А. Н. Маслобоев

Языки и методы программирования

Основы программирования в среде Lazarus

Учебное пособие

Санкт-Петербург
2020

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ПРОМЫШЛЕННЫХ ТЕХНОЛОГИЙ И ДИЗАЙНА»

ВЫСШАЯ ШКОЛА ТЕХНОЛОГИИ И ЭНЕРГЕТИКИ

Языки и методы программирования

Основы программирования в среде Lazarus

Учебное пособие

Санкт-Петербург
2020

УДК 681.3 (075)
ББК 22.18я7
М 316

Маслобоев А.Н. Языки и методы программирования. Основы программирования в среде Lazarus: учебное пособие; ВШТЭ СПбГУПТД. — СПб., 2020. — 88 с. - ISBN 978-5-91646-243-2

Учебное пособие разработано в соответствии с программой курсов «Языки и методы программирования» и «Визуальные среды программирования» для бакалавров по направлению подготовки 01.03.02 «Прикладная математика и информатика».

Рецензенты:

зав. кафедрой радиотехники и информационных технологий Санкт-Петербургского государственного университета кино и телевидения, профессор, д-р техн. наук В.М. Пестриков;

заведующий кафедрой ИИТиСУ ВШТЭ СПбГУПТД, кандидат техн. наук, профессор В.И. Сидельников

Утверждено Редакционно-издательским советом ВШТЭ СПбГУПТД в качестве учебного пособия

ISBN 978-5-91646-243-2

Оглавление

Введение

Данное учебное пособие посвящено основам программирования на языке Object Pascal в среде Lazarus. Object Pascal является результатом развития и совершенствования классического языка программирования Паскаль, разработанного швейцарским ученым Никлаусом Виртом еще в 70-е годы XX столетия.

Этот язык получил свое название в честь знаменитого французского ученого XVII столетия Блеза Паскаля. Паскаль был не только ученым-энциклопедистом, но и выдающимся изобретателем, создавшим одно из первых в мире механических счетных устройств. Изобретение Паскаля положило начало тому процессу, который привел к созданию современной вычислительной техники, без которой ныне немыслимо существование и дальнейшее развитие человеческой цивилизации. Поэтому название данного языка программирования, который, по замыслу его создателя, должен был стать образцовым языком, своего рода «латынью» программирования, является вполне закономерным.

Изначально Паскаль задумывался Никлаусом Виртом как средство для обучения студентов базовым конструкциям и принципам программирования. Но с течением времени этот язык вышел за чисто учебные рамки и стал использоваться также и для решения серьезных профессиональных задач.

Первые широко распространенные версии данного языка (в частности, Turbo Pascal) работали на базе операционной системы MS DOS, и в их основе лежало алгоритмическое программирование. Но уже в 1986 году группой специалистов фирмы Apple под руководством Ларри Теслера была разработана новая версия языка под названием Object Pascal. Данная версия, как видно из ее названия, была ориентирована на современные технологии объектно-ориентированного программирования (хотя некоторые элементы указанной технологии уже имелись и в Turbo Pascal).

Object Pascal был положен в основу интегрированной среды разработки программного обеспечения Delphi, которая изначально была создана американской компьютерной фирмой Borland. В настоящее время развитием данной среды программирования занимается другая американская компания Embarcadero Technologies. Эта среда была разработана в соответствии с концепцией визуального проектирования приложений. Очередная версия Delphi, известная под названием Delphi Sydney, была выпущена в мае 2020 года. Эта среда программирования позволяет разрабатывать современные полноценные приложения с графическим интерфейсом как для операционной системы Windows, так и для других широко используемых систем.

В то же время возникла насущная потребность в создании системы программирования, которая обладала бы такими же функциональными возможностями, как Delphi, но при этом была бы свободно распространяемым программным продуктом. За решение этой задачи взялась международная группа программистов, в которую изначально входили Клифф Бейсеман, Шейн Миллер и Майкл Хесс. Впоследствии к этой группе присоединился ряд других специалистов в области информационных технологий и программирования.

Эта была уже не первая попытка создания такой среды программирования. Предыдущая попытка, получившая название «проект Megido», закончилась неудачей. Новый (на сей раз успешный) проект был фактически возрождением предыдущего. Этот проект получил название «Lazarus» в честь Лазаря из Вифании, который, согласно библейскому преданию, был чудесным образом воскрешен из мертвых спустя несколько дней после смерти.

Первая версия среды Lazarus была выпущена в 2001 году и работала на платформе операционной системы Windows. Затем в течение двух десятилетий XXI века продолжалось развитие и совершенствование этой среды программирования, росло количество операционных систем, на базе которых Lazarus может работать. Последняя (на момент публикации данного учебного пособия) версия Lazarus была выпущена в июле 2020 года, и, помимо Windows, она работает на таких платформах, как Linux, BSD и MacOS.

