после имени функции, определяет некоторое значение.

Функция `chr` по известному числовому коду символа определяет сам этот символ. Общий вид функции:

```
chr(k)
```

где `k` — десятичный числовой код символа. Например, значение `chr(33)` будет равно `'!'`, `chr(233)` — `'щ'` и т. д.

Составим программу, которая будет выводить соответствующую группу символов для заданного диапазона числовых кодов. Числовые коды находящихся в таблице `ASCII` символов могут изменяться от 0 до 255. Правда, символы с кодами от 0 до 31 являются управляющими и правильно отображаться на экране компьютера не будут. Поэтому условимся, что начальный код диапазона должен быть не менее 32, а конечный не более 255. Кроме того, для правильной работы программы необходимо, чтобы начальный код диапазона был меньше, чем конечный.

Для защиты от неправильного ввода данных используем следующий прием. После операторов, осуществляющих ввод исходных данных, ставим сокращенный условный оператор `if`. В условии, находящемся в заголовке этого оператора, с помощью операций `or` объединены все перечисленные ранее условия. Если хотя бы одно из этих условий будет нарушено, то с помощью оператора безусловного перехода управление передается на начало программы, и пользователь заново должен ввести исходные данные, соответствующие указанным в программе условиям. Если пользователь ввел правильные исходные данные, то программа переходит к следующим операторам.

Далее в программе осуществляется непосредственно процесс вывода символов. Этот процесс организован в виде цикла с заранее известным числом повторений. Начальным значением переменной цикла `i` является введенный пользователем начальный код диапазона, конечным значением переменной соответственно — конечный код диапазона. Тело цикла в данной программе состоит всего из одного простого оператора. Это оператор вывода, который выводит на экран компьютера очередное значение функции `chr`, аргументом которой является текущее значение переменной цикла `i`. При каждом выполнении тела цикла на экран компьютера выводится символ, код которого на единицу больше кода предыдущего выведенного символа. Чтобы вывести результат работы программы в одной строчке, используем в теле цикла оператор `write`. Приводим текст данной программы.

Листинг 6.1. Программа, выводящая группу символов из заданного диапазона

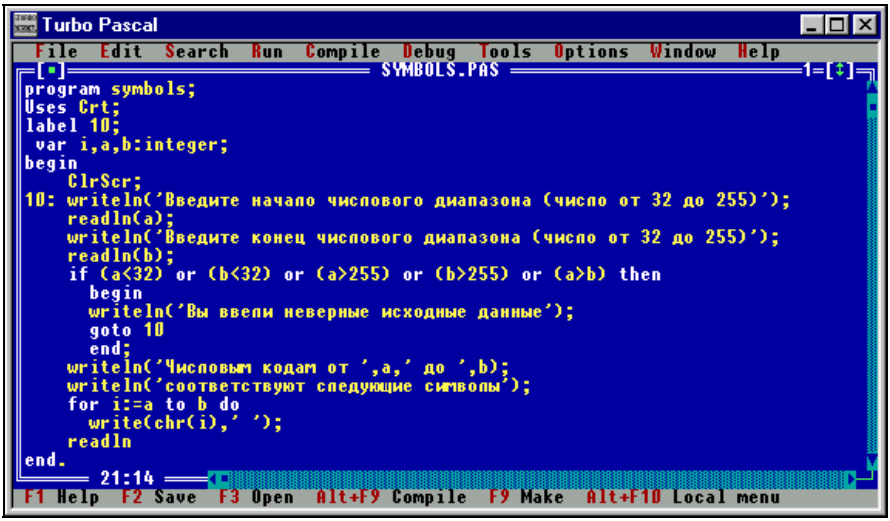
```
program symbols;
Uses Crt;
label 10;
 var i,a,b:integer;
begin
    ClrScr;
10: writeln('Введите начало числового диапазона (число от 32 до 255)');
    readln(a);
    writeln('Введите конец числового диапазона (число от 32 до 255)');
    readln(b);
    if (a<32) or (b<32) or (a>255) or (b>255) or (a>b) then
      begin
      writeln('Вы ввели неверные исходные данные');
      goto 10
      end;
    writeln('Числовым кодам от ',a,' до ',b)
    writeln('соответствуют следующие символы');
    for i:=a to b do
      write(chr(i),' ');
    readln
end.
```

Для диапазона числовых кодов, начальным из которых является 176, а конечным — 215, результаты работы программы показаны на рис. 6.2. В итоге работы программы для данного диапазона мы получили строку символов псевдографики. *Псевдографическими символами* называются различные вертикальные и горизонтальные линии, уголки, перекрестия линий, прямоугольники, которые используются для создания рамок, таблиц, несложных рисунков и тому подобных объектов.

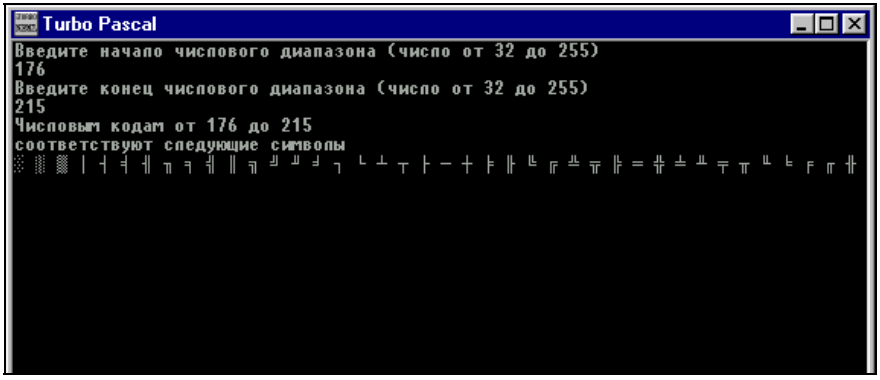
Действие функции `ord` противоположно функции `chr`. Аргументом данной функции является символ, а значением — десятичный код данного символа. Общий вид функции:

```
ord('s')
```

где `s` — символ, код которого мы определяем. Обратите внимание, что обрабатываемый функцией символ должен быть обязательно заключен в кавычки, иначе при компиляции программы с использованием данной функции будет выдано сообщение об ошибке. Поэтому запись функции должна выглядеть в программе следующим образом `ord('ж')`, `ord('ф')` и т. д. Для символа "ж" функция выдаст код 166, для символа "ф" — 228.



а



б

Рис. 6.2. Программа, выводящая символы кодовой таблицы ASCII *(а)*, и результаты ее работы *(б)*

Подобно целочисленной переменной, символьную переменную можно использовать в качестве селектора в операторе множественного выбора. Принцип использования символьной переменной тот же, что и для целочисленной переменной, только вместо числовых значений перед описанными в операторе вариантами действий ставятся любые символы.

Однако при использовании символьных переменных в операторе `case...of` необходимо учитывать один нюанс. Когда в операторе множественного выбора перед одним из вариантов указывается соответствующее ему значение символьной переменной, то это значение должно быть обязательно заключено в апострофы. Как может выглядеть такой оператор `case...of`, хорошо видно на следующем примере.

```
case letter of
'п':  writeln('привет!');
'з':  writeln('здравствуйте!');
'д':  writeln('добрый день!');
end;
```

В зависимости от значения, которое присваивается символьной переменной `letter`, компьютер будет выводить на экран различные варианты приветствия. Если символьной переменной присвоено значение "п", то на экране мы увидим слово "привет!", если значение будет равно "з", то компьютер напишет на экране слово "здравствуйте!", если значение будет равно "д", то компьютер скажет пользователю "добрый день!". Естественно, что для правильной работы данного оператора переменная `letter` должна быть описана в начале программы как переменная типа `char`.

Описанные ранее возможности символьных переменных используются в следующей задаче. Требуется составить программу, которая осуществляет ввод двух целых чисел. Над введенными числами по выбору пользователя может осуществляться одно из следующих действий: определение среднего арифметического, наибольшего общего делителя и среднего гармонического. Выбор варианта действия должен производиться путем нажатия соответствующей буквы: "а" — для среднего арифметического, "б" — для наибольшего общего делителя, "в" — для среднего гармонического.

Напомним читателю, что средним гармоническим чисел m и n называется число s, которое удовлетворяет следующему равенству:

$$\frac{1}{s}=\frac{1}{2}\cdot\left(\frac{1}{m}+\frac{1}{n}\right).$$

Путем несложных математических преобразований можно определить, что среднее гармоническое вычисляется по следующей формуле:

$$s=\frac{2}{\frac{1}{m}+\frac{1}{n}}.$$

В листинге 6.2 приводится текст программы, которая решает поставленную задачу.

Листинг 6.2. Программа, определяющая по выбору пользователя среднее арифметическое, наибольший общий делитель или среднее гармоническое

```
program charcase;
uses crt;
```

```
label 10;
var i,k,m,n,d:integer; rez:real; ch,p:char;
begin
10: ClrScr;
    writeln('Введите два числа');
    readln(m);
    readln(n);
    writeln('над введенными числами можно выполнить следующие операции');
    writeln('а - определение среднего арифметического');
    writeln('б - определение наибольшего общего делителя');
    writeln('в - определение среднего гармонического');
    writeln('для выбора операции нажмите соответствующую букву');
    readln(ch);
    case ch of
      'а': begin
           rez:=(m+n)/2;
           writeln('среднее арифметическое равно ',rez:6:2)
           end;
 'б': begin
        d:=1;
        if m<n
        then k:=m
        else k:=n;
        for i:=1 to k do
          if (m mod i =0) and (n mod i =0)
          then d:=i;
        writeln('наибольший общий делитель двух чисел равен ',d)
        end;
         'в':  begin
                rez:=2/(1/m+1/n);
                writeln('среднее гармоническое равно ',rez:7:3)
                end;
    end;
    writeln('будете продолжать вычисления (д/н)');
    readln(p);
    if p='д'
    then goto 10
    end.
```

Разберем текст данной программы. После команды очистки экрана с помощью операторов `readln` производится ввод двух целых чисел `m` и `n`. Затем на экран компьютера с помощью операторов `writeln` выводится текст меню,

который сообщает пользователю о том, какие действия можно выполнить над введенными числами и какому действию соответствует нажатие какой буквы. Введенная пользователем буква присваивается символьной переменной ch с помощью оператора ввода readln.

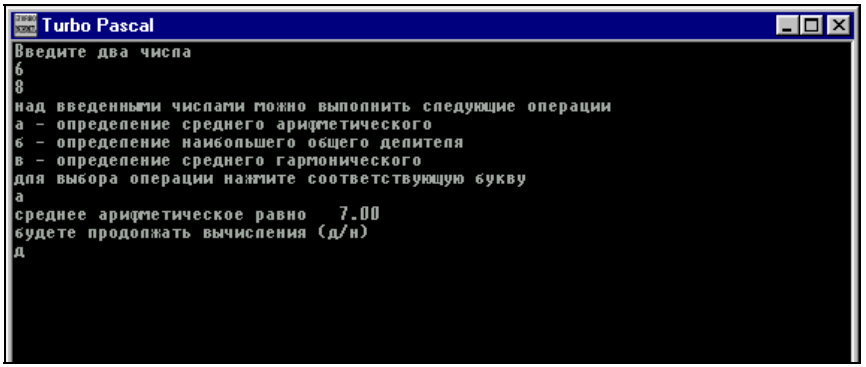
Далее переменная ch используется в качестве переменной-селектора в операторе множественного выбора case...of. Оператор множественного выбора включает в себя в данном случае три возможных варианта действия, соответствующих трем возможным значениям переменной-селектора. Вычисление среднего арифметического и среднего гармонического производится по соответствующим формулам, а полученный результат присваивается переменной rez, значение которой затем выводится на экран компьютера.

Несколько сложнее производится определение наибольшего общего делителя двух чисел d. Как известно, наибольшим общим делителем двух чисел m и n называется такое число d, на которое без остатка делятся оба эти числа. Понятно, что наибольший общий делитель не может быть больше, чем меньшее из двух исходных чисел. Поэтому в программе с помощью условного оператора if определяется меньшее из двух исходных чисел. Затем этот минимум присваивается переменной k. Таким образом, значение переменной k — это максимально возможное значение наибольшего общего делителя.

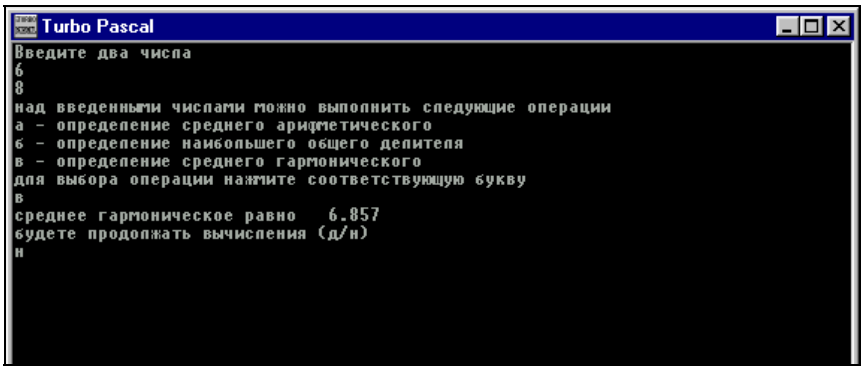
Затем для нахождения наибольшего общего делителя используется цикл с заранее известным числом повторений. В этом цикле перебираются все возможные значения наибольшего общего делителя, начиная с единицы и заканчивая k. Каждое из этих значений поочередно присваивается переменной цикла i. Затем проверяется, является ли текущее значение переменной i общим делителем для m и n, т. е., делятся ли одновременно оба этих числа без остатка на i. Для обеспечения совместной проверки двух условий используется логическая операция and. Если оба условия делимости выполняются одновременно, то текущее значение i присваивается переменной d. Затем переменной i присваивается новое значение, которое на единицу больше предыдущего, и для него осуществляется аналогичная проверка. Таким образом, в результате работы цикла в переменной d будет содержаться максимальное значение общего делителя двух чисел, т. е. искомая величина, которая затем выводится на экран компьютера.

В заключительной части программы компьютер запрашивает пользователя о том, хочет ли он продолжать вычисления. Для ответа пользователь должен ввести букву "д" или "н". Если пользователь дал положительный ответ ("д"), то управление программой передается на ее начало, обозначенное меткой 10. В результате производится очистка экрана и затем пользователю вновь предлагается ввести два числа, над которыми можно произвести одну из трех указанных ранее операций. Если пользователь дал отрицательный ответ, то про-

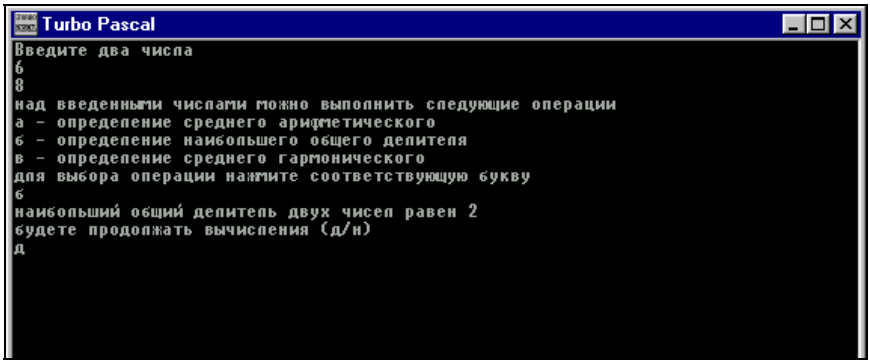
грамма завершает свою работу. На рис. 6.3 представлены результаты работы программы для одной и той же пары исходных чисел при выборе различных вариантов действий.



а



б



в

Рис. 6.3. Результаты работы программы `charcase` при вычислении среднего арифметического двух чисел *(а)*, наибольшего общего делителя *(б)* и среднего гармонического *(в)*

6.2. Строковые величины

Естественно, что для эффективной работы с текстом необходимо уметь обрабатывать не только отдельные символы, но и слова, фразы, строки, т. е. группы символов. Для обработки таких групп символов в Паскале используется строковый тип данных. Для описания символьных переменных применяется служебное слово `string`. Описание символьной переменной в общем виде выглядит следующим образом:

```
var имя_переменной: string[длина];
```

после слова `string` указывается длина описываемой переменной, т. е. максимальное количество символов, которое может содержать данная переменная. Например, если в разделе описания переменных в программе мы видим:

```
var i:integer;
    x:real
    stroka:string[25];
```

то это означает, что в данной программе наряду с переменными целого типа `i` и вещественного типа `x` используется и переменная строкового типа `stroka`, которая может содержать до 25 символов. Параметр, заключенный в квадратные скобки, не является обязательным и используется в том случае, если существуют причины, по которым необходимо жестко ограничить максимальную длину строковой переменной. Если же после имени переменной ее длина не указывается, то по умолчанию она считается равной 255 символам. В этом случае описание символьной переменной будет выглядеть так:

```
var stroka:string;
```

Значения символьных переменных так же, как и уже знакомые нам символьные константы (к текстовым константам относятся, в частности, сообщения, выводимые оператором `writeln`), должны обязательно заключаться в апострофы. Таким образом, оператор присваивания строковой переменной `stroka` значения "набор_символов" будет выглядеть следующим образом:

```
stroka:= 'набор_символов';
```

Подобно числовым переменным, строковые переменные можно складывать. Результат сложения также можно присваивать какой-либо символьной переменной. Например, если значение переменной `slovo1` равно "Турбо ", а переменной `slovo2` — "Паскаль", то в результате операции:

```
slovo:=slovo1+slovo2;
```

значение переменной `slovo` станет равным словосочетанию "Turbo Pascal". Можно прибавлять к строковой переменной текстовую константу.

Например, если в программе имеется следующий оператор:

```
fraza:=slovo + " версия 7.0";
```

то в результате значение строковой переменной `fraza` станет равным "Turbo Pascal версия 7.0".

Рассмотрим работу со строковыми величинами на примере программы, которая спрашивает у пользователя как его зовут, а затем здоровается с ним и желает ему успехов в изучении программирования. В листинге 6.3 приводится текст данной программы.

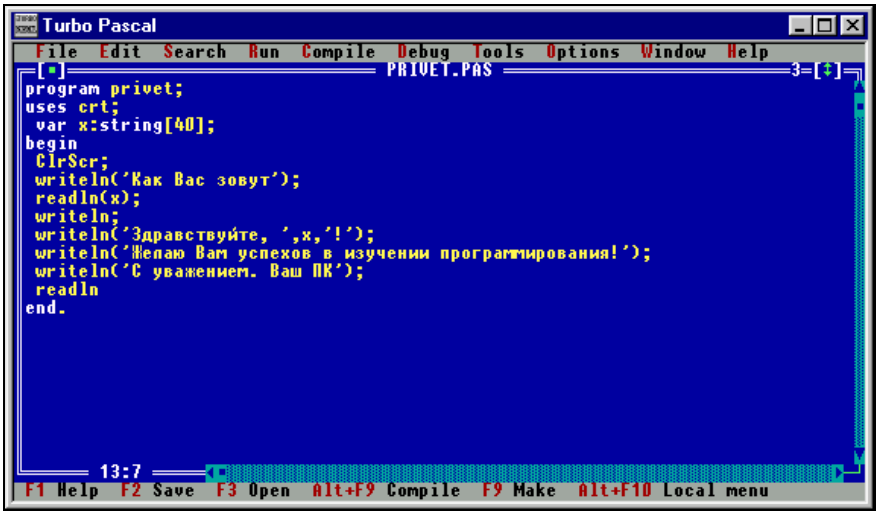
Листинг 6.3. Программа, которая приветствует пользователя

```
program privet;
uses crt;
 var x:string[40];
begin
 ClrScr;
 writeln('Как Вас зовут');
 readln(x);
 writeln;
 writeln('Здравствуйте, ',x,'!');
 writeln('Желаю Вам успехов в изучении программирования!');
 writeln('С уважением. Ваш ПК');
 readln
end.
```

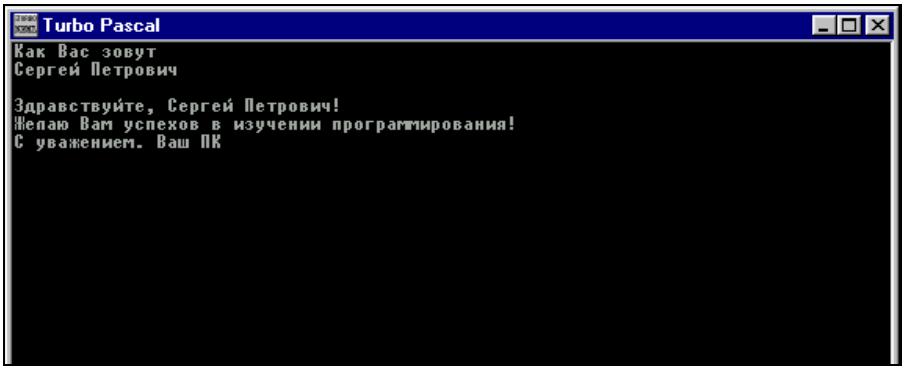
В данной программе в разделе описания переменных присутствует только одна переменная `x` строкового типа. В основной части программы с помощью оператора `writeln` пользователю задается вопрос о его имени. Затем введенное с клавиатуры имя с помощью оператора `readln` присваивается строковой переменной `x`. Максимальная длина строковой переменной принята равной 40, т. к. даже если пользователь в ответ на вопрос компьютера введет полностью свое имя, отчество и фамилию, то суммарная длина их не превысит 40 символов. Затем в программе ставится "пустой" оператор `writeln` для того, чтобы между ответом пользователя и последующим сообщением компьютера оставалась чистая строка.

Затем на экран выводится приветствие компьютера. В операторе `writeln` список вывода состоит из трех элементов. Первый — это заключенное в апострофы слово "Здравствуйте", т. е. текстовая константа. Второй — это значение строковой переменной `x`, т. е. введенное пользователем имя. Третий — это заключенный в апострофы восклицательный знак. В следующих строках про-

граммы операторами `writeln` завершается вывод текста приветствия. На рис. 6.4. показан экран с текстом программы и пример ее работы.



а



б

Рис. 6.4. Текст программы, которая спрашивает у пользователя его имя, а затем здоровается с ним *(а)*, и результат ее работы *(б)*

Для работы со строковыми переменными в языке Паскаль используются различные функции, среди которых отметим функцию `length`. Аргументом данной функции является имя какой-либо символьной переменной, а значением — фактическое количество символов, которое содержится в данной переменной (эта величина может быть меньше максимальной длины, заданной при описании переменной). Так значение функции `length`, аргументом которой является переменная `stroka` после выполнения указанного ранее оператора присваивания, будет равно 14, а не 25, как указано в разделе описаний, т. к. фраза "набор символов" содержит 14 символов.

Если есть необходимость работать не со всей переменной целиком, а с каким-либо отдельным содержащимся в ней символом, то к этому символу можно обратиться в программе непосредственно, указав имя содержащей его строковой переменной и его порядковый номер в этой переменной, заключенный в квадратные скобки. Например, если мы напишем в основной части программы (а не в разделе описаний) `stroka[5]`, то работать мы будем с буквой "р", которая является пятым по счету символом в данной переменной.

При обращении к элементу строковой переменной в скобках после ее имени может указываться не только числовая константа, но и имя переменной целого типа. В этом случае номер "вызываемого" символа будет равен значению данной целочисленной переменной. Например, если мы присвоим целочисленной переменной `i` значение 10, а затем укажем в программе `stroka[i]`, то в результате будет "вызвана" буква "в", которая является в строковой переменной десятой по счету.

В качестве еще одного примера использования строковых переменных рассмотрим программу, которая для введенной с клавиатуры строки символов выводит на экран компьютера последовательность соответствующих им числовых кодов таблицы ASCII. В листинге 6.4 приведен текст данной программы, которую мы далее будем разбирать.

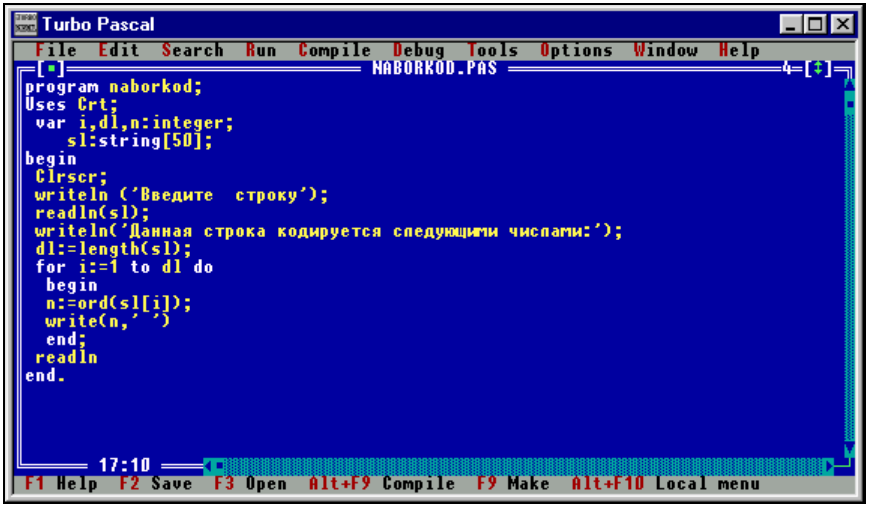
Листинг 6.4. Программа, определяющая, какими числами кодируется введенная пользователем строка

```
program naborkod;
Uses Crt;
 var i,dl,n:integer;
    sl:string[50];
begin
 Clrscr;
 writeln ('Введите  строку');
 readln(sl);
 writeln('Данная строка кодируется следующими числами:');
 dl:=length(sl);
 for i:=1 to dl do
  begin
  n:=ord(sl[i]);
  write(n,' ')
  end;
 readln
end.
```

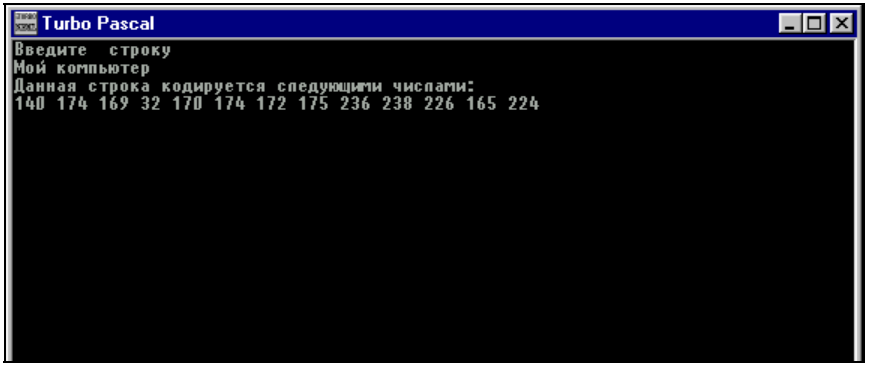
В начале данной программы пользователю предлагается ввести некоторую строку символов, длина которой не должна превышать 50. Эта строка симво-

лов, введенная с помощью оператора `readln`, присваивается строковой переменной `sl`. Далее с помощью известной нам функции `length` определяется фактическая длина введенной строки. Полученное числовое значение присваивается целочисленной переменной `dl`.

Вывод числовых кодов, содержащихся в строке символов, мы производим с помощью цикла с заданным числом повторений. Начальным значением переменной цикла `i` является единица, а конечным значением переменной цикла является значение переменной `dl`. Таким образом, в ходе работы цикла будут выведены значения десятичных кодов для всех содержащихся в строке символов, начиная с первого и заканчивая последним, порядковый номер которого равен значению переменной `dl`.



а



б

Рис. 6.5. Программа определения числовых кодов для вводимой строки символов *(а)* и результат ее работы *(б)*

В теле цикла, производящего вывод десятичных значений, содержатся следующие операторы. Это, во-первых, оператор присваивания. В этом операторе целочисленной переменной `n` присваивается значение, которое равно числовому коду текущего символа, т. е. символа, который имеет в строке порядковый номер, равный текущему значению переменной цикла `i`. Числовой код определяется посредством известной нам стандартной функции `ord`, аргументом которой является текущий символ. Второй оператор тела цикла осуществляет вывод полученного числового кода на экран компьютера. Коды выводятся в одну строку. Для того чтобы выводимые коды не сливались друг с другом, в операторе вывода предусмотрено, чтобы после каждого выводимого числа ставился пробел. По завершении работы цикла завершается и работа всей программы.

Если в ответ на приглашение программы мы введем фразу "Мой компьютер", то получим следующий результат, изображенный на рис. 6.5. На экран будут выведены числовые коды всех 13 символов, входящих в данную фразу, включая пробел, который является четвертым по порядку символом и имеет код 32.

Задания для самостоятельной работы

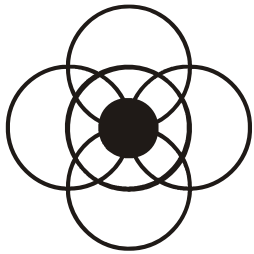
Задание 1. Составить программу, которая проверяет, является ли введенная с клавиатуры последовательность символов целым числом, записанным в двоичной системе счисления, если известно, что в данной системе счисления для записи числа используются только две цифры — нуль и единица.

Задание 2. Составить программу, которая подсчитывает число слов во введенной в компьютер с клавиатуры строке. Условимся считать словом любую последовательность символов, которая отделена от других таких же последовательностей пробелом. Например, в предложении "персональный компьютер фирмы Apple" имеются 4 слова, отделенные друг от друга 3 пробелами. Из приведенного примера видно, что количество слов в строке можно определить как сумму имеющихся в строке пробелов плюс единица.

Задание 3. Изменить предыдущую программу таким образом, чтобы она подсчитывала среднюю длину слова во введенной с клавиатуры строке.

Задание 4. Составить программу, которая зашифровывает введенный с клавиатуры в компьютер текст. Процесс шифровки производится следующим образом: из десятичного кода каждого введенного с клавиатуры символа вычитается какое-либо число (например, 10). Получившаяся в результате вычитания величина интерпретируется как десятичный код некоторого другого символа, который и выводится на экран компьютера. Таким образом, получается внешне бессмысленный набор символов, который можно расшифровать, только зная упомянутый ранее принцип шифрования исходного сообщения.

ГЛАВА 7



Массивы в Паскале

Массив представляет собой упорядоченную последовательность однородных элементов. Элементами массива могут быть различные величины, как числовые (целые и вещественные), так и символьные или строковые. Но при этом строго должно соблюдаться следующее правило: все элементы каждого отдельно взятого массива должны относиться к одному и тому же типу, что и называется однородностью массива. В массиве каждый элемент имеет свой порядковый номер, который называется *индексом*. Массивы бывают одномерные и двумерные. Начнем изучение массивов с более простого одномерного массива.

7.1. Одномерные массивы

Примером простейшего одномерного массива является список учеников одного школьного класса или студентов одной группы. В этом случае элементами массива будут фамилии учеников или студентов, а индексами — номера учеников или студентов в списке. Важной характеристикой массива является его *диапазон*, т. е. пределы, в которых может изменяться значение индекса массива.

В случае использования массива в программе, он предварительно должен быть описан в разделе описания переменных. Но описывается массив иначе, чем обычная переменная. В общем виде описание массива, состоящего из элементов-переменных, выглядит следующим образом:

```
var имя_массива: array[a..b] of тип_элементов;
```

где `var`, `array` и `of` — служебные слова. `Array` означает "массив", предлог `of` в данном случае "из", `a` и `b` — соответственно нижняя и верхняя границы диапазона массива.

Приведем примеры описания массивов.

```
var a: array[1..20] of integer;
```

Это описание массива с именем `a`, который может содержать до 20 элементов, причем все эти элементы целого типа.

```
var st: array[1..10] of string[15];
```

В данном случае описан массив с именем `st` из 10 строковых элементов, каждый из которых может содержать до 15 символов.

В программе можно работать не только со всем массивом целиком, но и с отдельными его элементами. Для того чтобы обратиться в программе к какому-либо элементу массива, нужно указать имя массива и индекс содержащегося в нем элемента. Например, если вы встретите в программе следующую запись: `a[10]`, то она означает, что мы обращаемся к элементу массива `a` с порядковым номером `10`.

Рассмотрим программу определения максимальной и минимальной температуры за месяц. В листинге 7.1 приведем текст данной программы и разберем ее.

Листинг 7.1. Программа определения максимальной и минимальной температуры месяца

```
program month;
 Uses Crt;
 var m:array[1..31] of real;
     i,v:integer;
     min,max,x:real;
begin
 ClrScr;
 writeln('Введите количество дней в месяце'); readln(v);
 writeln('Введите температуру за 1 день'); readln(m[1]);
 min:=m[1]; max:=m[1];
 for i:=2 to v do
   begin
   writeln('Введите температуру за ',i,' день');
   readln(x); m[i]:=x;
   if m[i]<min then min:=m[i];
   if m[i]>max then max:=m[i]
   end;
 writeln('Максимальная температура месяца равна ',max:7:2);
 writeln('Минимальная температура месяца равна ',min:7:2);
 readln
end.
```

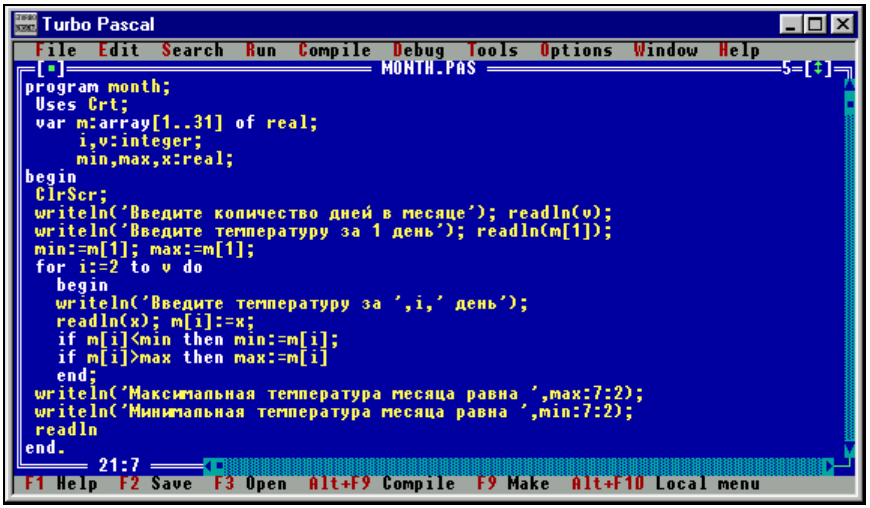
Данные о температуре за каждый день в программе вводятся с клавиатуры и хранятся в массиве. Нижней границей диапазона массива будет 1, а верхней 31 (максимальное число дней, которое может быть в месяце). Число дней, которое фактически содержится в том месяце, данные по которому обрабатываются программой, вводится с клавиатуры и содержится в переменной `v`. Заполнение массива будет производиться в цикле с заданным числом повторений. Счетчик цикла `i` будет меняться от 2 до `v`, т. к. первый элемент массива заполняется не в цикле. (Почему так делается, будет объяснено далее.) В ходе работы цикла очередному элементу массива присваивается соответствующее значение, равное температуре за `i`-й день месяца.

Если количество дней в месяце меньше, чем 31, то массив будет заполнен не до конца. Это вполне допустимо и никак не повлияет на работоспособность программы.

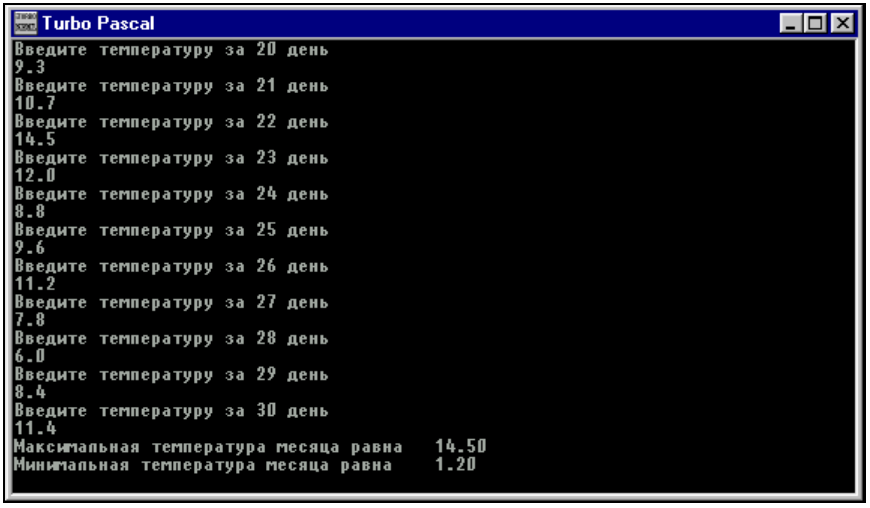
Для того чтобы определить максимальный и минимальный элементы массива (и соответственно температуры месяца), используем следующий прием. Введем две вспомогательные переменные `min` и `max`. До начала работы цикла присвоим каждой из этих переменных значение, равное первому элементу массива (температура за первый день месяца). Затем каждый вновь вводимый (в цикле) элемент массива будем сравнивать с этими двумя переменными. Если значение элемента меньше, чем `min`, то вспомогательной переменной присваивается новое (минимальное на текущем этапе) значение, равное этому элементу. Если же значение элемента больше или равно `min`, то переменная остается без изменения. Для этого используется сокращенный условный оператор. Аналогично производится работа с вспомогательной переменной `max`. В итоге после завершения работы цикла переменные `min` и `max` действительно будут содержать соответственно минимальное и максимальное значения температур, которые останется только вывести на печать оператором `writeln`.

Отметим еще две особенности массивов. Во-первых, элементами массива могут быть не только переменные, но и константы. Подобный массив, естественно, описывается в разделе объявлений как константа, причем константы описываются перед переменными. В этом же разделе такому массиву присваиваются значения. Во-вторых, нижней границей диапазона массива необязательно должна быть единица. Важно только, чтобы значение нижней границы было меньше, чем верхней.

Работу с массивом, состоящим из констант, хорошо иллюстрирует программа, которая по введенному пользователем номеру года по европейскому летоисчислению определяет его название по традиционному восточному календарю. В восточном календаре, как известно, каждый год называется по имени какого-либо животного, причем эти годы образуют 12-летний цикл, по истечении которого названия годов цикла снова повторяются. Первым годом



а



б

Рис. 7.1. Программа определения максимальной и минимальной температур за месяц *(а)* и результат ее работы *(б)*

восточного цикла считается год мыши. Нам предстоит составить программу, которая решает задачу для второй половины XX века и первой половины XXI века по европейскому летоисчислению. Известно, что начало очередного года мыши приходилось на 1948 год. Для решения данной задачи создадим в программе массив из строковых элементов. Элементами массива являются строковые константы, каждая из которых содержит название одного из годов 12-летнего восточного цикла. Нижней границей диапазона индекса данного массива является единица. Верхней границей 12 номер последнего года цик-

ла. Пользователь может ввести год, начиная с 1948 и заканчивая 2055 (последний год очередного цикла), и в ответ будет выведен строковый элемент массива с соответствующим индексом, т. е. название года.

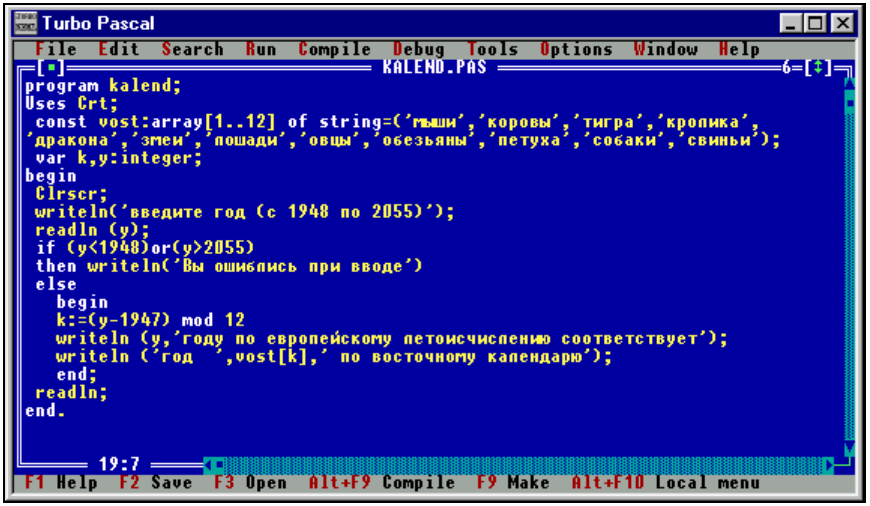
Так как пользователь может ошибиться при вводе исходных данных, то в программе предусмотрена их проверка, которая производится с помощью условного оператора `if`. При этом проверяются сразу два условия. Вводимое число не должно быть меньше минимального или больше максимального. Для того чтобы оба условия проверялись параллельно, они объединены служебным словом `or`, что означает "или". Если нарушена или верхняя, или нижняя граница диапазона, пользователю сообщается о допущенной им при вводе ошибке.

Затем в программе определяется номер года в 12-летнем цикле, обозначенный переменной `k`. Для определения величины `k` воспользуемся следующим приемом. Из введенного пользователем номера года `y` вычтем число 1947 (число лет, прошедшее с начала нашей эры по европейскому летоисчислению до начала очередного цикла восточного календаря). Полученная разность представляет собой количество лет, которое прошло с начала очередного цикла. Затем найдем остаток от деления полученной разности на 12. Это и будет номер года в цикле. Далее останется вывести на экран компьютера элемент массива с номером, равным значению переменной `k`. Таким образом, мы получим искомое название года по восточному календарю. В листинге 7.2 приводим текст данной программы.

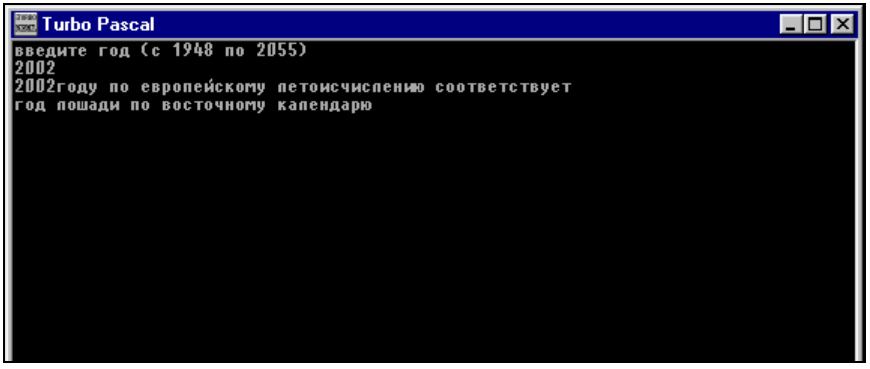
Листинг 7.2. Определение названия года по восточному календарю

```
program kalend;
Uses Crt;
 const vost:array[1..12] of string =('мыши','коровы','тигра','кролика',
'дракона','змеи','лошади','овцы','обезьяны','петуха','собаки','свиньи');
 var k,y:integer;
begin
 Clrscr;
 writeln('введите год (с 1948 по 2055)');
 readln (y);
 if (y<1948)or(y>2055)
 then writeln('Вы ошиблись при вводе')
 else
   begin
   k:=(y-1947) mod 12
   writeln (y,'году по европейскому летоисчислению соответствует');
   writeln ('год  ',vost[k],' по восточному календарю');
   end;
```

```
 readln;
end.
```



а



б

Рис. 7.2. Программа, определяющая название года по восточному календарю *(а)*, и результат ее работы *(б)*

7.2. Двумерные массивы

Простейшим примером двумерного массива является всем известная таблица умножения, в которой результат умножения двух чисел определяется по номеру строки, соответствующей одному из сомножителей, и номеру столбца, соответствующему другому. Соответственно и в любом двумерном массиве элемент определяется по двум индексам.

В языке Паскаль в общем виде двумерные массивы описываются следующим образом:

```
var имя_массива: array[a..b,c..d] of тип_элементов;
```

где `a` и `b` — соответственно верхняя и нижняя граница диапазона значений для первого индекса; `c` и `d` — верхняя и нижняя граница диапазона значений для второго индекса.

Двумерный массив таким образом можно представить в виде таблицы, имеющей `b-a+1` строк и `c-d+1` столбцов.

Пример описания двумерного массива:

```
var tabl: array [1..9,1..9] of integer;
```

таким образом описан целочисленный массив, содержащий 9 строк и 9 столбцов.

Рассмотрим основные приемы работы с двумерными массивами на примере программы `arrsum`, составленной для решения следующей задачи: дан двумерный массив размерностью 4×5. Элементами данного массива являются вещественные числа. Требуется: заполнить массив произвольными числами, затем вывести содержимое массива на экран компьютера, подсчитать сумму элементов для каждого столбца, имеющегося в массиве, и сформировать из этих сумм одномерный массив. В листинге 7.3 приводится текст программы.

Листинг 7.3. Программа, заполняющая двумерный массив произвольными числами и подсчитывающая сумму элементов для каждого столбца

```
program arrsum;
Uses Crt;
 var
   f:array[1..4,1..5] of real; x:array[1..5] of real;  i,j:integer; s:real;
begin
 Clrscr;
 for i:=1 to 4 do
   for j:=1 to 5 do
     begin
     writeln('Введите ',j,' элемент ',i,' строки');
     readln(f[i,j])
     end;
 writeln('элементы двумерного массива');
 for i:=1 to 4 do
   begin
   for j:=1 to 5 do write (f[i,j]:7:2,' ');
```

```
   writeln
   end;
 writeln('элементы одномерного массива');
 for j:= 1 to 5 do
   begin
s:=0;
   for i:=1 to 4 do
     s:=s+f[i,j];
   x[j]:=s;
   write(x[j]:7:2,' ')
   end;
 readln
end.
```

В программе `arrsum` в разделе описания переменных описываются два массива. Первый из них — это двумерный массив `f`, который в дальнейшем будет заполнен числами вещественного типа. В данном массиве, как видно из описания, имеется 4 строки и 5 столбцов. Второй описанный в разделе массив `x` является одномерным и в дальнейшем будет заполнен элементами, каждый из которых представляет собой сумму элементов, содержащихся в одном из столбцов двумерного массива. Всего в одномерном массиве `x` должно быть 5 элементов — по количеству столбцов в двумерном массиве `f`.

В начале программы производится заполнение массива `f` произвольными вещественными числами, вводимыми с клавиатуры компьютера. Этот процесс осуществляется с помощью двух вложенных циклов. Во внутреннем цикле производится поэлементное заполнение одной из строк массива числами. Этот цикл повторяется 5 раз в соответствии с количеством элементов в каждой строке (количество элементов двумерного массива, содержащихся в одной строке, равно количеству столбцов, имеющихся в данном массиве). Соответственно и счетчик данного цикла — переменная `j` — изменяет свое значение от 1 до 5. Перед непосредственным вводом очередного элемента, производимым оператором `readln`, оператор `writeln` выводит приглашение на ввод данного элемента, указывающее номер текущей строки и порядковый номер элемента в строке. Внешний цикл с переменной `i` обеспечивает повторение всех действий внутреннего цикла 4 раза (по количеству строк, имеющихся в массиве).

Чтобы наглядно убедиться в том, что двумерный массив `f` целиком заполнен данными, производим построчный вывод содержимого массива на экран компьютера. Эта операция, как и предыдущая, осуществляется с помощью двух циклов, один из которых вложен в другой. Внутренний цикл с переменной `j` обеспечивает вывод элементов одной строки. При этом, для того чтобы значения элементов массива представлялись на экране в удобном для воспри-

ятия человеком виде (а не в нормализованном), элементы выводятся в отформатированном виде. Под каждый элемент отводится по 7 позиций, в том числе 4 под целую часть, 1 — под точку и 2 — под дробную часть. Внешний цикл с переменной `i`, как и в предыдущем случае, обеспечивает четырехкратное повторение действий внутреннего цикла.

Третья группа из двух вложенных циклов используется для подсчета сумм элементов нового одномерного массива и его формирования. Внешний цикл в данном случае повторяет всю совокупность действий, производимых во внутреннем цикле, 5 раз — по числу элементов формируемого цикла (и, соответственно, столбцов двумерного массива). Внутренний же цикл с переменной `i` подсчитывает сумму элементов, содержащихся в одном из столбцов. Эта сумма подсчитывается посредством вспомогательной переменной `s`. Перед началом работы внутреннего цикла эта переменная обнуляется с помощью оператора присваивания — первого оператора внешнего цикла, а затем во внутреннем цикле к ее текущему значению каждый раз прибавляется значение очередного элемента из обрабатываемого столбца. В итоге работы внутреннего цикла в переменной `s` мы и получаем искомую сумму. Далее следующий оператор внешнего цикла присваивает получившееся значение соответствующему элементу одномерного массива. Наконец, последний из операторов внешнего цикла выводит значение данного элемента одномерного массива на экран. Таким образом, после завершения работы этого внешнего цикла массив будет заполнен данными, и поставленная задача будет решена.

На рис. 7.3 приведены результаты работы описанной программы.

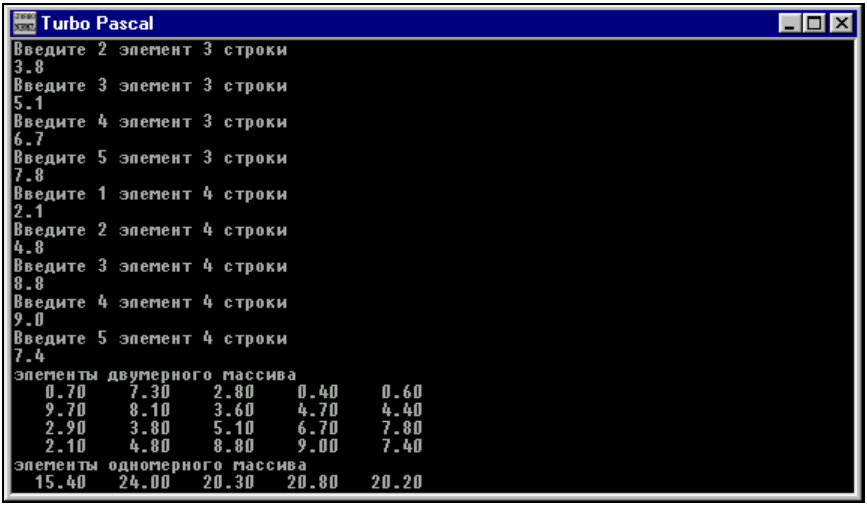


Рис. 7.3. Результат работы программы, заполняющей двумерный массив произвольными элементами и формирующей на его основе одномерный массив

7.3. Сортировка массивов

Одной из наиболее распространенных задач, с которыми приходится сталкиваться при обработке массивов, является *сортировка* массивов, т. е. расположение элементов массивов определенным упорядоченным образом. Можно встретить несколько способов сортировки массивов.

- Сортировка *по возрастанию*. В этом случае элементы массива располагаются таким образом, что в его начале располагается наименьший элемент, затем следующий по величине элемент и т. д. вплоть до наибольшего, т. е. каждый следующий элемент должен быть больше предыдущего.
- Сортировка *по убыванию*. В данном случае после проведения сортировки в начале массива оказывается наибольший по величине элемент, затем идет следующий по величине и т. д. вплоть до наименьшего, т. е. каждый следующий элемент должен быть меньше предыдущего.
- Сортировка *по неубыванию*. В массиве, отсортированном таким образом, каждый последующий элемент должен быть больше или равен предыдущему.
- Сортировка *по невозрастанию*. В отсортированном по невозрастанию массиве каждый последующий элемент должен быть меньше или равен предыдущему.

Существует несколько различных способов сортировки массивов, мы разберем два самых простых. Начнем со способа, который называется сортировка *методом простого выбора*. Если нужно упорядочить массив по возрастанию, то суть этого способа сводится к следующему: в массиве нужно найти максимальный по величине элемент и поставить его на последнее место в массиве. В результате максимальный элемент массива займет подобающее ему место. Тот элемент, который ранее находился в массиве на последнем месте, необходимо переместить на то место, которое ранее занимал в массиве максимальный элемент.

Далее работаем с группой элементов, которая начинается с первого элемента массива, а заканчивается предпоследним элементом. В этой группе также следует найти максимальный элемент и затем поставить его на последнее место в данной группе элементов, т. е. на предпоследнее место во всем массиве. Элемент, стоявший предпоследним, также помещается в массиве на освободившееся место. Затем рассматривается группа элементов без последнего и предпоследнего. В этой группе также находится максимальный элемент, который помещается на положенное ему последнее в группе место. Такие операции будут продолжаться до тех пор, пока каждый элемент массива не займет подобающее ему место. Последняя операция сведется к выбору максимального в группе из двух оставшихся элементов.

Описанный метод реализован в программе, текст которой приведен в листинге 7.4.

Листинг 7.4. Сортировка массива методом простого выбора

```
program vybsort;
Uses Crt;
 var
    m:array[1..10] of integer; i,k,max,n:integer;
 begin
 ClrScr;
 for i:=1 to 10 do
   begin
   writeln('Введите ',i,' элемент массива');
   readln(m[i])
   end;
 for n:=10 downto 2 do
   begin
   max:=m[1];
   k:=1;
   for i:=2 to n do
     begin
     if m[i]>max then
       begin
       max:=m[i];
       k:=i
       end;
     end;
   m[k]:=m[n];
   m[n]:=max;
   end;
 writeln;
 for i:=1 to 10 do write(m[i],' ');
 readln
end.
```

Основная часть данной программы начинается с заполнения одномерного массива `m`, состоящего из десяти элементов целого типа. Заполнение массива производится в цикле с заранее заданным числом повторений. В теле данного цикла содержится два оператора. Оператор вывода, содержащий приглашение на ввод очередного элемента массива и указывающий его номер, и оператор ввода, заполняющий данный массив.

После заполнения массива начинается собственно процесс его сортировки. Данный процесс является однотипным, но при этом он производится в соот-

ветствии с приведенным ранее алгоритмом сначала для группы из десяти элементов, затем для девяти (без последнего элемента), потом для группы из восьми элементов и далее по убывающей. Поэтому процесс сортировки будет осуществляться в цикле с заранее заданным числом повторений и с убывающим значением переменной цикла `n`. Начальное значение данной переменной будет равно 10, а конечное — 2, т. к. на последнем этапе сортировки мы будем иметь дело с группой, состоящей только из двух элементов.

В самом теле цикла мы будем производить следующие действия. В начале вспомогательной переменной `max`, которая должна содержать значение максимального элемента в рассматриваемой группе, мы присваиваем значение первого по счету элемента массива. Одновременно мы присваиваем начальное значение и другой вспомогательной переменной `k`. Эта переменная должна содержать индекс максимального по значению элемента в рассматриваемой группе. Так как в начале поиска максимума мы условно считаем максимальным первый элемент, то переменной `k` мы присваиваем значение 1.

Далее для нахождения действительно максимального элемента в группе мы используем внутренний цикл, вложенный во внешний цикл с переменной `n`. Переменной внутреннего цикла будет переменная `i`, показывающая текущее значение индекса элемента. В процессе поиска максимального элемента значение этой переменной будет изменяться от 2 до `n` — индекса последнего по счету элемента в группе. Для каждого элемента группы, начиная со второго, мы сравниваем его значение с максимальным. Если значение данного элемента окажется больше значения переменной `max`, то переменной `max` присваивается новое значение, которое равно значению данного элемента массива `m[i]`, а переменной `k` также присваивается новое значение, которое равно `i`-индексу данного элемента. В итоге, по окончании работы внутреннего цикла после перебора всех элементов группы переменная `max` будет содержать значение действительно максимального элемента группы, а переменная `k` — индекс этого элемента.

Теперь осталось в соответствии с алгоритмом поставить максимальный элемент на последнее место в группе, а последний элемент на то место, которое ранее занимал максимальный. Для этого мы элементу с номером `k` присваиваем значение последнего элемента массива `m[n]`, а последнему элементу — найденное максимальное значение. Таким образом, в результате работы двух вложенных циклов мы получим исходный массив уже в упорядоченном виде. Последний имеющийся в программе цикл с переменной `i` выводит на экран компьютера в одну строку элементы упорядоченного массива `m`. Результаты работы данной программы приведены на рис. 7.4.

Еще одним распространенным способом сортировки массива является сортировка *методом простого обмена*. Суть этого метода (здесь мы также рас-

сматриваем вариант с сортировкой элементов по возрастанию) сводится к следующему: на начальном этапе берется группа, включающая в себя все элементы массива. В этой группе производится попарное сравнение всех соседних элементов массива, т. е. сравниваются первый и второй элементы массива, второй и третий, третий и четвертый и т. д. В том случае, если оказывается, что в паре первый элемент по величине больше, чем второй, то данные элементы меняются местами для того, чтобы максимальный из двух элементов находился в конце пары. Таким образом, в результате попарных сравнений и перемещений наибольший элемент массива постепенно смещается в его конец.

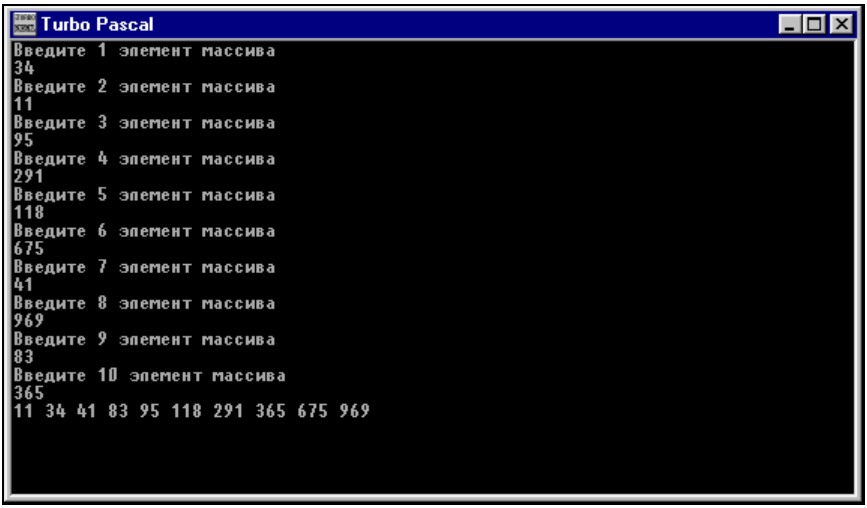


Рис. 7.4. Результат работы программы сортировки массива методом простого выбора

На следующем этапе обработки массива с помощью данного метода берется группа, включающая все элементы массива за исключением последнего. При обработке этой группы описанным ранее методом максимальный из оставшихся элементов также перемещается в конец группы. Далее берется следующая группа элементов (без последнего и предпоследнего) и в ней смещается в конец последний элемент, и так продолжается до тех пор, пока не останется последняя группа из двух элементов, которая также упорядочивается указанным образом.

Листинг 7.5. Сортировка массива методом обмена

```
program obmsort;
Uses Crt;
 var m:array[1..10] of integer;
     i,n,t:integer;
```

```
begin
 ClrScr;
 for i:=1 to 10 do
   begin
   writeln('Введите ',i,' элемент массива');
   readln(m[i]);
   end;
 for n:=10 downto 2 do
   if m[i]<m[i-1] then
   begin
   t:=m[i];
   m[i]:=m[i-1];
   m[i-1]:=t
   end;
 writeln;
 writeln('Массив в отсортированном виде:');
 for i:=1 to 10 do
   write(m[i],' ');
 readln
 end.
```

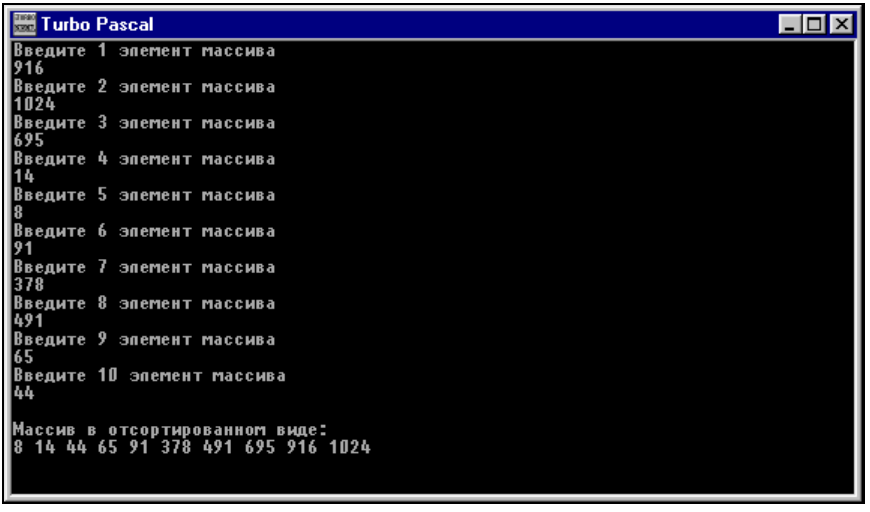


Рис. 7.5. Результат работы программы сортировки массива методом простого обмена

Разберем текст программы сортировки методом обмена, приведенный в листинге 7.5. Как и предыдущая, данная программа начинается с заполнения одномерного массива, содержащего 10 элементов целого типа. Заполнение производится в цикле с переменной `i`. Далее вновь используем структуру, кото-

рая представляет собой два вложенных цикла. Переменной внешнего цикла является `n` — количество элементов в группе, которое последовательно уменьшается с 10 (в начале работы внешнего цикла) до 2 (в конце его работы). Подобное уменьшение имеет место за счет того, что на каждом последующем этапе сортировки сокращается группа элементов, в которой производится сравнение элементов.

Во внутреннем цикле производится попарное сравнение всех элементов очередной группы. Переменной внутреннего цикла является `i` — индекс текущего элемента группы. Начальное значение переменной внутреннего цикла принимаем равным 2. Второй по счету элемент группы мы сравниваем с предыдущим, т. е. с первым. Аналогичным образом каждый последующий `i`-й элемент мы сравниваем с предыдущим элементом с индексом, равным `i-1`. В случае если оказывается, что текущий элемент меньше предыдущего, производим обмен значений между этими двумя соседними элементами. Операторы, осуществляющие эту замену, расположены в сокращенном условном операторе после слова `then`. Обмен значениями производится с помощью вспомогательной переменной `t`. Для этого `t` первоначально присваивается значение второго элемента из пары. Затем второму элементу присваивается значение первого. Наконец, первому элементу присваивается старое значение второго, которое было сохранено во вспомогательной переменной `t`. В конце программы с помощью оператора цикла выводятся значения элементов массива уже в отсортированном виде. Результаты работы программы сортировки методом обмена приведены на рис. 7.5.

Приведенные ранее алгоритмы сортировки массивов можно применять не только для сортировки по возрастанию, но и по убыванию. Отличие состоит в том, что для первого способа производится поиск и помещение на последнее место в группе не максимального, а минимального элемента. Для второго же способа отличие заключается в том, что при сравнении двух соседних элементов обмен их значениями производится в том случае, если в сравниваемой паре первый элемент меньше второго.

Задания для самостоятельной работы

Задание 1. Составить программу, заполняющую массив, состоящий из `n` элементов (где `n` не больше 20), введенными с клавиатуры целыми числами. В этом массиве требуется определить число содержащихся в нем положительных элементов.

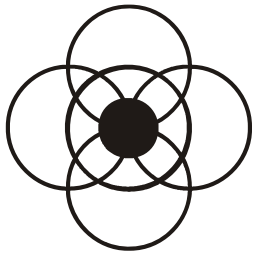
Задание 2. Составить программу, заполняющую массив из `n` элементов случайными целыми числами, находящимися в интервале от 1 до 50. Вывести на экран компьютера созданный массив и найти количество тех элементов, зна-

чения которых находятся в диапазоне от a до b. Число элементов массива и значения a и b вводятся с клавиатуры.

Задание 3. Составить программу, заполняющую массив, состоящий из n элементов (где n не больше 20), введенными с клавиатуры целыми числами. Требуется вывести массив на экран компьютера и найти индекс последнего по счету в массиве отрицательного элемента.

Задание 4. Составить программу, которая определяет, какие символы встречаются во вводимой с клавиатуры текстовой строке, а также определяет, сколько раз встречается во введенном тексте тот или иной символ. Признаком окончания ввода текстовой строки является ввод нуля, который сам в числе символов не учитывается. При этом в статистику символов должны быть включены наряду с буквами, цифрами и знаками препинания также и пробелы.

ГЛАВА 8



Подпрограммы в Паскале

При разработке ряда сложных программ достаточно часто приходится сталкиваться с ситуацией, когда в различных частях программы выполняется одна и та же последовательность действий. В этом случае гораздо рациональней бывает оформить эту последовательность в виде отдельной части программы, которая называется *подпрограммой*, а затем обращаться к ней из основной части программы по мере необходимости.

Кроме того, сам процесс программирования значительно облегчается, если разбить одну сложную задачу на ряд более простых, а затем уже объединить эти части между собой. Данный метод разработки программ также требует использования подпрограмм, которые подразделяются на подпрограммы-функции и подпрограммы-процедуры.

8.1. Подпрограмма-функция

Такие операции, как возведение числа в квадрат, извлечение квадратного корня, определение модуля (абсолютной величины) числа, вычисление тригонометрических функций, округление дробного числа до ближайшего целого, определение длины строковой переменной и многие другие, используются при решении самых разнообразных задач из области программирования.

Для того чтобы облегчить процесс составления программ в языке Паскаль, предусмотрены специальные функции, которые применяются для автоматического выполнения этих действий. Такие функции называются *стандартными* и являются неотъемлемой частью языка Паскаль.

Каждая из этих функций по вводимым в нее исходным данным определяет некоторый результат, который далее используется в программе. Исходные данные называются *аргументом* функции, а вычисленный результат — ее *значением*. При обращении к функции аргумент указывается в скобках.

К широко употребляемым стандартным функциям (кроме уже известных нам функций `length`, `ord` и `char`, используемых для работы со строковыми переменными) относятся также следующие:

- `abs(x)` — определяет абсолютное значение аргумента, которым может быть число или выражение целого или вещественного типа;
- `arctan(x)` — вычисляет арктангенс угла, значение которого выражено в радианах;
- `copy(x,n,l)` — выделяет в строковой переменной `x` группу символов, начиная с позиции с номером `n`. Длина группы равна `l`. Параметры `n` и `l` должны быть величинами целого типа;
- `cos(x)` — вычисляет косинус угла, значение которого выражено в радианах;
- `delete(x,n,l)` — удаляет из строковой переменной `x` группу символов, начиная с позиции с номером `n`. Количество символов равно `l`;
- `exp(x)` — вычисляет экспоненту аргумента, т. е. `e` в степени `x`;
- `int(x)` — определяет целую часть аргумента. Значением функции является величина вещественного типа;
- `ln(x)` — вычисляет натуральный логарифм аргумента (т. е. логарифм по основанию `e`);
- `pi` — данная функция не имеет параметров, а значением ее является число Пи;
- `round(x)` — округляет значение аргумента до ближайшего целого числа;
- `sin(x)` — вычисляет синус угла, значение которого выражено в радианах;
- `sqr(x)` — вычисляет квадрат аргумента, которым может быть число или выражение целого или вещественного типа;
- `sqrt(x)` — вычисляет квадратный корень из аргумента;
- `trunc(x)` — определяет целую часть аргумента. Значением функции является величина целого типа.

В качестве примера программы с использованием стандартных функций рассмотрим программу решения квадратичного уравнения $ax^2 + bx + c = 0$.

Листинг 8.1. Программа решения квадратного уравнения с использованием стандартных функций

```
program qwur;
Uses Crt;
 var a,b,c,d,x1,x2:real;
```

```
begin
 Clrscr;
 writeln('Введите коэффициенты уравнения - a,b,c');
 readln(a);
 readln(b);
 readln(c);
 d:=sqr(b)-4*a*c;
 if d<0 then writeln('корней нет')
        else
          begin
          x1:=(-b+sqrt(d))/(2*a);
          x2:=(-b-sqrt(d))/(2*a);
          writeln('Корни квадратного уравнения');
          writeln(x1:6:2,' ', x2:6:2)
          end;
 readln
end.
```

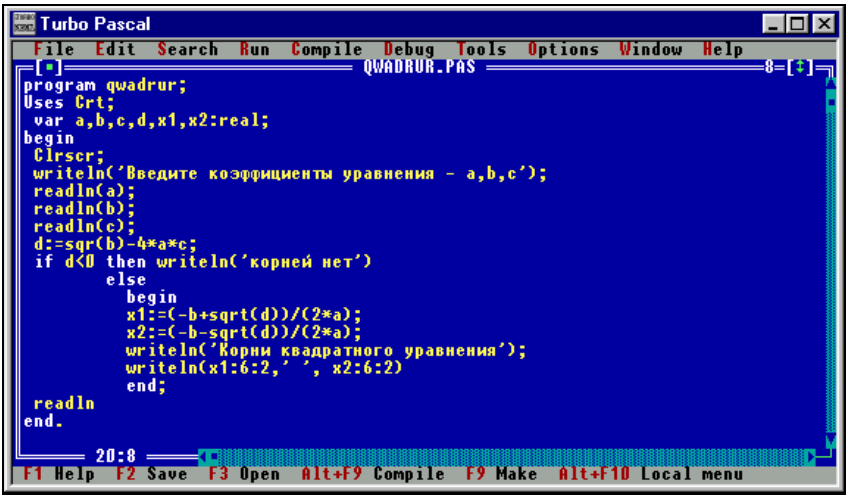
В данной задаче в начале с помощью стандартной функции `sqr` вычисляется дискриминант уравнения. В зависимости от значения дискриминанта выясняется, имеет ли данное уравнение решение. Если решения нет, то соответствующее сообщение выводится на экран компьютера. В случае же, если решение имеется, оно определяется с помощью стандартной функции `sqrt`.

При значениях коэффициентов уравнения 2, 7 и 3 получаем решение, изображенное на рис. 8.1.

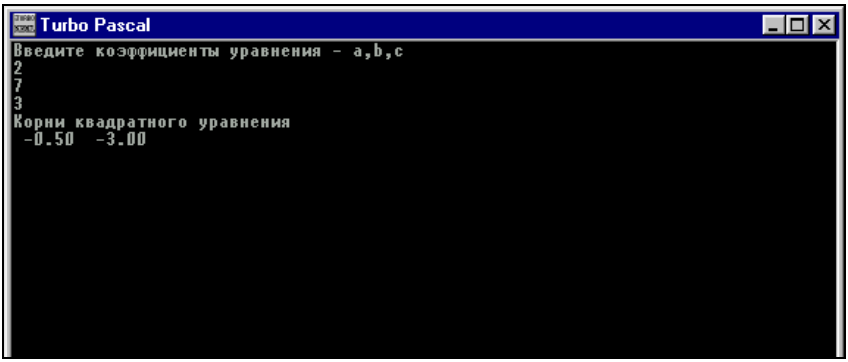
Набор стандартных функций языка Паскаль достаточно обширен (он не ограничивается приведенным ранее списком), однако в ряде случаев программисту может потребоваться для решения поставленной задачи создать свою собственную функцию. Такая функция должна быть описана в тексте программы после раздела описания констант и переменных и до начала ее основной части (т. е. до слова `begin`). Подобную функцию часто называют *функцией-подпрограммой*, т. к. она представляет собой отдельный блок внутри основной программы, который по своей структуре напоминает основную программу. Структура описания создаваемой программистом функции выглядит следующим образом:

```
заголовок функции;
раздел описания констант и переменных, используемых внутри функции;
begin
операторы функции
end;
```

Теперь разберем более подробно элементы этой структуры.



а



б

Рис. 8.1. Программа решения квадратичного уравнения *(а)* и результат ее работы *(б)*

Общий вид заголовка функции следующий:

`function` *`имя_функции (параметры функции): тип функции`*`;`

где `function` — служебное слово, означающее "функция". Имя функции дается по тем же правилам, что и имя переменной. В скобках указываются аргументы функции, называемые ее *параметрами*, причем для каждого параметра обязательно должен быть указан его тип. Если параметры относятся к одному типу, то они перечисляются через запятую, а после двоеточия указывается их общий тип, если же параметры относятся к разным типам, то они отделяются друг от друга точкой с запятой. После скобок обязательно указывается *тип значения* самой функции.

Пример заголовка функции:

```
function beta (x,y:integer; z:real):real;
```

данная функция именуется `beta`, в ней используются 3 параметра: `x` и `y` — целого типа, `z` — вещественного, а значение самой функции является вещественным.

Таким образом, заголовок функции в целом напоминает описание переменной, но следом за заголовком в описании функции указываются оператор или группа операторов, по которым вычисляется ее значение. Эти действия записываются в виде составного оператора, который начинается со служебного слова `begin` и заканчивается словом `end`. В составном операторе могут использоваться свои, локальные переменные. Раздел описания этих переменных (для каждой из локальных переменных также должен быть указан ее тип) помещается между заголовком и составным оператором. Не забывайте о том, что в составном операторе обязательно должен быть оператор присваивания, который присваивает получившийся результат функции.

В основной части программы действия, указанные в разделе описания функции, выполняются тогда, когда необходимо найти ее значение. Для этого необходимо осуществить *обращение* к функции. Обращение к функции в основной части программы включает в себя имя функции и следующий за ним список параметров, заключенный в скобки. Параметры, указанные при обращении к функции, называются *фактическими* параметрами, а параметры указанные в описании функции — *формальными*. Фактические параметры должны быть того же типа, что и формальные.

Например, правая часть оператора присваивания

```
d:=beta(3,4,7.5)
```

представляет собой обращение к функции `beta`, а 3, 4 и 7.5 — фактические параметры данной функции в отличие от формальных параметров `x`, `y` и `z`. В качестве фактических параметров функции могут выступать и константы (как в приведенном ранее примере) и переменные. Обратите внимание, что все фактические параметры перечисляются через запятую, даже если они принадлежат к разным типам.

Для того чтобы на практике закрепить все изложенное ранее, рассмотрим программу, в которой описывается и применяется создаваемая пользователем функция. Это программа определения числа сочетаний из n по m. Данное число определяется по следующей формуле:

$$C_n^m = \frac{n!}{m!(n-m)!}.$$

где $n!$, $m!$ и $(n-m)!$ — соответственно факториалы n, m и $(n-m)$. Факториалом числа n называется произведение всех натуральных чисел от 1 до n.

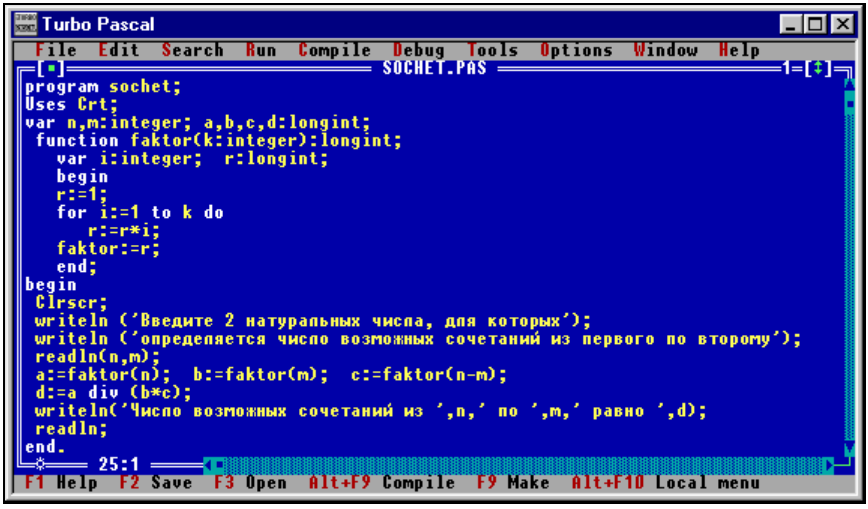
Листинг 8.2. Программа, вычисляющая число сочетаний из *n* по *m*

```
program sochet;
Uses Crt;
 var n,m:integer;
    a,b,c,d:longint;
 function faktor(k:integer):longint;
   var i:integer;  r:longint;
   begin
   r:=1;
   for i:=1 to k do
      r:=r*i;
   faktor:=r;
   end;
begin
 Clrscr;
 writeln ('Введите 2 натуральных числа, для которых');
 writeln ('определяется число возможных сочетаний');
 writeln ('из первого по второму');
 readln(n,m);
 a:=faktor(n);
 b:=faktor(m);
 c:=faktor(n-m);
 d:=a div (b*c);
 writeln('Число возможных сочетаний из ',n,' по ',m,' равно ',d);
 readln;
end.
```

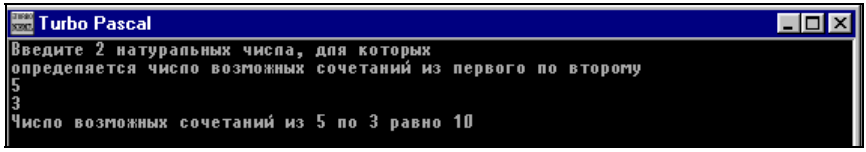
Разберем текст данной программы, приведенный в листинге 8.2. Так как в программе нам предстоит три раза вычислять факториал различных чисел, то в целях ее рационализации вычисление факториала оформим в виде отдельной функции `faktor`.

Для того чтобы было понятно, как работает данная функция, рассмотрим в начале алгоритм вычисления факториала. Проще всего начать вычисления с нахождения факториала единицы. Согласно определению факториала он будет равен произведению единицы на единицу, т. е. единице. Далее определяем факториал двух. Для этого факториал единицы умножаем на два и получаем два. Факториал трех находим путем умножения факториала двух, т. е. двойки, на число три и получаем шесть. Таким образом, алгоритм нахождения факториала некоторого числа сводится к тому, что мы последовательно умножаем каждое следующее число на факториал предыдущего. Понятно, что для многократного повторения аналогичных действий удобнее всего использовать циклическую структуру.

Следовательно, в функции вычисления факториала должны использоваться следующие переменные. Это переменная `k`, которой присваивается значение того натурального числа, факториал которого вычисляется. Далее будет использоваться переменная `r`, которой будет присвоено начальное значение, равное единице, а затем будут поочередно присваиваться значения факториалов всех чисел от единицы до `k` включительно. Наконец, мы используем переменную `i` — переменную цикла, в котором будет происходить это последовательное присваивание. Переменная `k` является параметром функции, а переменные `r` и `i` являются вспомогательными.



а



б

Рис. 8.2. Программа подсчета числа сочетаний из `n` по `m` *(а)* и результат ее работы *(б)*

Тело функции вычисления факториала состоит из следующих операторов. Это оператор присваивания, в котором вспомогательной переменной `r` присваивается начальное значение, равное 1. Далее в теле функции содержится оператор цикла с заданным числом повторений. При каждом очередном выполнении тела цикла переменная `r` получает новое значение, равное произведению предыдущего значения этой же переменной на текущее значение переменной цикла `i` (в ходе работы цикла `i` будет изменять значение от 1 до `k`). Тело функции завершается оператором присваивания, в котором функция `faktor` получает значение, равное конечному значению `r`. Так как факториалы

даже небольших натуральных чисел представляют собой достаточно большие величины (например, факториал числа 10 равен 3 628 800), то для описания значения функции используется тип `longint`.

В основной части программы осуществляется ввод исходных данных, затем производятся вычисления по формуле с использованием значений функции `faktor`. С помощью этой функции мы находим `a` — факториал числа `n`, `b` — факториал числа `m`, и `c` — факториал разности этих чисел. Для вычисления этих величин в программе трижды производится обращение к функции `faktor` с различными фактическими параметрами. После нахождения `a`, `b` и `c` мы вычисляем `d` — искомое число сочетаний. Полученный результат оператором `writeln` выводится на экран. На рис. 8.2 приведен результат работы программы при `n`, равном 5, и `m`, равном 3.

8.2. Подпрограмма-процедура

Процедуры по своему внешнему виду напоминают функции, но если при обращении к функции мы можем получить только один результат — значение данной функции, то процедура может вырабатывать на выходе несколько значений. Кроме того, процедуры используются в программах для выполнения ряда стандартных действий. Такие процедуры, подобно стандартным функциям, являются частью языка Паскаль. К стандартным процедурам относятся уже известные нам команды `ClrScr`, `TextColor` и `TextBackground`. Процедурами являются и операторы ввода и вывода `read`, `readln`, `write` и `writeln`. Наряду с этими стандартными процедурами в языке Паскаль широко используются также следующие:

- `Delay(i)` — осуществляет задержку выполнения программы на `i` миллисекунд (тысячных долей секунды);
- `Exit` — эта процедура не имеет параметров. Она осуществляет выход из процедуры или функции в основную часть программы (не путать с командой **Exit** меню системы программирования Turbo Pascal, которая закрывает систему программирования и производит выход в среду операционной системы);
- `GotoXY(a,b)` — переводит курсор в точку экрана с координатами `a`, `b`;
- `Halt` — эта процедура не имеет параметров. Она завершает выполнение программы и передает управление операционной системе;
- `Str(x,s)` — производит преобразование числовой величины `x` в строковую `s`;
- `Val(s,x,n)` — производит преобразование строки `s`, изображающей число, в числовую величину `x`. В процедуре также должен присутствовать пара-

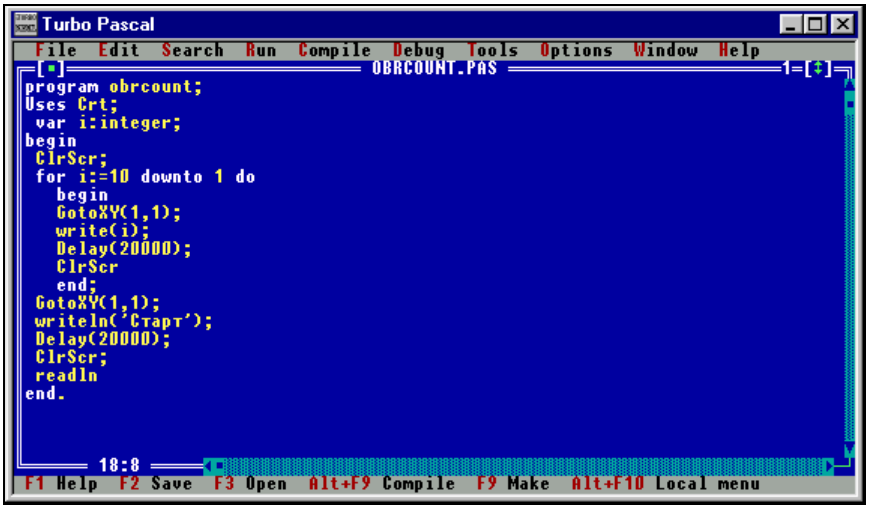
метр `n`, относящийся к целочисленному типу. Если преобразование было выполнено успешно, то значение `n` будет равно нулю. Если преобразование не может быть произведено, то в переменную `n` записывается номер символа, который явился причиной ошибки при преобразовании.

В качестве примера использования стандартных процедур Паскаля приведем программу "Обратный отсчет". Эта программа создает на экране компьютера "электронное табло", на котором в одном и том же месте последовательно выводятся числа от 10 до 1, т. е. производится обратный отсчет времени, как перед стартом космического корабля, а затем выводится слово "Старт".

Листинг 8.3. Программа "Обратный отсчет"

```
program obrcount;
Uses Crt;
 var i:integer;
begin
 ClrScr;
 for i:=10 downto 1 do
   begin
   GotoXY(1,1);
   write(i);
   Delay(20000);
   ClrScr
   end;
 GotoXY(1,1);
 writeln('Старт');
 Delay(20000);
 ClrScr;
 readln
end.
```

Разберем программу, текст которой приведен в листинге 8.3 и рис. 8.3. В данной программе цифры выводятся с помощью цикла с уменьшающимся значением счетчика. В теле цикла используется ряд стандартных процедур. Так, перед выводом очередного числа курсор посредством стандартной процедуры `GotoXY` каждый раз перемещается в левый верхний угол экрана для того, чтобы все числа выводились в одной и той же позиции. Далее, для того чтобы очередное число сразу не исчезало с экрана, а оставалось на некоторое время, используется процедура `Delay`. Затем экран очищается с помощью процедуры `ClrScr`, для того чтобы освободить место для вывода следующего числа. После завершения работы цикла с помощью тех же стандартных процедур на экран выводится слово "Старт".



а

б

Рис. 8.3. Программа "Обратный отсчет" — исходный текст *(а)* и программа в действии *(б)*

Конечно, возможности языка Паскаль не ограничиваются применением только стандартных процедур. Программист, использующий этот язык, может создавать и свои собственные процедуры. Но для того чтобы такую процедуру можно было использовать в программе, ее, подобно вновь создаваемым функциям, следует предварительно описать. Описание процедуры, так же как и описание функции, должно содержаться в программе в разделе описаний после описания констант и переменных. Структура описания процедуры сходна со структурой основной программы, поэтому создаваемую программистом процедуру часто называют *процедурой-подпрограммой*. Описание включает в себя заголовок процедуры, раздел описаний и раздел операторов. К процедуре можно обращаться из основной части программы, из другой процедуры или из функции. Такое обращение называют также *вызовом* процедуры.

Общий вид описания процедуры следующий:

```
заголовок процедуры;
раздел описаний процедуры;
begin
раздел операторов процедуры
end;
```

Теперь разберем более подробно составные части данного описания.

Общий вид заголовка процедуры следующий:

```
procedure имя_процедуры (параметры процедуры);
```

где `procedure` — служебное слово, имя процедуры дается по тем же правилам, что и имя переменной в Паскале, параметры перечисляются в скобках через запятую с указанием их типа. Эти параметры являются *формальными*. При вызове же процедуры в обращении к ней указываются ее *фактические* параметры. Тип каждого фактического параметра должен быть таким же, как тип соответствующего ему формального параметра. Количество формальных параметров, имеющихся в описании процедуры, и фактических параметров, используемых при обращении к ней, должно совпадать. При этом первому по счету формальному параметру в описании ставится в соответствие первый по счету фактический параметр в обращении, второму формальному — второй фактический и т. д.

Все формальные параметры делятся на два вида. Если перед именем параметра в заголовке процедуры стоит служебное слово `var`, то это — *параметр-переменная*. Если служебное слово `var` перед именем переменной в заголовке отсутствует, то данный параметр является *параметром-значением*. При обращении к процедуре формальному параметру-значению присваивается значение соответствующего ему фактического параметра, причем в качестве такого значения может выступать константа, переменная или выражение. Во время работы процедуры параметр-значение не может изменяться даже в том случае, если он является переменной. При обращении же к процедуре, в которой имеются формальные параметры-переменные, соответствующие им фактические параметры могут быть только переменными (не константами и не выражениями). Вызываемая процедура получает доступ к ячейкам памяти, в которых хранятся эти фактические параметры, и может изменять значения этих параметров в ходе своей работы.

Пример заголовка процедуры:

```
procedure vspomog(a,b:integer; var c,d:real);
```

где `vspomog` — имя процедуры, `a`, `b`, `c`, `d` — имена формальных параметров, причем `a` и `b` являются параметрами-значениями, а `c` и `d` — параметрами-переменными.

В разделе описаний процедуры константы и переменные описываются по тем же правилам, что и в основной программе. При этом следует иметь в виду, что эти переменные могут использоваться только внутри данной процедуры. Такие переменные называются *локальными*. В процедуре могут применяться и переменные, которые описаны в основной части программы. Такие переменные называются *глобальными*.

Раздел операторов процедуры принципиально не отличается от такого же раздела в основной программе, он может содержать обращения к другим функциям и процедурам, но после служебного слова `end` ставится не точка, а точка с запятой, т. к. конец описания процедуры — это не конец программы.

Рассмотрим программу определения максимального и минимального числа в группе из 4 чисел, текст которой приведен в листинге 8.4.

Листинг 8.4. Программа определения максимального и минимального из 4 чисел

```
program four;
Uses Crt;
 var a,b,c,d,l,m,big,lit,min1,max1,min2,max2:integer;
 procedure minmax(x1,x2:integer; var min,max:integer);
  begin
  if x1>x2
  then
    begin
    max:=x1;
    min:=x2 end
  else
    begin
    max:=x2;
    min:=x1 end
  end;
begin
 ClrScr;
 writeln ('введите 1 число');
 readln(a);
 writeln ('введите 2 число');
 readln(b);
 writeln ('введите 3 число');
 readln(c);
 writeln ('введите 4 число');
 readln(d);
 minmax(a,b,min1,max1);
```

```
 minmax(c,d,min2,max2);
 minmax(min1,min2,lit,l);
 minmax(max1,max2,m,big);
 writeln('Максимальное число - ',big,'; минимальное число - ',lit);
 readln
end.
```

Исходные значения введем с клавиатуры. Для определения максимального и минимального из них воспользуемся следующим алгоритмом. Напишем несложную процедуру `minmax`, которая определяет максимальное и минимальное из двух чисел. Затем разобьем введенные числа на пары и в каждой паре с помощью этой процедуры определим максимум и минимум. Далее с помощью той же процедуры `minmax` определим наибольший из двух максимумов и наименьший из двух минимумов. Это и будет искомый результат.

В заголовке процедуры `minmax` описаны 2 формальных параметра-значения `x1` и `x2`, которые используются для ввода в процедуру исходных данных, и 2 параметра-переменные `min` и `max`, используемые для вывода полученных результатов. Внутри процедуры параметры-значения сравниваются между собой и значение меньшего из них присваивается переменной `min`, а большего — переменной `max`.

В основной части программы обращение к процедуре `minmax` встречается 4 раза. В первых двух случаях в качестве фактических параметров выступают введенные с клавиатуры значения переменных `a` и `b` при первом вызове процедуры и `c` и `d` — при втором. Результатами являются соответственно фактические параметры `min1` и `max1` (минимум и максимум в первой паре чисел) и `min2` и `max2` (минимум и максимум во второй паре).

При третьем обращении к процедуре в качестве исходных данных используются 2 найденных минимума, а результатом ее работы является минимальное из этих 2 значений, передаваемое в переменную `lit`. Вспомогательная переменная `l`, в принципе, для решения данной задачи не нужна, но вводится, т. к. число фактических параметров в обращении должно соответствовать числу формальных параметров в описании. Аналогично при четвертом обращении к процедуре находится наибольший из двух максимумов `big`, а переменная `m` играет подобно переменной `l` только вспомогательную роль.

Искомые максимальное и минимальное значения выводятся на экран с помощью оператора `writeln`.

На рис. 8.4 приводятся результаты работы данной программы при `a` равном 65, `b` — 111, `c` — 678 и `d` — 215.

Ранее уже говорилось о том, что в процедуру могут передаваться не только значения переменных, но и числовые константы. Приведем пример програм-

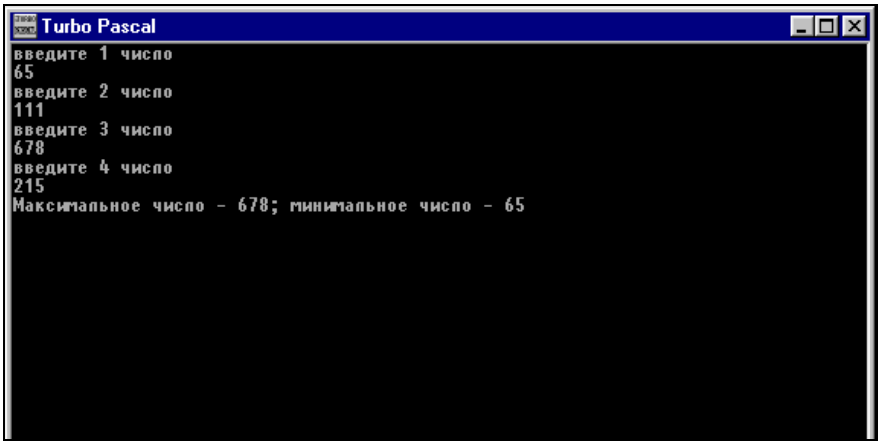


Рис. 8.4. Результаты работы программы определения максимального и минимального из четырех чисел

мы, где реализована как раз такая возможность. Это программа, которая по введенному пользователем текущему числу и номеру месяца определяет дату следующего дня. Данная задача не представляет сложности (нужно только прибавить единицу к текущему числу), но лишь для всех чисел месяца кроме последнего. В этом случае следующий день будет первым числом следующего месяца, а если последний день месяца — 31 декабря, то следующим днем будет первый день нового года. Необходимо также предусмотреть защиту от неверного ввода данных (чтобы пользователь случайно не ввел число, большее, чем количество дней в текущем месяце). В листинге 8.5 приводится текст программы, разработанной для решения данной задачи.

Листинг 8.5. Программа, которая определяет дату завтрашнего дня

```
program tomorrow;
Uses Crt;
 label 100;
 var
      d,m:integer; otv:string[3];
procedure next(kol:integer;var mes,n:integer);
begin
 if n>kol then
     begin
     writeln('Вы ошиблись. Введите верное число');
     Delay(60000);
     Halt
     end;
```

```
 if (n=31) and (mes=12) then
      begin
      n:=1;
      mes:=1;
      exit
      end;
 if n=kol then
       begin
       n:=1;
       mes:=mes+1
       end
          else
       n:=n+1;
end;
begin
 ClrScr;
 writeln('Введите сегодняшнее число');
 readln(d);
 writeln('Введите номер текущего месяца');
 readln(m);
 case m of
     1,3,5,7,8,10,12:next(31,m,d);
     4,6,9,11:next(30,m,d);
     2: begin
         100: writeln('Является ли текущий год високосным?(ДА/НЕТ)');
         readln(otv);
         if (otv<>'ДА') and (otv<>'НЕТ') then
             begin
             writeln('Вы дали неверный ответ. Введите ДА или НЕТ');
             goto 100;
             end
                                       else
            if otv='ДА' then next(29,m,d)
                        else next(28,m,d) end;
        end;
 writeln('Завтра будет ',d,'.',m);
 readln
end.
```

Разберем текст данной программы. Непосредственное определение числа и номера месяца следующего дня производится в процедуре `next`. В заголовке процедуры `next` описываются один параметр-значение `kol` — количество дней в текущем месяце, и два параметра-переменные — `mes` и `n`. При обраще-

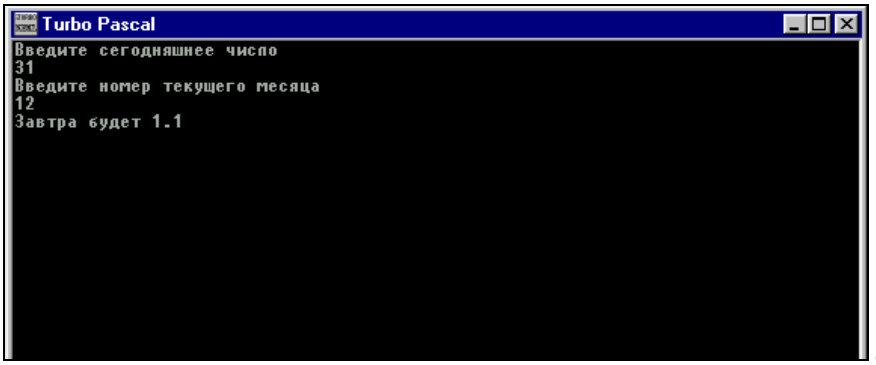
нии к процедуре next параметрам mes и n присваиваются значения соответственно текущего месяца и номера текущего дня в месяце. На выходе процедуры эти параметры должны получить значения, соответствующие завтрашнему месяцу и номеру дня. При этом возможны следующие варианты:

- день не является последним днем месяца. В этом случае номер следующего дня (n) увеличивается на единицу. Номер месяца (mes) при этом остается без изменений;
- день является последним днем месяца. В таком случае номер следующего дня становится равным единице, а номер месяца увеличивается на единицу;
- день является не только последним днем месяца, но и последним днем года. Тогда номер следующего дня становится равным единице. Первым становится и номер месяца, т. к. следующий день будет 1 января;
- при вводе данных была допущена ошибка (например, пользователь ввел номер дня в месяце, равный 32). В такой ситуации пользователю нужно сообщить о допущенной им ошибке, после чего программа в аварийном порядке должна прекратить работу.

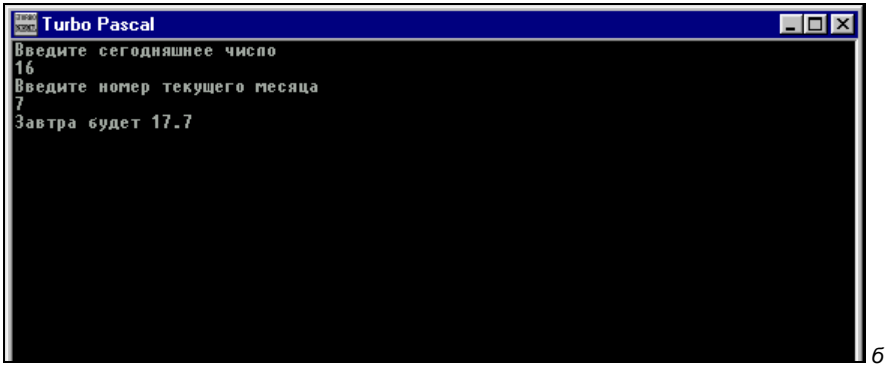
Все эти варианты и рассмотрены в процедуре next. В начале работы данной процедуры с помощью сокращенного условного оператора if проверяется правильность введенных данных (т. е. не больше ли переданное в процедуру текущее число n, чем количество дней в текущем месяце). Если при вводе была допущена ошибка, то процедура выводит сообщение об этом и прекращает работу программы, используя стандартную процедуру Halt. Чтобы пользователь успел прочитать сообщение о допущенной им ошибке, сообщение задерживается на некоторое время на экране компьютера с помощью стандартной процедуры Delay.

Следующий сокращенный условный оператор проверяет, не является ли введенное число 31 декабря. В этом случае параметрам mes и n присваиваются значения 1 и 1, а затем работа процедуры next завершается с помощью стандартной процедуры exit, после чего происходит возвращение в основную программу.

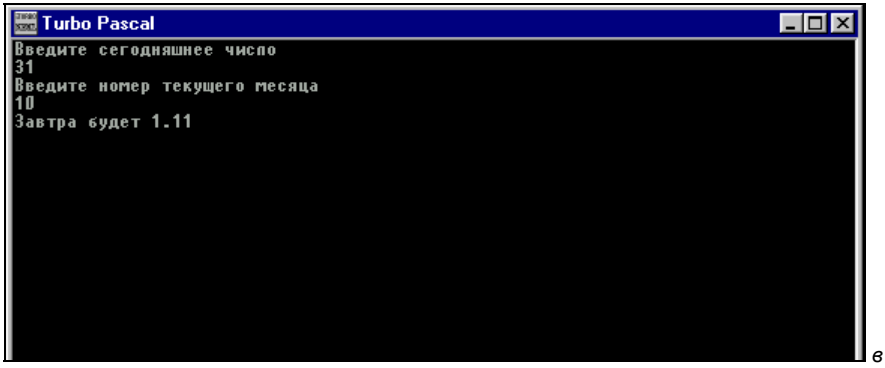
Посредством последнего в данной процедуре оператора if определяется следующее число для любого из остальных дней года. Если число не является последним в данном месяце (n<kol), то значение параметра n увеличивается на единицу, а mes остается без изменений. В противном случае n присваивается значение 1 (следующее число — начало нового месяца), а значение параметра mes увеличивается на единицу. На этом работа процедуры завершается, соответствующие формальным параметрам mes и n фактические параметры получают новые значения, которые передаются в основную программу.



а



б



в

Рис. 8.5. Результаты работы программы, определяющей дату завтрашнего дня, для последнего дня месяца *(а)*, для обычного дня в середине месяца *(б)* и для 31 декабря *(в)*

Из всего сказанного понятно, что для правильной работы процедуры нужно передать в нее правильную величину параметра-значения, которая равна действительному количеству дней в текущем месяце. Это количество определяется в основной части программы. Так как количество дней в месяце зависит от номера текущего месяца `m`, то оно определяется с помощью условного опе-

ратора case, в котором в роли переменной-селектора как раз и выступает m. Для m, равного 1, 3, 5, 7, 8, 10, 12 (т. е. для января, марта, мая, июля, августа, октября, декабря), данное значение равно 31. Для m, равного 4, 6, 9, 11 (т. е. для апреля, июня, сентября, ноября), значение будет равно 30.

Сложнее решается вопрос с февралем (m = 2), т. к. в этом случае значение, передаваемое в процедуру, зависит от того, является ли текущий год високосным или нет. В первом случае значение будет равно 29, во втором — 28. Необходимую информацию программа запрашивает у пользователя, который на вопрос о том, является ли текущий год високосным, должен дать положительный или отрицательный ответ. Для защиты от ошибочного ответа используется сокращенный условный оператор if. Если пользователь вместо "ДА" или "НЕТ" введет что-либо другое, программа выдаст сообщение об ошибке и с помощью оператора безусловного перехода вернет пользователя к тому месту программы, где вновь повторяется заданный вопрос.

При обращении из основной программы к процедуре next формальному параметру kol ставится в соответствие найденная с помощью оператора case величина, а формальным параметрам mes и n ставятся в соответствие фактические параметры m и d, содержащие введенные с клавиатуры значения номера текущего дня и текущего месяца.

После обращения к процедуре и производимых процедурой расчетов полученные новые значения параметров-переменных возвращаются в основную программу. Затем на экран компьютера выводятся эти новые значения (завтрашнее число и соответствующий ему номер месяца), и программа завершает свою работу.

На рис. 8.5 приводятся результаты работы описанной программы для различных исходных данных (для обычного дня в середине месяца, для последнего дня месяца и для 31 декабря).

8.3. Рекурсия

В предыдущем разделе данного пособия мы уже рассматривали программу, в которой имеется процедура, содержащая обращения к другим процедурам. Обращения к другим функциям и процедурам (как стандартным, так и созданным программистом) могут содержать и функции. При этом возможен и такой вариант, когда какая-либо функция или процедура содержит обращение к самой себе. Такая функция или процедура называется *рекуррентной* (от английского слова recurrence, что в переводе означает "возвращение" или "повторение"), а процесс вызова функции или процедуры из нее самой — *рекурсией*.

Процесс рекурсии осуществляется следующим образом: вначале происходит вызов некоторой функции (процедуры). Затем, еще до завершения своей работы, эта же функция (процедура) вызывает саму себя, и в результате в памяти компьютера появляется второй экземпляр той же функции (процедуры), второй экземпляр в ходе работы создает третий и т. д. Естественно, что подобный процесс может продолжаться до бесконечности, поэтому для решения какой-либо конкретной задачи в рекурсивной функции (процедуре) обязательно должен содержаться условный оператор, одна из ветвей которого обеспечивает прерывание этой бесконечной цепочки и завершение работы всех экземпляров функции (процедуры). Внутри этого оператора и должен находиться вызов функции.

В качестве несложного примера, наглядно иллюстрирующего процесс рекурсии, рассмотрим задачу вычисления целой положительной степени вещественного числа (x^y). Текст соответствующей программы приведен в листинге 8.6.

Листинг 8.6. Программа, вычисляющая степень вещественного числа

```
program stepint;
Uses Crt;
 var x,st:real; y:integer;
 function rek(a:real;k:integer):real;
   begin
   if k=1
   then rek:=a
   else rek:=rek(a,k-1)*a
   end;
begin
 ClrScr;
 writeln('Введите число');
 readln(x);
 writeln('Введите показатель степени ');
 readln(y);
 st:=rek(x,y);
 writeln('Степень числа равна ',st:12:3);
 readln
end.
```

В данной программе в основной ее части производится только ввод исходных данных (самого числа `x` и показателя степени `y`), вызов рекурсивной функции `rek` и вывод полученного результата, а само вычисление степени осуществляется в функции `rek`. Разберем более подробно механизм действия данной функции.

Для этого нужно ответить на вопрос: чему равно любое число в первой степени. Ответ очевиден — самому этому числу. Это мы и запишем в начале условного оператора: для k=1 результат rek равен исходному числу a. Для нахождения же квадрата числа нужно это число умножить само на себя, для нахождения куба — квадрат числа умножить на это число и т. д. Таким образом, существует следующая закономерность: k степень числа равна этому числу в степени k-1, умноженному на само число.

$a^k=a^{(k-1)}*a$

Эту формулу запишем в той ветви условного оператора, которая находится после служебного слова else.

Теперь рассмотрим непосредственно работу функции. Данная функция имеет два формальных параметра a и k, которым соответствуют в основной части программы фактические параметры x и y. Значением функции является a^k. Для вычисления значения функции rek с параметрами k и a вызывается другой экземпляр той же функции с параметрами k-1 и a (a остается неизменным для всех экземпляров функции), затем этот экземпляр вызывает следующий с параметром k-2 и т. д., пока процесс не доходит до обращения к экземпляру для k=1. Значение функции для k=1 уже известно, т. к. в этом случае оно равно a. Это значение передается в экземпляр для k=2 и т. д., пока не определяется значение для k. Это последнее значение передается в основную часть программы для вывода на экран компьютера.

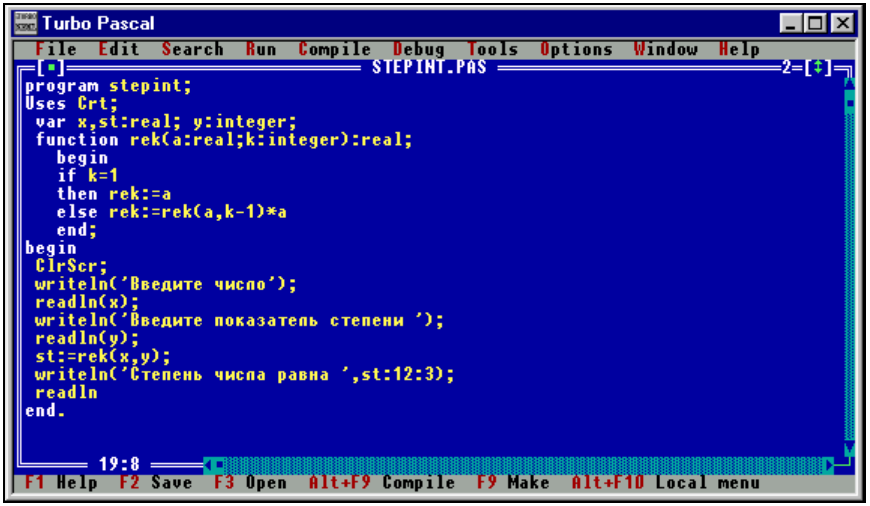
На рис. 8.6 приведен результат работы программы для x=2,53 и y=10.

В заключение данной главы приведем еще один пример использования рекурсии для решения классической математической задачи, известной как "Числа Фибоначчи". Эта задача была решена в XIII веке выдающимся итальянским математиком Леонардо Фибоначчи. Вот ее условия: пара кроликов каждый месяц дает потомство — двух кроликов, которые через два месяца сами способны давать новое потомство. Сколько кроликов будет через год, если в начале года была одна пара кроликов. Текст программы, которая решает поставленную задачу, приведен в листинге 8.7.

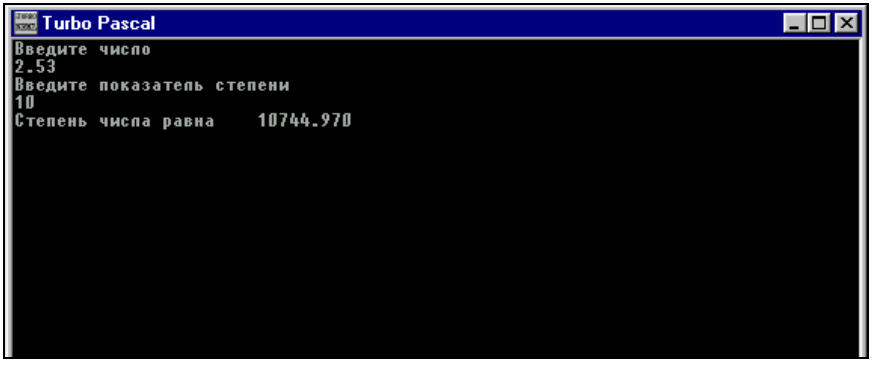
Листинг 8.7. Программа для решения задачи Фибоначчи

```
program rabbit;
Uses Crt;
 var i,k:integer;
 function rab(n:integer):integer;
  begin
  if n=0
  then rab:=1
```

```
  else
   if n=1
   then rab:=2
   else rab:=rab(n-2)+rab(n-1)
  end;
begin
 ClrScr;
 writeln('Введите количество месяцев');
 readln(i);
 k:=rab(i)*2;
 writeln('Число кроликов через ',i,' месяцев  равно ', k);
 readln
end.
```



а



б

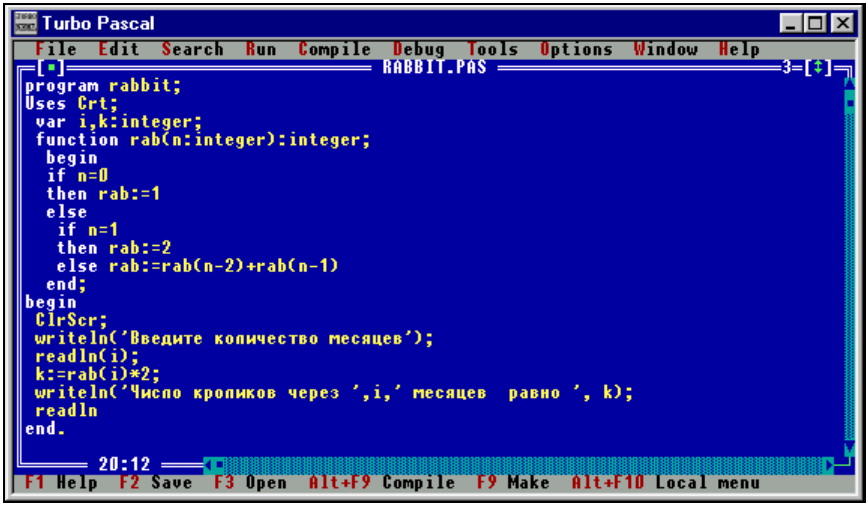
Рис. 8.6. Программа вычисления целой положительной степени вещественного числа *(а)* и результат ее работы *(б)*

Для решения данной задачи используем рекурсивную функцию `rab(n)`, которая определяет количество пар кроликов через `n` месяцев после начала года. В начале года существовала только одна пара кроликов, т. е. `rab(0)=1`, через месяц их стало две, следовательно `rab(1)=2`. Через 2 месяца количество пар кроликов увеличивается еще на 1, т. е. на столько, сколько их было в начале года. Таким образом `rab(2)=3` или `rab(2)=rab(1)+rab(0)`. Продолжая наши рассуждения, получаем следующую зависимость:

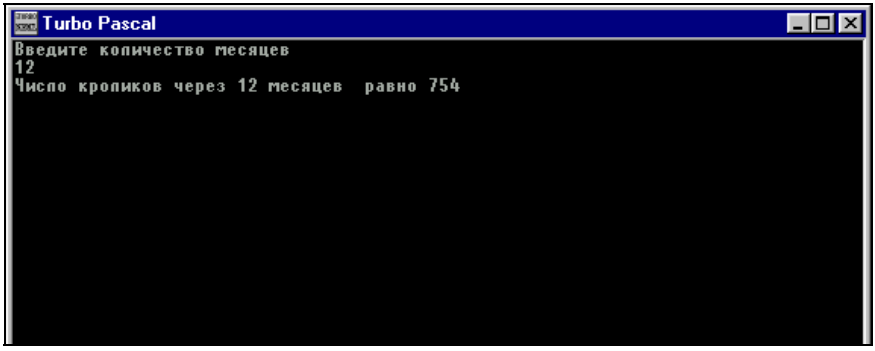
```
rab(n)=rab(n-1)+rab(n-2)
```

Эту рекурсивную функцию используем в программе `rabbit` (рис. 8.7), которая подсчитывает и выводит на экран количество кроликов через `n` месяцев.

Количество кроликов через `n` месяцев будет равно `rab(n)×2`. На рис. 8.7 приведены результаты работы программы при `n=12`, т. е. искомое количество кроликов через год.



а



б

Рис. 8.7. Программа "Числа Фибоначчи" *(а)* и результат ее работы *(б)*

8.4. Программируемые модули

В программах, которые мы ранее составляли в среде Turbo Pascal, широко использовался модуль `Crt`. Однако данный модуль является не единственным в этой системе программирования. Наряду с модулем `Crt` здесь имеется еще несколько модулей. Более того, сам язык Turbo Pascal имеет модульную структуру.

При запуске на выполнение головного файла системы turbo.exe в оперативную память компьютера по умолчанию загружается основной модуль языка Turbo Pascal, который называется `System`. Все остальные модули подключаются к основному с помощью уже известной нам команды `Uses`. Если возникает необходимость подключить к основному модулю сразу несколько дополнительных модулей, то они перечисляются после слова `Uses` через запятую. Каждый такой модуль представляет собой библиотеку, в которой содержится перечень ряда функций и процедур, а также описание каждой функции и процедуры, имеющейся в этом перечне.

Кроме уже известных нам модулей `System` и `Crt` в системе Turbo Pascal имеется еще ряд стандартных модулей. Это модуль `Graph`, который используется для создания рисунков, графиков и диаграмм в графическом режиме Turbo Pascal. Графический режим обеспечивает более высокое качество изображения, чем текстовый. Для использования возможностей операционной системы MS-DOS применяется модуль, который так и называется `Dos`. Для управления принтером, подключенным к персональному компьютеру, используется модуль `Printer`. Для того чтобы обеспечить совместимость версии языка Turbo Pascal 7.0 с более ранними версиями того же языка, используются модули `Turbo3` и `Graph3`. Подключение к основному модулю модулей `Turbo3` и `Graph3` позволяет запускать в среде Turbo Pascal 7.0 программы, созданные в более ранних версиях Turbo Pascal. Модуль `Overlay` используется для разработки программ, имеющих большой объем и сложную внутреннюю структуру. Для обеспечения более быстрой и эффективной работы со стандартными модулями они объединены в специальный библиотечный файл, который называется turbo.tpl. Расширение этого файла tpl является сокращением от английского *Turbo Pascal library* — библиотека Turbo Pascal.

Все перечисленные ранее модули не случайно названы стандартными, поскольку они уже имеются в системе Turbo Pascal в готовом виде и требуют лишь подключения с помощью команды `Uses`. Наряду с использованием готовых модулей, программист может создавать в системе Turbo Pascal свои собственные модули. Использование модулей во многих случаях оказывается более удобным и рациональным, чем создание процедур и функций внутри основной программы. Если к процедуре или функции можно обращаться

только из той программы, частью которой они являются, то обращение к модулю можно производить из любой программы, созданной в Turbo Pascal. Таким образом, один раз создав определенную функцию или процедуру и включив ее в состав модуля можно в дальнейшем многократно употреблять ее для расчетов в самых различных программах.

Модуль первоначально создается в виде исходного текста, который набирается с помощью встроенного редактора системы Turbo Pascal и сохраняется в файле с расширением pas. В общем виде модуль имеет такую внутреннюю структуру:

```
unit имя_модуля;
interface
var имя_перем1:тип1; имя_перем2:тип2; ...;
function имя_функции1(параметр1:тип1; параметр2:тип2; ...): значение;
function имя_функции2(параметр1:тип1; параметр2:тип2; ...): значение;
. . .
procedure имя_процедуры1(параметр1:тип1; параметр2:тип2; ...);
procedure имя_процедуры2(параметр1:тип1; параметр2:тип2; ...);
. . .
implementation
var имя_перем1:тип1; имя_перем2:тип2; ...;
function имя_функции1;
список локальных переменных функции1;
тело функции1;
function имя_функции2;
список локальных переменных функции2;
тело функции2;
. . .
procedure имя_процедуры1;
список локальных переменных процедуры1;
тело процедуры1;
procedure имя_процедуры2;
список локальных переменных процедуры2;
тело процедуры2;
begin
оператор1;
оператор2;
. . .
end.
```

Рассмотрим составные части приведенной структуры. Текст модуля начинается с заголовка. Заголовок состоит из служебного слова `unit` (что в переводе с английского и означает "модуль") и собственного имени модуля. Имя моду-

ля дается по тем же правилам, что и имя обычной программы, но на одном моменте здесь все же следует остановиться. Как читатель помнит, в обычной программе рекомендуется указывать в заголовке то же имя, что и имя файла, в котором эта программа хранится. Для модуля это не просто рекомендация, а *обязательное* правило. Если имя файла с исходным текстом модуля и имя модуля, указанное в его заголовке, будут разными, то модуль не будет правильно работать.

Вслед за заголовком модуля находятся его разделы, которые часто называют *секциями* модуля. Первой из них по порядку является *секция интерфейса*, которая начинается со служебного слова `interface`. Как известно, интерфейсом называются средства взаимодействия между компонентами и участниками компьютерной системы. В данном случае речь идет о программном интерфейсе, т. е. средстве взаимодействия между различными программами, входящими в состав программного обеспечения компьютера. В этом разделе перечисляются те переменные, функции и процедуры, которые входят в данный модуль и могут использоваться другими программами.

После служебного слова `interface` в секции интерфейса находится список внешних переменных. Этот список начинается со служебного слова `var`, и далее в нем перечисляются внешние переменные, для каждой из которых указывается ее тип.

Вслед за списком переменных идет список функций, содержащихся в данном модуле. Для каждой функции должен быть полностью указан ее заголовок. Этот заголовок включает служебное слово `function`, собственное имя функции, список ее параметров, заключенный в скобки (для каждого параметра должен быть указан его тип), и тип самой функции. Далее в секции интерфейса идет список содержащихся в модуле процедур. Заголовок каждой процедуры также должен быть приведен в этом разделе в полном виде, т. е. он должен включать в себя служебного слово `procedure`, собственное имя процедуры и список ее параметров.

Вслед за секцией интерфейса в модуле идет *секция реализации*, которая содержит тексты тех функций и процедур, заголовки которых имеются в интерфейсной секции. Секция реализации начинается со служебного слова `implementation` (это слово в переводе с английского означает "выполнение", "осуществление"). После слова `implementation` в разделе указывается список внутренних переменных, которые могут использоваться только внутри данного модуля.

Далее в секции реализации находятся описания функций данного модуля. Описание функции начинается с ее заголовка. В отличие от интерфейсной секции в данном случае в заголовке можно указывать только служебное слово `function` и собственное имя функции. Список параметров и тип функции

здесь указывать необязательно. После заголовка функции приводится список локальных переменных, которые используются только внутри данной функции. После списка локальных переменных находится тело функции. Тело функции в модуле имеет ту же структуру, что и тело функции в обычной программе. Оно начинается со слова `begin`, затем идет последовательность операторов, отделенных друг от друга точками с запятой. Завершается тело функции словом `end`, после которого идет точка с запятой. После текстов функций в секции реализации расположены описания процедур. Описание процедуры по своей внутренней структуре аналогично описанию функции.

Последней по порядку в модуле является *секция инициализации*, которая начинается со служебного слова `begin`, может содержать ряд операторов и заканчивается служебным словом `end`, после которого ставится точка. Следует сказать о том, что далеко не все перечисленные элементы модуля всегда присутствуют в нем. Часто в модуле отсутствует секция инициализации, но слово `end` с точкой в конце модуля должно находиться обязательно.

После того как исходный текст модуля набран, его следует сохранить в файле с расширением pas. Сохранять файл с исходным текстом модуля можно в том же каталоге prog, что и остальные файлы программ. Затем программу нужно запустить на компиляцию. Обратите внимание на то, что компиляция модуля должна производиться через меню **Compile**. Открыв этот раздел, нужно выполнить в нем одноименную команду. Если в модуле нет ошибок, то на экране компьютера на переднем плане вы увидите окно с сообщением об итогах компиляции. В нижней строке этого окна будет видна следующая надпись: **Compile successful: Press any key**. Это означает, что компиляция прошла успешно и для закрытия окна с сообщением нужно нажать любую клавишу (рис. 8.8). Запускать модуль сразу на компиляцию и на выполнение с помощью команды **Run | Run**, как мы это делали с обычными программами, нельзя.

После успешной компиляции система программирования создает исполняемый файл, который имеет то же имя, что и исходный, и расширение tpu. Например, если исходный файл назывался modul1.pas, то исполняемый файл будет называться modul1.tpu. Для того чтобы этот модуль могли использовать другие программы, нужно соответствующим образом настроить систему Turbo Pascal.

Для этого нужно открыть меню **Options**, а в нем выбрать команду **Directories**. После выполнения этой команды откроется диалоговое окно **Directories** (каталоги). Здесь в поле, называемом **EXE & TPU directory**, следует указать диск и каталог, в которых компилятор будет сохранять исполняемые файлы модулей с расширением tpu. В поле под названием **Unit directories** нужно указать этот же каталог для того, чтобы все программы, использующие модули, могли найти их по указанному адресу (рис. 8.9).

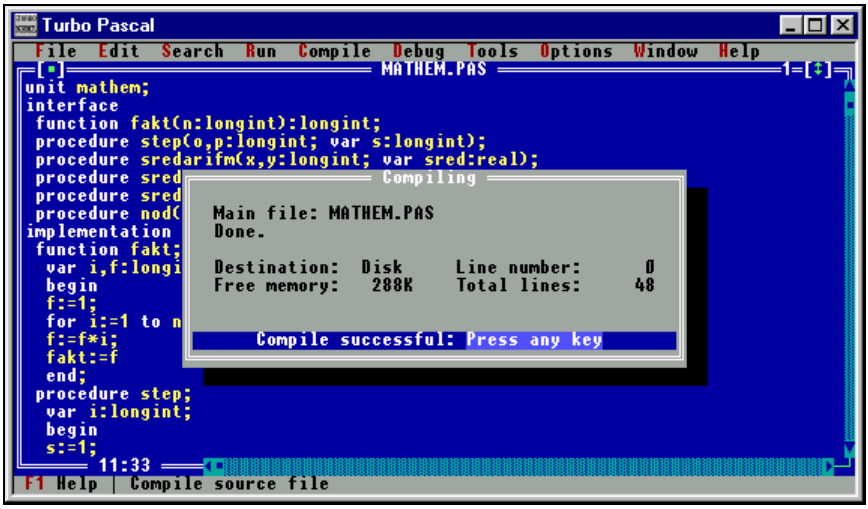


Рис. 8.8. Окно с сообщением об успешной компиляции модуля, созданного в системе Turbo Pascal

Если, например, на вашем компьютере папка TP7 находится в корневом каталоге диска D:, а файлы программ вы сохраняете в папке prog, то в обоих полях нужно указать адрес d:\TP7\prog. Для подтверждения внесенных изменений нужно щелкнуть мышью экранную кнопку **OK**, что приведет к закрытию диалогового окна. Теперь созданный вами модуль смогут использовать другие программы.

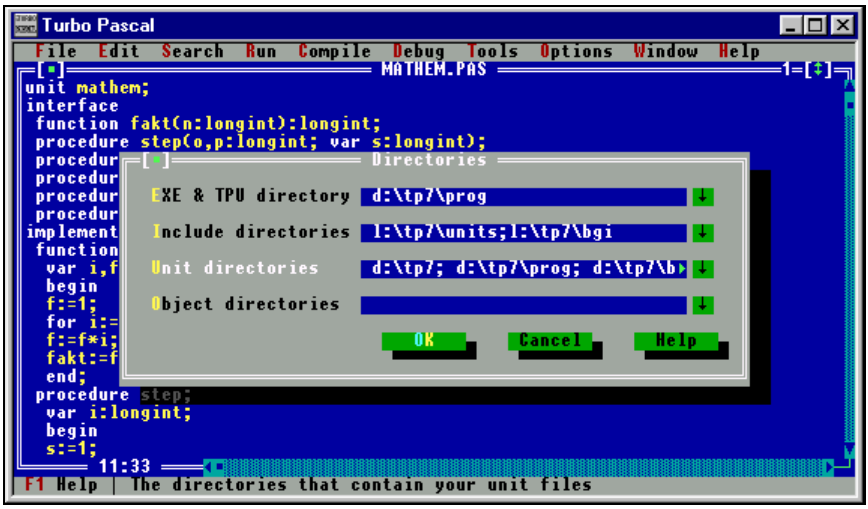


Рис. 8.9. Настройка системы Turbo Pascal, необходимая для правильного использования программных модулей

Рассмотрим создание и использование программного модуля на следующем примере. Создадим модуль под названием mathem, выполняющий те математические операции, для которых в языке Паскаль не существует стандартных функций или процедур. К таким операциям относится вычисление факториала числа, нахождение среднего арифметического, среднего геометрического и среднего гармонического двух чисел, определение наибольшего общего делителя двух чисел, возведение числа в произвольную целую степень. Заготовки для выполнения почти всех этих операций у нас уже имеются в виде функций или процедур, которые мы уже разрабатывали в предыдущих программах. Описание этих функций и процедур можно скопировать из этих программ и вставить в создаваемый нами модуль.

Исключение составляет только определение среднего геометрического, но эта задача не является сложной. Как известно, согласно определению, средним геометрическим двух чисел x и y является число z, которое удовлетворяет следующему равенству:

$$z = \sqrt{x \cdot y}\ .$$

На языке Паскаль эту же формулу можно записать в виде оператора:

```
z:=sqrt(x*y);
```

Из этого единственного оператора и будет состоять тело соответствующей процедуры. В листинге 8.8 приводится текст модуля mathem.

Листинг 8.8. Текст модуля mathem

```
unit mathem;
interface
 function fakt(n:longint):longint;
 procedure step(o,p:longint; var s:longint);
 procedure sredarifm(x,y:longint; var sred:real);
 procedure sredgeom(x,y:longint; var sred:real);
 procedure sredgarm(x,y:longint; var sred:real);
 procedure nod(x,y:longint; var d:longint);
implementation
 function fakt;
  var i,f:longint;
  begin
  f:=1;
  for i:=1 to n do
  f:=f*i;
  fakt:=f
  end;
```

```
 procedure step;
  var i:longint;
  begin
  s:=1;
for i:=1 to p do
  s:=s*o
  end;
 procedure sredarifm;
  begin
  sred:=(x+y)/2
  end;
 procedure sredgeom;
  begin
  sred:=sqrt(x*y)
  end;
 procedure sredgarm;
  begin
  sred:=2/(1/x+1/y);
  end;
 procedure nod;
  var i,k:longint;
  begin
  d:=1;
  if x>y
  then k:=x
else k:=y;
  for i:=1 to k do
    if (x mod i =0) and (y mod i =0)
    then d:=i;
  end;
end.
```

Модуль начинается с заголовка, в котором указано его собственное имя — `mathem`. В секции интерфейса данного модуля содержатся заголовки одной функции и пяти процедур, находящихся в данном модуле. В секции реализации содержится описание указанных функций и процедур. Функция `fakt` вычисляет факториал целого числа. Процедура `step` применяется для возведения числа в любую целую степень. Процедура `sredarifm` находит среднее арифметическое двух целых чисел. Процедура `sredgeom` вычисляет среднее геометрическое двух чисел, а процедура `sredgarm` находит среднее гармоническое. Наконец, процедура `nod` находит наибольший общий делитель двух чисел. Подробно описывать эти процедуры нет необходимости, т. к. это было

уже сделано ранее. Секция инициализации в данном модуле отсутствует. Завершается модуль служебным словом `end` с точкой.

После компиляции модуля образуется исполняемый файл, который называется mathem.tpu. Этот файл могут использовать другие программы, написанные в Turbo Pascal. В качестве примера использования возможностей созданного модуля приведем две небольшие программы. Первая программа вычисляет сумму факториалов двух введенных с клавиатуры чисел. В листинге 8.9 приводится текст данной программы.

Листинг 8.9. Программа, вычисляющая сумму факториалов двух чисел

```
program sumfakt;
uses crt,mathem;
 var a,b,sumf:longint;
begin
 ClrScr;
 writeln('Введите первое число');
 readln(a);
 writeln('Введите второе число');
 readln(b);
 sumf:=fakt(a)+fakt(b);
 writeln('Сумма факториалов двух чисел равна ',sumf);
 readln
end.
```

В данной программе после ее заголовка указываются подключаемые к программе модули. После стандартного модуля `Crt` через запятую указывается созданный нами модуль `mathem`. Далее в программе после ввода исходных данных дважды идет обращение к функции `fakt`, содержащейся в модуле `mathem`. С помощью этой функции вычисляется искомый результат — сумма факториалов двух чисел, который затем выводится на экран компьютера. Результаты работы программы для `a`, равного 3, и `b`, равного 5, приведены на рис. 8.10.

Следующая программа для двух введенных с клавиатуры чисел определяет их среднее арифметическое, среднее геометрическое и среднее гармоническое. В листинге 8.10 приводится текст данной программы.

Листинг 8.10. Программа, вычисляющая среднее арифметическое, геометрическое и гармоническое двух чисел

```
program srednee;
  uses crt,mathem;
 var a,b:integer; arifm,geom,garm:real;
```

```
begin
 ClrScr;
 writeln('Введите первое число');
 readln(a);
 writeln('Введите второе число');
 readln(b);
 sredarifm(a,b,arifm);
 sredgeom(a,b,geom);
 sredgarm(a,b,garm);
 writeln('Среднее арифметическое двух чисел равно ',arifm:6:2);
 writeln('Среднее геометрическое равно ',geom:7:3);
 writeln('Среднее гармоническое равно ',garm:7:3);
 readln
end.
```

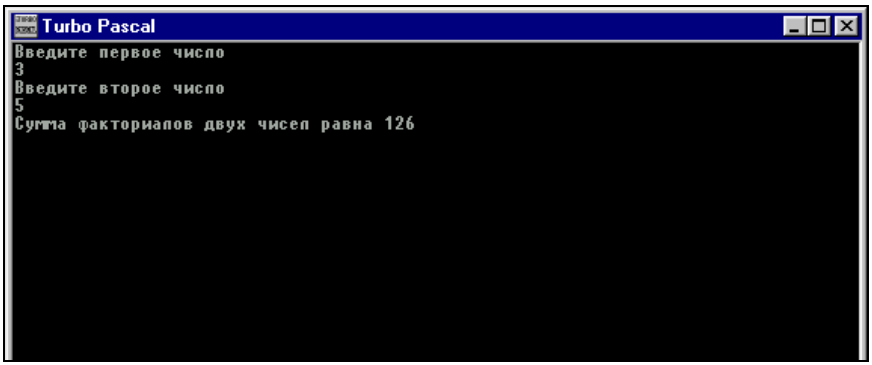


Рис. 8.10. Результат работы программы, которая определяет сумму факториалов двух чисел

В данной программе наряду с модулем `Crt` также производится подключение модуля `mathem`. После ввода исходных данных в основной части программы трижды присутствует обращение к процедурам, содержащимся в модуле `mathem`. С помощью процедур определяются искомые результаты, которые затем выводятся на экран компьютера. Результат работы программы для тех же исходных данных, что и в предыдущей программе (числа 3 и 5) показан на рис. 8.11.

Простота и компактность двух приведенных ранее программ обеспечивается за счет использования в обеих программах возможностей созданного нами модуля `mathem`. Обращаться к этому же модулю можно и из других программ, созданных в системе Turbo Pascal, что в значительной степени облегчает их разработку.

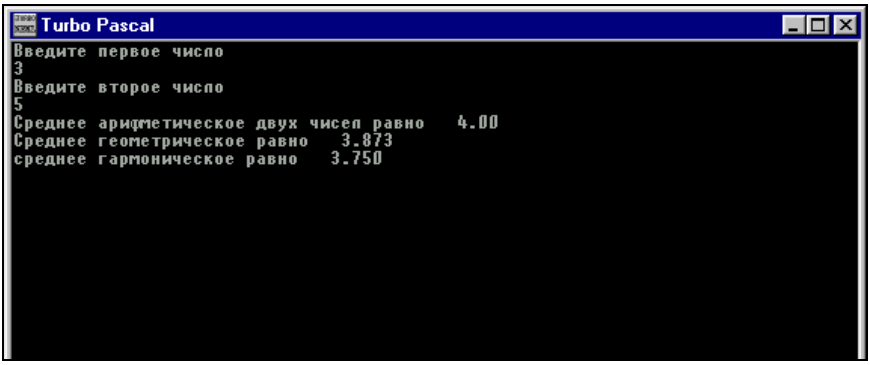


Рис. 8.11. Результат работы программы, которая определяет для двух введенных чисел их среднее арифметическое, среднее геометрическое и среднее гармоническое

Задания для самостоятельной работы

Задание 1. Составьте программу, которая определяет площадь треугольника по формуле Герона. Данная формула выглядит следующим образом:

$$S = \sqrt{p(p-a)(p-b)(p-c)},$$

где a, b и c — длины сторон треугольника, а p — половина периметра треугольника. Для извлечения квадратного корня используйте стандартную функцию `sqrt`.

Задание 2. Составьте программу, которая по выбору пользователя возводит введенное с клавиатуры целое число в квадрат, определяет его абсолютную величину или извлекает квадратный корень числа. В программе следует предусмотреть вывод сообщения об ошибке, если пользователь попытается извлечь квадратный корень из отрицательного числа. Используйте в программе стандартные функции `abs`, `sqr` и `sqrt`.

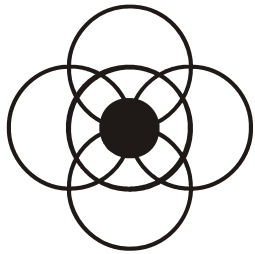
Задание 3. Дополните предыдущую программу функцией, которая определяет, действительно ли введенное с клавиатуры число является целым (целое число может содержать только цифры от 0 до 9). Результат функции должен быть логического типа, т. е. либо `true`, либо `false`. В зависимости от значения функции программа должна предложить пользователю либо произвести на выбор один из указанных ранее вариантов вычислений, либо вывести сообщение об ошибке и вернуть пользователя к вводу исходных данных.

Задание 4. Составьте программу, которая заполняет одномерный массив из 20 элементов случайными целыми числами от 1 до 99, а затем определяет,

сколько в массиве имеется простых чисел (простым числом называется такое, которое делится только на единицу или само на себя). Процесс определения того, является ли число простым, оформить в виде отдельной процедуры.

Задание 5. Составьте программу, которая определяет в интервале от 1 до 10 000 все совершенные числа. Совершенным числом называется число, которое равно сумме всех своих делителей, включая единицу. Процесс определения того, является ли данное число совершенным, оформите в виде отдельной процедуры.

ГЛАВА 9



Работа с файлами в Паскале

Еще на начальных этапах работы в системе Turbo Pascal *(см. разд. 3.1)* мы научились сохранять в виде файлов тексты написанных на языке Паскаль программ (исходный код). Но до сих пор мы теряли результаты работы созданных нами программ. Эти результаты пропадали после выхода из системы программирования. А ведь именно получение конкретных результатов в конечном счете и является целью создания программ. Отсюда можно сделать следующий вывод: желательно предусмотреть возможность сохранения этих результатов в долговременной памяти компьютера. Как известно, именно файл является основной единицей хранения любой информации на носителях информации вне зависимости от их устройства и физической природы (к таким носителям относятся и жесткий магнитный диск — винчестер, и гибкие магнитные диски, называемые также дискетами или флоппи-дисками, и лазерные компакт-диски).

Каждый файл имеет уникальное, т. е. присущее только ему, имя и расширение, определяющее его тип. При этом имя файла от расширения отделяется точкой. Нам уже знакомы файлы с расширением pas, содержащие исходный код программ на Паскале, и файлы модулей с расширением tpu, но кроме них существуют еще десятки различных видов расширений, некоторые из них мы приведем:

- exe — такое расширение имеют готовые к выполнению программы (в том числе, как вы конечно помните, головной файл системы программирования Turbo Pascal);
- doc — это расширение имеют файлы документов, созданных в текстовом процессоре Microsoft Word;
- xls — данное расширение имеют файлы электронных таблиц, созданных программой Microsoft Excel;

- bmp — это расширение получают файлы рисунков, созданные стандартным приложением операционной системы Windows графическим редактором Paint.

Соответствующие файлы могут иметь, например, такие имена:

progr.exe doklad.doc tabl.xls picture.bmp

Над файлами средствами языка Паскаль можно производить различные операции, которые можно разделить на две большие группы. Это операции записи информации в файл из оперативной памяти компьютера и считывания информации из файла в оперативную память. При записи информации в файл необходимо выполнить следующие операции: открытие файла для записи, собственно запись информации в файл, закрытие файла. При чтении информации из файла выполняются операции: открытие файла для чтения, собственно чтение файла и закрытие файла.

Существует два способа записи и чтения информации — метод последовательного доступа и метод прямого доступа. *Метод последовательного доступа* можно сравнить с процессом записи на магнитофонную пленку и воспроизведения ее содержимого. Для того чтобы произвести одну или другую операцию, требуется предварительно перемотать пленку до нужного места. Так же и для того, чтобы добраться в файле до интересующей нас информации методом последовательного доступа, нужно предварительно "прокрутить" всю предшествующую ей информацию. При считывании информации *методом прямого доступа* можно непосредственно получить доступ к нужному нам компоненту (т. е. составной части файла) подобно тому, как можно непосредственно проиграть любую запись на лазерном диске.

Файлы, с которыми можно работать средствами языка Паскаль, делятся на три основные разновидности: текстовые, типизированные и нетипизированные. Используемые файлы, подобно переменным, константам и массивам, должны быть обязательно описаны, причем для каждого из трех видов файлов существует свой способ описания.

9.1. Текстовые файлы

Начнем рассмотрение процесса работы с *текстовыми* файлами. Перед использованием такого файла в программе в разделе описания переменных должна быть обязательно описана связанная с ним переменная. Общий вид описания следующий:

```
var имя_переменной:text;
```

где *имя_переменной* — имя переменной, которая впоследствии будет связана с файлом; `text` — соответствующий тип переменной.

Пример описания:

```
var k:text;
```

Текстовые файлы состоят из строк символов. Каждая такая строка заканчивается символами возврата каретки CR (этот символ имеет в таблице ASCII код 13) и перевода строки LF (данный символ имеет в таблице код 10). Упомянутые символы относятся к так называемым *управляющим* символам, которые, как правило, при просмотре и редактировании файла не выводятся на экран компьютера. В конце файла находится признак конца файла, который сокращенно обозначается EOF (сокращение от английской фразы end of file — конец файла). Файл такого вида имеет расширение txt. Этот файл можно создавать как с помощью программы, написанной на Паскале, так и с помощью какого-либо текстового редактора. Чтобы потом не возникало проблем с чтением файла, лучше использовать редакторы, в которых по умолчанию используется кодировка букв кириллицы, принятая в MS-DOS, например встроенный редактор файловой оболочки Norton Commander или самой системы программирования Turbo Pascal.

При этом следует иметь в виду: для того чтобы с файлом можно было в дальнейшем производить какие-либо действия, необходимо выполнить еще две операции. Во-первых, связать имя файла с именем соответствующей ему переменной. Во-вторых, указать направление работы с файлом — запись в него информации или чтение информации.

Общий вид процедуры, связывающей имя файла с именем переменной, следующий:

```
assign (имя_переменной, 'имя_файла');
```

где assign — служебное слово, означающее в переводе "назначить"; *имя_переменной* — имя соответствующей файлу переменной; '*имя_файла*' — полное имя текстового файла. Обратите внимание, что оно должно быть заключено в апострофы. Полное имя файла содержит имя диска, на котором находится данный файл, путь к файлу, который включает в себя имена каталогов, содержащих данный файл, и собственное имя файла с расширением. Если текстовый файл находится на том же диске и в том же каталоге, что и вызывающая его программа, то диск и каталог можно не указывать.

Пример использования процедуры:

```
assign (k,'A:otchet.txt');
```

здесь переменной k ставится в соответствие файл otchet.txt, который находится на дискете, содержащейся в дисководе A: того компьютера, с которым работает пользователь.

Данная процедура устанавливает соответствие между описанной ранее переменной k и текстовым файлом otchet.txt. В дальнейшем в программе имя тек-

стового файла более не будет упоминаться, а при всех операциях с файлом будет указываться имя "назначенной" ему текстовой переменной.

Дальнейший ход работы зависит от того, какие именно действия необходимо произвести с данным файлом. Если содержимое файла необходимо прочитать, то его открытие производится следующей процедурой:

```
reset (имя_переменной);
```

где *имя_переменной* — имя связанной с файлом переменной.

Что касается непосредственно самого процесса чтения информации из файла в программу, то он происходит с помощью уже знакомых нам операторов `read` или `readln`. При этом в скобках после служебного слова `read` или `readln`, как правило, указываются имена двух переменных: первой пишется имя переменной, связанной с файлом, а за ней указывается имя какой-либо вспомогательной переменной, в которую и считывается информация. По окончании операции чтения (как и рассматриваемой далее операции записи) файл следует закрыть с помощью процедуры `close`.

Общий вид данной процедуры:

```
close(имя_переменной);
```

где *имя_переменной* — имя связанной с файлом переменной.

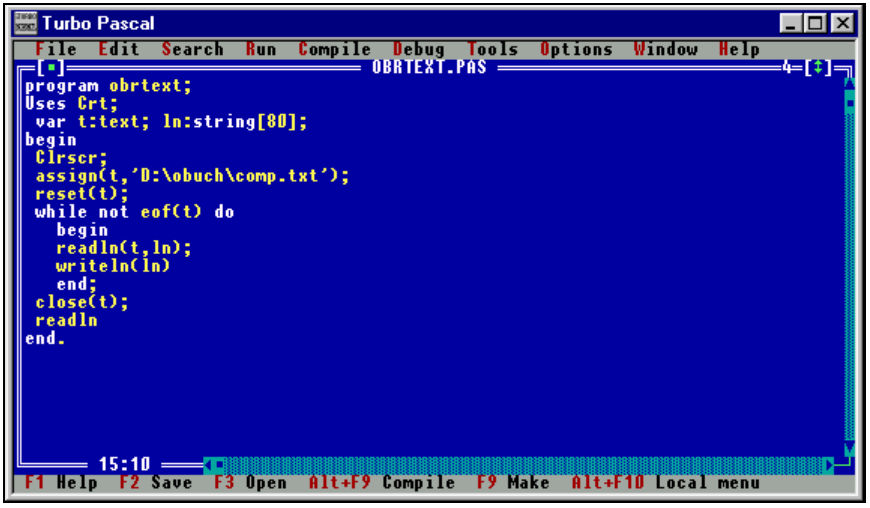
Рассмотрим теперь операцию чтения информации из файла на конкретном примере.

Составим программу, которая считывает информацию из файла, созданного с помощью встроенного редактора файловой оболочки Norton Commander. Этот файл находится на логическом диске D: компьютера в каталоге obuch. (Напомним, что для того, чтобы в Norton Commander создать новый текстовый файл, нужно нажать комбинацию клавиш <Shift>+<F4>, а в дальнейшем редактировать уже существующий файл можно, выделив его курсорной рамкой и нажав клавишу <F4>.) Файл имеет имя comp.txt и содержит информацию об устройстве компьютера. Рассмотрим текст программы, выполняющей указанные действия — листинг 9.1.

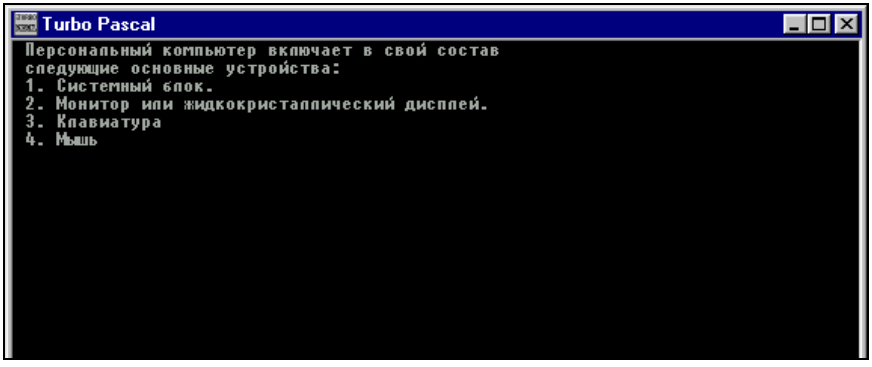
Листинг 9.1. Программа, считывающая информацию из текстового файла и выводящая ее на экран компьютера

```
program obrtext;
Uses Crt;
 var t:text; ln:string[80];
begin
 Clrscr;
 assign(t,'D:\obuch\comp.txt');
```

```
 reset(t);
 while not eof(t) do
   begin
   readln(t,ln);
   writeln(ln)
   end;
 close(t);
 readln
end.
```



а



б

Рис. 9.1. Программа чтения текстового файла *(а)* и результат ее работы *(б)*

Программа называется `obrtext`, и в ней используются две переменные: `t`, которой ставится в соответствие файл comp.txt, и `ln` — вспомогательная переменная строкового типа, в которую строка за строкой будет считываться со-

держимое текстового файла. В начале основной части программы между файлом и файловой переменной устанавливается соответствие с помощью процедуры assign. Затем файл открывается для чтения процедурой reset.

Сам процесс считывания информации производится в цикле с предусловием типа while. Условием завершения работы цикла является обнаружение признака конца файла (eof), т. е. цикл будет выполняться при условии, что конец файла в ходе считывания информации еще не найден. Это условие записывается следующим образом: not eof(t), т. е. здесь используется стандартная функция Паскаля, которая также называется eof и относится к логическим функциям, значением которых может быть только false или true. Пока не обнаружен конец файла, функция имеет значение false, а при его обнаружении она меняет значение на true. Аргументом же данной функции является имя файловой переменной. При каждом выполнении тела этого цикла из файла оператором readln считывается очередная строка и передается в строковую переменную ln. Затем содержимое строковой переменной (т. е. очередная строка файла) выводится на экран компьютера оператором writeln. По окончании работы цикла файл закрывается процедурой close.

Процесс записи информации в файл также имеет свои особенности. Назначение файлу соответствующей ему файловой переменной производится так же, как и в случае чтения файла, но открытие файла для записи производится либо процедурой rewrite, либо процедурой append. Общий вид этих процедур следующий:

```
rewrite (имя_переменной);
append (имя_переменной);
```

т. е. он аналогичен виду процедуры reset. Надо помнить, что процедуры rewrite и append открывают файл для записи по-разному. Процедура rewrite при открытии файла очищает его, т. е. предыдущее содержимое файла (если таковое было) удаляется и вместо него записывается новый текст. Если же содержимое файла нужно сохранить, то используют процедуру append, которая дополняет текущее содержимое текстового файла новой информацией, дописывая ее в конец данного файла. Сам же процесс записи информации в файл производится либо с помощью оператора write, либо оператором writeln. После служебного слова write или writeln в скобках указывается имя файловой переменной, а затем имя той строковой переменной, из которой осуществляется запись данных в файл.

Рассмотрим использование процедуры append на следующем примере. Пусть в конец файла comp.txt необходимо дописать следующий текст:

```
  К компьютеру также могут подключаться:
5. Принтер
6. Сканер
```

```
7. Колонки
8. Модем
```

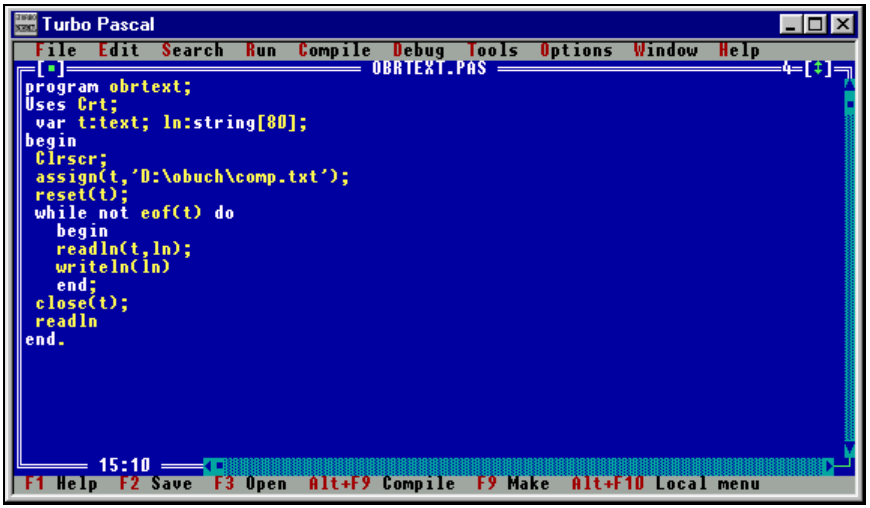
Для того чтобы выполнить требуемое действие, составим соответствующую программу. Текст программы приведен в листинге 9.2.

Листинг 9.2. Программа, которая позволяет добавлять информацию в конец текстового файла

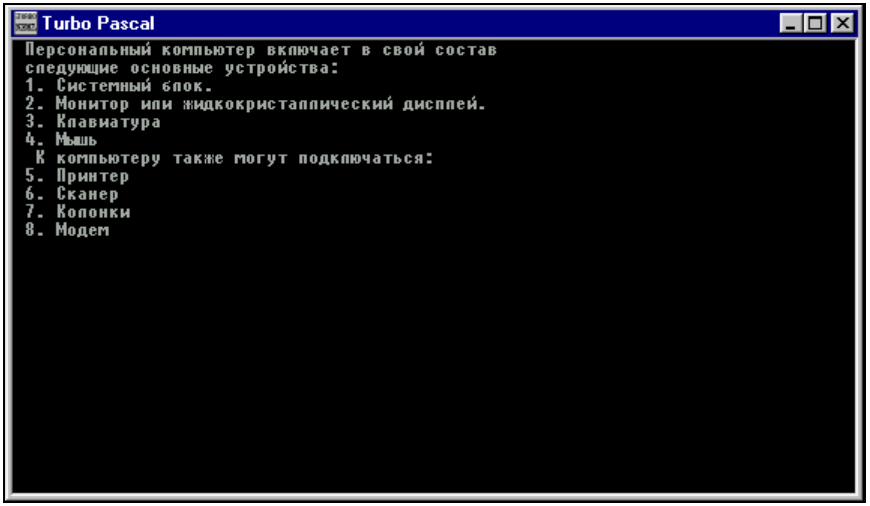
```
program doptext;
Uses Crt;
 var t:text; ln:string[80];
     i,n:integer;
begin
 Clrscr;
 assign(t,'D:\obuch\comp.txt');
 append(t);
 writeln('Укажите количество вводимых строк');
 readln(n);
 writeln('Вводите текст. Ввод каждой строки заканчивайте нажатием
"Enter"');
 for i:=1 to n do
   begin
   readln(ln);
   writeln(t,ln)
   end;
 close(t);
 Clrscr;
 reset(t);
 while not eof(t) do
   begin
   readln(t,ln);
   writeln(ln)
   end;
 close(t);
 readln;
end.
```

В этой программе помимо тех переменных, которые мы использовали в предыдущей, нам понадобятся еще две: `n` — количество строк, добавляемых в конец файла, и `i` — счетчик цикла. Затем после открытия файла для записи указываем количество вводимых строк. Сам процесс ввода происходит в цикле с заранее заданным числом повторений (это число равно количеству вводимых строчек; в данном случае оно равно 5). В теле цикла содержатся

два оператора. В первом из них каждая вводимая строка оператором `readln` считывается во вспомогательную переменную `ln`. Во втором операторе содержимое переменной `ln` (т. е. очередная строка) переписывается в файл с помощью оператора `writeln`. По окончании операции записи файл закрывается.



а



б

Рис. 9.2. Просмотр результатов добавления строк в текстовый файл: программа *(а)* и результат ее работы *(б)*

Для того чтобы можно было проконтролировать правильность ввода в файл дополнительной информации, после записи данных произведем чтение всего

содержимого файла с выводом его на экран компьютера аналогично тому, как это было сделано в предыдущей программе. Результат выполненных операций можно наглядно увидеть на рис. 9.2.

9.2. Типизированные файлы

Типизированным называется файл, который состоит из однородных элементов, относящихся к одному и тому же типу. Такими элементами могут быть переменные различных типов и массивы. Каждый элемент такого файла имеет свой индекс, подобно элементам массива. Но в отличие от обычного массива, который хранится в оперативной памяти компьютера и содержимое которого стирается после выхода из системы программирования, типизированный файл может длительное время сохраняться на жестком диске компьютера или на дискете. К каждому элементу такого файла возможен как прямой (по индексу элемента), так и последовательный доступ, но недостатком его является то, что в отличие от текстового файла его нельзя просматривать и редактировать с помощью обычного текстового редактора.

Для того чтобы в типизированный файл можно было производить запись или считывать из него информацию, его следует связать с помощью процедуры `assign` с файловой переменной подобно тому, как мы это уже делали с текстовыми файлами. Например, так:

```
assign(nab,'D:\inf\nabor');
```

Здесь мы связываем имеющуюся в программе файловую переменную `nab` с типизированным файлом nabor, находящимся на логическом диске D: в каталоге inf. Обратите внимание, что в отличие от текстового файла, типизированный файл необязательно должен иметь расширение.

В общем виде описание файловой переменной для типизированного файла выглядит следующим образом:

`var` *имя_переменной*`: file of` *тип*`;`

где *имя_переменной* — имя переменной, связанной с типизированным файлом; `file` — служебное слово; *тип* — тип элементов, из которых состоит данный файл. Это могут быть уже известные нам типы, используемые для описания обычных переменных и массивов (`integer`, `real`, `character`, `string`, `array`).

Пример описания типизированного файла:

```
var nab: file of character;
```

таким образом описан файл, состоящий из элементов вещественного типа, с которым связана в программе файловая переменная `nab`.

Подобно текстовым файлам, типизированные файлы открываются для чтения процедурой `reset` и для записи процедурой `rewrite`, а закрываются процедурой `close`.

Непосредственное чтение данных из файла производится с помощью следующего оператора:

`read(`*имя_переменной1*`,`*имя_переменной2*`);`

где *имя_переменной1* — имя файловой переменной; *имя_переменной2* — имя обычной переменной, относящейся к тому же типу, что и элементы типизированного файла.

Пример: `read (nab,r)`, где `r` — обычная переменная вещественного типа.

Запись в файл производится оператором:

`write(`*имя_переменной1*`, `*имя_переменной2*`);`

Операторы `writeln` и `readln` для записи и чтения в типизированном файле использовать нельзя, т. к. данный файл не содержит строк. Поэтому при попытке их использования компилятор выдаст сообщение об ошибке.

Типизированные файлы, ввиду упорядоченности их структуры, удобно использовать для создания электронных справочников, что мы и рассмотрим на следующем примере.

Известно, что в математике, физике и других точных науках для обозначения различных величин широко используются буквы греческого алфавита. Составим программу, которая облегчит пользователю запоминание названий этих букв и порядка их расположения в алфавите. Эта программа должна по порядковому номеру буквы в алфавите определять ее название и, наоборот, по названию буквы указывать ее положение в алфавите.

Поставленную задачу мы будем решать в два этапа. Вначале создадим типизированный файл, элементами которого будут строковые переменные — названия букв греческого алфавита. Этот файл, имеющий имя alphabet и находящийся в каталоге obuch на логическом диске D: компьютера, мы создадим и заполним данными с помощью программы, текст которой приведен в листинге 9.3.

Листинг 9.3. Программа, записывающая названия букв греческого алфавита в типизированный файл

```
program greek;
Uses Crt;
 var
    a:file of string; b:string; i,j:integer;
```

```
begin
 Clrscr;
 assign(a,'D:\obuch\alphabet');
 rewrite(a);
 for i:=1 to 24 do
   begin
   writeln('Введите название очередной буквы');
   readln(b);
   write(a,b)
   end;
 close(a);
 Clrscr;
 reset(a);
 for i:=1 to 4 do
   begin
   writeln;
   for j:=1 to 6 do
     begin
     read(a,b);
     write(b,' ')
     end;
   end;
 close(a);
 readln;
end.
```

Вначале мы связываем типизированный файл с файловой переменной `a` с помощью процедуры `assign`, затем открываем файл для записи. Если файла с таким именем на диске не существовало, то он будет создан программой. Процесс заполнения созданного файла данными мы производим в цикле с заранее заданным числом повторений. Тело цикла будет повторено 24 раза (по числу букв греческого алфавита). В самом теле цикла производятся следующие действия: название очередной буквы, введенное с клавиатуры, считывается оператором `readln` во вспомогательную переменную строкового типа `b`. Затем оператором `read` значение переменной `b` передается файловой переменной `a` (т. е. в конечном счете в типизированный файл alphabet).

Для того чтобы проконтролировать правильность заполнения файла alphabet, выведем его содержимое на экран компьютера. Элементы файла выводятся на экран в 4 строки, в каждой из которых должны быть видны названия 6 букв. Такой способ вывода осуществляется с помощью двух циклов, один из которых вложен в другой. Элементы цикла (названия букв) считываются из файла в ту же вспомогательную переменную `b`. По окончании вывода данных эта программа завершает свою работу.

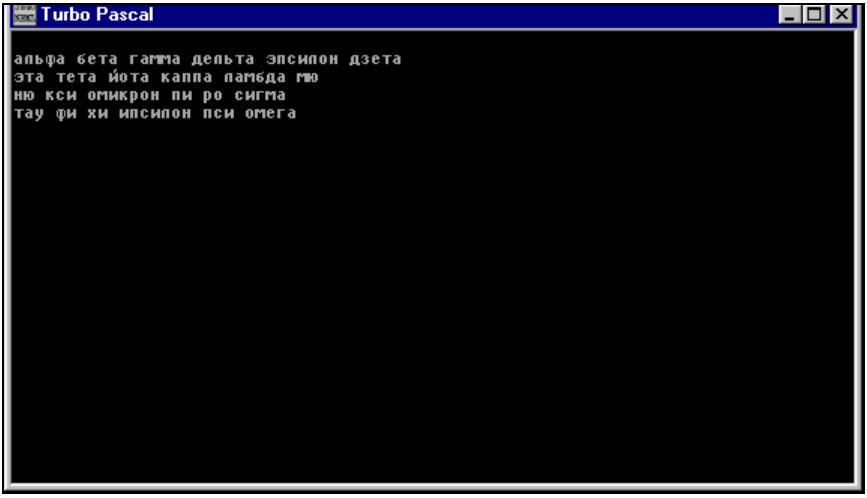


Рис. 9.3. Результаты работы программы создания типизированного файла, содержащего буквы греческого алфавита

Данные из файла, созданного в программе `greek`, мы используем во второй программе, которая уже непосредственно будет выполнять поставленную задачу. Для того чтобы разобраться в том, как работает эта вторая программа, необходимо предварительно рассмотреть используемые в ней стандартные процедуру и функцию, которые применяются для работы с типизированными файлами.

Процедура `seek` осуществляет прямой доступ к элементам типизированного файла. Общий вид данной процедуры:

```
seek(имя_переменной, n);
```

где `seek` — служебное слово, которое в переводе с английского означает "искать"; *имя_переменной* — имя файловой переменной; `n` — порядковый номер элемента в файле (необходимо иметь в виду, что нумерация элементов в типизированном файле начинается не с единицы, а с нуля).

Результатом работы данной процедуры является то, что элемент файла с номером `n` становится текущим. При считывании данных оператором `read` будет считан именно этот элемент, при записи данных оператором `write` они также будут записаны в этот элемент.

Если в ходе работы программы необходимо установить, какой элемент файла является текущим, то для этого используется функция `filepos`.

Общий вид описания данной функции:

```
filepos(имя_переменной):longint;
```

аргументом данной функции является имя файловой переменной, а значением — порядковый номер текущего элемента в файле.

Теперь можно приступить к разбору второй программы, текст которой приводится в листинге 9.4.

Листинг 9.4. Программа, определяющая название буквы греческого алфавита по ее порядковому номеру или номер буквы в алфавите по ее названию

```
program grsearch;
Uses Crt;
label 10;
var a:file of string; s:char; b,c:string; l,m,n,x:longint;
begin
assign(a,'D:\obuch\alphabet');
reset(a);
 repeat
 ClrScr;
 10: writeln('Выберите режим работы программы');
 writeln('1 - определение названия буквы по ее номеру в алфавите');
 writeln('2 - определение порядкового номера буквы по ее названию');
 readln(m);
 case m of
     1: begin
        writeln('Введите порядковый номер буквы в алфавите');
        readln(n);
        seek(a,n-1);
        read(a,b);
        writeln(n,' буква греческого алфавита - ',b);
        end;
     2: begin
        writeln('Введите название буквы');
        readln(c);
        l:=0;
        repeat
            seek(a,l);
            read(a,b);
            l:=l+1
        until c=b;
        x:=filepos(a);
        writeln('Буква ',c,' является ',x,' буквой греческого алфавита')
        end;
     else begin
        writeln('Вы ввели неверную цифру');
```

```
        goto 10
        end;
 end;
 writeln('Будете продолжать работу с программой(Д/Н)');
 readln(s);
 until s='H';
close(a);
end.
```

В начале работы данной программы открывается для чтения типизированный файл alphabet, созданный предыдущей программой. Затем в программе используется цикл с постусловием. В этом цикле после очистки экрана операторами `writeln` на экран выводится текст меню, позволяющий выбрать один из двух вариантов работы программы — определение названия буквы или определение порядкового номера путем ввода соответствующей цифры 1 или 2. Эта цифра присваивается вспомогательной переменной `m`. Затем выбор варианта работы производится с помощью условного оператора `case`, в котором роль переменной-селектора играет `m`. Перед началом работы условного оператора файловая переменная `a` связывается с типизированным файлом alphabet, а затем файл открывается для чтения процедурой `reset`.

В случае если пользователь выбирает первый вариант работы программы, ему необходимо ввести порядковый номер буквы — `n`. Значение `n` используем в качестве одного из параметров процедуры `seek` (другим является имя файловой переменной), но при этом нужно учесть, что в алфавите нумерация букв начинается с единицы, следовательно, первой букве алфавита будет соответствовать нулевой элемент типизированного файла, второй букве — первый элемент, `n`-й букве — элемент с номером `n-1`. Поэтому, для того чтобы правильно определить нужный нам элемент файла, содержащий название буквы, в качестве входного параметра в процедуру вводим значение `n`, уменьшенное на единицу. Затем найденный процедурой `seek` элемент типизированного файла (название буквы с порядковым номером `n`) передается оператором `read` во вспомогательную переменную `b`. Затем сообщение о номере буквы и ее названии выводится на экран компьютера операторами `write`.

Если же пользователь выбирает второй вариант работы, то вначале он должен ввести название буквы, порядковый номер которой ему надо определить. Это название считывается во вспомогательную строковую переменную `c`. Далее в программе открывается цикл с постусловием `repeat`. В этом цикле с помощью процедуры `seek` последовательно перебираются все элементы типизированного файла, начиная с нулевого. Входными параметрами данной процедуры являются имя файловой переменной и значение вспомогательной переменной целого типа `l`. Перед началом работы цикла переменной `l` присваи-

ваем значение 0. В теле цикла каждый найденный процедурой элемент считывается во вспомогательную строковую переменную `b`, а значение переменной `l` увеличивается на единицу. Значение `b` сравнивается с "эталонным" значением, находящимся в переменной `c`.

Равенство значений этих двух переменных, свидетельствующее о том, что искомый элемент файла, соответствующий введенному названию буквы, найден, и является условием окончания работы цикла. Теперь осталось определить номер найденного элемента и увеличить его на единицу. Это и будет искомый порядковый номер буквы в алфавите. Данную операцию мы выполняем с помощью процедуры `filepos`, значение которой присваивается целочисленной переменной `x`. Введенное название буквы и соответствующий ей порядковый номер, находящийся в переменной `x`, будут выведены на экран компьютера с помощью оператора `writeln`.

В случае если пользователь в ходе выбора вариантов работы ошибочно ввел не 1 или 2, а какую-либо другую цифру или иной символ, то выполняется та ветвь оператора `case`, которая находится после служебного слова `else`. В результате выполнения этой ветви выводится сообщение об ошибке, а затем пользователь возвращается к выбору возможных вариантов работы посредством использования оператора безусловного перехода.

После завершения работы оператора `case...of` пользователю предлагается на выбор возможность продолжить работу с программой или завершить ее. Этот выбор производится путем присваивания соответствующего значения символьной переменной `s`. Проверка значения этой переменной производится в

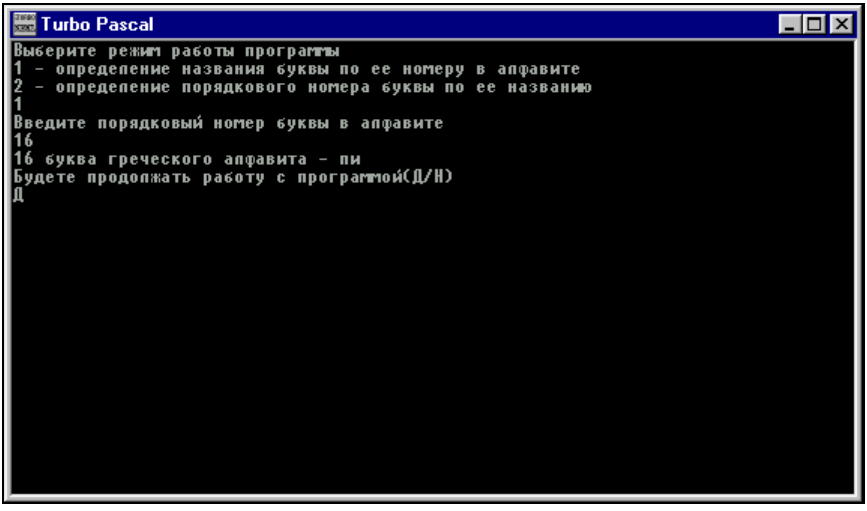


Рис. 9.4. Результаты работы электронного справочника, созданного с использованием типизированного файла

последней строке цикла с постусловием. Если значение переменной `s` равно "Н", то цикл завершает свою работу, а если значение иное, то происходит повторное выполнение тела цикла с возможностью выбора варианта работы программы.

По окончании работы с файлом закрываем его процедурой `close` (это необходимо сделать вне зависимости от того, какой вариант работы программы мы выбрали, чтобы сохранить файл для дальнейшей работы).

Используя созданный средствами языка Паскаль компьютерный справочник, нетрудно будет узнать, что десятой по счету буквой греческого алфавита является буква "каппа", а буква "пи" является шестнадцатой по счету буквой алфавита (рис. 9.4).

9.3. Нетипизированные файлы

Нетипизированными называются такие файлы, в которых в отличие от двух рассмотренных ранее видов, не указывается тип входящих в них элементов. Элементами одного и того же нетипизированного файла могут быть данные различных типов: целого, вещественного, символьного, строкового и других. Такие файлы используются для высокоскоростного обмена информацией между внешними файлами и программой, находящейся в оперативной памяти компьютера. При этом нетипизированные файлы, подобно типизированным, обеспечивают прямой доступ к данным.

Описание нетипизированной файловой переменной имеет следующий общий вид:

```
var имя_переменной:file;
```

где *имя_переменной* — имя связываемой с нетипизированным файлом файловой переменной.

Пример описания:

```
var unt:file;
```

связывание файловой переменной с файлом производится так же, как и для других файлов с помощью процедуры `assign`. Открытие файла для чтения и для записи производится соответственно процедурами `reset` и `rewrite`. Формат этих процедур также аналогичен формату для других видов файлов. Операторы же, используемые для непосредственной записи информации в файл и для чтения информации из файла, отличаются от обычных операторов ввода-вывода, которые мы использовали в предыдущих случаях.

Для непосредственной записи информации в нетипизированный файл используется оператор `blockwrite`.

Общий вид данного оператора следующий:

```
blockwrite(имя_переменной1, имя_переменной2, n);
```

где *имя_переменной1* — имя файловой переменной; *имя_переменной2* — имя вспомогательной переменной, из которой данные передаются в файл; `n` — количество элементов, в которые записывается информация за одно обращение к нетипизированному файлу. Значение `n` должно быть целого типа.

Пример использования данного оператора:

```
blockwrite(unt,y,1);
```

где `unt` — файловая переменная, `y` — вспомогательная переменная (может относиться к любому из обычных типов), `1` — данный оператор обрабатывает один элемент.

Чтение информации из файла производится оператором `blockread`. Общий вид данного оператора:

```
blockread(имя_переменной1, имя_переменной2, n);
```

где *имя_переменной1* — имя файловой переменной; *имя_переменной2* — имя вспомогательной переменной, в которую передаются данные из файла; `n` — количество элементов, из которых считывается информация за одно обращение к нетипизированному файлу. Значение `n` должно быть целого типа.

Нетипизированные файлы удобно использовать для долговременного хранения результатов работы программ. Эти результаты могут включать в себя как численные величины, так и тексты, комментирующие их содержание. Модифицируем уже знакомую нам по *разд. 8.1* программу решения квадратичного уравнения таким образом, чтобы как вводимые пользователем исходные данные, так и получающиеся при решении уравнения результаты, а также поясняющие их тексты передавались в нетипизированный файл, находящийся в другом каталоге, откуда в случае необходимости их можно было бы прочитать. Текст модифицированной программы приведен в листинге 9.5.

Листинг 9.5. Программа, записывающая результаты решения квадратного уравнения в нетипизированный файл

```
program qwurrez;
Uses Crt;
 var
   r:file; a,b,c,d,x1,x2:real; k,x:string;
begin
 Clrscr;
 assign(r,'d:\TP7\rez');
 writeln('Введите коэффициенты уравнения');
```

```
 writeln('После ввода очередного коэффициента нажимайте Enter');
 writeln('Введите a');
 readln(a);
 writeln('Введите b');
 readln(b);
 writeln('Введите c');
 readln(c);
 k:='Коэффициенты уравнения';
 d:=b*b-4*a*c;
 if d>=0 then
    begin
    x1:=(-b-sqrt(d))/(2*a);
    x2:=(-b+sqrt(d))/(2*a);
 x:='Корни уравнения';
    rewrite(r);
    blockwrite(r,k,1);
    blockwrite(r,a,1);
    blockwrite(r,b,1);
    blockwrite(r,c,1);
    blockwrite(r,x,1);
    blockwrite(r,x1,1);
    blockwrite(r,x2,1);
    writeln('Ввод данных в файл завершен');
    close(r)
    end
         else writeln('Уравнение не имеет решения');
 readln;
end.
```

В начале модифицированной программы, названной `qwurrez`, с помощью процедуры `assign` файловой переменной `r` "назначается" нетипизированный файл `rez`, находящийся на логическом диске D: в каталоге TP7. Затем по уже известному нам алгоритму решается квадратичное уравнение $ax^2 + bx + c = 0$. В том случае, если уравнение имеет решение, файл rez открывается для записи процедурой `rewrite`. Если файла с таким именем на диске не существовало, то он будет создан данной процедурой.

Далее в этот файл с помощью операторов `blockwrite` производится передача данных из текущей программы. Вначале в файловую переменную `r` передается текстовая строка, содержащаяся в переменной `k`. Эта строка поясняет, что далее в эту файловую переменную, т. е. в итоге в типизированный файл `rez`, будут передаваться исходные данные для решения задачи — коэффициенты уравнения. Эти коэффициенты передаются в файловую переменную из веще-

ственных переменных `a`, `b` и `c`. Затем в переменную `r` пересылается из строковой переменной `x` текстовая строка, в которой сообщается о том, что далее в файл будут записаны результаты решения задачи — найденные в программе корни уравнения. Эти корни находятся в вещественных переменных `x1` и `x2` и также передаются с помощью соответствующих операторов `blockwrite` в файловую переменную. По окончании передачи данных оператором `writeln` на экран компьютера выводится сообщение об этом. Завершается процесс записи в файл, как и всегда, оператором `close`. В случае же, если уравнение не имеет решения, процесс создания файла с последующей записью в него данных не происходит. Результаты работы программы приведены на рис. 9.5.

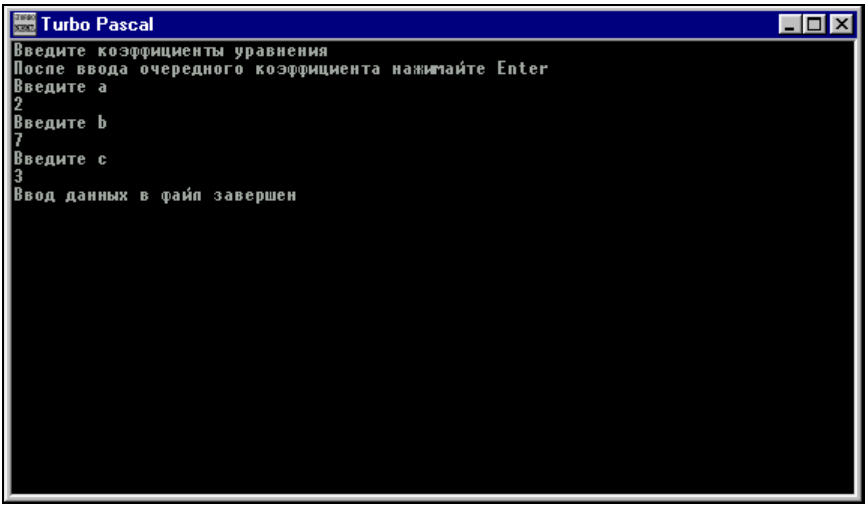


Рис. 9.5. Результаты работы программы записи итогов решения квадратного уравнения в нетипизированный файл

Созданный в результате работы нетипизированный файл обладает тем же недостатком, что и типизированные файлы: его нельзя прочитать обычным текстовым редактором. Поэтому, для того чтобы проконтролировать правильность создания и заполнения информацией текстового файла, необходимо создать еще одну программу, которая позволяла бы прочитать текст созданного предыдущей программой файла rez с выводом его содержимого на экран компьютера. Эту контролирующую функцию и выполняет программа `rezread`, текст которой приведен в листинге 9.6.

Листинг 9.6. Программа, считывающая данные из нетипизированного файла

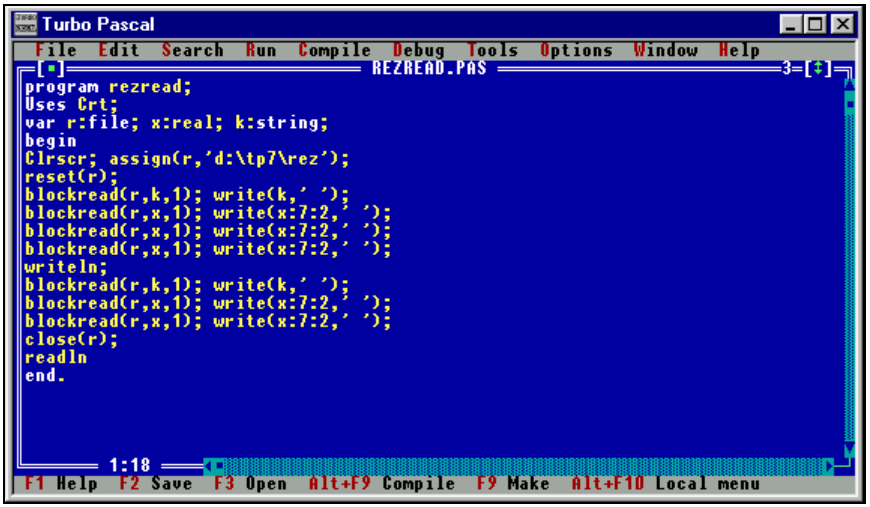
```
program rezread;
Uses Crt;
```

```
var r:file; x:real; k:string;
begin
Clrscr; assign(r,'d:\TP7\rez');
reset(r);
blockread(r,k,1); write(k,' ');
blockread(r,x,1); write(x:7:2,' ');
blockread(r,x,1); write(x:7:2,' ');
blockread(r,x,1); write(x:7:2,' ');
writeln;
blockread(r,k,1); write(k,' ');
blockread(r,x,1); write(x:7:2,' ');
blockread(r,x,1); write(x:7:2,' ');}
close(r);
readln
end.
```

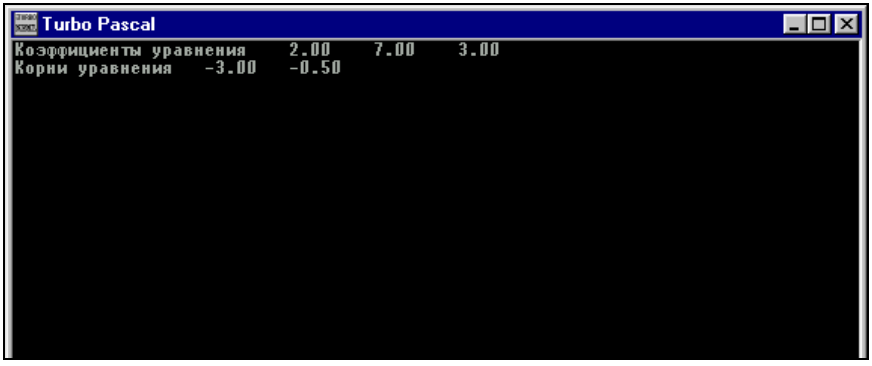
Уже на первый взгляд заметно, что в данной программе используется меньшее количество переменных, чем в предыдущей. Это обстоятельство объясняется тем, что если запись информации в программе `qwurrez` производилась из 7 различных переменных вещественного и строкового типа, то для чтения информации из созданного предыдущей программой файла нам понадобятся всего две вспомогательные переменные: одна строкового типа и одна — вещественного. Следует обратить внимание на то, что каждая из этих переменных будет использована в программе несколько раз. Если бы все данные, содержащиеся в файле, были одного типа, то можно было бы обойтись вообще одной вспомогательной переменной, но в данном случае в нетипизированном файле содержатся данные двух разных типов, что и требует наличия для каждого типа своей переменной. Как видно из раздела описаний программы, помимо вспомогательных переменных в программе используется еще только одна переменная — файловая переменная `r`.

В начале основной части программы эта файловая переменная связывается с типизированным файлом rez. Затем файл открывается для чтения процедурой `reset`. Сами операции чтения информации из файла с передачей ее в соответствующую по типу вспомогательную переменную производятся операторами `blockread`, а последующий вывод значения переменной на экран — операторами `write`. Вывод информации на экран производится в две строки. В первой строке в ее начале посредством строковой переменной `k` выводится поясняющий текст, а затем с помощью вещественной переменной `x` по одному выводятся три коэффициента уравнения, т. е. данная строка содержит начальные условия задачи. Далее с помощью пустого оператора `writeln` осуществляется перевод на следующую строку. Во вторую строку выводятся результаты решения задачи: сперва текстовый комментарий, а затем два по-

следних элемента файла rez — корни квадратичного уравнения. Для вывода информации в эту строку используются те же переменные, что и для первой. Закрыть файл нужно в обязательном порядке и после операций чтения.



а



б

Рис. 9.6. Программа чтения данных из нетипизированного файла *(а)* и результат ее работы *(б)*

Задания для самостоятельной работы

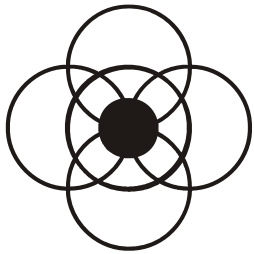
Задание 1. Составьте программу, которая создает на дискете текстовый файл, записывает в него построчно информацию (количество строк заранее неизвестно, а признаком окончания ввода является ввод в конце очередной строки символа *) и подсчитывает количество строк, содержащихся в этом файле.

Задание 2. Дополните предыдущую программу таким образом, чтобы она в конец текстового файла дописывала информацию об имеющемся в файле количестве строк (включая последнюю строку с данными о количестве строк).

Задание 3. Составьте программу, которая создает на дискете типизированный файл, содержащий `n` целочисленных элементов (количество элементов пользователь вводит с клавиатуры), записывает в файл `n` чисел, введенных пользователем, и подсчитывает сумму элементов, содержащихся в этом файле, а затем добавляет в файл еще один элемент — подсчитанную сумму. Для контроля правильности ввода данных предусмотреть в программе вывод всех данных из файла на экран компьютера.

Задание 4. Составьте программу, которая создает на дискете нетипизированный файл, содержащий следующую введенную пользователем информацию о прочитанной им книге: номера глав и названия этих глав, причем номера должны записываться в файл как целочисленные элементы, а названия — как строковые. Количество глав в прочитанной книге пользователь вводит с клавиатуры. Для контроля правильности вводимых данных следует вывести содержимое файла на экран компьютера, причем номер и название каждой главы должны выводиться в отдельной строке.

ГЛАВА 10



Пользовательские типы данных в Паскале

В ходе создания различных программ на языке Паскаль мы уже познакомились с самыми различными типами данных, которые используются для описания констант, переменных, значений функций. Эти типы, как читатель уже имел возможность убедиться, весьма многочисленны и разнообразны. В программах можно встретить числовые типы, как целые, так и вещественные, символьный и строковый типы, используемые для работы с текстовой информацией, логический тип данных, специальные типы, созданные для описания файловых переменных.

Однако перечисленными типами не исчерпываются возможности Паскаля. Свобода творчества программиста в данном отношении не ограничивается рамками уже существующих в языке типов, т. к. на их базе можно создавать новые, "авторские" типы данных, которые впоследствии можно с успехом применять для решения многих задач. О том, какими могут быть эти нестандартные типы данных, для чего они применяются, как их создавать и работать с ними рассказывается в данной главе.

10.1. Перечисляемый тип данных

Если программист хочет создать свой собственный тип данных, то один из способов создания заключается в том, чтобы перечислить все возможные значения, которые могут принимать переменные, относящиеся к данному типу. Такой тип данных и получил соответствующее название — *перечисляемый*.

Для того чтобы создать подобный тип данных, его следует описать в программе в разделе описаний перед описанием переменных, т. к. среди переменных, которые далее будут описаны, могут быть и переменные нового,

"авторского" типа. Описание нового перечисляемого типа, в общем виде, выглядит следующим образом:

```
type имя_типа =(значение1, значение2, ... значение n);
```

где `type` — служебное слово, в переводе с английского означающее "тип"; *имя_типа* — имя вновь создаваемого типа; *значение1, значение2, ... значение n* — перечень возможных значений для переменных нового типа (всего таких значений `n`).

Пример описания нового типа:

```
type week=(Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday);
```

Данный тип имеет название `week` (что по-английски означает "неделя"). Переменные этого типа могут принимать всего 7 значений. Эти значения являются английскими названиями дней недели. Описание переменной, относящейся к данному типу, будет выглядеть следующим образом:

```
var ned:week;
```

т. е. переменная с именем `ned` относится к типу `week`. Переменная `ned` (как и все переменные, относящиеся к перечисляемым типам) может принимать только значения, указанные в описании типа, к которому она относится. В нашем случае эта переменная может принимать значения `Monday`, `Tuesday` или `Wednesday`, но не может принимать, например, значение `Day` (по английски — "день").

К недостаткам переменных перечисляемого типа следует отнести то, что их нельзя использовать в операторах ввода-вывода. При этом, однако, их можно использовать в ряде других операторов (в частности, в условных операторах и операторах присваивания). Переменная перечисляемого типа может использоваться в качестве переменной-селектора в операторе `case`. Преимущества же использования переменных перечисляемого типа следующие: во-первых, использование таких переменных в ряде случаев делает программу более наглядной; во-вторых, используя переменные такого типа, легче защитить программу от неправильного ввода данных, что мы и продемонстрируем на следующем примере.