В системе используется компилятор Free Pascal Compiler (сокращенно FPC), который также является свободно распространяемыми программным продуктом. Lazarus позволяет создавать как консольные приложения, так и приложения с полноценным графическим интерфейсом.

Среда Lazarus была использована при разработке многих широко используемых программных продуктов. К их числу относятся файловые менеджеры Total Commander и Double Commander, приложение для моделирования водораспределительных сетей Epanet, архиваторы, аудиоредакторы, текстовые редакторы, панель управления операционной системой Ubuntu и целый ряд других. Поэтому изучение данной среды и работа в ней позволит студентам не только получить теоретические знания, но и приобрести опыт разработки приложений, который будет востребован в их будущей профессиональной деятельности.

Глава 1. Разработка простой программы

В этой главе описывается создание в среде Lazarus простой программы и в ходе ее разработки будут рассмотрены основные элементы интерфейса среды и основные принципы работы в ней.

Как было сказано во введении к данному пособию, Lazarus является свободно распространяемым программным продуктом. Дистрибутив Lazarus можно загрузить на официальном сайте lazarus-ide.org. В зависимости от того, какая операционная система и какая версия этой системы установлена на компьютере, можно выбрать необходимую версию дистрибутива. В частности, для операционной системы Windows существует 32-разрядная и 64-разрядная версия дистрибутива Lazarus.

В процессе установки среды на компьютере пользователя создается папка с именем Lazarus. Для того чтобы начать работу с данной средой программирования, нужно найти в этой папке файл с именем Startlazarus.exe. Это – головной файл, который запускает среду. На экране компьютера появляется окно-заставка, а затем окно настройки системы Lazarus (см. рис. 1.1).

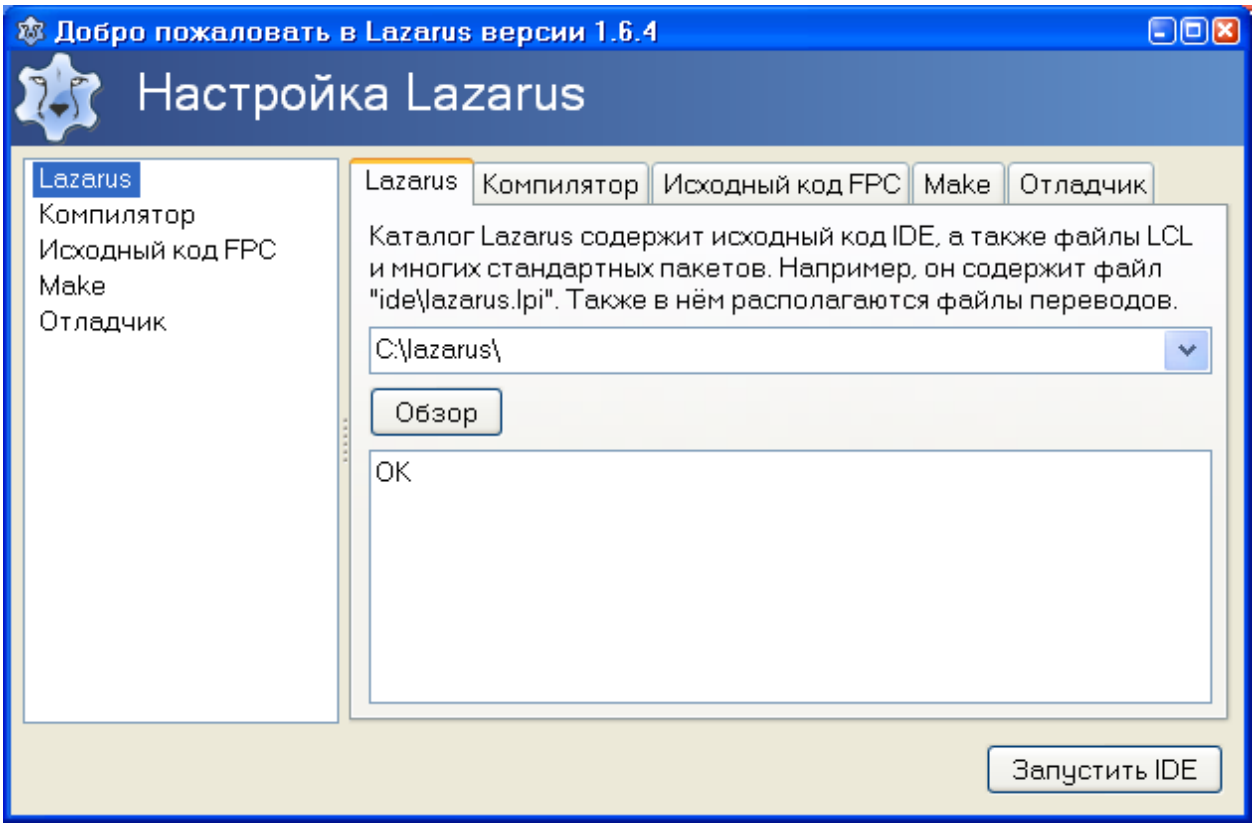

Рис. 1.1. Начальный экран системы Lazarus

В правой нижней части окна настройки расположена экранная кнопка «Запустить IDE». Щелчок мышью на этой кнопке позволяет открыть основной

экран среды программирования, внешний вид который показан на рис.1.2. Рассмотрим основные элементы данного экрана.

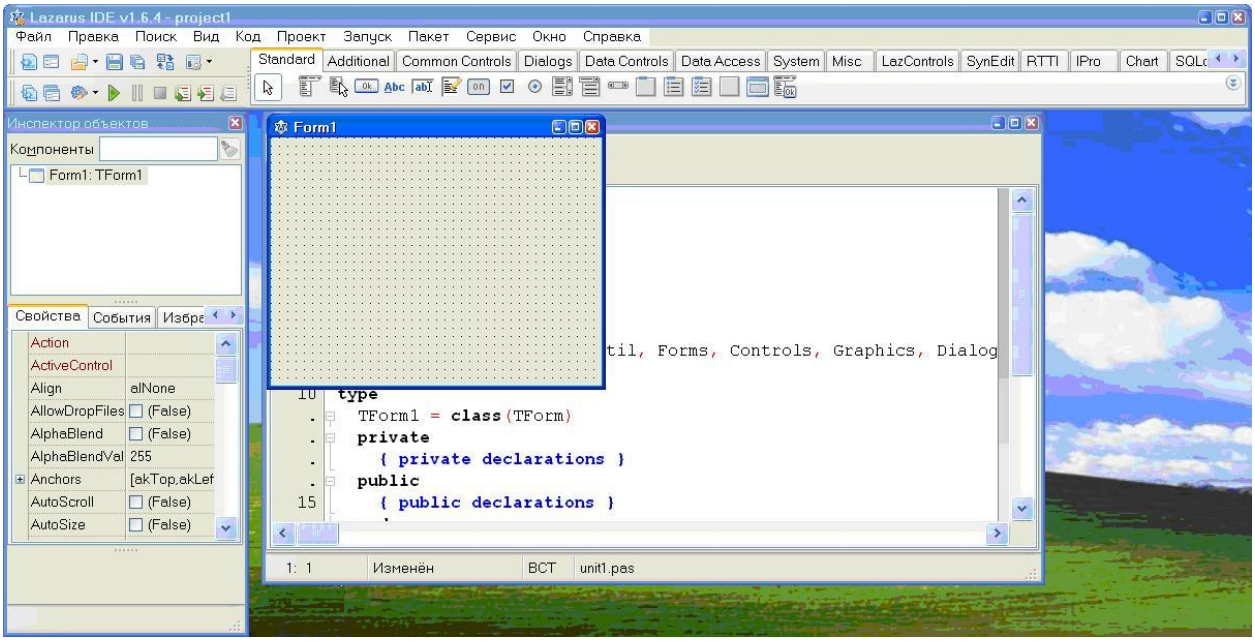

Рис. 1.2. Основной экран системы Lazarus

В верхней части окна находится строка меню, под которой слева расположены панели инструментов, а справа набор вкладок, которые называются палитрами компонентов (см. рис. 1.3 и рис. 1.4). Щелчком мыши на заголовке пользователь может активизировать необходимую палитру.

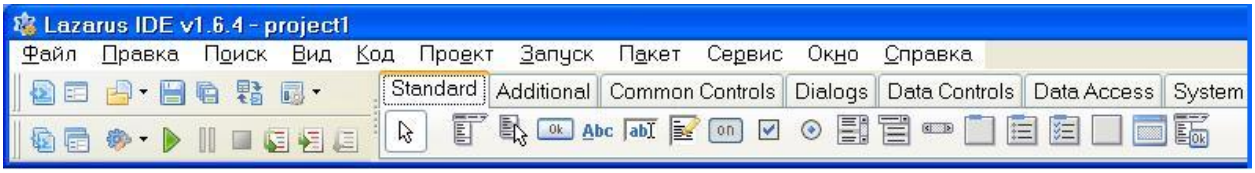

Рис. 1.3. Меню, панели инструментов и палитры компонентов. Активной является стандартная палитра

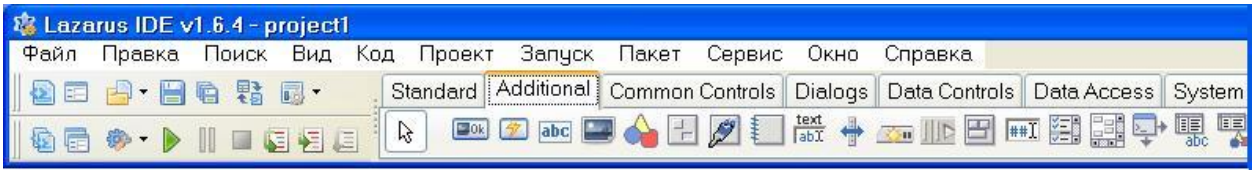

Рис. 1.4. Меню, панели инструментов и палитры компонентов. Активной является дополнительная палитра