Пользователям персональных компьютеров хорошо известно, что большинство компьютеров в нашей стране собирается из импортных комплектующих. Из-за рубежа привозится большая часть эксплуатируемых у нас материнских плат, мониторов, принтеров и т. д. Естественно, что такие устройства зачастую не имеют даже инструкций, составленных на русском языке. Отечественным пользователям приходится разбираться в документации, написанной чаще всего по-английски.

Даже людям, хорошо владеющим английским языком, бывает непросто разобраться в технической терминологии, поэтому желательно под рукой иметь

словарик по вычислительной технике, в котором содержались бы наиболее употребительные термины из данной области. Такой словарь, созданный средствами языка Turbo Pascal, можно хранить в памяти вашего компьютера в виде программного файла. Достаточно ввести с клавиатуры интересующий вас термин на английском языке, чтобы получить его русский эквивалент. Программу, решающую данную задачу, назовем `vocab` (сокращение от английского *vocabulary* — "словарь"). Текст программы приведен в листинге 10.1.

Листинг 10.1. Программа, создающая англо-русский словарь по вычислительной технике

```
program vocab;
Uses Crt;
 type
    sl=(error,main,key,mouse,hard,soft,user,power,cpu,sock,freq);
 var
    wrd:sl; term:string;
begin
 ClrScr;
 writeln('Введите термин');
 readln(term);
 if term='mainboard' then wrd:=main;
 if term='keyboard' then wrd:=key;
 if term='mouse' then wrd:=mouse;
 if term='hardware' then wrd:=hard;
 if term='software' then wrd:=soft;
 if term='user' then wrd:=user;
 if term='power' then wrd:=power;
 if term='cpu' then wrd:=cpu;
 if term='socket' then wrd:=sock;
 if term='frequency' then wrd:=freq;
 case wrd of
   main: writeln('материнская плата');
   key: writeln('клавиатура');
   mouse: writeln('мышь');
   hard: writeln('аппаратура');
   soft: writeln('программное обеспечение');
   user: writeln('пользователь');
   power: writeln('питание');
   cpu: writeln('центральный процессор');
   sock: writeln('разъем');
   freq: writeln('частота');
```

```
   error: writeln('Такого термина в словаре нет')
 end;
 readln
end.
```

В данной программе перед разделом описания переменных описан новый тип данных `sl`. Переменные данного типа могут принимать значения, представляющие собой англоязычные термины из области вычислительной техники (в ряде случаев в сокращенном виде). В случае же, если переменной данного типа не присваивается в программе никакое значение, то она получает по умолчанию первое по порядку из ряда значений, перечисленных в описании типа. В нашем случае таким значением по умолчанию будет значение `error` (по-английски — "ошибка"). Почему именно это значение было сделано первым, станет ясно из дальнейшего описания программы. В разделе описания переменных имеется переменная `wrd`, относящаяся к созданному в программе типу `sl`. Кроме того, в том же разделе описывается вспомогательная переменная `term` строкового типа.

В основной части программы после соответствующего приглашения на ввод во вспомогательную переменную `term` заносится с клавиатуры какой-либо термин. Затем в программе следует ряд условных операторов, каждый из которых проверяет, соответствует ли введенное слово одному из терминов, содержащихся в нашем электронном словаре. Если такой термин находится, то переменной-селектору `wrd` присваивается соответствующее значение из перечисленного ряда. Это значение используется в операторе `case`. В зависимости от значения переменной-селектора одна из ветвей этого оператора выводит на экран компьютера русскоязычный термин, соответствующий англоязычному. Например, при вводе слова mainboard будет выдан его перевод — "материнская плата", для слова frequency — "частота" и т. д.

Если же пользователь сделал ошибку при вводе термина или ввел тот термин, который отсутствует в словаре, то переменная-селектор сохранит свое первоначальное значение — `error`. В этом случае оператор `case` осуществит выдачу сообщения:

```
Такого термина в словаре нет.
```

Конечно, можно было бы в качестве значений переменной-селектора указать и целые числа (1, 2, 3 и т. д.) и использовать их в условном операторе, но тогда программа лишилась бы в значительной степени своей наглядности.

На рис. 10.1 показана данная программа во время работы.

Данную программу несложно дополнить, добавив в нее переводы каких-либо новых терминов. Эту операцию можно провести в три этапа.

Рис. 10.1. Программа "Англо-русский словарь по вычислительной технике" во время ее работы

1. Вставить английский термин, перевод которого вы хотите добавить, в перечень значений типа `sl` (можно в сокращенном виде).
2. В основную часть программы добавить новый условный оператор `if`, в котором проверяется равенство вспомогательной переменной `term` данному термину (обязательно приведенному в полном виде), а переменной-селектору `wrd` в случае равенства присваивается значение, вставленное в список значений `sl`.
3. В оператор `case` вставить новую ветвь, в которой оператором `writeln` выводится русский перевод данного термина.

Теперь в вашем электронном словаре появится перевод еще одного слова.

10.2. Ограниченный тип данных

При решении ряда задач из области программирования бывает удобно на основе уже существующего типа данных создать новый, который может принимать только строго определенный ряд значений. Например, оценка, выставленная учащемуся за контрольную или экзамен, является целым числом, но переменная, которой будет присвоено значение-оценка, может принимать далеко не все значения из ряда целых чисел. Во-первых, оценка не может быть отрицательным числом или нулем. Во-вторых, она не может быть больше 5. Поэтому было бы удобно отнести переменную для оценки к особому типу данных, который может принимать только значения 1, 2, 3, 4, 5. Такой тип данных называется *ограниченным*.

Основой для создания ограниченного типа данных может быть не любой стандартный тип данных, а только порядковый. *Порядковыми* называются такие типы данных, в которых для каждого данного можно указать предыду-

щее или последующее (либо то и другое вместе). К порядковым типам данных относятся в частности типы integer, longint, char, boolean, а также перечисляемый тип данных. В качестве ряда значений для ограниченного типа данных и берется какой-либо отрезок одного из порядковых типов. Такой ряд должен иметь начальное и конечное значение. Порядковый тип, на основе которого создается перечисляемый, является для вновь создаваемого типа *базовым*.

Общий вид описания ограниченного типа:

```
type имя_типа = диапазон;
```

где type — служебное слово; *диапазон* — диапазон возможных значений, которые могут принимать переменные и константы этого типа.

Примеры описания ограниченных типов данных:

```
type mark=1..5;
```

Базовым здесь является тип integer. Значениями переменных типа mark могут быть только целые числа от 1 до 5.

```
type latin='a'..'z';
```

Для данного типа базовым является тип char. Значениями переменных такого типа могут быть только строчные буквы латинского алфавита.

```
type weekend=Friday..Sunday;
```

Для этого типа базовым является перечисляемый тип ned, описанный в предыдущем разделе. Значениями переменных типа weekend могут быть только переменные Friday, Saturday и Sunday.

Соответственно используемые в программе переменные данных типов будут описаны таким образом:

```
otmetka:mark;
bukva:latin;
day:weekend;
```

Над переменными ограниченного типа можно выполнять все те же арифметические и логические операции, что и над переменными соответствующего базового типа.

Ограниченные типы данных удобно использовать в программе для контроля вводимых данных. Например, в программе переменная bukva должна принимать только значения букв латинского алфавита. Пользователь же ошибочно присвоил данной переменной значение русской буквы "с" вместо латинской "c" (такая ошибка часто встречается, т. к. эти буквы расположены на одной

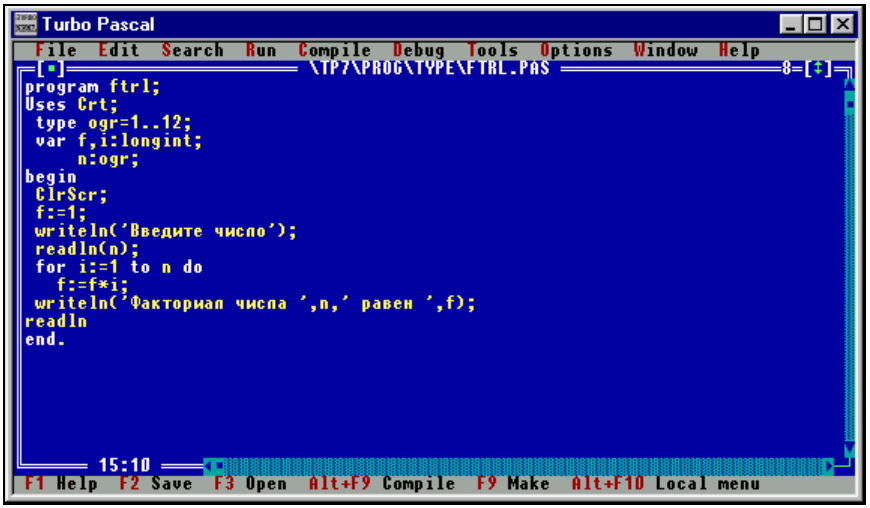
клавише). В этом случае при компиляции сразу будет выдано сообщение об ошибке.

Разберем использование данных ограниченного типа на следующем примере. Нам уже известна задача нахождения факториала целого числа. Факториал представляет собой геометрическую прогрессию, значение которой с увеличением исходного числа возрастает очень быстро. Но если мы возьмем программу вычисления факториала и введем в качестве исходного значения число 40, то получим результат, равный нулю, т. е. результат явно ошибочный. В чем заключается причина того, что правильно составленная программа выдает неправильный результат? Оказывается дело в том, что при вводе достаточно больших исходных значений происходит переполнение ячеек памяти, отведенных для результата, что и приводит к ошибке. Причем это будет происходить не только, если мы опишем переменную, содержащую результат, как переменную типа `integer`, но и для переменной типа `longint`.

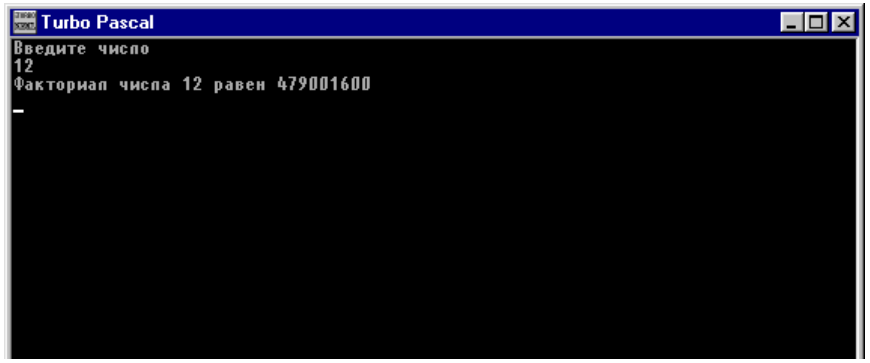
Для того чтобы избежать этой ошибки, следует сделать переменную, которой присваивается исходное значение, переменной ограниченного типа. Создадим в программе специальный ограниченный тип данных `ogr`, который может принимать только значения из диапазона от 1 до 12. К этому типу отнесем переменную `n`, в которую вводится исходное значение. Тогда для указанного диапазона программа будет выдавать правильные значения, а при попытке ввести слишком большое значение компилятор прямо выдаст сообщение об ошибке. В листинге 10.2 приведен текст программы, а на рис. 10.2 — результат ее работы.

Листинг 10.2. Программа вычисления факториала с ограничением значения вводимых данных

```
program ftrl;
Uses Crt;
 type ogr=1..12;
 var f,i:longint;
     n:ogr;
begin
 ClrScr;
 f:=1;
 writeln('Введите число');
 readln(n);
 for i:=1 to n do
   f:=f*i;
 writeln('Факториал числа ',n,' равен ',f);
readln
end.
```



а



б

Рис. 10.2. Усовершенствованная программа вычисления факториала *(а)* и результат ее работы *(б)*

Примечание

Возможна ситуация, когда вы использовали в программе ограниченный тип данных, а программа все-таки "проглатывает" значения, которые выходят за границы его диапазона. Например, в программе вычисления факториала вводим исходное значение 20, а программа производит расчеты с выдачей естественно неправильного результата. Тогда нужно настроить компилятор среды Turbo Pascal таким образом, чтобы он автоматически контролировал значения переменных ограниченного типа.

Для этого нужно войти в раздел меню **Options**, в нем щелкнуть пункт **Compiler**, в открывшемся диалоговом окне найти группу флажков **Runtime errors** и в этой группе установить флажок **Range checking**, а затем щелкнуть экранную кнопку **OK** для подтверждения сделанных настроек. В результате такой последовательности действий включится контроль значений переменных ограниченного

типа. Если все же режим работы компилятора не изменится, то следует выйти из системы программирования и запустить ее снова, после чего новые настройки компилятора вступят в силу.

10.3. Записи

В предыдущих главах мы уже познакомились с массивами, которые позволяют работать с большими объемами информации. Но информация, содержащаяся в массиве, должна состоять из однородных элементов. На практике же часто приходится сталкиваться с задачами, которые сводятся к обработке разнородной информации. В качестве примера возьмем работу отдела кадров какого-либо предприятия, в котором нужно обрабатывать анкеты сотрудников данного предприятия. Каждая из таких анкет содержит различные данные о сотруднике: фамилию, имя и отчество сотрудника, год и место рождения, пол, образование, общий стаж работы, стаж работы на данном предприятии, занимаемая должность и т. д.

Если перед программистом стоит задача автоматизировать процесс обработки этих анкет, то он столкнется с тем, что ему нужно будет создать некую структуру, состоящую из ряда элементов, каждый из которых в свою очередь содержит данные различных типов. Рассмотрим один такой элемент — данные о сотруднике предприятия Иванове Петре Сергеевиче, 1960 года рождения, уроженце Санкт-Петербурга, имеющем высшее образование, общий стаж работы которого составляет 20 лет, в том числе 10 лет на данном предприятии, который работает в настоящее время инженером. Для фамилии, имени, отчества, места рождения, образования понадобятся переменные строкового типа:

```
fam,im,otch,mesto,obr:string;
где в данном случае: fam:='Иванов'; im:='Петр'; otch:='Сергеевич';
mesto:='Санкт-Петербург'; obr:='высшее';.
```

Для пункта анкеты "пол" можно использовать переменную символьного типа, т. к. для указания пола хватит одной буквы:

```
pol:char;
```

в нашем случае `pol:='м';`.

Для работы с данными о годе рождения, общем стаже работы и стаже на данном предприятии нужны будут переменные целого типа:

```
god,stob,stpr:integer;
```

где в нашем случае `god:=1960; stob:=20; stpr:=10;`.

Если же в анкете для специалистов с высшим образованием имеется, например, дополнительный пункт о среднем балле в дипломе, то придется ввести

еще одну переменную вещественного типа (т. к. средний балл является, как правило, дробным числом)

```
ball:real;
```

для инженера Иванова, имеющего средний балл 4,78: `ball:=4.78;`.

Естественно, что аналогичный набор данных разного типа будет в анкете у каждого сотрудника, работающего на предприятии. Для описания такого набора будет нужен специальный тип данных. Такой тип существует в языке Паскаль и называется *записью*. Запись состоит из ряда полей. Каждое поле представляет собой элемент записи, имеющий свой конкретный тип. Описание поля в общем виде выглядит следующим образом:

```
type имя_зап = record
  имя_поля1: тип_поля1;
  имя_поля2: тип_поля2;
  ....
  имя_поля_n: тип_поля_n;
  end;
```

где `type`, `record` и `end` — служебные слова (record в переводе с английского и означает "запись"); *имя_зап* — общее имя записи; *имя_поля* — имя одного из полей, составляющих запись. Всего запись содержит `n` полей.

Возвращаясь к нашему примеру, можно сказать, что описание записи с анкетными данными по какому-либо сотруднику будет выглядеть следующим образом:

```
type anketa=record
  god,stob,str:integer;
  ball:real;
  pol:char;
  fam,im,otch,mesto,obr:string;
  end;
```

т. е. здесь приведена запись, имеющая имя `anketa`, содержащая 11 полей, относящихся к 4 различным типам. Из приведенного примера видно, что, как и в случае с описанием обычных переменных, однотипные поля можно перечислять через запятую, а после двоеточия указывать их общий тип.

Некоторые из полей, входящих в состав записи, могут в свою очередь сами быть записями, содержащими ряд полей. В частности, если в нашем примере в анкете требуется указать не просто год рождения, а точную дату рождения, включающую день, месяц и год, то эту информацию можно представить в виде записи `date`, содержащей поля `den`, `mes` и `gr`. В данном случае для описания даты и года употребим целый тип, а для месяца — строковый.

Таким образом, описание записи `date` будет выглядеть следующим образом:

```
date:record
  den, gr:integer;
  mes:string;
  end;
```

причем вся эта структура является составной частью более крупной структуры `anketa`. Естественно, что в этом случае поле `god`, содержавшее в записи `anketa` год рождения, становится ненужным. Теперь измененная запись будет выглядеть следующим образом:

```
type anketa=record
  stob,stpr:integer;
  ball:real;
  pol:char;
  fam,im,otch,mesto,obr:string;
  date:record
    den, gr:integer;
    mes:string;
    end;
  end;
```

В данном варианте записи присутствуют два служебных слова `end`, причем первый из них закрывает запись `date`, а второй всю запись `anketa` в целом. В свою очередь запись `anketa` может быть элементом массива либо типизированного файла. Например, массив `otdel`, содержащий элементы, оформленные в виде записей анкеты сотрудников одного из отделов организации, будет описываться в программе в разделе переменных следующим образом:

```
var otdel:array[1..n] of anketa;
```

Данный массив содержит информацию о `n` сотрудниках отдела. Каждая запись будет содержать в общей сложности 10 полей, причем 9 полей являются простыми, а одно также является записью, содержащей 3 поля. Таким образом, с помощью типа "запись" можно создавать весьма сложные информационные структуры.

Информацию же обо всех сотрудниках организации можно поместить в случае необходимости в специальный типизированный файл kadry. Описание файловой переменной `kad`, связанной в программе с файлом kadry, будет выглядеть так:

```
var kad: file of anketa;
```

К такому типизированному файлу применимы все те же операции, что и к типизированному файлу, содержащему простые элементы. Естественно, что

в разделе описаний описание самой записи должно предшествовать описанию того массива или файловой переменной, которые включают в себя данную запись.

В программах на языке Паскаль можно работать не только со всей записью в целом, но и производить определенные действия над отдельными полями (присваивать им значения, выводить эти значения на экран компьютера, сравнивать однотипные поля и т. д.). Обращение к полю в общем виде производится следующим образом:

имя_зап.имя_поля

Например, если инженер П. С. Иванов отработал в данной организации еще год после занесения его данных в компьютер, и нужно прибавить этот год к его стажу, то это можно сделать следующими операторами:

```
anketa.stob:=21;
```
(для общего стажа)

```
anketa.stob:=11;
```
(для стажа на данном предприятии)

Если же требуется в запись поместить данные о точной дате рождения сотрудника (например, 22 августа 1960 года), то соответствующие операторы присваивания будут выглядеть так:

```
anketa.date.den:=22;
```
(для дня рождения)

```
anketa.date.mes:='август';
```
(для месяца рождения)

```
anketa.date.gr:=1960;
```
(для года рождения)

Рассмотрим способы работы с записями на примере практической задачи (рис. 10.3). Создадим электронный телефонный справочник, в который пользователь сможет заносить следующие данные о своих родственниках, друзьях, знакомых, коллегах по работе: их фамилии, имена, номер телефона, домашний адрес. Далее по введенной фамилии пользователь сможет получать из справочника необходимую ему информацию: телефон и адрес. Первая часть этой задачи решается с помощью программы, текст которой приведен в листинге 10.3.

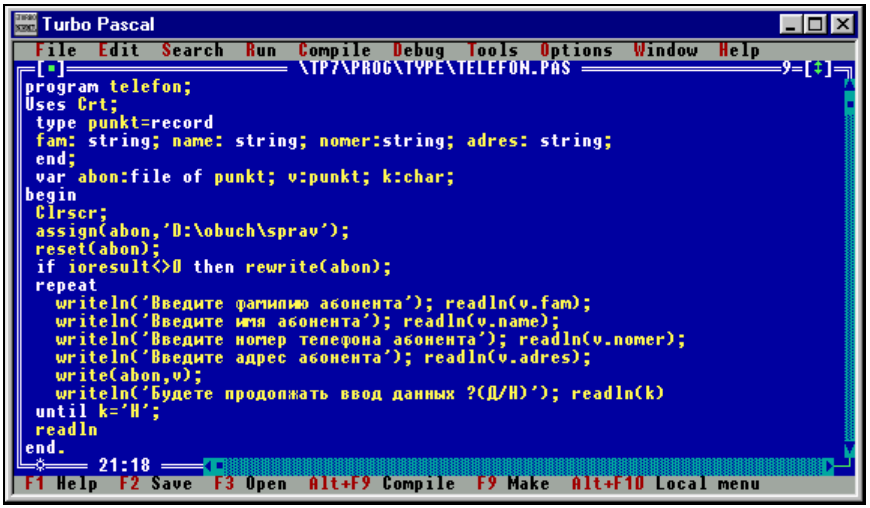
Листинг 10.3. Программа, заносящая данные в телефонный справочник

```
program telefon;
Uses Crt;
 type punkt=record
 fam: string; name: string; nomer:string; adres: string;
 end;
 var abon:file of punkt; v:punkt; k:char;
begin
 Clrscr;
```

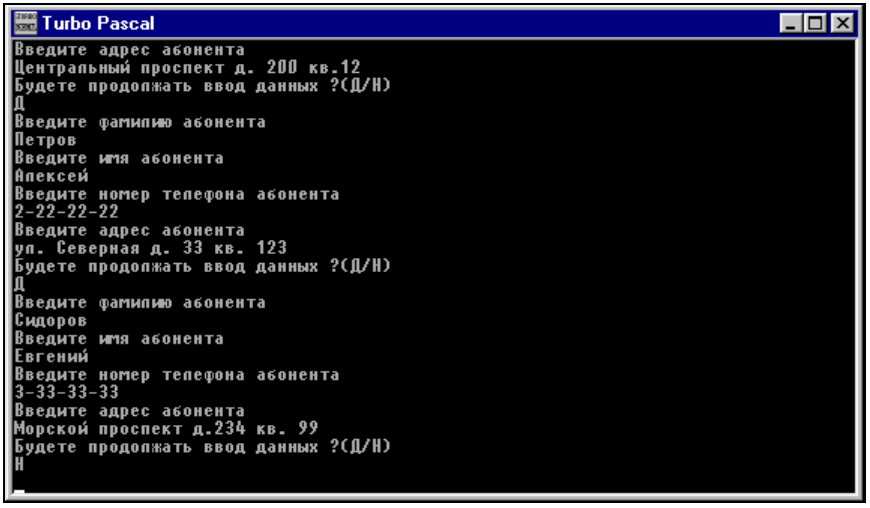
```
 assign(abon,'D:\obuch\sprav');
 reset(abon);
 if ioresult<>0 then rewrite(abon);
 repeat
   writeln('Введите фамилию абонента');
   readln(v.fam);
   writeln('Введите имя абонента');
   readln(v.name);
   writeln('Введите номер телефона абонента');
   readln(v.nomer);
   writeln('Введите адрес абонента');
   readln(v.adres);
   write(abon,v);
   writeln('Будете продолжать ввод данных? (Д/Н)');
   readln(k)
 until k='H';
 readln
end.
```

В разделе описаний программы вначале описывается вновь создаваемый тип `punkt`, представляющий собой запись, содержащую 4 поля: для фамилии, имени, телефона и адреса, заносимых в справочник абонента. Каждое из перечисленных полей относится к строковому типу. Затем в разделе переменных описывается файловая переменная `abon`. В связанном с ней типизированном файле sprav будут содержаться элементы типа `punkt`. Вспомогательная переменная `v` также относится к типу `punkt`. Еще одна вспомогательная переменная `k` является переменной строкового типа.

Далее в основной части программы после процедуры связывания файл sprav открывается процедурой `reset` для операций записи и чтения. В случае же, если такого файла на диске не существует, он создается заново. Для проверки факта существования файла используется встроенная функция языка Паскаль `ioresult`. Данная функция используется для контроля за осуществлением операций ввода-вывода при работе с файлами (в том числе открытия и закрытия файла). Значением этой функции является целое число. В случае если операция ввода-вывода была проведена успешна, функция `ioresult` принимает нулевое значение. В нашей программе после операции открытия файла и производится проверка на равенство или неравенство значения этой функции нулю. Если значение `ioresult` равно нулю, то операция открытия файла прошла успешно, и, следовательно, такой файл уже существует. Если же значение функции не равно нулю, то, вероятно, такого файла на диске нет и его нужно создать командой `reset`.



а



б

Рис. 10.3. Программа, создающая телефонный справочник и заполняющая его данными *(а)*, и результат ее работы *(б)*

После осуществления проверки можно переходить непосредственно к операциям ввода данных в файл. Процесс ввода представляет собой тело цикла `repeat...until`. В теле цикла после соответствующей подсказки, выводимой оператором `writeln`, пользователь вводит фамилию, имя, телефон и адрес добавляемого в справочник абонента. Информация по каждому из указанных ранее пунктов оператором `readln` передается в соответствующее поле вспомогательной переменной `v`. Затем, после того как всем полям переменной-записи `v` присвоены значения, вся эта запись целиком передается в типизиро-

ванный файл sprav посредством записи в связанную с ним файловую переменную abon. Потом программа запрашивает пользователя, будет ли он продолжать ввод данных. Полученный ответ, которым должна быть буква "Д" или "Н", присваивается символьной переменной k. В завершающей строке цикла производится проверка условия его завершения. Этим условием является равенство значения переменной k букве "Н". Таким образом можно за один сеанс работы с программой ввести данные на нескольких человек.

Программа telsprav производит считывание информации из типизированного файла sprav, который был создан и заполнен информацией с помощью предыдущей программы. В разделе описаний программы описывается уже знакомый нам тип punkt, а также файловая переменная abon типа punkt, переменная целого типа i и переменная строкового типа fff. Текст программы приведен в листинге 10.4.

Листинг 10.4. Программа поиска информации в телефонном справочнике

```
program telsprav;
Uses Crt;
 type punkt=record
   fam: string; name: string; nomer:string; adres:string;
 end;
 var
   abon:file of punkt; v:punkt; i:integer; fff:string;
begin
 Clrscr;
 assign(abon,'D:\obuch\sprav');
 reset(abon);
 i:=0;
 writeln('Укажите фамилию абонента');
 readln(fff);
 repeat
   seek(abon,i);
   read(abon,v);
   i:=i+1;
   if v.fam=fff then
     begin
     writeln('абонент ', v.fam, ' ', v.name);
     writeln('проживает по адресу ',v.adres);
     writeln('телефон абонента ', v.nomer)
     end;
 until eof(abon);
 readln
end.
```

Основная часть программы начинается со связывания файловой переменной с внешним файлом sprav и открытия этого файла для чтения. Далее, после вывода на экран соответствующей подсказки, пользователь должен ввести фамилию абонента, по которому нужна справка. Эта фамилия считывается в переменную `fff`. Затем по введенной фамилии производится поиск информации, представляющей собой запись — элемент типизированного файла.

Как и в предыдущей программе, для организации процесса используется цикл `repeat...until`, но содержание тела цикла здесь иное. Сам поиск осуществляется с помощью уже знакомой нам процедуры `seek`. Поиск ведется, начиная с нулевого элемента (поэтому параметру процедуры `i`, соответствующему номеру элемента, еще до начала работы цикла было присвоено нулевое значение). Значение каждого `i`-го элемента типизированного файла по очереди присваивается вспомогательной переменной-записи `v`. Затем в условном операторе `if` производится сравнение того из полей переменной `v`, которое соответствует фамилии (`v.fam`) с искомым значением, содержащимся в переменной `fff`. Если эти величины совпадают, то на экран компьютера операторами `writeln` выводится содержимое всех полей, имеющихся в данной записи: фамилия, имя, телефон и адрес абонента, т. е. интересующая пользователя информация. Работа программы, однако, на этом не заканчивается, т. к. чтение и сравнение информации, содержащейся в элементах файла, будет продолжаться до тех пор, пока в ходе этой процедуры не будет обнаружен конец файла (`eof`). После обнаружения конца файла завершается работа цикла, а затем и работа программы. Таким образом, если в вашем справочнике имеется несколько однофамильцев, то по всем этим людям будут найдены и выведены их данные. Пример работы программы показан на рис. 10.4.

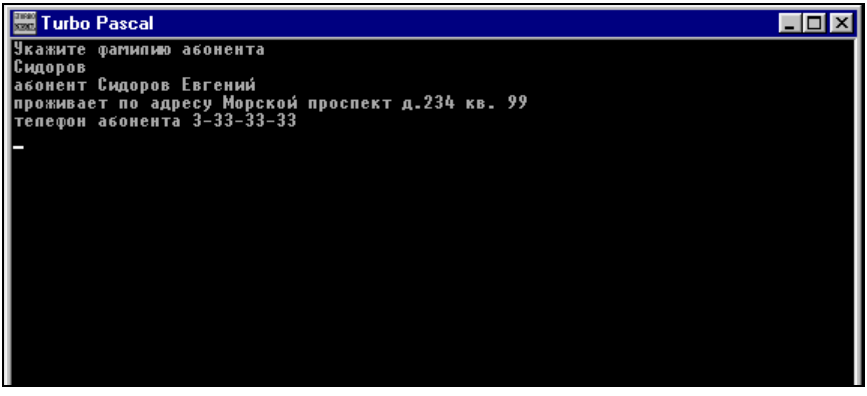


Рис. 10.4. Результат работы программы "Телефонный справочник"

Возвращаясь к тому примеру, с которого мы начали этот раздел, можно сказать, что аналогичным образом можно составить и программу, которая бы

создавала файл с данными по каждому из сотрудников организации и по введенному с клавиатуры запросу выдавала бы о нем всю необходимую информацию. Но в связи с тем, что в организации может быть достаточно большое количество сотрудников, среди которых может встретиться немало однофамильцев, поиск информации лучше производить по какому-либо другому параметру, который для каждого работающего в данной организации человека является уникальным. Таким параметром может служить, например, табельный номер сотрудника. В таком случае в запись с данными о сотруднике следует добавить еще одно поле целого типа, в котором и будет содержаться табельный номер.

Описание этого поля будет выглядеть так:

```
tabel:integer;
```

а присваивание ему значения так:

```
anketa.tabel:=4567;
```

Поле `tabel` и можно будет использовать в качестве ключевого, т. е. такого, по которому однозначно идентифицируется содержащая его запись.

10.4. Множества

При работе над различными задачами из области программирования часто возникает необходимость использования структур, которые представляют собой некий набор, состоящий из однородных элементов. При этом элементы данного набора не должны быть жестко закреплены в этой структуре в определенном порядке как в массиве или в типизированном файле, а могут компоноваться различным образом. Примером подобной структуры, с которой человек часто сталкивается в повседневной жизни, является набор букв того или иного языка. Для удобства изучения и использования в определенных целях (например, при создании справочников, энциклопедий, разного рода списков, ведомостей) буквы упорядочивают в виде алфавита, в котором за каждой буквой раз и навсегда закреплено строго определенное место. Алфавит, таким образом, представляет собой аналог массива. Но чаще буквы компонуют различным образом так, что из 33 букв кириллицы или 26 букв латинского алфавита, сочетая их по определенным правилам, можно создать деловое письмо и научный труд, техническую инструкцию и газетную статью, прозаический роман и поэму в стихах, и многое другое.

В программировании такая структура, состоящая из элементов одного типа, которые можно по-разному располагать, называется *множеством*. Общий вид описания переменной множественного типа в языке Паскаль следующий:

```
var имя_множества: set of тип;
```

где `var` и `set` — служебные слова. Слово set в переводе с английского и означает "набор, множество"; *имя_множества* — имя создаваемого типа-множества, которое может дать по своему усмотрению программист; *тип* — один из уже существующих типов, к которому относятся элементы вновь создаваемого множества.

В качестве базового типа может выступать любой из порядковых типов *(см. разд. 10.2)*. Если набор элементов множества представляет собой какой-либо отрезок значений одного из порядковых типов, то можно указать только начальное и конечное значение, а между ними поставить символ "..". Если же элементы, составляющие множество, не идут подряд в базовом типе, то эти элементы нужно перечислить в скобках через запятую.

Приведем примеры описания множеств.

```
var triznak: set of 100..999;
```

Это множество натуральных трехзначных чисел. Базовый тип для данного множества — `integer`.

```
var prop: set of 'А'..'Я';
```

Это множество прописных букв русского алфавита. Базовый тип — `char`. Описание множества строчных букв русского алфавита будет выглядеть сложнее, т. к. строчные буквы от "а" до "п" располагаются в одном месте расширенной кодовой таблицы, а буквы от "р" до "я" — в другом.

```
var compon: set of (keyboard, motherboard, mouse, display)
```

Это множество устройств, входящих в состав персонального компьютера.

При описании множества необходимо учитывать два немаловажных момента. Во-первых, в описании элементы множества не должны повторяться. Во-вторых, если множество создается на базе числового типа данных, то значения элементов множества не должны быть с одной стороны отрицательными, а с другой — не превышать число 255.

Наряду с описанными ранее переменными типа "множество" в программах могут встречаться и постоянные множества. Элементы такого множества перечисляются в квадратных скобках: `[1,2,3,4,5,6,7,8,9]` — так будет выглядеть описание постоянного множества, состоящего из однозначных целых чисел. Множество может содержать несколько элементов, один элемент — `[1]` или вообще ни одного — `[]`. В последнем случае множество называется *пустым*.

Над множествами, так же как и над обычными переменными, можно выполнять ряд операций. При этом нельзя забывать о том, что хотя некоторые из этих операций обозначаются так же, как арифметические, это особые операции, отличающиеся от обычных.

Операция, обозначающаяся знаком * (звездочка), хотя и описывается тем же значком, что и операция умножения, называется и осуществляется по-другому — *пересечение* двух множеств. Результатом выполнения этой операции над двумя множествами является нахождение их общих элементов, принадлежащих как одному, так и другому множеству. Например, если мы присвоим переменной типа "множество" A значение [20,30,40], что записывается в программе следующим образом:

```
A:=[20,30,40];
```

а переменной того же типа B присвоим значение [30,40,50], что записывается как

```
B:=[30,40,50];
```

то в результате операции C:=A*B; множество C получит значение [30,40].

Если при операции над двумя множествами используется знак +, то такая операция называется *объединением* двух множеств. Ее результатом будет множество, в которое входят все элементы, содержащиеся хотя бы в одном из исходных множеств. В качестве примера возьмем два множества: lat, значением которого будет [q,w,e,r,t,y], и rus, значением которого будет [й,ц,у,к,е,н,г] (в качестве элементов данных множеств взяты буквы, составляющие название традиционной английской и русской раскладок клавиатуры компьютера). В результате операции latrus:=lat+rus; множество latrus получит следующее значение: [q,w,e,r,t,y,й,ц,у,к,е,н,г].

Операция, обозначаемая знаком -, называется *разностью* двух множеств. В результате выполнения данной операции над двумя множествами мы получаем новое множество, содержащее те элементы первого, которые не входят во второе. Рассмотрим эту операцию на примере двух множеств: language и low. Первому из этих множеств присвоим значение, представляющее собой перечень различных языков программирования, а второе представляет собой список только тех языков, которые не являются языками высокого уровня. К последним относится язык ассемблера и машинные коды. Операции присваивания будут выглядеть следующим образом:

```
language:=[basic,pascal,c,fortran,algol,assembler_language,machine_code];
low:=[assembler_language,machine_code];
```

Далее выполним операцию нахождения разности и результат присвоим множеству high:

```
high:=language-low;
```

В результате данной операции множество high получит следующее значение: [basic,pascal,c,fortran,algol], т. е. данное множество будет содержать перечень языков высокого уровня.

Наряду с описанными ранее операциями над множествами можно также производить *сравнение* множеств друг с другом. В сравнении также участвуют два множества. Для сравнения двух множеств используются операции отношения, которые обозначаются теми же знаками, что и операции сравнения обычных переменных, но имеют иной смысл. Результатом операции отношения является логическая величина, которая может принимать значения `true` или `false`. Рассмотрим более подробно данные операции.

= — равенство двух множеств (все элементы первого множества совпадают с элементами второго).

< > — неравенство двух множеств (одно из множеств содержит хотя бы один элемент, отсутствующий в другом).

Пример использования данных операций: даны два множества `G` и `H`. Значение множества `G` — `[4,5,6]`. `H` также имеет значение `[4,5,6]`. Для данных множеств результат операции сравнения `G = H` будет равен `true`, а операции `G < > H` будет равен `false`. В случае же, если мы множеству `H` присвоим другое значение — `[6,7,8]`, а `G` оставим без изменения, то результат операции `G = H` будет равен `false`, а операции `G < > H` будет равен `true`.

< — проверка на строгое вхождение первого множества во второе (т. е. все элементы первого множества должны одновременно входить во второе. Кроме того, второе множество должно содержать какие-либо дополнительные элементы).

<= — проверка на нестрогое вхождение первого множества во второе (все элементы первого множества должны одновременно входить во второе. Помимо элементов первого множества второе множество может как содержать, так и не содержать какие-либо дополнительные элементы.).

> — проверка на строгое вхождение второго множества в первое.

>= — проверка на нестрогое вхождение второго множества в первое.

Пример использования операций вхождения: даны два множества `K` и `L`. Значение множества `K` равно `[10,15,20]`, множества `L` — `[10,15,20,25,30]`. Результаты операций `K<L` и `K<=L` будут равны `true`. Результаты операций `K>L` и `K>=L` будут равны `false`. Присвоим множеству `L` новое значение `[10,15,20]`, значение `K` оставим без изменения. После выполнения операции присваивания результаты операций `K<=L` и `K>=L` будут равны `true`, а результаты операций `K>L` и `K<L` будут равны `false`.

При работе с множествами часто используется еще одна операция, в которой участвуют не два множества, а множество и какой-либо элемент. Такая опе-

рация называется операцией определения *принадлежности* элемента множеству. Общий вид данной операции следующий:

```
элемент in [значение_множества];
```

где *элемент* — элемент, относящийся к базовому для данного множества типу; `in` — служебное слово, обозначающее в переводе с английского "в"; *значение_множества* — список всех элементов, входящих в множество.

Операция имеет значение `true`, если элемент входит в данное множество (т. е. совпадает с одним из элементов множества), и `false`, если он в данное множество не входит.

Например, выражения `D in [A..E]` и `5 in [3,5,7,9]` имеют значение `true`, а выражения `D in [E..N]` и `5 in [7,9,11]` — значение `false`.

При осуществлении операций над множествами, так же как и при выполнении арифметических и логических операций над обычными переменными, соблюдается определенный приоритет.

Наибольшим приоритетом обладает операция `*`. Следующими по приоритету являются операции + и - . Наименьший приоритет имеют операции `in`, `=`, `< >`, `<`, `<=`, `>`, `>=`.

При необходимости изменить порядок выполнения операций над множествами, как и для обычных переменных, используют круглые скобки. Значение выражения, заключенного в скобки вычисляется в первую очередь.

Принадлежность каких-либо элементов к определенному множеству широко используется для различных статистических подсчетов. Рассмотрим работу с множествами на примере программы, которая для любого введенного с клавиатуры текста, написанного на русском языке, определяет количество имеющихся в нем гласных и согласных букв. Подобные программы могут, в частности, использоваться для исследований в области филологии. Текст программы приведен в листинге 10.5.

Листинг 10.5. Программа, подсчитывающая количество гласных и согласных букв в тексте

```
program filolog;
Uses Crt;
 var
   gl,sog:integer; a:char;
begin
 ClrScr;
 gl:=0;
 sog:=0;
```

```
 writeln('Вводите текст построчно. ');
 write('Ввод каждой строки заканчивайте нажатием клавиши Enter');
 writeln('Для завершения ввода всего текста введите 0 и Enter');
 repeat
   repeat
   read(a);
   if a in ['а','е','и','о','у','ы','э','ю','я',
   'А','Е','И','О','У','Ы','Э','Ю','Я'] then gl:=gl+1;
   if a in ['б'..'д','ж','з','й'..'н','п','р'..'т','ф'..'щ',
   'Б'..'Д','Ж','З','Й'..'Н','П'..'Т','Ф'..'Щ'] then sog:=sog+1
   until eoln;
 until a='0';
 readln;
 writeln('Число гласных букв в тексте равно ',gl);
 writeln('Число согласных букв в тексте равно ',sog);
 readln
end.
```

В программе пользователь должен вводить текст построчно. Считывается же текст оператором `read` посимвольно. Процесс считывания символов, содержащихся в строке, производится во внутреннем цикле `repeat...until`. Каждый считываемый символ проверяется условным оператором `if` на принадлежность к множеству гласных букв. Если вводимый символ принадлежит данному множеству, то значение переменной-счетчика `gl` увеличивается на единицу. (В начале программы переменной-счетчику было присвоено нулевое значение.) Второй условный оператор аналогичным образом проверяет принадлежность символа к множеству согласных букв. Переменной-счетчиком здесь является `sog`. Условием окончания внутреннего цикла является обнаружение признака конца строки. Нахождение конца строки осуществляется с помощью соответствующей стандартной функции `eoln`. Эта функция по умолчанию имеет значение `false`, а при обнаружении ею двух управляющих символов возврата каретки CR и перевода строки LF *(см. разд. 9.1)* получает значение `true`, что приводит к завершению работы внутреннего цикла.

Внешний цикл `repeat...until` повторяет процесс построчного ввода до тех пор, пока в ходе его работы не будет обнаружен признак конца всего текста. Таким признаком, завершающим работу цикла, является ввод нуля. По окончании работы цикла полученные статистические данные выводятся на экран компьютера операторами `writeln`. На рис. 10.5 показаны результаты работы программы `filolog`.

Множества используются при решении математических задач, особенно тех из них, в которых для получения искомого результата требуется фильтрация

данных, т. е. отсеивание ненужных данных для успешного решения задачи. В качестве примера рассмотрим задачу нахождения всех простых чисел в диапазоне от 1 до некоторого n. Как известно, простым числом называется такое число, которое делится только на единицу и само на себя. Способ решения данной задачи был разработан еще в III веке до нашей эры выдающимся древнегреческим ученым Эратосфеном и в честь него получил название "Решето Эратосфена". В принципе, составить программу решения данной задачи можно и не прибегая к множественному типу (сходная задача имеется в заданиях для самостоятельной работы к *главе 8*), но его использование делает создаваемую программу более простой и наглядной.

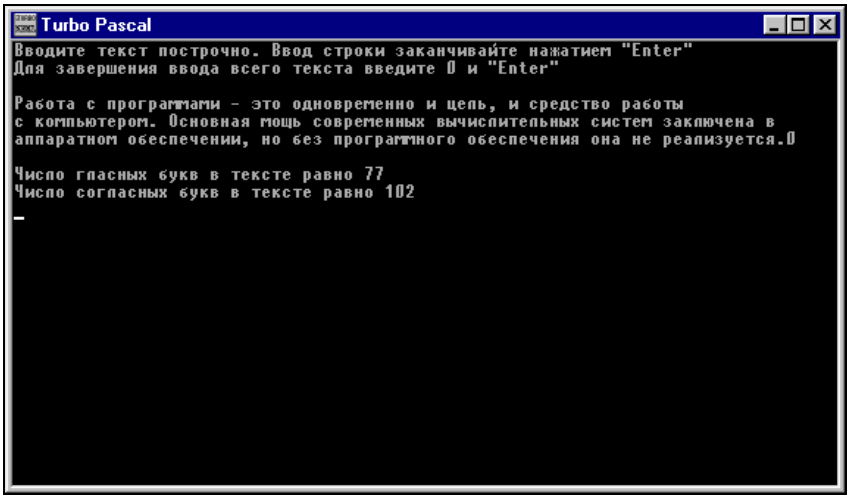


Рис. 10.5. Результаты работы программы, подсчитывающей количество гласных и согласных букв в тексте

Алгоритм решения этой задачи сводится к следующему. В начале в последовательности натуральных чисел от 1 до n вычеркивается (или заменяется на нуль) каждое второе число, начиная с числа 2 (при этом само число 2 в последовательности остается, т. к. оно является простым). Затем из оставшихся в последовательности чисел берется первое по порядку после числа 2 ненулевое число. Это будет число 3. Далее в последовательности вычеркивается или заменяется на нуль каждое третье число, начиная с 3, кроме самого числа 3. Затем берется следующее по порядку ненулевое число (число 5) и производится аналогичная операция замены на нуль кратных ему чисел. Эти действия выполняются для всех чисел вплоть до числа, равного n/2 (если число n нечетное, то берется целая часть числа n/2). В результате в интервале натуральных чисел от 1 до n останутся невычеркнутыми (ненулевыми) только простые числа. В листинге 10.6 приводится текст программы, реализующей алгоритм "Решето Эратосфена".

Листинг 10.6. Программа поиска простых чисел ("Решето Эратосфена")

```
program eratos;
 Uses Crt;
 label 1;
 var
    dpn:set of 1..255; i,j,k,l,n:integer;
begin
 ClrScr;
 writeln('Программа вычисления простых чисел в интервале от 1 до n');
 1: writeln ('Введите число n (натуральное число не больше 255)');
 readln(n);
 if n>255 then goto 1;
 k:= n div 2;
 dpn:=[2..n];
 for i:=2 to k do
   for j:=2 to n do
     if (j mod i =0) and (j<>i) then dpn:=dpn-[j];
 writeln;
 writeln('Печатаем простые числа в интервале от 1 до ',n);
 writeln;
 write(1:4);
 l:=2;
 for j:=2 to n do
   if j in dpn then
     begin
     write(j:4);
     if l mod 10 =0 then writeln;
     l:=l+1  end;
 readln
end.
```

В начале работы программы пользователь после соответствующего приглашения должен ввести число `n` — верхний предел диапазона, в котором будет производиться поиск простых чисел. Эта верхняя граница не может быть больше 255, т. к. в программе используется целочисленное множество. Если все же при вводе была совершена ошибка (было введено число большее, чем 255), то с помощью оператора безусловного перехода пользователь отсылается к той строке программы, где указана данная граница.

Как видно из приведенного алгоритма, одна и та же последовательность действий повторяется для всех чисел от 2 до целой части n/2. Последняя величина в программе обозначена через `k` и вычислена перед началом работы основ-

ного алгоритма. Кроме того, множеству целых чисел `dpn` присваивается новое значение, равное диапазону, в котором далее будет производиться поиск простых чисел. В процессе дальнейшей работы программы из этого множества будут удалены все непростые числа, в итоге `dpn` превратится в множество простых чисел из данного диапазона.

Процесс непосредственного определения множества простых чисел производится с помощью двух вложенных друг в друга циклов. Переменной внешнего цикла является `i`, которая меняет свои значения от 1 до `k`, т. е. эта величина принимает все значения, с помощью каждого из которых производится процедура вычеркивания "лишних" чисел, не относящихся к множеству простых. Сам же процесс удаления из множества этих чисел осуществляется во внутреннем цикле, переменная которого `j` принимает все значения из заданного в начале программы диапазона. Во внутреннем цикле производится проверка текущего значения `j` с помощью условного оператора. Если текущее значение `j` делится без остатка на текущее значение `i` и при этом не равно делителю, то, следовательно, оно не является простым, и его смело можно удалять из множества простых чисел, что и делает соответствующий оператор. Данный оператор присваивает множеству `dpn` новое значение, которое равно предыдущему за вычетом удаляемого элемента.

В итоге работы двух циклов искомое множество сформировано, и остается только вывести его элементы на экран компьютера. Данный вывод производится в цикле с переменной `j`, значение которой изменяется от 1 до `n`. Но здесь можно столкнуться со следующей проблемой. Вывод элементов осуществляется в строку оператором `write` (если вывод производить оператором `writeln`, то каждый элемент будет выведен в отдельной строке, общее число которых в текстовом режиме составляет 25, и для большого диапазона все значения просто не поместятся на экране компьютера). Но т. к. мы заранее не знаем общее количество элементов и значение каждого из них, то может оказаться, что какое-либо число (элемент множества) будет при выводе "разрезано" — одна или две его цифры окажутся на одной строке, а остальные — на другой, что, естественно, затруднит восприятие пользователем результатов работы программы.

Выход из данной ситуации заключается в следующем: каждый из элементов найденного множества будет выводиться в отформатированном виде — под каждое из чисел будет отведено по 4 позиции, чего вполне достаточно, т. к. даже самые большие из выводимых чисел могут быть трехзначными, и, кроме того, одна позиция резервируется под пробел между числами. Числа в программе выводятся по 10 в одну строку, и в каждой строке будет занято, таким образом, по 40 позиций. Всего же в каждой строке содержится 80 позиций для записи различных символов. Разбивка элементов по 10 про-

изводится с помощью следующего приема: в теле цикла, осуществляющего вывод элементов множества, используется вспомогательная переменная-счетчик `l`. Значение данной переменной увеличивается на единицу при выводе каждого очередного элемента множества. Следовательно, как только выводятся очередные 10 элементов множества, значение переменной `l` становится кратным 10. Эта кратность проверяется с помощью операции нахождения остатка при делении на 10. Если значение `l` действительно оказывается кратным 10, то курсор переводится на следующую строчку пустым оператором `writeln`. Перед началом же работы данного цикла на экран выводится первое простое число — единица, а переменной `l` сразу присваивается значение 2 (соответствующее второму элементу множества). Результаты работы программы как при неправильном, так и при правильном вводе исходных данных приведены на рис. 10.6.

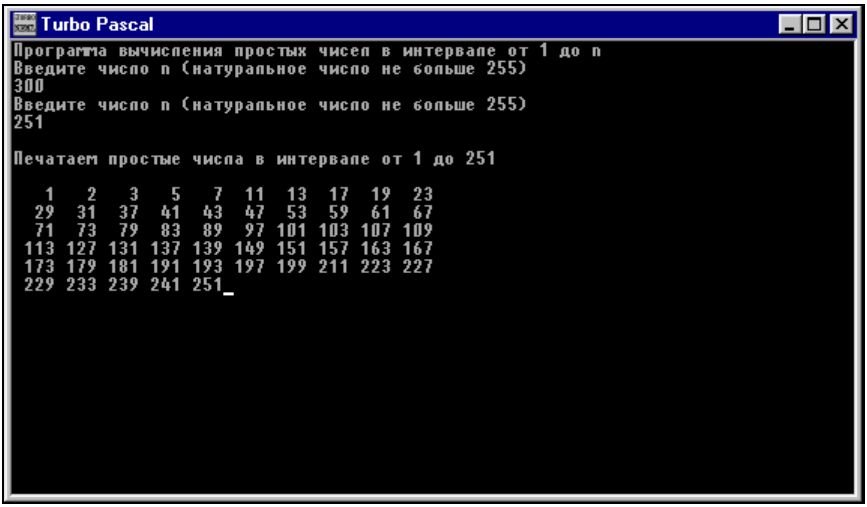


Рис. 10.6. Результаты работы программы поиска простых чисел

Задания для самостоятельной работы

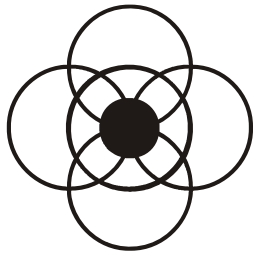
Задание 1. Составить программу, которая по названию месяца определяет текущее время года. Для описания переменной-селектора в условном операторе использовать перечисляемый тип. В программе предусмотреть защиту от неправильного ввода данных.

Задание 2. Составить программу, которая по номеру года определяет, является ли он високосным или нет. Год является високосным в том случае, если его номер делится на 4 (например, високосными являются 1996 или 2004).

Исключение составляют годы, номера которых делятся на 100. Эти годы являются високосными в том случае, если их номера делятся также на 400 (например, 1600 и 2000 годы являются високосными, а 1800 и 1900 — нет). Переменную, которой присваивается значение номера года, описать как переменную ограниченного типа (значение данной переменной должно изменяться в диапазоне от 1600 до 2100). Предусмотреть в программе защиту от неверного ввода данных (вне зависимости от настроек системы Turbo Pascal).

Задание 3. Составить программу, которая создает типизированный файл и вводит в него данные о результатах сдачи экзаменов классом из `n` школьников. Данные по успеваемости каждого из студентов должны быть оформлены в виде записи. Каждая запись должна содержать следующие данные: фамилию и имя школьника, его имя и оценки по следующим предметам: математика, физика, литература, история. Количество учеников, сдающих экзамен, заранее не известно, поэтому после ввода очередной записи программа должна запрашивать пользователя о том, будет ли он вводить следующую запись. После завершения ввода данных на экран должно быть выведено сообщение о количестве записей, имеющихся в файле, а затем должна быть выведена экзаменационная ведомость.

Задание 4. Составить программу, которая должна определять, является ли введенный с клавиатуры символ цифрой, прописной буквой английского алфавита, прописной буквой русского алфавита, строчной буквой английского алфавита, строчной буквой русского алфавита или каким-либо другим символом. При решении задачи использовать множественный тип данных.



ГЛАВА 11

Turbo Pascal в Интернете

Авторы данной книги стремились дать пользователю те необходимые основы теоретических знаний и практических навыков, без которых невозможно дальнейшее совершенствование в области алгоритмического программирования. У многих любознательных и стремящихся к творческой деятельности людей, естественно, возникнет потребность в расширении и углублении полученных знаний путем самообразования. Важным подспорьем в осуществлении этой задачи может стать информация, которую пользователь может получить во всемирной информационной сети Интернет.

Система программирования Turbo Pascal широко используется для обучения основам программирования и для решения практических задач во всем мире уже на протяжении двух десятилетий. В течение столь продолжительного периода времени для такой бурно развивающейся науки, как информатика, было создано много различного рода теоретических трудов и учебников по программированию, а также был накоплен огромный практический опыт по разработке программ в системе Turbo Pascal. Значительная часть этих материалов нашла свое отражение в глобальной сети.

В Интернете существует значительное количество сайтов, основной тематикой которых является программирование в системе Turbo Pascal. На этих сайтах их создателями для общедоступного использования выложено много важных и полезных материалов по программированию в Turbo Pascal. В данной главе приводится обзор тех сайтов, в которых, на взгляд авторов книги, аккумулировано многое из того багажа знаний, о котором говорилось ранее. При этом особый упор авторами сделан на сайты Рунета, т. е. русскоязычного сектора Интернета.

Это объясняется двумя причинами. Во-первых, не все пользователи в достаточной мере владеют английским языком, на котором в Интернете представ-

лена большая часть информации. Во-вторых, на русскоязычных сайтах можно найти много ценных сведений по использованию русифицированных версий системы Turbo Pascal и по разработке программ применительно к потребностям пользователей в нашей стране.

Перед информацией о каждом из сайтов, указанных в данной главе, приводится его адрес в Интернете. Далее сообщается о том, какого рода сведения может почерпнуть для себя пользователь на том или ином сайте.

http://borlpasc.narod.ru — здесь можно найти самую разнообразную информацию, относящуюся к программированию на Паскале. Во-первых, как уже говорилось в *главе 1*, с данного сайта можно скачать саму систему Turbo Pascal, причем не только Turbo Pascal 7.0, но и другие версии языка Паскаль. Во-вторых, на сайте содержится много различной документации по языку Паскаль, включая руководства для пользователей и программистов, справочники и энциклопедию, учебники, курсы лекций и практикумы по Turbo Pascal (рис. 11.1).

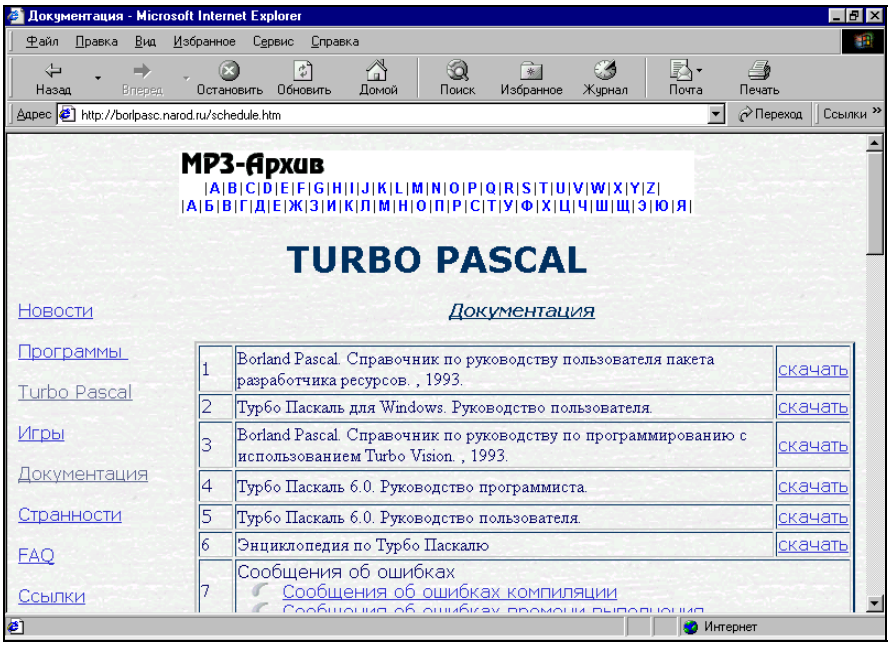


Рис. 11.1. Страница сайта **borlpasc.narod.ru**, посвященная документации по Turbo Pascal

Кроме того, на сайте имеется страница, посвященная ответам на многочисленные вопросы, возникающие у пользователей, осваивающих программирование в системе Turbo Pascal (рис. 11.2). Часто такие страницы в Интернете (в том числе и в Рунете) обозначаются англоязычным сокращением FAQ, что

расшифровывается как Frequently Asked Questions, т. е. "часто задаваемые вопросы".

Рис. 11.2. Страница сайта **borlpasc.narod.ru**, посвященная ответам на вопросы пользователей

На сайте имеется большой архив, содержащий исходные тексты программ на языке Паскаль (рис. 11.3). Для того чтобы пользователям было удобнее ориентироваться в этом архиве, он разбит на ряд разделов. Эти разделы посвящены работе с целочисленными величинами, условными операторами, циклами, массивами, множествами, строковыми величинами, программированию графики, звука и др.

Сайт регулярно обновляется, поэтому здесь имеется специальный журнал, в котором сообщается о том, какая новая информация поступила на сайт в последнее время (рис. 11.4).

http://pascal.sources.ru — это адрес сайта "Паскальные исходники" (рис. 11.5). На сайте можно найти исходные тексты программ на Паскале, ряд статей по программированию на Паскале, различную документацию, а также ссылку на онлайновый учебник по языку Паскаль, находящийся по адресу (рис. 11.6):

http://www.kolasc.net.ru/cdo/metod/programmer'scourse/language/index.htm#pascwil

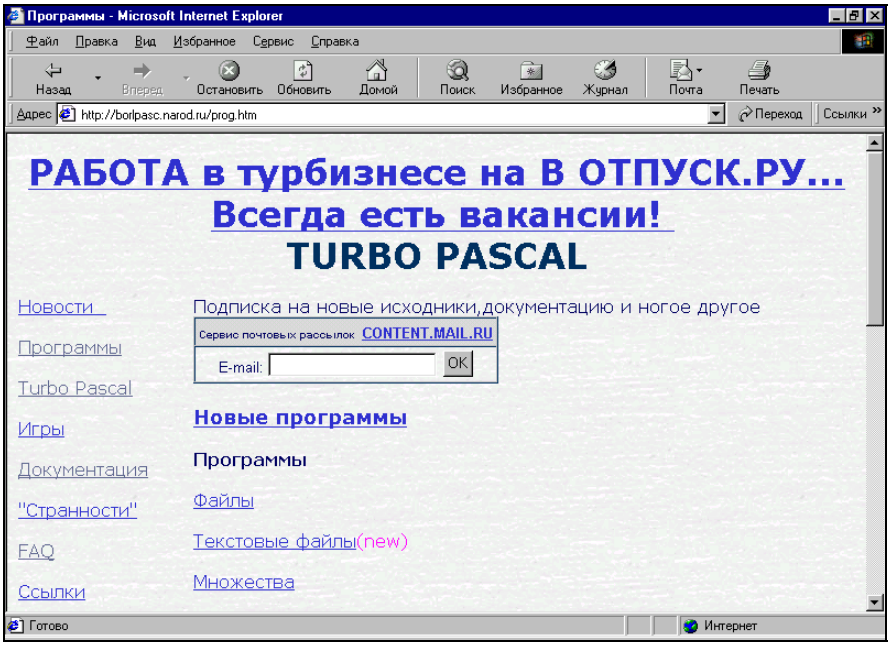


Рис. 11.3. Страница сайта **borlpasc.narod.ru**, содержащая архив с текстами программ на Паскале

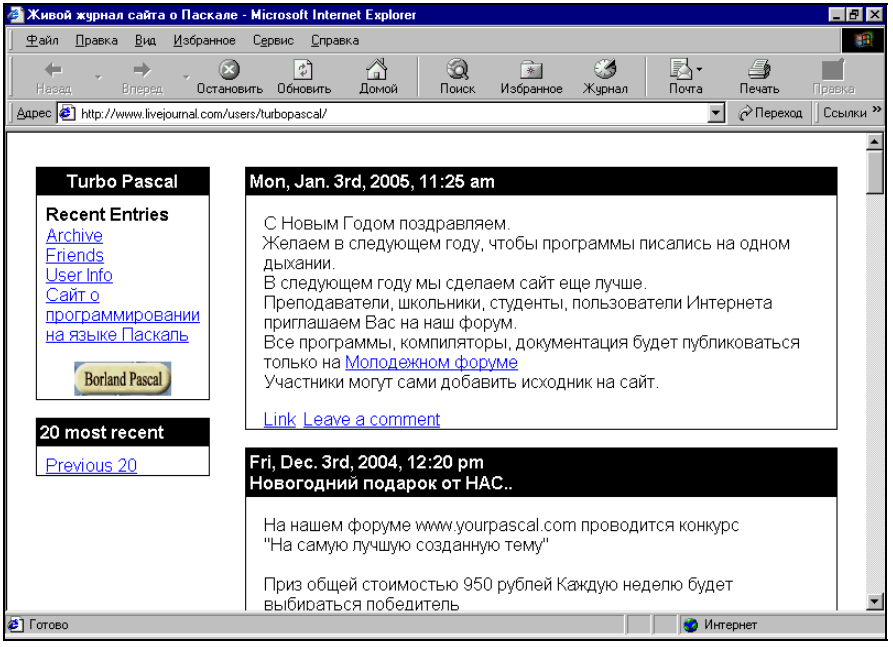


Рис. 11.4. Журнал новостей сайта **borlpasc.narod.ru**

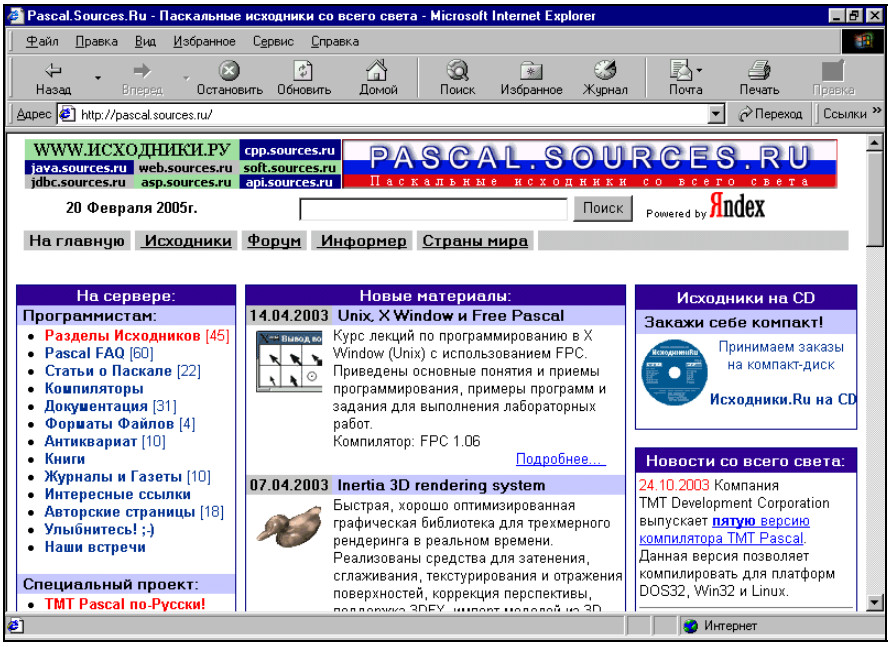


Рис. 11.5. Главная страница сайта "Паскальные исходники"

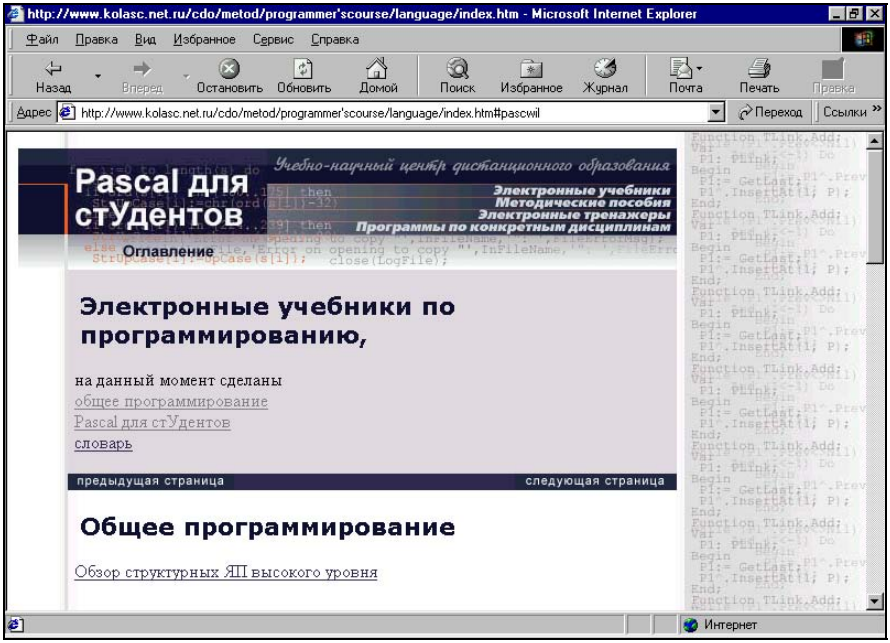


Рис. 11.6. Оглавление онлайнового учебника по программированию на Паскале

http://delphid.dax.ru/pascal — основным направлением сайта "Development и Дельфи", как видно из его названия, является программирование в среде Delphi, но есть на сайте и раздел, посвященный программированию на классическом Паскале (рис. 11.7). В этом разделе есть возможность скачать среду программирования Borland Pascal 7.0, которая является расширенной версией Turbo Pascal 7.0. Имеющаяся на сайте среда Borland Pascal русифицирована и имеет справочную систему (help) на русском языке. Кроме того, на сайте есть учебник по Turbo Pascal, в котором вопросы программирования в этой среде рассмотрены достаточно глубоко и серьезно.

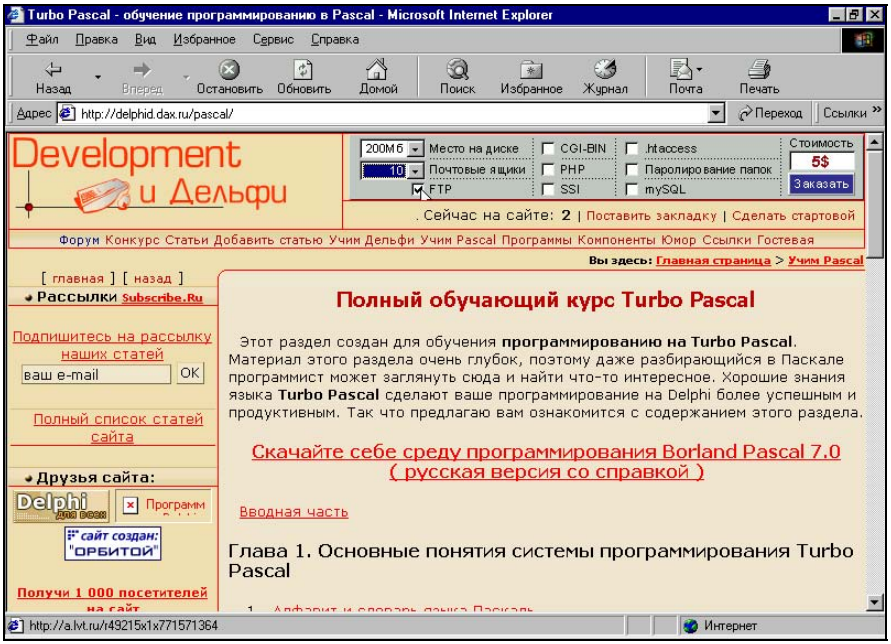


Рис. 11.7. Страница сайта "Development и Дельфи", посвященная программированию на Turbo Pascal

http://web-pascal.narod.ru — этот сайт полностью посвящен программированию на Паскале. Здесь имеется раздел документации, содержащий различную литературу как по программированию на Паскале, так и вообще по компьютерной тематике (рис. 11.8). В разделе документации имеются многочисленные руководства по разработке программного обеспечения, выпущенные фирмой Borland, лекции и книги по программированию, включая сборники олимпиадных задач, задачники с решениями, а также литература по данной тематике на английском языке.

Кроме того, на указанном сайте есть страница, с которой можно загрузить различные версии среды программирования Turbo Pascal, начиная с самой

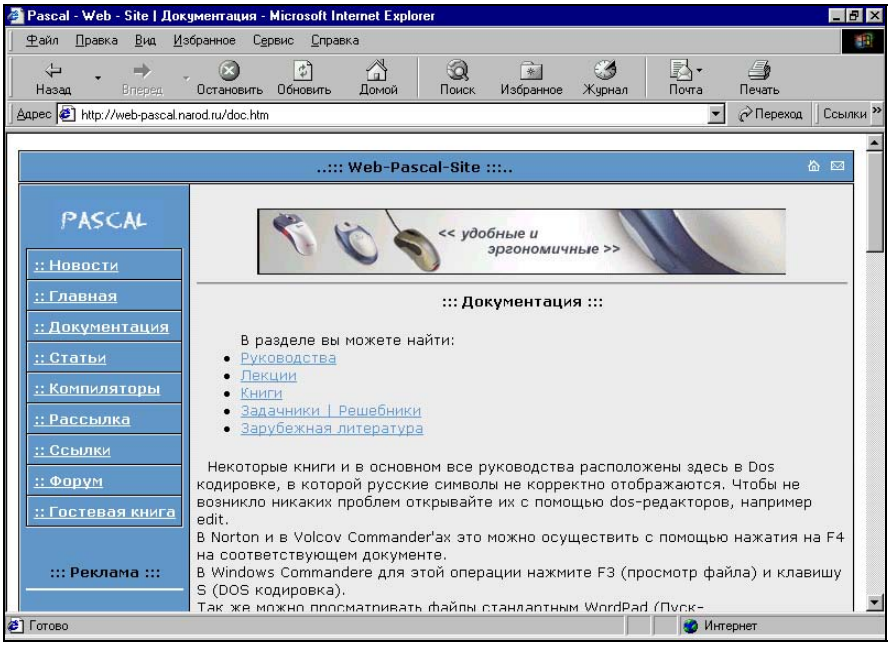


Рис. 11.8. Страница сайта "Web-Pascal", посвященная документации по программированию

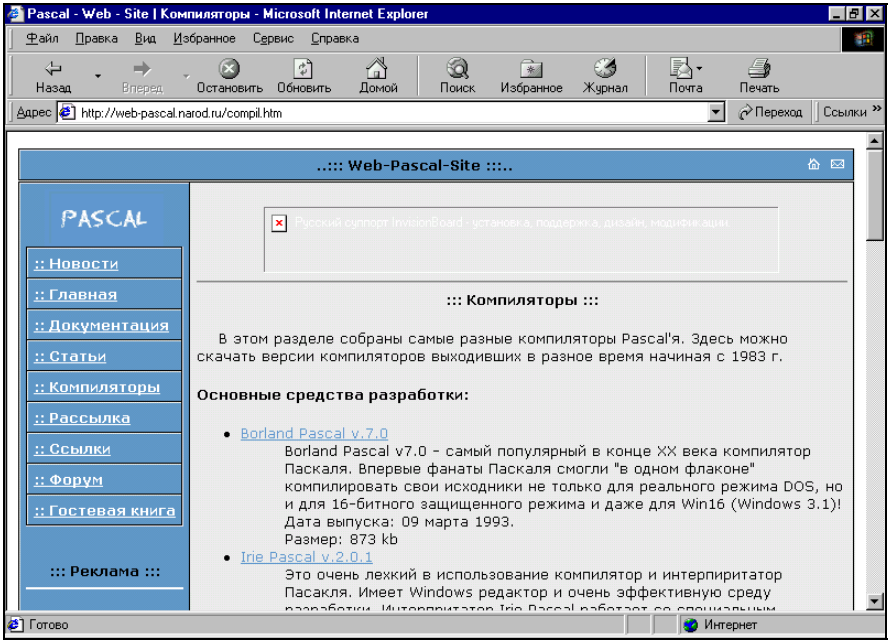


Рис. 11.9. Страница сайта "Web-Pascal", содержащая различные версии компиляторов Паскаля

первой версии 1.0 и заканчивая шестой, а также ряд других компиляторов языка Паскаль (рис. 11.9).

http://www.pascal.dax.ru — по этому адресу расположен сайт под названием "Все о Паскале". Он имеет недолгую историю существования, и в настоящее время идет активное пополнение сайта новой информацией. В частности, на сайте есть раздел, содержащий исходные тексты программ на Паскале, распределенные по разделам **Математика**, **Файлы**, **Целочисленная арифметика**, **Условный оператор**, **Циклы**, **Массивы**, **Множества**, **Графика** и др. (рис. 11.10).

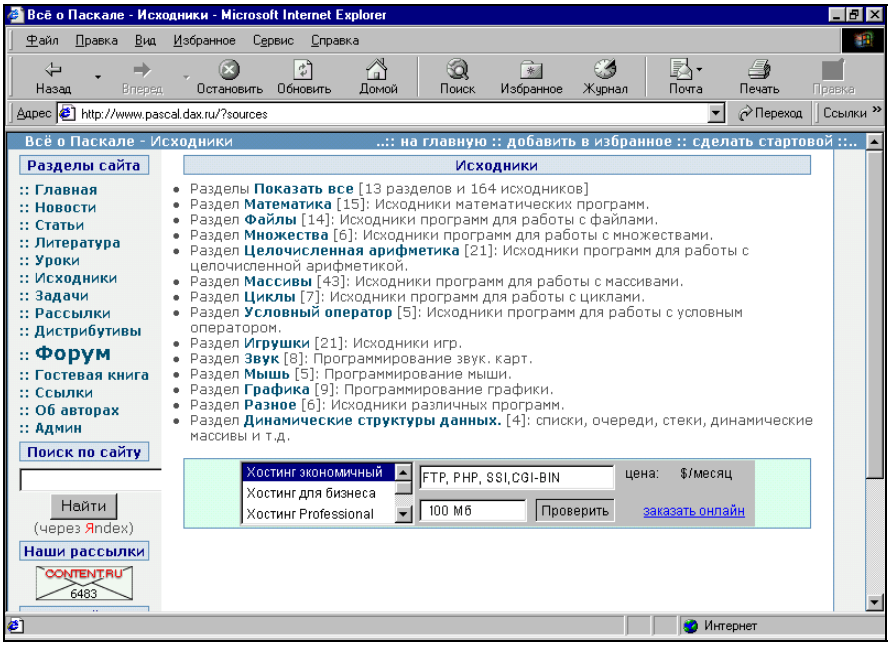


Рис. 11.10. Страница сайта **pascal.dax.ru**, содержащая исходные тексты программ на Паскале

http://www.karelia.ru/psu/Chairs/IMO/pascal — это страница сайта Петрозаводского государственного университета, содержащая онлайновое пособие, в котором изложены основы программирования на Паскале для начинающих, а также задания для самостоятельных и лабораторных работ по программированию (рис. 11.11).

http://www.kbsu.ru/~book — на этом сайте находится интернет-версия издания базового учебного пособия Л. З. Шауцуковой по информатике (рис. 11.12). Данное пособие содержит как теоретические основы информатики, включая теорию программирования, так и практикум по программированию.

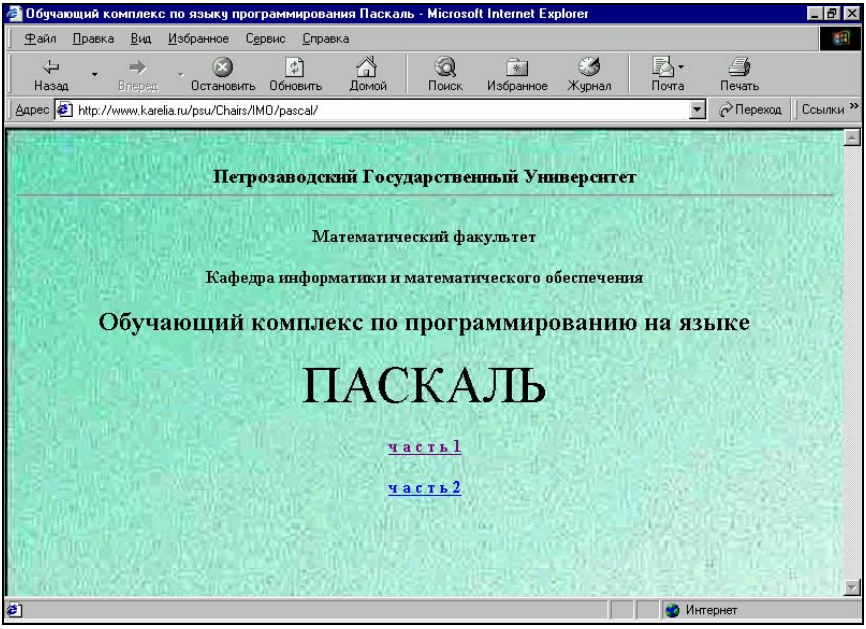


Рис. 11.11. Страница сайта Петрозаводского государственного университета, посвященная программированию на Паскале

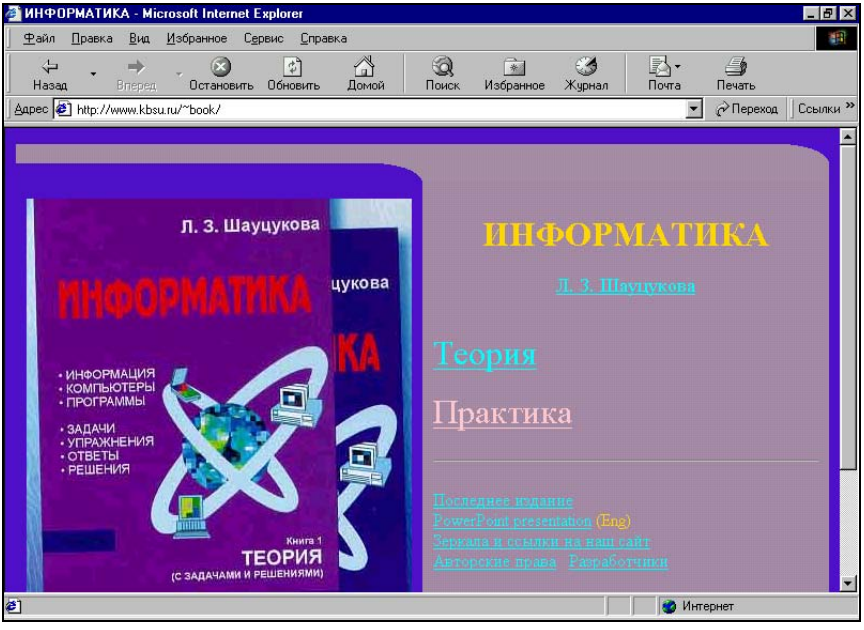


Рис. 11.12. Главная страница сайта, содержащего интернет-версию учебного пособия Л. З. Шауцуковой по информатике

Практикум по программированию, входящий в состав пособия, содержит многочисленные демонстрационные примеры, показывающие реализацию базовых структур программирования на различных языках программирования, в число которых входит и Turbo Pascal (рис. 11.13). Отдельный раздел практикума посвящен программированию графики и звука в Turbo Pascal.

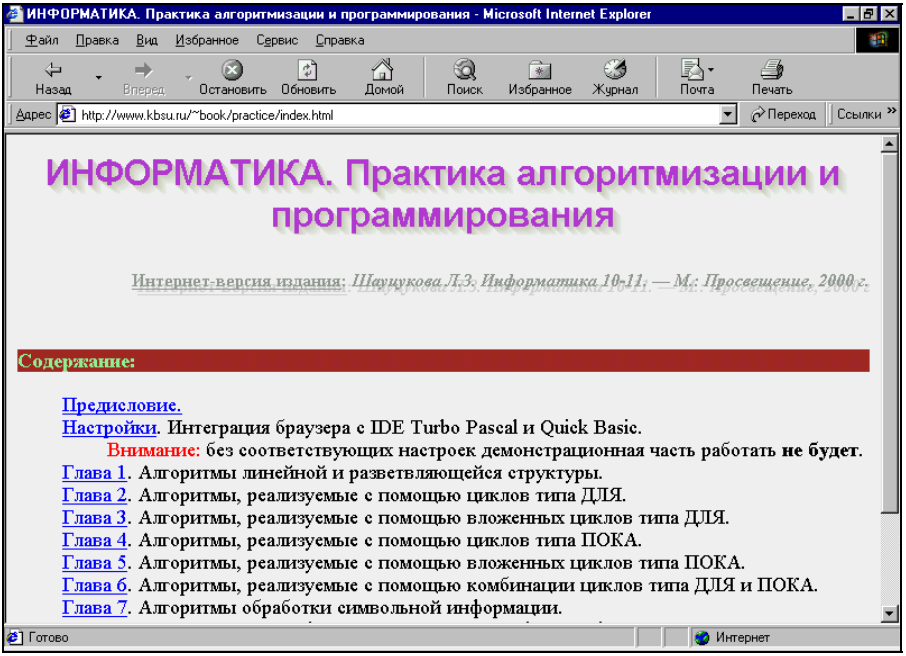


Рис. 11.13. Страница, содержащая оглавление практикума по программированию интернет-издания учебного пособия Л. З. Шауцуковой по информатике

http://trushinov.chat.ru/sp.htm — на этой странице сайта О. В. Трушина содержится ряд сборников, каждый из которых представляет собой набор программ, посвященных одному из разделов программирования в Turbo Pascal: основы работы в текстовом режиме, массивы и циклы, рекурсия, программирование графики и др. (рис. 11.14).

http://www.markbook.chat.ru — это адрес сайта преподавателя информатики М. Б. Львовского, на котором находится методическое пособие по информатике для старшеклассников Book (рис. 11.15). В данной книге имеется ряд уроков, посвященных основам программирования в системе Turbo Pascal.

http://www.belc.narod.ru/train.htm — это страница сайта П. П. Павловского, с которой можно загрузить две обучающие программы по Turbo Pascal, а также ряд других полезных обучающих программ (рис. 11.16).

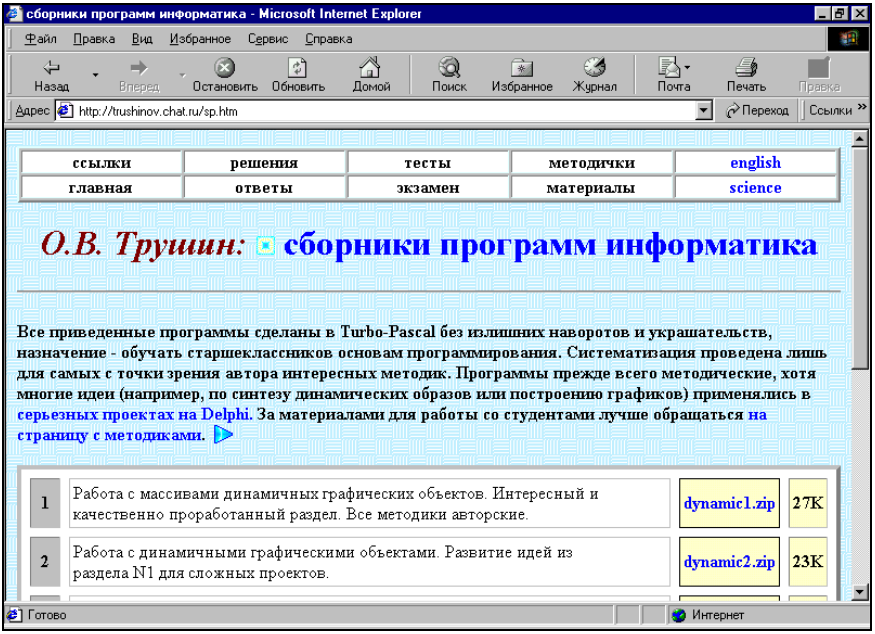


Рис. 11.14. Страница сайта О. В. Трушина, содержащая ряд сборников программ на Turbo Pascal

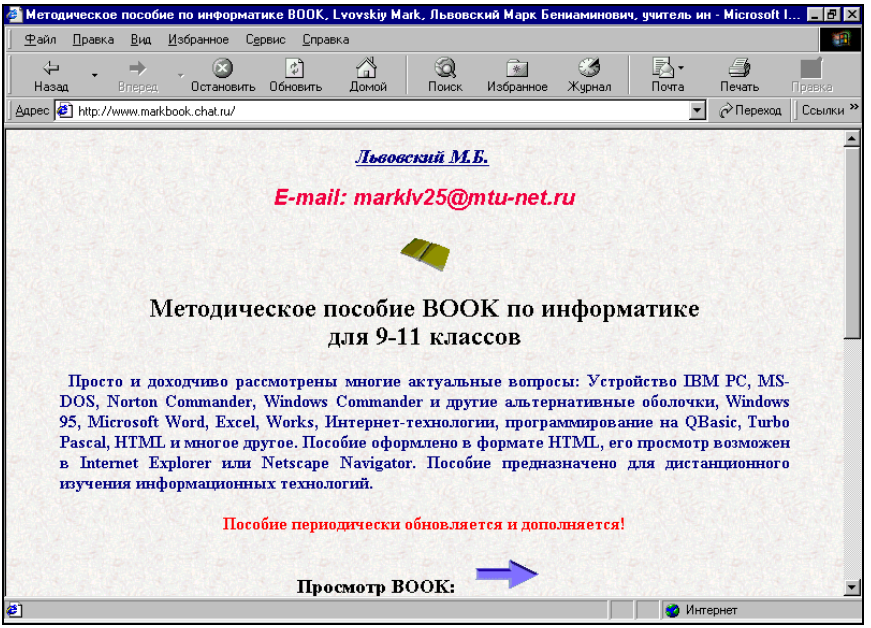


Рис. 11.15. Страница сайта М. Б. Львовского, содержащая методическое пособие по информатике BOOK

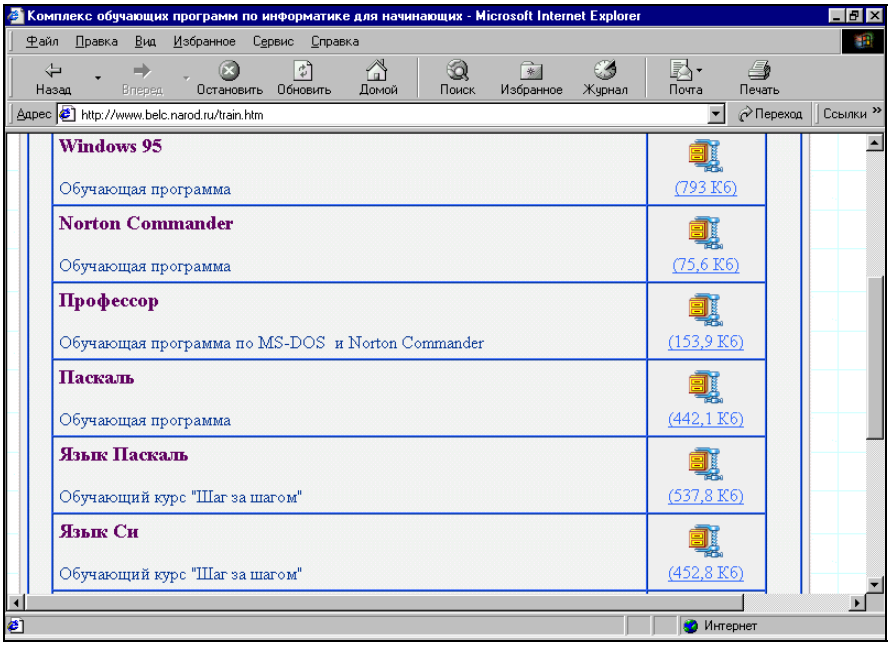
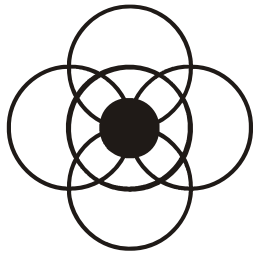


Рис. 11.16. Страница сайта П. П. Павловского, с которой можно загрузить обучающие программы по Паскалю

Помимо указанных ранее страниц, полезную информацию о языке Паскаль можно найти также на ряде сайтов, посвященных программированию в системе Delphi, которой будут посвящены следующие главы данной книги.



ГЛАВА 12

Предварительные сведения о среде Delphi

После освоения программирования на языке Паскаль можно переходить к программированию в среде Delphi. Дело в том, что в системе Delphi для написания кода используется язык Object Pascal, который является дальнейшим развитием языка Turbo Pascal. Установка среды Delphi может быть осуществлена с оптического диска или из Интернета с сайта производителя этого программного продукта. Установка с оптического диска более предпочтительна, т. к. программа имеет очень большой размер.

12.1. Основные характеристики системы Delphi

В свое время система программирования Turbo Pascal была одной из ведущих систем программирования для операционной системы MS-DOS, но в 90-е годы XX столетия ведущей операционной системой для персональных компьютеров стала ОС Windows, и встал вопрос о создании мощной и в то же время достаточно простой и удобной для работы среды создания Window-приложений. К разработке такой системы и приступила компания Borland. Новый программный продукт, как уже говорилось в предисловии, получил название Delphi. Каковы же основные отличительные особенности, позволившие занять Delphi достойное место в ряду современных средств программирования?

Система программирования Delphi *опирается на* возможности операционной системы *Windows*. Как и в Windows, в Delphi используется графический интерфейс, обеспечивающий взаимодействие компьютера с пользователем. Значительную часть операций в процессе программирования пользователь может выполнять с помощью мыши, что повышает его производительность

труда (при использовании мыши производительность труда пользователя повышается в 2—3 раза по сравнению с использованием только клавиатуры).

Начав работу в системе Delphi, пользователь увидит многие элементы интерфейса, которые уже знакомы ему по работе в Windows. К их числу относятся окна, которые можно закрывать, сворачивать или, наоборот, разворачивать до полноэкранного размера. Так же как в Windows, размеры большинства окон можно изменять методом протягивания. Этот метод заключается в том, что пользователь наводит указатель мыши на границу окна с тем, чтобы указатель превратился в двунаправленную стрелку, а затем перемещает границы окна при нажатой кнопке мыши. Имеющиеся в системе программирования окна можно перемещать по экрану, "ухватившись" указателем мыши за строку заголовка, аналогично тому, как это делается в Windows. Если информация целиком не помещается в отведенное для нее окно, то справа или снизу в рабочей области окна появляются полосы прокрутки (скроллеры).

При работе в Delphi пользователю часто придется пользоваться меню, которые организованы так же, как меню в офисных приложениях Windows, т. е. имеется горизонтальная строка, содержащая ряд разделов, каждый из которых является ниспадающим вертикальным меню. В системе Borland Delphi имеются многочисленные диалоговые окна, содержащие те же элементы управления, что и в Windows. К числу таких элементов относятся флажки, переключатели, или радиокнопки, раскрывающиеся списки, палитры и др. Использование всех этих элементов точно такое же, как в Windows.

В Delphi в полном масштабе *реализовано объектно-ориентированное программирование*. Если при выполнении программ, созданных в Turbo Pascal, возможные действия пользователя жестко регламентировались самой программой, то в Delphi мы видим иную картину. Готовое приложение, как правило, реализуется в Delphi в виде прямоугольной формы, которая выводится на экране на передний план при запуске программы на выполнение. Данная форма содержит различные элементы, к числу которых относятся надписи, текстовые поля, экранные кнопки, различные графические объекты, а также уже упоминавшиеся радиокнопки и флажки.

В ходе работы с приложением пользователь может воспользоваться всеми имеющимися на форме элементами, а может использовать только их часть, причем он сам будет определять, какими именно элементами следует воспользоваться, задавая тем самым свой собственный режим работы. Таким образом обеспечивается больший демократизм работы с программным продуктом по сравнению с приложениями MS-DOS.

Элементы, расположенные на форме, называются *объектами приложения*. К числу объектов относится и сама форма или формы, имеющиеся в приложении. Каждый объект обладает определенным набором *свойств*, которые

можно настраивать либо с помощью средств графического интерфейса, либо программным образом. Каждое из свойств может иметь различные значения. Минимальное количество значений свойства равно двум. Например, надпись на форме может быть видимой или невидимой в зависимости от значения свойства объекта "видимость" (по английски — Visible). Если значение свойства `Visible` равно `true` (истина), то объект будет видимым, а если свойство `Visible` имеет значение `false` (ложь), то объект будет невидимым. Другие свойства того же объекта могут иметь большее количество значений. К таким свойствам относится, например, цвет надписи, который может быть черным, белым, красным, синим, зеленым, желтым и т. д.

Свойства объекта не являются раз и навсегда неизменными. Они могут изменяться в ходе работы приложения при воздействии пользователя на объект. Такие действия, производимые пользователями в отношении объекта, называются *событиями*. Для каждого объекта существует свой перечень событий, на которые он может реагировать. К числу наиболее распространенных событий относится щелчок левой кнопкой мыши на объекте, щелчок правой кнопкой, двойной щелчок, наведение мыши на объект, нажатие клавиши <Enter>, нажатие кнопки мыши без ее отпускания, перетаскивание объекта, изменение текста, который содержит объект и др. Помимо событий, инициированных пользователем, могут быть еще и системные события, о которых мы поговорим позже.

Поскольку ход работы приложения в значительной степени определяется реакцией объектов на соответствующие им события, то систему программирования Delphi часто называют также *системой событийно-ориентированного программирования*. Реакция объекта на какое-либо событие, как правило, записывается в виде отдельной процедуры на языке программирования. В системе Delphi языком, используемым для написания кода, является Object Pascal, который представляет собой дальнейшее логическое развитие языка Turbo Pascal. Процедуру, описывающую реакцию объекта на некоторое событие, часто называют также *обработчиком* этого события.

Delphi является *системой визуального программирования*. Это означает, что в системе программирования имеются готовые библиотеки элементов, которые можно использовать для создания будущего приложения. Такие элементы называются *компонентами приложения*. Выбор нужного компонента не требует написания специального кода, а производится с помощью мыши, т. е. графическим или, иначе говоря, визуальным способом. Когда объект уже перенесен посредством перетаскивания на форму, то его дальнейшая настройка большей частью также производится визуальным способом с помощью мыши.

Например, если нам нужно изменить цвет объекта, то мы можем открыть раскрывающийся список и из него выбрать подходящий цвет. Изменять гео-

метрические размеры объекта можно методом протягивания, подобно тому как это делается для окон, хотя надо сказать, что система Delphi допускает и более точный способ регулировки размеров объекта путем изменения числовых значений свойств `Height` и `Width` (соответственно высота и ширина).

В некоторых случаях без использования клавиатуры все же не обойтись, например при изменении заголовка объекта. Но и в этом случае характеристики шрифта можно изменять визуальным способом. Для этого достаточно открыть диалоговое окно **Font** (Шрифт). В нем путем выбора значений из раскрывающихся списков можно выбрать гарнитуру шрифта, его начертание, кегль (размер шрифта). Из имеющейся в диалоговом окне палитры можно выбрать цвет шрифта, а установив щелчком мыши соответствующий флажок, можно сделать текст подчеркнутым или зачеркнутым.

В ходе дальнейшего изложения материала мы более подробно остановимся на указанных ранее характеристиках системы и покажем, как они используются для разработки конкретных приложений.

12.2. Как установить Delphi

Понятно, что для того чтобы научиться программировать в системе Delphi, нужно эту систему иметь на компьютере. Система не будет работоспособна, если не произвести предварительно процедуру ее установки на компьютер. Установка системы программирования возможна двумя способами. Во-первых, через Интернет. Для этого нужно выйти на сайт компании Borland Inc., которая является производителем данной системы. С сайта можно установить на компьютер условно-бесплатную версию системы программирования.

Однако данный способ обладает двумя существенными недостатками. Во-первых, даже "облегченная" версия системы программирования занимает свыше 100 Мбайт, что в условиях нашей страны, где выход в Интернет осуществляется в основном по телефонным линиям, потребует для установки очень большого количества времени и соответствующих материальных затрат. Если учесть, что передача одного мегабайта информации по отечественным телефонным линиям занимает в среднем около 10 минут, то нетрудно подсчитать, что для загрузки всех необходимых файлов системы потребуется порядка 20 часов непрерывного подключения к Интернету. Во-вторых, установленная версия системы будет иметь ограничения как по времени ее использования, так и по своим функциональным возможностям. Поэтому наиболее рациональным путем в наших условиях является установка системы программирования с компакт-диска, на котором имеется полноценная версия данной системы. Поскольку для начинающего пользователя процедура установки программного комплекса с лазерного диска может представлять определенные трудности, мы подробно опишем эту процедуру в данной главе.

На момент выхода данной книги в свет наиболее распространенными версиями системы программирования Delphi являются 5.0, 6.0 и 7.0. В данной главе мы рассмотрим установку Delphi на примере шестой версии (процесс установки пятой и седьмой версии никаких принципиальных отличий не имеет).

После установки диска в дисковод вы увидите экран оболочки, на котором вам предлагается выбрать устанавливаемую версию программного продукта.

В меню щелчком мыши выбираем английскую версию системы программирования, т. к. без ее предварительной установки нельзя будет потом установить русифицированную версию программы. После выбора версии начинается сам процесс установки. На экране компьютера появляется окно оболочки программы установки Delphi 6.0 (рис. 12.1).

Рис. 12.1. Окно оболочки программы установки Delphi

Подводя указатель мыши к каждому компоненту системы, отмеченному треугольной стрелкой, можно получить краткую информацию (на английском языке) о нем. Например, если подвести указатель мыши к надписи Delphi 6 (основное ядро системы программирования) и задержать его там на несколько секунд (при этом указатель приобретает форму человеческой руки), то вы сможете увидеть следующее сообщение: "Requires 350 mb hard disk space for full install". Это означает, что для полноценной установки системы программирования на ваш компьютер требуется 350 Мбайт свободного места на жестком диске компьютера. Если у вас имеется требуемое количество свободного места, то можно приступать к дальнейшей установке системы программирования. Для этого достаточно щелкнуть мышью указанную надпись.

Подготовку к процессу установки производит специальная программа-инсталлятор (Windows Installer) (рис. 12.2). Пока эта программа выполняет подготовительную работу, никаких действий производить не следует.

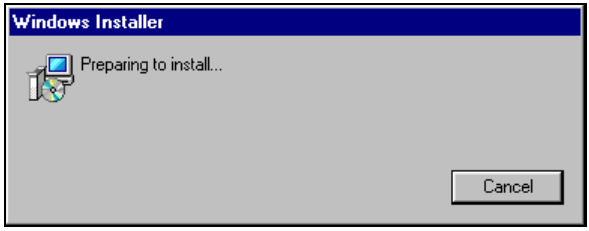


Рис. 12.2. Окно программы-инсталлятора

По завершении работы инсталлятора на экране появляется окно специальной программы, которая и производит непосредственную работу по установке Delphi (рис. 12.3). Эта программа называется Installation Wizard, что в переводе с английского означает "мастер установки".

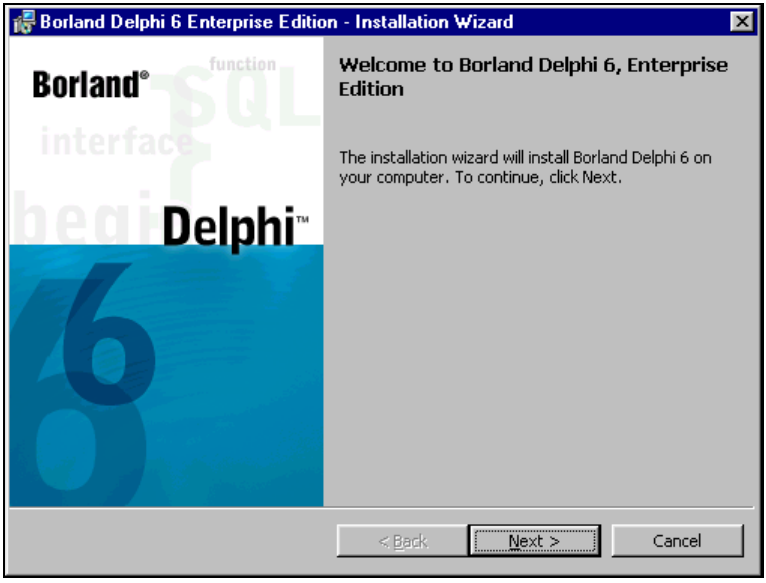


Рис. 12.3. Окно мастера установки Delphi

Для перехода к следующему этапу установки программы нужно щелкнуть мышью на экранной кнопке **Next** (Далее). После щелчка на этой кнопке на экране появляется следующее окно, в котором нужно ввести серийный номер устанавливаемого программного продукта (**Serial Number**) и ключ програм-

мы (**Authorization Key**) (рис. 12.4). Без правильного указания этих данных дальнейшая установка системы программирования будет невозможна. Как правило, серийный номер и ключ указываются на упаковке компакт-диска.

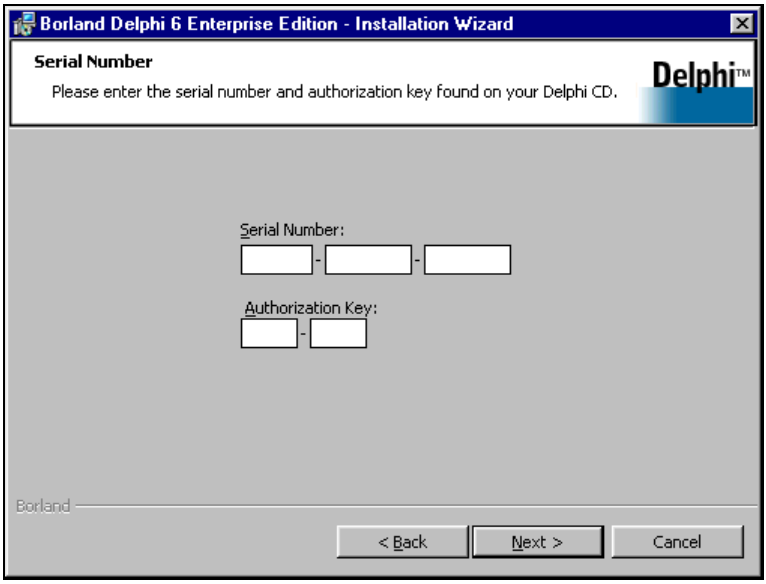


Рис. 12.4. Окно ввода серийного номера продукта и ключа программы

Серийный номер состоит из трех полей. После заполнения очередного поля курсор (имеющий вид вертикальной черты) переходит в следующее поле. Когда все три поля правильно заполнены, можно переходить к вводу ключа программы. Этот переход осуществляют нажатием клавиши <Tab> либо щелкнув мышью первое поле для ввода ключа. Принцип ввода ключа тот же, что и для серийного номера. После того как все поля правильно заполнены, можно переходить к следующему этапу установки системы программирования щелчком на кнопке **Next**.

Если пользователь ошибся при вводе указанных данных, на экране компьютера появляется окно с предупреждением о том, что следует еще раз сверить введенную информацию с тем, что указано на упаковке диска, и исправить данные, содержащиеся в полях. Наиболее типичная ошибка при вводе заключается в том, что пользователь перед вводом номера и ключа забыл переключить клавиатуру с русского регистра на английский. Окно с предупреждением об ошибке следует закрыть щелчком на экранной кнопке **OK**, исправить допущенные ошибки и снова нажать кнопку **Next** для перехода к следующему этапу установки.

Далее на экране компьютера появляется окно с текстом лицензионного соглашения об установке системы программирования (на английском языке). Так как текст лицензионного соглашения достаточно велик по объему, он целиком не помещается в окне, и для того чтобы с ним полностью ознакомиться, нужно использовать расположенную справа от текста полосу прокрутки (рис. 12.5).

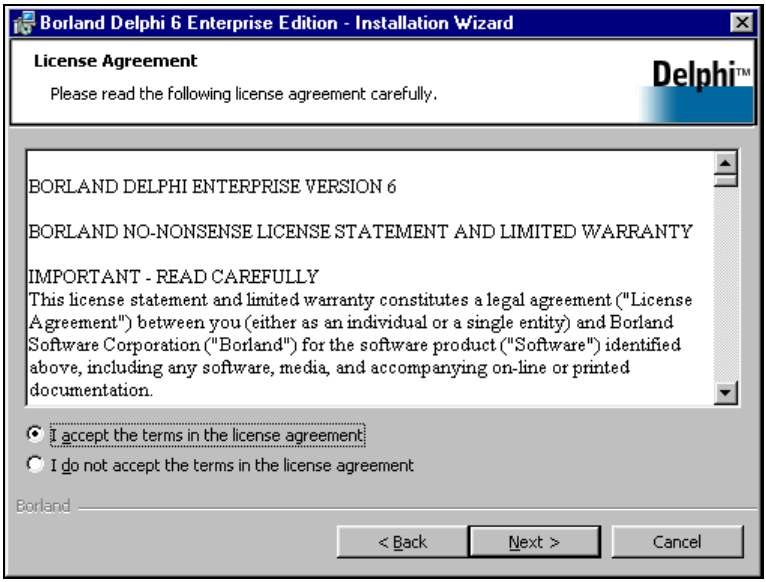


Рис. 12.5. Окно с текстом лицензионного соглашения об установке Delphi

Под текстом соглашения находится группа из двух переключателей. Если пользователя не устраивают условия лицензионного соглашения, то нужно щелкнуть нижний переключатель, на котором написано **I do not accept the terms in the license agreement** (Я не принимаю условий лицензионного соглашения). В этом случае установка системы программирования отменяется. Если пользователь согласен с лицензионным соглашением, то нужно щелкнуть верхний переключатель с надписью **I accept the terms in the license agreement** (Я принимаю условия лицензионного соглашения), а затем щелкнуть на кнопке **Next** для продолжения процесса установки.

Далее на экране компьютера появляется окно с текстом руководства по дальнейшей установке системы программирования (рис. 12.6). Текст этого руководства также целиком не помещается в отведенном для него окне, поэтому для перемещения по тексту, как и в предыдущем случае, следует использовать полосы прокрутки. Так как текст написан на английском языке, то для пользователей, не владеющих этим языком, приведем следующую информа-

цию, содержащуюся в данном руководстве и имеющую важное значение для нормальной установки и дальнейшей работы системы программирования.

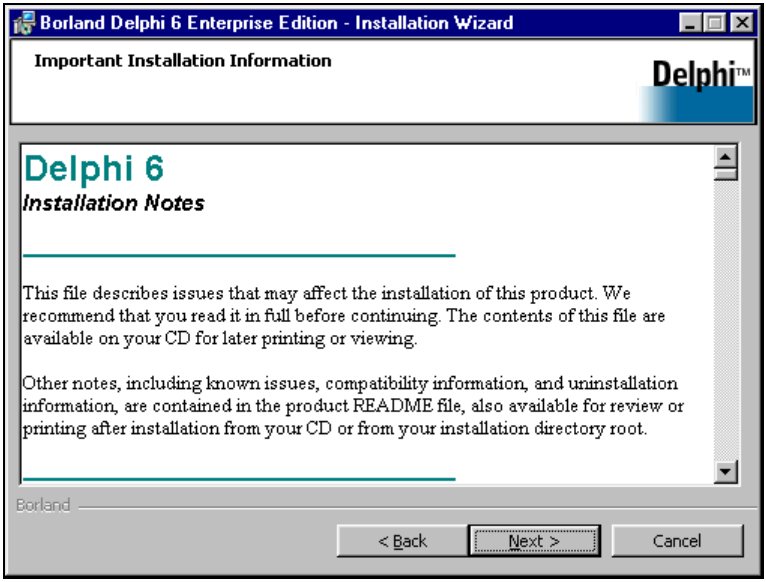


Рис. 12.6. Окно с текстом руководства по установке системы программирования

Для того чтобы систему программирования можно было установить на компьютере, он должен отвечать следующим требованиям:

- компьютер должен иметь как минимум центральный процессор Pentium с тактовой частотой не менее 166 МГц (рекомендуется Pentium 2 с тактовой частотой не менее 400 МГц);
- на компьютере должна быть установлена операционная система Windows 98 либо другие более новые версии той же операционной системы (Windows 2000, Windows ME и т. д.);
- компьютер должен иметь оперативную память объемом не менее 64 Мбайт (рекомендуется 128 Мбайт);
- компьютер должен иметь не менее 115 Мбайт свободного места на жестком диске для компактной установки системы и не менее 350 Мбайт для установки полноценной версии системы;
- компьютер должен иметь дисковод для компакт-дисков;
- монитор компьютера должен поддерживать разрешение не менее чем 640×480 пикселов;
- компьютер должен иметь мышь или какое-либо другое координатное устройство (трекбол, сенсорную панель).

Если ваш компьютер удовлетворяет указанным требованиям, то вы можете быть уверены в том, что дальнейшая установка системы пройдет без проблем, а сама работа с системой программирования будет достаточно уверенной и комфортной. В этом случае после ознакомления с руководством нужно нажать кнопку **Next** для продолжения процесса установки.

На следующем этапе вам нужно выбрать вариант установки системы программирования (рис. 12.7). Мастер установки предлагает на выбор три варианта: **Typical** (Обычный), **Compact** (Компактный) и **Custom** (Выборочная установка). В большинстве случаев рекомендуется выбирать обычный способ установки. Компактный способ установки нужно использовать, когда на жестком диске компьютера мало свободного места. В этом случае устанавливаемая версия будет работоспособна, но в то же время она будет иметь ряд функциональных ограничений. Выборочная установка рекомендуется для опытных пользователей, которые уже работали с более ранними версиями той же системы программирования, и знают, какие компоненты им будут нужны для успешной работы с системой, а какие можно не устанавливать. При выборочной установке можно также указать диск и папку, в которую будет установлена система программирования.

Выбор наиболее подходящего пользователю варианта установки производится щелчком на соответствующем переключателе. После выбора оптимального варианта щелчком на кнопке **Next** переходим к следующему этапу установки программы.

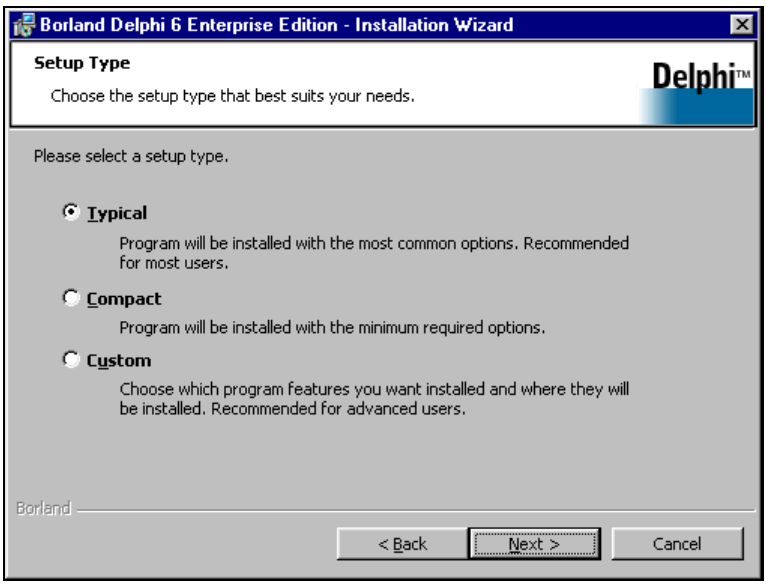


Рис. 12.7. Окно выбора варианта установки системы программирования

Мастер установки запрашивает пользователя, нужно ли устанавливать дополнительный компонент VisiBroker. Выбор одного из двух возможных вариантов установки или отказ от установки компонента производится щелчком на одном из трех имеющихся в окне мастера переключателей (рис. 12.8). Поскольку данный компонент в случае необходимости можно будет установить позже, то на этом этапе можно отказаться от его установки, щелкнув переключатель **No VisiBroker/CORBA Support**. Затем щелчком на кнопке **Next** переходим к очередному этапу установки.

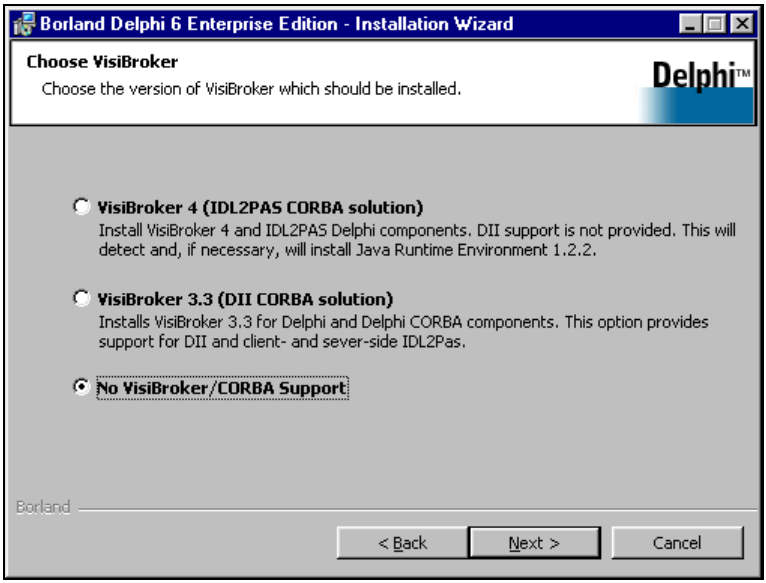


Рис. 12.8. Окно выбора варианта установки дополнительного компонента VisiBroker

На следующем этапе мастер установки запрашивает пользователя, какой набор элементов управления для пакета Microsoft Office следует использовать по умолчанию (рис. 12.9). В принципе система программирования обеспечивает поддержку элементов управления и для версии Office 97 и для версии Office 2000, но если у вас на компьютере установлена версия пакета Office 97, то рекомендуется щелкнуть соответствующий переключатель. Если же на компьютере установлена версия пакета Office 2000 или Office XP, то щелкаем переключатель **Office 2000**. После выбора переключателя щелчком на экранной кнопке **Next** переходим к следующему этапу.

Далее мастер установки запрашивает, нужно ли при установке пакета также устанавливать дополнительный компонент InterBase Client (рис. 12.10). Установка данного компонента требуется для того, чтобы по окончании процесса установки можно было полностью использовать все возможности системы

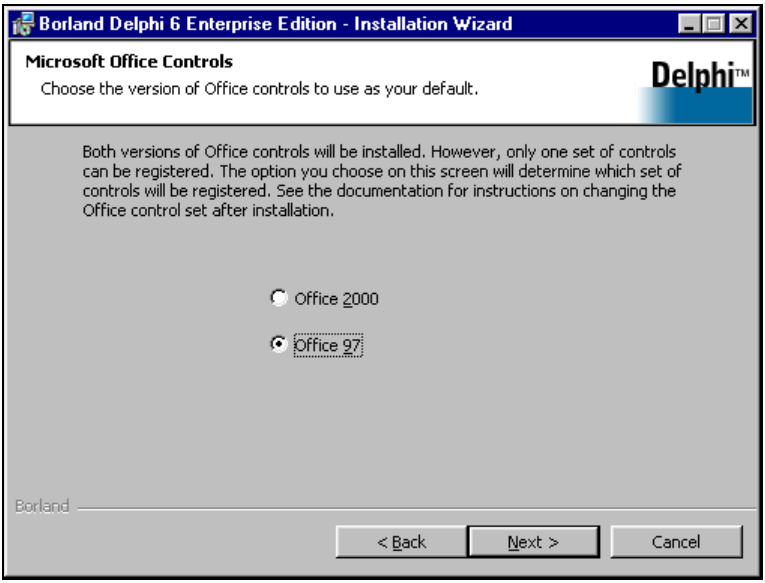


Рис. 12.9. Окно выбора набора элементов управления для Microsoft Office

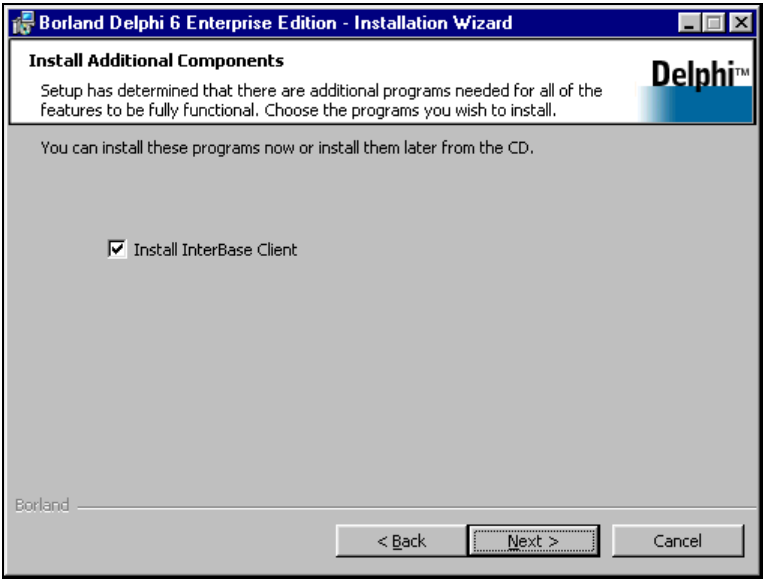


Рис. 12.10. Окно выбора установки компонента InterBase Client

программирования. Для установки этого компонента требуется порядка 10 Мбайт на жестком диске компьютера, поэтому если на винчестере достаточно свободного места, то можно подтвердить его установку, отметив соот-

ветствующий флажок. Если же на винчестере имеется дефицит свободного места, то флажок можно снять щелчком мыши на нем. Затем щелчком на кнопке **Next** переходим к следующему этапу установки.

Далее мастер установки выводит на экран дополнительное лицензионное соглашение по установке указанного ранее компонента (рис. 12.11). Если пользователь согласен с условиями данного соглашения, то подтверждаем это щелчком на верхнем переключателе **I agree with the terms of this license**. Если же пользователь щелкнет нижний переключатель, на котором написано **I do not agree. Installation will continue, but without remote data functions**, то процесс установки самой системы программирования будет продолжен, но при этом указанные дополнительные компоненты устанавливаться не будут. Затем щелчком на кнопке **Next** переходим к дальнейшей установке системы.

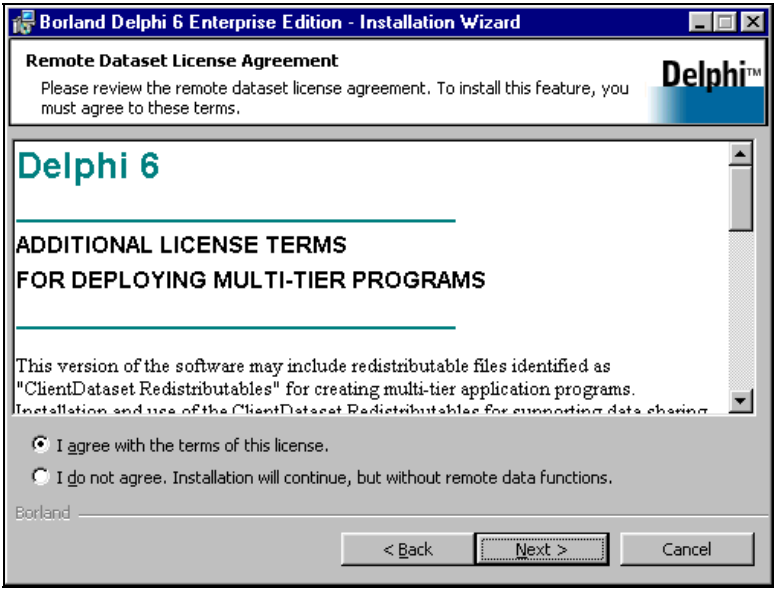


Рис. 12.11. Окно, содержащее текст лицензионного соглашения по установке дополнительного компонента системы программирования

На следующем этапе мастер установки сообщает пользователю о том, на каком логическом диске и в каких папках будут установлены компоненты программы (рис. 12.12). Например, основные файлы системы программирования будут установлены на логическом диске C: в папке, находящейся по адресу Program Files\Borland\Delphi6. В принципе можно изменить логический диск и папку для установки программных файлов, щелкнув на экранной кнопке **Change**. Щелчок на данной кнопке открывает диалоговое окно, позволяющее

выбрать нужный диск и папку. Однако если на логическом диске имеется достаточное количество свободного места, то лучше указанные адреса не менять, а перейти щелчком на кнопке **Next** к следующему этапу установки.

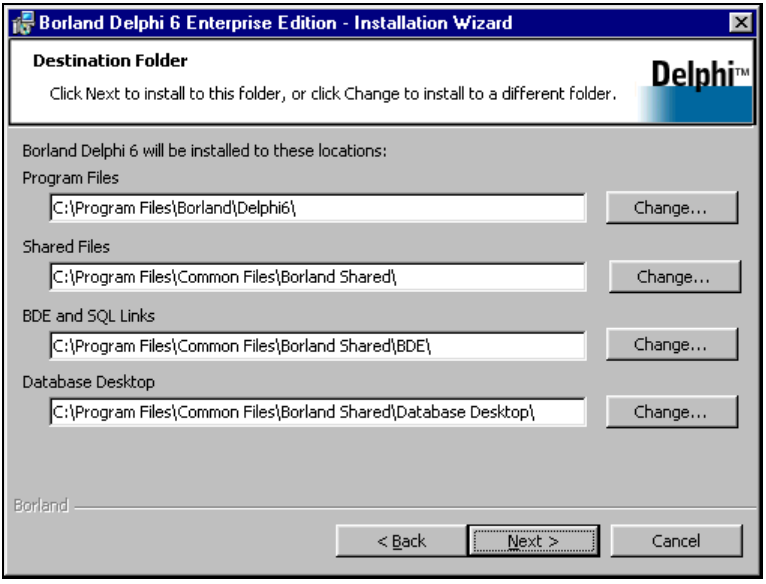


Рис. 12.12. Окно выбора дисков и папок для установки системы программирования

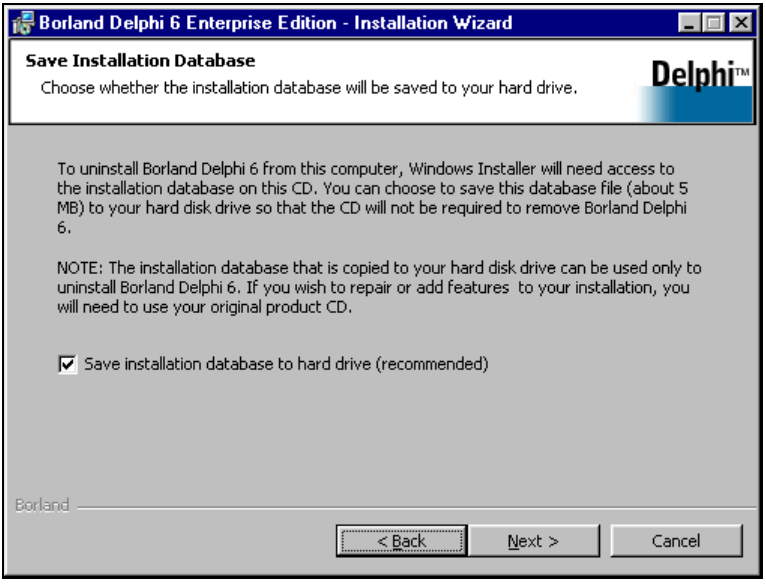


Рис. 12.13. Окно сохранения базы данных по установке системы программирования

Далее мастер установки запрашивает пользователя, нужно ли сохранять на жестком диске компьютера базу данных по установке системы программирования (рис. 12.13). Файлы базы данных занимают на винчестере около 5 Мбайт. Они могут понадобиться в том случае, если необходимо удалить систему программирования с компьютера, не используя при этом сам компакт-диск. Рекомендуется создать такую базу данных, для чего нужно оставить без изменений имеющийся в окне флажок. Затем щелчком на кнопке **Next** переходим к следующему этапу установки.

Наконец, на экране компьютера появляется окно мастера, в котором сообщается о том, что он готов к началу установки системы программирования (рис. 12.14). Для того чтобы начать процесс установки системы, необходимо щелкнуть на экранной кнопке **Install** (Установить), что мы и делаем.

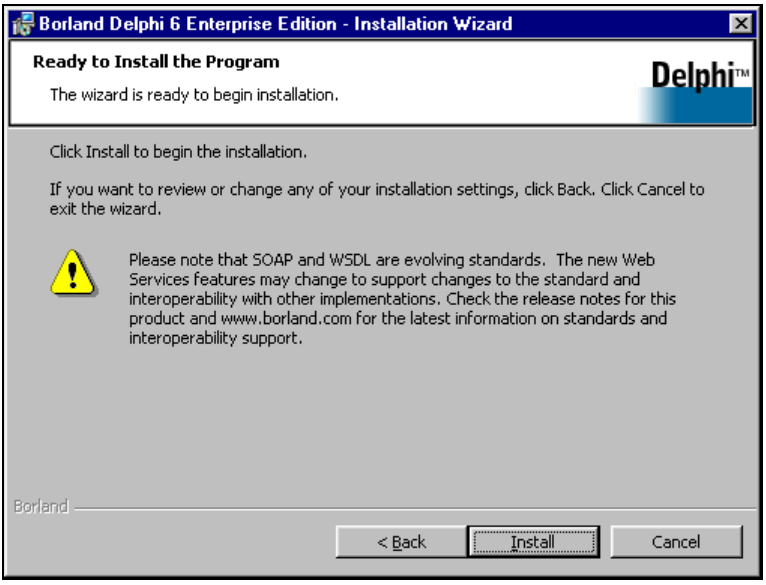


Рис. 12.14. Окно, содержащее сообщение о том, что программа-мастер готова начать установку системы программирования

После щелчка на кнопке **Install** на экран компьютера выводится окно, в котором пользователю демонстрируется процесс копирования файлов, необходимых для установки программы (рис. 12.15). Для наглядности ход установки представляется в виде специальной диаграммы. Серая полоса в нижней части окна мастера постепенно заполняется прямоугольниками синего цвета. Над диаграммой в центральной части окна выводятся различные сообщения на английском языке, в которых рассказывается об основных характеристиках и достоинствах устанавливаемой системы программирования. Когда серая по-

лоса оказывается целиком заполнена синими прямоугольниками, это означает, что процесс копирования завершен.

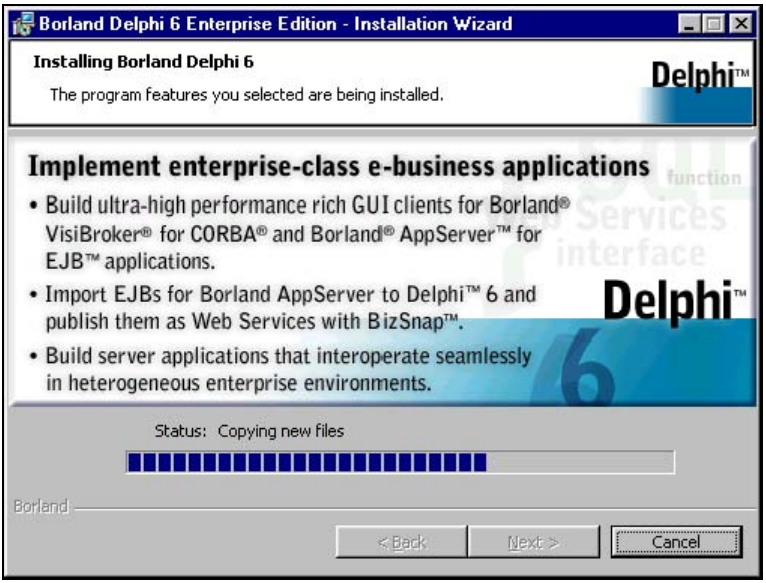


Рис. 12.15. Окно программы-мастера, показывающее в виде диаграммы процесс копирования файлов, необходимых для установки программы

Затем мастер в автоматическом режиме удаляет вспомогательные файлы, которые более не нужны для установки программы и переходит к установке дополнительного компонента InterBase Client (рис. 12.16).

Для того чтобы начать установку этого компонента, следует щелкнуть в окне установки экранную кнопку **Next**. Затем на передний план выводится окно, которое содержит руководство по установке компонента InterBase Client (рис. 12.17).

Затем на передний план выводится окно, которое содержит текст лицензионного соглашения по использованию данного компонента (рис. 12.18).

Подтверждаем согласие с условиями лицензионного соглашения щелчком на экранной кнопке **Yes** (Да). Далее на переднем плане появляется окно, которое содержит следующую информацию (рис. 12.19). В верхней части данного окна содержится список устанавливаемых компонентов. Под ним имеется небольшое окно, в котором указаны логический диск и папка, куда по умолчанию должны быть установлены программные файлы. Список компонентов, а также указанные по умолчанию диск и папку изменять не рекомендуется. Для того чтобы приступить к непосредственной установке набора дополнительных компонентов, следует щелкнуть на экранной кнопке **Install**.

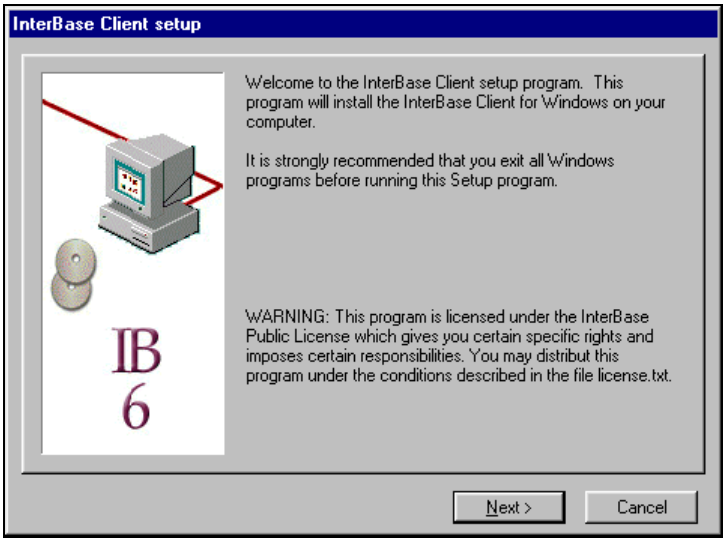


Рис. 12.16. Начальное окно установки компонента InterBase Client

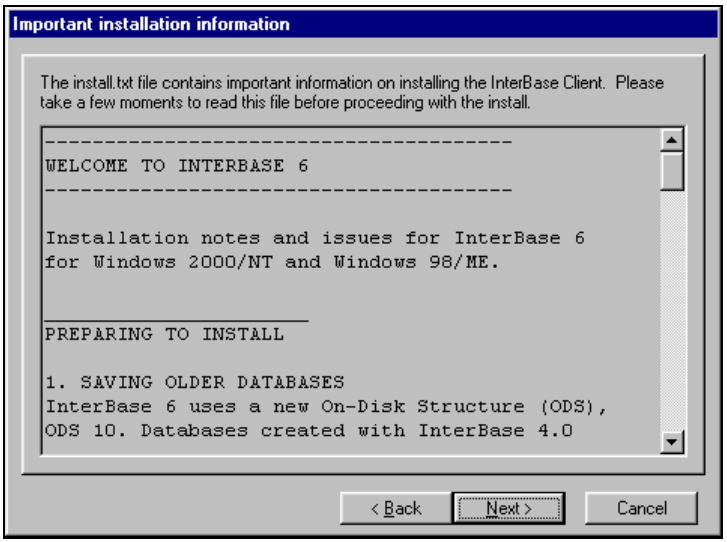


Рис. 12.17. Окно, содержащее руководство по установке компонента InterBase Client

Далее на экране компьютера появляется окно с диаграммой, которая наглядно показывает процесс установки дополнительных компонентов. Когда процесс копирования файлов завершен, необходимо для завершения процесса установки дополнительных компонентов щелкнуть на экранной кнопке **Finish** (Готово) (рис. 12.20).

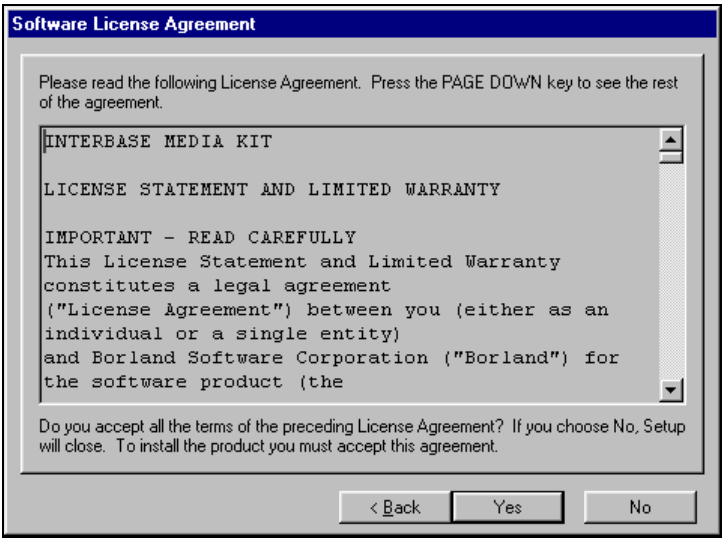


Рис. 12.18. Окно с текстом лицензионного соглашения по использованию дополнительного компонента

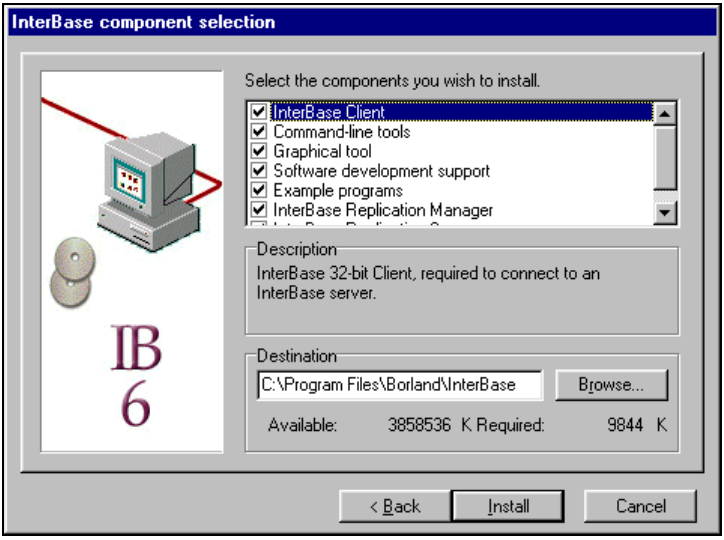


Рис. 12.19. Окно, содержащее информацию о готовности приступить к установке компонента InterBase Client

После щелчка на этой кнопке окно установки InterBase Client закрывается, и на экране компьютера появляется окно мастера установки системы программирования Delphi 6, в котором сообщается, что установка системы успешно

произведена, и для ее корректного завершения необходимо щелкнуть кнопку **Finish** (рис. 12.21).

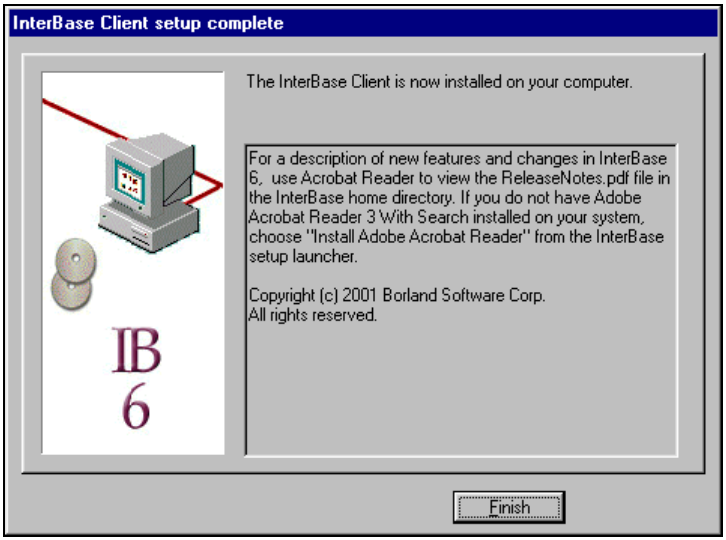


Рис. 12.20. Окно, сообщающее о завершении процесса установки InterBase Client

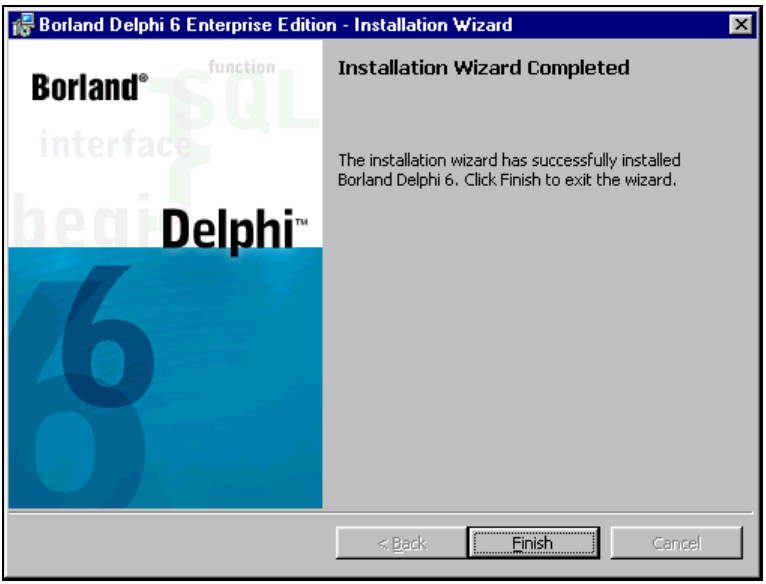


Рис. 12.21. Окно, сообщающее о том, что процесс установки системы программирования Delphi 6 успешно завершен

После этого на экран компьютера выводится окно, которое информирует пользователя о том, что для окончательной установки программы необходимо перезагрузить компьютер. Для подтверждения необходимости перезагрузки нужно щелкнуть экранную кнопку **Yes**, а для отказа от немедленной перезагрузки — щелкнуть экранную кнопку **No**. Рекомендуется сразу произвести перезагрузку, щелкнув кнопку **Yes**. Далее автоматически производится перезагрузка компьютера. По завершении перезагрузки можно извлечь компакт-диск из дисковода, и на этом процесс установки системы программирования Delphi 6.0 на компьютер будет завершен.

Для того чтобы это проверить, достаточно войти в главное меню компьютера, нажав кнопку **Пуск** (Start), открыть раздел **Программы** (Programs) и убедиться, что в данном разделе появились пункты **Borland Delphi** и **InterBase**.

Для русификации имеющейся английской версии нужно вновь запустить программу-оболочку, имеющуюся на компакт-диске, и в ней выбрать кнопку **Русская версия**, а затем установить эту версию в соответствии с имеющимися на диске инструкциями. Однако при дальнейшем изложении основ программирования в системе Delphi мы будем все же опираться на английскую версию системы, как имеющую наиболее широкое распространение среди программистов (в том числе и в нашей стране).

12.3. Запуск среды программирования

Так как система программирования Delphi является приложением операционной системы Windows, то, подобно большинству действий в данной операционной системе, запуск системы программирования можно производить несколькими разными способами. Рассмотрим эти способы (конечно, подразумевается, что система Delphi была правильно инсталлирована на ваш компьютер в соответствии с рекомендациями, изложенными в предыдущем разделе).

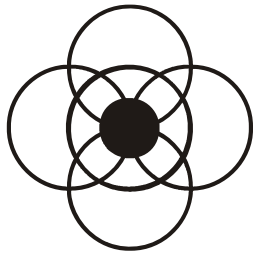
- Нужно открыть папку, содержащую головной файл, имеющий название Delphi32.exe. Для этого необходимо на компьютере найти и открыть папку Program Files (эта папка, как правило, находится в корневом каталоге диска C:), затем найти в этой папке вложенную папку Borland, открыть ее и в ней найти вложенную папку Delphi, а в той в свою очередь открыть папку bin. Эта папка и будет содержать файл Delphi32.exe. Запуск файла производится двойным щелчком на значке файла, если для данного компьютера установлен классический стиль работы с мышью, и одиночным щелчком на значке, если на компьютере используется стиль Web.
- Открыть главное меню операционной системы нажатием клавиши **Пуск**. Найти в главном меню раздел **Программы**, а в этом разделе найти под-

раздел **Borland Delphi**, в котором есть пункт **Delphi**. Щелчок на этом пункте приводит к запуску системы программирования.

- Если на рабочий стол Windows выведен ярлык для файла Delphi32, то систему программирования можно запустить, не открывая главное меню или какие-либо папки. Для этого достаточно два раза щелкнуть на этом ярлыке для обычного стиля работы с мышью или один раз щелкнуть в том случае, если для мыши используется стиль Web.

После выполнения одной из указанных ранее операций на экране компьютера на некоторое время появляется специальная заставка, которая содержит надпись Borland Delphi, а также номер установленной версии Delphi. После успешного завершения процедуры запуска на экране компьютера появляется основной экран системы программирования Delphi. Рассмотрением элементов экрана мы и займемся в следующем разделе.

ГЛАВА 13



Приступаем к программированию в Delphi

Система Delphi постоянно совершенствуется, и поэтому все время появляются ее новые версии. Освоив работу в какой-либо версии, не трудно при желании перейти к более совершенной версии и продолжать в ней работать, т. к. у всех версий очень похожи основные элементы. Изучив интерфейс среды Delphi, можно приступать к написанию первой несложной программы, а после ее отладки и к запуску. Приведенные в этой главе готовые проекты должны помочь обучающемуся при выполнении самостоятельного задания в конце главы.

13.1. Элементы экрана среды Delphi

Внешний вид экрана системы программирования может несколько отличаться от версии к версии, но основные его элементы одинаковы или сходны между собой во всех версиях Delphi. В книге для сравнения приводится внешний вид основного экрана для пятой версии (рис. 13.1), шестой (рис. 13.2) и седьмой версии (рис. 13.3) среды Delphi.

Во всех этих версиях содержатся следующие элементы экрана (см. рис. 13.3):

- строка заголовка;
- главное меню системы программирования (строка меню);
- панели инструментов;
- палитра компонентов;
- окно инспектора объектов;
- окно формы;
- окно кода.

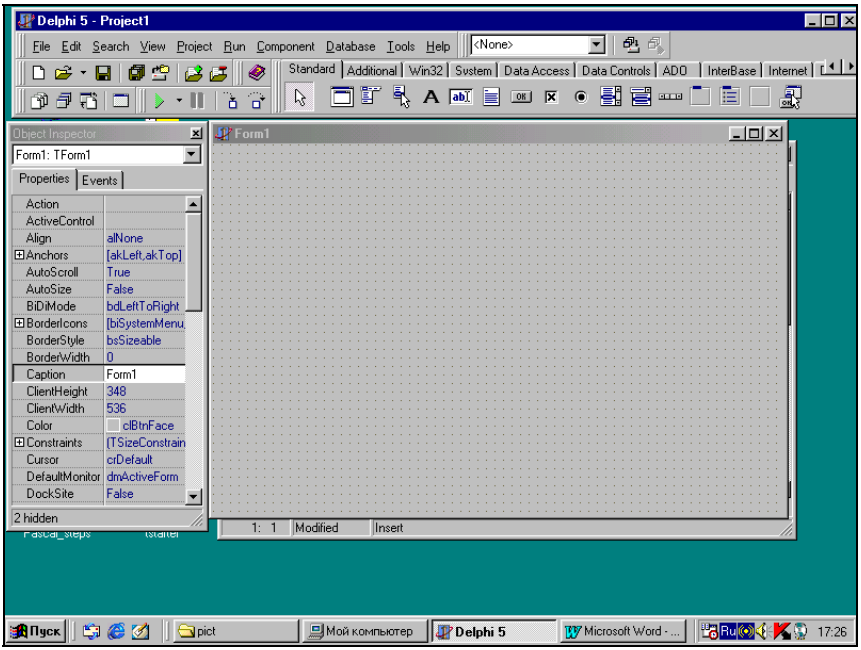


Рис. 13.1. Экран среды программирования Delphi 5

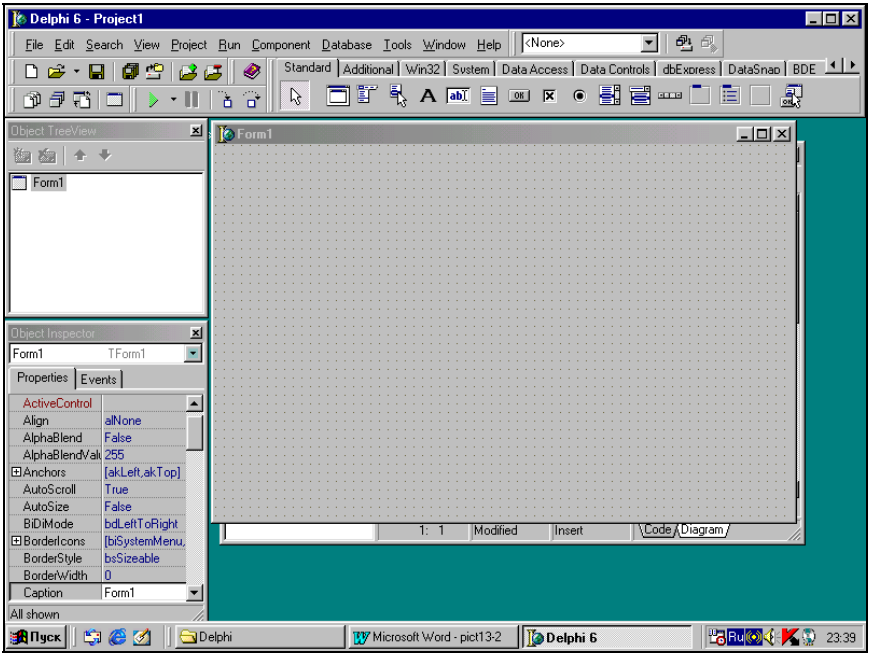


Рис. 13.2. Экран среды программирования Delphi 6

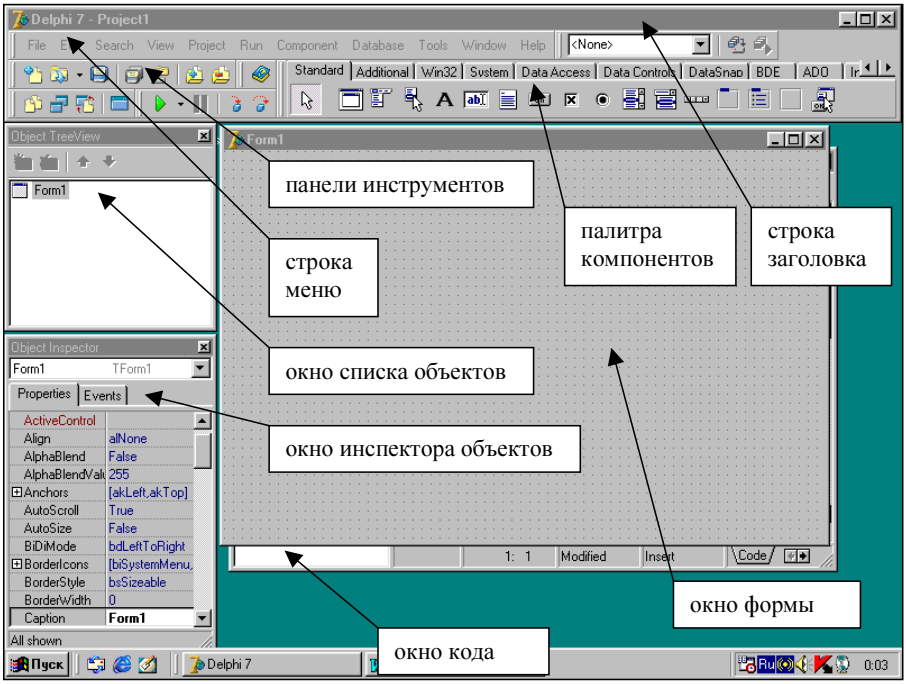


Рис. 13.3. Экран среды программирования Delphi 7

В шестой и седьмой версиях имеется еще одно дополнительное окно со списком объектов. В пятой версии вместо него можно использовать раскрывающийся список, имеющийся в инспекторе объектов.

В *строке заголовка* содержится номер установленной на данном компьютере версии системы Delphi и название открытого в данный момент проекта. *Проектом* в Delphi называется программа, находящаяся в стадии разработки. Вновь открываемый проект по умолчанию называется Project1. Это имя мы, скорее всего, и увидим в строке заголовка после загрузки системы программирования.

Под строкой заголовка находится *строка главного меню* системы. Эта строка содержит ряд разделов, каждый из которых представляет собой ниспадающее меню. Для того чтобы открыть ниспадающее меню, достаточно щелкнуть мышью на названии соответствующего раздела. Каждое такое ниспадающее меню содержит ряд команд. Если справа от названия команды стоит многоточие, то при выполнении этой команды открывается диалоговое окно. Если название команды закрашено серым цветом, то эта команда в данный момент времени невыполнима. Названия же тех команд, которые можно в данный момент использовать, написаны символами черного цвета. Если справа от названия команды стоит треугольная стрелка, то это означает, что данная

команда содержит еще одно вложенное меню, в котором можно выбрать один из предложенных вариантов.

Например, если сразу после того, как будет установлен основной экран, щелкнуть раздел меню **File** (Файл), содержащий команды для работы с файлами в системе программирования, то в нем все команды написаны черным цветом, т. е. доступны для выполнения. Справа от пункта **Save as** (Сохранить как) находится многоточие, означающее, что в ходе выполнения команды появится диалоговое окно, которое позволит сохранить файл в указанной пользователем папке. Справа же от пункта **New** (Новый — создание нового элемента проекта) находится треугольная стрелка, которая показывает, что при выполнении этой команды откроется меню, содержащее перечень тех элементов (приложение, форма, программный модуль и т. д.), которые можно добавить в программу.

Если же открыть другой раздел меню **Edit** (Редактировать — он содержит команды, используемые для редактирования текста проекта), то первоначально в этом разделе ряд команд будет закрашен серым цветом. Это команды, используемые для работы с выделенным фрагментом текста. Так как изначально ни один фрагмент текста не выделен, то использование этих команд лишено смысла. Когда в процессе работы над программой будет выделен какой-либо фрагмент текста, то команды **Cut** (вырезать выделенный фрагмент текста в буфер обмена), **Copy** (скопировать фрагмент в буфер обмена), **Paste** (вставить фрагмент в то место программы, на котором стоит курсор), **Delete** (удалить выделенный фрагмент) станут доступными и будут закрашены черным цветом.

Под строкой меню находятся *панели инструментов*, каждая из которых представляет собой набор экранных кнопок. На каждой кнопке имеется пиктограмма, т. е. небольшой рисунок, который поясняет назначение данной кнопки. Кроме того, если навести на кнопку указатель мыши и задержать его на несколько секунд, то рядом с кнопкой появляется всплывающая подсказка, которая кратко поясняет назначение кнопки в словесной форме. В нерусифицированных версиях Delphi эта подсказка выводится на английском языке. Большинство кнопок панелей инструментов дублируют наиболее часто употребляемые команды меню.

Справа от панелей инструментов находится *палитра компонентов*, которая содержит большое количество вкладок. Каждая из вкладок также фактически является панелью инструментов, состоящей из ряда экранных кнопок. Каждая из этих кнопок представляет собой заготовку для будущих экранных объектов создаваемого проекта. Переключение между вкладками производится щелчком на корешке соответствующей вкладки.

Ниже расположено окно формы. *Форма* представляет собой заготовку для окна будущей программы. Именно на форме в процессе работы над проектом

размещаются различные объекты, переносимые с вкладок панели компонентов. Эти объекты в совокупности составляют пользовательский интерфейс, т. е. средство взаимодействия между пользователем и программой. На форме имеется специальная координатная сетка, которая используется для более точного расположения объектов. При запуске программы на выполнение координатная сетка исчезает, и фон формы становится однородным.

Слева от формы находится окно, содержащее *список всех объектов*, используемых во время работы над данным проектом. Данный список представлен в форме древовидной структуры наподобие дерева файлов в программе Проводник, являющейся стандартным приложением ОС Windows. Если какой-либо объект содержит другие, вложенные в него объекты, то слева от значка объекта имеется квадратик со знаком "плюс". Для того чтобы узнать, какие это вложенные объекты, достаточно щелкнуть мышью на квадратике. После этого соответствующие объекты будут выведены в окне, а "плюс" изменится на "минус". Для того чтобы "спрятать" вложенные объекты, достаточно снова щелкнуть на квадратике.

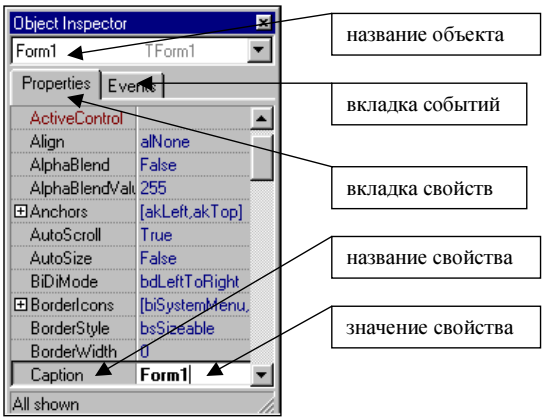


Рис. 13.4. Окно инспектора объектов

Если же щелкнуть мышью на значке самого объекта, то он будет выделен синим цветом, а в расположенном ниже окне *инспектора объектов* можно будет ознакомиться с его свойствами. Окно инспектора объектов (рис. 13.4) содержит две вкладки. На одной из них, имеющей название **Properties** (Свойства), содержится перечень свойств, присущих данному объекту. Большинство свойств объекта можно изменять непосредственно в этом окне либо путем ввода соответствующего текста или числового значения, либо выбором нужного значения из раскрывающегося списка. На другой вкладке, имеющей название **Events** (События), содержится перечень событий, на которые может реагировать данный объект. Для того чтобы запрограммировать реакцию

объекта на произошедшее событие, как правило, нужно написать соответствующий программный код, отображаемый в окне кода.

Окно кода обычно не видно целиком на экране системы, т. к. большая его часть закрыта окном формы, и пользователь видит только нижнюю строку этого окна. Для того чтобы активизировать окно кода (т. е. вывести его на передний план), нужно щелкнуть на любом видном пользователю его фрагменте, либо, если окно закрыто целиком, использовать команды из меню **View**. Данное окно содержит исходный текст программы (называемый также исходным кодом), написанный на языке Object Pascal.

Элементы экрана системы Delphi расположены таким образом, что они не закрывают основные элементы управления ОС Windows. Поэтому в ходе работы всегда можно воспользоваться главным меню кнопки **Пуск** или панелью задач, например, для того, чтобы переключаться между Delphi и другими открытыми пользователем во время работы за компьютером приложениями, щелкая мышью соответствующие кнопки на панели задач.

13.2. Первая программа на Delphi

Как это ни странно, создание первой программы в системе Delphi начнем с того, что сохраним программу, хотя на первый взгляд кажется, что программы как таковой еще не существует. Однако, как только пользователь войдет в систему программирования, он может убедиться, что существуют не только заготовки для экранных объектов в виде кнопок палитры компонентов, но и заготовка самой программы. Для этого достаточно активизировать окно кода так, как это было сказано в предыдущем разделе. Когда это окно выйдет на передний план, вы увидите, что в данном окне уже имеются определенные строки, содержащие текст программы. Эти строки автоматически генерируются системой программирования. В дальнейшем пользователь сам сможет по мере необходимости добавлять в окне строки, дополняющие эту основу. При этом необходимо иметь в виду, что при разработке проекта система создает целую группу связанных между собой программных файлов.

Поэтому рекомендуется каждый вновь создаваемый пользователем проект сохранять в отдельной папке с тем, чтобы программист не путал файлы, относящиеся к разным проектам. В качестве места, где будут храниться все создаваемые в процессе программирования папки, лучше всего взять папку, уже имеющуюся в системе программирования. Эта папка так и называется Projects и расположена по следующему адресу (предполагается, что система программирования правильно инсталлирована и расположена на диске C:): C\Program Files\Borland\Delphi. В данной папке и будут создаваться вложенные папки, сохраняющие файлы разрабатываемых пользователем проектов.

Рассмотрим конкретные шаги, которые необходимы для начального сохранения проекта.

Прежде всего необходимо открыть раздел **File** (Файл) главного меню и в нем найти команду **Save as** (Сохранить как). При выполнении этой команды откроется диалоговое окно **Save Unit1 as** (сохранение модуля 1) (рис. 13.5). В данном окне будет отображаться содержимое папки Projects. Для того чтобы создать новую папку внутри папки Projects, следует щелкнуть кнопку  на панели инструментов данного окна. Если пользователь не может сразу обнаружить эту кнопку, то следует поочередно подводить указатель мыши ко всем кнопкам, пока на одной из них не появится всплывающая подсказка "Создание новой папки". После щелчка по этой кнопке в окне появится папка, которая так и будет называться "Новая папка".

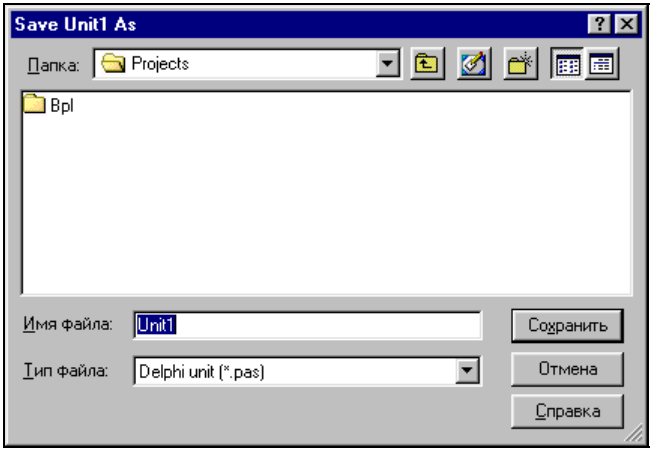


Рис. 13.5. Диалоговое окно сохранения файла исходного текста программы

Понятно, что такое название неинформативно, т. к. ничего не говорит о содержании данной папки. Поэтому рекомендуется дать ей какое-либо осмысленное название. Например, папку с проектом вашей первой программы можно назвать First (по-английски это слово и означает "первый"). Для того чтобы переименовать вновь созданную папку, следует нажать на клавиатуре клавишу <Delete>, в результате чего исчезнет название "Новая папка", и ввести с клавиатуры подходящее название, а затем нажать клавишу <Enter>. Затем нужно щелкнуть экранную кнопку **Открыть**, имеющуюся в диалоговом окне. При этом вновь созданная папка будет открыта и в окне будет видно, что папка пустая. Если щелкнуть на кнопке **Сохранить**, то в этой папке будет сохранен файл Unit1, содержащий исходный текст разрабатываемой вами программы. Если пользователь допустил какую-либо ошибку (например, от-

крыл не ту папку), то можно закрыть окно без сохранения изменений, щелкнув кнопку **Отмена**.

Следующим шагом должно стать сохранение файла Project1, в котором будет содержаться откомпилированная и готовая к выполнению программа. Для этого следует вновь войти в меню **File** и использовать команду **Save Project as**. В результате выполнения этой команды будет открыта вложенная папка (в нашем случае это папка First) и для сохранения файла проекта останется только щелкнуть экранную кнопку **Сохранить**. Окно сохранения проекта показано на рис. 13.6. Остальные связанные с проектом файлы будут сохраняться автоматически. В дальнейшем изменения в проекте рекомендуется сохранять после добавления и настройки каждого нового объекта, но повторять каждый раз всю изложенную ранее последовательность действий нет необходимости. Достаточно дать команду **Save all** все из того же меню **File**, и все нужные изменения будут внесены в соответствующие файлы автоматически.

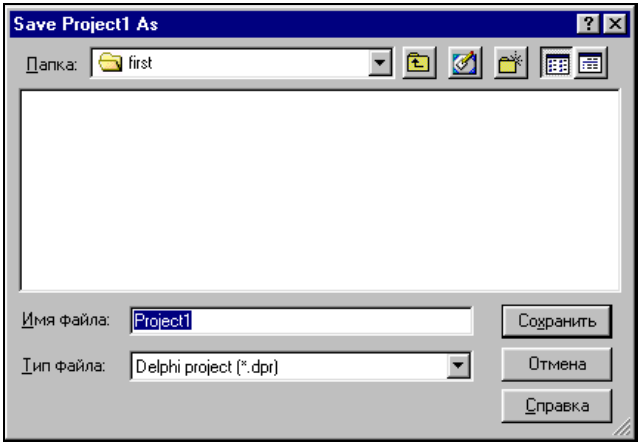


Рис. 13.6. Диалоговое окно сохранения файла проекта

Теперь можно приступать к непосредственной разработке проекта. Создание программы включает в себя два взаимосвязанных компонента. Первым из них является *создание графического интерфейса пользователя*, который будет представлен в виде различных объектов и элементов управления, расположенных на экране программы. К ним относятся:

- выводимые на экран компьютера надписи;
- экранные кнопки, щелкая мышью на которых можно производить различные действия;
- текстовые окна, используемые для ввода текста или чисел в ходе работы программы;

- различные списки и меню;
- группы флажков и переключателей, используемые для выбора одного из вариантов работы программы;
- панели и рисунки, используемые для оформления экрана;
- другие элементы.

Вторым компонентом, который необходим для разработки приложения, является *программный код*, описывающий реакцию имеющихся на экране объектов и элементов управления на различные события, которые могут происходить в ходе выполнения программы. Реакция на событие описывается в виде соответствующей подпрограммы, входящей в основной программный модуль. Подобно заготовкам для экранных объектов, текстовое окно, содержащее исходный код программы, также имеет заготовки для подпрограмм. В частности, по правилам языка Object Pascal каждая подпрограмма должна иметь заголовок, начинающийся со служебного слова `procedure`, в начале основной части подпрограммы должно содержаться служебное слово `begin`, а завершаться она должна служебным словом `end`, после которого ставится точка с запятой. Все эти слова вам не нужно будет вводить вручную, т. к. они уже имеются в автоматически создаваемой для них заготовке.

После того как эти два компонента будут созданы, можно проверить работоспособность программы, запустив ее на выполнение. Перед запуском программы рекомендуется ее сохранить, используя для этого команду **File | Save all**. Если в программе обнаружатся ошибки, то текстовая строка, их содержащая, будет выделена другим цветом, что облегчит пользователю поиск ошибок и их исправление. После завершения "работы над ошибками" программист получит готовую к выполнению, работоспособную программу.

Правда при этом нужно иметь в виду, что автоматически система находит только грамматические ошибки, т. е. строки, написанные с нарушением правил языка Object Pascal. Помимо них в программе могут содержаться и семантические, т. е. смысловые ошибки. Поиск семантических ошибок составляет наиболее трудную часть работы программиста. Но и эти трудности могут быть преодолены и конечным итогом работы станет программный продукт, который будет не только правильно и грамотно работать, но и будет обеспечивать для пользователя дружественный интерфейс (т. е. будет прост и удобен в использовании) и будет удовлетворять эстетическим запросам потребителя.

Первая программа, которую мы создадим в системе Delphi, будет производить следующие действия: выводить на экран компьютера надпись "Моя первая программа", при щелчке мышью на одной из экранных кнопок выводить на экран компьютера сведения об авторе программы и при щелчке на другой кнопке закрывать программу.

Создание графического интерфейса программы начнем с создания надписи. Перенос любого элемента графического интерфейса в форму можно производить двумя способами.

- Дважды щелкнуть мышью на соответствующей кнопке на панели компонентов. В результате выбранный элемент интерфейса разместится посередине формы.
- Один раз щелкнуть мышью на кнопке на панели компонентов и затем щелкнуть мышью в любом месте формы. В результате выбранный элемент интерфейса появится в указанном месте. Этот способ обладает тем преимуществом, что программист может сам определять, где размещать выбранный элемент.

В данном случае элемент, который мы выбираем, находится в палитре компонентов (она занимает всю правую часть, отведенную на экране под панели инструментов) на вкладке **Standard**, которая расположена на данной панели первой слева. Этим элементом является надпись, которая по-английски называется `Label`. На соответствующей кнопке имеется пиктограмма в виде большой буквы A —  .

Далее, нужно перенести надпись в форму любым из двух описанных ранее способов. В результате в форме вы увидите надпись под названием `Label1`. Затем пользователь должен отрегулировать положение надписи в форме (она должна находиться в верхней ее части) и ее размер. Содержанием данной надписи должна быть фраза "Моя первая программа". Как изменить содержание надписи, мы разберем чуть позже, а сейчас необходимо научиться регулировать положение и размеры надписи. Наиболее простой, хотя и не очень точный способ достижения поставленной задачи заключается в использовании мыши. Перемещать надпись по форме проще всего методом перетаскивания. Для этого нужно навести указатель мыши на центральную часть созданной надписи и нажать левую кнопку мыши. Затем, не отпуская кнопку, нужно переместить указатель мыши в нужное место формы. Вслед за указателем мыши переместится и надпись. Далее нужно отпустить кнопку мыши и надпись останется на выбранном пользователем месте.

Для изменения же размеров надписи используем следующий прием. Наведем указатель мыши на надпись и установим указатель на любую из границ надписи. Границы надписи отмечены специальными черными квадратиками, которые называются маркерами. На один из маркеров и нужно поместить указатель мыши. При этом указатель мыши изменит свою форму: он приобретет вид двунаправленной стрелки. Затем нужно нажать левую кнопку мыши и, не отпуская ее, переместить указатель мыши. Вслед за указателем переместится и граница надписи. Подобный прием, часто используемый для изменения размеров экранных объектов, называется *протягиванием*. Размеры

объекта (в данном случае объектом является надпись) можно изменять по трем направлениям: по горизонтали, по вертикали и по диагонали. В последнем случае размеры надписи будут изменяться одновременно по горизонтали и по вертикали. Для изменения вертикальных размеров нужно поместить указатель мыши на верхний или на нижний маркер, для изменения размеров по горизонтали указатель помещаем на один из боковых маркеров, а для изменения диагональных размеров используем любой из угловых маркеров.

В принципе, размеры надписи можно было бы и не изменять вручную, т. к. они впоследствии автоматически изменятся в соответствии с содержанием надписи, но надпись является хорошим объектом для тренировки. Поупражняйтесь с изменением размеров надписи и полученные навыки в дальнейшем пригодятся вам при работе с другими объектами, размеры которых не изменяются автоматически в зависимости от их содержания.

Следующей задачей, которую необходимо решить при разработке проекта, является изменение содержания и шрифта надписи. Для этого нужно, чтобы в окне инспектора объектов был выбран объект `Label1`. Соответствующая надпись должна быть в текстовом поле, расположенном в верхней части окна инспектора объектов. В инспекторе объектов может отображаться и другой объект — сама форма, имеющая имя `Form1`. Это возможно, например, в том случае, если пользователь, настраивая размеры надписи, случайно щелкнул мышью не на ней, а на любом свободном месте формы. Исправить такое положение очень просто — для этого достаточно один раз щелкнуть мышью на надписи, расположенной в форме.

После того, как вы убедились, что в инспекторе объектов выделена именно надпись, нужно выбрать в инспекторе вкладку **Properties** (Свойства). На этой вкладке в алфавитном порядке перечислены все свойства обрабатываемого объекта. Каждое свойство в этом списке представлено в виде двух полей. В левом поле черным или коричневым цветом написано название свойства, в правом поле синим цветом указано значение, которое имеет в данный момент это свойство. Нас в данном случае интересует свойство `Caption`, т. е. содержание надписи, называемое также ее заголовком. По умолчанию свойство `Caption` имеет значение `Label1`, которое и отображается в поле, находящемся справа от названия (по умолчанию объект получает такой же заголовок, как и его имя).

Для того чтобы изменить значение этого свойства, нужно вначале удалить старое значение `Label1`. Для чего нужно один раз щелкнуть мышью в правом поле. В результате в поле появится вертикальная черта, называемая курсором. Напомним, что для удаления символа, расположенного слева от курсора, нужно нажать на клавиатуре клавишу <Backspace>, а для удаления символа, который находится справа от курсора, следует нажать клавишу <Delete>. По-

сле того, как правое поле очищено, вводим новое значение поля — фразу "Моя первая программа". В результате соответствующая фраза появится и в надписи, расположенной на форме.

При этом имя надписи `Label1` останется неизменным, т. к. имя надписи и ее заголовок — разные свойства объекта. В этом нетрудно убедиться, увидев значение свойства `Name` (имя). Для того чтобы найти это свойство в инспекторе объектов, нужно использовать скроллер (полосу прокрутки), расположенную в правой части инспектора объектов. Прокручивать же содержимое инспектора объектов удобнее всего, перетаскивая мышью расположенный на полосе прокрутки серый прямоугольник, называемый лифтом. Найдя свойство `Name`, убедимся, что справа от названия по-прежнему указано значение `Label1`. На начальном этапе работы с системой имена объектов вообще лучше не изменять.

Взглянув на надпись, имеющуюся в форме, нетрудно убедиться, что выглядит она весьма неказисто. А ведь именно эта надпись будет основным элементом интерфейса нашей первой программы. Для того чтобы улучшить внешний вид надписи, следует использовать ее свойство, называемое `Font` (шрифт). Для того чтобы найти это свойство в окне инспектора объектов, мы используем уже знакомую нам полосу прокрутки с лифтом. Когда мы найдем его, то увидим, что слева от названия свойства находится квадратик с плюсом. Это означает, что свойство является не простым, а составным, т. е. оно включает в себя ряд подсвойств.

Для того чтобы ознакомиться со списком этих подсвойств, можно щелкнуть на квадратике, и ниже свойства `Font` в инспекторе объектов вы увидите соответствующий список. Конечно, для того чтобы изменить внешний вид шрифта, можно изменить значения ряда подсвойств, аналогично тому, как мы это делали со свойством `Caption`. Но в данном случае мы воспользуемся более простым и наглядным способом настройки. Если щелкнуть один раз мышью в поле значений свойства `Font`, то в этом поле появится кнопка, на которой изображено многоточие. Это многоточие имеет то же значение, что и соответствующий элемент в главном меню системы. Если щелкнуть на этой кнопке  , называемой также построителем, то откроется диалоговое окно, в котором можно изменять основные характеристики шрифта. Внешний вид данного окна показан на рис. 13.7. Подобное окно хорошо знакомо пользователям, которые работали с программой Microsoft Word или другими приложениями ОС Windows. Удобство работы с этим окном заключается еще и в том, что в отличие от большинства элементов интерфейса Delphi оно русифицировано.

В данном случае мы изменим следующие настройки. Во-первых, мы увеличим размер шрифта с 8 (по умолчанию) до 12. Во-вторых, мы изменим цвет

шрифта — заменим используемый по умолчанию черный цвет на выбранный из раскрывающегося списка темно-синий. Помимо указанных настроек при желании можно изменить гарнитуру шрифта. Например, вместо используемого по умолчанию шрифта MS Sans Serif можно поставить шрифт Arial, Times New Roman или любой другой из числа установленных на вашем компьютере. Кроме того, можно изменить начертание шрифта, сделав его наклонным (курсив) или полужирным. Наконец, в случае необходимости, щелкнув мышью соответствующий флажок, можно сделать текст подчеркнутым. В центре диалогового окна находится окошко, называемое **Образец**, в котором будут отображаться внешний вид текста надписи и все сделанные пользователем изменения. В случае если пользователь удовлетворен результатами своей работы, он может сохранить произведенные настройки, щелкнув на экранной кнопке **OK**. После закрытия диалогового окна все сделанные пользователем настройки вступят в силу, и можно будет увидеть, как преобразовалась надпись.

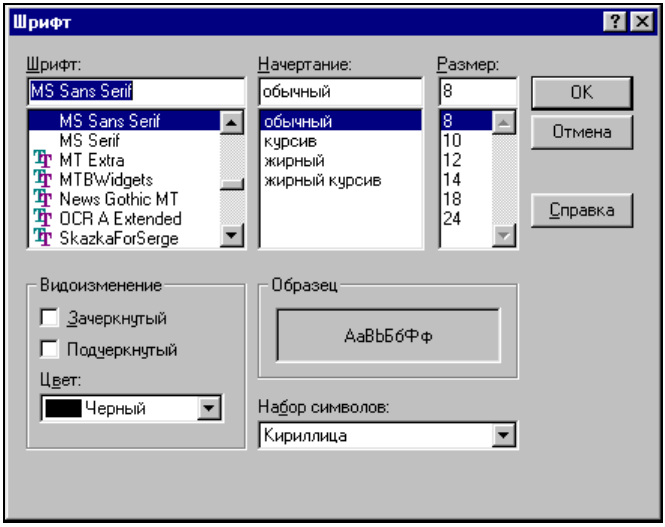


Рис. 13.7. Диалоговое окно настройки шрифта

Следующим элементом интерфейса, который мы поместим в форму, будет закрывающая кнопка. Строго говоря, данный элемент не является для программы обязательным, т. к. созданную нами программу, как и любое приложение Windows, можно закрыть с помощью закрывающей кнопки [×], которая находится в правом углу строки заголовка. Однако для удобства пользователя принято, чтобы в программе имелся и другой способ ее закрытия (обычно с помощью экранной кнопки или пункта меню). Для того чтобы поместить на форму экранную кнопку, следует вновь воспользоваться вкладкой

Standard палитры компонентов. Искомый элемент имеет английское название `Button` и выглядит следующим образом —  . После выбора заготовки данного объекта мы поместим его в правой нижней части формы. По умолчанию на кнопке будет выведен заголовок `Button1`.

Желательно, чтобы заголовок кнопки соответствовал производимому ею действию. Сделать это можно, как и в случае с объектом `Label1`, изменив значение свойства `Caption` (такое свойство есть и у экранных кнопок). Снова вводим текст надписи — "Закрыть программу". Если внешний вид текста пользователя не удовлетворяет, то, как и в случае с надписью, можно отрегулировать свойство `Font`, которое имеет точно такую же внутреннюю структуру, что и у надписи. Единственное отличие в настройке свойств этого объекта заключается в том, что кнопка не "растягивается" автоматически в зависимости от величины заголовка. Поэтому размеры экранной кнопки следует отрегулировать путем перетаскивания маркеров. Теперь экранная кнопка приобрела желаемый внешний вид, а полученный в результате указанных ранее действий интерфейс программы приведен на рис. 13.8.

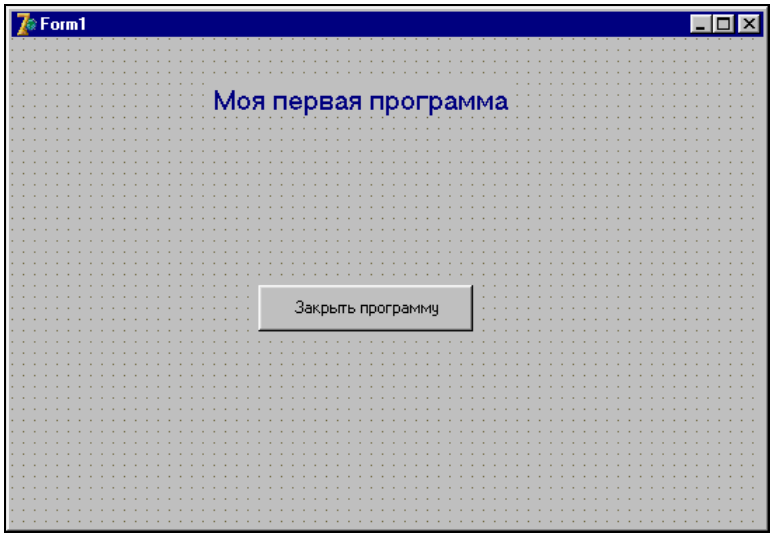


Рис. 13.8. Интерфейс первой программы на Delphi

Если на данном этапе работы запустить программу на выполнение, то сколько мы ни будем щелкать мышью на кнопке или нажимать клавишу <Enter>, никакой реакции на наши действия не будет. Для того чтобы такую реакцию запрограммировать, нужно написать соответствующий программный код. Сам по себе код, закрывающий программу, крайне прост. Он состоит из одного-единственного слова `Close`, которое в переводе с английского и означает

"закрыть". Вопрос состоит в том, что нужно знать в каком месте программы его нужно вставить. Для того чтобы облегчить себе работу, воспользуемся уже знакомым нам окном инспектора объектов.

Выделим в окне формы объект `Button1` и в инспекторе объектов перейдем на вкладку **Events** (События). В качестве события, на которое должна реагировать наша кнопка, выберем щелчок мышью (по-английски — "click"). Если мы обратимся к инспектору объектов, то увидим, что там этому событию соответствует пункт `OnClick`. Если щелкнуть два раза мышью на поле, находящемся справа от события `OnClick`, то перед нами откроется окно кода с уже имеющейся заготовкой процедуры, описывающей это событие. Между служебными словами процедуры `begin` и `end` вставляем требуемое слово `close`. Внешний вид окна кода программы приведен на рис. 13.9.

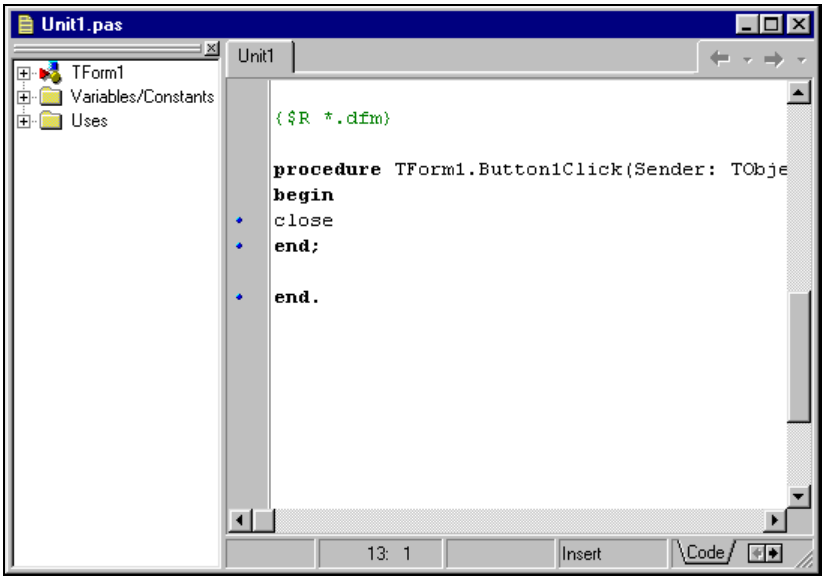


Рис. 13.9. Окно кода программы на Delphi

В данном окне должен быть виден следующий текст процедуры `Button1Click`, описывающей реакцию на щелчок мышью на кнопке **Закрыть программу**:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
close;
end;
```

Как видно из приведенного текста, заголовок процедуры и служебные слова система программирования вставляет автоматически. Для того чтобы убе-

диться, что созданная программа работоспособна, можно произвести запуск программы на выполнение. Запуск программы можно выполнить тремя способами.

- Через главное меню системы программирования. В меню нужно найти раздел **Run**. Открыв этот раздел, найти в нем пункт, который тоже называется **Run**, и щелкнуть его мышью.
- С помощью клавиатуры. Нажать на клавиатуре компьютера клавишу <F9>.
- С помощью панели инструментов системы программирования. Найти кнопку запуска программы в виде треугольной стрелки  и щелкнуть ее мышью.

После запуска программы на выполнение любым из указанных способов произойдет следующее: на переднем плане экрана появится окно созданной вами программы. О том, что это именно окно работающей программы, а не заготовка программы в виде формы, можно догадаться по отсутствию координатной сетки. Эту программу можно закрыть теперь двумя способами: либо через закрывающую кнопку в строке заголовка, либо посредством созданной нами кнопки в нижней части окна. После закрытия программы мы вновь окажемся в системе программирования Delphi.

Несмотря на то, что программа уже работоспособна, работа над ней еще не закончена. Интерфейс программы должен включать в себя еще один элемент — экранную кнопку, при щелчке на которой на экран будут выводиться сведения о создателе программы. Такие сведения о себе сообщает любой уважающий себя программист. Вторая экранная кнопка создается и настраивается точно так же, как и первая, только называться она будет не `Button1`, а `Button2`. Содержимое же заголовка, помещаемого на этой кнопке, будет "Сведения об авторе". Разместим мы эту кнопку в левом нижнем углу формы. На этом создание графического интерфейса данной программы завершено. Получившийся в результате описанных действий интерфейс показан на рис. 13.10.

Далее мы должны составить программный код, описывающий действие, происходящее при щелчке мышью на кнопке `Button2`. Действие же это будет заключаться в следующем. При щелчке на кнопке **Сведения об авторе** на экране вместо слов "Моя первая программа" мы увидим надпись "Программу составил..." (вместо многоточия составитель программы может поставить свое имя и фамилию). Для этого при щелчке на экранной кнопке должно изменяться свойство надписи `Caption`. Когда мы в программе ссылаемся на свойства какого-либо объекта, то эта ссылка должна состоять из двух частей: имени объекта и названия свойства. Между этими двумя частями ставится точка. Поэтому заголовок нашей надписи будет описываться как

`Label1.Caption`. Изменение свойства осуществляется путем присваивания ему нового значения. Для этого используется оператор присваивания языка Паскаль, который состоит из двоеточия и знака равенства. Если присваиваемое значение является текстом, как в нашем случае, то оно должно быть обязательно заключено в кавычки. Таким образом, код, изменяющий содержание надписи, будет выглядеть так:

```
Label1.Caption:='Программу составил ...'
```

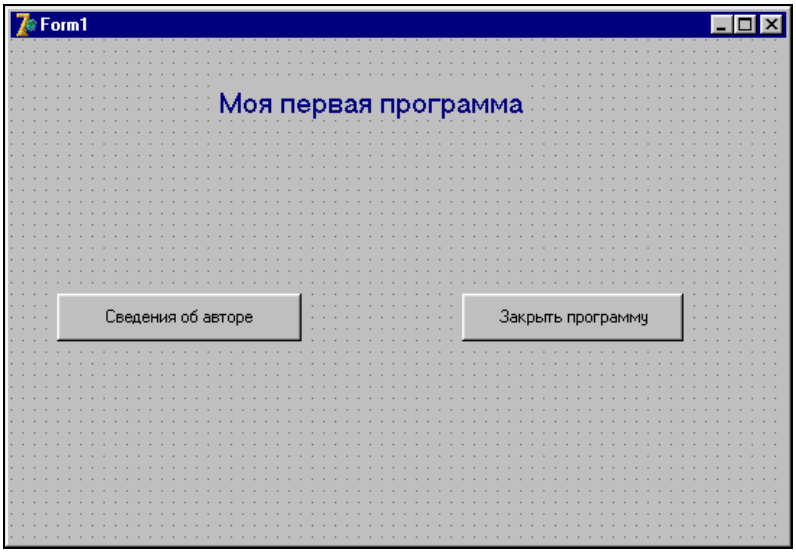


Рис. 13.10. Усовершенствованный интерфейс первой программы

Для того чтобы вставить эту строчку в нужное место программы, следует выделить объект `Button2`, перейти на вкладку **Events** и дважды щелкнуть поле справа от пункта `OnClick`. В открывшуюся заготовку процедуры `Button2.Click` следует ввести приведенную строку. Обратим внимание на следующее обстоятельство: как только мы вводим слово `Label1` и ставим после него точку, появляется список слов, которые можно ввести после этой точки. Для того чтобы выбрать какое-либо слово из списка, достаточно два раза щелкнуть его мышью. В результате оно появится в текстовой строке. Эта возможность автоматизации ввода текста особенно удобна в том случае, когда пользователь сомневается в правильности написания того или иного термина. Если воспользоваться указанной возможностью, то правильное с орфографической точки зрения слово подставит в текст сама система программирования. В листинге 13.1 приводится полный текст программы, который должен находиться в окне кода.

Листинг 13.1. Текст первой программы в Delphi

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls;

type
  TForm1 = class(TForm)
    Label1: TLabel;
    Button1: TButton;
    Button2: TButton;
        procedure Button1Click(Sender: TObject);
       procedure Button2Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.Button1Click(Sender: TObject);
begin
close;
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
Label1.Caption:='программу составил А.Н. Маслобоев';
end;

end.
```

Просмотрев приведенный текст, нетрудно убедиться, что львиную долю работы по написанию программного кода берет на себя сама система програм-

мирования. Программистом собственноручно были написаны здесь только две строчки.