Основная экранная форма (рис. 1.5) представляет собой окно, на котором размещаются элементы пользовательского интерфейса разрабатываемого приложения. К таким элементам относятся надписи, текстовые окна, экранные кнопки, флажки, переключатели, элементы обычного и контекстного меню и ряд других. Также на форме могут находиться

управляющие элементы, которые не относятся к пользовательскому интерфейсу, но оказывают влияние на работу приложения (например, таймер). Когда идет процесс создания приложения, на форме находится пунктирная сетка, которая удобна для позиционирования элементов интерфейса. Когда уже созданное приложение запускается на выполнение, пунктирная сетка исчезает, и поверхность экранной формы становится гладкой. Таким образом, по внешнему виду экранной формы достаточно легко отличить режим создания приложения от рабочего режима.

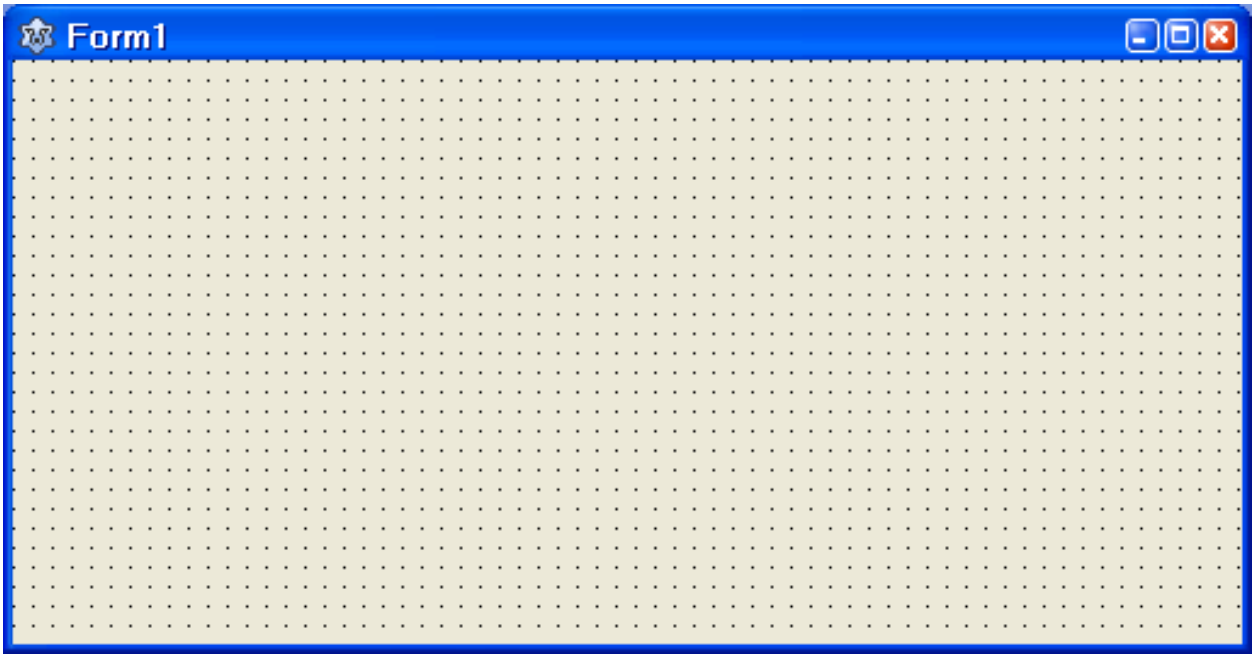

Рис. 1.5. Основная экранная форма

Если форма по каким-либо причинам не видна на экране, то ее всегда можно активизировать и вывести на экран через раздел меню **Вид → Переключить форму/модуль.**

Еще одним важным и необходимым элементом интерфейса среды Lazarus является **Инспектор объектов**. Этот элемент имеет достаточно сложную внутреннюю структуру. Инспектор объектов содержит верхнее окно, в котором находится название текущего компонента, и внизу ряд вкладок. Важнейшими из этих вкладок, которые используются постоянно, являются вкладка свойств и вкладка событий.

На вкладке свойств в алфавитном порядке перечислены все свойства активного компонента. Примерами таких свойств являются имя объекта, цвет, содержание надписи, имеющейся на объекте (если она есть), характеристики шрифта, которым сделана эта надпись, геометрические размеры объекта (его высота и ширина), положение объекта на форме (оно определяется относительно верхней и левой границы окна), видимость объекта и другие.