После произведенного нами усовершенствования программы вновь запускаем ее на выполнение одним из трех указанных способов. Напомним, что перед запуском программы на выполнение рекомендуется сохранить все сделанные настройки и дополнения с помощью команды **Save all**. Внешний вид программы во время ее работы приведен на рис. 13.11. Убедившись в том, что программа работает так, как положено, можно закрыть систему программирования. Выход из системы производится либо щелчком на закрывающей кнопке в строке заголовка системы, либо с помощью главного меню. В последнем случае нужно найти раздел меню **File** и в нем команду **Exit**. Если перед выходом из системы вы забыли сохранить изменения в проекте, то на экран компьютера автоматически выводится диалоговое окно, в котором пользователю предлагается либо сохранить изменения в проекте, либо отказаться от изменений. Если же пользователь нажимает в диалоговом окне экранную кнопку **Cancel**, то выхода не происходит, и пользователь возвращается в систему программирования.

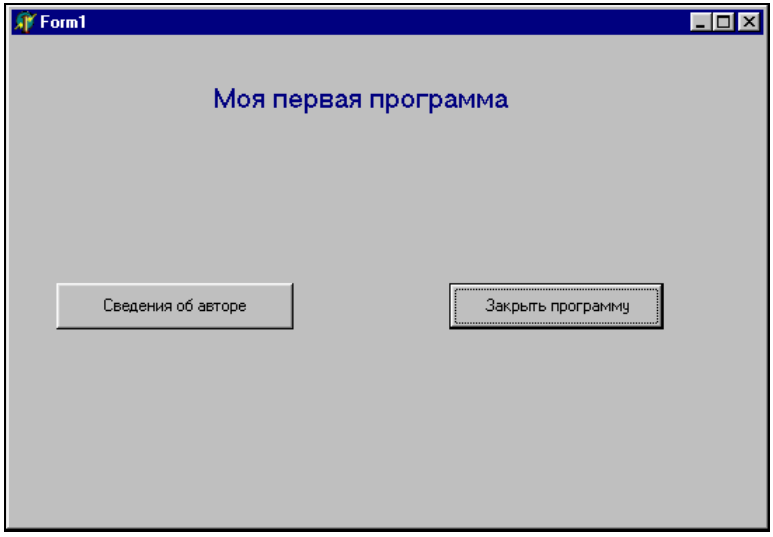


Рис. 13.11. Первая программа во время работы

Если проект был корректно сохранен, то созданный системой исполняемый файл будет работать и автономно от системы. Для того чтобы убедиться в этом, достаточно открыть папку проекта, не запуская систему Delphi, найти в ней исполняемый файл (в пятой и шестой версиях Delphi значок этого файла выглядит так —  , а в седьмой версии — ) и запустить его на выпол-

нение двойным щелчком мыши. В результате программа запустится и будет работать должным образом.

13.3. Программа "Редактор"

Следующий проект, который мы будем разрабатывать в системе Delphi, отличается тем, что пользователь получит возможность самостоятельно изменять содержимое надписи, расположенной в окне программы. Для этого в окне формы должно быть предусмотрено специальное текстовое окно, в котором пользователь может вводить любой текст. Соответствующий текст будет отображаться в виде надписи, расположенной в верхней части окна программы.

Работу над проектом, как и в прошлый раз, начнем с его сохранения. Для этого в папке C:\Program Files\Borland\Delphi\Projects создадим вложенную папку Redaktor и сохраним файлы будущего проекта аналогично тому, как мы это делали в предыдущем разделе.

Как обычно, составление самой программы мы начнем с создания для нее интерфейса. Интерфейс данной программы будет включать в себя следующие элементы:

- две надписи. Одна из них — это основная надпись, значение которой можно изменять путем ввода нового текста в текстовое окно. Вторая надпись вспомогательного характера содержит пояснение к текстовому окну;
- текстовое окно. Введенный в него текст будет дублироваться в укрупненном виде в надписи;
- две экранные кнопки. Одна из них закрывает программу. Вторая выводит сведения об авторе;
- сама форма, в которую мы помещаем все перечисленные компоненты.

Настройку элементов интерфейса мы начнем с формы. Форма является таким же объектом программы, как и все прочие ее компоненты, и имеет ряд свойств, которые можно изменять при ее настройке. Для того чтобы начать работу с формой, достаточно щелкнуть мышью на любом ее свободном месте. При этом в окне инспектора объектов будет отображаться объект по имени `Form1`, т. е. сама основная форма. В окне инспектора выделяем вкладку **Properties** (Свойства) и находим на этой вкладке свойство `Caption`. В данном случае это свойство отображает ту надпись, которая выводится в строке заголовка формы. Вводим вместо имеющегося в строке `Caption` по умолчанию слова `Form1` слово `Редактор`. Это же слово появится и в строке заголовка формы, выделенной синим цветом.

Следующим шагом станет изменение цвета формы. При создании формы по умолчанию она получает серый цвет. Для того чтобы сделать фон формы более ярким и красочным, нужно на вкладке **Properties** (Свойства) объекта `Form1` найти свойство `Color`. Оно содержит раскрывающийся список с названиями цветов, которые можно выбрать в качестве фона для формы. Слева от названия каждого цвета в небольшом прямоугольнике показано как выглядит этот цвет. Для нашей формы выбираем цвет `ClSkyBlue` (небесно-голубой). Приставка Cl, которую включает в себя название каждого из представленных в палитре цветов, представляет собой сокращение от английского слова Color (цвет). На этом работу с самой формой мы завершаем.

Затем необходимо создать надпись, которую в дальнейшем можно будет редактировать, заменяя ее своим текстом. Для создания надписи используем компонент `Label`, а соответствующий объект получит название `Label1`. Чтобы по содержанию надписи можно было сразу понять ее назначение, свойству `Caption` присваиваем значение: `Здесь отображается текст, вводимый пользователем`. Затем переходим к свойству `Font` того же объекта и щелкаем кнопку построителя. В открывшемся диалоговом окне выбираем какой-либо цвет, контрастирующий с фоном формы (например, ярко-красный) и увеличиваем размер шрифта до 14 пунктов.

Далее с помощью панели компонентов выводим на форму еще один объект — надпись. Он получит от системы наименование `Label2`. Расположим данный объект ниже первого. Он должен находиться примерно в середине формы по вертикали и ближе к левому ее краю по горизонтали. Содержанием данной надписи будет сообщение о том, что в окно, находящееся рядом с этой надписью, нужно вводить текст, который в укрупненном виде будет отображаться выше. Поэтому свойству `Caption` данной надписи мы присваиваем значение `Вводите текст в это окно`. Цвет данной надписи мы изменять не будем (он останется по умолчанию черным), а размер ее сделаем несколько меньшим, чем у основной надписи `Label1` (например, 12 пунктов).

Справа от надписи `Label2` расположим то самое текстовое окно, в которое пользователь может вводить текст произвольного содержания. Для создания окна используется компонент `Edit`, который находится на той же стандартной вкладке панели компонентов, что и компонент `Label` (не путайте компонент `Edit` с разделом меню, имеющим то же название). Выглядит компонент `Edit` следующим образом —  . Созданный с помощью этот компонента объект получает имя `Edit1`. Методом протягивания расширяем данный объект почти до правого края формы. Данный объект не имеет свойства `Caption`, но имеет сходное с ним свойство `Text`. Сразу после создания объекта данное свойство имеет значение `Edit1`, оно совпадает с именем объекта. Мы его удалим, а новое значение вводить не будем. Таким образом, после запуска программы на

выполнение текстовое окно останется пустым и будет свободно для ввода текста. Подобно надписи, объект "текстовое окно" имеет свойство `Font` с таким же построителем, открывающим диалоговое окно настройки шрифта. Цвет шрифта мы в данном случае изменять не будем, а размер его сделаем равным 10.

Когда в ходе работы программы мы будем вводить в данное окно текст, то в нем будет виден курсор, имеющий форму вертикальной черты и отмечающий текущую позицию ввода текста. Текст, который мы будем вводить в данное окно, можно по ходу дела исправлять. Напомним, что для удаления отдельных символов используются клавиши <Delete> и <Backspace>. Если требуется удалить сразу целый фрагмент введенного текста, то можно выделить этот фрагмент клавишей <стрелка влево> или <стрелка вправо> при нажатой клавише <Shift> и затем нажать клавишу <Delete>. Для быстрого перехода в начало или конец вводимого текста достаточно нажать соответственно клавиши <Home> или <End>. Все описанные действия не надо специальным образом программировать. Эту задачу берет на себя сама система Delphi.

Для завершения создания интерфейса программы осталось разместить на форме еще два объекта — командные кнопки, которые будут управлять работой приложения. Для создания этих кнопок используем компонент `Button`. Первый из этих объектов получит по умолчанию имя `Button1`. Мы поместим его в левой нижней части формы. Щелчок на этой кнопке будет приводить к тому, что введенная пользователем в текстовом окне фраза будет продублирована в виде надписи вверху формы. Изменим для этой кнопки свойство `Caption`. Вместо имеющегося по умолчанию заголовка `Button1` введем следующий: `Изменить текст надписи`. Воспользуемся свойством `Font` объекта `Button1` для того, чтобы увеличить шрифт надписи на кнопке до 10 пунктов. Кроме того, нужно увеличить методом протягивания размер объекта таким образом, чтобы надпись была целиком видна на экранной кнопке. Если этого не сделать, то при выполнении программы от надписи на кнопке будет виден только "хвост", а ее начало будет скрыто от пользователя.

После этого создаем второй объект-кнопку, которая по умолчанию получит имя `Button2`. Эту кнопку мы разместим в правой нижней части формы и свойству `Caption` присвоим значение `Закрыть программу`. Увеличим шрифт на кнопке до 10 пунктов и увеличим размеры кнопки для нормального отображения на ней надписи. Интерфейс программы приведен на рис. 13.12.

Завершающий этап составления программы заключается в написании кода, описывающего реакцию экранных кнопок при щелчке мышью по ним. Для объекта `Button2` выбираем на вкладке **Events** (События) событие `OnClick` и пишем для него код, состоящий из слова `Close`. Эта кнопка будет закрывать программу подобно кнопке **Выход** из предыдущей программы. Вид создан-

ной процедуры также аналогичен соответствующей процедуре в предыдущем проекте.

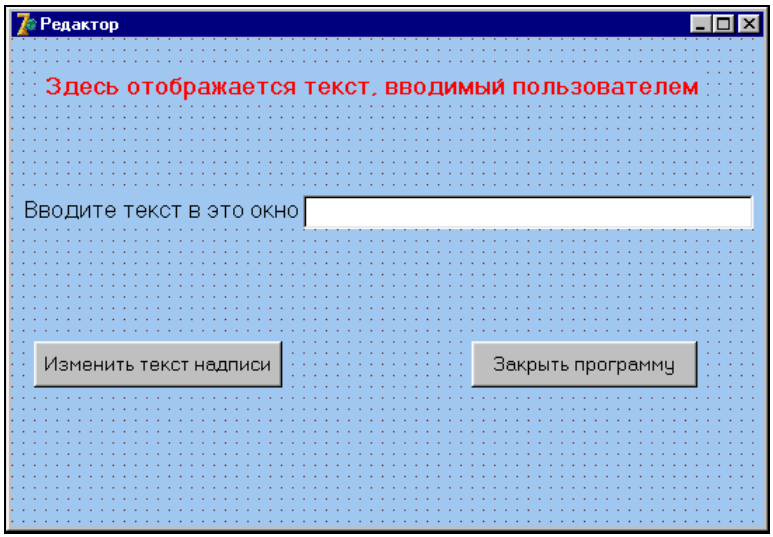


Рис. 13.12. Интерфейс программы "Редактор"

Для того чтобы разобраться в том, какой код нужно писать для кнопки `Button1`, необходимо прежде всего понять, что это кнопка должна делать. Как мы уже знаем, она должна изменять надпись, т. е. изменять ее свойство `Caption`. Изменение это производится путем присвоения свойству `Label1.Caption` нового значения, отличного от предыдущего. Как уже известно по предыдущей программе, операция присваивания состоит из знака равенства и двоеточия. Само же это новое значение содержится в объекте `Edit1`, а точнее в его свойстве `Text`. Поэтому мы и запишем это свойство так: `Edit1.Text`. Следовательно операция изменения заголовка надписи будет выглядеть следующим образом:

```
Label1.Caption:=Edit1.Text;
```

Для того чтобы вставить эту строку в подпрограмму, описывающую реакцию кнопки `Button1`, нужно выделить эту кнопку на форме, выделить в инспекторе объектов вкладку **Events** (События) и найти событие `OnClick`, для которого и указать этот код. В результате указанных действий процедура `Button1Click` будет выглядеть следующим образом:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
Label1.Caption:=Edit1.Text;
end;
```

Теперь программа готова к запуску ее на выполнение. В листинге 13.2 приводится полный текст данной программы.

Листинг 13.2. Текст программы "Редактор"

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls;

type
  TForm1 = class(TForm)
    Label1: TLabel;
    Label2: TLabel;
    Edit1: TEdit;
    Button1: TButton;
    Button2: TButton;
    procedure Button2Click(Sender: TObject);
    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.Button1Click(Sender: TObject);
begin
Label1.Caption:=Edit1.Text;
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
close;
end;

end.
```

После запуска программы мы увидим в ее окне чистое окошко для ввода текста с мигающим в левой его части курсором. В это текстовое окно пользователь может вводить любой понравившийся ему текст и корректировать его указанным ранее способом, если в том есть необходимость. После щелчка на экранной кнопке **Изменить текст надписи** курсор в текстовом окне исчезнет, и мы увидим введенный текст, написанный вверху формы большими красными буквами (рис. 13.13).

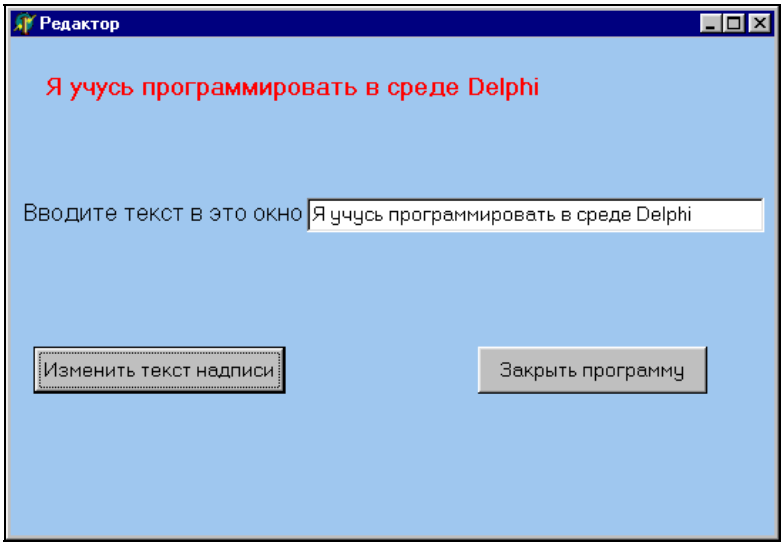


Рис. 13.13. Программа "Редактор" во время работы

В течение одного сеанса работы с программой содержание надписи можно изменять неоднократно. Для ввода нового текста достаточно щелкнуть мышью текстовое окно, в результате чего в нем вновь появится курсор. Тогда можно вводить в окно новый текст, который отобразится в верхней части формы после щелчка на экранной кнопке.

13.4. Программа "Хамелеон"

Следующим шагом в освоении системы Delphi станет создание программы, которая будет изменять в процессе работы свойства самой формы. Для этого мы используем еще один элемент управления, а точнее группу элементов. Элементы, относящиеся к этой группе, называются переключателями или радиокнопками (по аналогии с кнопками на старых радиоприемниках). Переключатели представляют собой группу элементов, в которой может быть включен только один из них, а остальные находятся в отключенном состоя-

нии. Работа группы переключателей аналогична работе исправного светофора, на котором всегда горит только одна лампочка, а остальные погашены.

В нашем проекте с помощью группы переключателей мы будем изменять цвет самой формы, которая после щелчка на одном из переключателей из стандартной серебристо-серой будет превращаться в красную, зеленую, синюю и т. д. Цвет формы будет меняться по нашему желанию подобно тому, как меняет свою окраску в зависимости от окружающей обстановки хамелеон, поэтому мы и назовем наш проект "хамелеон".

После запуска системы программирования в окне свойств объекта `Form1` (основная форма проекта) мы выделяем свойство `Caption` и в поле справа вводим значение `Хамелеон`, которое и будет отображаться в заголовке формы после запуска программы на выполнение. Затем в левом верхнем углу формы мы поместим надпись (объект `Label1`) следующего содержания: `Укажите цвет формы`. Для чего снова используем свойство `Caption`, только это должно быть свойство объекта `Label1`.

Под надписью мы расположим группу радиокнопок, которые и будут изменять цвет основной формы. Создание группы переключателей производится в два этапа. На первом этапе в экранную форму нужно поместить рамку для группы переключателей, в которой в дальнейшем будут размещены сами переключатели. Заготовка для этой рамки находится на палитре компонентов на вкладке **Standard**. Этот компонент называется `RadioGroup` и выглядит следующим образом —  . Выберем его щелчком мыши на вкладке, а затем щелчком на форме поместим его под надписью. В результате наших действий в экранной форме появится рамка, в которую в дальнейшем можно будет помещать переключатели. Растянем с помощью мыши эту рамку таким образом, чтобы в ней хватило места для всех восьми переключателей, которые будут в нашем проекте. Рамка для радиокнопок по умолчанию получает имя `RadioGroup1`. Такое же название будет отображаться и в верхней части рамки в экранной форме. Это значение свойства `Caption` объекта `RadioGroup1`. Имя объекта мы, как обычно, изменять не будем, а свойство `Caption` очистим, т. к. никакой надписи на самой рамке в нашем проекте быть не должно.

Следующим, вторым этапом создания группы элементов управления в нашем проекте станет создание самих переключателей в экранной форме. Заготовка для переключателей находится на той же вкладке **Standard** палитры компонентов и называется `RadioButton`. Выглядит же эта заготовка следующим образом —  . Выбираем этот компонент щелчком мыши и затем щелкаем мышью на экранной форме внутри рамки `RadioGroup1`. В результате внутри рамки появится объект с именем `RadioButton1` — первый из группы переключателей. Затем таким же образом перенесем внутрь рамки еще 7 переключателей, которые получат имена, начиная с `RadioButton2` и заканчивая

RadioButton8. Таким образом, вся внутренняя область рамки RadioGroup1 будет заполнена переключателями.

Следующим шагом в разработке интерфейса программы должна быть настройка всех элементов RadioButton. Начнем с элемента RadioButton1. Когда мы перенесли этот элемент на форму, то сама радиокнопка оказалась в левой части элемента, а надпись — в правой части. В нашем проекте для того, чтобы форма лучше смотрелась с точки зрения ее дизайна, мы поменяем эти части местами. Для этой цели мы в окне свойств элемента RadioButton1 найдем свойство Align (выравнивание). Это свойство имеет всего два фиксированных значения. Если его значение равно taRightJustify, то надпись выравнивается по правому краю элемента, а сам переключатель оказывается слева. Такой способ выравнивания задан по умолчанию. Если же значение свойства — taLeftJustify, то выравнивание производится по левому краю, а переключатель находится справа. Для изменения значения этого свойства нужно щелкнуть кнопку  , находящуюся справа от значения в окне свойств, и из раскрывающегося списка выбрать щелчком мыши значение taLeftJustify. Такую же операцию следуют произвести для всех остальных радиокнопок, находящихся внутри рамки.

Затем необходимо изменить надписи таким образом, чтобы было понятно, какой переключатель соответствует какому цвету. Для этого нужно изменить свойство Caption всех элементов RadioButton. Для переключателя RadioButton1 мы в поле Caption вместо RadioButton1 напишем Красный, для переключателя RadioButton2 в поле Caption мы напишем Травяной, для RadioButton3 — Синий, для RadioButton4 — Бирюзовый, для RadioButton5 — Желтый, для RadioButton6 — Сиреневый, для RadioButton7 — Зеленый и для RadioButton8 — Голубой.

Следующим шагом в создании интерфейса программы станет создание иллюстрации, демонстрирующей все цвета, в которые можно окрашивать экранную форму в данном проекте. Эту иллюстрацию мы не будем создавать сами, а возьмем ее в готовом виде из имеющейся на компьютере коллекции рисунков. Но до того как мы поместим на форме саму иллюстрацию, необходимо создать еще один объект, представляющий собой основу, на которой будет находиться эта иллюстрация. Для создания такой основы мы используем компонент Panel (панель), находящийся на вкладке **Standard** палитры компонентов и выглядящий следующим образом —  . Перетащив этот компонент на форму, получаем объект Panel1, на котором впоследствии будет находиться картинка.

Затем необходимо произвести настройку свойств объекта Panel1. Во-первых, т. к. данный объект будет только фоном для другого — самого рисунка, заголовок объекта Panel1 не должен отображаться на экране. Поэтому на вкладке

Properties (Свойства) данного объекта находим свойство `Caption` и очищаем его. Если размеры панели нас не устраивают, изменяем их как обычно методом перетаскивания.

Для создания визуального эффекта желательно, чтобы изображение было как бы немного вдавлено в экран. Этого можно добиться, изменив характеристики фаски, которая окружает по краям панель. По умолчанию ширина этой фаски (свойство `Bevelwidth`) равна 1 — одному пикселу. Увеличим эту ширину до двух пикселов, введя в поле справа от названия свойства соответствующее число. Кроме того, изменим сам характер этой фаски. Для чего щелкнем раскрывающийся список, содержащийся в свойстве `BevelOuter`. Значение данного свойства определяет внешний вид фаски. Вместо имеющегося по умолчанию значения `BvNone` мы устанавливаем значение `BvLowered`. В результате создается впечатление, что панель "утоплена" в экран.

Теперь на панель поместим еще один объект — сам рисунок. Компонент для его создания находится на вкладке **Additional** палитры компонентов и называется `Image`. Выглядит компонент `Image` так —  . Перетаскивая на панель этот компонент, получаем объект под названием `Image1`. На следующем этапе нужно изменить расположение этого объекта на экране и его размеры таким образом, чтобы он закрывал большую часть панели и в то же время оставлял на виду фаски панели.

Для того чтобы это сделать, нужно, прежде всего, иметь в виду, что месторасположение графического объекта на экране компьютера определяется положением его верхнего левого угла. Поэтому нужно совместить верхний левый угол объекта `Image1` с левым верхним углом фоновой панели `Panel1`. Выделяем в окне формы объект `Image1` и в инспекторе объектов просматриваем вкладку свойств данного объекта (вкладка **Properties**). На данной вкладке находим свойство `Top` (в переводе с английского — "верх"). Данное свойство определяет положение рисунка относительно верха панели, на которой он расположен. Задаем для свойства `Top` значение 2, т. е. верх рисунка будет на два пиксела ниже верха панели, для того чтобы не закрывать фаску. Затем находим на той же вкладке свойство `Left` (в переводе с английского — "левый"), которое определяет положение рисунка относительно левого края панели. Для свойства `Left` также устанавливаем значение, равное 2.

Теперь, когда рисунок "привязан" к верхнему левому углу панели, нужно определить его геометрические размеры. Размеры рисунка как по горизонтали, так и по вертикали должны быть на 4 пиксела меньше, чем соответствующие размеры панели, для того чтобы не закрывать фаску. Размеры же панели можно определить наведением указателя мыши на ее любое свободное место. Если задержать указатель мыши на панели на несколько секунд, то на ней

появится небольшое окно с основными характеристиками объекта, включая и его размеры. В этом окне будет информация следующего вида:

Size: 193*185

слово Size по-английски означает "размер", первое число — это размер панели в пикселах по горизонтали, а второе — размер по вертикали. Естественно, что у вас при разработке проекта числовые значения могут отличаться от указанных.

Для приведенного примера размер рисунка по горизонтали должен быть равен 189, а по вертикали — 181. Снова выделяем на форме объект `Image1` и задаем для свойства `Width` (ширина) значение 189, а для свойства `Height` (высота) значение 181. После определения геометрических размеров рисунка осталось сделать так, чтобы картинка, которую будет содержать рисунок, автоматически подгонялась под размеры рисунка, растягивалась соответственно рисунку. Для этого находим свойство `Stretch` (растянуть). Это свойство логическое и имеет только два значения: `false` и `true`. Устанавливаем для данного свойства значение `true`, и растяжка картинки будет производиться автоматически.

Теперь у пользователя может возникнуть вопрос о том, где взять картинку, которая будет размещена на рисунке. Здесь возможны следующие варианты: вы можете найти подходящую картинку в Интернете, скопировать ее с одного из компакт-дисков с коллекциями рисунков, поискать подходящую иллюстрацию в коллекции картинок, входящих в состав пакета Microsoft Office, либо нарисовать ее самостоятельно в любом графическом редакторе, например в Paint, являющемся стандартным приложением операционной системы Windows. В данном случае мы воспользуемся коллекцией рисунков Clipart, которая входит в состав пакета Microsoft Office 97. В настоящее время пакет Microsoft Office установлен практически на всех компьютерах, работающих под управлением ОС Windows.

Для того чтобы вставить в форму готовую картинку, мы выделяем в форме объект `Image1` и на вкладке свойств объекта находим свойство `Picture` (картинка). По умолчанию значение этого свойства равно `None` (ничего), а справа от значения находится уже знакомая нам кнопка-построитель. Щелкнув мышью на построителе, мы открываем диалоговое окно **Picture Editor** (редактор картинок).

Затем, щелкнув в окне редактора картинок на экранной кнопке **Load** (загрузить), мы открываем новое диалоговое окно **Load picture** (загрузка картинки). В этом окне мы производим навигацию (перемещение) по файловой структуре компьютера для поиска интересующего нас рисунка. Для навигации используются элементы управления, расположенные на панели инструментов в верхней части диалогового окна. Вначале мы используем кнопку 

(переход на один уровень вверх). Щелкая несколько раз по этой кнопке, можно подняться вверх по файловой структуре компьютера, пока не будет достигнут уровень Мой компьютер. Текущий уровень отображается в небольшом окне, расположенном в левой части панели инструментов. В основном же окне, расположенном ниже, отображаются диски, папки и файлы текущего уровня. Для того чтобы открыть диск или папку, нужно дважды щелкнуть мышью на соответствующем значке. Открываем диск, на котором находятся файлы пакета Microsoft Office (при установке данного пакета по умолчанию эти файлы записываются на диск C:). Затем находим там папку Program Files, а внутри нее открываем папку Microsoft Office.

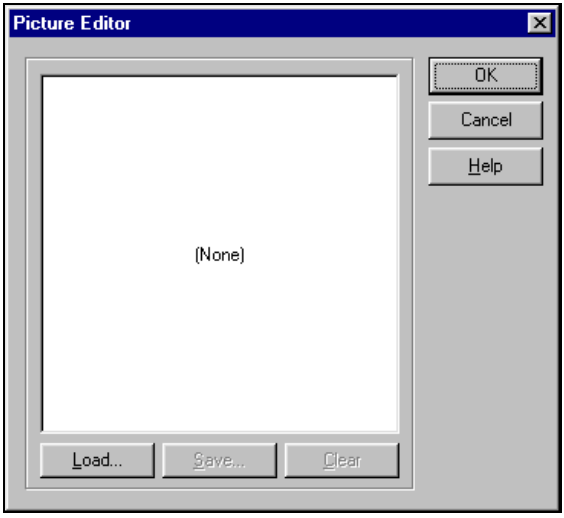


Рис. 13.14. Диалоговое окно редактора картинок

Дальнейшая последовательность действий зависит от той версии Microsoft Office, которая установлена на данном компьютере. Для версии Microsoft Office 97 последовательность действий будет следующей. В папке Microsoft Office находим вложенную папку Clipart. Внутри папки Clipart содержится папка Popular, в которой и находятся файлы с картинками. Это графические файлы, созданные средствами векторной графики и имеющие расширение wmf. Когда мы выбираем в окне один из этих файлов, то справа от основного окна появляется изображение, находящееся в данном файле. Сверху от изображения указывается его размер в пикселах (эти сведения могут пригодиться при размещении файла внутри формы). Если изображение слишком мелкое и пользователь хочет рассмотреть его более подробно, то для этого нужно щелкнуть находящуюся также сверху кнопку  (предварительный просмотр). После щелчка на этой кнопке появляется еще одно окно, в котором

изображение представлено в натуральную величину. Закрыть это окно можно обычным способом — щелчком на закрывающей кнопке. Для нашего проекта выбираем изображение Arrows4.wmf. Данное изображение представляет собой группу стрелок восьми разных цветов, расположенных по кругу, т. е. соответствует теме нашего проекта (рис. 13.15). Для загрузки этого изображения щелкаем экранную кнопку **Открыть** и подтверждаем сделанный выбор щелчком на кнопке **OK** в окне редактора картинок. В результате выбранное изображение в формате wmf будет размещено на объекте `Image1`.

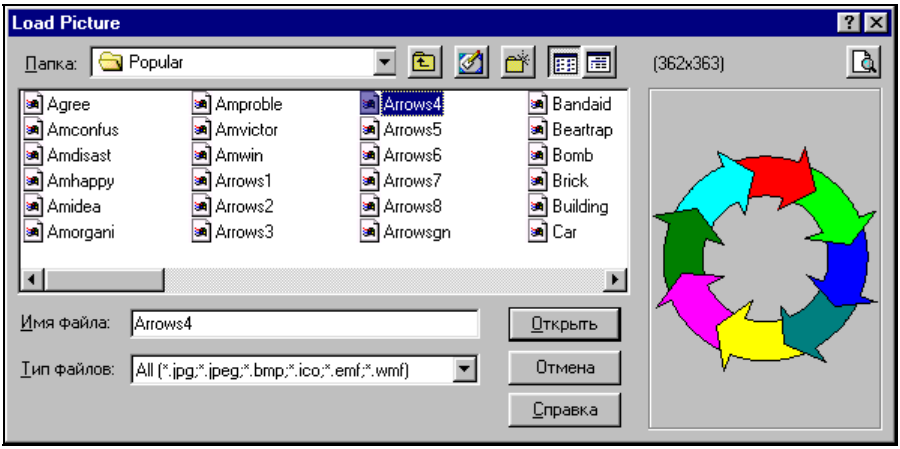


Рис. 13.15. Диалоговое окно **Load Picture**

Если на компьютере установлена другая версия пакета Microsoft Office, то адрес папки, содержащей изображения, которые можно вставить в проект, может быть иным. Например, для версии Microsoft Office XP нужно в папке Microsoft Office открыть вложенную папку media, а внутри папки media открыть папку cagcat10, которая и содержит коллекцию рисунков. Другими могут быть и сами рисунки. Однако описанные ранее методы загрузки файлов с помощью окна редактирования картинок и окна загрузки остаются при этом неизменными.

На завершающем этапе разработки графического интерфейса программы мы разместим на экранной форме две кнопки. Одна из этих кнопок (`Button1`) будет выводить на форму сведения об авторе программы, а другая (`Button2`) будет закрывать форму и вместе с ней сам проект. Сведения об авторе будут выводиться на форме посредством надписи `Label2`. Аналогичные объекты уже создавались нами в предыдущих программах, поэтому их создание не должно вызвать у читателя затруднений. Общий вид получившегося интерфейса приведен на рис. 13.16.

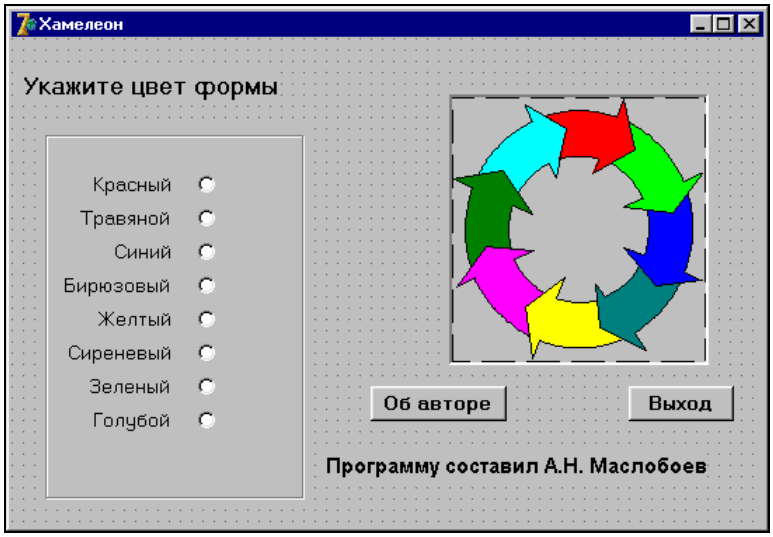


Рис. 13.16. Графический интерфейс программы "Хамелеон"

Далее, для того чтобы программа заработала, нужно написать программный код, описывающий действие каждого из имеющихся на форме переключателей. Событием, которое включает очередной переключатель и окрашивает форму в новый цвет, является обычный щелчок мышью на нем. Щелчку мыши на каком-либо из переключателей в Delphi соответствует событие `RadioButtonClick`. Название этого события нам не понадобится вводить вручную, соответствующая заготовка появится в окне кода после двойного щелчка на переключателе. Например, если мы щелкнем на первом переключателе в группе, то в окне кода появится заготовка процедуры `RadioButton1Click`.

В эту заготовку для переключателя нужно вставить только одну строку кода. Она должна изменить имеющийся цвет формы, т. е. свойство `Form.Color`. Это изменение производится путем присваивания свойству `Form.Color` нового значения, соответствующего одному из стандартных цветов системы. Перед названием такого стандартного цвета обязательно должна быть приставка cl (сокращение от color). Например, если мы хотим выкрасить форму в красный цвет, то для этого нужно воспользоваться следующим оператором:

```
Form1.Color:=clRed;
```

Целиком же эта процедура будет выглядеть так:

```
procedure TForm1.RadioButton1Click(Sender: TObject);
begin
Form1.Color:=clRed;
end;
```

Для того чтобы выкрасить экранную форму в ярко-зеленый (травяной) цвет, нужно для второго переключателя в группе написать код следующего вида:

```
Form1.Color:=clime;
```

Таким образом необходимо будет ввести соответствующие коды для каждого из оставшихся переключателей. Наконец, необходимо ввести программные коды, которые определяют действия, осуществляемые после щелчка на одной из экранных кнопок (`Button1` или `Button2`). Эти коды аналогичны таким же кодам в предыдущих программах. В результате общий вид программы, написанный на языке Object Pascal, будет следующим — листинг 13.3.

Листинг 13.3. Код программы "Хамелеон"

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, ExtCtrls;

type
  TForm1 = class(TForm)
    Label1: TLabel;
    RadioGroup1: TRadioGroup;
    RadioButton1: TRadioButton;
    RadioButton2: TRadioButton;
    RadioButton3: TRadioButton;
    RadioButton4: TRadioButton;
    RadioButton5: TRadioButton;
    RadioButton6: TRadioButton;
    RadioButton7: TRadioButton;
    RadioButton8: TRadioButton;
    Panel1: TPanel;
    Image1: TImage;
    Button1: TButton;
    Button2: TButton;
    Label2: TLabel;
    procedure RadioButton1Click(Sender: TObject);
    procedure Button1Click(Sender: TObject);
    procedure RadioButton2Click(Sender: TObject);
    procedure RadioButton3Click(Sender: TObject);
    procedure RadioButton4Click(Sender: TObject);
```

```
    procedure RadioButton5Click(Sender: TObject);
    procedure RadioButton6Click(Sender: TObject);
    procedure RadioButton7Click(Sender: TObject);
    procedure RadioButton8Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);

  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.RadioButton1Click(Sender: TObject);
begin
Form1.Color:=clRed;
end;

procedure TForm1.RadioButton2Click(Sender: TObject);
begin
Form1.Color:=clLime;
end;

procedure TForm1.RadioButton3Click(Sender: TObject);
begin
Form1.Color:=clBlue;
end;

procedure TForm1.RadioButton4Click(Sender: TObject);
begin
Form1.Color:=clTeal;
end;

procedure TForm1.RadioButton5Click(Sender: TObject);
begin
Form1.Color:=clYellow;
end;
```

```
procedure TForm1.RadioButton6Click(Sender: TObject);
begin
Form1.Color:=clFuchsia;
end;

procedure TForm1.RadioButton7Click(Sender: TObject);
begin
Form1.Color:=clGreen;
end;

procedure TForm1.RadioButton8Click(Sender: TObject);
begin
Form1.Color:=clAqua;
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
close;
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
Label2.Visible:=True;
end;

end.
```

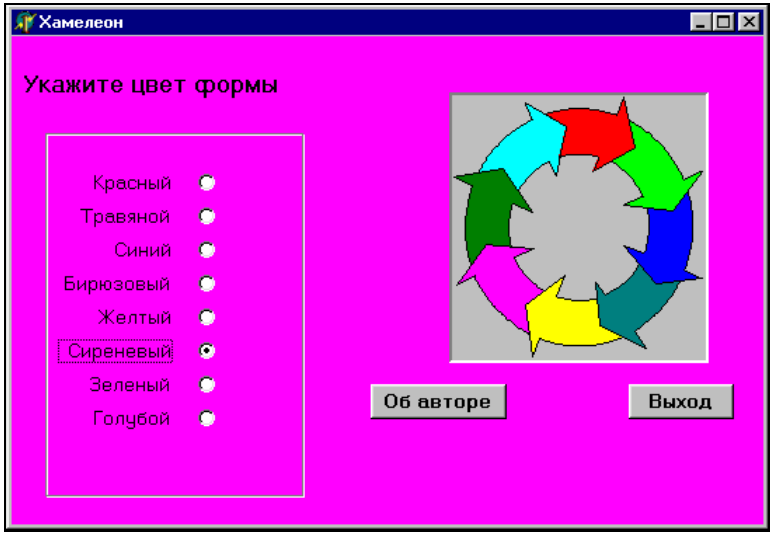


Рис. 13.17. Программа "Хамелеон" во время работы

Для лучшего понимания текста программы приведем русскоязычные названия имеющихся в программе цветов. Цвет Red переводится как красный, Lime — это травяной цвет, Blue — синий, Teal — бирюзовый, Fuchsia — сиреневый, Yellow — желтый, Green — темно-зеленый, Aqua — голубой. Вид программы после ее запуска на выполнение показан на рис. 13.17.

Задание для самостоятельной работы

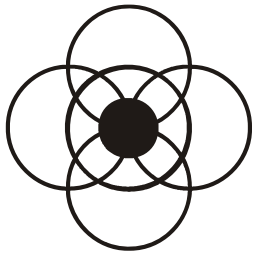
Составить проект "Светофор". В данном проекте в основной форме должна выводиться панель с тремя лампочками. Слева от панели с лампочками должна находиться группа, состоящая из трех переключателей. На каждом из переключателей должен быть обозначен цвет лампочки, который включает пользователь. После щелчка на одном из переключателей на панели должна загораться лампочка соответствующего цвета (красная, желтая или зеленая), а две другие лампочки должны гаснуть. На форме необходимо предусмотреть кнопку для ее закрытия.

Указания для выполнения задания. На панель следует вывести три объекта типа `Shape`, которые в дальнейшем будут представлять собой лампочки светофора. Объект `Shape` находится на вкладке **Additional** палитры компонентов. Этот объект представляет собой контур геометрической фигуры, которая в зависимости от значения свойства, тоже называемого `Shape`, может принимать форму прямоугольника, квадрата, круга, эллипса, а также прямоугольника или квадрата с закругленными углами.

При этом внутренность создаваемого объекта по умолчанию не закрашивается. Для того чтобы залить внутреннюю область объекта, необходимо использовать составное свойство `Brush`. Это свойство содержит два подсвойства: `Color` — определяет цвет заливки, и `Style` — задает стиль заливки. Следующим важным свойством объекта `Shape` является свойство `Pen`, которое также является составным. Свойство `Pen` содержит такие подсвойства, как `Color` (цвет контура фигуры), `Style` (стиль границы), `Width` (ширина границы).

Для свойства `Shape` каждого из этих объектов указать значение `stCircle` (в результате объект `Shape` примет форму окружности). Границу каждого из объектов можно обозначить черным цветом с помощью подсвойства `Pen.Color`. При начальной настройке каждого из объектов `Shape` ему нужно придать такой же серебристо-серый цвет, как и самой панели. Для этого следует выбрать соответствующее значение для подсвойства `Brush.Color`. При включении очередной лампочки (т. е. изменения цвета одного из объектов `Shape`) нужно возвращать двум другим объектам их изначальный серебристо-серый цвет (`clSilver`). Таким образом можно добиться того, чтобы на панели светофора всегда горела только одна лампочка.

ГЛАВА 14



Программы линейной структуры в Delphi

Программы, которые были ранее разработаны в системе Delphi, позволяли только выводить информацию на экран компьютера (первая программа, папка First) или вводить информацию с клавиатуры с отображением ее на экране (программа "Редактор", папка Redaktor). Однако основными задачами, ради которых были разработаны компьютеры, являются задачи по обработке информации, в том числе задачи вычислительного характера (само слово "компьютер" в переводе с английского означает "вычислитель"). Поэтому данная глава книги посвящена составлению программ для решения вычислительных задач.

14.1. Программа "Сложение"

Программа, которую мы будем составлять, должна выполнять следующие действия: она должна обеспечивать ввод двух целых чисел с клавиатуры компьютера (введенные числа будут отображаться в двух текстовых окнах) и выводить в третьем окне сумму этих двух чисел. На примере такой несложной задачи мы и освоим основные приемы разработки программ вычислительного характера.

Для этого нам нужно ознакомиться с использованием переменных в языке Object Pascal. Основные правила работы с переменными в Object Pascal те же, что и в Turbo Pascal, но есть и некоторые особенности. Правила наименования переменных аналогичны тем же правилам в Turbo Pascal, т. е. для обозначения переменных должны использоваться только латинские буквы и цифры. Оператор присваивания в Object Pascal выглядит так же, как и в Turbo Pascal.

Так же, как и в Turbo Pascal, переменные, используемые в программе, должны быть описаны в специальном разделе программы, который начинается со

слова var. Если этого не сделать, то при запуске программы на выполнение компилятор выдаст сообщение об ошибке. Пример, показывающий, как может выглядеть сообщение об ошибке:

```
[Error] Unit1.pas(37): Undeclared identifier: 'x'
[Fatal Error] Project1.dpr(5): Could not compile used unit 'Unit1.pas'
```

В первой строке данного сообщения указывается номер строки в тексте программы, в которой была обнаружена ошибка (в приведенном примере строка № 37), а также приводится имя той переменной, которая не была заранее описана (в примере — это переменная x). Во второй строке говорится о том, что данная ошибка является фатальной, т. е. из-за нее программа не может работать.

Как и в Turbo Pascal, все переменные, используемые в языке Object Pascal, подразделяются на два основных вида. Это *глобальные* переменные, которые могут использоваться во всей программе, и *локальные* переменные, которые можно использовать только внутри определенной процедуры. Практика программирования показывает, что в программе должно быть как можно меньше глобальных переменных, т. к. если программист допускает ошибку, связанную с использованием такой переменной, то для ее выявления и устранения ему приходится анализировать весь текст программы, который может быть достаточно большим по объему. Соответственно и поиск ошибки может занять много времени. Если же ошибка была допущена при работе с локальной переменной, то для ее устранения достаточно будет поработать только с текстом соответствующей процедуры, что займет значительно меньше времени.

Как помнит читатель, в программах, написанных в Turbo Pascal, значения, присваиваемые переменным, обычно вводились с клавиатуры в числовом виде. При разработке же нашего проекта нам придется столкнуться с иной ситуацией. Числовые величины будут вводиться с клавиатуры в текстовые окна и затем уже использоваться для вычислений. Данные же, вводимые пользователем в текстовое окно, интерпретируются системой программирования как текст, даже в том случае, если на самом деле эти данные являются числами. Поэтому, для того чтобы перевести текстовую информацию в числовую, необходимо использовать стандартную функцию, являющуюся частью языка Object Pascal. Эта функция называется StrtoInt. В названии данной функции Str — это сокращение от английского слова string (строка), а Int — от integer (целое). Таким образом, название данной функции можно расшифровать как "преобразовать строку в целое число". Аргументом данной функции является текстовая величина, а значением — соответствующая числовая величина. Функции в языке Object Pascal не являются самостоятельными операторами,

а входят составной частью в другие операторы, например оператор присваивания. Приведем пример оператора с использованием функции:

```
x:=StrtoInt(t);
```

В данном операторе происходит преобразование текстовой величины `t` в числовую, и эта числовая величина присваивается целочисленной переменной `x`. В программе же в качестве аргумента чаще всего будет выступать свойство какого-либо объекта, например свойство `Text` объекта `Edit`. Значением данного свойства, как мы уже знаем по предыдущему проекту, является текст, введенный пользователем в текстовое окно. Поэтому операция преобразования введенного в окно `Edit1` текста в числовую величину может выглядеть следующим образом:

```
x:=StrtoInt(Edit1.Text);
```

Когда введенные пользователем данные будут преобразованы в числовые величины и над ними будут произведены все необходимые действия, перед программистом встанет обратная задача: как полученный численный результат преобразовать в текст, который может быть выведен на экран компьютера. Для этой цели используется другая стандартная функция `InttoStr`. Как нетрудно догадаться по ее названию, она выполняет операцию, обратную операции `StrtoInt`. Функция переводит числовую величину в соответствующую ей строковую величину. Значениями этой функции могут быть свойства имеющихся в проекте графических объектов. В качестве свойства, которому может присваиваться значение функции `InttoStr`, может выступать, например, свойство `Caption` объекта `Label` или то же самое свойство `Text` для объекта `Edit`. Если мы в нашем проекте хотим вывести полученный в программе числовой результат (например, значение переменной `z`) в текстовом окне `Edit2`, то оператор, выполняющий в программе это действие, будет выглядеть так:

```
Edit2.Text:=InttoStr(z);
```

Теперь, освоив основные понятия и операторы, используемые при работе с переменными, мы можем приступать к непосредственному созданию нашего проекта. Как обычно, работу над программой мы начнем с создания ее графического интерфейса. Разработку интерфейса мы начнем с настройки самой формы. Изменим заголовок формы (свойство `Caption` объекта `Form1`). Теперь окно формы будет называться `Сложение`. Для того чтобы изменить цвет формы на менее унылый, чем стандартный серый, мы откроем раскрывающийся список цветов, находящийся в свойстве `Color`, и выберем из списка бледно-зеленый цвет, который по-английски называется `ClMoneyGreen`, т. е. цвет денег. Очевидно, имеется в виду цвет долларовых купюр старого образца (в 5 версии данный цвет в списке отсутствует, поэтому можно подобрать какой-либо другой неяркий цвет или оставить по умолчанию светло-серый).

Кроме основной формы, графический интерфейс данной программы будет включать в себя следующие элементы. Во-первых, это два текстовых окна, в которые будут вводиться исходные данные. Во-вторых, это третье текстовое окно, в котором будет отображаться сумма, т. е. результат работы программы. Для этого в форме нужно создать объекты `Edit1`, `Edit2` и `Edit3`. Для всех трех объектов мы удаляем содержимое свойства `Text`, чтобы в начале работы программы текстовые окна были пустыми.

Для того чтобы пользователю было понятно, куда именно нужно вводить числа и где получится их сумма, поместим в форму три поясняющие надписи — по одной надписи к каждому текстовому окну. Это будут объекты с именами `Label1`, `Label2` и `Label3`. Создание этих элементов интерфейса не должно вызвать затруднений у пользователя, т. к. аналогичные объекты мы создавали в предыдущих программах. Размер шрифта надписей увеличиваем до 10, шрифт делаем полужирным и цвет надписей изменяем на синий. Для каждого из этих объектов изменяем свойство `Caption`. Для первой надписи заголовок будет `Введите первое число`, для второй — `Введите второе число` и для третьей — `Сумма двух чисел равна`.

Завершаем создание графического интерфейса выводом в форму двух экранных кнопок. При нажатии на первую кнопку `Button1` будет производиться операция сложения двух чисел, а нажатие второй кнопки `Button2` будет закрывать программу. Как обычно, для экранных кнопок мы изменяем свойство `Caption` (надписи на кнопках). На первой кнопке должно быть написано `Сложить`, а на второй — `Закрыть программу`. Графический интерфейс данной программы пользователь может увидеть на рис. 14.1.

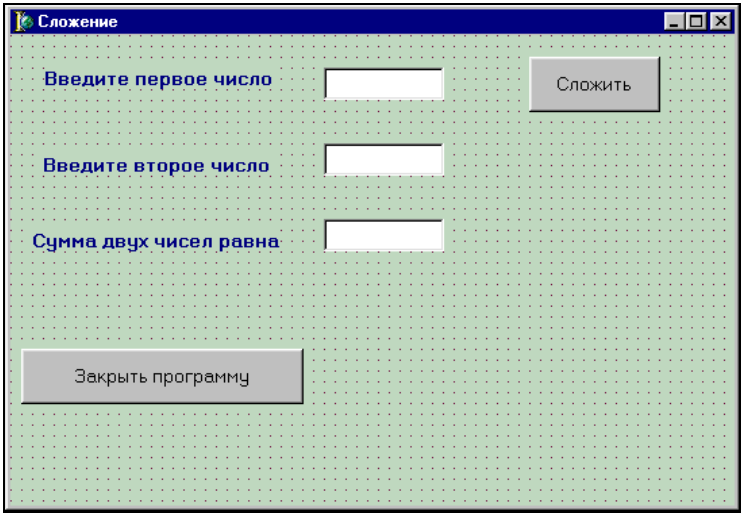


Рис. 14.1. Графический интерфейс программы "Сложение"

Теперь мы можем приступить к написанию кода для данной программы. Введем в программе три переменные, значением каждой из которых будет числовое значение текстовой информации, отображаемой в одном из текстовых окон. Поэтому данные переменные мы назовем `e1`, `e2` и `e3` (в соответствии с окнами `Edit1`, `Edit2` и `Edit3`). Действия с данными переменными будут производиться при нажатии экранной кнопки `Button1`. Поэтому описание этих переменных мы вставим в процедуру `TForm1.Button1Click`. Данная процедура описывает реакцию кнопки на щелчок мышью по ней. Как читатель, конечно, помнит, для того чтобы создать заготовку данной процедуры, достаточно выделить в форме объект `Button1`, выделить вкладку **Events** (События) для данного объекта в инспекторе объектов и дважды щелкнуть мышью справа от события `OnClick`. Описание переменных будет выглядеть так:

```
var e1,e2,e3:integer;
```

и находиться оно будет сразу после строки заголовка процедуры `TForm1.Button1Click`.

Переменным `e1` и `e2` будут присвоены значения первого и второго слагаемых, вводимых в текстовые окна `Edit1` и `Edit2`. Для преобразования текста, вводимого в эти окна, в числовые значения переменных `e1` и `e2`, мы воспользуемся известной нам стандартной функцией языка Object Pascal `InttoStr`. Преобразование выполним с помощью операторов:

```
e1:=StrtoInt(Edit1.text);
e2:=StrtoInt(Edit2.text);
```

Следующий оператор в программе производит сложение значений двух переменных и присваивает сумму переменной `e3`:

```
e3:=e1+e2;
```

Наконец, осталось преобразовать полученное числовое значение в текстовое, которое будет отображаться в окне `Edit3`. Для этого используется оператор:

```
Edit3.Text:=InttoStr(e3)
```

поскольку данный оператор является последним перед служебным словом `end`, обозначающим конец процедуры, то точку с запятой в конце оператора можно не ставить (хотя, если мы ее поставим, это также не будет считаться ошибкой). Целиком процедура, описывающая сложение двух чисел при нажатии экранной кнопки **Сложить**, будет выглядеть следующим образом:

```
procedure TForm1.Button1Click(Sender: TObject);
var e1,e2,e3:integer;
begin
e1:=strtoint(Edit1.text);
```

```
e2:=strtoint(Edit2.text);
e3:=e1+e2;
Edit3.Text:=InttoStr(e3)
end;
```

Строка заголовка процедуры, а также служебные слова `begin` и `end` генерируются системой программирования автоматически. Остальные строки в процедуру мы добавили сами. Для завершения работы осталось вставить строку кода в процедуру `Button2Click`, которая выполняется при щелчке мышью на экранной кнопке `Button2`. Эта строка, как мы уже знаем, состоит из одного слова `close`. Теперь программа готова к работе. В листинге 14.1 приводится полный текст данной программы.

Листинг 14.1. Текст программы, складывающей два числа

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls;

type
  TForm1 = class(TForm)
    Label1: TLabel;
    Label2: TLabel;
    Edit1: TEdit;
    Edit2: TEdit;
    Label3: TLabel;
    Edit3: TEdit;
    Button1: TButton;
    Button2: TButton;
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation
```

```
{$R *.dfm}

procedure TForm1.Button1Click(Sender: TObject);
var e1,e2,e3:integer;
begin
e1:=strtoint(Edit1.text);
e2:=strtoint(Edit2.text);
e3:=e1+e2;
Edit3.Text:=InttoStr(e3)
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
close
end;

end.
```

В ходе работы программы исходные данные вводятся в текстовые окна, причем переход между окнами может производиться либо нажатием клавиши <Tab>, либо щелчком мыши в соответствующем окне. Значения слагаемых во время работы программы можно неоднократно менять, и при каждом щелчке на кнопке **Сложить** в окне результата будет видна сумма, полученная с учетом этих изменений. Окно программы во время работы показано на рис. 14.2.

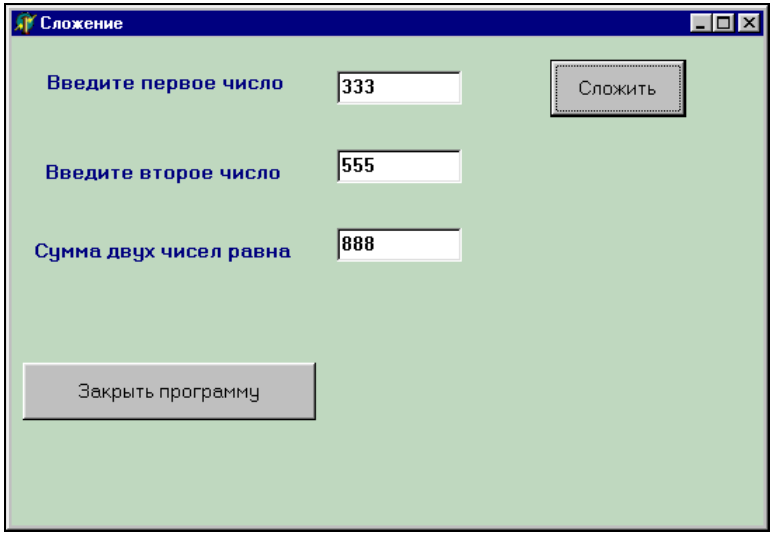


Рис. 14.2. Окно программы "Сложение" во время работы

14.2. Программа "Параллелепипед"

Следующая задача, которую мы будем решать в системе Delphi, заключается в создании программы, определяющей объем параллелепипеда по известной длине, ширине и высоте данной геометрической фигуры, а также вычисляющей общую площадь его поверхности. Как читатель помнит из школьного курса математики, первая часть задачи является очень простой, т. к. объем параллелепипеда определяется путем вычисления произведения его длины, ширины и высоты. Задача нахождения общей площади поверхности чуть сложнее, но и она требует знания лишь элементарной математики, т. к. общая площадь поверхности представляет собой сумму площадей отдельных граней параллелепипеда. Однако именно на примере такой простой задачи легче всего освоить новые элементы интерфейса, которые мы в дальнейшем будем использовать в других программах.

Для начала, как всегда, мы настроим саму форму, изменив ее заголовок на `Параллелепипед` и цвет на бледно-зеленый. Затем поместим в форму уже знакомые нам элементы интерфейса. Это будет пять текстовых окон. В первое, второе и третье окно будут вводиться соответственно длина, ширина и высота параллелепипеда. Эти объекты именуются `Edit1`, `Edit2` и `Edit3`. В окне `Edit4` будет отображаться вычисленный компьютером объем параллелепипеда, а в окне `Edit5` — подсчитанная компьютером общая площадь поверхности данной фигуры. Для того чтобы все эти окна в начале работы программы оставались чистыми, мы очищаем свойство `Text` для всех этих объектов.

Каждому текстовому окну должна соответствовать надпись, которая поясняет его назначение. Поэтому в форме мы также размещаем пять надписей с именами, начиная с `Label1` и заканчивая `Label5`. Для каждой из этих надписей мы, как и в предыдущей программе, изменяем размер, начертание и цвет шрифта. Естественно, что у каждой надписи нужно изменить свойство `Caption`, чтобы их заголовки говорили о содержимом находящихся справа от надписи окон. Кроме того, добавим в форму две экранные кнопки: `Button1` и `Button2`. При нажатии на первой из этих кнопок будет происходить вычисление искомых величин. Поэтому мы дадим ей заголовок `Подсчитать`. Вторая кнопка будет закрывать программу и получит соответствующий заголовок.

Все компоненты, используемые для создания указанных ранее элементов интерфейса, хорошо известны нам по предыдущим программам. Но, помимо них, для наглядности программы хорошо было бы, чтобы в ее окне был изображен тот самый геометрический объект, характеристики которого мы рассчитываем. Поэтому в окно программы желательно поместить изображение параллелепипеда. Как и в проекте, описанном в предыдущей главе, создание изображения мы начнем с того, что в экранной форме расположим объект

`Panel1` и настроим его аналогично тому, как это было сделано в описанном ранее проекте. Затем поверх объекта-фона `Panel1` мы поместим объект `Image1`, который будет содержать саму иллюстрацию. После того как рисунок будет "привязан" к верхнему левому углу панели, нужно определить его геометрические размеры. Однако здесь необходимо учесть некоторые особенности текущего проекта.

В данном случае мы создадим рисунок в графическом редакторе Paint, т. к. картинку с изображением параллелепипеда нетрудно нарисовать самостоятельно. Созданную картинку сохраняем в той же папке, где находятся файлы нашего проекта под именем "Картинка 1". Графический редактор Paint — это стандартное приложение Windows, которое есть на всех компьютерах, где установлена данная операционная система, он прост и удобен в использовании, однако к его недостаткам следует отнести то, что он является растровым. Растровые изображения представляют собой совокупность точек, которая плохо поддается масштабированию. При изменении размеров растрового рисунка он может заметно искажаться.

Поэтому в данном проекте лучше сначала определить размеры созданного вами рисунка. Это можно сделать с помощью команды **Атрибуты**, которая находится в меню **Рисунок** редактора Paint. Затем следует под полученные размеры подогнать высоту и ширину объектов `Image1` и `Panel1`. Для объекта `Image1` размеры можно сделать равными размерам рисунка, а для `Panel1` высоту и ширину можно сделать на 4 пиксела больше, чем у рисунка (с учетом размеров фаски). В этом случае созданный нами рисунок будет воспроизводиться на форме проекта без искажений. Загрузка картинки в `Image1` производится путем использования свойства `Picture`, как и в предыдущем проекте.

В итоге нашей работы графический интерфейс программы приобретет законченный вид, который представлен на рис. 14.3. Остается дополнить проект кодом, описывающим реакцию программы на действие экранных кнопок.

Создание кода начинаем с того, что двойным щелчком справа от события `OnClick` для объекта `Button1` открываем процедуру `TForm1.Button1Click`. В процедуре используются переменные `a`, `b`, `c`, которые содержат числовые значения длины, ширины и высоты параллелепипеда, а также `s` — площадь поверхности параллелепипеда и `v` — объем параллелепипеда. Описание переменных вставляем после строки заголовка процедуры:

```
var a,b,c,s,v:integer;
```

Затем в основной части процедуры (после слова `begin`) вставляем группу операторов, которые преобразуют введенные в текстовые окна длину, ширину и высоту в числовые значения. Эти операторы, использующие стандартную функцию `InttoStr`, будут выглядеть так:

```
a:=strtoint(Edit1.Text);
b:=strtoint(Edit2.Text);
c:=strtoint(Edit3.Text);
```

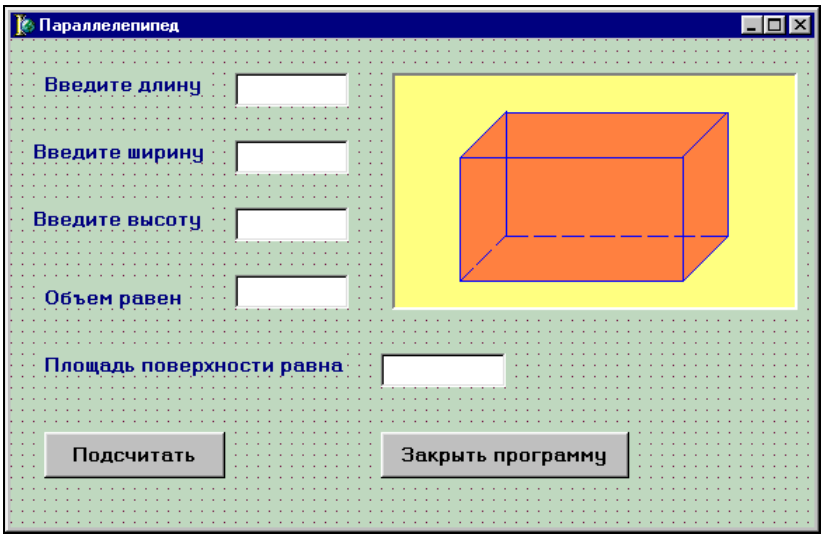


Рис. 14.3. Графический интерфейс программы "Параллелепипед"

Следующие операторы вычисляют объем, который присваивается переменной `v`, и преобразуют числовое значение этой переменной в текст, выводимый в окне `Edit4`.

```
v:=a*b*c;
Edit4.Text:=inttostr(v);
```

Далее нужно определить общую площадь поверхности параллелепипеда, которая представляет собой сумму площадей всех его шести граней. Площади граней представляют собой произведение длины и ширины, длины и высоты, ширины и высоты, причем площади передней и задней грани, верхней и нижней, левой боковой и правой боковой граней равны между собой. Поэтому достаточно найти по отдельности площади только трех граней и каждую полученную площадь умножить на два, а затем сложить полученные произведения. После вычисления значения переменной `s` нужно преобразовать ее в текст, выводимый в окне `Edit5`. Указанная последовательность действий реализуется с помощью следующих операторов:

```
s:=2*a*b+2*a*c+2*b*c;
Edit5.Text:=inttostr(s)
```

Процедура `TForm1.Button1Click` в целом будет выглядеть так:

```
procedure TForm1.Button1Click(Sender: TObject);
var a,b,c,s,v:integer;
begin
a:=strtoint(Edit1.Text);
b:=strtoint(Edit2.Text);
c:=strtoint(Edit3.Text);
v:=a*b*c;
Edit4.Text:=inttostr(v);
s:=2*a*b+2*a*c+2*b*c;
Edit5.Text:=inttostr(s)
end;
```

Последнее, что нам осталось сделать, — это вставить команду `close`, закрывающую окно программы в процедуру `TForm1.Button2Click`, описывающую реакцию на щелчок мышью на экранной кнопке `Button2`. В листинге 14.2 приведен полный текст программы.

Листинг 14.2. Текст программы, вычисляющей объем параллелепипеда

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, ExtCtrls;

type
  TForm1 = class(TForm)
    Label1: TLabel;
    Label2: TLabel;
    Edit1: TEdit;
    Edit2: TEdit;
    Label3: TLabel;
    Edit3: TEdit;
    Label4: TLabel;
    Edit4: TEdit;
    Label5: TLabel;
    Edit5: TEdit;
    Button1: TButton;
    Button2: TButton;
    Panel1: TPanel;
    Image1: TImage;
```

```
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.Button1Click(Sender: TObject);
var a,b,c,s,v:integer;
begin
a:=StrtoInt(Edit1.Text);
b:=StrtoInt(Edit2.Text);
c:=StrtoInt(Edit3.Text);
v:=a*b*c;
s:=2*(a*b+a*c+b*c);
Edit4.Text:=InttoStr(v);
Edit5.Text:=InttoStr(s);
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
close
end;

end.
```

Теперь программу можно запускать на выполнение. После ввода исходных данных в текстовые окна и щелчка на экранной кнопке **Подсчитать** программа выдаст на экран сразу два результата: объем параллелепипеда и площадь его поверхности. Экран программы в процессе ее работы показан на рис. 14.4.

14.3. Программа "Хронометр"

Как известно, хронометром называется прибор, используемый для точного измерения времени. Проект, который мы будем разрабатывать в этом разделе, также позволит измерять время. В данном случае это будет период,

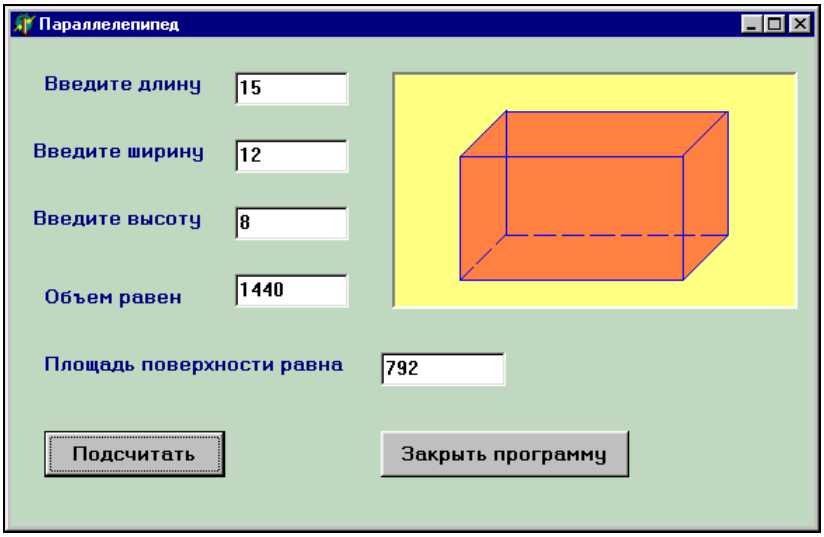


Рис. 14.4. Программа "Параллелепипед" во время работы

прошедший от одного момента времени до другого. И первый и второй моменты времени в проекте будут задаваться в часах и минутах. Период между этими двумя моментами мы также будем определять в часах и минутах.

Для того чтобы разобраться, как правильно решать поставленную задачу, необходимо использовать математические действия, имеющиеся в языке Object Pascal. Всего в этом языке над числами можно производить шесть различных математических операций, которые аналогичны соответствующим действиям в Turbo Pascal. Это четыре действия, известные из элементарной математики, а также операции целочисленного деления `div` и нахождения остатка от деления `mod`. Все операции пригодятся нам в дальнейшем при составлении кода программы, а работу над проектом мы начнем как обычно с создания его графического интерфейса.

В данном проекте интерфейс должен включать в себя следующие элементы:

- шесть текстовых окон. В первом и втором окнах будут вводиться исходные данные о первом моменте времени, в третьем и четвертом окнах — данные о втором моменте. В пятом и шестом окнах должен выводиться результат работы программы — разница между двумя моментами времени;
- поясняющие надписи к текстовым окнам — к каждой группе из двух окон на экране должно выводиться по три надписи. Первая из надписей к группе поясняет, что за информация содержится в данной паре окон. Вторая и третья надписи в группе указывают на используемые единицы измерения

времени — минуты и часы. Общее количество поясняющих надписей равно девяти;

- три экранные кнопки. Первая из них будет производить процесс вычисления результата. Вторая должна выводить информацию об авторе. Третья кнопка закрывает программу;
- панель, на которой находится рисунок, изображающий часы (данный рисунок является декоративным элементом программы);
- надпись, содержащая сведения об авторе программы.

Создание интерфейса начинаем как обычно с настройки основной формы. В начале изменяем заголовок формы. Для этого справа от свойства `Caption` ставим его значение `Хронометр`. Затем изменяем фоновый цвет формы. Для этого щелкаем раскрывающийся список справа от свойства `Color` и выбираем из списка ярко-голубой цвет — `ClAqua`.

Затем размещаем в форме первую группу текстовых окон с надписями. Это будут объекты `Edit1` (сюда окно будет введено количество часов для первого момента времени) и `Edit2` (количество минут для первого момента времени). Для того чтобы лучше смотрелась вводимая в окна информация, мы изменим характеристики шрифта, для чего работаем со свойством `Font` данных объектов. Щелкнув кнопку-построитель, открываем диалоговое окно для настройки шрифта и увеличиваем его размер до 10 пунктов, а начертание делаем полужирным.

Над описанными ранее текстовыми окнами помещаем надпись `Label1`, заголовок которой говорит о том, что в эти окна вводятся данные о первом моменте времени. Справа от каждого из окон размещаем надписи `Label2` и `Label3`, заголовки которых содержат информацию о единицах измерения времени, используемых в каждом из окон. Заголовок `Label2` — `час`, а `Label3` — `мин`. Для всех надписей мы изменяем характеристики шрифта — увеличиваем его размер до 12 пунктов и делаем его полужирным.

Под первой группой текстовых окон и надписей размещаем вторую группу для ввода данных о втором моменте времени. Эта группа содержит окна `Edit3` и `Edit4` и надписи `Label4`, `Label5` и `Label6`. Настройки этих окон и надписей аналогичны соответствующим настройкам для первой группы. Под второй группой оставляем немного свободного места для размещения экранной кнопки.

Третья группа окон и надписей размещается под второй и содержит два текстовых окна `Edit5` и `Edit6`, служащие для вывода результатов работы программы. Над окнами должна находиться поясняющая надпись следующего содержания: `Разница между двумя моментами времени составляет`. Однако эта надпись слишком длинна для того, чтобы поместиться в форме в одну строку.

Поэтому в данном случае придется воспользоваться следующим приемом: разбить длинную надпись на две более коротких и вывести их одна под одной. Таким образом, пользователь программы увидит на экране текст, состоящий из двух строк, которым будут соответствовать надписи `Label7` и `Label8`. Кроме того, справа от текстовых окон будут располагаться еще две вспомогательные надписи: `Label9` и `Label10`, аналогичные таким же надписям в предыдущих двух группах. Настройки окон и надписей также производятся аналогично соответствующим настройкам в предыдущих группах.

В правой верхней части формы мы расположим панель с находящимся на ней рисунком. Для этого мы создаем в форме объект `Panel1`, на котором размещаем объект `Image1`. Настройка данных объектов сводится к подгонке расположения и размеров рисунка (объект `Image1`) к расположению и размерам панели (объект `Panel1`). Мы производим эту операцию аналогично тому, как мы это делали при работе над проектом "параллелепипед". Единственное существенное отличие заключается в следующем: мы не будем рисовать загружаемую картинку вручную, а воспользуемся уже готовой. Для этого в папке Clipart, находящейся в пакете Microsoft Office 97, мы найдем файл Clock.wmf, содержащий изображение часов, и загрузим его в объект `Image1` описанным ранее способом.

Последней группой элементов интерфейса, которую необходимо разместить в форме, являются экранные кнопки. Первая из этих кнопок `Button1` отвечает за выполнение расчетов по определению искомой разницы. Вторая кнопка `Button2` должна выводить на экран формы сведения об авторе программы. Сведения будут выводиться на экран в виде надписи. Так как данная надпись вновь оказывается слишком длинной для вывода ее на экран в одну строку, то мы составляем ее из двух объектов: `Label11` и `Label12`, расположенных друг под другом. Так как непосредственно после запуска программы на выполнение надпись со сведениями об авторе не должна быть видна на экране, то производим настройку объектов `Label11` и `Label12`, которая заключается в том, что для свойства `Visible` каждого из этих объектов устанавливаем значение `False`. В результате подобной настройки по умолчанию эти объекты не будут видны. Завершаем создание интерфейса помещением на форму объекта `Button3` — кнопки, производящей закрытие программы. Интерфейс данной программы представлен на рис. 14.5.

Теперь приступаем к написанию кода для данной программы. Начнем этот процесс с описания действий, производимых при нажатии кнопки `Button1`. В начале необходимо разобрать ход производимых вычислений, что позволит нам определить, какие переменные понадобятся при написании кода. Как читатель помнит, все используемые в процедуре переменные необходимо описать после служебного слова `var`. Во-первых, нам нужно описать переменные

для исходных величин. Количество часов для первого момента времени обозначим `c1`, а количество минут для первого момента времени обозначим `m1`. Количество часов и минут для второго момента времени назовем соответственно `c2` и `m2`. Этих величин в готовом виде у нас нет, но можно воспользоваться стандартной функцией `StrtoInt`, которая преобразует текстовые величины в числовые. Тогда операторы, преобразующие значения, введенные пользователем в текстовые окна `Edit1`, `Edit2`, `Edit3` и `Edit4`, в числовые переменные, будут выглядеть так:

```
c1:=StrtoInt(Edit1.Text);
m1:=StrtoInt(Edit2.Text);
c2:=StrtoInt(Edit3.Text);
m2:=StrtoInt(Edit4.Text);
```

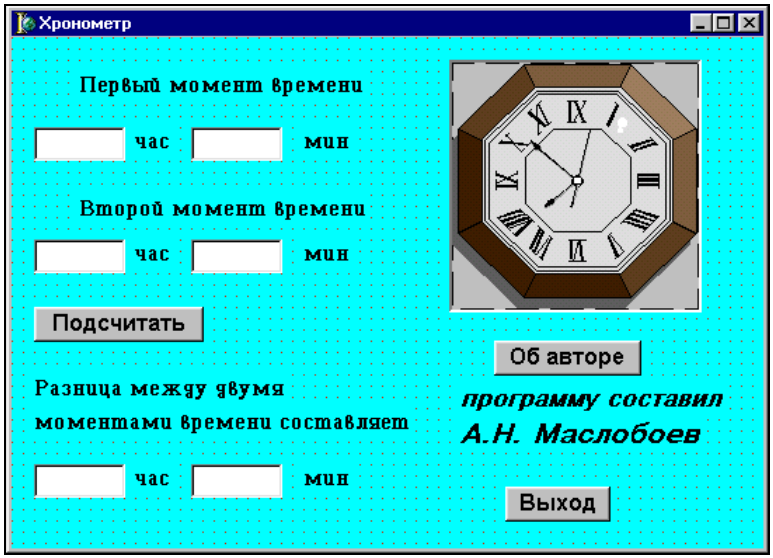


Рис. 14.5. Графический интерфейс программы "Хронометр"

Дальнейший ход решения задачи будет выглядеть следующим образом. Переведем первый момент времени из часов и минут только в минуты. Для этого нужно количество часов для первого момента умножить на 60 и к нему прибавить количество минут. Величину первого момента времени, выраженную только в минутах, обозначим `t1`. Теперь таким же образом преобразуем второй момент времени, а соответствующую величину обозначим `t2`. В программе эти преобразования будут выражены следующими операторами:

```
t1:=c1*60+m1;
t2:=c2*60+m2;
```

Для того чтобы найти величину `t3` — разницу между этими моментами времени, достаточно вычесть из второго момента первый, что делает оператор:

```
t3:=t2-t1;
```

Мы получили искомый результат, но выраженный только в минутах. Желательно этот результат представить, так же как и исходные данные в часах и минутах. Получить количество целых часов, которые содержатся в этой разнице, можно использовав целочисленное деление, т. е. разделив величину `t3` на 60 с помощью операции `div`. Обозначим найденную величину через `c3`. Для определения количества оставшихся минут `m3` мы прибегнем к операции `mod`, которая, как известно, находит остаток при целочисленном делении. В качестве делителя будет снова выступать число 60. Таким образом, операторы, вычисляющие количество часов и минут в конечном результате, будут выглядеть так:

```
c3:=t3 div 60;
m3:=t3 mod 60;
```

Искомые величины определены, но необходимо их преобразовать в текст, который будет отображаться в текстовых окнах `Edit5` и `Edit6`. Это действие, обратное тому, что мы производили в начале процедуры, можно реализовать с помощью функции `InttoStr`. Операторы, осуществляющие преобразование, будут выглядеть так:

```
Edit5.Text:=InttoStr(c3);
Edit6.Text:=InttoStr(m3);
```

Теперь работа над процедурой `TForm1.ButtonClick`, описывающей реакцию на нажатие кнопки **Подсчитать разницу**, завершена и целиком эта процедура будет выглядеть следующим образом:

```
procedure TForm1.Button1Click(Sender: TObject);
var c1,m1,c2,m2,c3,m3,t1,t2,t3:integer;
begin
c1:=StrtoInt(Edit1.Text);
m1:=StrtoInt(Edit2.Text);
c2:=StrtoInt(Edit3.Text);
m2:=StrtoInt(Edit4.Text);
t1:=c1*60+m1;
t2:=c2*60+m2;
t3:=t2-t1;
c3:=t3 div 60;
m3:=t3 mod 60;
Edit5.Text:=InttoStr(c3);
Edit6.Text:=InttoStr(m3);
end;
```

Следующим шагом в работе над программой должно стать создание процедуры, которая описывает реакцию на нажатие кнопки `Button2` — сведения об авторе. Действие, описываемое в данной процедуре, должно заключаться в том, что невидимая по умолчанию надпись со сведениями об авторе должна становиться видимой. Напомним, что эта надпись фактически состоит из двух с именами `Label11` и `Label12`. Превращение невидимого объекта в видимый, как уже известно читателю, производится путем присваивания свойству `Visible` данного объекта значения `True` (вместо имевшегося по умолчанию значения `False`). Следовательно, процедура, описывающая эти действия, будет выглядеть так:

```
procedure TForm1.Button2Click(Sender: TObject);
begin
Label11.Visible:=True;
Label12.Visible:=True
end;
```

Последнее, что осталось сделать, — это записать команду закрытия программы в процедуру, описывающую реакцию на нажатие кнопки **Выход**. Процедура эта будет выглядеть так:

```
procedure TForm1.Button3Click(Sender: TObject);
begin
Close
end;
```

Целиком же вся программа должна выглядеть следующим образом — листинг 14.3.

Листинг 14.3. Текст программы "Хронометр"

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, ExtCtrls;

type
  TForm1 = class(TForm)
    Edit1: TEdit;
    Edit2: TEdit;
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
```

```
    Label4: TLabel;
    Edit3: TEdit;
    Edit4: TEdit;
    Label5: TLabel;
    Label6: TLabel;
    Label7: TLabel;
    Label8: TLabel;
    Edit5: TEdit;
    Edit6: TEdit;
    Label9: TLabel;
    Label10: TLabel;
    Button1: TButton;
    Panel1: TPanel;
    Image1: TImage;
    Button2: TButton;
    Label11: TLabel;
    Label12: TLabel;
    Button3: TButton;
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure Button3Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.Button1Click(Sender: TObject);
var c1,m1,c2,m2,c3,m3,t1,t2,t3:integer;
begin
c1:=StrtoInt(Edit1.Text);
m1:=StrtoInt(Edit2.Text);
c2:=StrtoInt(Edit3.Text);
m2:=StrtoInt(Edit4.Text);
t1:=c1*60+m1;
t2:=c2*60+m2;
t3:=t2-t1;
c3:=t3 div 60;
```

```
m3:=t3 mod 60;
Edit5.Text:=InttoStr(c3);
Edit6.Text:=InttoStr(m3);
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
Label11.Visible:=True;
Label12.Visible:=True
end;

procedure TForm1.Button3Click(Sender: TObject);
begin
Close
end;

end.
```

Обратите внимание на то, что список использованных в программе элементов графического интерфейса, а также список использованных в программе процедур система программирования создает автоматически.

После запуска готовой программы на выполнение она будет производить теперь все требуемые от нее действия, причем нажатие экранных кнопок можно выполнять в произвольном порядке и реакция на них будет правильной. Пример того, как может выглядеть экран программы во время работы, показан на рис. 14.6.

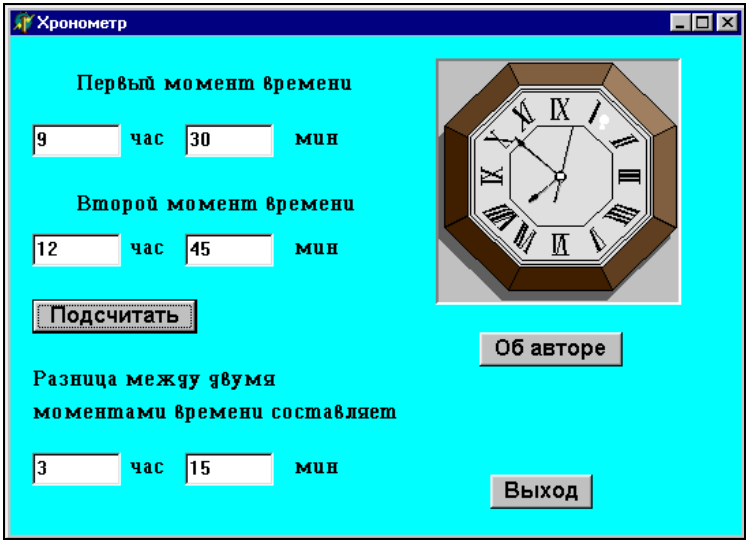


Рис. 14.6. Программа "Хронометр" во время работы

14.4. Программа "Цилиндр"

Во всех рассмотренных нами до сих пор задачах мы работали только с целочисленными переменными, значениями которых могут быть лишь целые числа. На практике же чаще всего полученные при расчетах значения выражаются дробными, вещественными числами. Вещественные результаты часто получаются и тогда, когда исходными данными при вычислениях являются целые числа. Пример подобной задачи — это вычисление объема и площади поверхности цилиндра по его высоте и радиусу основания. Даже в том случае, если высота и радиус основания являются целыми числами, полученные результаты заведомо будут вещественными, т. к. для их вычисления используется число Пи, которое, как известно, является бесконечной дробью.

На примере этой задачи мы освоим работу с вещественными переменными, а также научимся более рационально разрабатывать графический интерфейс программы. Интерфейс данной программы будет включать следующие элементы:

- два текстовых окна для ввода исходных данных (высоты и радиуса цилиндра) и две поясняющие надписи к этим окнам;
- два текстовых окна для вывода полученных результатов (объема и площади поверхности цилиндра) и две надписи с информацией о содержимом этих окон;
- панель с рисунком, на котором должен быть изображен геометрический объект, параметры которого вычисляются в программе;
- три экранные кнопки. Щелчок на первой из этих кнопок приводит к вычислению искомых величин. Щелчок на второй кнопке будет выводить на экран компьютера информацию об авторе программы. Щелчок на третьей кнопке закрывает программу;
- дополнительное окно с информацией об авторе программы и кнопкой, которая закрывает это окно.

Как видно из перечисления элементов интерфейса, в целях экономии места на основной форме мы вводим дополнительную форму с информацией об авторе, которая может выводиться на экран и закрываться при отсутствии надобности в ней. Как создавать дополнительную форму, мы разберем позже, а сейчас рассмотрим процесс создания графического интерфейса основной формы.

Основная форма получит название `Цилиндр`, для чего мы изменим свойство `Caption` объекта `Form1`, цвет же формы в данном случае оставим без изменения. На форме помещаем текстовые окна `Edit1` и `Edit2`, в которые будут вводиться данные соответственно о высоте цилиндра и радиусе его основания. Объекты `Label1` и `Label2` будут содержать поясняющие надписи к содержа-

нию данных окон. Далее на форме размещаем объекты Edit3 и Edit4, которые будут содержать результаты вычислений. В окне Edit3 будет выводиться информация о вычисленном объеме цилиндра, а в окне Edit4 — данные о найденной площади его поверхности. Объекты Label3 и Label4 подобно предыдущим двум надписям служат пояснениями к окнам с соответствующими номерами.

В нижней части экрана расположим три кнопки: Button1, Button2 и Button3. При нажатии на кнопку Button1 производятся вычисления объема и площади поверхности параллелепипеда по исходным данным, введенным в окна Edit1 и Edit2, а полученные результаты отображаются в окнах Edit3 и Edit4. При нажатии на вторую кнопку Button2 на экран выводится дополнительное окно, содержащее сведения об авторах программы. При нажатии на кнопку Button3 основная форма закрывается, и работа программы на этом завершается.

В правой части экрана программы должен находиться компонент Panel1, на котором размещается объект Image1, т. е. иллюстрация к данной программе, изображающая цилиндр. Рисунок, который изображает цилиндр, мы вначале создадим с помощью графического редактора Paint и разместим его в папке, содержащей проект "цилиндр", а потом под размеры созданного рисунка подгоним размеры панели и иллюстрации.

На этом разработку графического интерфейса основной формы проекта можно считать завершенной (рис. 17.7). Далее мы должны перейти к созданию кода, описывающего действия различных объектов формы. Однако, делая это, необходимо помнить, что величины, используемые в данной программе, являются вещественными. Вещественные же величины имеют определенные особенности при их использовании в программе.

Для описания вещественных величин используется специальный тип Real, аналогичный такому же типу в Turbo Pascal. Подобно преобразованию строковых величин в целые и обратно нам придется производить подобные действия и с вещественными числами. Для преобразования строковой величины в вещественную используется стандартная функция языка Object Pascal StrtoFloat. Для обратного преобразования используется стандартная функция FloattoStr. Теперь мы готовы к написанию кода данной программы.

Начнем написание программы с описания реакции на нажатие кнопки Button1. Эта кнопка позволяет рассчитать объем и площадь поверхности цилиндра по имеющимся исходным данным. Обозначим высоту цилиндра h, а радиус его основания — r. Искомые величины обозначим следующим образом: объем цилиндра — v, а общую площадь его поверхности — s. Все эти величины должны быть описаны в разделе описаний подпрограммы Tform1.Button1Click таким образом:

```
var h,r:integer; v,s:real;
```

т. е. исходные данные описываются как величины целого типа, а результаты как вещественные величины.

Для того чтобы исходные данные были представлены в числовой форме, мы должны преобразовать в числовой (в данном случае — целый) вид, те данные которые пользователь вводит в окно `Edit1` и `Edit2` в строковой форме. Это преобразование будет выполнено с помощью операторов:

```
h:=StrtoInt(Edit1.Text);
r:=StrtoInt(Edit2.Text);
```

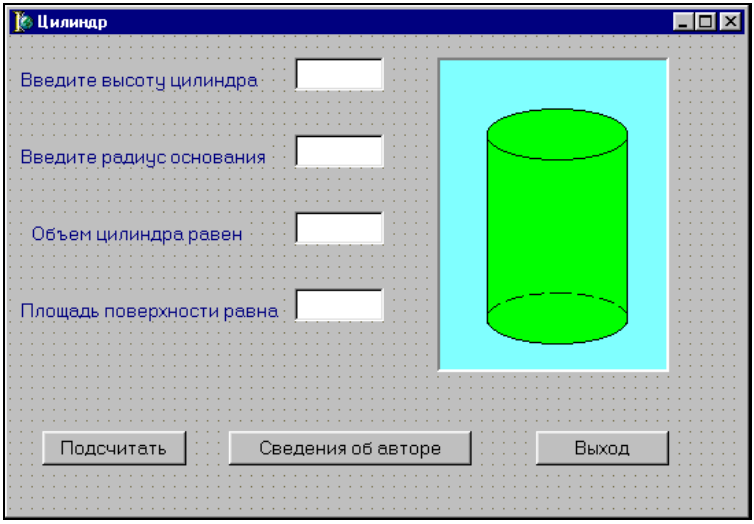


Рис. 14.7. Графический интерфейс проекта "Цилиндр"

Следующим шагом станет вычисление значений искомых величин. Как известно из стереометрии, объем цилиндра определяется как произведение высоты цилиндра на площадь его основания, которая в свою очередь вычисляется как площадь круга с радиусом `r`. Площадь же круга, как известно, представляет собой квадрат его радиуса, умноженный на число Пи.