На вкладке событий также в алфавитном порядке перечислены те события, на которые может реагировать данный объект. Примерами событий являются одиночный щелчок левой или правой клавишей мыши на объекте, двойной щелчок мыши, создание объекта, изменение его размеров и т.д. Если данное событие будет задействовано в программе, то для него создается отдельная процедура. На рис.1.6 показан внешний вид инспектора объектов в случае, когда активизированы разные вкладки.

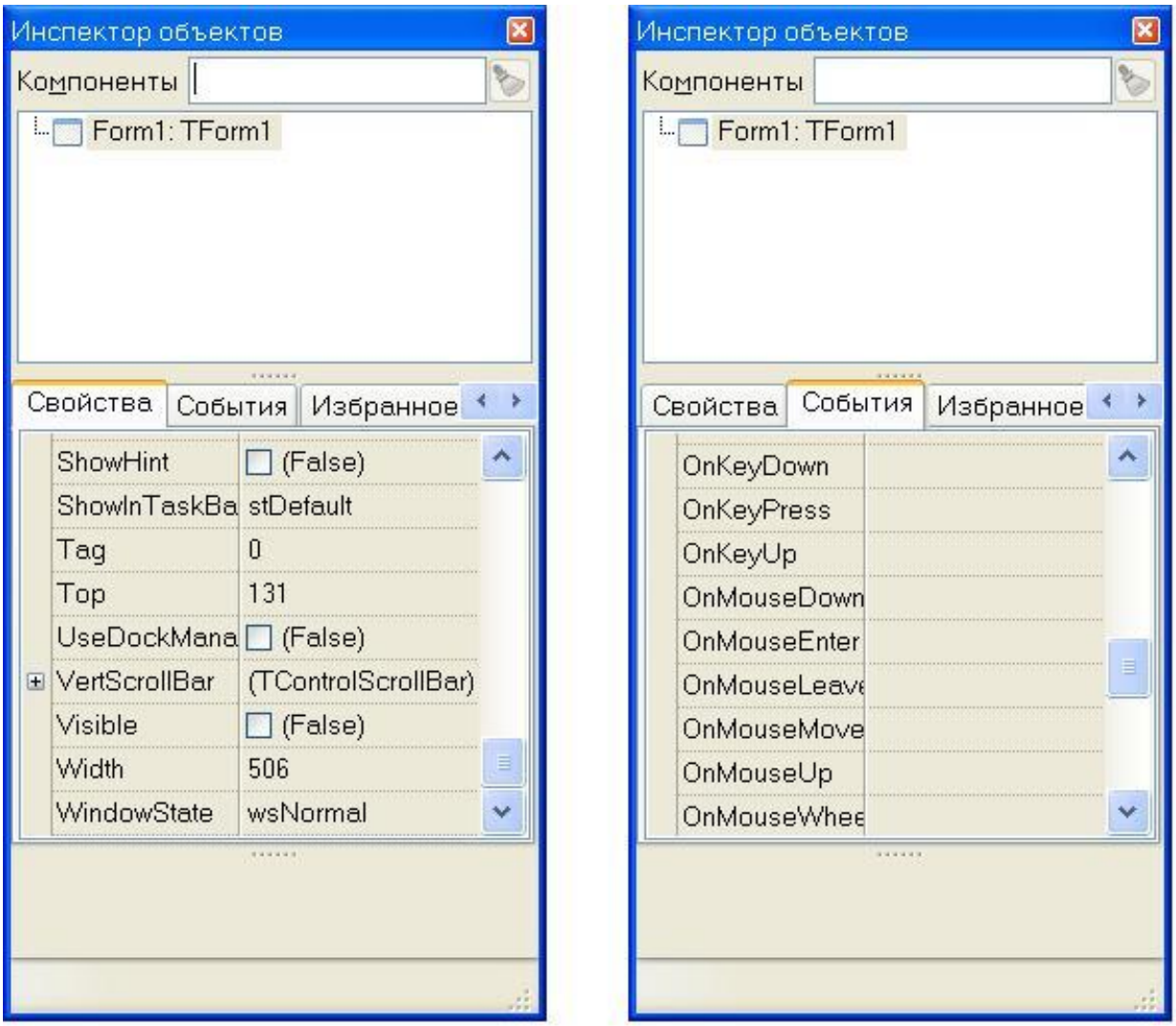

Рис. 1.6. Инспектор объектов. Слева показана ситуация, когда в Инспекторе объектов активизирована вкладка свойств, справа – вкладка событий

После того как объекты приложения размещены на форме и произведена настройка их свойств с помощью инспектора объектов, как правило, возникает необходимость написания программного кода. Этот код пишется на языке Object Pascal в специальном окне, внешний вид которого приведен на рис. 1.7.

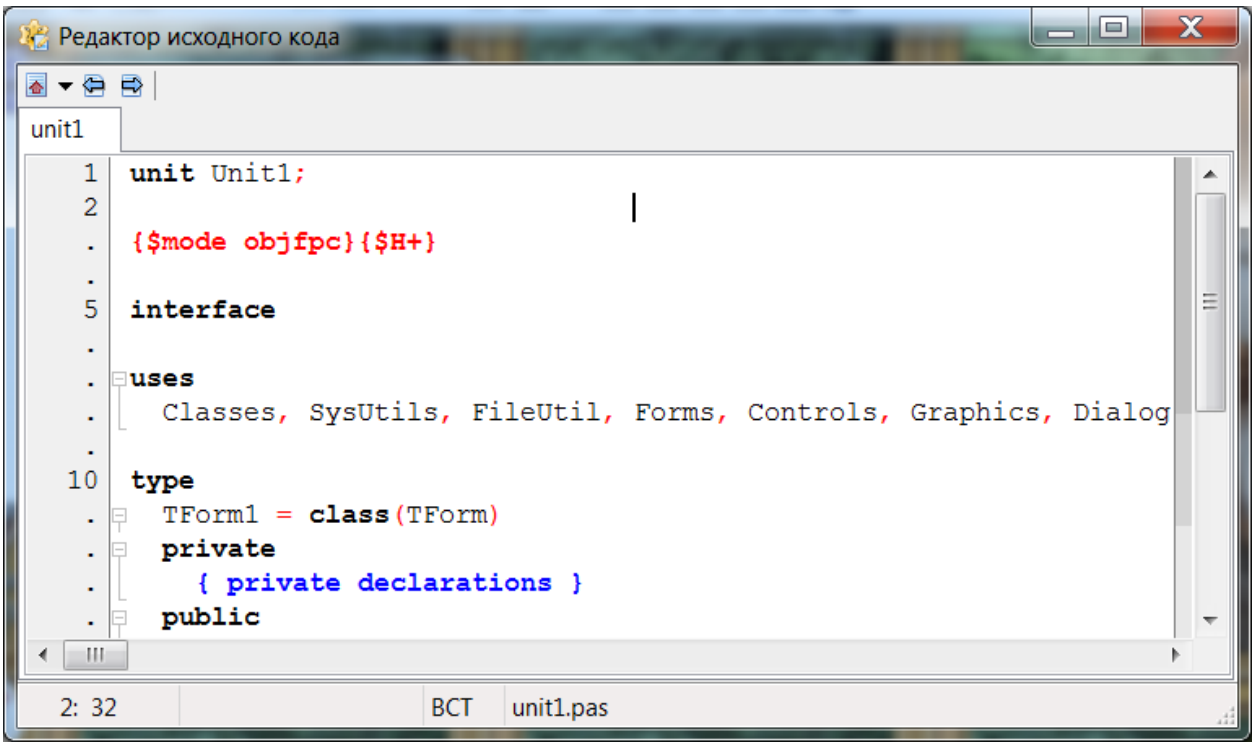

Рис. 1.7. Окно для редактирования программного кода

После ознакомления с основными элементами окна среды Lazarus рассмотрим пример создания приложения в данной среде. Нашей задачей будет создание приложения, которое будет менять цвет основной экранной формы. Изменение цвета формы будет производиться с помощью группы из 8 переключателей. Кроме этой группы на форме должна быть экранная кнопка «**Выход**», которая закрывает окно приложения, и кнопка «**Сведения об авторе**», щелчок на которой открывает или закрывает надпись с информацией о разработчике приложения.

Работа над новым приложением будет начинаться с создания папки для хранения файлов этого приложения. Для этого в разделе меню **Файл** выбираем пункт «**Сохранить как»**. Щелчок мышью на этом пункте открывает специальное диалоговое окно (см. рис. 1.8). В этом окне выбираем подходящее место на компьютере для создания папки. В данном случае создание новой вложенной папки производится в папке **Projects**, вложенной в папку **Lazarus**. Для создания новой папки достаточно щелкнуть правой кнопкой мыши и в открывшемся контекстном меню выбрать пункт «**Создать»**, а затем подпункт **Папку**. В результате появляется заготовка для создания новой папки в виде рамки, в которую нужно ввести имя этой папки. В нашем примере папке было присвоено имя **Colors**. Затем, необходимо щелкнуть экранную кнопку «**Открыть»,** находящуюся в правой нижней части данного диалогового окна. После открытия папки название кнопки меняется на «**Сохранить»** (см. рис. 1.9)**.** Щелчок на этой кнопке позволяет сохранить два основных файла проекта **project1.lpi** и затем **unit1.pas**. Проектом называется приложение, находящееся в стадии разработки.