Для возведения радиуса в квадрат мы воспользуемся стандартной функцией `sqr`, имеющейся в языке Object Pascal. Для определения значения числа Пи мы используем функцию `pi`, также входящую в состав языка Object Pascal. Эта функция в отличие от предыдущей не имеет аргумента, а значением ее является определяемое компьютером с высокой степенью точности число Пи. Следовательно, площадь основания можно представить следущим образом: `pi*sqr(r)`. Исходя из всего сказанного ранее, на языке Object Pascal вычисление объема цилиндра можно записать следующим образом:

```
v:=pi*sqr(r)*h;
```

Следующая величина, которую нам необходимо определить, — это площадь поверхности цилиндра. Общая площадь складывается из трех площадей: площади боковой поверхности цилиндра, площади его верхнего основания и площади нижнего основания, причем последние две величины равны между собой. Площадь одного основания, как мы уже знаем, можно записать как `pi*sqr(r)`, а сумму площадей запишем как `2*pi*sqr(r)`. Можно ту же сумму записать и по-другому: `2*pi*r*r`. Площадь же боковой поверхности цилиндра вычисляется как произведение длины окружности основания на высоту цилиндра. Длину окружности можно записать как `2*pi*r`, а боковая поверхность цилиндра тогда определяется по формуле `2*pi*r*h`. Таким образом, общая площадь поверхности цилиндра будет равна `2*pi*r*r+2*pi*r*h`. Вынося общие множители за скобки, получим следующее выражение: `2*pi*r*(h+r)`. Теперь осталось присвоить значение этого выражения искомой величине `s`:

```
s:=2*pi*r*(h+r);
```

Для того чтобы преобразовать полученные вещественные величины в строковые, которые будут отображаться в текстовых окнах `Edit3` и `Edit4`, следует воспользоваться стандартной функцией `FloattoStr`. Операции преобразования будут записаны в программе следующим образом:

```
Edit3.Text:=FloattoStr(v);
Edit4.Text:=FloattoStr(s)
```

Целиком подпрограмма, описывающая действия, осуществляемые при нажатии кнопки `Button1`, будет выглядеть так:

```
procedure TForm1.Button1Click(Sender: TObject);
var h,r:integer; v,s:real;
begin
h:=StrtoInt(Edit1.Text);
r:=StrtoInt(Edit2.Text);
v:=pi*sqr(r)*h;
s:=2*pi*r*(h+r);
Edit3.Text:=FloattoStr(v);
Edit4.Text:=FloattoStr(s)
end;
```

Следующая подпрограмма, которую мы создадим, должна описывать реакцию на нажатие кнопки `Button2`. Результатом такого нажатия должен стать вывод дополнительного окна, которое содержит сведения об авторе или авторах данной программы. Для того чтобы это правильно сделать, нужно знать последовательность действий, выполняемых при подключении к проекту дополнительного окна.

Создание дополнительного окна и работа с ним включает в себя следующие этапы:

1. Создание заготовки для нового окна. Такая заготовка представляет собой дополнительную форму. Для подключения к проекту новой формы необходимо в главном меню системы программирования открыть раздел **File**, в разделе **File** найти команду **New**. Если задержать указатель мыши на несколько секунд на данной команде, то справа от нее появится список элементов, которые можно добавить в проект (в 5 версии нужно один раз щелкнуть команду **New** мышью). Выберем из списка вариант **Form**. После этого в проекте появится дополнительная форма. В нашем случае это будет `Form2`.

2. Вставка оператора в основной программный модуль. В основном программном модуле (этот модуль, как правило, называется `Unit1`) в подпрограмму, описывающую работу кнопки или другого элемента, открывающего дополнительное окно, вставить оператор следующего вида:

   ```
   FormN.Show;
   ```

   где `N` — порядковый номер открываемого дополнительного окна. В нашем случае этот оператор будет выглядеть так:

   ```
   Form2.Show;
   ```

   То, что находится в данном операторе после точки, называется методом объекта. *Метод* — это подпрограмма, которая изменяет свойства связанного с ней объекта. В нашем примере метод `Show` изменяет свойства вспомогательной формы, которая из невидимой становится видимой. Целиком же подпрограмма, описывающая работу второй кнопки в нашем проекте, будет выглядеть следующим образом:

   ```
   procedure TForm1.Button2Click(Sender: TObject);
   begin
   Form2.Show;
   end;
   ```

3. Создание графического интерфейса подключаемого окна. Для того чтобы вывести на экран необходимую форму, поступаем следующим образом: входим в главное меню системы программирования и выбираем в нем раздел **View**. В этом разделе находим команду **Forms**. После выполнения данной команды на экран компьютера выводится диалоговое окно, которое содержит список всех форм, используемых в данном проекте. В нашем случае поиск нужной формы не составит никаких затруднений, т. к. в проекте всего две формы. Выбираем вторую из них, после чего данная форма будет выведена на передний план. Сам же процесс создания графического интерфейса аналогичен соответствующему процессу для основной формы.

По завершении работы над графическим интерфейсом дополнительное окно можно закрыть, щелкнув стандартную закрывающую кнопку, расположенную в правом верхнем углу окна.

4. Разработка кода, описывающего действия, производимые пользователем в дополнительном окне. Данный код будет содержаться в дополнительном модуле `Unit2`. Для того чтобы вывести на экран текст данного модуля, нужно открыть раздел **View** главного меню системы программирования, в данном разделе найти команду **Units**. После щелчка на этой команде открывается диалоговое окно со списком модулей, в котором выбираем нужный нам (в данном случае — это `Unit2`).

   Сам по себе процесс написания кода аналогичен соответствующему процессу для основного окна, но нужно учесть, что модуль `Unit2` должен быть подключен к основному модулю `Unit1`. Данное подключение производится так: в основной части программы после заголовка указывается список используемых в данной программе дополнительных модулей. Этот список начинается со служебного слова `Uses`, после которого через запятую перечисляются модули. В этот список следует добавить модуль `Unit2`. Тогда список будет выглядеть следующим образом:

```
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls,
  Forms, Dialogs, ExtCtrls, StdCtrls, Menus, Unit2;
```

После выполнения всех указанных ранее операций дополнительное окно становится полноценной частью проекта.

Следующая кнопка `Button3`, которую мы используем в основной форме, закрывает эту форму. Данная операция производится с помощью команды `Close`, а текст подпрограммы, описывающей это действие, будет выглядеть так:

```
procedure TForm1.Button3Click(Sender: TObject);
begin
Close
end;
```

На этом мы завершаем написание программного модуля `Unit1` для основной формы. Целиком данный модуль будет выглядеть следующим образом — листинг 14.4.

Листинг 14.4. Текст программы "Цилиндр" (первый модуль)

```
unit Unit1;

interface
```

```
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, ExtCtrls, StdCtrls, Menus, Unit2;

type
  TForm1 = class(TForm)
    Label1: TLabel;
    Edit1: TEdit;
    Label2: TLabel;
    Edit2: TEdit;
    Label3: TLabel;
    Edit3: TEdit;
    Label4: TLabel;
    Edit4: TEdit;
    Button1: TButton;
    Button2: TButton;
    Button3: TButton;
    Panel1: TPanel;
    Image1: TImage;
    procedure Button1Click(Sender: TObject);
    procedure Button3Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.Button1Click(Sender: TObject);
var h,r:integer; v,s:real;
begin
h:=StrtoInt(Edit1.Text);
r:=StrtoInt(Edit2.Text);
v:=pi*sqr(r)*h;
s:=2*pi*r*(h+r);
Edit3.Text:=FloattoStr(v);
Edit4.Text:=FloattoStr(s)
end;
```

```
procedure TForm1.Button2Click(Sender: TObject);
begin
Form2.Show;
end;

procedure TForm1.Button3Click(Sender: TObject);
begin
Close
end;

 end.
```

Теперь нам осталось создать графический интерфейс и написать код для вспомогательного окна со сведениями об авторе проекта. Вначале переименуем саму форму `Form2`, изменив значение ее свойства `Caption` на `Сведения об авторе`. Цвет формы мы изменять не будем, на форме же создадим следующие объекты: две надписи `Label1` и `Label2` со сведениями об авторе и закрывающую кнопку `Button1`. Единственный объект, выполняющий действия в данном окне — это кнопка `Button1`. Для нее мы и напишем код, состоящий из одного оператора `Close`. Целиком же программный модуль `Unit2` будет выглядеть следующим образом — листинг 14.5.

Листинг 14.5. Текст программы "Цилиндр" (второй модуль)

```
unit Unit2;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls;

type
  TForm2 = class(TForm)
    Label1: TLabel;
    Label2: TLabel;
    Button1: TButton;
    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
```

```
var
  Form2: TForm2;

implementation

{$R *.dfm}

procedure TForm2.Button1Click(Sender: TObject);
begin
Close
end;
end.
```

После написания кода для второго окна работу над проектом можно считать завершенной. Результаты работы программы можно увидеть на рис. 14.8.

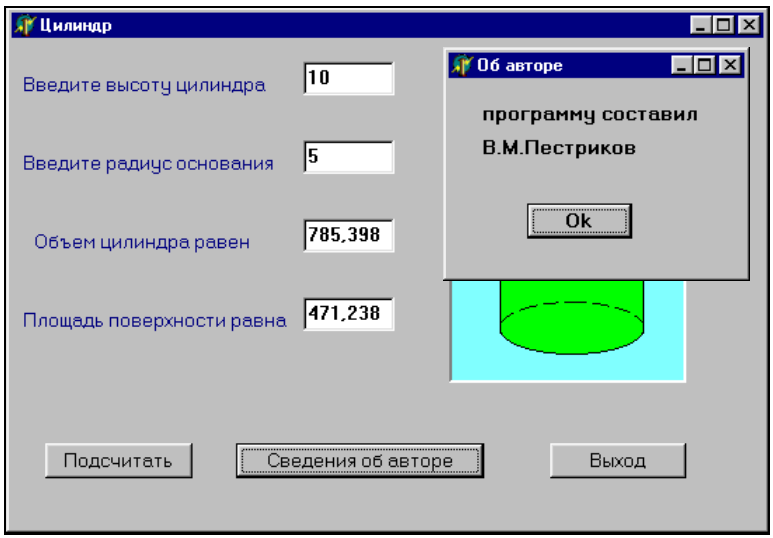


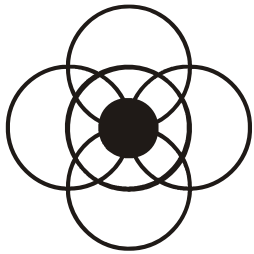
Рис. 14.8. Программа "Цилиндр" во время работы

Задание для самостоятельной работы

Разработать в системе Delphi проект "Банкомат", взяв за основу соответствующую программу на языке Turbo Pascal, которая описана в *главе 3*. В интерфейсе формы данной программы должны присутствовать следующие элементы: текстовое окно для ввода суммы, которую хочет получить пользователь, поясняющая надпись к этому окну и экранная кнопка для ввода суммы.

Под окном ввода должны располагаться 4 текстовых окна с сообщениями о количестве купюр по 1000, 500, 100 и 50 рублей, которыми должна быть выдана сумма, и поясняющими надписями к ним. В правом нижнем углу формы должна быть предусмотрена кнопка для выхода из программы. В правой верхней части формы желательно поместить рисунок. В качестве иллюстрации к программе можно использовать графический файл Money.wmf из коллекции Clipart, входящей в состав пакета Microsoft Office 97.

ГЛАВА 15



Программы с ветвлением в Delphi

Все программы, которые мы до сих пор разрабатывали в среде Delphi, состояли из операторов, выполняющихся последовательно друг за другом. Такого типа программы обычно называют программами линейной структуры, т. к. в них ход решения задачи представляет собой как бы прямую линию, ведущую от постановки задачи и ввода исходных данных к получению конечного результата. Однако это не единственно возможный вид программы. Существуют так называемые программы с ветвлением, в которых используются операторы организации переходов, позволяющие нарушить последовательный порядок выполнения инструкций программы.

15.1. Программа "Тест"

В языке Object Pascal, как и в Turbo Pascal, широко используются условные операторы. Синтаксис условного оператора `if` аналогичен синтаксису этого оператора в Turbo Pascal. Так же записываются в Object Pascal операции сравнения, используемые в условных операторах.

Условный оператор `if` широко используется в Object Pascal. Одной из областей его применения является составление различных программ-тестов, проверяющих знания пользователя. В *главе 4* была приведена программа, которая проверяет пользователя на знание даты первого полета человека в космос *(см. листинг 4.4)*. Взяв ее за основу, создадим тест на эту же тему в системе Delphi.

Графический интерфейс основной формы программы "Тест" будет включать в себя следующие компоненты: во-первых, окно комментария, содержащее вопрос, на который должен дать ответ тестируемый. Во-вторых, текстовое окно, куда тестируемый вводит свой вариант ответа. Слева от текстового окна должно находиться еще одно, вспомогательное окно комментария, пояс-

няющее, куда нужно вводить ответ. Справа от текстового окна поместим иллюстрацию к тесту, представляющую собой фотографию первого искусственного спутника Земли. Под текстовым окном должна располагаться экранная кнопка, нажатием на которую вводится ответ тестируемого пользователя. Ниже экранной кнопки должна быть расположена еще одна надпись, которая сообщает пользователю о том, правильно или неправильно ответил он на заданный вопрос.

Помимо кнопки **Ввод** в основной форме будут находиться еще две экранные кнопки: кнопка, закрывающая окно программы, и кнопка, нажатие на которую выводит на экран компьютера дополнительное окно со сведениями об авторе. Это дополнительное окно в свою очередь должно содержать окно комментария со сведениями об авторах или авторе программы и экранную кнопку, закрывающую это дополнительное окно.

Разработку графического интерфейса мы начнем как обычно с работы над самой формой. Изменяем заголовок формы — она будет называться `Тест`. Цвет формы изменяем на небесно-голубой. Размеры формы оставляем без изменений. В левой верхней части формы располагаем объект `Memo1` — окно комментария с текстом задаваемого вопроса.

Для создания этого объекта мы используем компонент `Memo`, который находится в палитре компонентов на вкладке **Standard** и представлен кнопкой  . Использование данного компонента удобнее, чем использование компонента `Label` в том случае, если на экран компьютера нужно выводить текст в несколько строк. Для того чтобы ввести текст в поле комментария, нужно найти у объекта `Memo` свойство `Lines` и щелкнуть справа кнопку-построитель. После указанных действий на экране на переднем плане появится окно **String List Editor**, что в переводе с английского означает "редактор строк". Затем в рабочей области окна мы построчно вводим текст комментария, при этом нужно не забывать в конце каждой строки нажимать клавишу <Enter>. По окончании ввода текста нужно щелкнуть экранную кнопку **OK**. Если требуется, чтобы окно с комментарием не выделялось на общем фоне формы, то нужно свойство `Color` объекта `Memo` привести в соответствие со свойством `Color` объекта `Form`. Затем для свойства `BorderStyle` (граница объекта) окна `Memo` нужно из раскрывающегося списка выбрать значение `bsNone` (отсутствие границы). Текст вопроса `В каком году был запущен первый искусственный спутник Земли?` вводим в поле комментария в три строки, а размер шрифта (используя свойство `Font`) увеличиваем до 12.

Под окном комментария размещаем текстовое окно `Edit1`. В него будет вводиться текст ответа. Для удобства контроля пользователя за вводимой им информацией размер текста в окне также увеличиваем до 12. Слева от объекта `Edit1` помещаем вспомогательное окно комментария `Memo2`, содержащее поясс-

нение о том, куда нужно вводить ответ. В этом окне информацию также вводим в три строчки. Для того чтобы текст поместился в указанном месте, размер шрифта принимаем равным 9. Цвет текста в этом окне изменяем на темно-синий.

Ниже окна `Edit1` помещаем экранную кнопку `Button1`. Заголовок этой кнопки изменяем на слово `Ввод`, а размер шрифта делаем равным 12. В правом нижнем углу основной формы будет расположена надпись `Label1`, информирующая пользователя о результатах теста. Эта надпись будет выводиться в одну строку шрифтом красного цвета. Начертание шрифта сделаем полужирным, а размер также примем равным 12. Сразу после начала работы программы эта надпись вообще не должна быть видна, т. к. было бы абсурдно, если бы пользователь узнал о результатах теста еще до его начала. Можно было бы найти свойство `Visible` объекта и придать ему значение `False`. Но такой путь усложнил бы написание кода. Поэтому поступим другим образом: найдем свойство `Caption` объекта `Label1` и очистим его. Тогда надпись после запуска программы не будет видна, хотя формально она будет считаться видимой.

Следующим объектом должна стать иллюстрация к тексту. Как мы это уже неоднократно делали раньше, вначале мы размещаем в форме объект `Panel1`, а уже затем поверх него располагаем `Image1`, т. е. непосредственно саму иллюстрацию. Иллюстрацию к данной программе, представляющую собой черно-белую фотографию первого спутника, авторы книги взяли из Интернета. Данная фотография находилась на Web-странице в формате JPEG. Графические изображения, созданные в данном формате, достаточно хорошо поддаются масштабированию, поэтому особых проблем, связанных с взаимной подгонкой размеров панели и самой иллюстрации, здесь возникать не должно.

Нам осталось разобраться еще с двумя элементами графического интерфейса — экранными кнопками `Button2` и `Button3`. Основные настройки для этих кнопок такие же, как и для кнопки `Button1`. Отличаться будут только заголовки кнопок: `Об авторах` для `Button2` и `Выход` для `Button3`. Нажатие кнопки `Button2` должно приводить к появлению на экране компьютера дополнительной формы с именем `Form2`. Последовательность создания такой формы и подключения ее к основному проекту была описана в предыдущей главе и поэтому не должна вызывать сложностей у пользователя.

Настройка дополнительной формы будет заключаться в том, что мы изменим ее заголовок на `Сведения об авторах`, а также уменьшим ее размеры, т. к. на ней будут расположены всего два объекта. Уменьшение размеров формы можно производить как путем перетаскивания мышью маркеров формы, так и посредством задания соответствующих величин для свойств `Height` и `Width`. Графические объекты, которые будут расположены на вспомогательной фор-

ме, — это окно комментария Memo1, которое непосредственно и будет содержать информацию об авторах программы, а также кнопка Button1, закрывающая вспомогательное окно. Текст комментария будет содержать надпись в две строки, имеющую размер шрифта, равный 10, а закрывающая кнопка должна иметь заголовок OK и размер шрифта, равный 12. Как должен выглядеть созданный нами графический интерфейс, можно увидеть на рис. 15.1.

Рис. 15.1. Графический интерфейс программы "Тест"

Далее переходим к написанию кода программы. Начнем мы с написания кода для основной программной формы. В первую очередь мы оформим процедуру, которая описывает реакцию на щелчок на кнопке Button1. После щелчка на этой кнопке должна выполняться проверка введенных пользователем данных. Строковая информация преобразуется в числовую с помощью стандартной функции StrtoInt. Введенная в окно Edit1 строковая информация присваивается целочисленной переменной k:

```
k:=StrtoInt(Edit1.Text);
```

Теперь ответ пользователя, содержащийся в переменной k, нужно проверить на совпадение с правильным ответом — числом 1957 (год запуска первого спутника). При совпадении двух этих ответов на экран выводится одна информация, при их несовпадении — другая, об ошибке. Такая проверка выполняется с помощью условного оператора if:

```
if k=1957
then Label1.Caption:='Вы ответили правильно'
else Label1.Caption:='Вы ошиблись'
```

Таким образом, целиком процедура, отслеживающая реакцию на нажатие кнопки **Ввод** после ввода ответа в текстовое окно, должна выглядеть так:

```
procedure TForm1.Button1Click(Sender: TObject);
var k:integer;
begin
k:=StrtoInt(Edit1.Text);
if k=1957
then Label1.Caption:='Вы ответили правильно'
else Label1.Caption:='Вы ошиблись'
end;
```

Следующая кнопка, для нажатия которой мы пишем код, — это кнопка `Button2`. В процедуру, описывающую реакцию на нажатие этой кнопки, вставляем всего одну команду:

```
Form2.Show;
```

При этом нужно не забыть, что в раздел `Uses`, находящийся в начале данного программного модуля, следует добавить ссылку на второй программный модуль, обслуживающий дополнительную форму `Form2`:

```
uses Unit2;
```

Наконец в процедуру, обслуживающую кнопку `Button3`, нужно вставить всего одно слово:

```
Close;
```

Теперь мы можем целиком обозреть получившийся программный модуль `Unit1`, содержащий указанные ранее элементы программного кода — листинг 15.1.

Листинг 15.1. Текст программы "Тест" (первый модуль)

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, jpeg, ExtCtrls;

type
  TForm1 = class(TForm)
    Memo1: TMemo;
    Memo2: TMemo;
    Edit1: TEdit;
```

```
    Button1: TButton;
    Label1: TLabel;
    Panel1: TPanel;
    Image1: TImage;
    Button2: TButton;
    Button3: TButton;
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure Button3Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

uses Unit2;

{$R *.dfm}

procedure TForm1.Button1Click(Sender: TObject);
var k:integer;
begin
k:=StrtoInt(Edit1.Text);
if k=1957
then Label1.Caption:='Вы ответили правильно'
else Label1.Caption:='Вы ошиблись'
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
Form2.Show
end;

procedure TForm1.Button3Click(Sender: TObject);
begin
Close
end;

end.
```

Последнее, что осталось сделать в плане написания кода, — это описать реакцию на нажатие кнопки `Button1`, расположенной на вспомогательной форме

`Form2`. Для этого нужно в списке программных модулей выделить модуль `Unit2` и в процедуру `TForm2.Button1Click` добавить команду `Close`. Полный текст модуля `Unit2` мы приводить не будем, т. к. он аналогичен такому же модулю из предыдущего проекта *(см. листинг 14.5)*. Внешний вид программы "тест" во время работы приведен на рис. 15.2.

Рис. 15.2. Программа "Тест" во время работы

15.2. Программа "Квадратное уравнение"

В примере, приведенном в предыдущем разделе, каждая из ветвей условного оператора `if` содержала только один оператор. Но часто возникает ситуация, когда при выполнении или невыполнении некоторого условия нужно произвести не одно какое-либо действие, а целый ряд действий. Тогда в программе на языке Object Pascal внутри условного оператора используется составной оператор. Как и в языке Turbo Pascal, в Object Pascal для выделения составного оператора используются операторные скобки `begin` и `end`.

В общем случае составной оператор может находиться внутри условного оператора и после служебного слова `then`, и после служебного слова `else`, и после них вместе. Следующей нашей задачей как раз и будет разработка проекта, в котором реализуется как раз последний вариант условного оператора. Результатом разработки должна стать программа решения квадратного уравнения вида:

$$ax^2 + bx + c = 0.$$

По введенным с клавиатуры исходным данным программа должна либо находить корни квадратного уравнения, либо выдавать сообщение о том, что данное уравнение не имеет решения. Напомним читателю алгоритм решения данного уравнения. В начале следует вычислить дискриминант уравнения, который определяется по формуле:

$$d = b^2 - 4ac.$$

Затем в зависимости от величины дискриминанта определяется возможность нахождения решения данного уравнения. Если дискриминант положителен или равен нулю, то уравнение имеет решение и можно найти два его корня $x1$ и $x2$ (в случае равенства дискриминанта нулю два корня будут равны между собой). Вычисление корней производится по следующим формулам:

$$x1 = \frac{-b - \sqrt{d}}{2a}; \qquad x2 = \frac{-b + \sqrt{d}}{2a}.$$

В случае же, если найденный дискриминант отрицателен, уравнение вообще не имеет решения. Как видно из сказанного ранее, данный алгоритм хорошо подходит для реализации с помощью условного оператора `if`.

Как всегда, работу над проектом мы начинаем с разработки интерфейса программы. Основное окно программы должно содержать три текстовых окна для ввода исходных данных — коэффициентов `a`, `b` и `c` квадратного уравнения. Слева от каждого из этих окон должна находиться надпись, которая поясняет, какой именно коэффициент вводится в то или иное окно. Таким образом, количество надписей в этой части основного окна также должно быть равно трем. Ниже текстовых окон для ввода исходных данных должны находиться два текстовых окна для вывода решения (конечно, в том случае, если такое решение имеется). Над этими текстовыми окнами также должна располагаться поясняющая надпись.

Между окнами с исходными данными и окнами с результатами в основной форме будут расположены две экранные кнопки. Слева должна находиться кнопка, нажатие на которую приводит к решению уравнения. Назовем эту кнопку **Найти решение**. Правее будет расположена кнопка **Сброс**, щелчок на которой очищает как текстовые окна с исходными данными, так и текстовые окна с результатами. Эта кнопка имеет двоякое предназначение. Во-первых, она может понадобиться в том случае, если пользователь допустил ошибку при вводе исходных данных. Пользователь может исправить ошибку, не закрывая окно программы, а только щелкнув мышью экранную кнопку. Во-вторых, если пользователь уже решил квадратное уравнение и хочет решить аналогичное уравнение, но уже при других значениях коэффициентов, ему также нет необходимости закрывать окно программы и запускать ее заново. Достаточно нажать кнопку **Сброс** и ввести затем эти новые коэффициенты.

Еще одним элементом основного окна программы, используемым для того, чтобы немного оживить его, станет иллюстрация юмористического характера, расположенная в правой части окна. Данная иллюстрация изображает персонажа, задумавшегося над решением сложной задачи.

Все перечисленные ранее элементы уже неоднократно использовались нами при составлении различных проектов. Но наряду с ними в этой программе будут использованы и новые элементы. Необходимость их введения объясняется следующим: для закрытия окна и вывода справочной информации мы ранее использовали специальные экранные кнопки. Это допустимо в том случае, если число элементов интерфейса в окне невелико, и, кроме того, в окне имеется достаточное количество свободного места для размещения кнопок. Если же в основном окне программы находится большое количество элементов интерфейса и к тому же ощущается дефицит свободного места в окне, то целесообразно поступить по-другому. В таком случае лучше организовать в верхней части окна программы специальное меню, с помощью пунктов которого можно производить различные операции с файлами (включая закрытие программы), получать необходимую информацию справочного характера и решать многие другие задачи. В системе программирования Delphi имеются все необходимые средства для создания такого меню.

Как обычно, непосредственную работу по созданию интерфейса программы начнем с работы над самой формой. Изменим заголовок основной формы. Вместо `Form1` она будет называться `Решение квадратного уравнения`. Кроме того, изменим цвет формы, сделав его бледно-зеленым.

Затем приступим к созданию текстовых окон для ввода исходных данных с сопровождающими их надписями. Текстовые окна для ввода данных получат наименования `Edit1`, `Edit2` и `Edit3`. Соответствующие надписи получат имена `Label1`, `Label2` и `Label3`. В окно `Edit1` будут вводиться данные о коэффициенте `a`. Поле справа от свойства `Text` данного объекта предварительно очистим, т. к. в начале работы программы текстовое окно должно быть пустым. Шрифт для данного объекта (свойство `Font`) мы изменим, сделав его полужирным и увеличив его размер до 10 пунктов. Аналогичным образом настроим окна `Edit2` и `Edit3`, которые используются для ввода коэффициентов `b` и `c`.

Настройка надписи `Label1` включает в себя следующие действия. Во-первых, мы изменим заголовок надписи (свойство `Caption`) на следующий: `Введите коэффициент a`. Во-вторых, изменим шрифт надписи. Подобно шрифту текстового окна сделаем шрифт полужирным и равным 10 пунктам. Подобные же настройки произведем и для объектов `Label2` и `Label3`. При этом содержание заголовка для `Label2` будет `Введите коэффициент b`, а для `Label3` соответственно `Введите коэффициент c`.

Следующими элементами интерфейса программы станут текстовые окна Edit4 и Edit5, которые используются для вывода корней уравнения x1 и x2. Настройка этих объектов большей частью аналогична настройке предыдущих текстовых окон, т. е. мы очищаем свойство Text, а шрифт для обоих текстовых окон делаем полужирным и равным 10 пунктам.

Есть, однако, еще момент, который надо учитывать при настройке этих окон. Дело в том, что уравнение, как мы уже знаем, может и не иметь решения. В таком случае наличие в окне программы текстовых окон с корнями уравнения лишено какого-либо смысла. Поэтому поступим с этими окнами следующим образом: изначально сделаем их невидимыми, а в том случае, если решение уравнения будет найдено, эти окна проявятся, и мы увидим в них найденный результат. Для того чтобы настроить окна соответствующим образом, мы найдем для каждого из элементов Edit4 и Edit5 свойство Visible и установим для него значение False (ложно), которое скроет их от глаз пользователя. На этом настройку текстовых окон можно считать завершенной.

Надпись Label4 должна содержать информацию об итогах решения уравнения (как при наличии корней, так и при их отсутствии). Поскольку эта надпись в любом случае должна появляться только после ввода исходных данных и щелчке на кнопке **Найти решение**, сделаем ее изначально невидимой, для чего свойству Visible данного объекта присвоим значение False. Заголовок данной надписи изменим на Корни уравнения, а шрифт заголовка сделаем полужирным и равным 10 пунктам. При отсутствии решения заголовок надписи будет иным, но соответствующее изменение заголовка мы обеспечим при написании кода.

Между верхней и нижней группой надписей расположим две экранные кнопки: Button1 и Button2. Для кнопки Button1 мы изменим заголовок, который будет теперь следующим: Найти решение. Размеры кнопки увеличим таким образом, чтобы данный заголовок был полностью виден. Шрифт заголовка сделаем полужирным и равным 12 пунктам. Аналогичные настройки шрифта мы произведем и для расположенной правее кнопки Button2. Заголовок же этой кнопки будет состоять из одного слова: Сброс.

В правой части окна поместим юмористическую иллюстрацию, которая изображает раздумья мультипликационного персонажа, ищущего решение поставленной перед ним задачи. Для размещения иллюстрации мы используем, как обычно, объект Panel1, на котором будет помещен объект Image1, т. е. сама иллюстрация. Картинку мы возьмем из коллекции рисунков, входящих в состав пакета Microsoft Office 97. К данной коллекции мы уже неоднократно обращались за сюжетами для иллюстраций. Искомая иллюстрация находится, как читатель уже знает, по адресу Program Files\Clipart\Popular. Сам же файл с данной иллюстрацией называется Amconfus и имеет расширение wmf. Иллю-

страция, содержащаяся в таком файле, хорошо поддается масштабированию, поэтому не требуется производить особо тщательную взаимную подгонку размеров панели и соответствующей ей иллюстрации.

Последним объектом, который осталось разместить на основной форме проекта, является главное меню программы. Этот новый для нас объект имеет более сложную структуру, нежели уже известные нам объекты. Меню может включать в свой состав ряд разделов, каждый из которых в свою очередь может состоять из одного или нескольких пунктов. Каждый из разделов и пунктов меню является особым объектом интерфейса, входящим в состав другого объекта — самого меню.

Если вы посмотрите на список объектов, входящих в состав проекта (он находится в окне **Object TreeView**), то увидите, что рядом с некоторыми объектами находится значок в виде плюса. Такой значок говорит о том, что данный объект является составным, и этот значок можно увидеть, в частности, рядом с объектом `MainMenu1` (главное меню). Если щелкнуть по значку "плюс", то можно увидеть список объектов, содержащихся в составном объекте. При этом значок "плюс" изменяется на "минус". Если щелкнуть на значке "минус", то составной объект снова свернется, и "минус" сменится на "плюс". Объекты, входящие в составной объект, сами в свою очередь тоже могут быть составными. На рис. 15.3 показан пример того, как может выглядеть список объектов проекта, где некоторые объекты являются составными.

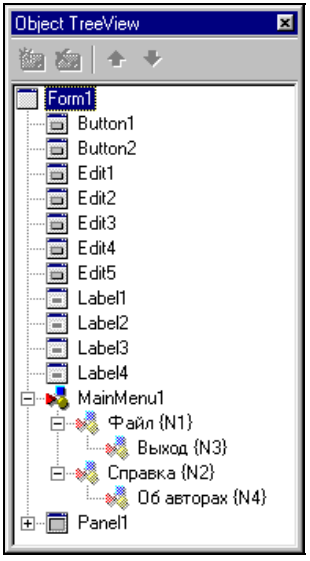
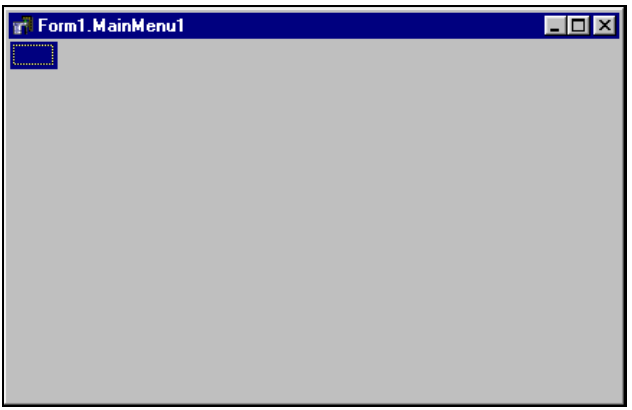
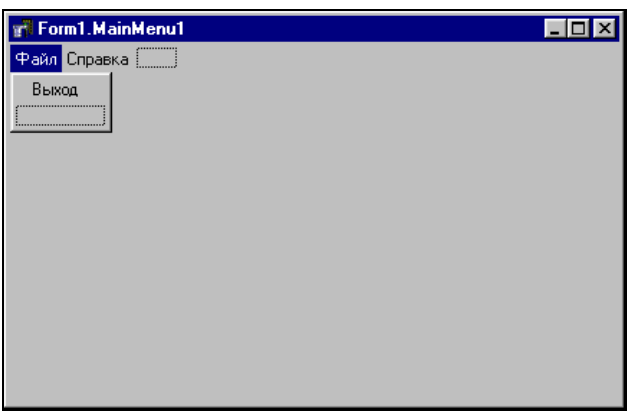


Рис. 15.3. Список объектов проекта, содержащий составные объекты `MainMenu1` и `Panel1` (составной объект `MainMenu1` представлен в развернутом виде)

Порядок создания главного меню программы следующий: вначале на панели компонентов на вкладке **Standard** нужно найти компонент `MainMenu`, который выглядит таким образом:  . Выделив данный компонент, щелкаем мышью в верхней части формы, в результате чего такой же значок появляется на форме, т. е. мы получаем заготовку для создания будущего меню. Для заполнения меню разделами и пунктами необходимо дважды щелкнуть значок  (расположенный в форме), в результате чего на экране компьютера появляется диалоговое окно **Form1.MainMenu1** (рис. 15.4). Это диалоговое окно используется для создания разделов и пунктов будущего меню.



а



б

Рис. 15.4. Диалоговое окно создания главного меню: *а)* диалоговое окно сразу после его открытия; *б)* окно по окончании разработки меню

Для того чтобы создать первый раздел меню, достаточно щелкнуть в окне инспектора объектов справа от свойства `Caption`, затем ввести название раздела (например, слово `Файл`) и подтвердить ввод нажатием клавиши <Enter>.

Если теперь вы посмотрите на содержимое диалогового окна **Form1.MainMenu1**, то увидите, что в нем тоже появился пункт с именем **Файл**. Если же посмотреть теперь список объектов, то можно убедиться в том, что в проекте появился новый объект с именем `N1`. Это и есть созданный нами раздел меню.

Справа от раздела **Файл** появилась заготовка для создания следующего раздела меню. Выделяем эту заготовку, затем снова щелкаем свойство `Caption`, вводим название нового раздела (на этот раз название раздела — `Справка`) и получаем еще один раздел меню с именем `N2`. Теперь необходимо дополнить существующие разделы соответствующими пунктами.

Для того чтобы добавить пункт в раздел **Файл**, нужно выделить этот раздел и затем щелкнуть мышью заготовку, расположенную снизу от слова **Файл**. Затем снова щелкаем то же свойство `Caption`, вводим имя раздела `Выход` и получаем в проекте еще один объект с именем `N3`. Как видно, нумерация объектов продолжается в порядке их создания. Это связано с тем, что система программирования воспринимает и разделы меню, и содержащиеся в разделах пункты как объекты одного класса. Аналогичным образом в разделе меню **Справка** создаем пункт **Об авторах**. Этот пункт становится объектом по имени `N4`. Закрываем диалоговое окно **Form1.MainMenu1**, и на этом создание графического интерфейса основной формы программы "квадратное уравнение" завершается. Создание вспомогательной формы, содержащей сведения об авторах проекта, производится аналогично тому, как мы это делали в предыдущих проектах. Получившийся в результате нашей работы графический интерфейс основной формы изображен на рис. 15.5.

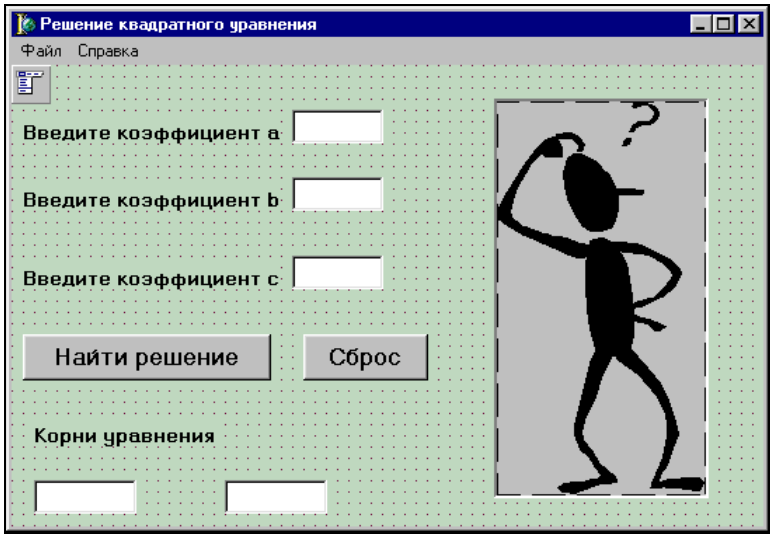


Рис. 15.5. Графический интерфейс программы решения квадратного уравнения

Написание кода для этой программы мы начинаем с описания реакции на нажатие экранной кнопки **Найти решение** (кнопка `Button1`). Эта реакция записывается в процедуре `TForm1.Button1Click`. Работу над процедурой начнем с того, что опишем используемые в ней переменные. Для решения поставленной задачи будут необходимы переменные для коэффициентов `a`, `b` и `c`, для дискриминанта `d` и для хранения полученных корней уравнения `x1` и `x2`. Исходные данные и результаты могут быть вещественными величинами, поэтому для их описания используем тип `real`:

```
var a,b,c,d,x1,x2:real;
```

Далее в основной части процедуры необходимо произвести преобразование вводимых в текстовые окна исходных данных — строковых величин в соответствующие им числовые величины вещественного типа. Это преобразование осуществляется с помощью знакомой нам стандартной функции `StrtoFloat`:

```
a:=StrtoFloat(Edit1.Text);
b:=StrtoFloat(Edit2.Text);
c:=StrtoFloat(Edit3.Text);
```

Следующим шагом в решении задачи является вычисление дискриминанта с помощью полученных в результате преобразования вещественных величин:

```
d:=sqr(b)-4*a*c;
```

Дальнейший ход решения задачи зависит от вычисленного значения дискриминанта. Если дискриминант неотрицателен, то необходимо произвести следующие действия:

1. Вычислить по известным формулам корни уравнения.
2. Сделать видимыми текстовые окна `Edit4` и `Edit5`, в которых будут выведены вычисленные корни.
3. Преобразовать полученные вещественные корни с помощью стандартной функции `FloattoStr` в текстовые величины, годные для вывода в текстовых окнах.
4. Вывести надпись, поясняющую полученные результаты, предварительно сделав ее видимой.

Вся перечисленная последовательность действий может быть реализована в программе с помощью следующих операторов:

```
x1:=(-b-sqrt(d))/(2*a);
x2:=(-b+sqrt(d))/(2*a);
Edit4.Text:= FloattoStr (x1);
Edit4.Visible:=True;
```

```
Edit5.Text:=FloattoStr(x2);
Edit5.Visible:=True;
Label4.Visible:=True;
Label4.Caption:='Корни уравнения'
```

В том случае если найденный дискриминант отрицателен, требуемые действия сводятся к тому, чтобы:

1. Вывести на экран надпись, говорящую об отсутствии решения уравнения, сделав надпись перед этим видимой.
2. Сделать невидимыми текстовые окна для вывода результатов, т. к. в этом случае необходимость в них отсутствует.

Перечисленные действия реализуются с помощью операторов:

```
Label4.Visible:=True;
Label4.Caption:='Уравнение не имеет решения';
Edit4.Visible:=False;
Edit5.Visible:=False
```

В целом условный оператор будет выглядеть следующим образом:

```
if d>=0
then
        begin
        x1:=(-b-sqrt(d))/(2*a);
        x2:=(-b+sqrt(d))/(2*a);
        Edit4.Text:=FloattoStr(x1);
        Edit4.Visible:=True;
        Edit5.Text:=FloattoStr(x2);
        Edit5.Visible:=True;
        Label4.Visible:=True;
        Label4.Caption:='Корни уравнения'
        end
else
        begin
        Label4.Visible:=True;
        Label4.Caption:='Уравнение не имеет решения';
        Edit4.Visible:=False;
        Edit5.Visible:=False
        end
```

Вся же разобранная ранее процедура в целом будет выглядеть так:

```
procedure TForm1.Button1Click(Sender: TObject);
var a,b,c,d,x1,x2:real;
```

```
begin
a:=StrtoFloat(Edit1.Text);
b:=StrtoFloat(Edit2.Text);
c:=StrtoFloat(Edit3.Text);
d:=sqr(b)-4*a*c;
if d>=0
then
        begin
        x1:=(-b-sqrt(d))/(2*a);
        x2:=(-b+sqrt(d))/(2*a);
        Edit4.Text:=FloattoStr(x1);
        Edit4.Visible:=True;
        Edit5.Text:=FloattoStr(x2);
        Edit5.Visible:=True;
        Label4.Visible:=True;
        Label4.Caption:='Корни уравнения'
        end
else
        begin
        Label4.Visible:=True;
        Label4.Caption:='Уравнение не имеет решения';
        Edit4.Visible:=False;
        Edit5.Visible:=False
        end
end;
```

Следующая процедура, которую мы рассматриваем, описывает реакцию на нажатие экранной кнопки **Сброс** (кнопка `Button2`). Эта процедура называется `TForm1.Button2Click` и в ней должны выполняться следующие действия:

1. Очистка текстовых окон, в которые вводятся исходные данные.
2. Превращение окон с результатами и поясняющей надписи в невидимые окна.

Первая операция осуществляется путем присвоения свойству `Text` каждого из окон пустой строки, которая записывается в виде пары апострофов. Вторая операция производится путем присвоения свойству `Visible` каждого из указанных объектов значения `False`. В итоге процедура будет выглядеть следующим образом:

```
procedure TForm1.Button2Click(Sender: TObject);
begin
Edit1.Text:='';
Edit2.Text:='';
Edit3.Text:='';
```

```
Edit4.Visible:=False;
Edit5.Visible:=False;
Label4.Visible:=False
end;
```

Для того чтобы завершить работу над основной формой проекта, осталось запрограммировать пункты меню. Написание кода для пунктов меню производится таким же образом, как и для прочих объектов, т. е. вначале мы выделяем объект (например, пункт меню `N3`, с помощью которого программа закрывается), затем выбираем вкладку **Events** (События) в инспекторе объектов и на вкладке находим событие `OnClick`. Дважды щелкнув мышью в поле справа от названия события, мы автоматически открываем соответствующую процедуру, которая для пункта `N3` будет называться `TForm1.N3Click`. Для закрытия программы в процедуру нужно вставить единственную команду `close`, а процедура целиком будет выглядеть так:

```
procedure TForm1.N3Click(Sender: TObject);
begin
close
end;
```

Пункт меню, представленный объектом `N4`, должен вызывать дополнительное окно со сведениями об авторах. Данное вспомогательное окно должно содержать сведения об авторе и кнопку, которая его закрывает. Для включения окна в состав проекта нужно его создать, а затем дополнить его необходимыми элементами интерфейса. Эта задача не вызовет у читателя никаких сложностей, потому что она выполняется точно так же, как и в предыдущих проектах. Не должно вызвать проблем и написание соответствующей подпрограммы. Подпрограмма, описывающая реакцию на выбор пункта меню **Об авторах**, называется `TForm1.N4Click` и должна выглядеть следующим образом:

```
procedure TForm1.N4Click(Sender: TObject);
begin
Form2.Show
end;
```

В целом программный модуль `Unit1` для основной формы проекта имеет следующий вид — листинг 15.2.

Листинг 15.2. Текст программы "Квадратное уравнение" (первый модуль)

```
unit Unit1;

interface
```

```
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, ExtCtrls, Menus;

type
  TForm1 = class(TForm)
    Label1: TLabel;
    Edit1: TEdit;
    Label2: TLabel;
    Label3: TLabel;
    Edit2: TEdit;
    Edit3: TEdit;
    Panel1: TPanel;
    Image1: TImage;
    Button1: TButton;
    Label4: TLabel;
    Edit4: TEdit;
    Edit5: TEdit;
    Button2: TButton;
    MainMenu1: TMainMenu;
    N1: TMenuItem;
    N2: TMenuItem;
    N3: TMenuItem;
    N4: TMenuItem;
    procedure Button1Click(Sender: TObject);
    procedure N3Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure N4Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation
 uses Unit2;
{$R *.dfm}

procedure TForm1.Button1Click(Sender: TObject);
var a,b,c,d,x1,x2:real;
begin
a:=StrtoFloat(Edit1.Text);
```

```
b:=StrtoFloat(Edit2.Text);
c:=StrtoFloat(Edit3.Text);
d:=sqr(b)-4*a*c;
if d>=0
then
        begin
        x1:=(-b-sqrt(d))/(2*a);
        x2:=(-b+sqrt(d))/(2*a);
        Edit4.Text:=FloattoStr(x1);
        Edit4.Visible:=True;
        Edit5.Text:=FloattoStr(x2);
        Edit5.Visible:=True;
        Label4.Visible:=True;
        Label4.Caption:='Корни уравнения'
        end
else
        begin
        Label4.Visible:=True;
        Label4.Caption:='Уравнение не имеет решения';
        Edit4.Visible:=False;
        Edit5.Visible:=False
        end
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
Edit1.Text:='';
Edit2.Text:='';
Edit3.Text:='';
Edit4.Visible:=False;
Edit5.Visible:=False;
Label4.Visible:=False
end;

procedure TForm1.N3Click(Sender: TObject);
begin
close
end;

procedure TForm1.N4Click(Sender: TObject);
begin
Form2.Show
end;

end.
```

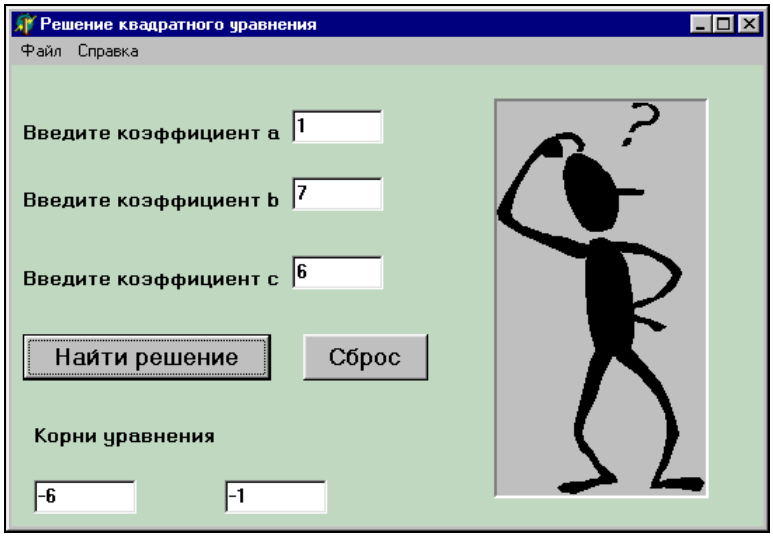


Рис. 15.6. Пример работы программы решения квадратного уравнения при положительном значении дискриминанта

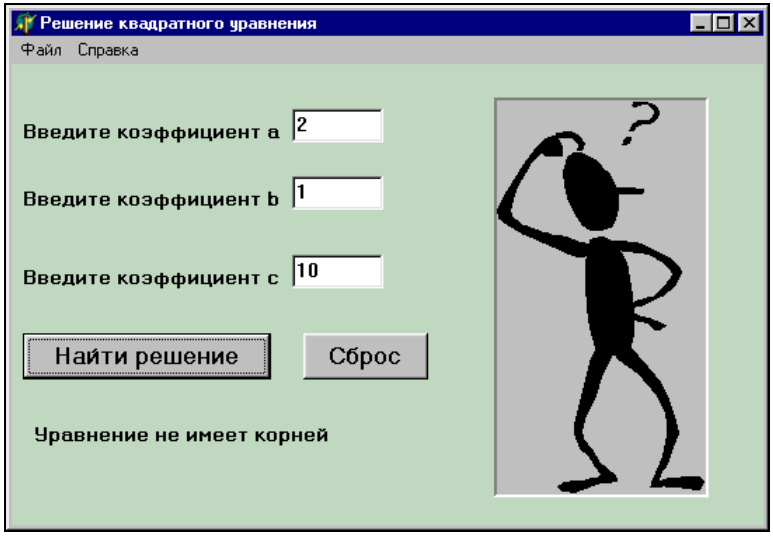


Рис. 15.7. Пример работы программы решения квадратного уравнения при отрицательном значении дискриминанта

Для полного завершения работы над проектом осталось написать программный код для модуля `Unit2`, который описывает реакцию объектов, входящих в состав второй, вспомогательной формы. Этот код аналогичен соответствующему коду из предыдущих проектов, написание его также не вызовет

у читателя затруднений. После написания кода для вспомогательной формы работа над проектом завершена и его можно запускать на выполнение.

На рис. 15.6—15.7 приведены примеры, демонстрирующие, как будет выглядеть основное окно программы при различных исходных данных, дающих различные значения дискриминанта. На рис. 15.6 приведен пример для положительного дискриминанта, на рис. 15.7 — для отрицательного дискриминанта.

15.3. Программа "Калькулятор"

Темой следующего проекта, который мы будем разрабатывать в Delphi, станет создание виртуального калькулятора, который будет производить основные арифметические операции (сложение, вычитание, умножение и деление) над двумя целыми числами. Кроме того, данный калькулятор должен иметь возможность выполнения двух дополнительных операций. Это нахождение квадрата числа и извлечение квадратного корня из числа.

Подобную задачу мы уже выполняли в Turbo Pascal. В программе, написанной в Turbo Pascal, для выбора одного из возможных вариантов использовался оператор множественного выбора `case`. В этой программе ввод и вывод данных производился в текстовом виде, а система Delphi обеспечивает графический интерфейс для ввода и вывода данных в виде специальных текстовых окон. Это позволяет выбрать необходимый вариант вычислений без использования ветвлений, что существенно облегчит написание и отладку программного кода. Однако полностью избежать использования условных операторов в данной программе не удастся. Эти операторы будут использоваться, во-первых, для защиты программы от неправильного ввода данных, а во-вторых, для другой цели, о которой мы расскажем в этой главе позднее.

Графический интерфейс программы "Калькулятор" включает в себя следующие основные элементы:

- два текстовых окна для ввода исходных данных — двух операндов;
- текстовое окно для вывода полученного результата;
- поясняющие надписи к текстовым окнам;
- четыре экранные кнопки для выполнения основных арифметических действий;
- две экранные кнопки для выполнения дополнительных действий — возведения числа в квадрат и извлечения квадратного корня;
- надписи, которые сообщают пользователю о сделанной им ошибке в том случае, если он неправильно ввел исходные данные (например, попытался разделить число на нуль);

- экранная кнопка **Сброс**, которая используется для очистки всех текстовых окон (как с исходными данными, так и окна результата);
- экранная кнопка **Выход**, которая закрывает форму и вместе с ней сам проект.

Кроме перечисленных ранее основных элементов в форме могут присутствовать и некоторые дополнительные, к которым относятся:

- иллюстрация, изображающая калькулятор и записную книжку;
- флажок, установка которого позволяет получать результат вычислений в округленном виде (о том, что такое элемент управления "флажок", для чего он нужен и как с ним работать, мы расскажем далее).

Работу над проектом мы начнем с самой формы. В заголовке формы (свойство `Caption` объекта `Form1`) мы вводим ее название — `Калькулятор`. Затем изменяем цвет формы на бледно-зеленый. После этого мы приступаем к выводу на форму элементов графического интерфейса. Вначале мы выводим на форму три текстовых окна. В окнах `Edit1` и `Edit2` будут вводиться исходные данные для вычислений, а в окне `Edit3` будет виден результат вычислений. Рядом с каждым из упомянутых текстовых окон имеется поясняющая надпись. Эти объекты в нашей программе будут называться `Label1`, `Label2` и `Label3`.

Затем мы создаем в программе ряд экранных кнопок. Кнопка `Button1` будет использоваться для выполнения операции сложения, `Button2` — для вычитания чисел, `Button3` — для умножения, `Button4` — для деления, `Button5` — для извлечения квадратного корня из числа, `Button6` — для возведения числа в квадрат. Для выхода из программы используется кнопка `Button7`, для сброса данных, т. е. для очистки всех текстовых окон — кнопка `Button8`.

Справа от текстовых окон мы поместим дополнительные надписи с сообщениями об ошибках. Справа от окна `Edit1` расположим надписи `Label4` и `Label5`. По умолчанию эти надписи мы сделаем невидимыми (для свойства `Visible` мы установим значение `False`). В случае же, если пользователь совершил недопустимое действие (ввел в окно отрицательное число, а затем нажал кнопку `Button5` для извлечения квадратного корня), эти надписи должны стать видимыми. Тогда пользователь увидит на форме сообщение следующего содержания: "Вводимое число должно быть неотрицательным".

Рядом с окном `Edit2` мы поместим надпись `Label6`, которую также по умолчанию сделаем невидимой. В том случае, если пользователь ввел какое-либо число в окно `Edit1`, а затем ввел в окно `Edit2` нуль и нажал экранную кнопку `Button4` для того, чтобы произвести деление первого числа на второе, надпись `Label6` станет видимой. На форме справа от окна `Edit2` пользователь увидит сообщение: "Делить на нуль нельзя".

Еще одним элементом управления на экранной форме проекта "калькулятор" станет флажок `Checkbox1`. Это новый для нас элемент. Если на форме имеется группа флажков, то в этой группе могут быть включены все флажки, может быть включена часть флажков, один флажок или вообще ни одного. Этим флажки отличаются от переключателей, в группе которых всегда должен быть включен один и только один. Флажки могут находиться на форме и по одному, как в нашем случае.

Если щелчком мыши включить флажок, то он обеспечит выполнение некоторого дополнительного действия при работе программы. В нашем проекте если флажок будет включен, то получившийся в окне `Edit3` результат будет округляться до ближайшего целочисленного значения. Для того чтобы поместить флажок в форму, нужно выбрать в палитре компонентов находящийся на вкладке **Standard** компонент `Checkbox`, который выглядит следующим образом: ☒. Во время работы программы на форме выключенный флажок будет отображаться в виде пустого квадратика ☐, а включенный флажок будет выглядеть так: ☑.

Неотъемлемой частью флажка является расположенная рядом с ним надпись. Содержание этой надписи определяется свойством `Caption` объекта `Checkbox`. Сразу после создания флажка мы удалим имеющееся по умолчанию значение `Checkbox1` и вместо него в поле справа от названия свойства введем следующий текст: `Целочисленный результат`. По умолчанию текст пояснительной надписи находится в элементе справа от флажка. Если же необходимо изменить взаимное расположение частей внутри объекта `Checkbox`, то для этого нужно использовать свойство `Alignment` объекта. `Alignment` — способ выравнивания надписи. По умолчанию это свойство имеет значение `taRightJustify`, т. е. выравнивание производится по правому краю элемента. Если щелкнуть справа от `Alignment` раскрывающую кнопку, то из появившегося списка можно выбрать значение `taLeftJustify` — выравнивание по левому краю. Тогда надпись окажется слева от самого флажка, как это и сделано в нашем примере.

Еще одним, дополнительным элементом формы является иллюстрация в виде записной книжки с калькулятором. Иллюстрацию мы взяли из обучающего курса Tour98, который входит в состав операционной системы Windows 98. Для создания иллюстрации мы поместили в форме фоновый объект `Panel`. Поверх него мы расположили объект `Image1`, т. е. саму иллюстрацию.

Однако при попытке загрузить в `Image1` иллюстрацию мы столкнулись со следующей проблемой. Указанная иллюстрация, которая называется `Chckbook`, была создана в графическом формате GIF, который система Delphi не понимает. Поэтому для того, чтобы все-таки воспользоваться рисунком, пришлось прибегнуть к следующему приему. Мы скопировали в буфер обмена рисунок Chckbook, запустили графический редактор Paint, являющийся стандартным

приложением Windows, и в документ Paint вставили эту копию. Затем мы сохранили документ в стандартном для программы Paint формате BMP. В папке проекта появился новый графический файл, которому мы дали название Калькулятор. Таким образом, мы сохранили иллюстрацию в том формате, который понимает система Delphi.

После этого появилась возможность загрузить данную картинку в объект `Image1`, используя свойство `Picture` этого объекта. Поскольку картинка была преобразована в растровый формат, который плохо поддается масштабированию, возникла необходимость в подгонке размеров `Panel1` и `Image1` под размеры картинки. Как это можно сделать, мы уже описывали ранее. На этом создание графического интерфейса программы было завершено (рис. 15.8).

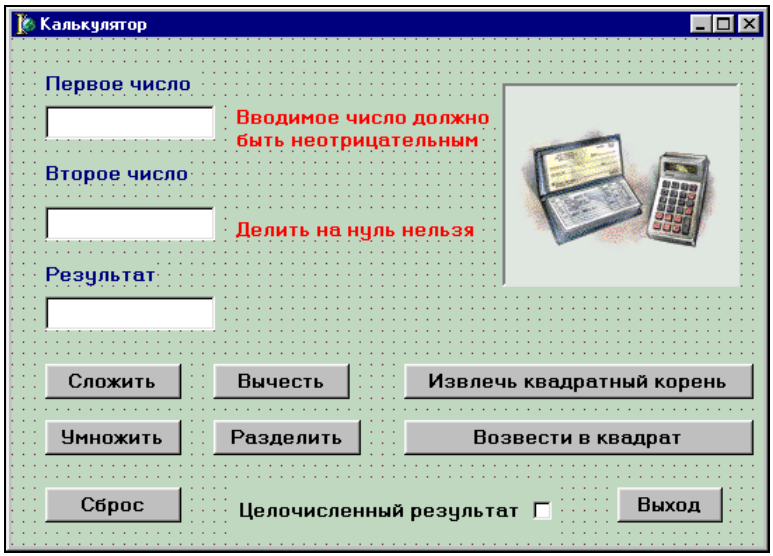


Рис. 15.8. Графический интерфейс программы "Калькулятор"

Написание программного кода мы начнем с программирования действий для кнопки `Button1`, производящей сложение чисел, которые пользователь вводит в текстовые окна `Edit1` и `Edit2`. Как это уже делалось в предыдущих проектах, мы преобразуем исходные строковые величины `Edit1.Text` и `Edit2.Text` в вещественные переменные `a` и `b`. Затем мы складываем значения этих переменных, и полученный текстовый результат присваиваем текстовой переменной `c`. Указанные действия выполняет следующий фрагмент программы:

```
a:=StrtoFloat(Edit1.Text);
b:=StrtoFloat(Edit2.Text);
c:=a+b;
```

Если бы в проекте не предусматривалась возможность округления полученного результата, то далее можно было бы ограничиться обратным преобразованием `c` в строковую величину `Edit3.Text`, которая будет выводиться в окне результата. Но в данном случае перед выполнением такого преобразования следует проверить, в каком состоянии находится флажок "целочисленный результат", т. е. проверить свойства объекта `Checkbox1`. Данный объект имеет свойство `Checked`. Если флажок включен, то свойство `Checked` имеет значение `True`, а если флажок выключен, то значение `False`. Проверку состояния флажка можно вставить в условный оператор `if` в качестве условия. Если флажок включен, то будет выполняться та ветвь условного оператора, которая находится после служебного слова `then`, а если он выключен — та часть, которая расположена после слова `else`.

В ветви оператора, находящейся после `then`, мы выполним следующие действия. Во-первых, мы преобразуем получившийся действительный результат `c` в целочисленный `okr` с помощью стандартной функции `round`, которая округляет вещественную величину до ближайшей целой, преобразуя ее при этом к целому типу. Во-вторых, получившуюся целочисленную величину `okr` мы с помощью функции `InttoStr` преобразуем в строковое значение объекта `Edit3`. Указанные действия мы выполним с помощью следующих операторов:

```
okr:=round(c);
Edit3.Text:=InttoStr(okr)
```

В той части условного оператора, которая находится после слова `else`, выполняется преобразование результата `c` в `Edit3.Text` с помощью функции `FloattoStr`:

```
Edit3.Text:=FloattoStr(c)
```

В целом же подпрограмма, описывающая действия, которые происходят при нажатии на кнопку **Сложить**, выглядит следующим образом:

```
procedure TForm1.Button1Click(Sender: TObject);
var a,b,c:real; okr:integer;
begin
a:=StrtoFloat(Edit1.Text);
b:=StrtoFloat(Edit2.Text);
c:=a+b;
if Checkbox1.Checked
then
       begin
       okr:=round(c);
       Edit3.Text:=InttoStr(okr)
       end
```

```
else   Edit3.Text:=FloattoStr(c)
end;
```

Аналогичным образом составляются подпрограммы, которые описывают действие кнопок **Вычесть** (`Button2`) и **Умножить** (`Button4`). При составлении же подпрограммы для кнопки **Разделить** (`Button4`) необходимо учесть еще один момент. Дело в том, что как уже говорилось ранее, в программе необходимо предусмотреть защиту от неправильного ввода данных. Это может произойти в том случае, если пользователь попытается разделить какое-либо число на нуль путем ввода нулевого значения в текстовое окно `Edit2`. Следовательно, перед выполнением дальнейших вычислений при нажатии кнопки `Button4` нужно проверить, чему равно значение в текстовом окне `Edit2`.

Поэтому после преобразования текстовой величины `Edit2.Text` в вещественную величину `b` в тексте подпрограммы должен находиться условный оператор. В заголовке этого условного оператора переменная `b` проверяется на неравенство нулю. В случае если это неравенство ложно, выполняется та ветвь оператора, которая находится после `else`. В этой ветви делается видимой надпись `Label6`, которая сообщает пользователю об ошибке, допущенной им при вводе исходных данных. Если же неравенство выполняется, то после служебного слова `then` записывается та же последовательность действий, что и для кнопок **Сложить**, **Вычесть** и **Умножить**. Напомним, что указанная последовательность действий также включает в себя условный оператор. Таким образом, данная подпрограмма имеет более сложную структуру, чем предшествующие ей. Эта структура включает в себя условный оператор, вложенный в другой условный оператор. В целом подпрограмма, описывающая работу кнопки **Разделить**, выглядит следующим образом:

```
procedure TForm1.Button4Click(Sender: TObject);
var a,b,c:real; okr:integer;
begin
a:=StrtoFloat(Edit1.Text);
b:=StrtoFloat(Edit2.Text);
if b<>0 then
            begin
             c:=a/b;
             if Checkbox1.Checked
            then
                  begin
                 okr:=round(c);
                Edit3.Text:=InttoStr(okr)
                end
            else Edit3.Text:=FloattoStr(c)
          end
```

```
        else
           begin
           Label6.Visible:=True;
           Edit3.Text:=''
          end
end;
```

Подпрограмма, которая описывает действия, производимые в результате щелчка на кнопке **Извлечь квадратный корень** (`Button5`), также должна включать в себя защиту от неправильного ввода исходных данных, которая может заключаться во вводе отрицательного значения в окно `Edit1`. В начале подпрограммы значение, введенное в текстовое окно `Edit1`, преобразуется в вещественную величину `a`, а второе текстовое окно очищается. Это делается ввиду того, что данная операция является унарной, т. е. выполняется только над одним числом. Данные действия описываются следующими операторами:

```
a:=StrtoFloat(Edit1.Text);
Edit2.Text:='';
```

Затем в подпрограмме должен быть предусмотрен условный оператор, который проверяет вещественную переменную `a` на неотрицательность. Если `a` больше нуля или равна нулю, то выполняются операторы, находящиеся после слова `then`, т. е. вычисляется квадратный корень `c`, который преобразуется затем в строковую величину `Edit3.Text`. Если же условие оказывается ложным (`a` меньше нуля), то становятся видимыми надписи `Label4` и `Label5`, которые сообщают об ошибке ввода. В целом данная подпрограмма выглядит так:

```
procedure TForm1.Button5Click(Sender: TObject);
var a,c:real; okr:integer;
begin
a:=StrtoFloat(Edit1.Text);
Edit2.Text:='';
 if a>=0 then
            begin
            c:=sqrt(a);
            if Checkbox1.Checked
            then
                 begin
                 okr:=round(c);
                 Edit3.Text:=InttoStr(okr)
                 end
```

```
            else Edit3.Text:=FloattoStr(c)
            end
          else
            begin
            Edit3.Text:='';
            Label4.Visible:=True;
            Label5.Visible:=True
            end
end;
```

Подпрограмма, которая описывает реакцию на нажатие кнопки **Возвести в квадрат** (Button6), по своей структуре сходна с предыдущей подпрограммой, но проще ее, т. к. в данной подпрограмме не предусматривается защита от неправильного ввода данных. Текст подпрограммы для экранной кнопки Button6 следующий:

```
procedure TForm1.Button6Click(Sender: TObject);
var a,c:real; okr:integer;
begin
a:=StrtoFloat(Edit1.Text);
Edit2.Text:='';
c:=sqr(a);
if Checkbox1.Checked
then
     begin
     okr:=round(c);
     Edit3.Text:=InttoStr(okr)
     end
else Edit3.Text:=FloattoStr(c)
end;
```

Кнопка **Выход** (Button7) закрывает форму, а кнопка **Сброс** (Button8) очищает все текстовые окна, имеющиеся в форме, и делает невидимыми все надписи с сообщениями об ошибках, которые могут выводиться на форме. Программирование этих кнопок никакой сложности не представляет. В листинге 15.3 приводится полный текст программного модуля для проекта "калькулятор".

Листинг 15.3. Текст программы "Калькулятор"

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, ExtCtrls, StdCtrls;
```

```
type
  TForm1 = class(TForm)
    Label1: TLabel;
    Label2: TLabel;
    Edit1: TEdit;
    Edit2: TEdit;
    Button1: TButton;
    Button2: TButton;
    Button3: TButton;
    Button4: TButton;
    Button5: TButton;
    Button6: TButton;
    Button7: TButton;
    Panel1: TPanel;
    Button8: TButton;
    Image1: TImage;
    Label3: TLabel;
    Edit3: TEdit;
    Label4: TLabel;
    Label5: TLabel;
    Label6: TLabel;
    CheckBox1: TCheckBox;
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure Button3Click(Sender: TObject);
    procedure Button4Click(Sender: TObject);
    procedure Button5Click(Sender: TObject);
    procedure Button6Click(Sender: TObject);
    procedure Button7Click(Sender: TObject);
    procedure Button8Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.Button1Click(Sender: TObject);
var a,b,c:real; okr:integer;
```

```
begin
a:=StrtoFloat(Edit1.Text);
b:=StrtoFloat(Edit2.Text);
c:=a+b;
if Checkbox1.Checked
then
       begin
       okr:=round(c);
       Edit3.Text:=InttoStr(okr)
       end
else   Edit3.Text:=FloattoStr(c)
end;

procedure TForm1.Button2Click(Sender: TObject);
var a,b,c:real; okr:integer;
begin
a:=StrtoFloat(Edit1.Text);
b:=StrtoFloat(Edit2.Text);
c:=a-b;
if Checkbox1.Checked
then
     begin
     okr:=round(c);
     Edit3.Text:=InttoStr(okr)
     end
else Edit3.Text:=FloattoStr(c)
end;

procedure TForm1.Button3Click(Sender: TObject);
var a,b,c:real; okr:integer;
begin
a:=StrtoFloat(Edit1.Text);
b:=StrtoFloat(Edit2.Text);
c:=a*b;
if Checkbox1.Checked
then
     begin
     okr:=round(c);
     Edit3.Text:=InttoStr(okr)
     end
else Edit3.Text:=FloattoStr(c)
end;

procedure TForm1.Button4Click(Sender: TObject);
var a,b,c:real; okr:integer;
```

```
begin
a:=StrtoFloat(Edit1.Text);
b:=StrtoFloat(Edit2.Text);
if b<>0 then
           begin
           c:=a/b;
           if Checkbox1.Checked
           then
                begin
                okr:=round(c);
                Edit3.Text:=InttoStr(okr)
                end
           else Edit3.Text:=FloattoStr(c)
           end
         else
           begin
           Label6.Visible:=True;
           Edit3.Text:=''
           end
end;

procedure TForm1.Button5Click(Sender: TObject);
var a,c:real; okr:integer;
begin
a:=StrtoFloat(Edit1.Text);
Edit2.Text:='';
 if a>=0 then
            begin
            c:=sqrt(a);
            if Checkbox1.Checked
            then
                 begin
                 okr:=round(c);
                 Edit3.Text:=InttoStr(okr)
                 end
            else Edit3.Text:=FloattoStr(c)
            end
          else
            begin
            Edit3.Text:='';
            Label4.Visible:=True;
            Label5.Visible:=True
            end
end;
```

```
procedure TForm1.Button6Click(Sender: TObject);
var a,c:real;  okr:integer;
begin
a:=StrtoFloat(Edit1.Text);
Edit2.Text:='';
c:=sqr(a);
if Checkbox1.Checked
then
     begin
     okr:=round(c);
     Edit3.Text:=InttoStr(okr)
     end
else Edit3.Text:=FloattoStr(c)
end;

procedure TForm1.Button7Click(Sender: TObject);
begin
close
end;

procedure TForm1.Button8Click(Sender: TObject);
begin
Edit1.Text:='';
Edit2.Text:='';
Edit3.Text:='';
Label4.Visible:=False;
Label5.Visible:=False;
Label6.Visible:=False
end;

end.
```

На рис. 15.9 показан пример работы программы "Калькулятор" при правильном вводе исходных данных, а на рис. 15.10 показано, как реагирует данная программа на попытку неправильного ввода исходных данных (в этом случае была предпринята попытка деления числа на нуль). На рис. 15.11 приведен пример работы программы с округлением полученного результата.

15.4. Усовершенствованная программа "Светофор"

Как и в Turbo Pascal, в языке Object Pascal наряду с условным оператором `if...then...else` для организации ветвлений в программе используется также оператор множественного выбора `case`. Рассмотрим возможности приме-

нения этого оператора на примере программы "Светофор". Эта программа уже предлагалась пользователю в *главе 13* для самостоятельной разработки. При этом предполагалось, что светофор будет управляться пользователем программы вручную, т. е. лампочки светофора будут включаться щелчком на соответствующем переключателе.

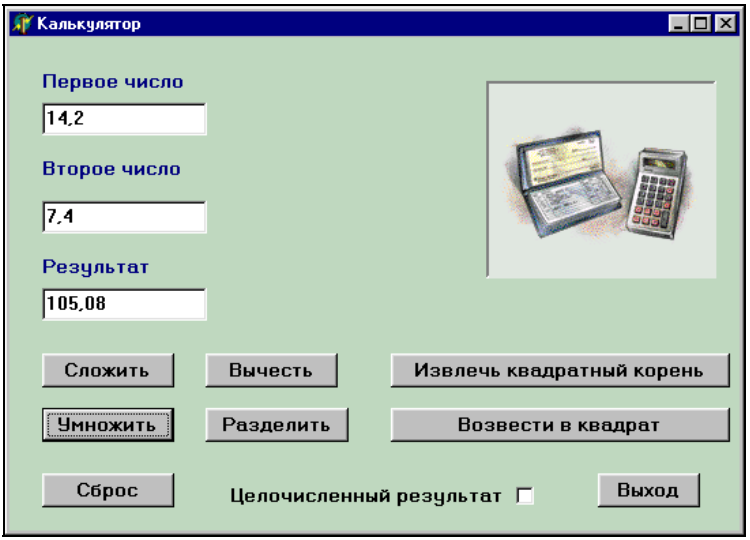


Рис. 15.9. Пример работы программы "Калькулятор" при правильном вводе исходных данных без округления результата

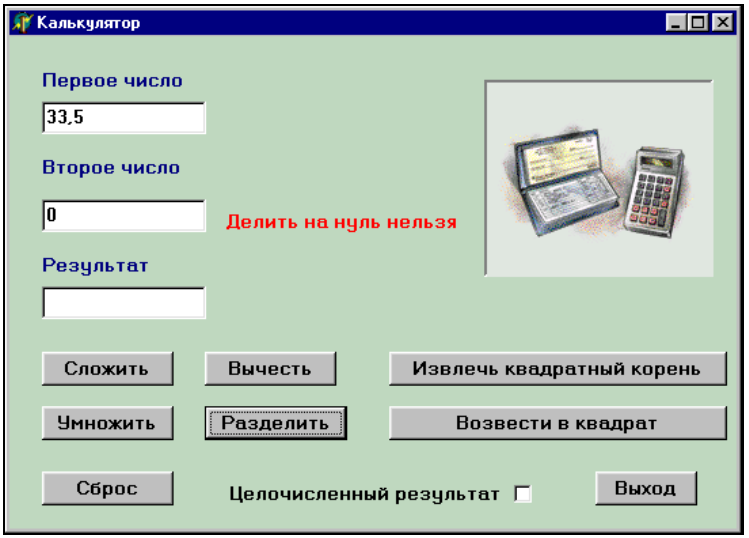


Рис. 15.10. Пример работы программы "Калькулятор" при неправильном вводе исходных данных

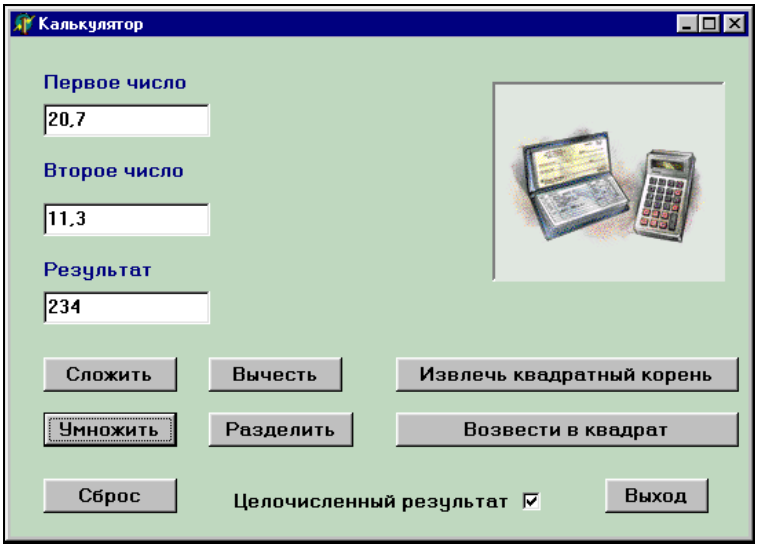


Рис. 15.11. Пример работы программы "Калькулятор" при правильном вводе исходных данных с округлением полученного результата

Как читатель, конечно, знает, светофоры, которые реально используются для регулировки уличного движения, обычно работают в автоматическом режиме. Поэтому естественно, что следующим шагом в усовершенствовании данной программы станет создание возможности автоматического переключения лампочек светофора. Для этого понадобится несколько изменить графический интерфейс программы.

Такой элемент интерфейса, как группа переключателей при отсутствии ручного режима, не нужен, поэтому в новом варианте проекта данный элемент управления можно удалить. Вместо него в форме появятся два новых элемента. Во-первых, в форме создается еще одна дополнительная панель `Panel2`, на которой будет помещаться иллюстрация `Image1`. Эта иллюстрация будет наглядно показывать, в каком режиме работает светофор в данный момент времени. В то время, когда будет гореть красный сигнал светофора, помимо красной лампочки пользователь будет видеть на экране изображение красной автомашины, говорящее о том, что пешеходы должны остановиться и пропустить движущийся транспорт. Изображение автомашины для проекта можно взять, например, из галереи Clipart программного пакета Office 97, где имеется изображение car.wmf. Копию этого рисунка мы поместим в папку нашего проекта. Рисунок car.wmf мы и возьмем в качестве свойства `Picture` объекта `Image1`. В дальнейшем при выключении красной лампочки и включении желтой или зеленой мы будем менять изображение программным путем.

Вторым новым элементом управления в экранной форме станет таймер. Таймер относится к невизуальным компонентам, т. е. он не будет виден в экранной форме во время работы программы, а увидеть его можно только во время проектирования. Тем не менее этот невидимый компонент может оказывать существенное влияние на ход работы программы. Для того чтобы поместить таймер на форму, нужно щелкнуть на палитре компонентов вкладку **System**, а на ней найти значок таймера, который выглядит следующим образом:  . Затем щелчком в основной форме мы можем поместить таймер в любом ее свободном месте. Таким образом, у нас в форме появляется объект по имени `Timer1`.

Подобно другим объектам, таймер имеет собственные свойства и события. Единственным событием для него является событие `OnTimer`, т. е. посылка таймером сигнала, на который могут реагировать другие объекты. У таймера имеется несколько свойств, важнейшим из которых является `Interval` — интервал, с которым таймер посылает эти сигналы. Этот интервал измеряется в миллисекундах, т. е. тысячных долях секунды. В нашем проекте по сигналу таймера будет происходить переключение лампочек светофора. Установим значение свойства `Interval` равным 3000 для того, чтобы переключение лампочек происходило каждые 3 секунды.

Остальные элементы интерфейса в данном проекте остаются такими, как и в его начальной версии. Это панель `Panel1`, на которой размещаются три объекта класса `Shape`. Объект `Shape1` соответствует красной лампочке светофора, `Shape2` — желтой лампочке, а `Shape3` — зеленой. В форме также сохраняется кнопка `Button1`, щелчок на которой закрывает программу. Остается в форме и надпись `Label1`, но содержание ее станет иным: `Переключение сигналов светофора производится автоматически`. На этом работу над графическим интерфейсом проекта можно считать завершенной. Графический интерфейс данного проекта приведен на рис. 15.12.

Написание программного кода мы начнем с описания того момента, когда программная форма будет создана и выведена на экран компьютера. Этому моменту для формы соответствует событие `OnCreate`. При создании формы должны происходить также следующие события: должна включаться красная лампочка светофора, в форме должно появляться изображение автомашины, причем для того, чтобы это изображение не искажалось, желательно, чтобы размеры объектов `Image1` и `Panel2` подгонялись под размеры самого изображения. Поэтому для процедуры `FormCreate` мы записываем следующие операторы:

```
Shape1.Brush.Color:=clRed;
Panel2.Height:=64;
Image1.Height:=60;
```

```
Panel2.Width:=233;
Image1.Width:=229
```

Напомним читателю, что для объекта `Shape` существует составное свойство `Brush`, которое определяет и цвет самого объекта, и стиль его заполнения. Цвет объекта определяется с помощью подсвойства `Brush.Color`, которому мы и сообщаем значение `clRed` (красный цвет).

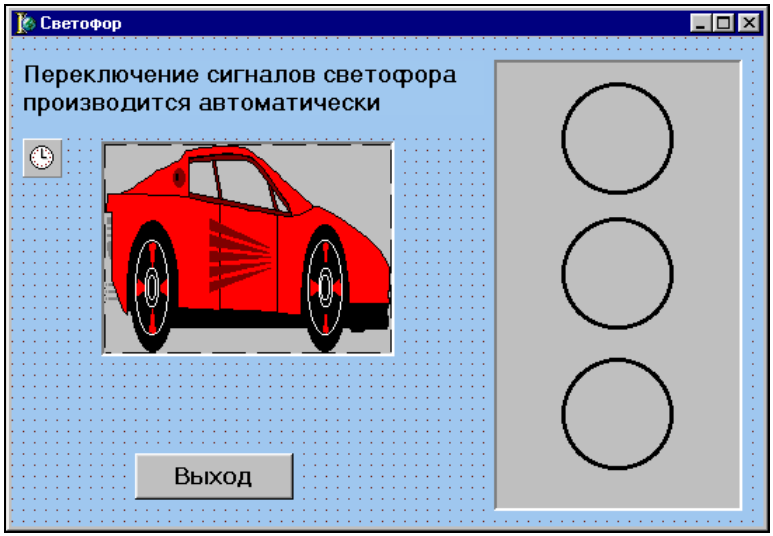


Рис. 15.12. Графический интерфейс программы "Светофор" после ее усовершенствования

Дальнейшие действия по написанию программного кода связаны с объектом `Timer1`. Два раза щелкнув его мышью, мы создаем процедуру `Timer1Timer`, для которой и будем писать код. Для того чтобы каждый раз по истечении заданного промежутка времени происходило переключение лампочек и изменение внешнего вида иллюстрации, мы поступим следующим образом: в начале процедуры мы поместим группу из трех условных операторов `if`. По прошествии каждых 3 секунд происходит очередное срабатывание таймера. При посылке очередного сигнала таймера мы программным образом задаем проверку состояния всех трех лампочек светофора, которая осуществляется с помощью следующих операторов:

```
if Shape1.Brush.Color=clRed
then n:=1;
if Shape2.Brush.Color=clYellow
then n:=2;
if Shape3.Brush.Color=clLime
then n:=3;
```

В зависимости от того, какая из лампочек включена (такая лампочка имеет активный красный, желтый или зеленый цвет в отличие от двух других лампочек, окрашенных в серый), программа присваивает некоторое значение вспомогательной переменной `n`.

Далее в процедуре находится оператор множественного выбора, в котором роль переменной-селектора играет упомянутая ранее `n`. Если текущий цвет был красным, то `n` присваивается значение, равное единице. Этому значению `n` соответствует следующий вариант, описанный в операторе `case`:

```
Image1.Picture.LoadFromFile('Stop.wmf');
Image1.Height:=178;
Panel2.Height:=182;
Image1.Width:=107;
Panel2.Width:=111;
Shape1.Brush.Color:=clSilver;
Shape2.Brush.Color:=clYellow;
```

Поясним, какие действия производит эта группа операторов. Вначале для иллюстрации `Image1` мы загружаем новую картинку Stop.wmf, взятую из той же электронной галереи, что и предыдущая картинка. Предварительно мы копируем картинку в папку нашего проекта. Эта картинка с изображением дорожного сигнала Stop должна соответствовать желтому сигналу светофора. Для загрузки картинки мы используем метод `LoadFromFile` (загрузка из файла). Обратите внимание, что имя метода должно находиться справа от того свойства, которое изменяется с помощью этого метода и должно быть отделено от него точкой. Затем мы меняем габариты объектов `Panel1` и `Image1` таким образом, чтобы не искажалось загружаемое из файла изображение. Далее мы закрашиваем стандартным серебристо-серым цветом объект `Shape1` (красную лампочку), а объект `Shape2` (центральная лампочка) делаем ярко-желтого цвета.

Аналогичную группу операторов можно написать и в том случае, если текущий цвет светофора был зеленым, а `n` равно 3. Здесь будет появляться тот же рисунок, и объект `Shape2` также будет приобретать желтый цвет, в то время как сама зеленая лампочка (объект `Shape3`) будет закрашиваться серебристо-серым цветом.

Более сложной становится ситуация в том случае, если во время получения сигнала таймера горит лампочка желтого цвета. В этом случае дальнейший переход может производиться как к зеленому цвету (в том случае, если до этого горел красный), так и к красному цвету, если перед желтым цветом был включен зеленый. Для того чтобы избавиться от такого рода неопределенности, мы введем в программе еще одну вспомогательную переменную `flag`, с помощью которой можно определить, какой свет горел перед желтым.

Если при очередной проверке текущий цвет светофора оказывается красным, то переменная `flag` получает значение, равное нулю. Если же текущий цвет — зеленый, то переменной `flag` присваивается значение, равное единице.

Когда включается желтый свет, в программе производится проверка состояния флага. Если флаг равен нулю, то после желтого цвета должен загораться зеленый и будет загружаться соответствующий рисунок. Этот рисунок взят из коллекции Microsoft Office 2000 и называется people.wmf, причем его также необходимо предварительно записать в папку проекта. Если же при проверке флаг оказывается равным единице, то после желтого света будет гореть красный, и загружаться уже знакомый нам файл Car.wmf. В программе проверка реализована с помощью условного оператора `if` и данный фрагмент программы выглядит следующим образом:

```
if flag=1
       then
           begin
           Shape2.Brush.Color:=clSilver;
           Shape1.Brush.Color:=clRed;
           Image1.Picture.LoadFromFile('Car.wmf');
           Panel2.Height:=64;
           Image1.Height:=60;
           Panel2.Width:=233;
           Image1.Width:=229
           end
       else
           begin
           Image1.Picture.LoadFromFile('People.wmf');
           Panel2.Height:=137;
           Image1.Height:=133;
           Panel2.Width:=152;
           Image1.Width:=148;
           Shape2.Brush.Color:=clSilver;
           Shape3.Brush.Color:=clLime
           end;
```

Обратите внимание, что при каждой загрузке новой картинки для объекта `Image1` динамически изменяются размеры как этого объекта, так и панели, на которой он расположен. Следует также отметить, что переменные `flag` и `n` являются в программе глобальными переменными, и поэтому их описание содержится в программе в разделе `interface`. В листинге 15.4 приводится полный текст усовершенствованной программы "Светофор".