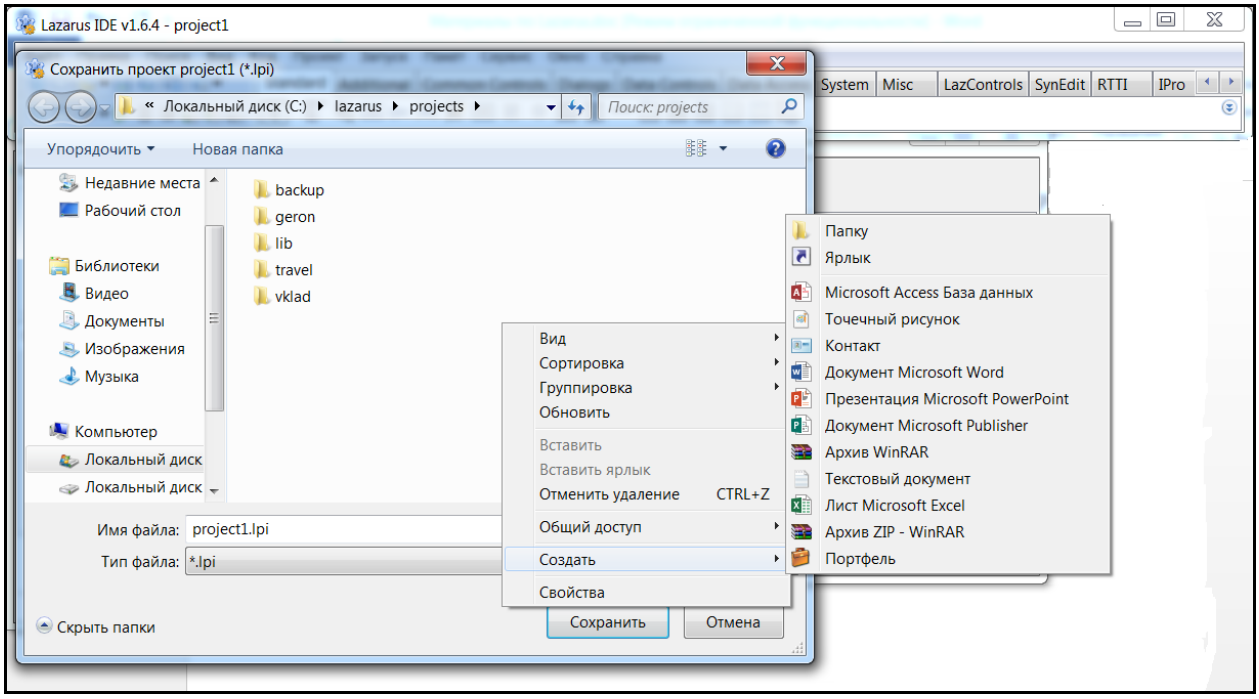

Рис. 1.8. Создание новой папки в диалоговом окне для сохранения файлов проекта

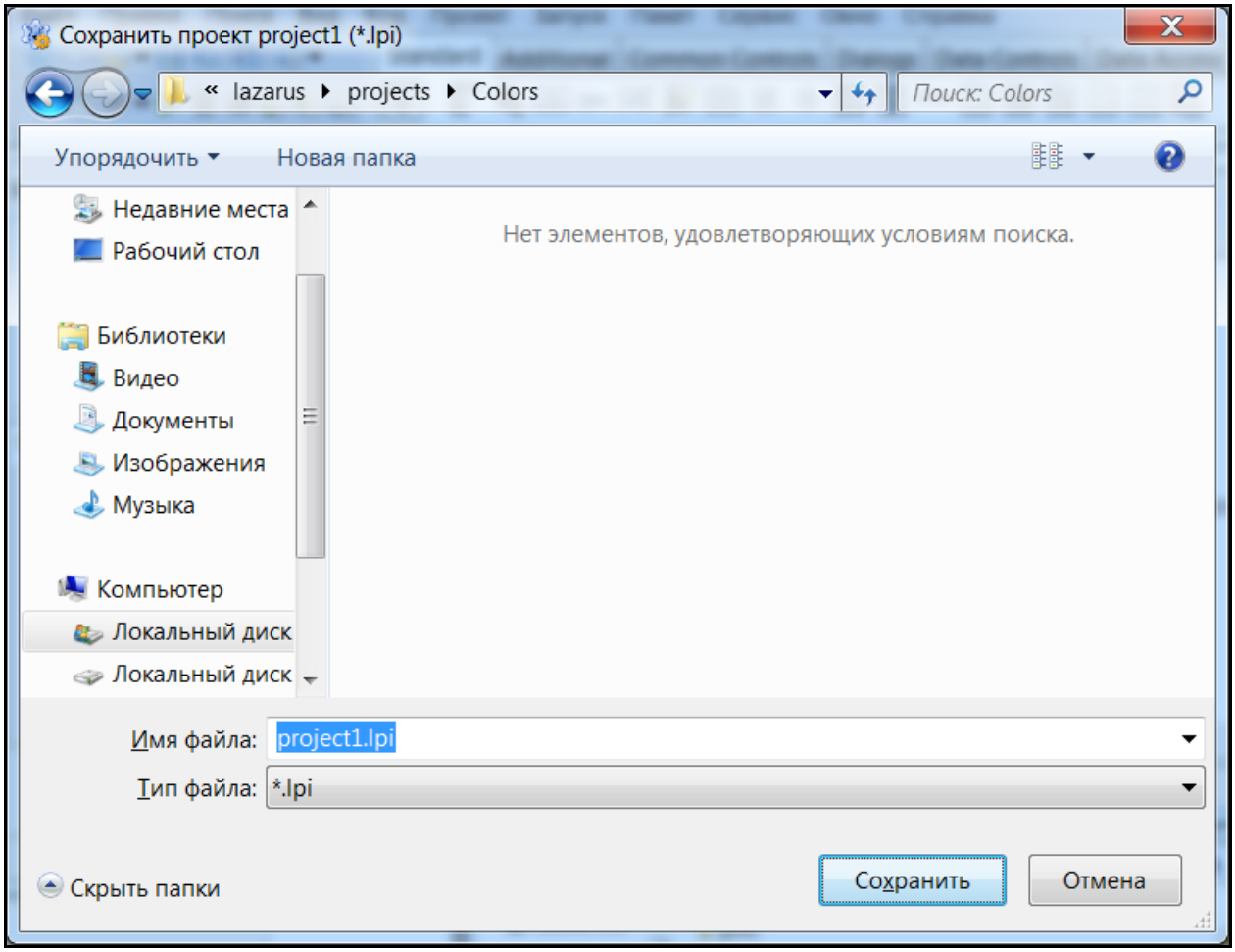

Рис. 1.9. Сохранение файлов проекта в новой папке

После сохранения основы проекта можно приступать собственно к его разработке. Процесс разработки начинается с создания пользовательского интерфейса. Для того чтобы в окне основной экранной формы создать новый элемент, достаточно выбрать его на палитре компонентов щелчком мыши, а затем щелчком на форме разместить его в заданном месте.

Первым создаваемым элементом будет надпись «Я программирую в среде Lazarus», которая должна быть размещена вверху главной формы. Заготовка для надписи находится на палитре компонентов на вкладке **Standard** (Стандартная) и имеет следующий вид  . Чтобы убедиться, что выбрана именно нужная заготовка, достаточно подвести к ней указатель мыши и задержать его на несколько секунд. Рядом с заготовкой появляется всплывающая подсказка «**TLabel**», т.е. название заготовки по-английски. Затем после щелчка на форме там появляется объект **Label1**, т.е. **Надпись1.**

Затем для настройки свойств надписи следует обратиться к окну «**Инспектор объектов**». В этом окне должна быть активной вкладка **Properties** (Свойства). Выбираем на вкладке **Properties** свойство **Caption**, которое отвечает за содержание надписи. Вместо имеющегося по умолчанию значения свойства «Label1» вводим следующее: «Я программирую в среде Lazarus». После нажатия клавиши **Enter** это же содержание отобразится на основной экранной форме.