Листинг 15.4. Текст усовершенствованной программы "Светофор"

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, ExtCtrls, StdCtrls;

type
  TForm1 = class(TForm)
    Panel1: TPanel;
    Label1: TLabel;
    Button1: TButton;
    Shape1: TShape;
    Shape2: TShape;
    Shape3: TShape;
    Timer1: TTimer;
    Panel2: TPanel;
    Image1: TImage;
    Label2: TLabel;

    procedure Timer1Timer(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1; n:integer; flag:integer;

implementation

{$R *.dfm}
procedure TForm1.FormCreate(Sender: TObject);
begin
Shape1.Brush.Color:=clRed;
Panel2.Height:=64;
Image1.Height:=60;
Panel2.Width:=233;
Image1.Width:=229
end;
```

```
procedure TForm1.Timer1Timer(Sender: TObject);

begin
if Shape1.Brush.Color=clRed
then n:=1;
if Shape2.Brush.Color=clYellow
then n:=2;
if Shape3.Brush.Color=clLime
then n:=3;
case n of
    1:begin
      Image1.Picture.LoadFromFile('Stop.wmf');
      Image1.Height:=178;
      Panel2.Height:=182;
      Image1.Width:=107;
      Panel2.Width:=111;
      Shape1.Brush.Color:=clSilver;
      Shape2.Brush.Color:=clYellow;
      flag:=0
      end;

    2: begin
       if flag=1
       then
           begin
           Shape2.Brush.Color:=clSilver;
           Shape1.Brush.Color:=clRed;
           Image1.Picture.LoadFromFile('Car.wmf');
           Panel2.Height:=64;
           Image1.Height:=60;
           Panel2.Width:=233;
           Image1.Width:=229
           end
       else
           begin
           Image1.Picture.LoadFromFile('People.wmf');
           Panel2.Height:=137;
           Image1.Height:=133;
           Panel2.Width:=152;
           Image1.Width:=148;
           Shape2.Brush.Color:=clSilver;
           Shape3.Brush.Color:=clLime
           end;
       end;
```

```
        3: begin
           Image1.Picture.LoadFromFile('Stop.wmf');
           Image1.Height:=178;
           Panel2.Height:=182;
           Image1.Width:=107;
           Panel2.Width:=111;
           Shape3.Brush.Color:=clSilver;
           Shape2.Brush.Color:=clYellow;
           flag:=1
           end;
end;
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
close
end;

end.
```

На рис. 15.13 показано, как выглядит программа во время работы при включенном красном свете, на рис. 15.14 — при включенном желтом цвете и на рис. 15.15 в том случае, когда на форме горит зеленый свет.

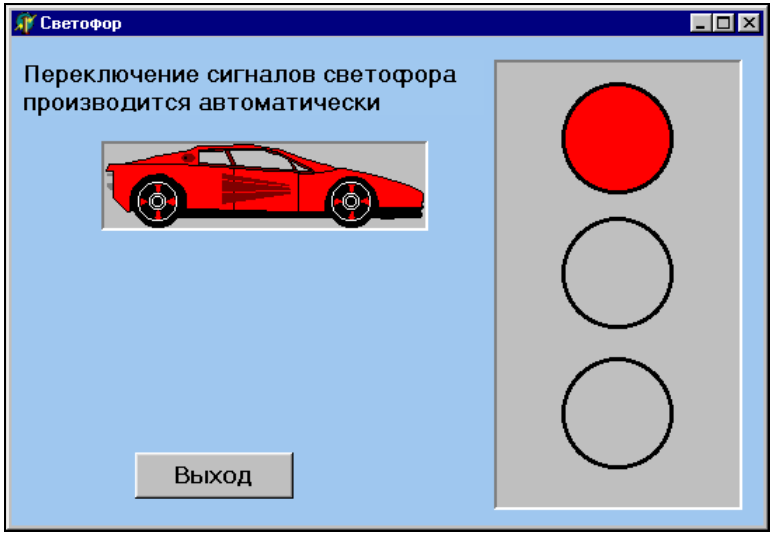


Рис. 15.13. Программа "Светофор" во время работы. Горит красный свет

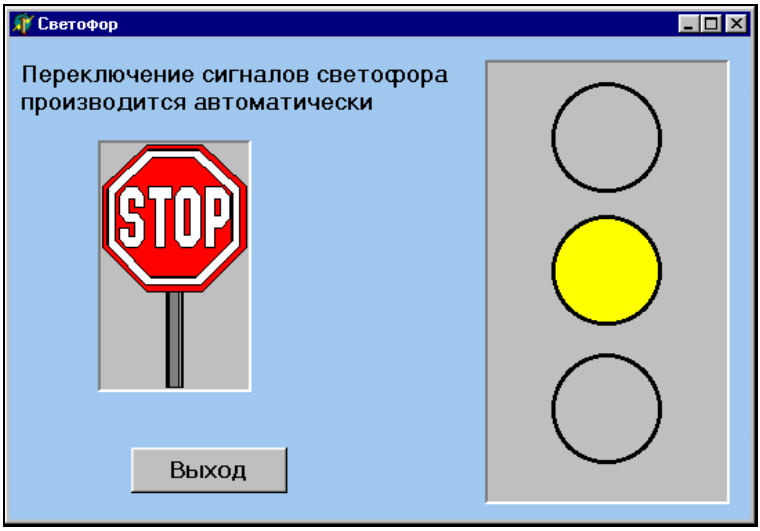


Рис. 15.14. Программа "Светофор" во время работы. Горит желтый свет

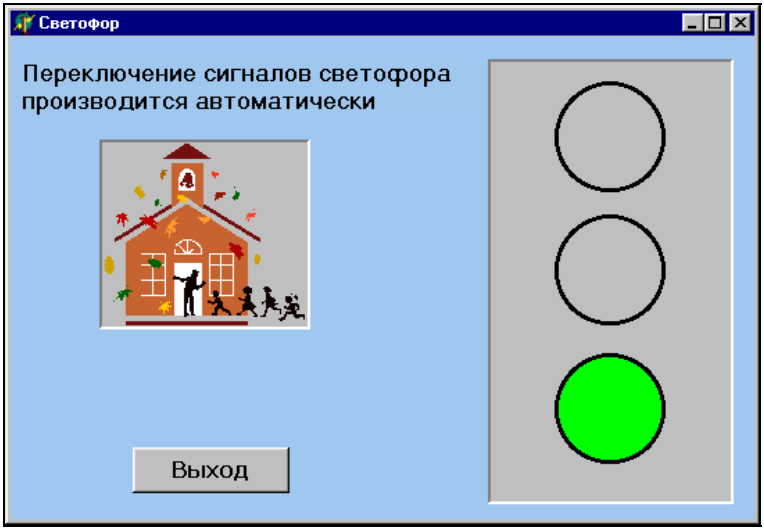


Рис. 15.15. Программа "Светофор" во время работы. Горит зеленый свет

Задание для самостоятельной работы

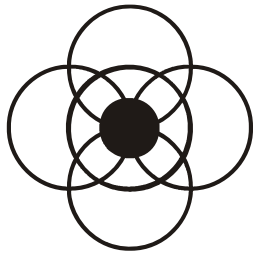
Составить программу "Календарь", которая определяет, является ли данный год високосным. Определение производится по следующим правилам. Если порядковый номер года не делится без остатка на 100, то год считается висо-

косным, если его номер делится на 4 (например, 2004-й), и невисокосным, если его номер не делится на 4 (например, 2005-й). Если же номер года делится без остатка на 100, то год считается високосным, если его порядковый номер делится на 400 (например, 2000-й), и невисокосным, если его порядковый номер не делится на 400 (например, 1900-й).

Указания для выполнения задания. В форме должны быть выведены одно текстовое поле для ввода исходных данных (номера года) и три экранные кнопки: для подтверждения ввода, для очистки текстового окна и для выхода из программы. Для того чтобы поместить в форму иллюстрацию, можно поступить следующим образом: двойным щелчком на системных часах, находящихся на панели задач, открыть диалоговое окно **Свойства: дата и время**, в котором имеется специальное вложенное окно с изображением календаря. Затем сделать копию экрана и в графическом редакторе Paint сохранить изображение календаря в виде отдельного рисунка. Далее этот рисунок поместить в объект `Image`.

При написании кода рекомендуется использовать структуру с вложенными условными операторами. В заголовке основного условного оператора следует поместить условие делимости исходной даты на 100. Затем в каждой из ветвей основного условного оператора необходимо предусмотреть свой алгоритм определения високосного года, который реализуется в виде вложенного условного оператора.

ГЛАВА 16



Программы с циклами в Delphi

В Object Pascal, как и в языке Turbo Pascal, для решения различных задач, в которых происходит многократное повторение одних и тех же действий, широко используются циклы. В данной главе рассматривается разработка проектов в системе Delphi с использованием циклических структур.

16.1. Программа "Факториал"

В качестве примера использования циклов в Object Pascal рассмотрим задачу вычисления факториала некоторого натурального числа. Алгоритм решения данной задачи читателю уже известен на примере Turbo Pascal. Как читатель, конечно, помнит, в данном случае для нахождения искомой величины удобнее всего использовать цикл с заранее известным числом повторений.

Теперь приступим к реализации данной задачи в среде Delphi. Для этого разработаем новый проект, который назовем faktor. Как всегда, работу над проектом мы начинаем с его сохранения и создания графического интерфейса программы. Интерфейс будет включать в себя следующие основные компоненты:

- строка меню, содержащая разделы **Файл** и **Справка**;
- текстовое окно для ввода исходных данных, слева от которого будет расположено поле комментария, поясняющее содержание данного окна;
- кнопка **Ввод**, нажатие которой должно приводить к вычислению искомого результата;
- текстовое окно с результатом, дополненное поясняющей надписью;
- экранная кнопка **Сброс**, используемая для очистки текстовых окон, и кнопка **Выход**, используемая для закрытия окна программы;

- панель с рисунком юмористического содержания, изображающая человека, излучающего радость от того, что ему удалось добиться решения сложной задачи.

Начнем работу над интерфейсом с создания строки меню. Сам процесс создания меню не должен вызывать у пользователя затруднений, т. к. он был подробно описан в предыдущей главе. Отметим только, что каждый из разделов меню содержит по вложенному пункту. Для меню **Файл** (объект `N1`) таким пунктом является **Выход** (объект `N3`), а для раздела **Правка** (объект `N2`) — пункт **Об авторе** (объект `N4`). Выбор этого пункта меню приведет к появлению на экране компьютера дополнительного второго окна, содержащего информацию об авторах программы. Данное окно содержит поясняющую надпись и закрывается при нажатии экранной кнопки **OK**.

Теперь приступаем к настройке основной формы программы. Присваиваем основной форме новый заголовок (`Факториал`) и изменяем цвет формы на светло-зеленый. Следующим компонентом основной формы является текстовое окно `Edit1`. Слева от него расположим окно комментария `Memo1`. Для того чтобы данное окно не выделялось на общем фоне формы, поступаем так же, как и в предыдущем проекте. Цвет окна делаем светло-зеленым, как и у самой формы, а свойству `BorderStyle` присваиваем значение `None`, т. е. границу делаем невидимой. Под текстовым окном располагаем экранную кнопку (объект `Button1`), которой присваиваем заголовок `Ввод`.

Под кнопкой **Ввод** располагаем второе текстовое окно для вывода результатов (объект `Edit2`), а слева от него надпись, говорящую о содержимом данного окна (объект `Label1`). Внизу экрана располагаем две экранные кнопки. Это кнопка **Сброс** (объект `Button2`) и кнопка **Выход** (объект `Button3`). Назначение этих кнопок понятно из их заголовков.

В правой части формы размещаем объект `Panel1`, а на этом объекте поместим иллюстрацию (объект `Image1`). Картинку для данной иллюстрации найдем в коллекции Clipart, входящей в состав пакета Microsoft Office 97. Картинка содержится в файле под названием Amidea. Такие изображения хорошо поддаются масштабированию, поэтому производить точную подгонку объектов под размеры картинки нет необходимости. Затем к основной форме подключаем вспомогательную форму со сведениями об авторе программы и создаем ее интерфейс. На этом создание графического интерфейса программы завершается (рис. 16.1).

Следующим этапом работы над программой является написание кода для пунктов меню и экранных кнопок. Пункт меню `N3` закрывает окно формы, поэтому соответствующий код будет содержать только оператор `close`. Для пункта меню `N4` необходимо написать оператор `Form2.Show`, который выводит на экран компьютера дополнительную форму со сведениями об авторе. При

этом нужно предварительно создать новую форму с помощью раздела меню системы программирования **File | New** (Файл | Новый), а также описать подключение дополнительного модуля для этой формы `Unit2` в разделе основного модуля `Unit1`, который начинается со служебного слова `Uses`.

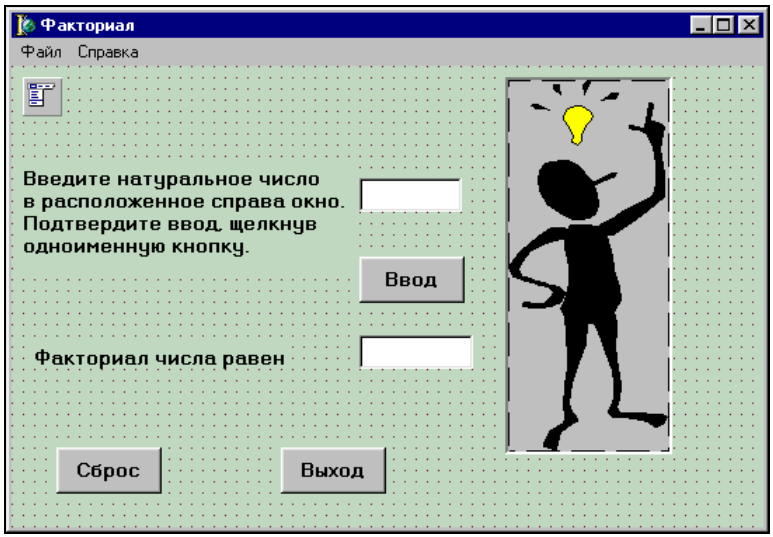


Рис. 16.1. Графический интерфейс программы "Факториал"

Далее нужно написать код, описывающий реакцию на нажатие экранной кнопки **Ввод**, которое должно приводить к вычислению факториала с отображением его в соответствующем текстовом окне. Подпрограмма, описывающая данную реакцию, должна содержать описание используемых переменных (ранее мы рассматривали, какие переменные здесь необходимы), а также, как уже говорилось, средства ввода данных и вывода результатов. Получившаяся подпрограмма должна выглядеть следующим образом:

```
procedure TForm1.Button1Click(Sender: TObject);
var i,f,n:integer;
begin
n:=StrtoInt(Edit1.Text);
f:=1;
for i:=1 to n do
f:=f*i;
Edit2.Text:=InttoStr(f)
end;
```

Теперь нам осталось запрограммировать действия кнопки **Сброс**, очищающей текстовые окна, и кнопки **Выход**, закрывающей основную форму. Напи-

сание соответствующих кодов не должно вызвать у пользователя каких-либо затруднений, т. к. мы уже делали это в предыдущих проектах. В листинге 16.1 приводится исходный код основного модуля данного проекта.

Листинг 16.1. Текст программы вычисления факториала

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, Menus,Unit2, StdCtrls, ExtCtrls;

type
  TForm1 = class(TForm)
    MainMenu1: TMainMenu;
    N1: TMenuItem;
    N2: TMenuItem;
    N3: TMenuItem;
    N4: TMenuItem;
    Memo1: TMemo;
    Edit1: TEdit;
    Button1: TButton;
    Label1: TLabel;
    Edit2: TEdit;
    Panel1: TPanel;
    Image1: TImage;
    Button2: TButton;
    Button3: TButton;
    procedure N3Click(Sender: TObject);
    procedure N4Click(Sender: TObject);
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure Button3Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}
```

```
procedure TForm1.N3Click(Sender: TObject);
begin
close
end;

procedure TForm1.N4Click(Sender: TObject);
begin
Form2.Show
end;

procedure TForm1.Button1Click(Sender: TObject);
var i,f,n:integer;
begin
n:=StrtoInt(Edit1.Text);
f:=1;
for i:=1 to n do
f:=f*i;
Edit2.Text:=InttoStr(f)
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
Edit1.Text:='';
Edit2.Text:=''
end;

procedure TForm1.Button3Click(Sender: TObject);
begin
Close
end;

end.
```

Написание дополнительного модуля `Unit2`, описывающего объекты вспомогательной формы, аналогично написанию таких же модулей в предыдущих проектах и не должно вызвать у читателя затруднений. После написания этого модуля программа готова к работе.

Внешний вид программы после запуска ее на выполнение приведен на рис. 16.2. При работе с данной программой следует обратить внимание на то, что в качестве исходных данных следует вводить натуральные числа не больше 15, т. к. факториалы даже сравнительно небольших по модулю чисел являются большими величинами. Например, факториал числа 12 равен 479 001 600.

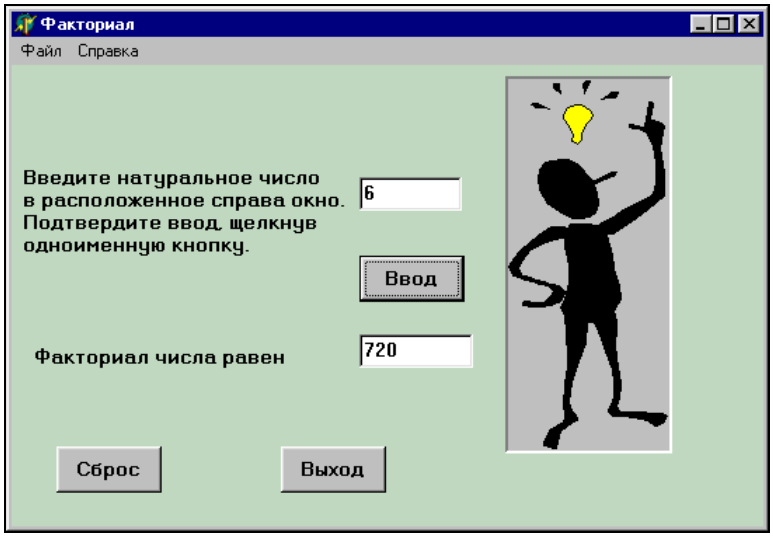


Рис. 16.2. Программа "Факториал" во время работы

16.2. Программа "Вклад"

Следующим проектом, который мы будем разрабатывать в системе Delphi с использованием циклических структур, станет "Вклад". С помощью данного проекта вкладчик, положивший деньги в банк, сможет рассчитать, какую сумму он должен получить по окончании срока действия договора с банком, если по условиям договора вклад положен в банк на определенное количество лет под определенный процент, который не должен изменяться до окончания срока действия договора.

Таким образом, данный проект должен иметь в качестве входных параметров начальную сумму вклада, срок действия договора (для упрощения решения поставленной задачи мы считаем, что количество лет, в течение которых будет действовать договор, является целым числом) и процент по вкладу. Выходным параметром данного проекта будет сумма вклада, накопившаяся после окончания договора. Исходя из сказанного ранее, можно составить общее представление об интерфейсе программы в системе Delphi, который должен включать в себя следующие элементы:

- три текстовых окна, используемых для ввода исходных данных;
- поясняющие надписи к этим окнам;
- две экранные кнопки: одна служит для подтверждения ввода исходных данных (**Ввод**), а вторая — для очистки всех текстовых окон (**Сброс**);

- два текстовых окна для вывода полученных результатов (отдельно для рублей и для копеек);
- пояснительные надписи к упомянутым ранее текстовым окнам;
- иллюстрацию к проекту;
- кнопку, которая закрывает форму.

Создание графического интерфейса программы мы, как обычно, начнем с обработки формы. Для этого мы придаем форме стандартный бледно-зеленый цвет, а в заголовке формы пишем ее название — слово `Банк`. Затем создаем в форме текстовые окна `Edit1`, `Edit2` и `Edit3` для ввода исходных данных. Над каждым из этих окон должна быть сделана надпись, что и реализовано в виде объектов `Label1`, `Label2` и `Label3`. Под окнами ввода располагаются две экранные кнопки: `Button1` и `Button2`. Первая из них используется для подтверждения ввода исходных данных, а вторая — для очистки текстовых окон.

Ниже в форме расположены два текстовых окна `Edit4` и `Edit5`, которые используются для вывода результатов. Окно `Edit4` отображает количество получаемых вкладчиком рублей, а окно `Edit5` — копеек. Над кнопками расположена общая для них пояснительная надпись, которая реализована в виде окна комментария `Memo1`. Кроме того, справа от каждого из двух текстовых окон располагается по небольшой надписи, указывающей единицу измерения для данного окна. Это будут надписи `Label4` и `Label5`.

В правой верхней части формы мы расположим иллюстрацию к проекту в виде растрового рисунка (объекты `Panel1` и `Image1`). В нижней правой части формы будет расположена кнопка **Выход** (`Button3`), которая закрывает форму и завершает работу всего проекта. На этом создание графического интерфейса в данном проекте будет завершено (рис. 16.3).

Создание программного кода начинаем с написания процедуры для экранной кнопки **Ввод** (`Button1`). В данной процедуре мы будем использовать ряд локальных переменных целого и вещественного типа. К целому типу относятся: переменная `srok` — количество лет, на которые заключается договор, `rub` — количество рублей, которое должен получить клиент банка по окончании срока действия договора, `kop` — количество копеек, которое должен получить клиент, `i` — вспомогательная переменная цикла. К вещественному типу относятся: `vkl` — общая сумма вклада, `pro` — проценты по вкладу, `drob` и `drob1` — вспомогательные переменные, назначение которых станет ясно позднее.

В начале работы процедуры мы преобразуем содержимое текстового окна с указанием суммы вклада (`Edit1`) в вещественную величину `vkl`, содержимое

окна Edit2 с указанием срока действия договора — в целую величину srok, содержимое окна Edit3 с указанием процентов по вкладу — в вещественную переменную pro. Преобразование выполняется следующими операторами:

```
vkl:=StrtoFloat(Edit1.Text);
srok:=StrtoInt(Edit2.Text);
pro:=StrtoFloat(Edit3.Text);
```

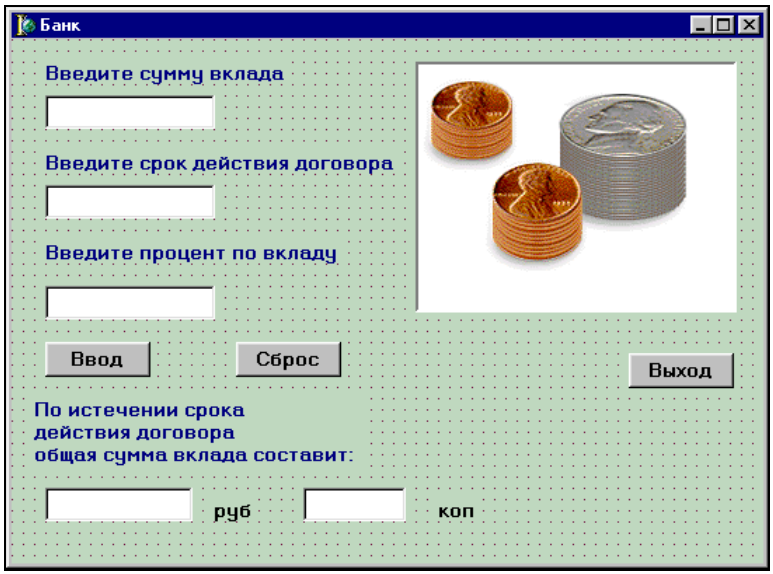


Рис. 16.3. Графический интерфейс программы "Вклад"

Далее в процедуре вычисляется общая сумма вклада по окончании срока действия договора, для чего используется оператор с заранее известным числом повторений. При каждом повторении тела цикла высчитывается прибавка к вкладу за очередной год, для чего сумма, накопившаяся за предыдущий год, умножается на процент по вкладу и делится на 100. Затем получившаяся прибавка добавляется к сумме за прошедший год. Указанные действия реализуются в процедуре следующим образом:

```
for i:=1 to srok do
vkl:=vkl+vkl*pro/100;
```

В полученной таким образом общей сумме вклада необходимо вначале выделить целое количество рублей. Для этой цели мы используем стандартную фунцию trunc, которая как раз и находит целую часть числа, отбрасывая все остальное. Получившаяся величина проходит теперь обратное преобразова-

ние из числовой в строковую, а затем это строковое значение присваивается текстовому окну `Edit`:

```
rub:=trunc(vkl);
Edit4.Text:=InttoStr(rub);
```

Затем из целой части вклада вычитаем количество рублей и получаем дробную часть `drb`, соответствующую количеству копеек. Для того чтобы преобразовать величину `drb` в количество копеек, мы умножаем эту величину на 100 и получаем значение переменной `drb1`. Таким образом мы находим количество копеек `drb1`, но данная переменная содержит наряду с целой и дробную часть. Для того чтобы выделить только целое число копеек `kop`, мы используем функцию `round`, которая округляет вещественную величину до ближайшего целого по математическим правилам. Эти действия описываются с помощью следующих операторов:

```
drb:=vkl-rub;
drb1:=drb*100;
kop:=round(drb1);
```

Целиком же процедура, описывающая работу кнопки `Button1`, будет выглядеть следующим образом:

```
procedure TForm1.Button1Click(Sender: TObject);
var i,srok,rub,kop:integer;
    vkl,pro,drb,drb1:real;
begin
vkl:=StrtoFloat(Edit1.Text);
srok:=StrtoInt(Edit2.Text);
pro:=StrtoFloat(Edit3.Text);
  for i:=1 to srok do
  vkl:=vkl+vkl*pro/100;
rub:=trunc(vkl);
Edit4.Text:=InttoStr(rub);
drb:=vkl-rub;
drb1:=drb*100;
kop:=round(drb1);
Edit5.Text:=InttoStr(kop)
end;
```

Процедура для кнопки `Button2`, которая очищает при сбросе все текстовые окна, и процедура для кнопки `Button3`, которая закрывает окно программы, аналогичны таким же процедурам, рассмотренным ранее, поэтому в детальном их описании необходимости нет. В целом же программный модуль будет выглядеть следующим образом — листинг 16.2.

Листинг 16.2. Текст программы "Вклад"

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, ExtCtrls;

type
  TForm1 = class(TForm)
    Panel1: TPanel;
    Image1: TImage;
    Label1: TLabel;
    Edit1: TEdit;
    Label2: TLabel;
    Edit2: TEdit;
    Label3: TLabel;
    Edit3: TEdit;
    Memo1: TMemo;
    Button1: TButton;
    Edit4: TEdit;
    Button2: TButton;
    Button3: TButton;
    Label4: TLabel;
    Edit5: TEdit;
    Label5: TLabel;
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure Button3Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.Button1Click(Sender: TObject);
var i,srok,rub,kop:integer;
    vkl,pro,drb,drb1:real;
```

```
begin
vkl:=StrtoFloat(Edit1.Text);
srok:=StrtoInt(Edit2.Text);
pro:=StrtoFloat(Edit3.Text);
  for i:=1 to srok do
  vkl:=vkl+vkl*pro/100;
rub:=trunc(vkl);
Edit4.Text:=InttoStr(rub);
drb:=vkl-rub;
drb1:=drb*100;
kop:=round(drb1);
Edit5.Text:=InttoStr(kop)
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
close
end;

procedure TForm1.Button3Click(Sender: TObject);
begin
Edit1.Text:='';
Edit2.Text:='';
Edit3.Text:='';
Edit4.Text:='';
Edit5.Text:=''
end;

end.
```

Результат работы программы для клиента, который положил 5000 рублей в банк сроком на 3 года под 15%, показан на рис. 16.4.

16.3. Программа "Контрольная по математике"

Целью данного проекта является создание тестового задания по математике, с помощью которого можно было бы проверить пользователя на знание всех четырех основных арифметических действий. Для того чтобы произвести такую проверку, пользователю предлагается решить 12 примеров, т. е. по 3 примера на каждую из основных арифметических операций. Для проверки пользователя на знание им правил деления в тесте представлены примеры на нахождение среднего арифметического. Это сделано для упрощения

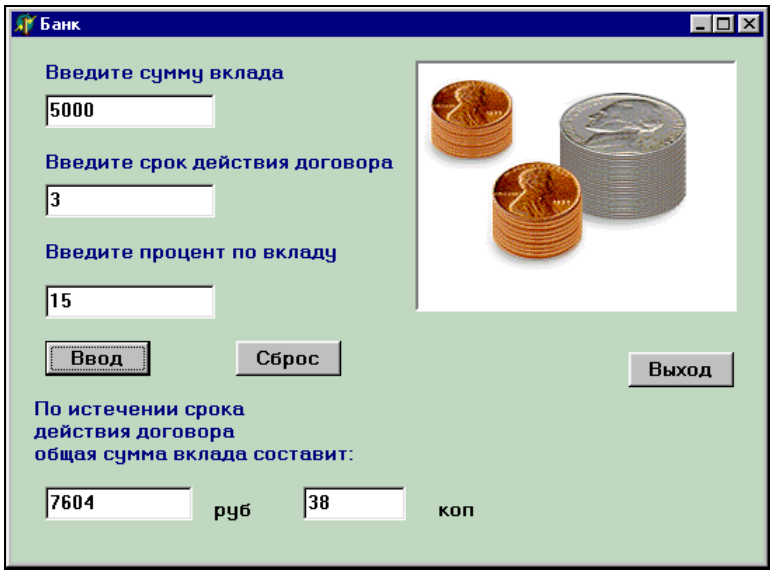


Рис. 16.4. Программа "Вклад" во время работы

разработки программы. По итогам теста пользователю должна выставляться оценка. Если он решил все примеры правильно, то пользователь получает оценку "отлично". Если были допущены одна или две ошибки, то ему ставится оценка "хорошо". За три или четыре ошибки ставится оценка "удовлетворительно", за большее количество ошибок — "неудовлетворительно". По окончании теста полученная оценка должна выводиться в экранную форму.

Создание проекта, как всегда, начнем с создания графического интерфейса пользователя. На экранной форме должны присутствовать следующие элементы:

- надпись, которая сообщает пользователю о том, какой именно пример ему предстоит решать: на сложение, вычитание, умножение или нахождение среднего арифметического;
- надпись, которая выводит исходные данные для примера, т. е. два числа, над которыми будет выполнено действие;
- текстовое поле для ввода ответа пользователя с поясняющей надписью к нему;
- экранная кнопка для подтверждения ввода ответа пользователя;
- экранная кнопка для закрытия формы и выхода из программы;
- иллюстрация к проекту.

Вначале в экранную форму выводим надпись `Label1`, которая будет иметь содержание такого рода: `Чему равна сумма?`, `Чему равна разность?`, `Чему равно`

произведение? или Чему равно среднее арифметическое?. Затем под ней помещаем в форму надпись Label2. Эта надпись будет содержать конкретные числа и знак арифметической операции между ними. Например: 2+5, 8-6, 5*7 и т. д. При решении задачи по нахождению среднего арифметического знак операции будет заменен предлогом "и": 4 и 7.

Под надписями мы расположим текстовое окно Edit1 для ввода ответа пользователя. Над этим текстовым окном будет находиться надпись Label3. Эта надпись может содержать как предписание пользователю вводить ответ именно в окно Edit1, так и сообщение об ошибке в том случае, если пользователь вместо результата вычислений случайно ввел в окно другую информацию нечислового характера. Ниже текстового окна будет находиться экранная кнопка **Ввод** (Button1). При нажатии на эту кнопку программа должна сопоставлять значение, введенное пользователем, с правильным результатом вычислений, который был определен самим компьютером. В случае совпадения этих значений, программа должна добавить пользователю дополнительный балл, а в случае их несовпадения, не должна ничего добавлять к имеющейся сумме баллов. В правом нижнем углу формы будет находиться еще одна экранная кнопка **Выход** (Button2), щелчок на которой завершает работу всей программы.

В правой верхней части формы мы поместим панель Panel1, а поверх нее будет расположена иллюстрация Image1, о создании которой следует сказать особо. Картинка, которая использована в данной иллюстрации, содержится в справочном материале программного комплекса StudyWorks. StudyWorks — это программный комплекс, разработанный фирмой Mathsoft для обучения основам математики. Картинка не является отдельным графическим объектом, как картинки из коллекции Clipart, входящей в состав пакета Microsoft Office, а интегрирована в страницу со справочной информацией.

Для того чтобы этот рисунок можно было вставить в создаваемый проект, используем следующую технологию. Открываем страницу программы StudyWorks со справочной информацией. Создаем копию окна программы комбинацией клавиш <Alt>+<PrintScreen> (содержимое окна при этом копируется в буфер обмена).

Далее запускаем стандартное приложение Windows — программу Paint. Командой **Правка | Вставить** вставляем в открытый документ Paint копию окна из буфера. Затем с помощью кнопки  (выделение) в окне программы Paint выделяем фрагмент, который содержит рисунок. Командой меню **Правка | Копировать** копируем в буфер обмена выделенный фрагмент. Затем командой **Рисунок | Очистить** очищаем окно программы Paint и командой **Правка | Вставить** вставляем в окно Paint фрагмент с рисунком. Получившийся документ Paint с помощью команды **Файл | Сохранить как** сохраняем в папке проекта Delphi для дальнейшего использования под именем kontrol.

Обратите внимание на то, что созданная в Paint картинка является растровым объектом и плохо поддается масштабированию. Поэтому следует определить размеры данного рисунка в пикселах с помощью команды меню **Рисунок | Атрибуты**, а затем, исходя из уже известных размеров, подобрать размеры объектов `Panel1` и `Image1` в проекте. Объект `Image1` может иметь те же размеры, что и рисунок, а объект `Panel` по вертикали и по горизонтали больше на 4 пиксела. Если установить указанные ранее размеры объектов, то внешний вид рисунка не будет искажен. На этом создание интерфейса программы завершено (рис. 16.5).

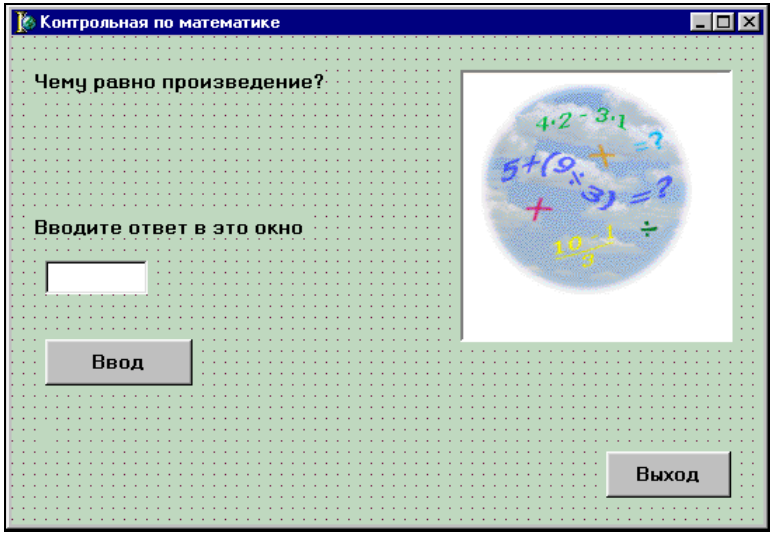


Рис. 16.5. Графический интерфейс программы "Контрольная по математике"

Следующим этапом работы над проектом станет написание программного кода. Для того чтобы разобраться, с чего в данном случае нужно начинать создание программы, следует обратить внимание на то обстоятельство, что сразу после запуска программы на выполнение в программной форме должен уже присутствовать текст первого примера и должен начаться отсчет как общего количества выводимых примеров, так и баллов, получаемых пользователем. Поэтому задание начальных условий целесообразно связать с созданием самой формы. Для этого в инспекторе объектов открываем для формы `Form1` вкладку **Events** (События) и на указанной вкладке находим событие `OnCreate` (создание формы). Дважды щелкнув мышью в поле справа от `OnCreate`, мы открываем заготовку для процедуры `FormCreate`.

Внутри этой процедуры мы выполняем следующие действия. Во-первых, мы запускаем генератор случайных чисел. Он нам понадобится для того, чтобы

при каждом новом запуске проекта в примерах у нас не повторялись каждый раз одни и те же числа, а компьютер каждый раз генерировал новые числа для примеров. Как и в Turbo Pascal, в Object Pascal для этой цели используется процедура `Randomize`. Во-вторых, в проекте необходимо иметь две переменные-счетчики. Одна переменная-счетчик `k` будет подсчитывать общее количество выводимых программой примеров. Вторая переменная-счетчик `t` будет подсчитывать количество правильных ответов, данных пользователем, которому соответствует количество набранных баллов. В процедуре мы должны проинициализировать эти переменные, т. е. присвоить им начальные значения. Причем эти начальные значения будут разными. Для переменной `k` начальное значение будет равно единице, т. к. сразу после создания формы мы уже выведем первый пример. Для переменной `t` начальное значение будет равно нулю, т. к. пользователь не дал еще ни одного ответа.

Следующие операторы процедуры выбирают случайные значения и присваивают их вспомогательным переменным `x` и `y`. Эти действия, как и в Turbo Pascal, производятся с помощью функции `Random`. Затем с помощью функции `InttoStr` значения целочисленных переменных `x` и `y` преобразуются в строковые величины `a` и `b`. Далее пользователю задается первый вопрос `Чему равна сумма?`, для чего эта строка присваивается объекту `Label1`. В заголовке надписи `Label2` выводятся строковые величины `a` и `b`, между которыми ставится знак "+". Таким образом в данной процедуре мы сформировали текст первого вопроса теста. Далее приводится текст данной процедуры:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
randomize;
k:=1; t:=0;
x:=random(9)+1;
y:=random(9)+1;
a:=InttoStr(x);
b:=InttoStr(y);
Label1.Caption:='Чему равна сумма?';
Label2.Caption:=a+'+'+b;
end;
```

После вывода формы на экран и появления в форме первого вопроса пользователь должен ввести в текстовое окно свой вариант ответа и подтвердить его нажатием кнопки **Ввод**. Поэтому следующая процедура, которую мы создаем в данном проекте, должна быть связана со щелчком на кнопке **Ввод**. Найдя для кнопки `Button1` событие `OnClick`, мы открываем заготовку процедуры `Button1Click`. Именно в этой процедуре будет производиться сравнение введенных пользователем результатов с правильными, вывод новых примеров, подсчет количества набранных баллов и выставление конечной оценки.

Разберем, как реализуется указанная последовательность действий в программе. Вначале необходимо для уже созданного компьютером примера найти правильное решение и присвоить его некоторой вспомогательной переменной, которую мы назовем z. Затем следует преобразовать введенный пользователем в текстовое окно ответ из строкового значения в числовое (причем лучше, чтобы эта числовая величина была вещественной, т. к. при нахождении среднего арифметического необязательно получится целый результат). Полученную с помощью функции StrtoFloat числовую величину с мы сравним с "эталонной" z. Если эти величины будут равны между собой, то тестируемый получает дополнительный балл, что выражается в увеличении значения переменной t на единицу.

Далее для следующего вопроса должны быть сгенерированы новые числовые значения x и y, и они должны быть преобразованы в строковые величины a и b для их показа в форме. Затем счетчик количества примеров k должен быть увеличен на единицу, а текстовое окно должно быть очищено для того, чтобы в него можно было вводить новые данные.

Кроме того, желательно, чтобы текстовое окно было в форме активным, т. е. чтобы в нем уже находился текстовый курсор (этот курсор имеет вид вертикальной черты) и не надо было специально щелкать окно мышью перед вводом нового ответа. Такая активизация окна или другого элемента формы называется *установкой фокуса* на нем. Установка фокуса производится с помощью метода SetFocus, который записывается через точку после имени элемента формы, на котором этот фокус устанавливается. Например, для установки фокуса на Edit1 нужно набрать команду Edit1.SetFocus. Для примера на сложение двух чисел вся описанная ранее последовательность действий может быть реализована с помощью следующих операторов:

```
z:=x+y;
c:=StrtoFloat(Edit1.Text);
if c=z then t:=t+1;
x:=random(9)+1;
y:=random(9)+1;
a:=InttoStr(x);
b:=InttoStr(y);
Label1.Caption:='Чему равна сумма?';
Label2.Caption:=a+'+'+b;
Label3.Caption:='Введите ответ в это окно';
k:=k+1;
Edit1.Text:='';
Edit1.SetFocus
```

Рассматривая эту последовательность операторов, следует обратить внимание на следующее обстоятельство. Она может быть реализована в приведен-

ном ранее виде только при значении счетчика k, равном 1 или 2. Данное обстоятельство объясняется следующим образом: всего в программе выводится три примера на сложение. При k, равном 1, проверяется первый пример и формируется текст второго примера, при k, равном 2, выводится второй пример и формируется текст третьего примера — последнего примера на сложение. Если же k равно 3, то проверяется третий пример, а четвертый пример, генерируемый программой, должен быть примером на другое действие, например на вычитание. Для четвертого же и для пятого примеров проверка будет производиться уже на вычитание и генерироваться примеры будут тоже на вычитание. Для того чтобы одну и ту же конструкцию можно было использовать для третьего, четвертого и пятого примеров, можно в начале ее поставить условный оператор. Этот оператор будет проверять, не равно ли k трем. В последнем случае проверка правильности будет производиться для примера на сложение. Далее приводится описанная последовательность операторов:

```
if k=3 then z:=x+y
       else z:=x-y;
c:=StrtoFloat(Edit1.Text);
if c=z then t:=t+1;
x:=random(9)+1;
y:=random(9)+1;
a:=InttoStr(x);
b:=InttoStr(y);
Label1.Caption:='Чему равна разность?';
Label2.Caption:=a+'-'+b;
Label3.Caption:='Введите ответ в это окно';
k:=k+1;
Edit1.Text:='';
Edit1.SetFocus
```

Аналогичную последовательность операторов можно написать для шестого, седьмого и восьмого вопросов (в данном случае генерируются примеры на умножение). Для девятого, десятого и одиннадцатого вопросов подобным же образом генерируются примеры на вычисление среднего арифметического. После того как все примеры сгенерированы программой, остается проверить последний двенадцатый пример, очистить ненужные больше надписи, сделать невидимым текстовое окно, а затем на основании полученных тестируемым баллов вывести ему оценку. Определение оценки производится с помощью оператора case, в котором t выполняет роль переменной-селектора. Эти действия могут быть реализованы с помощью следующих операторов:

```
if c=z then t:=t+1;
Edit1.Visible:=False;
```

```
Label1.Caption:='';
Label2.Caption:='';
Label3.Caption:='';
Button1.Visible:=False;
Label4.Visible:=True;
case t of
0..7:  Label4.Caption:='Ваша оценка равна 2';
8,9:   Label4.Caption:='Ваша оценка равна 3';
10,11: Label4.Caption:='Ваша оценка равна 4';
12:    Label4.Caption:='Ваша оценка равна 5';
end;
```

Таким образом, мы рассмотрели все возможные варианты действий, которые осуществляются при щелчке на кнопке `Button1`. Как понятно из сказанного ранее, конкретный вариант действий определяется значением переменной-счетчика `k`. Следовательно, реакцию кнопки **Ввод** удобнее всего описать в виде оператора множественного выбора `case` с `k` в качестве переменной-селектора. Этот оператор множественного выбора будет выглядеть таким образом:

```
case k of
1..2:  begin
       z:=x+y;
       c:=StrtoFloat(Edit1.Text);
       if c=z then t:=t+1;
       x:=random(9)+1;
       y:=random(9)+1;
      a:=InttoStr(x);
      b:=InttoStr(y);
      Label1.Caption:='Чему равна сумма?';
      Label2.Caption:=a+'+'+b;
      Label3.Caption:='Введите ответ в это окно';
      k:=k+1;
      Edit1.Text:='';
      Edit1.SetFocus
      end;
3..5:  begin
       if k=3 then z:=x+y
       else z:=x-y;
       c:=StrtoFloat(Edit1.Text);
       if c=z then t:=t+1;
       x:=random(9)+1;
       y:=random(9)+1;
       a:=InttoStr(x);
       b:=InttoStr(y);
```

```
       Label1.Caption:='Чему равна разность?';
       Label2.Caption:=a+'-'+b;
       Label3.Caption:='Введите ответ в это окно';
       k:=k+1;
       Edit1.Text:='';
       Edit1.SetFocus
       end;
6..8:  begin
       if k=6 then z:=x-y
              else z:=x*y;
       c:=StrtoFloat(Edit1.Text);
       if c=z then t:=t+1;
       x:=random(9)+1;
       y:=random(9)+1;
       a:=InttoStr(x);
       b:=InttoStr(y);
       Label1.Caption:='Чему равно произведение?';
       Label2.Caption:=a+'*'+b;
       Label3.Caption:='Введите ответ в это окно';
       k:=k+1;
       Edit1.Text:='';
       Edit1.SetFocus
       end;
9..11: begin
       if k=9 then z:=x*y
              else z:=(x+y)/2;
       c:=StrtoFloat(Edit1.Text);
       if c=z then t:=t+1;
       x:=random(9)+1;
       y:=random(9)+1;
         a:=InttoStr(x);
         b:=InttoStr(y);
         Label1.Caption:='Чему равно среднее арифметическое?';
         Label2.Caption:=a+' и '+b;
         Label3.Caption:='Введите ответ в это окно';
         k:=k+1;
         Edit1.Text:='';
         Edit1.SetFocus
        end;
12:    begin
       if c=z then t:=t+1;
       Edit1.Visible:=False;
       Label1.Caption:='';
```

```
        Label2.Caption:='';
        Label3.Caption:='';
        Button1.Visible:=False;
        Label4.Visible:=True;
        case t of
          0..7:  Label4.Caption:='Ваша оценка равна 2';
          8,9:   Label4.Caption:='Ваша оценка равна 3';
          10,11: Label4.Caption:='Ваша оценка равна 4';
          12:    Label4.Caption:='Ваша оценка равна 5';
        end;
      end;
end;
```

Такая процедура, описывающая реакцию экранной кнопки **Ввод**, будет работать правильно, но только до тех пор, пока пользователь правильно вводит исходные данные. Если же пользователь случайно ошибся при вводе данных, например, ввел вместо числового варианта ответа какой-либо словесный, то система программирования отреагирует на это тем, что приостановит работу программы и выдаст пользователю сообщение, подобное тому, которое показано на рис. 16.6.

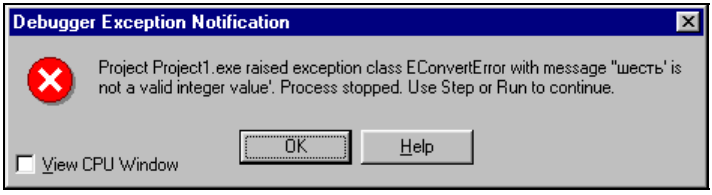


Рис. 16.6. Сообщение об ошибке, выдаваемое системой программирования при попытке пользователя ввести неправильные исходные данные (например, в случае ввода текста в окно, в которое можно вводить только числовые величины)

Неподготовленный пользователь вряд ли поймет что-либо из этого сообщения, и, кроме того, ему скорее всего придется запускать программу на исполнение заново. Для того чтобы избежать подобной ситуации, рекомендуется в программе предусматривать *перехват ошибок*. Под перехватом ошибок понимается возможность обнаружения ошибок пользователя средствами самой программы до их выявления компилятором. При этом должна производиться выдача простых и понятных пользователю сообщений об ошибке и не должно быть принудительного прерывания выполнения самой программы.

Так как в нашем случае основная угроза заключается в возможности неправильного ввода данных в текстовое окно, то после ввода очередного результата необходимо проверить, можно ли его интерпретировать как число. Для этого введенный в окно `Edit1` результат должен быть проверен посимвольно.

Для того чтобы текст можно было преобразовать в числовую величину, нужно, чтобы он содержал только цифры от 0 до 9. Кроме цифр в окне могут находиться запятые, необходимые для отделения целой части от дробной, а также знак "минус". Если хотя бы один символ не принадлежит к указанному множеству, то нужно сообщить об этом пользователю для того, чтобы он мог ввести правильные исходные данные и продолжить выполнение программы. Перехват ошибок реализован с помощью следующего фрагмента программы:

```
flag:=true;
s:=Edit1.Text;
for i:=1 to length(s) do
    if (s[i]<>'0') and (s[i]<>'1') and (s[i]<>'2')
    and (s[i]<>'3')and (s[i]<>'3') and (s[i]<>'4')
    and (s[i]<>'5') and (s[i]<>'6') and (s[i]<>'7')
    and (s[i]<>'8') and (s[i]<>'9') and (s[i]<>'-')
    and (s[i]<>',')
    then flag:=false;
if flag=false
then Label3.Caption:='в это окно можно вводить только числа'
else
```

В приведенном фрагменте для определения корректности или некорректности ввода данных используется вспомогательная переменная `flag`, которая относится к логическому типу. Изначально этой переменной присваивается значение `true`. Строковой переменной `s` присваивается значение, введенное в текстовое окно. Затем с помощью стандартной функции `length` определяется длина введенной строки. Далее работает цикл с заранее известным числом повторений. Для всех символов, входящих в строку, начиная с первого и заканчивая последним, порядковый номер которого равен `length(s)`, производится проверка их на принадлежность к указанному множеству.

Если хотя бы один символ в строке не удовлетворяет заданному условию, то значение переменной `flag` изменяется на `false`. Сразу после завершения работы цикла производится проверка значения переменной `flag`. Если значение ее оказалось равным `false`, то выводится сообщение об ошибке, которое присваивается объекту `Label3`, расположенному над текстовым окном. Если значение `flag` оказалось равным `true`, т. е. ошибок ввода обнаружено не было, то выполняются действия, которые находятся после `else`. После `else` мы и расположим тот оператор множественного выбора, который был создан ранее.

Теперь осталось внести в программу еще одно усовершенствование. Если внимательно посмотреть на варианты, описанные в операторе множественного выбора, то можно увидеть, что все они (за исключением последнего) очень похожи. Начало и конец каждого варианта идентичны и только в централь-

ной части имеются некоторые отличия. Поэтому начало каждого варианта мы можем описать в виде процедуры `preobr1`, входящей в качестве составной части в процедуру `Button1Click`, а конец варианта можно представить также в виде подчиненной процедуры `preobr2`. Обе эти процедуры мы опишем внутри основной процедуры `Button1Click`, а вызываться они будут по мере необходимости из основной процедуры. Данная модификация программы позволит сделать ее более компактной. В листинге 16.3 приводится полный текст программного модуля.

Листинг 16.3. Текст программы "Контрольная по математике"

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, ExtCtrls;

type
  TForm1 = class(TForm)
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    Edit1: TEdit;
    Button1: TButton;
    Label4: TLabel;
    Button2: TButton;
    Panel1: TPanel;
    Image1: TImage;
    procedure FormCreate(Sender: TObject);
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);

     private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;
  x,y,k,t:integer;
  c,z:real;
```

```
  flag:boolean;
  a,b,s:string;

implementation

{$R *.dfm}

procedure TForm1.FormCreate(Sender: TObject);
begin
randomize;
k:=1; t:=0;
x:=random(9)+1;
y:=random(9)+1;
a:=InttoStr(x);
b:=InttoStr(y);
Label1.Caption:='Чему равна сумма?';
Label2.Caption:=a+'+'+b;
end;

procedure TForm1.Button1Click(Sender: TObject);
var i:integer;

procedure preobr1;
begin
c:=StrtoFloat(Edit1.Text);
if c=z then t:=t+1;
x:=random(9)+1;
y:=random(9)+1;
a:=InttoStr(x);
b:=InttoStr(y);
end;

procedure preobr2;
begin
Label3.Caption:='Введите ответ в это окно';
k:=k+1;
Edit1.Text:='';
Edit1.SetFocus
end;

begin
randomize;
flag:=true;
s:=Edit1.Text;
for i:=1 to length(s) do
    if (s[i]<>'0') and (s[i]<>'1') and (s[i]<>'2')
```

```
    and (s[i]<>'3')and (s[i]<>'3') and (s[i]<>'4')
    and (s[i]<>'5') and (s[i]<>'6') and (s[i]<>'7')
    and (s[i]<>'8') and (s[i]<>'9') and (s[i]<>'-')
    and (s[i]<>',')
    then flag:=false;
if flag=false
then Label3.Caption:='в это окно можно вводить только числа'
else
case k of
1..2:  begin
       z:=x+y;
       preobr1;
       Label1.Caption:='Чему равна сумма?';
       Label2.Caption:=a+'+'+b;
       preobr2
       end;
3..5:  begin
       if k=3 then z:=x+y
              else z:=x-y;
         preobr1;
         Label1.Caption:='Чему равна разность?';
         Label2.Caption:=a+'-'+b;
         preobr2
         end;
6..8:  begin
       if k=6 then z:=x-y
              else z:=x*y;
        preobr1;
        Label1.Caption:='Чему равно произведение?';
        Label2.Caption:=a+'*'+b;
        preobr2
        end;
9..11: begin
       if k=9 then z:=x*y
              else z:=(x+y)/2;
        preobr1;
        Label1.Caption:='Чему равно среднее арифметическое?';
        Label2.Caption:=a+' и '+b;
        preobr2;
        end;
12:    begin
       if c=z then t:=t+1;
       Edit1.Visible:=False;
```

```
        Label1.Caption:='';
        Label2.Caption:='';
        Label3.Caption:='';
        Button1.Visible:=False;
        Label4.Visible:=True;
          case t of
          0..7:  Label4.Caption:='Ваша оценка равна 2';
          8,9:   Label4.Caption:='Ваша оценка равна 3';
          10,11: Label4.Caption:='Ваша оценка равна 4';
          12:    Label4.Caption:='Ваша оценка равна 5';
          end;
        end;
end;
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
close
end;

end.
```

Рис. 16.7. Программа "Контрольная по математике" во время работы при правильном вводе данных

Этот программный модуль включает в себя также еще одну процедуру `Button2Click`, которая описывает реакцию на нажатие кнопки **Выход**

(Button2). Эта процедура аналогична другим таким же и поэтому не требует специальных пояснений. На рис. 16.7 показано, как выглядит программа тестирования по математике во время работы при правильном вводе данных, а на рис. 16.8 показано, что будет происходить при попытке ввода неправильных исходных данных.

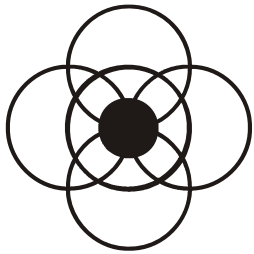
Рис. 16.8. Реакция программы "Контрольная по математике" на неправильный ввод данных

Задание для самостоятельной работы

Разработать в среде Delphi проект, который бы решал задачу Фибоначчи, описанную в *разд. 8.3*. Для решения задачи можно использовать тот же рекурсивный алгоритм, который был описан в *главе 8*.

Указания к проекту. Графический интерфейс данного проекта должен включать в себя текстовое окно для ввода исходных данных (количества месяцев), поясняющую надпись к текстовому окну и надпись для вывода результата. На форме проекта должны присутствовать три экранные кнопки: кнопка **Ввод** — для получения результата, **Сброс** — для очистки текстового окна и надписи с результатом, **Выход** — для завершения работы программы. В правой части формы желательно предусмотреть место для вывода иллюстрации к проекту. В качестве такой иллюстрации можно взять рисунок rabbit.wmf, который в пакете Microsoft Office 97 находится в папке Clipart\Popular.

ГЛАВА 17



Решение математических задач в Delphi

Интегрированная среда Delphi может быть использована не только для решения задач элементарной математики. Данная среда позволяет создавать проекты для удобного и наглядного решения задач из области высшей математики. Примеры таких задач и рассматриваются в данной главе.

17.1. Программа "Вычисление производной"

Согласно определению, производная некоторой функции $y = f(x)$ в точке x_0 является пределом отношения приращения функции к приращению ее аргумента при стремлении приращения аргумента к нулю:

$$y'(x_0) = f'(x_0) = \lim_{\Delta x \to 0} \frac{\Delta y}{\Delta x}, \text{ где } \Delta x = x - x_0,\ \Delta y = f(x_0 + \Delta x) - f(x_0).$$

Алгоритм вычисления производной по ее определению сводится к следующему: последовательно вычисляют величину $\Delta y/\Delta x$ с последовательным уменьшением величины Δx. Это уменьшение производится до тех пор, пока разность между последними двумя вычисленными значениями частного не станет меньше некоторой заданной величины ε. Последнее полученное частное и принимают в качестве искомой величины производной.

Вычисление производной с помощью описанного ранее алгоритма мы реализуем в среде Delphi, где его конкретная реализация будет осуществлена в проекте "вычисление производной" для функции $y = 3 \cdot x^2 - 4 \cdot x$. В проекте мы создадим форму с тремя текстовыми окнами. В первое из этих окон мы будем вводить значение аргумента в той точке функции, где мы будем вычислять производную. Во втором окне будем указывать требуемую точность вычислений. В третьем окне будет выводиться результат — значение производной в заданной точке.

Работу над проектом начнем, как обычно, с создания его графического интерфейса. В основной форме мы изменяем значение ее свойства `Caption`, введя `Вычисление производной`. Цвет формы сделаем серебристым (`clSilver`). Затем в самой форме необходимо поместить надпись или окно комментария, которое говорило бы о том, для какой именно функции мы вычисляем производную.

Здесь, однако, мы сталкиваемся со следующей проблемой: для функции $y = 3 \cdot x^2 - 4 \cdot x$ цифру 2 нужно написать в верхнем индексе. Это можно сделать в текстовом процессоре Microsoft Word, но в строковом редакторе Delphi, который употребляется для заполнения строк надписи или окна комментария, возможность использования верхних и нижних индексов отсутствует. Поэтому воспользуемся следующим приемом. Вначале мы напишем в две строки текст "Вычисление производной функции $y=3 \cdot x^2 - 4 \cdot x$" в MS Word. В программе Word выделим цифру 2 в тексте и переведем ее в верхний индекс, для чего используем команду меню **Формат | Шрифт**. Эта команда открывает диалоговое окно, в котором, установив флажок **верхний индекс**, мы переведем в верхний индекс цифру 2. С помощью команды **Вставка | Символ** вставляем точку — знак умножения между сомножителями. В результате текст, набранный нами в программе Word, будет иметь следующий вид:

```
Вычисление производной
функции y=3·x2–4·x
```

Далее необходимо произвести заливку созданного текста светло-серым цветом, идентичным цвету формы нашего проекта. Затем сделаем копию окна Word с помощью комбинации клавиш <Alt>+<PrintScrn>. Полученную копию можно вставить в программу Paint, являющуюся стандартным приложением Windows, где можно затем скопировать только интересующий нас фрагмент окна, т. е. две текстовые строки. Для этого командой **Правка | Вставить** вставляем в открытый документ Paint копию окна из буфера. Далее с помощью кнопки  (выделение) в окне программы Paint выделяем фрагмент, который содержит две указанные ранее текстовые строки. Командой меню **Правка | Копировать** копируем в буфер обмена выделенный фрагмент. Затем командой меню **Рисунок | Очистить** очищаем окно программы Paint и командой **Правка | Вставить** вставляем в окно Paint фрагмент с текстовыми строками.

Получившийся документ Paint с помощью команды **Файл | Сохранить как** сохраняем в папке нашего проекта под именем funct для дальнейшего использования. Затем в основной форме проекта мы создаем объект `Image1`. Для свойства `Picture` данного объекта в качестве источника мы указываем уже имеющийся в папке проекта файл funct, который мы и загружаем в `Image1`. Таким образом, мы создаем на форме необходимую надпись, но уже в виде иллюстрации.

На следующем этапе мы создаем текстовое окно `Edit1` для ввода значения `x`, окно `Edit2` для ввода требуемой точности вычислений и окно `Edit3` для вывода интересующего нас результата. Для того чтобы пользователю было понятно, что именно нужно вводить в данные окна, мы снабжаем каждое из них поясняющей надписью. Таким образом, на форме появляются надписи `Label1`, `Label2`, `Label3` и `Label4` (надписи `Label2` и `Label3` относятся к одному окну `Edit2`). Рядом с окном `Edit2` помещаем кнопку `Button1` (**Ввод**). Щелчок мышью на этой кнопке должен приводить к вводу исходных данных и вычислению искомого значения производной. В правой нижней части формы мы расположим еще две экранные кнопки. Кнопка `Button2` (**Выход**) должна закрывать форму и завершать работу программы, а кнопка `Button3` (**Сброс**) — очищать все текстовые окна в том случае, если пользователь хочет ввести в процессе работы новые исходные данные.

В верхней правой части формы мы поместим иллюстрацию, которая наглядно показывает геометрический смысл вычисления производной. Для этого мы создаем в форме объект `Panel1`, на котором располагаем объект `Image1`. Картинка, которая использована в данной иллюстрации, содержится в справочном материале программного комплекса StudyWorks. Картинка не является отдельным графическим объектом, как картинки из коллекции Clipart, входящей в состав пакета Microsoft Office, а интегрирована в страницу со справочной информацией. Технология создания на основе этой страницы растрового графического файла и его использования в проекте аналогична той, которую мы использовали для разработки интерфейса проекта "Контрольная по математике" *(см. разд. 16.3)*. На этом создание интерфейса программы завершено (рис. 17.1).

Далее мы приступаем к разработке программного кода для нашего проекта. Вычисление производной будет происходить при щелчке на экранной кнопке `Button1`. Поэтому мы создаем в программе процедуру `Button1Click`. В этой процедуре мы используем ряд локальных переменных, к которым относятся:

- `x` — значение аргумента, задаваемое пользователем с клавиатуры;
- `dx` — приращение значения аргумента;
- `dy1` и `dy2` — приращения значения функции;
- `e` — разница между двумя приращениями значения функции;
- `ekon` — требуемая точность вычислений, задаваемая пользователем с клавиатуры.

Кроме того, внутри процедуры создается функция `dy`. Параметрами функции являются `iks` — значение аргумента в некоторой точке, и `diks` — приращение значение аргумента. Значением данной функции является отношение прира-

щения функции y и приращения аргумента x. Приращение же функции y в свою очередь определяется как разница между значением функции в точке iks+diks и значением функции в точке iks. Тогда функция dy будет выглядеть следующим образом:

```
function dy(iks,diks:real):real;
begin
  dy:=((3*(iks+diks)*(iks+diks)-4*(iks+diks))-(3*iks*iks-4*iks))/diks
end;
```

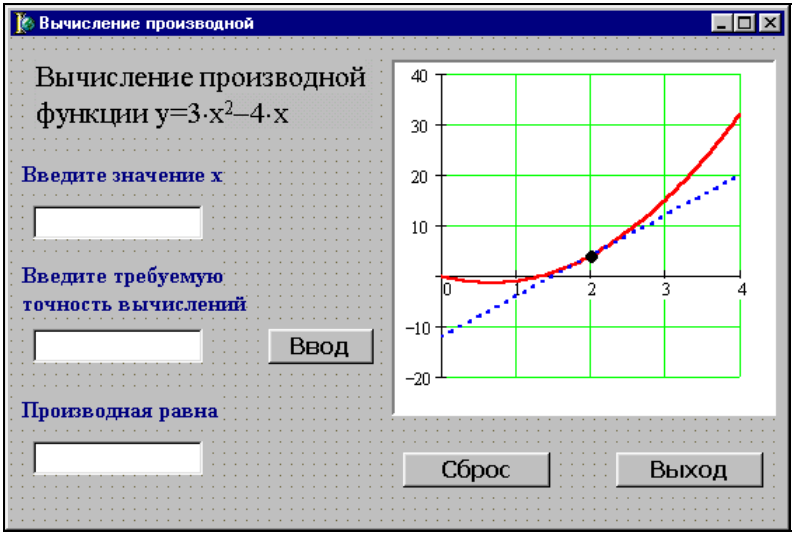


Рис. 17.1. Графический интерфейс программы "Вычисление производной"

В основной части программы мы сперва определяем значения x и ekon путем перевода строковых величин, введенных пользователем в текстовые окна Edit1 и Edit2, в вещественные. Затем задается начальное значения приращения аргумента dx, которое берется равным 0,0001. Далее из основной части программы производится обращение к функции dy, что позволяет вычислить приращение функции dy1, соответствующее начальному приращению аргумента.

Затем в программе организуется цикл с постусловием. При каждом очередном выполнении тела цикла производится уменьшение значения приращения аргумента в k раз. Коэффициент k мы берем равным 10. После уменьшения приращения аргумента мы с помощью обращения к функции dy находим соответствующее приращение значения функции dy2. После этого мы определяем разницу между двумя приращениями и присваиваем приращению dy1

значение приращения `dy2`. При следующем повторении тела цикла (если оно будет) мы будем сравнивать с последующим уже это новое значение. Цикл будет выполняться до тех пор, пока очередная разница между двумя соседними приращениями не станет меньше заданной точности вычислений. Полученное значение `dy2` и будет искомой производной.

Теперь осталось вывести полученное значение переменной в текстовом окне `Edit3`. Для этого мы, как обычно, воспользуемся стандартной функцией `FloattoStr`. Однако здесь мы сталкиваемся со следующей проблемой: полученное значение будет выведено со многими знаками после запятой. Практически же данным методом мы может вычислить значение до 3—4 знака после запятой. Поэтому желательно "обрезать" полученную строковую величину. Для этого мы воспользуемся еще одной стандартной функцией языка Object Pascal. Стандартная функция `copy` позволяет вывести только часть строковой переменной, начиная с символа, номер которого указан пользователем. В данном случае мы выведем в текстовом окне первые шесть символов строковой переменной, начиная с первого. На этом работа над процедурой закончена.

Кроме описанной ранее в программе используются еще две процедуры для кнопок **Выход** (`Button3`) и **Сброс** (`Button2`). Процедура `Button3Click` закрывает экранную форму и всю программу, а процедура `Button2Click` очищает все текстовые окна, если нужно ввести новые исходные данные. Эти процедуры аналогичны таким же процедурам, описанным в предыдущих главах. В листинге 17.1 приводится полный текст программы.

Листинг 17.1. Текст программы "Вычисление производной"

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, ExtCtrls;

type
  TForm1 = class(TForm)
    Image1: TImage;
    Label1: TLabel;
    Edit1: TEdit;
    Label2: TLabel;
    Label3: TLabel;
    Edit2: TEdit;
```

```
    Button1: TButton;
    Label4: TLabel;
    Edit3: TEdit;
    Button2: TButton;
    Panel1: TPanel;
    Image2: TImage;
    Button3: TButton;
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure Button3Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.Button1Click(Sender: TObject);

  const
          k=10;
  var
          dx,dy1,dy2,e,ekon,x:real;
function dy(iks,diks:real):real;
begin
  dy:=((3*(iks+diks)*(iks+diks)-4*(iks+diks))-(3*iks*iks-4*iks))/diks
end;

begin
 x:=StrtoFloat(Edit1.Text);
 ekon:=StrtoFloat(Edit2.Text);
 dx:=0.0001;
 dy1:=dy(x,dx);
 repeat
   dx:=dx/k;
   dy2:=dy(x,dx);
   e:=dy2-dy1;
   dy1:=dy2
```

```
 until e<ekon;
 Edit3.Text:=copy(FloattoStr(dy2),1,6)

end;

procedure TForm1.Button2Click(Sender: TObject);
begin
Edit1.Text:='';
Edit2.Text:='';
Edit3.Text:=''
end;

procedure TForm1.Button3Click(Sender: TObject);
begin
close
end;

end.
```

На рис. 17.2 показано, как выглядит окно программы "Вычисление производной" во время ее работы при вычислении производной функции для $x = 2$.

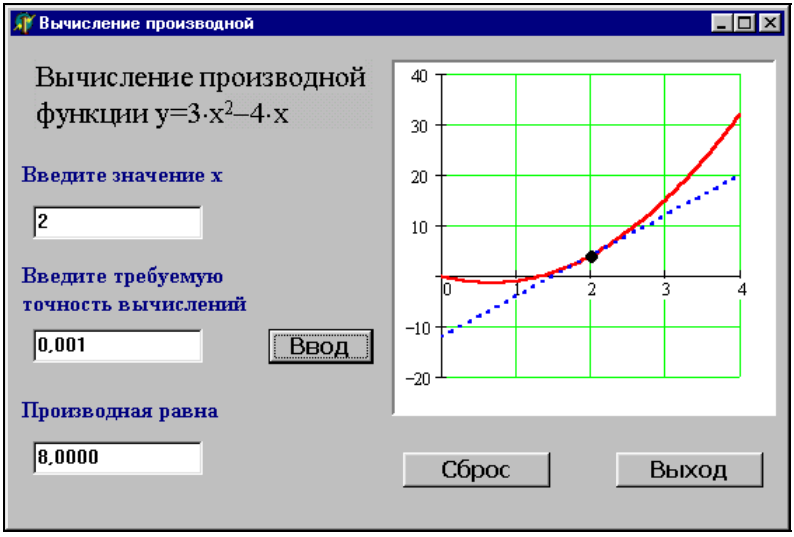


Рис. 17.2. Программа "Вычисление производной" во время работы

17.2. Программа "Интеграл"

Вычисление определенного интеграла геометрически можно интерпретировать как вычисление площади фигуры, которая ограничена линией графика

подинтегральной функции, осью абсцисс и двумя вертикальными линиями, соответствующими верхнему и нижнему пределам интегрирования. Такую фигуру можно разбить на ряд участков, каждый из которых по своей форме близок к трапеции. Площадь каждой такой трапеции можно вычислить, а затем сложить получившиеся площади и найти таким образом общую площадь фигуры, т. е. искомый интеграл. На этом и основан способ приближенного вычисления определенного интеграла, который так и называется методом трапеций.

Вычисление значеия определенного интеграла $\int_a^b f(x)dx$ по методу трапеций производится следующим образом: участок от a до b разбивается на n равных по величине отрезков. Длина каждого отрезка $h = (b - a)/n$. Значения аргумента в точках разбиения будут соответственно равны: $x_0 = a$, $x_1 = a + h$, ..., $x_i = x_0 + ih$, ..., $x_n = b$. Значения функции $f(x)$ в точках x_i будут соответственно обозначаться y_i. Тогда согласно формуле трапеций определенный интеграл можно будет вычислить следующим образом:

$$\int_a^b f(x)dx \approx h(\frac{y_0 + y_n}{2} + y_1 + ...y_i + ... + y_n) .$$

Перед тем как приступить к разработке проекта "интеграл", составим фрагмент программы на языке Object Pascal, который реализует указанный ранее метод на компьютере. В качестве подинтегральной функции мы используем тригонометрическую функцию $y = \sin(x)$. В данном фрагменте будут использованы следующие обозначения:

- `n` — количество отрезков, на которые разбивается участок интегрирования;
- `i` — вспомогательная переменная цикла;
- `a` — начальный предел интегрирования;
- `b` — конечный предел интегрирования;
- `h` — длина отрезка интегрирования;
- `yn` — значение подинтегральной функции в начальной точке (точка `a`);
- `yk` — значение подинтегральной функции в конечной точке (точка `b`);
- `yp` — одно из промежуточных значений подинтегральной функции;
- `s` — искомое значение определенного интеграла.

Первые две переменные должны относиться к целочисленному типу, а все остальные — к вещественному. Тогда описание переменных должно выглядеть следующим образом:

```
var i,n:integer;
    a,b,s,h,x,yp,yn,yk:real;
```

Сама формула трапеции реализована на языке Object Pascal в следующем фрагменте программы, в котором для расчетов использован цикл с заранее известным числом повторений:

```
h:=(b-a)/n;
yp:=0;
x:=a;
for i:=1 to n-1 do
begin
x:=x+h;
yp:=yp+sin(x)
end;
yn:=sin(a);
yk:=sin(b);
s:=((yk+yn)/2+yp)*h;
```

Далее мы приступаем к разработке графического интерфейса данной программы. Интерфейс программы должен включать в себя следующие элементы:

- строка меню, содержащего разделы **Файл** и **Справка**. Первый раздел включает в себя пункт **Выход**, а второй — пункт **Об авторе**. Выполнение данного пункта выводит на экран компьютера дополнительную форму со справочной информацией и закрывающей кнопкой;
- текстовое окно для ввода верхнего предела интегрирования, над которым находится поле комментария, поясняющее, что вводится в текстовое окно;
- текстовое окно для ввода нижнего предела интегрирования, над которым находится соответствующее поле комментария;
- текстовое окно для ввода количества участков, на которые разбивается отрезок интегрирования;
- кнопка **Вычислить**. Щелчком на этой кнопке должно производиться вычисление определенного интеграла;
- кнопка **Сброс**. Щелчок на данной кнопке очищает окна, содержащие исходные данные и результат вычислений;
- текстовое окно для вывода полученного значения интеграла с расположенным над ним окном комментария;
- панель с иллюстрацией, поясняющей суть процесса вычисления определенного интеграла.

Создание интерфейса начинаем с работы над основной формой проекта. В начале переименовываем объект, который получает название `Вычисление определенного интеграла функции y=sin(x)`. Затем изменяем цвет формы на небесно-голубой.

Далее создаем главное меню, которое имеет разделы **Файл** (объект `N1`) и **Справка** (объект `N2`). Эти разделы имеют соответственно пункты **Выход** (объект `N3`) и **Об авторе** (объект `N4`). Создание и настройка меню производится аналогично тому, как это было сделано в предыдущих проектах. Выбор пункта меню об авторе приводит к выводу на экран дополнительной формы справочного характера. Создание и настройка данного окна также производится аналогично предыдущим проектам.

Следующим этапом работы над проектом является создание трех групп аналогичных объектов, используемых для ввода исходных данных. Это текстовое окно `Edit1` для ввода верхнего предела интегрирования и окно комментария `Memo1`, текстовое окно `Edit2` для ввода нижнего предела интегрирования и окно комментария `Memo2`, текстовое окно `Edit3` для ввода количества участков, на которые разбивается отрезок интегрирования, и окно комментария `Memo3`.

Под этими объектами создаем две экранные кнопки. Это кнопка **Вычислить** (объект `Button1`) и кнопка **Сброс** (объект `Button2`). Первая кнопка подсчитывает значения определенного интеграла функции $\sin(x)$, а вторая очищает все текстовые окна, включая окно результата. Под экранными кнопками располагаем текстовое окно `Edit4` для вывода полученного значения интеграла и поясняющую надпись `Label1`.

Последнюю группу объектов данной программы мы создаем в правой части формы. Это панель (объект `Panel1`), содержащая иллюстрацию к программе, поясняющую смысл указанного ранее метода вычисления (объект `Image1`). На создании графического объекта остановимся подробнее.

Картинка, которая использована в данной иллюстрации, содержится в справочном материале программного комплекса StudyWorks. Для того чтобы этот рисунок можно было вставить в создаваемый проект, используем ту же технологию, что и в предыдущем проекте. На этом создание интерфейса данного проекта завершено (рис. 17.3).

Далее приступаем к созданию программного кода. Подпрограммы, соответствующие пунктам меню `N3` и `N4`, аналогичны соответствующим пунктам в предыдущих проектах. В качестве основы для процедуры `Button1Click`, описывающей реакцию на нажатие кнопки **Вычислить**, берем фрагмент программы, приведенный в начале данной главы, и дополняем его операторами, описывающими преобразование исходных данных из строковой формы в числовую, и оператором, преобразующим полученный результат из числовой

формы в строковую. Подпрограмма `Button2Click`, описывающая реакцию на щелчок по кнопке **Сброс**, аналогична соответствующим подпрограммам в предыдущих проектах. В листинге 17.2 приводится полный текст программного кода данного проекта.

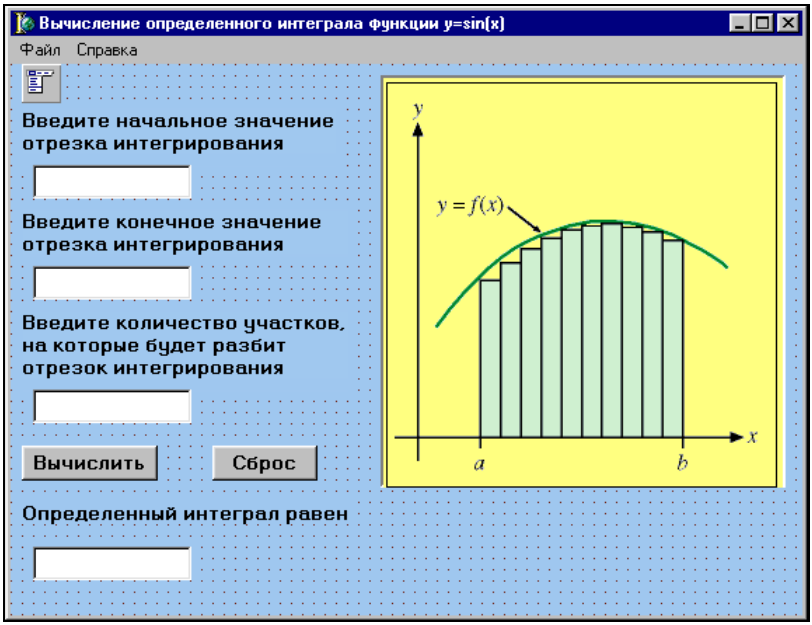


Рис. 17.3. Графический интерфейс программы "Интеграл"

Листинг 17.2. Текст программы вычисления определенного интеграла

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, Menus, StdCtrls, ExtCtrls,Unit2;

type
  TForm1 = class(TForm)
    MainMenu1: TMainMenu;
    N1: TMenuItem;
    N2: TMenuItem;
    N3: TMenuItem;
    N4: TMenuItem;
```

```
    Memo1: TMemo;
    Edit1: TEdit;
    Memo2: TMemo;
    Edit2: TEdit;
    Memo3: TMemo;
    Edit3: TEdit;
    Button1: TButton;
    Button2: TButton;
    Label1: TLabel;
    Edit4: TEdit;
    Panel1: TPanel;
    Image1: TImage;
    procedure N3Click(Sender: TObject);
    procedure N4Click(Sender: TObject);
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.N3Click(Sender: TObject);
begin
close
end;

procedure TForm1.N4Click(Sender: TObject);
begin
Form2.Show
end;

procedure TForm1.Button1Click(Sender: TObject);
var a,b,s,h,x,yp,yn,yk:real; i,n:integer;
begin
a:=StrtoFloat(Edit1.Text);
b:=StrtoFloat(Edit2.Text);
n:=StrtoInt(Edit3.Text);
h:=(b-a)/n;
```

```
yp:=0;
x:=a;
for i:=1 to n-1 do
begin
x:=x+h;
yp:=yp+sin(x)
end;
yn:=sin(a);
yk:=sin(b);
s:=((yk+yn)/2+yp)*h;
Edit4.Text:=copy(FloattoStr(s),1,6)
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
Edit1.Text:='';
Edit2.Text:='';
Edit3.Text:='';
Edit4.Text:=''
end;

end.
```

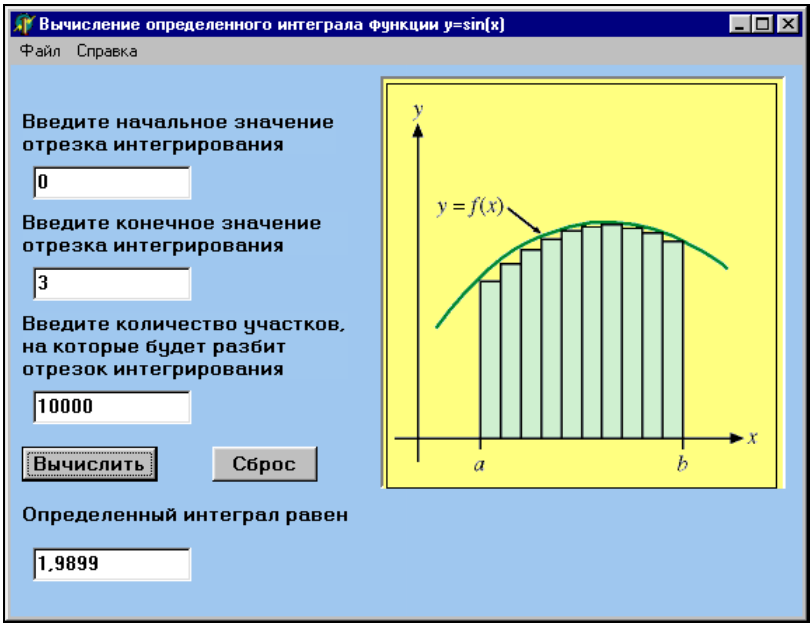
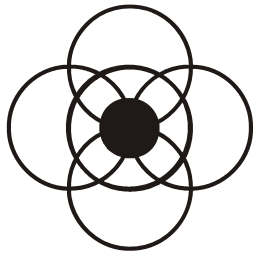


Рис. 17.4. Программа "Интеграл" во время работы

Завершающим этапом работы над программой является, как обычно, написание кода для вспомогательной формы. После этого программа готова к запуску на выполнение.

Внешний вид программы после ее запуска на выполнение приведен на рис. 17.4. При работе с программой следует обратить внимание на то, что для правильной ее работы исходные данные (верхний и нижний пределы интегрирования) нужно вводить не в градусах, а в радианах.



ГЛАВА 18

Delphi в Интернете

В настоящее время Delphi является одной из ведущих систем, используемых для разработки программного обеспечения, как в нашей стране, так и за рубежом, и, естественно, что в глобальной информационной сети имеется большое количество материалов полностью или частично посвященных программированию в Delphi. Например, в популярной поисковой системе **Апорт** имеется тематический каталог, в котором есть, конечно, и раздел **Программирование**. Внутри данного раздела имеется и специальная рубрика, посвященная программированию в Delphi. По состоянию на февраль 2005 года данная рубрика содержала ссылки на 67 различных сайтов. Исходя из этих данных, Delphi занимает по популярности первое место, опережая C (53 ссылки), Visual Basic (31 ссылка) и все прочие языки (рис. 18.1). Кроме перечисленных в каталогах, существует еще много других сайтов по этой тематике.

Понятно, что пользователю, который только начинает осваивать программирование в Delphi, трудно сразу правильно сориентироваться в таком большом потоке информации. Поэтому авторы книги решили в данной главе указать те источники во всемирной сети, которые на их взгляд могут быть наиболее полезны для начинающих программистов. Естественно, что первоочередное внимание в этом обзоре будет уделено русскоязычным сайтам, содержащим важную и полезную информацию по работе в системе Delphi.

Знакомство с этими источниками мы начнем с сайта компании Borland — разработчика системы.

http://www.borland.com — на этом сайте пользователь может найти информацию о самой компании, производимых ею программных продуктах и многом другом (рис. 18.2). Информация на сайте представлена на английском языке.

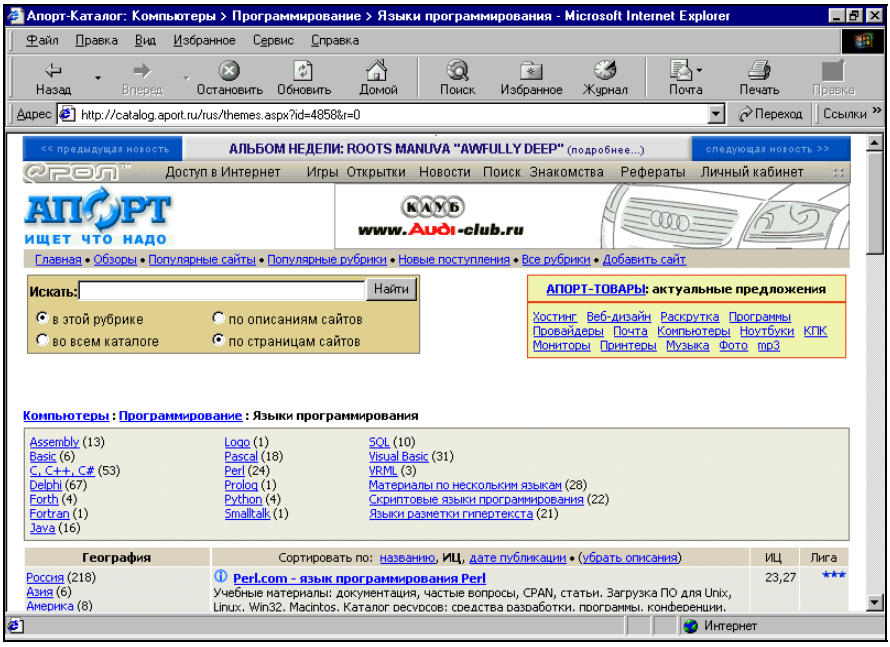


Рис. 18.1. Раздел каталога поисковой системы **Апорт,** посвященный языкам программирования

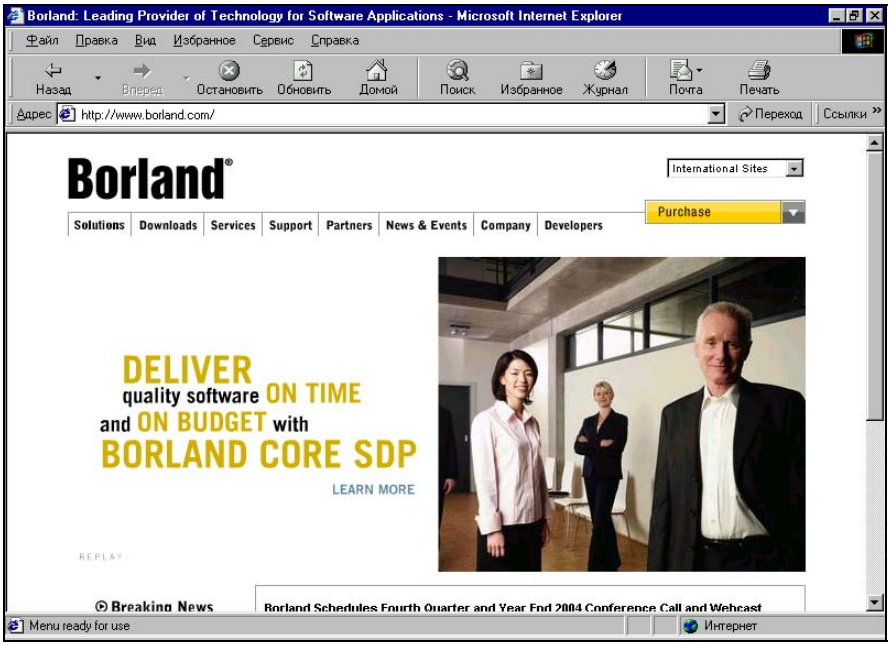


Рис. 18.2. Основная страница сайта компании Borland

На сайте имеется раздел **Downloads**, в котором есть подраздел **Delphi**. В этом подразделе в заархивированном виде имеются файлы, содержащие оригинальную документацию по различным версиям Delphi (рис. 18.3). Имеющаяся документация охватывает версии, начиная с третьей и заканчивая шестой.

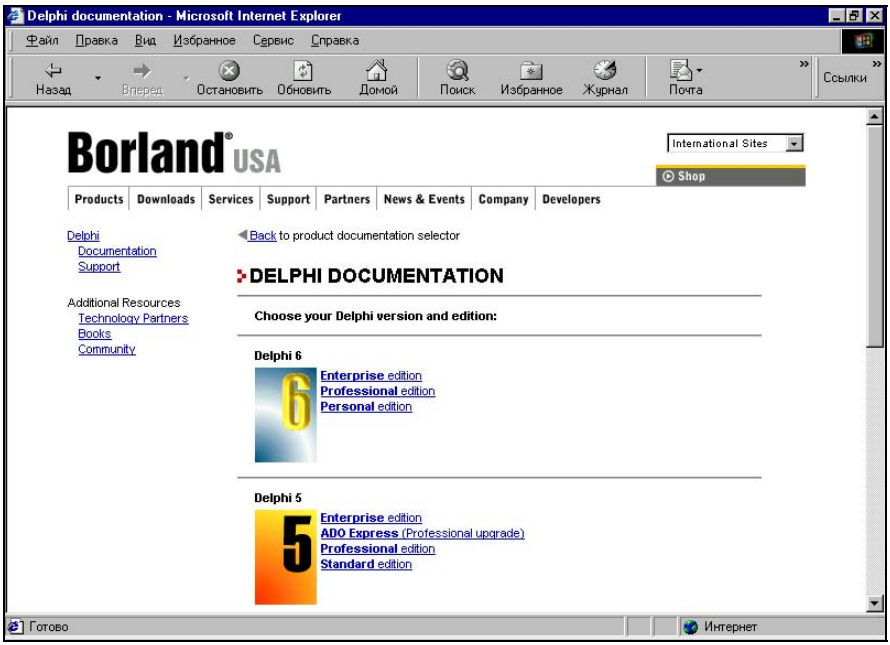


Рис. 18.3. Страница сайта компании Borland, содержащая файлы документации по различным версиям Delphi

Фирма Borland имеет представительство в России и странах СНГ. У этого представительства также есть свой сайт.

http://www.borland.ru — на данном сайте есть информация на русском языке, но есть и ссылки на англоязычные страницы сайта фирмы Borland (рис. 18.4).

Сайт "Development и Дельфи" расположен в глобальной сети по следующему адресу **http://delphid.dax.ru**. Здесь имеется подборка статей по программированию в Delphi, примеры программ на Delphi, ссылки на другие сайты со сходной тематикой (рис. 18.5).

На сайте есть специальный раздел **Курс начинающего программиста в Delphi** (рис. 18.6). В этом разделе имеется ряд электронных книг и учебников по системе Delphi, которые можно скачать.

Сайт "Все для Delphi программиста" находится по адресу **http://www.delphi.dax.ru**. Здесь можно найти ссылки на ресурсы по Delphi, приме-

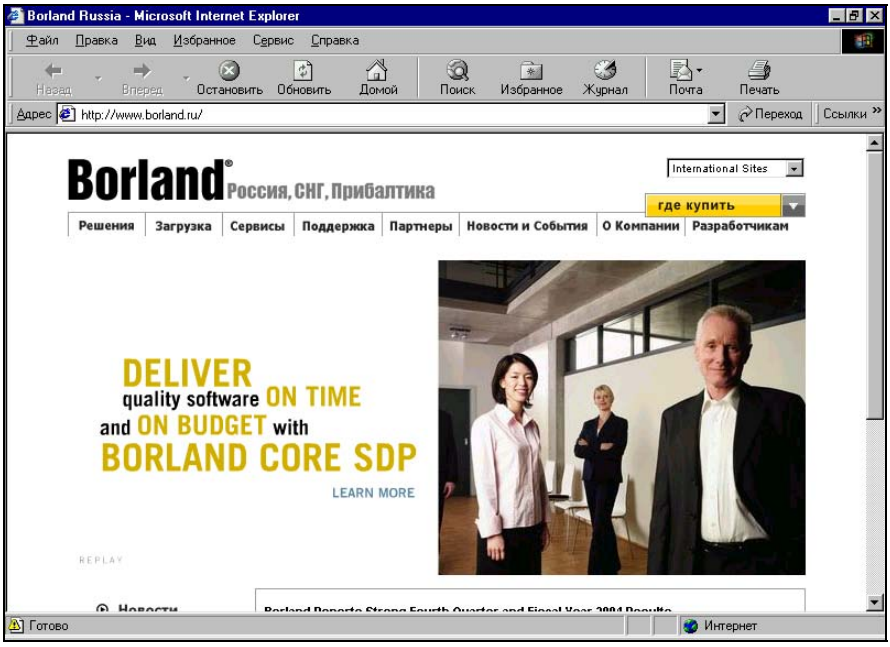


Рис. 18.4. Сайт представительства компании Borland в России и странах СНГ

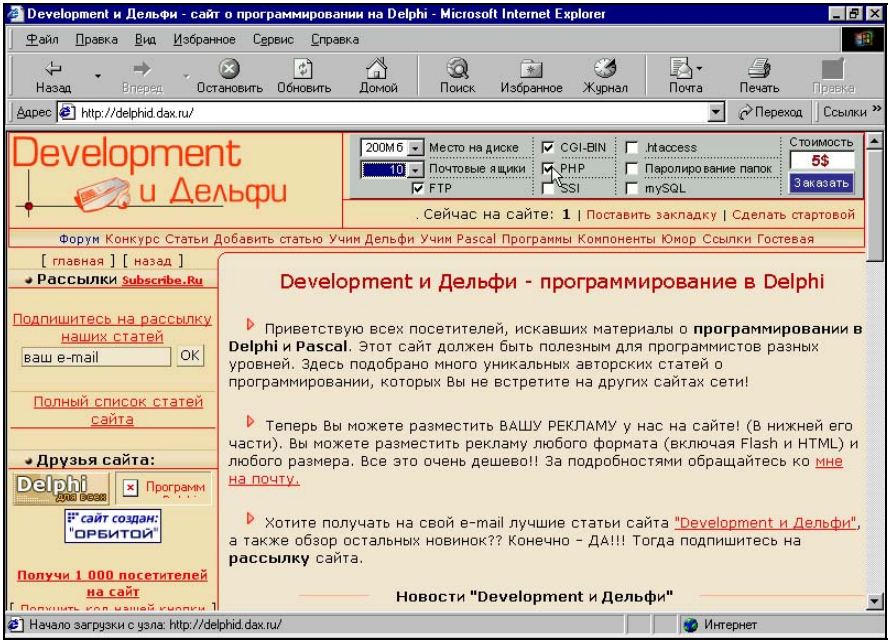


Рис. 18.5. Главная страница сайта "Development и Дельфи"

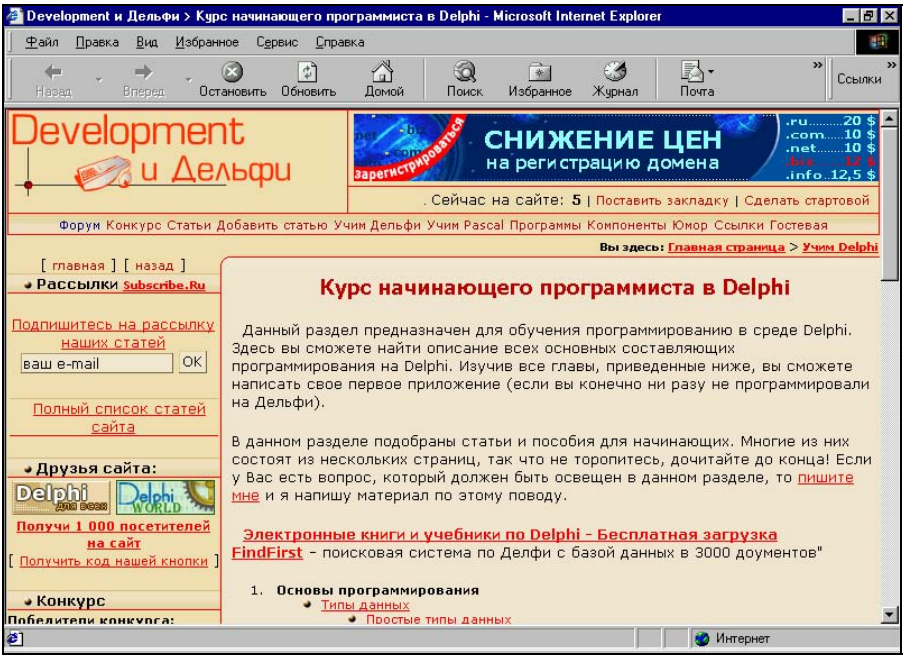


Рис. 18.6. Страница сайта "Development и Дельфи", содержащая электронные книги и пособия по Delphi

ры программ, FAQ (ответы на часто задаваемые пользователями вопросы) и другую информацию (рис. 18.7).

На сайте имеется страница, содержащая учебники и справочные пособия по Delphi, которые можно скачать (рис. 18.8).

Сайт "Королевство Delphi — виртуальный клуб программистов" расположен в сети по адресу **http://www.delphikingdom.ru**. Здесь находится ряд статей и обзоров книг по программированию в Delphi, сборник решений задач, часто задаваемые вопросы, информация о ресурсах сети по программированию и многое другое (рис. 18.9).

Сайт "Delphi по-русски — Delphi for Russian" находится в сети по адресу **http://atrussk.ru/delphi**. Данный сайт содержит on-line (доступный при подключении к Интернету) справочник по использованию стандартных процедур и функций в Delphi (рис. 18.10).

Сайт "Мастера DELPHI" расположен в сети по адресу **http://delphi.mastak.ru**. Здесь можно найти статьи и книги по программированию, исходные тексты программ, сборник часто задаваемых вопросов, ссылки на другие сайты (рис. 18.11). Для нахождения нужной информации на сайте имеется собственная поисковая система.

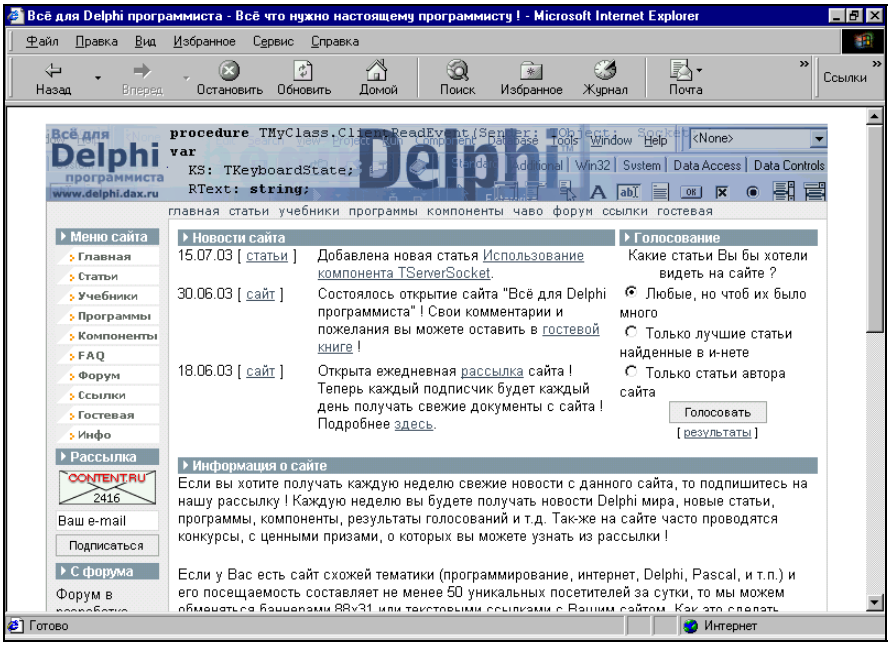


Рис. 18.7. Главная страница сайта "Все для Delphi программиста"

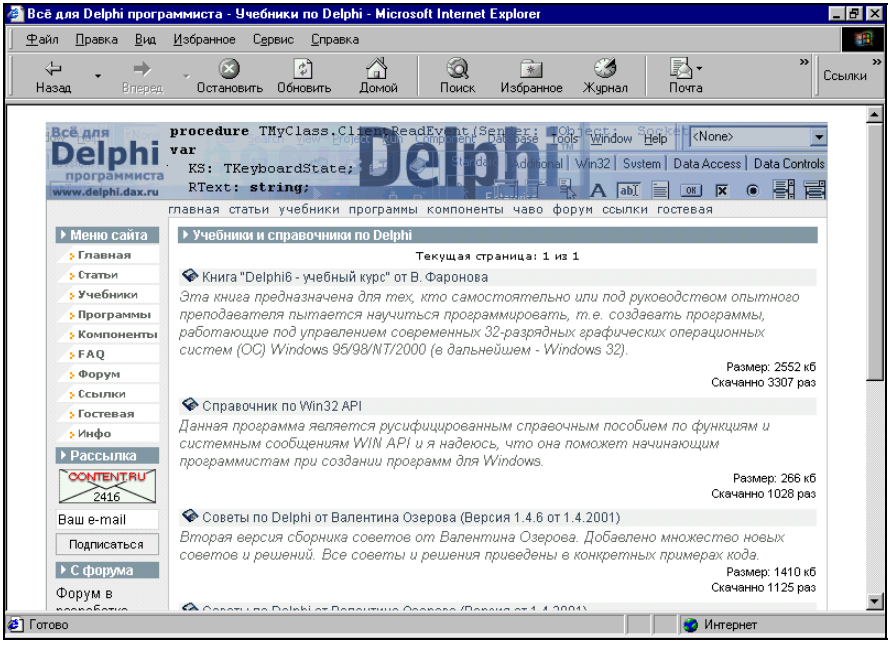


Рис. 18.8. Страница сайта "Все для Delphi программиста", содержащая учебники и справочники по Delphi

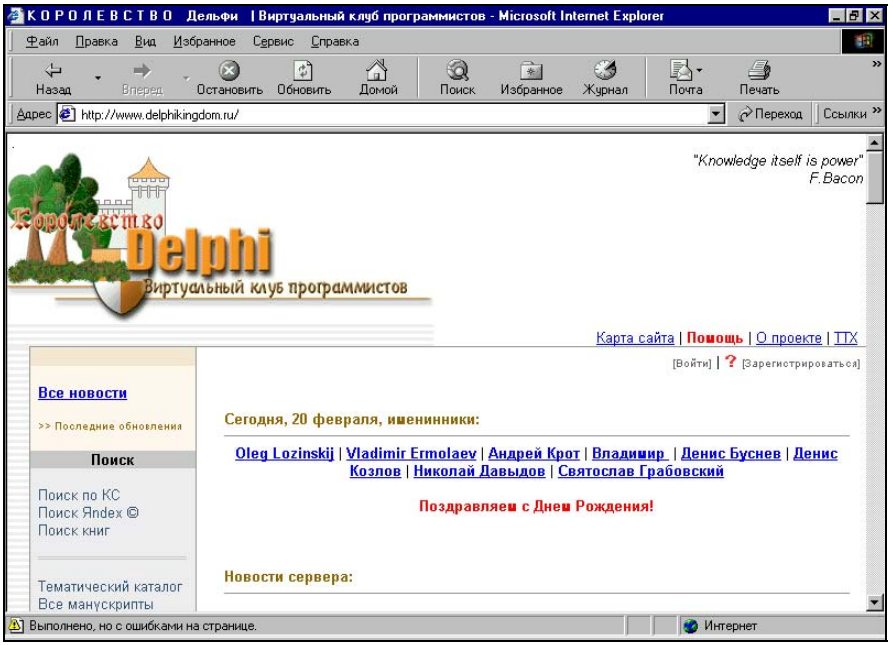


Рис. 18.9. Главная страница сайта "Королевство Delphi — виртуальный клуб программистов"

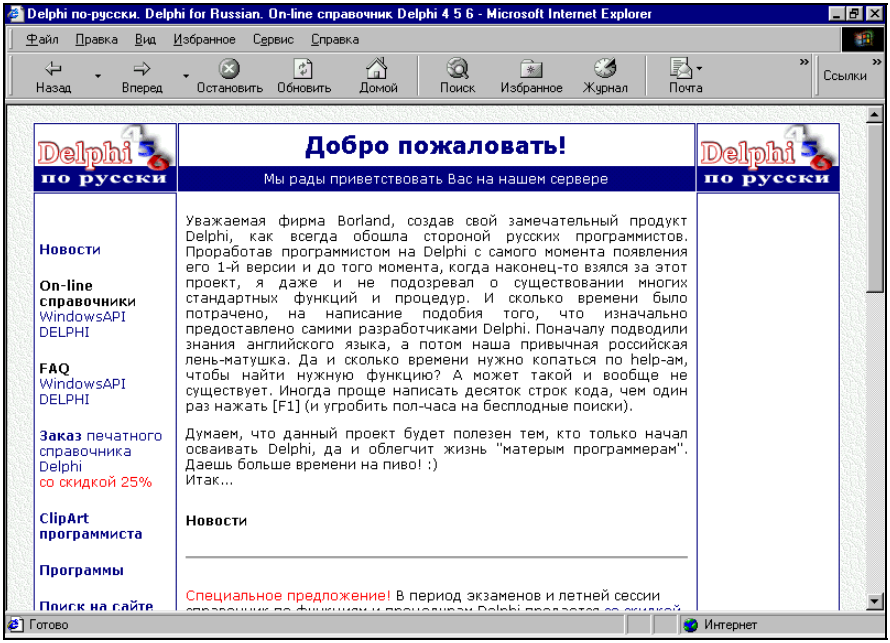


Рис. 18.10. Главная страница сайта "Delphi по-русски — Delphi for Russian"

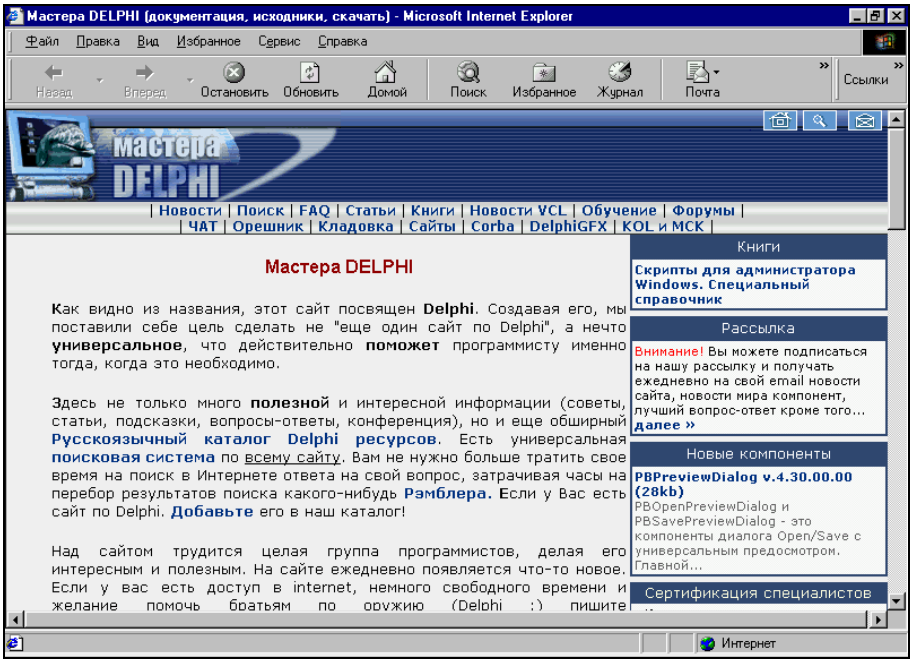


Рис. 18.11. Главная страница сайта "Мастера DELPHI"

Сайт DelphiN находится в Интернете по адресу **http://delphinsworld.narod.ru**. Здесь можно найти книги, статьи по программированию, примеры исходных кодов и готовые программы на Delphi, есть поисковая система и многое другое (рис. 18.12). Особый интерес представляет онлайновая система перевода, которая может быть очень полезна при чтении справочной системы Delphi, написанной по-английски.

Сайт "Уголок Delphi-программиста" находится по адресу **http://www.delphiugolok.narod.ru**. Целью автора данного сайта было собрать на нем в концентрированном виде наиболее интересную и полезную информацию по программированию в Delphi из уже имеющейся в сети (рис. 18.13). Информация, имеющаяся на сайте, может представлять интерес для программистов самого разного уровня: от начинающих до опытных.

Сайт "Delphi32 — все о Дельфи" расположен в сети по адресу **http://delphi-32.narod.ru**. Этот сайт посвящен программированию в различных версиях системы Delphi, начиная с первой и заканчивая шестой (рис. 18.14). Здесь в разделе **Статьи** наряду со статьями имеется хорошая подборка учебных и справочных пособий по Delphi.

Сайт "Borland X Portal" находится в Интернете по адресу **http://borland.xportal.ru**. На данном сайте содержится много информации по рабо-

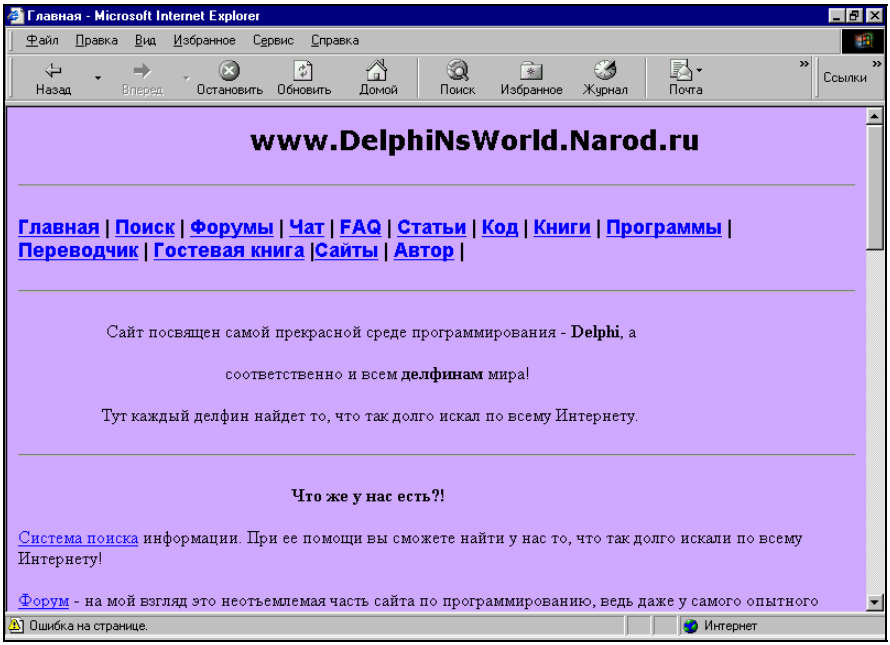


Рис. 18.12. Главная страница сайта "DelphiN"

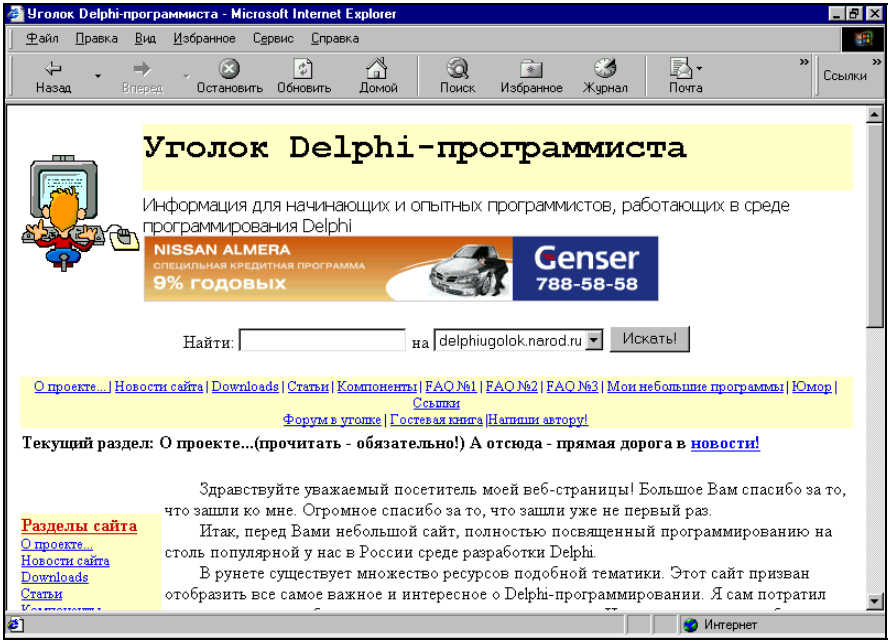


Рис. 18.13. Главная страница сайта "Уголок Delphi-программиста"

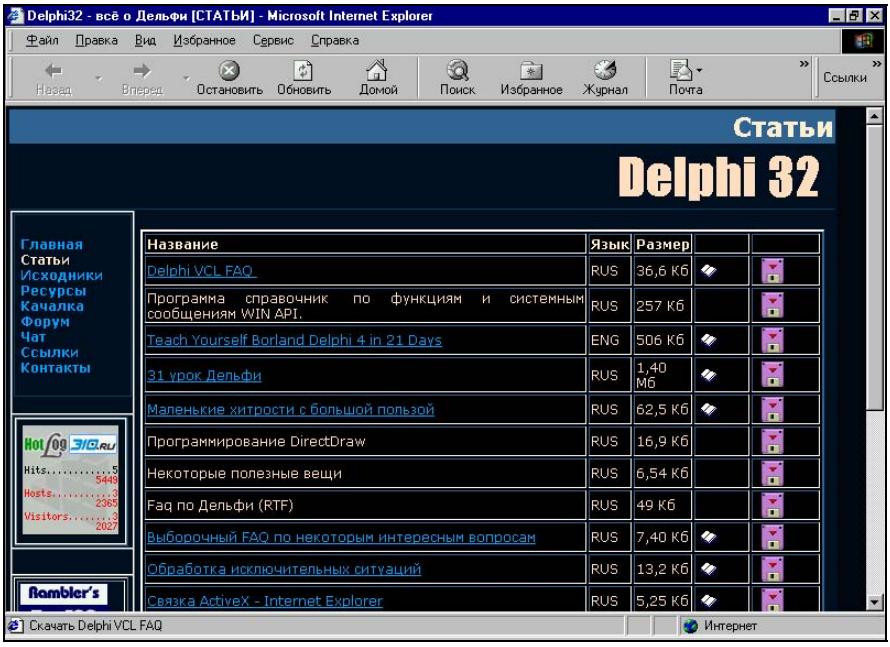


Рис. 18.14. Страница сайта "Delphi32 — все о Дельфи", посвященная статьям, учебным и справочным пособиям по Delphi

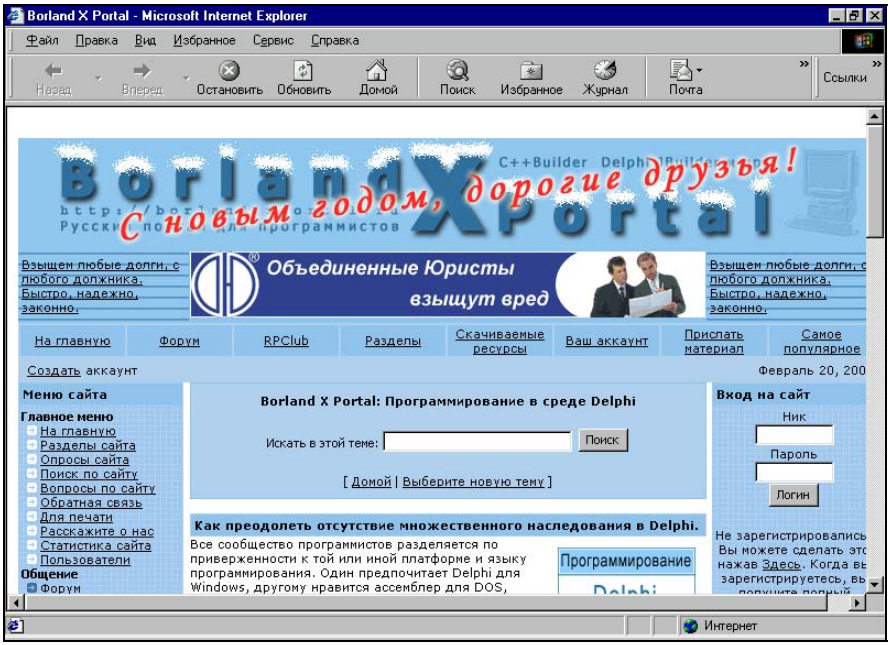
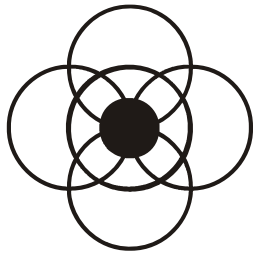


Рис. 18.15. Страница сайта "Borland X Portal", посвященная программированию в среде Delphi

те с различными программными продуктами фирмы Borland, включая, естественно, и Delphi (рис. 18.15). Особо хотелось бы отметить подборку рецензий на различную литературу по программированию.

Естественно, что перечисленным ранее не исчерпывается все богатство материалов по системе программирования Delphi, имеющихся во всемирной информационной сети. Но надеемся, что данный перечень может стать основой и путеводителем для всех желающих совершенствоваться путем самообразования в такой интересной и полезной профессии, как программирование.



ГЛАВА 19

Ответы и решения к заданиям

В данной главе приводятся ответы и решения ко всем заданиям, приведенным в книге в главах с 3 по 10 и с 13 по 16.

Глава 3

Задание 1

```
program z3_1;
Uses Crt;
begin
 ClrScr;
 Textcolor(Magenta);
 TextBackground(LightGray);
 writeln('Существуют следующие основные виды программного обеспечения');
 writeln('персональных компьютеров:');
 writeln('1) Системные программы');
 writeln('2) Прикладные программы');
 writeln('3) Инструментальные системы или системы программирования');
 readln
end.
```

Задание 2

```
program z3_2;
Uses Crt;
begin
 ClrScr;
```

```
 Textcolor(green);
 writeln('       *');
 writeln('      ***');
 writeln('     *****');
 writeln('    *******');
 writeln('   *********');
 writeln('  ***********');
 writeln(' *************');
 writeln('***************');
 readln
end.
```

Задание 3

```
program z3_3;
Uses Crt;
 var
     c1,m1,c2,m2,c3,m3,r,t1,t2:integer;
begin
 ClrScr;
 writeln('Введите начальное время');
 writeln('Введите часы, затем нажмите клавишу Enter');
 readln(c1);
 writeln('Введите минуты, затем нажмите клавишу Enter');
 readln(m1);
 writeln('Введите конечное время');
 writeln('Введите часы, затем нажмите клавишу Enter');
 readln(c2);
 writeln('Введите минуты, затем нажмите клавишу Enter');
 readln(m2);
 t1:=c1*60+m1; t2:=c2*60+m2;
 r:=t2-t1;
 c3:=r div 60; m3:=r mod 60;
 writeln('Между начальным и конечным моментами времени');
 writeln('прошло ',c3,' ч ',m3,' мин');
 readln
end.
```

Задание 4

```
program z3_4;
Uses Crt;
const
     p=3.1416;
```

```
var
     r,l,s:real;
begin
 ClrScr;
 writeln('Введите радиус окружности');
 readln(r);
 l:=2*p*r;
 s:=p*r*r;
 writeln('Длина окружности равна ',l:7:2);
 writeln('Площадь круга равна',s:7:2);
 readln
end.
```

Задание 5

```
program z3_5;
Uses Crt;
  var
      kg1,kg2,kg3:integer;
      r1,r2,r3,c1,c2,c3,st1,st2,st3,stoim:real;
begin
 ClrScr;
 writeln('Введите количество купленного картофеля (в кг)');
 readln(kg1);
 writeln('Введите стоимость 1 кг картофеля в рублях и копейках,');
 writeln('сначала введите рубли и нажмите клавишу Enter,');
 writeln('затем копейки и нажмите Enter');
 readln(r1); readln(c1);
 writeln('Введите количество купленных огурцов (в кг)');
 readln(kg2);
 writeln('Введите стоимость 1 кг огурцов в рублях и копейках,');
 writeln('сначала введите рубли и нажмите клавишу Enter,');
 writeln('затем копейки и нажмите Enter');
 readln(r2); readln(c2);
 writeln('Введите количество купленных помидоров (в кг)');
 readln(kg3);
 writeln('Введите стоимость 1 кг помидоров в рублях и копейках,');
 writeln('сначала введите рубли и нажмите клавишу Enter,');
 writeln('затем копейки и нажмите Enter');
 readln(r3); readln(c3);
 st1:=(r1+c1/100)*kg1;
 st2:=(r2+c2/100)*kg2;
 st3:=(r3+c3/100)*kg3;
 stoim:=st1+st2+st3;
```

```
 writeln('Общая стоимость покупки составляет ',stoim:6:2,' руб');
 readln
end.
```

Глава 4

Задание 1

```
program z4_1;
Uses Crt;
 var
     n:integer;
begin
 ClrScr;
 writeln('Введите целое число');
 readln(n);
 if n mod 5 =0 then writeln('Введенное число делится на 5')
               else writeln('Введенное число не делится на 5');
 readln
end.
```

Задание 2

```
program z4_2;
Uses Crt;
var
    n:integer;
begin
 ClrScr;
 writeln('Введите целое число');
 readln(n);
 if (n>0) and (n mod 2 =0)
     then writeln('число является одновременно положительным и четным')
     else writeln('число не является одновременно положительным и
                  ↳ четным');
 readln
end.
```

Задание 3

```
program z4_3;
Uses Crt;
```

```
 var
    so,sk,n,c:integer;
begin
 ClrScr;
 writeln('Введите номер зоны для станции отправления');
 readln(so);
 writeln('Введите номер зоны для конечной станции');
 readln(sk);
 if sk>so  then n:=sk-so
               else n:=so-sk;
 c:=n*3;
 writeln('Стоимость проезда составляет ',c,' руб');
 readln
end.
```

Задание 4

```
program z4_4;
Uses Crt;
 var
     so,sk,n,c,k:integer;
begin
 ClrScr;
 writeln('Выберите вид приобретаемого билета');
 writeln('Для осуществления выбора нажмите соответствующую цифру:');
 writeln('1 - для покупки билета в один конец');
 writeln('2 - для покупки билета в оба конца');
 writeln('3 - для покупки месячного билета');
 readln(k);
 writeln('Введите номер зоны для станции отправления');
 readln(so);
 writeln('Введите номер зоны для конечной станции');
 readln(sk);
 if sk>so  then n:=sk-so
           else n:=so-sk;
 case k of
    1: c:=n*3;
    2: c:=(n*3)*2;
    3: c:=(n*3)*15;
 end;
 writeln('Стоимость проезда составляет ',c,' руб');
 readln
end.
```

Глава 5

Задание 1

```
program z5_1;
Uses Crt;
  const
      n=25;
  var
      i,k,r:integer;
begin
 ClrScr;
 k:=0; {k - счетчик количества студентов, удовлетворяющих заданному
        условию}
 writeln('Программа подсчета количества кандидатов в баскетбольную
         ↳ команду');
 for i:=1 to n do
     begin
     writeln('Введите рост ',i,' студента');
     readln(r);
     if r>=175 then k:=k+1
     end;
 writeln('Количество кандидатов в баскетбольную команду');
 writeln('в данной группе равно ',k);
 readln
end.
```

Задание 2

```
program z5_2;
Uses Crt;
const
     n=5;
 var
     i:integer; d,s,p:real;
begin
 ClrScr;
 writeln('Введите исходную сумму вклада');
 readln(s);
 writeln('Введите ежегодный процент по вкладу');
 readln(p);
```

```
 for i:=1 to n do
    begin
    d:=s*p/100;{подсчет дохода по вкладу за год}
    s:=s+d {подсчет общей суммы вклада по истечении очередного года}
    end;
 writeln('Общая сумма вклада по истечении срока хранения');
 writeln('составит ',s:8:2,' руб');
 readln
end.
```

Задание 3

```
program z5_3;
Uses Crt;
const
    s=1000000;
 var
    i,k:longint;
begin
 ClrScr;
 i:=1; {i - номер клетки на доске}
 k:=1; {k - количество зерен на одной клетке доски}
 repeat
     k:=k*2;
     i:=i+1
 until k>s;
 writeln('Количество зерен превысит один миллион на ',i,' клетке');
 writeln('шахматной доски');
 readln
end.
```

Задание 4

```
program z5_4;
Uses Crt;
 var
     i,k,s:longint;
begin
 ClrScr;
 i:=1; {i - номер клетки на доске}
 k:=1; {k - количество зерен на одной клетке доски}
 s:=1; {общее количество зерен, находящееся на i клетках}
```

```
 while s<=1000000000 do
      begin
      k:=k*2;
      i:=i+1;
      s:=s+k
      end;
 writeln('Количество клеток, на которых в общей сложности разместится');
 writeln('более одного миллиарда зерен, составлет ',i+1);
 readln
end.
```

Глава 6

Задание 1

```
program z6_1;
Uses Crt;
 var
     i,l:integer; flag:boolean; c:char; s:string;
begin
 ClrScr;
 flag:=true;
 writeln('Введите число');
 readln(s);
 l:=length(s);
 for i:=1 to l do
    begin
    c:=s[i];
    if (c<>'0') and (c<>'1') then flag:=false
    end;
 if flag=true
    then writeln('введенное число является двоичным')
    else writeln('введенное число не является двоичным');
 readln
end.
```

Задание 2

```
program z6_2;
Uses Crt;
 var
     i,k,ks,l:integer; c:char; s:string;
```

```
begin
 ClrScr;
 writeln('Введите строку. Ввод завершите нажатием клавиши Enter');
 writeln;
 readln(s);
 l:=length(s);
 k:=0;{k - счетчик количества пробелов}
 for i:= 1 to l do
     begin
     c:=s[i];
     if c=' ' then k:=k+1
     end;
 ks:=k+1;
 writeln;
 writeln('количество слов в данной строке составляет ',ks);
 readln
end.
```

Задание 3

```
program z6_3;
Uses Crt;
 var
     i,k,ks,l,n:integer; dl:real; c:char; s:string;
begin
 ClrScr;
 writeln ('Введите строку. Ввод завершите нажатием клавиши Enter');
 writeln;
 readln(s);
 l:=length(s);
 k:=0;{k - счетчик количества пробелов}
 for i:= 1 to l do
     begin
     c:=s[i];
     if c=' ' then k:=k+1
     end;
 ks:=k+1;
 n:=l-k; {n - количество символов в строке, не являющихся пробелами}
 dl:=n/ks;
 writeln;
 writeln('Средняя длина слова в строке составляет ',dl:6:2,' символа');
 readln
end.
```

Задание 4

```
program z6_4;
Uses Crt;
 var
     i,l,o:integer; s:string; c,x:char;
begin
 ClrScr;
 writeln('Введите текст исходного сообщения');
 readln(s);
 l:=length(s);
 writeln;
 writeln('Зашифрованный текст выглядит следующим образом:');
 writeln;
 for i:=1 to l do
    begin
    c:=s[i];
    o:=ord(c)-10;
    x:=chr(o);
    write(x)
    end;
 writeln;
 readln
end.
```

Глава 7

Задание 1

```
program z7_1;
Uses Crt;
 var
     i,k,n:integer; m:array[1..20] of integer;
begin
 ClrScr;
 writeln('Введите число элементов массива');
 readln(n);
 k:=0;
 writeln;
 for i:= 1 to n do
     begin
     writeln('Введите ',i,'элемент массива');
```

```
     readln(m[i]);
     if m[i]>0 then k:=k+1
     end;
 writeln;
 writeln('Число положительных элементов в данном массиве равно ',k);
 readln
end.
```

Задание 2

```
program z7_2;
Uses Crt;
 var
     a,b,i,k,n:integer; m:array[1..25] of integer;
begin
 ClrScr;
 Randomize;
 k:=0;
 writeln('Введите количество элементов массива');
 readln(n);
 writeln('Введите a');
 readln(a);
 writeln('Введите b');
 readln(b);
 for i:=1 to n do
     begin
     m[i]:=random(50)+1;
     write(m[i],' ');
     if (m[i]>=a) and (m[i]<=b) then k:=k+1
     end;
 writeln;
 writeln('Количество элементов массива, значения которых находятся');
 writeln('в диапазоне от ',a,' до ',b,' равно ',k);
 readln
end.
```

Задание 3

```
program z7_3;
Uses Crt;
 var
      i,k,n:integer; a:array[1..20] of integer;
```

```
begin
 ClrScr;
 k:=0;
 writeln('Введите число элементов массива');
 readln(n);
 writeln;
 for i:=1 to n do
     begin
     writeln('Введите ',i,' элемент массива');
     readln(a[i]);
     if a[i]<0 then k:=i
     end;
 for i:=1 to n do
     write(a[i],' ');
 writeln;
 writeln('Номер последнего по счету отрицательного элемента');
 writeln('равен ',k);
 readln
end.
```

Задание 4

```
program z7_4;
Uses Crt;
  var
      a,k:char; i,g:integer; mas:array[0..255] of integer;
begin
 ClrScr;
 writeln('Введите текстовую строку');
 writeln('Ввод всего текста заканчивайте нажатием клавиши "0"');
 writeln;
 repeat
    read(a);
    i:=ord(a);
    mas[i]:=mas[i]+1;
 until a='0'; {признак окончания ввода текста}
 readln;
 g:=0;
 writeln;
 writeln('В данном тексте встречаются следующие символы ');
 writeln('следующее количество раз:');
 for i:=0 to 255 do
    begin
```

```
    if (mas[i]<>0) and (i<>48) then
    {48 - код нуля в таблице ASCII}
         begin
         g:=g+1;
         k:=chr(i);
         if i<>32 {32 - код пробела в таблице ASCII}
                   then write('"',k,'"  - ',mas[i],' | ')
                   else write('"пробел" - ',mas[i],' | ')
         end;
    if g mod 5 = 0 then
         begin
         g:=1;
         writeln
         end;
    end;
 readln
end.
```

Глава 8

Задание 1

```
program z8_1;
Uses Crt;
 var
      a,b,c:integer; s,p:real;
begin
 ClrScr;
 writeln('Введите длину первой из сторон треугольника');
 readln(a);
 writeln;
 writeln('Введите длину второй стороны треугольника');
 readln(b);
 writeln;
 writeln('Введите длину третьей стороны треугольника');
 readln(c);
 writeln;
 p:=(a+b+c)/2;
 s:=sqrt(p*(p-a)*(p-b)*(p-c));
 writeln('Площадь треугольника равна ',s:8:2);
 readln
end.
```

Задание 2

```
program z8_2;
Uses Crt;
var
    n,x,r1,r2:integer;  r3:real;
begin
 ClrScr;
 writeln('Введите целое число');
 readln(n);
 writeln;
 writeln('Укажите операцию, которая  будет производиться');
 writeln('над введенным числом');
 writeln;
 writeln('Для выбора введите соответствующую цифру:');
 writeln;
 writeln('1 - определение абсолютной величины числа');
 writeln('2 - возведение числа в квадрат');
 writeln('3 - извлечение квадратного корня');
 readln(x);
 writeln;
 case x of
     1: begin
        r1:=abs(n);
        writeln('Абсолютная величина числа равна ',r1)
        end;
     2: begin
        r2:=sqr(n);
        writeln('Квадрат числа равен ',r2)
        end;
     3: begin
        if n>=0 then
             begin
             r3:=sqrt(n);
             writeln('Квадратный корень числа равен ',r3:8:3)
        end
      else
         begin
         writeln('Число отрицательное');
         writeln('квадратный корень из него не извлекается');
         end;
        end;
 end;
```

```
 readln
end.
```

Задание 3

```
program z8_3;
Uses Crt;
 label 10;
 var
       flag,n,x,r1,r2:integer; r3:real; t:string;
function c(s:string):boolean;
{Данная функция определяет, является ли исходная строка символов
 целым числом.
 Для этого определяется длина данной строки и затем для каждого символа,
 входящего в строку, проверяется, относится ли он к последовательности
 символов от 0 до 9}
 var
       i,k:integer;
begin
  c:=true;
  k:=length(s);
  for i:=1 to k do
       if (s[i]<>'0') and (s[i]<>'1') and (s[i]<>'2') and (s[i]<>'3')
       and (s[i]<>'4') and (s[i]<>'5') and (s[i]<>'6') and (s[i]<>'7')
       and (s[i]<>'8') and (s[i]<>'9')
       then c:=false
end;
begin
  ClrScr;
  10: writeln('Введите целое число');
  readln(t); {число вводится как строковая величина}
  writeln;
  if c(t)=true then val(t,n,flag)
{В том случае, если введенная величина действительно является
 целым числом,то она преобразуется посредством процедуры
 из строкового типа в целочисленный}
               else
                  begin
                  writeln('Введенная величина не является целым числом');
                  goto 10
                  end;
   writeln('Укажите операцию, которая будет производиться');
   writeln('над введенным числом');
   writeln;
```

```
  writeln('Для выбора введите соответствующую цифру:');
  writeln;
  writeln('1 - определение абсолютной величины числа');
  writeln('2 - возведение числа в квадрат');
  writeln('3 - извлечение квадратного корня');
  readln(x);
  writeln;
  case x of
     1: begin
         r1:=abs(n);
         writeln('Абсолютная величина числа равна ',r1)
         end;
     2: begin
         r2:=sqr(n);
         writeln('Квадрат числа равен ',r2)
         end;
     3: begin
         if n>=0 then
             begin
             r3:=sqrt(n);
             writeln('Квадратный корень числа равен ',r3:8:3)
             end
                      else
             begin
             writeln('Число отрицательное');
             writeln('квадратный корень из него не извлекается');
             end;
         end;
  end;
  readln
end.
```

Задание 4

```
program z8_4;
Uses Crt;
  var
       i,k,x:integer; mas:array[1..20] of integer; flag:boolean;
procedure prost(x:integer; var f:boolean);
  var
       j:integer;
           begin
  f:=true;
```

```
  if (x=1) or (x=2) then exit;
  {числа 1 и 2 заведомо являются простыми}
  for j:=2 to x-1 do        {в цикле перебираются все возможные}
        if x mod j =0 then        {делители числа кроме самого числа}
           begin f:=false; exit end  {и единицы}
end;
begin
  ClrScr;
  Randomize;
  k:=0; {k - счетчик количества простых чисел}
  writeln('Массив состоит из следующих элементов:');
  for i:=1 to 20 do
     begin
     mas[i]:=random(99)+1;
     write(mas[i],' ');
     x:=mas[i];
     prost(x,flag); {процедура определения того,}
                    {является ли число простым}
     if flag=true then k:=k+1
     end;
  writeln;
  writeln;
  writeln('Количество элементов массива, являющихся простыми числами');
  writeln('равно ',k);
  readln
end.
```

Задание 5

```
program z8_5;
Uses Crt;
  var
      i:integer; flag:boolean;
procedure sov(k:integer; var f:boolean);
  var
      s,j:integer;
           begin
  f:=false;
  s:=0;{s - сумма всех делителей числа}
  for j:=1 to k-1 do
          if k mod j =0 then s:=s+j;
          {в случае нахождения очередного делителя числа прибавляем
          полученную величину к их общей сумме}
```

```
  if k=s then f:=true
end;
begin
  ClrScr;
  writeln('Список совершенных чисел в интервале от 1 до 10000');
  writeln;
  for i:=1 to 10000 do
        begin
        sov(i,flag);
        if flag=true then writeln(i)
        end;
  readln
end.
```

Глава 9

Задание 1

```
program z9_1;
Uses Crt;
  var
       k,l:integer; t:text; ln:string[80];
begin
  Clrscr;
  k:=0;
  assign(t,'A:\tekst.txt');
  rewrite(t);
  writeln('Вводите текст. Ввод каждой строки заканчивайте нажатием
          ↳ Enter');
  writeln;
  writeln('Для завершения ввода текста в файл в конце последней строки');
  writeln('данного текста введите символ *');
  writeln;
  repeat
     readln(ln);
     writeln(t,ln);
     k:=k+1;
     l:=length(ln)
  until ln[l]='*';
  writeln;
  writeln('Ввод текста в файл на дискете завершен');
  writeln('общее количество строк во введенном тексте равно ',k);
```

```
  close(t);
  readln
end.
```

Задание 2

```
program z9_2;
Uses Crt;
 var
      k,l:integer; t:text; ln,kol:string[80];
begin
  Clrscr;
  k:=0;
  assign(t,'A:\tekst.txt');
  rewrite(t);
  writeln('Вводите текст. Ввод каждой строки заканчивайте нажатием
          ↳ Enter');
  writeln;
  writeln('Для завершения ввода текста в файл в конце последней строки');
  writeln('данного текста введите символ *');
  writeln;
  repeat
     readln(ln);
     writeln(t,ln);
     k:=k+1;
     l:=length(ln)
  until ln[l]='*';
  str(k+1,kol);
  kol:='Количество строк в данном тексте равно '+kol;
  writeln(t,kol);
  writeln;
  writeln('Ввод текста в файл на дискете завершен');
  writeln('общее количество строк во введенном тексте равно ',k+1);
  close(t);
  readln
end.
```

Задание 3

```
program z9_3;
Uses Crt;
  var
       i,n,s,x:integer; f:file of integer;
```

```
begin
  ClrScr;
  assign(f,'A:\number');
  rewrite(f);
  s:=0;
  writeln('Укажите количество чисел, вводимых в файл');
  readln(n);
  writeln;
  for i:=1 to n do
     begin
     writeln('Введите ',i,' число');
     readln(x);
     s:=s+x;
     write(f,x)
     end;
  write(f,s);
  writeln;
  writeln('Ввод данных завершен');
  writeln;
  writeln('В файле содержатся следующие числа');
  reset(f);
  for i:=0 to n do
       begin
       read(f,x);
       write(x,' ')
      end;
  close(f);
  readln
end.
```

Задание 4

```
program z9_4;
Uses Crt;
  var
       i,n,x:integer; g:string; u:file;
begin
  ClrScr;
  assign(u,'A:\contents');
  rewrite(u);
  writeln('Укажите количество глав в книге');
  readln(n);
  writeln;
```

```
  for i:=1 to n do
        begin
        writeln('Введите номер главы');
        readln(x);
        blockwrite(u,x,1);
        writeln('Введите название главы');
        readln(g);
        blockwrite(u,g,1);
        end;
  close(u);
  writeln('Ввод данных завершен');
  writeln;
  reset(u);
  writeln('В файле "contents" содержится следующая информация о книге');
  for i:=1 to n do
     begin
     blockread(u,x,1);
     write('глава ',x,' ');
     blockread(u,g,1);
     write(g);
     writeln
     end;
  close(u);
  readln
end.
```

Глава 10

Задание 1

```
program z10_1;
Uses Crt;
  type month=(error,jan,feb,mar,apr,may,jun,jul,aug,sep,oct,nov,dec);
     var i:integer; name:string; mes:month;
begin
  ClrScr;
  writeln('Введите название месяца');
  readln(name);
  if name='январь' then mes:=jan;
  if name='февраль' then mes:=feb;
  if name='март' then mes:=mar;
  if name='апрель' then mes:=apr;
```

```
  if name='май' then mes:=may;
  if name='июнь' then mes:=jun;
  if name='июль' then mes:=jul;
  if name='август' then mes:=aug;
  if name='сентябрь' then mes:=sep;
  if name='октябрь' then mes:=oct;
  if name='ноябрь' then mes:=nov;
  if name='декабрь' then mes:=dec;
  case mes of
      jan,feb,dec: writeln('На дворе зима');
      mar,apr,may: writeln('Пришла весна');
      jun,jul,aug: writeln('Наступило  лето');
      sep,oct,nov: writeln('За окном осень');
      else writeln('Такого месяца не существует');
  end;
  readln
end.
```

Задание 2

```
program z10_2;
Uses Crt;
  label 10;
  type
        god=1600..2100;
  var
        n:god;
begin
  ClrScr;
  writeln;
  10: writeln('Введите номер года (с 1600 по 2100)');
  readln(n);
  if (n<1600) or (n>2100) then goto 10;
  writeln;
  if n mod 100=0
    then
       if n mod 400=0 then writeln('год високосный')
                      else writeln('год невисокосный')
    else
       if n mod 4 =0 then writeln('год високосный')
                     else writeln('год невисокосный');
  readln
end.
```

Задание 3

```
program z10_3;
Uses Crt;
label 10;
  type uchenik=record
      fam,name:string;
      mat,fiz,lit,ist,inos:integer
  end;
  var
       i,k:integer; s,otvet:string;
       zap:uchenik; ved:file of uchenik;
begin
  ClrScr;
  assign(ved,'A:\vedomost');
  rewrite(ved);
  k:=0;
  repeat
      writeln('Введите фамилию ученика');
      readln(zap.fam);
      writeln('Введите имя ученика');
      readln(zap.name);
      writeln('Введите оценку по математике');
      readln(zap.mat);
      writeln('Введите оценку по физике');
      readln(zap.fiz);
      writeln('Введите оценку по литературе');
      readln(zap.lit);
      writeln('Введите оценку по истории');
      readln(zap.ist);
      write(ved,zap);
      k:=k+1;
     ClrScr;
     writeln('Будете вводить данные по следующему ученику');
     10: writeln('Введите "Да" или "Нет"');
     readln(otvet);
     if (otvet<>'Да') and (otvet<>'Нет') then goto 10
  until otvet='Нет';
  ClrScr;
  writeln('количество записей в файле равно ',k);
  writeln;
  writeln('Экзаменационная ведомость');
  writeln;
```

```
  write('    Фамилия|        Имя| Математика|     Физика|    История|');
  writeln('Литература |');
  write('           |           |           |           |           |');
  writeln('           |');
  reset(ved);
  for i:=0 to k-1 do
     begin
     read(ved,zap);
     write(zap.fam:11,'|',zap.name:11,'|',zap.mat:11,'|',zap.fiz:11,'|');
     writeln(zap.lit:11,'|',zap.ist:11,'|');
  end;
  close(ved);
  readln
end.
```

Задание 4

```
program z10_4;
Uses Crt;
  var
      k:integer; c:char;
      eng: set of 'A'..'Z'; rus: set of 'A'..'Я'; cifra: set of '0'..'9';
      leng: set of 'a'..'z';
      lrus1: set of 'a'..'п'; lrus2: set of 'р'..'я';
    {В программе используются два множества строчных букв
     русского алфавита, т. к. в кодовой таблице ASCII эти буквы разбиты
     на две группы, находящиеся в разных частях таблицы и, следовательно,
     они не могут образовать единого непрерывного множества}
begin
  ClrScr;
  cifra:=['0'..'9'];
  eng:=['A'..'Z'];
  rus:=['A'..'Я'];
  leng:=['a'..'z'];
  lrus1:=['a'..'п'];
  lrus2:=['р'..'я'];
  writeln('Введите символ');
  readln(c);
  writeln;
  if c in cifra then k:=1;
  if c in eng then k:=2;
  if c in rus then k:=3;
```

```
  if c in leng then k:=4;
  if (c in lrus1) or (c in lrus2) then k:=5;
  case k of
     1: writeln('данный символ является цифрой');
     2: writeln('данный символ является прописной буквой английского
                ↳ алфавита');
     3: writeln('данный символ является прописной буквой русского
                ↳ алфавита');
     4: writeln('данный символ является строчной буквой английского
                ↳ алфавита');
     5: writeln('данный символ является строчной буквой русского
                ↳ алфавита');
     else writeln('данный символ не является ни цифрой, ни прописной
                ↳ буквой')
  end;
  readln
end.
```

Глава 13

На рис. 19.1 представлен возможный вариант графического интерфейса программы "Светофор".

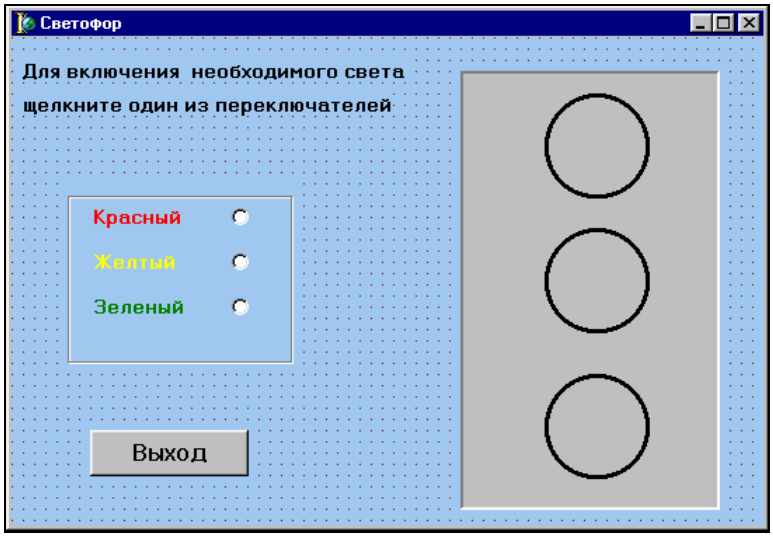


Рис. 19.1. Интерфейс программы "Светофор"

Программный код проекта "Светофор"

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls,
  Forms, Dialogs, ExtCtrls, StdCtrls;

type
  TForm1 = class(TForm)
    RadioGroup1: TRadioGroup;
    Panel1: TPanel;
    Label1: TLabel;
    Label2: TLabel;
    RadioButton1: TRadioButton;
    RadioButton2: TRadioButton;
    RadioButton3: TRadioButton;
    Button1: TButton;
    Shape1: TShape;
    Shape2: TShape;
    Shape3: TShape;
    procedure RadioButton1Click(Sender: TObject);
    procedure RadioButton2Click(Sender: TObject);
    procedure RadioButton3Click(Sender: TObject);
    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.RadioButton1Click(Sender: TObject);
begin
Shape1.Brush.Color:=clRed;
Shape2.Brush.Color:=clSilver;
Shape3.Brush.Color:=clSilver;
end;
```

```
procedure TForm1.RadioButton2Click(Sender: TObject);
begin
Shape2.Brush.Color:=clYellow;
Shape1.Brush.Color:=clSilver;
Shape3.Brush.Color:=clSilver;
end;

procedure TForm1.RadioButton3Click(Sender: TObject);
begin
Shape3.Brush.Color:=clGreen;
Shape1.Brush.Color:=clSilver;
Shape2.Brush.Color:=clSilver;
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
close
end;

end.
```

Глава 14

На рис. 19.2 представлен возможный вариант графического интерфейса программы "Банкомат".

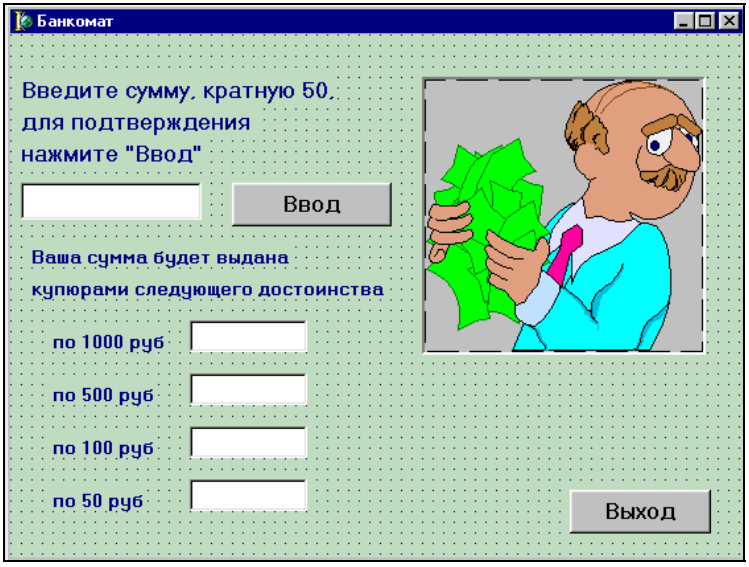


Рис. 19.2. Интерфейс программы "Банкомат"

Программный код проекта "Банкомат"

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls,
  Forms, Dialogs, StdCtrls, ExtCtrls;

type
  TForm1 = class(TForm)
    Label1: TLabel;
    Edit1: TEdit;
    Label2: TLabel;
    Label3: TLabel;
    Panel1: TPanel;
    Image1: TImage;
    Label4: TLabel;
    Label5: TLabel;
    Label6: TLabel;
    Label7: TLabel;
    Edit2: TEdit;
    Edit3: TEdit;
    Edit4: TEdit;
    Edit5: TEdit;
    Button1: TButton;
    Button2: TButton;
    Label8: TLabel;
    Label9: TLabel;
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}
```

```
procedure TForm1.Button1Click(Sender: TObject);
begin
close
end;

procedure TForm1.Button2Click(Sender: TObject);
var sum,k1000,k500,k100,k50,
    ost1000,ost500,ost100:integer;
begin
sum:=StrtoInt(Edit1.Text);
k1000:=sum div 1000;
ost1000:=sum mod 1000;
k500:=ost1000 div 500;
ost500:=ost1000 mod 500;
k100:=ost500 div 100;
ost100:=ost500 mod 100;
k50:=ost100 div 50;
Edit2.Text:=InttoStr(k1000);
Edit3.Text:=InttoStr(k500);
Edit4.Text:=InttoStr(k100);
Edit5.Text:=InttoStr(k50)
end;

end.
```

Глава 15

На рис. 19.3 представлен возможный внешний вид графического интерфейса программы "Календарь".

Программный код проекта "Календарь"

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls,
  Forms, Dialogs, ExtCtrls, StdCtrls;

type
  TForm1 = class(TForm)
    Label1: TLabel;
    Edit1: TEdit;
```

```
    Button1: TButton;
    Label2: TLabel;
    Label3: TLabel;
    Panel1: TPanel;
    Image1: TImage;
    Button2: TButton;
    Button3: TButton;
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure Button3Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.Button1Click(Sender: TObject);
var k:integer;
begin
 Label2.Visible:=True;
 Label2.Caption:='Данный год';
 Label3.Visible:=True;
 k:=StrtoInt(Edit1.Text);
 if k mod 100 <>0
 then
     if k mod 4 =0
     then Label3.Caption:='является високосным'
     else Label3.Caption:='не является вискосным'
 else
     if k mod 400 =0
     then Label3.Caption:='является високосным'
     else Label3.Caption:='не является вискосным'
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
close
end;
```

```
procedure TForm1.Button3Click(Sender: TObject);
begin
Edit1.Text:='';
Label2.Visible:=False;
Label3.Visible:=False
end;

end.
```

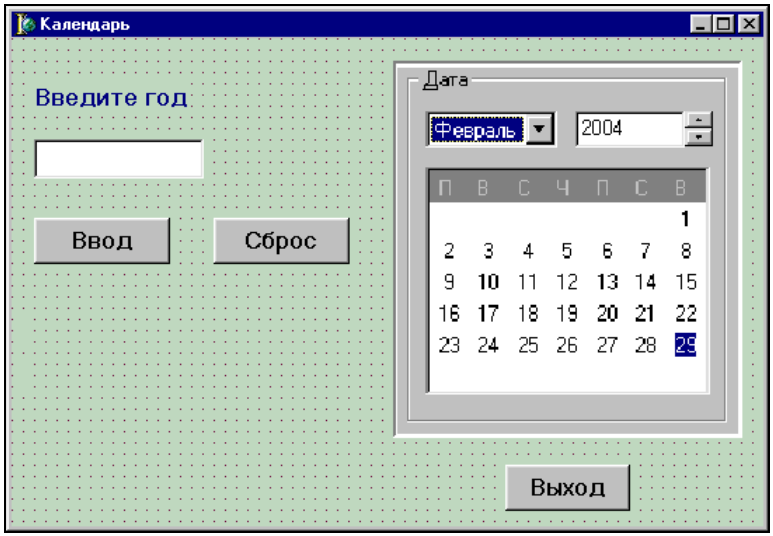


Рис. 19.3. Интерфейс программы "Календарь"

Глава 16

На рис. 19.4 представлен возможный внешний вид графического интерфейса программы "Задача Фибоначчи".

Программный код проекта "Задача Фибоначчи"

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls,
  Forms, Dialogs, StdCtrls, ExtCtrls;
```

```
type
  TForm1 = class(TForm)
    Label1: TLabel;
    Edit1: TEdit;
    Label2: TLabel;
    Label3: TLabel;
    Panel1: TPanel;
    Image1: TImage;
    Button1: TButton;
    Button2: TButton;
    Button3: TButton;
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure Button3Click(Sender: TObject);

  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.Button1Click(Sender: TObject);
var k,n:integer;
function rab(m:integer):integer;
begin
   if m=0
  then rab:=1
  else
   if m=1
   then rab:=2
   else rab:=rab(m-2)+rab(m-1)
end;
begin
n:=StrtoInt(Edit1.Text);
k:=rab(n)*2;
Label2.Visible:=True;
Label2.Width:=200;
```

```
Label2.Caption:='по прошествии указанного времени количество кроликов
                ↳ составит '+InttoStr(k);
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
close
end;

procedure TForm1.Button3Click(Sender: TObject);
begin
Edit1.Text:='';
Label2.Visible:=False;
Label2.Caption:='';
end.
```

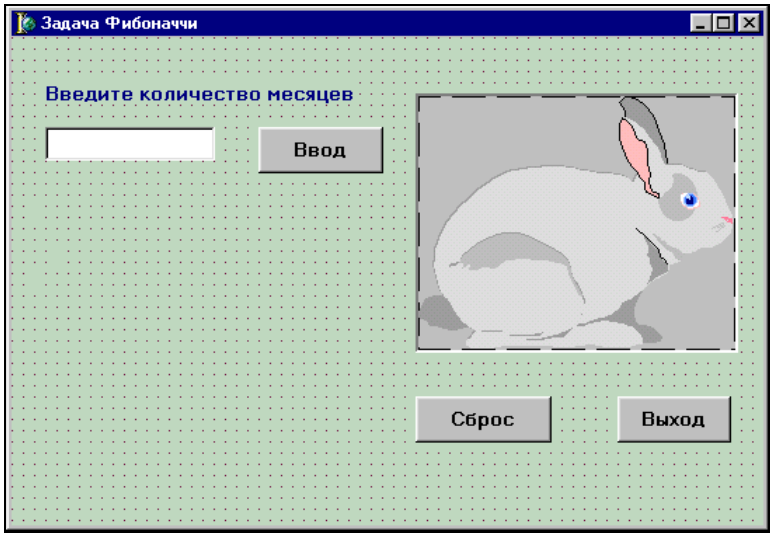


Рис. 19.4. Интерфейс проекта "Задача Фибоначчи"

ПРИЛОЖЕНИЕ 1

Приемы работы в среде Turbo Pascal

На начальном этапе работы в среде Turbo Pascal вполне достаточно тех приемов, которые были даны в книге, однако для эффективной работы этого бывает часто не достаточно. В таком случае следует освоить приведенные в этом приложении новые приемы, в частности, автоматический запуск среды программирования, ускоренный ввод текста, обращение и получение информации из справочной системы, а также использование во время работы нескольких окон.

П1.1. Автоматический запуск системы программирования

В первых главах книги мы говорили о том, как правильно запускать среду программирования Turbo Pascal. Но опытный пользователь может настроить файловую оболочку (например, Norton Commander) таким образом, что при обращении к файлу, содержащему текст программы на языке Паскаль, соответствующая среда программирования будет запускаться автоматически.

Если пользователь работает в файловой оболочке Norton Commander, то оболочку можно настроить таким образом: при двойном щелчке мышью на файле с расширением pas или при выделении этого файла мышью и нажатии клавиши <Enter> данный файл будет сразу открываться в системе Turbo Pascal. Для этого нужно вызвать меню Norton Commander (далее даны команды для русскоязычной версии), нажав функциональную клавишу <F9>. Затем в строке меню нужно с помощью курсорных стрелок выбрать раздел **Команды** и нажать клавишу <Enter>. После нажатия клавиши появится ниспадающее меню, в котором аналогичным образом выбираем пункт **Обработка расширений** и нажимаем клавишу <Enter>.

После нажатия клавиши <Enter> на экране компьютера появляется диалоговое окно, которое называется **Изменение обработки расширения** (рис. П1.1). С помощью этого окна можно изменять список расширений файлов, для обработки которых вызывается соответствующая программа (добавлять в этот список новые пункты, удалять их, а также редактировать). Для добавления нового пункта в список расширений нажимаем функциональную клавишу <F6>. Диалоговое окно увеличивается в размере и в нем появляется ряд полей, которые необходимо правильно заполнить.

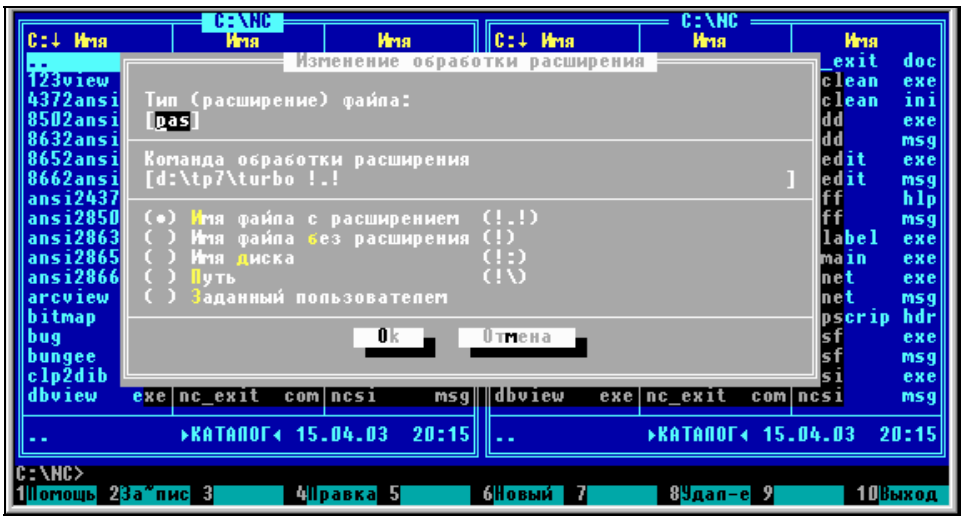


Рис. П1.1. Окно изменения обработки расширения файла

В верхнем поле **Тип (расширение) файла** нужно ввести имя расширения — `pas` (без точки). В среднем поле **Команда обработки расширения** нужно указать путь к приложению, которое должно запускаться вместе с файлом. Путь должен включать в себя имя диска, на котором находится файл, с двоеточием, имя каталога (подкаталогов), содержащего головной файл приложения, и имя самого головного файла. Имена каталогов отделяются от имени диска и от имени файла символом "\" (backslash или обратная косая черта). Например, если головной файл приложения находится на диске D: в каталоге TP7, то в данном поле нужно ввести: `d:\TP7\turbo`.

Далее в нижнем поле переключатель нужно установить в положение **Имя файла с расширением** и подтвердить сделанные изменения щелчком на экранной кнопке **OK**. Закрытие диалогового окна производится нажатием на клавиатуре компьютера клавиши <Esc>. При этом пользователю сообщается о том, что в файл были внесены изменения, и для их подтверждения нужно щелкнуть в окне экранную кнопку **Запись**. После выполнения всех указан-

ных ранее действий любой файл с расширением pas будет открываться в системе Turbo Pascal. После открытия можно редактировать текст файла, запускать его на компиляцию, выполнение и т. д.

П1.2. Перемещение по тексту и редактирование текста

Для ускорения и облегчения ввода текста можно воспользоваться возможностями встроенного редактора системы. Его команды находятся в меню **Edit**. Этот редактор позволяет не только набирать текст программы, но и быстро перемещаться по тексту, а также вносить в него необходимые исправления. Данный раздел меню в раскрытом виде представлен на рис. П1.2.

Для начала успешной работы в редакторе разберем основные команды, позволяющие перемещаться по тексту. Перемещение курсора на одну позицию влево или вправо, либо на одну строку вверх или вниз осуществляется нажатием соответствующих клавиш со стрелками на клавиатуре. Для того чтобы переместиться по тексту не на один символ, а сразу на целое слово влево, нужно использовать сочетание клавиш <Ctrl>+<стрелка влево>, для перемещения на одно слово вправо — <Ctrl>+<стрелка вправо>. Если нужно перейти в начало текущей строки, нажимаем клавишу <Home>, переход в конец текущей строки производится нажатием клавиши <End>. Нажатие клавиши <Page Up> позволяет переместиться по тексту на один экран вверх, клавиши <Page Down> — на экран вниз. Программы на языке Паскаль могут быть достаточно большими по объему и не помещаться на одном экране. В таком случае, для того чтобы попасть в начало программы, нужно нажать комбинацию клавиш <Ctrl>+<Page Up>, а для того, чтобы переместиться в конец программы, нажать комбинацию <Ctrl>+<Page Down>.

Кроме использования перечисленных ранее клавиш и их комбинаций для перемещения по тексту программы в ряде случаев удобно пользоваться мышью. В частности, для того чтобы установить курсор в любое место в пределах текущего экрана, достаточно поместить туда указатель мыши и сделать один щелчок левой кнопкой. Если программа имеет достаточно большой объем, то для просмотра ее содержимого с помощью мыши удобно использовать *скроллеры*. Скроллером или *полосой прокрутки* называются полосы (на экране системы Turbo Pascal они бирюзового цвета), находящиеся справа или снизу от основной рабочей области экрана. На каждом из скроллеров находится небольшой прямоугольник, называемый *лифтом* или бегунком. Ухватившись указателем мыши за лифт и перетаскивая его вдоль скроллера, можно быстро прокручивать текст программы как в вертикальном, так и в горизонтальном направлении.

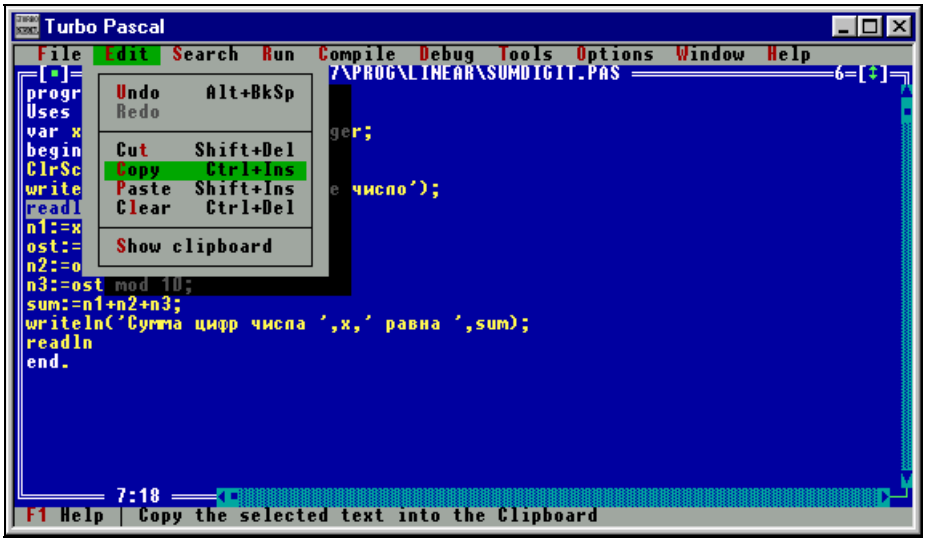


Рис. П1.2. Меню **Edit** среды Turbo Pascal

Следующая группа операций, которую мы рассмотрим, — это операции корректировки. Процесс корректировки сводится к трем основным операциям. Это удаление ошибочно набранных символов и строк, перемещение фрагментов текста из одной части программы в другую или даже из одной программы в другую и копирование фрагментов программы. Перед тем как производить ту или иную операцию, следует предварительно выделить удаляемый, копируемый или перемещаемый фрагмент текста.

Выделение фрагмента во всех этих случаях производится одинаково. Курсор устанавливается на какой-либо символ, относящийся к выделяемому фрагменту, а затем при нажатой клавише <Shift> нажимается нужное количество раз клавиша <стрелка влево> или <стрелка вправо>. В результате соответственно выделенный фрагмент расширяется влево или вправо. При этом выделенный текст будет отображаться не на синем, а на сером фоне. В случае если выделяемый фрагмент содержит не одну строку или часть строки, а несколько строк, следует нажать нужное количество раз клавишу <стрелка вверх> или <стрелка вниз>, в результате чего выделенный фрагмент расширится вверх или вниз. Для выделения фрагмента можно использовать и мышь. Для этого следует при нажатой левой кнопке мыши "протащить" указатель мыши по выделяемому фрагменту, а затем, когда выделение завершено, отпустить эту кнопку.

В том случае, если выделенный фрагмент текста необходимо удалить, для этого нужно раскрыть меню **Edit** и выбрать в нем пункт **Clear** (очистить) или нажать комбинацию клавиш <Ctrl>+<Del>. Если же нужно удалить только

одну строку, то для этого достаточно поместить курсор на удаляемую строку и нажать сочетание клавиш <Ctrl>+<Y>. При необходимости вставить перед текущей строкой дополнительную пустую строку следует воспользоваться комбинацией клавиш <Ctrl>+<N>.

Если же требуется выделенный фрагмент текста переместить либо скопировать, то пользуются буфером обмена. *Буфером* называется специальная область оперативной памяти компьютера, используемая для временного хранения различной информации. Принцип работы с буфером заключается в следующем: сначала в него заносится какая-либо информация, в частности фрагмент текста (который должен быть перед этим предварительно выделен). При перемещении фрагмента он сначала вырезается из того места программы, где он до этого находился, и помещается в буфер. Это выполняется командой **Cut**, находящейся в разделе **Edit** (аналогичный результат получим, нажав комбинацию клавиш <Shift>+<Del>). При копировании сам фрагмент остается на месте, а в буфер отправляется лишь его копия. Копирование фрагмента в буфер выполняется командой **Copy** из того же меню (либо комбинацией клавиш <Ctrl>+<Ins>). Затем, по команде вставки, информация, хранящаяся в буфере, вставляется в другое место данной программы или в другую программу, причем вставка производится туда, куда в данный момент установлен курсор. Команда вставки называется **Paste** (ее аналогом является сочетание клавиш <Shift>+<Ins>) и выполняется только в том случае, если буфер обмена не пустой.

При работе с буфером в системе Turbo Pascal необходимо помнить, что при копировании или перемещении в него какой-либо новой информации, она записывается в конец буфера, вслед за старой информацией, что раньше в нем хранилась. Поэтому, чтобы не запутаться, перед тем, как что-либо записывать в буфер, полезно просмотреть его содержимое командой **Show clipboard** (просмотр буфера). При этом на экране компьютера появится окно с находящимся в буфере текстом. Очистить содержимое буфера можно командой **Clear**.

Рассмотрим одну из операций редактирования на примере программы "банкомат", текст которой приведен в *главе 3*. Обратите внимание, что в тексте программы "банкомат" четыре строчки, содержащие операторы `writeln`, которые выводят полученные результаты, почти идентичны друг другу. После того как мы набрали первую из четырех строчек, три раза скопируем ее, после чего останется внести в скопированные строчки минимальные изменения. Для этого нужно будет в начале выделить первую строчку. Это можно сделать как с помощью клавиатуры, так и с помощью мыши. Когда строка выделена, открываем раздел меню **Edit**. В разделе выбираем пункт **Copy**, после чего содержимое данной строки будет скопировано в буфер. Затем нажатием клавиши <Enter> перемещаем курсор на строчку ниже. После этого снова от-

крываем раздел меню **Edit** и выбираем в нем команду **Paste**, после чего хранящаяся в буфере строка будет вставлена туда, где установлен курсор. Затем еще два раза спускаем курсор на одну строку и каждый раз даем команду **Paste**. Таким образом, под первой строкой появятся три ее копии, которые необходимо будет только немного откорректировать.

Если в процессе редактирования вы сделали ошибку, то можно отменить последнее изменение в тексте с помощью команды **Undo**. Если же вы решили повторить отмененное действие, то это можно сделать командой **Redo**.

Говоря о редактировании текста встроенным редактором, необходимо отметить еще один момент. Ввод текста в данном редакторе может производиться в двух режимах: вставки и замены. По умолчанию используется режим вставки, в котором при вводе какого-либо текста в середину строки текст, расположенный справа от текущей позиции курсора, сдвигается, освобождая место для вставки нового текста. Если при работе в редакторе нажать клавишу <Ins>, то вместо режима вставки будет установлен режим замены, о чем свидетельствует изменение внешнего вида курсора. Курсор, имевший форму горизонтальной черты, теперь будет выглядеть как прямоугольник. При работе в режиме замены ввод нового текста в середину строки будет сопровождаться автоматическим удалением старого текста, поверх которого появляется новый. Такой режим удобнее использовать при корректировке текста, т. к. не нужно тратить время на предварительное удаление старого текста. Если необходимо снова вернуться к режиму вставки, то для этого нужно снова нажать ту же клавишу <Ins>.

П1.3. Использование справочной системы

Естественно, что в процессе работы над текстом программы, при ее отладке, запуске на выполнение, сохранении созданных файлов у пользователей, особенно на начальных этапах работы, могут возникать различные сложности и проблемы. Если рядом с вами нет опытного в вопросах программирования консультанта, то в его роли может выступить сама система программирования, которая имеет мощную, обширную и разветвленную справочную систему, способную оказать вам своевременную и эффективную помощь при создании программ.

Единственным недостатком справочной системы для русскоязычного пользователя является то, что она не русифицирована (вся справочная информация выдается только на английском языке). Поэтому пользователям, не владеющим английским языком, для работы с данной системой желательно запастись двумя англо-русскими словарями: общего назначения и по вычислительной технике и программированию.

Помощь, предоставляемая справочной системой пользователю, может быть оказана несколькими способами. Во-первых, информацию можно получить через системное меню, в котором имеется раздел **Help**. В этом разделе есть ряд пунктов, важнейшими из которых являются **Contents** и **Index**. **Contents** открывает окно, содержащее основное оглавление справочной системы, включающее разделы **How to Use Help** (как работать со справочной системой), **Menus and Hot Keys** (работа с меню и использование функциональных клавиш и комбинаций клавиш), **Editor Commands** (справка по работе с текстовым редактором), **Functions and Procedures** (функции и процедуры Паскаля), **Error messages** (расшифровка сообщений об ошибках), **Reserved Words** (список зарезервированных слов с описанием их значений и применения) и др. При открытии каждого из этих разделов выводится подробная текстовая информация по соответствующей теме. Внешний вид окна содержания приведен на рис. П1.3.

Рис. П1.3. Окно содержания справочной системы

При активизации пункта **Index** пользователь получает доступ к окну с алфавитным списком терминов, используемых в системе программирования. Этот список похож на предметный указатель, расположенный в конце книги, но он имеет некоторые отличительные особенности. Во-первых, для того чтобы найти интересующий вас термин, не обязательно прокручивать весь список, достаточно набрать на клавиатуре начальные буквы интересующего вас слова. Во-вторых, данный список является гипертекстовым. *Гипертекстом* называется текст, имеющий ссылки на другие тексты. Так, если два раза щелк-

нуть мышью на любом из имеющихся в списке терминов или навести на этот термин курсорную рамку и нажать клавишу <Enter>, то в диалоговом окне **Help** появится подробная справочная статья о нем. Кроме того, в большинстве случаев будет приведен список разделов, связанных по содержанию с данной темой, и образцы программ, в которых этот термин используется.

Например, если мы хотим получить справку по работе с оператором `read`, нужно щелкнуть в меню **Help** мышью пункт **Index**, затем достаточно набрать первые буквы `re`, чтобы курсорная рамка установилась в указателе на термин **Read** и нажать клавишу <Enter>. В результате появляется текстовое окно, в котором подробно описывается использование оператора `read` (в частности, из статьи можно узнать, что `read` — это не просто оператор, а стандартная процедура Паскаля; подробнее о том, что такое процедура, рассказано в *главе 8* данной книги), приводится список "родственных" данной теме статей справочной системы (**ReadKey**, **Readln**, **Write**, **Writeln**) и указывается, что в справочной системе имеется пример программы (по-английски — **Sample code**), где используется оператор `read`. Для оператора `read` пример такой программы содержится в файле Eof.pas. Чтобы просмотреть его текст, достаточно два раза щелкнуть мышью название файла.

Помимо описанного ранее способа получения информации через системное меню существует еще один (в ряде случаев он является более быстрым и удобным) — использование контекстной подсказки. Прилагательное "контекстная" означает "относящаяся к данному случаю, к данной ситуации".

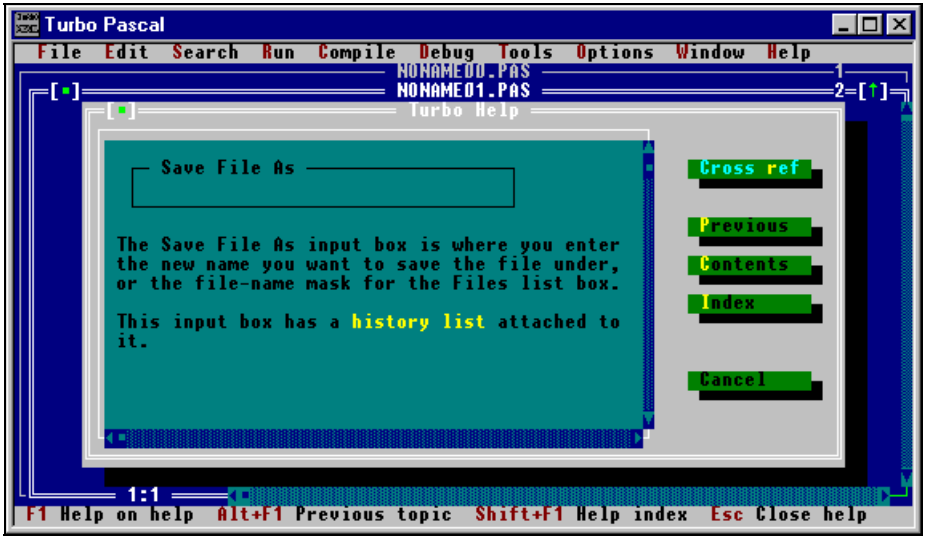


Рис. П1.4. Окно контекстной помощи

Нажав клавишу <F1>, вы можете получить справку по тому режиму работы (работа в текстовом редакторе, открытие файла, сохранение файла), который установлен в данный момент. Например, если мы сталкиваемся с какими-либо трудностями в процессе сохранения файла, то достаточно, находясь в окне сохранения файла, нажать клавишу <F1>, чтобы получить справку именно по данному вопросу (рис. П1.4).

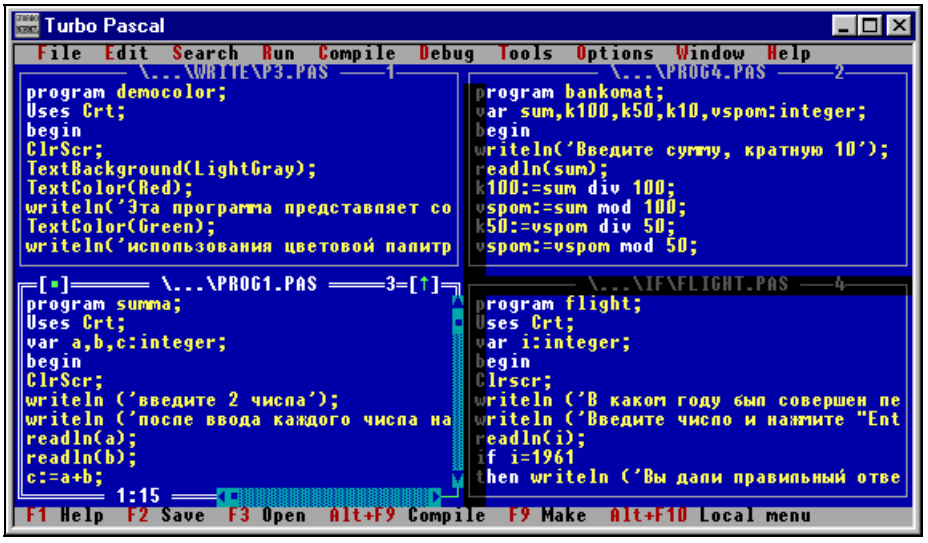
П1.4. Работа с окнами

Все операции, которые мы производим в системе программирования, выполняются в одном из открытых окон. *Окном* называется прямоугольная область, используемая для вывода информации. В окнах могут содержаться исходные тексты программ, результаты работы программ, различная справочная информация, в специальных диалоговых окнах производится сохранение файлов и их открытие, а также настройка различных параметров системы программирования. Окна могут быть развернуты во весь экран или занимать его часть. Размеры окон можно изменять (исключение составляют диалоговые окна, размеры которых фиксированы).

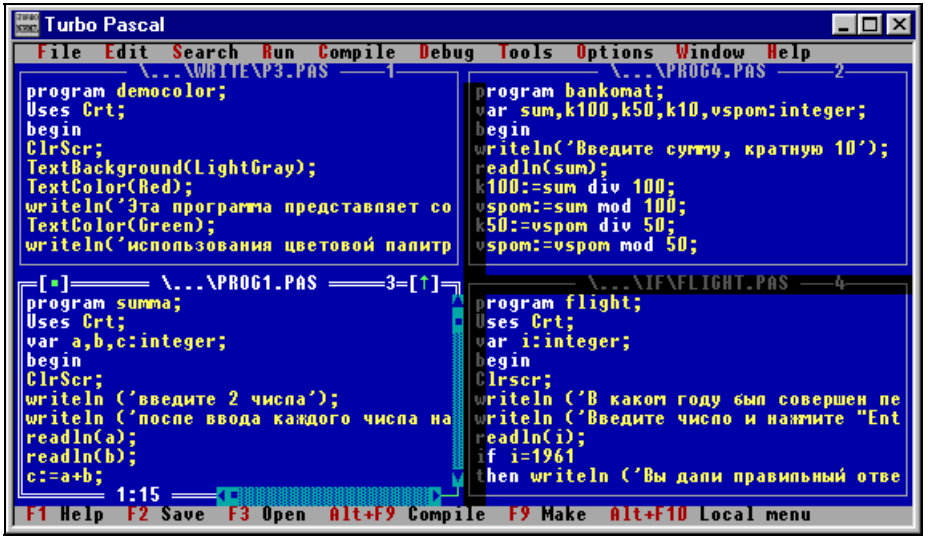
Система программирования Turbo Pascal позволяет в течение одного сеанса работать с несколькими файлами. Если во время сеанса работы в системе вы создаете новый файл или открываете уже существующий, то при этом открывается соответствующее данному файлу новое окно. Одно из окон является активным и находится на переднем плане. Такое окно имеет два управляющих элемента, один из которых позволяет закрыть окно (квадратик в левой верхней части окна), а другой — развернуть окно во весь экран (стрелка в правой верхней части окна). Если окно уже развернуто во весь экран, то разворачивающий элемент сменяется восстанавливающим, который позволяет вернуться к исходному размеру окна. Этот элемент тоже имеет форму стрелки, но двунаправленной. Для каждого окна с программой или результатами работы в правом верхнем углу указывается его порядковый номер. Диалоговые окна, в отличие от окон программ, имеют только закрывающий элемент и не имеют порядкового номера.

Для управления расположением окон на экране, их размерами, перехода между окнами используются команды раздела меню **Window**. Команды этого меню позволяют располагать окна на экране двумя различными способами: каскадом (команда **Cascade**) и черепицей (команда **Tile**) (рис. П1.5). При расположении окон каскадом одно (активное) окно находится на переднем плане, а для других окон выводятся только их заголовки. Этот способ расположения окон применяется по умолчанию. Если во время сеанса работы в системе программирования необходимо сделать активным другое окно, то следует щелкнуть мышью его заголовок, в результате чего это окно станет

активным, а от предыдущего активного окна будет виден только заголовок, хотя оно и не будет закрыто.



а



б

Рис. П1.5. Расположение окон на экране каскадом *(а)* и черепицей *(б)*

Если нужно сделать активным предыдущее по порядковому номеру окно, то используется команда **Previous**, если активным должно быть следующее по порядку окно, то — команда **Next** (здесь имеется в виду порядок открытия

окон в ходе текущего сеанса работы в системе). Если активное окно больше не нужно пользователю, то его можно закрыть командой **Close**. Для того чтобы просмотреть список всех открытых окон, применяется команда **List** (рис. П1.6). По этой команде на передний план выводится диалоговое окно, в котором перечислены все окна, причем первым в списке идет окно, которое было открыто последним. Если какое-либо из имеющихся в списке окон необходимо закрыть, то для этого его нужно выделить курсорной рамкой и щелкнуть экранную кнопку **Delete**, в результате чего окно будет удалено из списка и закрыто.

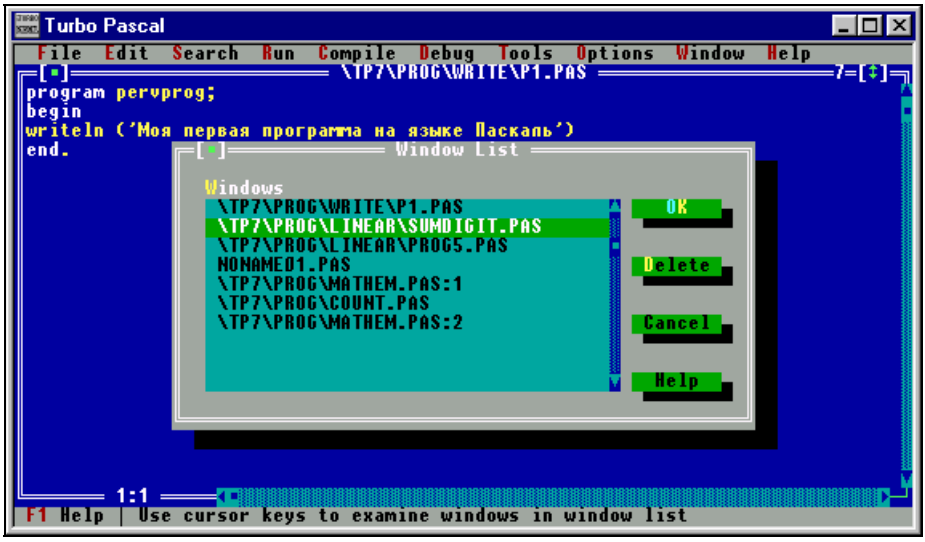


Рис. П1.6. Диалоговое окно, содержащее список окон, открытых во время текущего сеанса работы в системе Turbo Pascal

Режим работы с расположением окон каскадом удобен в том случае, если пользователь во время сеанса работы в каждый момент времени работает только с одним окном, но так бывает не всегда. Нередко возникают ситуации, когда пользователю нужно видеть на экране одновременно несколько окон с текстами программ. В частности, это необходимо в том случае, когда пользователь хочет использовать одну программу в качестве образца для разработки другой или напрямую скопировать фрагмент одной программы в другую. Для расположения нескольких окон таким образом, чтобы они не перекрывали друг друга, используется команда **Tile** (рис. П1.5, *б*). В данном режиме пользователю будут видны все открытые в текущий момент окна, каждое из которых при этом будет занимать только часть экрана. Одно из окон является активным. Его можно определить по наличию в нем управляющих элементов и скроллеров. Другое окно можно сделать активным щелчком по его заголовку

(как и при расположении окон каскадом), но при этом предыдущее активное окно остается на экране и сохраняет свои размеры и местоположение.

Если пользователю необходимо активное окно развернуть во весь экран, то для этого он должен дать команду **Zoom**. Если желательно видеть все открытые окна, но при этом пользователь хочет изменить их взаимное расположение либо размеры окон, то следует воспользоваться командой **Size/Move**. После выполнения этой команды система переходит в специальный режим (о чем свидетельствует изменение цвета заголовка активного окна — из белого он становится зеленым). В данном режиме можно передвигать окна по экрану клавишами <стрелка вверх>, <стрелка вниз>, <стрелка вправо>, <стрелка влево>. Если нужно увеличить или уменьшить размеры окна, то для этого следует использовать комбинации клавиш <Shift>+<стрелка вверх>, <Shift>+<стрелка вниз>, <Shift>+<стрелка вправо>, <Shift>+<стрелка влево>. Подтверждение сделанных изменений осуществляется нажатием клавиши <Enter>. Для выхода из режима перемещения и изменения размеров окон нажимаем клавишу <Esc>.

Если пользователь хочет закрыть все открытые во время сеанса работы окна, то он должен воспользоваться командой **Close all**.

ПРИЛОЖЕНИЕ 2

Основные команды меню системы Delphi

Для лучшего понимания основных команд меню в англоязычных версиях системы Delphi здесь приведен их вариант в русском эквиваленте.

Меню *File*

New — создать новый объект. (Таким объектом может быть: **Application** — проект, **Form** — экранная форма, **Unit** — модуль программы и др.).

Open project — открыть существующий проект.

Open — открыть форму, программный модуль или текстовый файл (этой же командой можно открыть и проект).

Save — сохранить все изменения, которые были произведены в открытых в данный момент файлах. При этом все файлы сохраняются под текущими именами.

Save as — сохранить изменения в открытых файлах проекта. При этом сами файлы можно сохранять под другими именами.

Save Project as — сохранить весь проект под другим именем.

Save All — сохранить все изменения, сделанные в проекте.

Exit — закрыть систему программирования.

Меню *Edit*

Undo — отменить последнее сделанное изменение.

Redo — вернуть отмененное действие.

Cut — переместить выделенный фрагмент в буфер обмена.

Copy — скопировать выделенный фрагмент в буфер обмена.

Paste — вставить фрагмент, находящийся в буфере обмена.

Delete — удалить выделенный фрагмент.

Select all — выделить все в активном окне (весь текст в окне модуля или все объекты в окне формы).

Меню *Search*

Find — открывает диалоговое окно для поиска заданной текстовой строки.

Replace — открывает диалоговое окно для замены одной текстовой строки на другую.

Меню *View*

Forms — открывает диалоговое окно, в котором можно выбрать любую форму проекта и сделать ее активной.

Units — открывает диалоговое окно, в котором можно выбрать любой модуль проекта и сделать его активным.

Toolbars — позволяет добавлять и удалять панели инструментов.

Меню *Run*

Run — запуск программы на выполнение.

Program Reset — прекращение выполнения программы в случае ее зависания.

ПРИЛОЖЕНИЕ 3

Распространенные сообщения об ошибках в Delphi и способы их устранения

Undeclared identifier — переменная не описана. Причиной появления этого сообщения может быть наличие в программе переменной, которая не была описана в разделе описаний. Другой причиной появления этого сообщения может быть ошибка в написании служебного слова, которое воспринимается компилятором как неизвестная переменная. Для устранения описать используемую переменную в разделе описаний или правильно написать служебное слово.

Not enough actual parameters — не хватает фактических параметров. При вызове функции или процедуры для нее забыли указать какой-либо параметр. Вставить недостающий параметр в вызов функции или процедуры.

Too many actual parameters — избыток фактических параметров. При вызове функции или процедуры для нее указали лишний параметр. Удалить лишний параметр в вызове функции или процедуры.

Unterminated string — незавершенная строка. Причиной ошибки обычно является отсутствие апострофа в конце текстовой строки. Для устранения вставить недостающий апостроф.

Unexpected end of file — неверный конец файла. Программист забыл закрыть скобку в комментарии к программе. Поставить недостающую фигурную скобку.

Incompatible types — несоответствие типов. Чаще всего такое сообщение появляется при попытке присвоить переменной недопустимое значение (например, присвоить переменной целого типа строковое значение). Для исправления ошибки ввести в программе дополнительную переменную того типа, который соответствует присваиваемому значению.

Missing operator or semicolon — отсутствует оператор или точка с запятой. Программист забыл поставить точку с запятой после одного из операторов. Вставить недостающую точку с запятой.

',' or ':' expected but ';' found — ожидается наличие запятой или двоеточия, но вместо них обнаружена точка с запятой. Причина обычно заключается в том, что в разделе описания переменных программист после имени переменной или имен группы переменных забыл указать их тип. Для исправления поставить недостающее двоеточие и после него тип переменной или переменных.

')' expected but ';' found — при вызове процедуры или функции после перечисления параметров забыли закрыть скобку. Добавить недостающую скобку.

Statement expected but 'PROCEDURE' found — обычной причиной такого сообщения является отсутствие в конце составного оператора служебного слова `end`. Поставить недостающее слово `end`.

Declaration expected but identifier found — обычной причиной подобного сообщения является отсутствие служебного слова `begin` в начале составного оператора. Вставить недостающее слово `begin`.

Литература

1. Borland Delphi 6 for Windows. Developers Guide. Borland Software Corporation. Scotts Valley, CA, 2001.
2. Borland Delphi 6 for Windows. Object Pascal Language Guide. Borland Software Corporation. Scotts Valley, CA, 2001.
3. Borland Delphi 6 for Windows. Quick Start. Borland Software Corporation. Scotts Valley, CA, 2001.
4. Kent Reisdorph. Teach Yourself Borland Delphi 4 in 21 Days. Macmillan Publishing, Indianapolis, IN, 1999.
5. Абрамов В. Г., Трифонов Н. П., Трифонова Н. Г. Введение в язык Паскаль. — М.: Наука, 1988.
6. Бабушкина И. А., Бушмелева Н. А., Окулова С. М., Черных С. Ю. Практикум по Turbo Pascal. — М.: АБФ, 1998.
7. Боон К. Паскаль для всех. — М.: Энергоатомиздат, 1988.
8. Васюкова Н. Д., Тюляева В. В. Практикум по основам программирования. Язык Паскаль. — М.: Высш. шк., 1991.
9. Гольденберг В. А. Введение в программирование. — Минск: ООО "Харвест", 1997.
10. Григас Г. Начала программирования. — М.: Просвещение, 1987.
11. Емелина Е. И. Основы программирования на языке Паскаль. — М.: Финансы и статистика, 1997.
12. Епанешников А. М., Епанешников В. А. Программирование в среде Turbo Pascal 7.0. — М: Диалог-МИФИ, 1995.
13. Жуков А. Изучаем Delphi. — СПб.: Питер, 2000.
14. Культин Н. Б. Turbo Pascal в задачах и примерах. — СПб.: БХВ-Петербург, 2000.

15. Культин Н. Б. Программирование в Turbo Pascal 7.0 и Delphi. — СПб.: БХВ-Петербург, 1999.
16. Львовский М. Б. Методическое пособие по информатике, 1999 (электронное издание).
17. Малышев О. В. Основы Delphi, 1999 (электронное издание).
18. Молчанова С. И. Основы программирования. Турбо-Паскаль 7.0 — М.: АСТ, 2000.
19. Немнюгин С. А. Turbo Pascal. Практикум. — СПб.: Питер, 2002.
20. Немнюгин С. А. Turbo Pascal. Учебник. — СПб.: Питер, 2002.
21. Пестриков В. М., Маслобоев А. Н. Turbo Pascal 7.0. Изучаем на примерах. — СПб.: Наука и техника, 2003.
22. Пестриков В. М., Маслобоев А. Н., Федоров О. К. Программирование в системе Turbo Pascal 7.0. Учебное пособие. — СПб.: СПбГТУРП, 2002.
23. Сергиевский М. В., Шалашов А. В. Turbo Pascal 7.0. Язык, среда программирования. — М.: Машиностроение, 1994.
24. Симонович С. В. и др. Информатика. Базовый курс. — СПб.: Питер, 2001.
25. Слабун Т. Интерактивный курс изучения Borland Delphi 6, 2001 (электронное издание).
26. Ставровский А. Б. Turbo Pascal 7.0. Учебник. — К.: BHV-Киев, 2000.
27. Фаронов В. В. Основы Турбо-Паскаля. — М.: Учебно-инженерный центр "МВТУ-ФЕСТО ДИДАКТИК", 1992.
28. Шауцукова Л. З. Информатика. — М.: Просвещение, 2000.

Предметный указатель

М

О

П

Р

С

Т

Ф

Ц