Следует обратить внимание, что для этого объекта и всех последующих меняется только значение свойства **Caption**. Значение свойства **Name**, т.е. имя объекта, должно оставаться неизменным.

Далее можно изменить свойства шрифта, которым отображается надпись. Для этого находим на вкладке **Properties** свойство **Font**. Справа от названия свойства находится кнопка, на которой изображено многоточие. Эта кнопка называется кнопка-построитель. Щелчок на этой кнопке открывает дополнительное диалоговое окно для настройки свойств шрифта (см. рис. 1.10). Это окно русифицировано, и поэтому работа с ним не вызовет у пользователя никаких трудностей. С помощью данного окна можно менять гарнитуру шрифта, его начертание, размер и цвет. В данном случае выбираем гарнитуру **Arial Unicode MS**, полужирное начертание, размер шрифта – 20 пунктов и темно-синий цвет. Для подтверждения внесенных изменений щелкаем в окне мышью экранную кнопку **Ok**. После этого диалоговое окно закрывается и сделанные изменения вступают в силу.

Сама основная экранная форма является таким же объектом, как и все остальные, и ее свойства также можно изменять. В частности, можно изменять фоновый цвет формы и содержание надписи, находящейся в заголовке формы. Перед этим нужно сделать форму активным объектом, щелкнув на любом свободном месте экранной формы. Теперь в Инспекторе объектов отражаются свойства самой формы. Для изменения фонового цвета следует обратиться к свойству **Color**. Щелкнув треугольную стрелку,

расположенную справа от названия свойства, можно открыть раскрывающийся список с названиями цветов на английском языке. В данном проекте для фона формы был выбран светло-зеленый цвет, который по-английски называется **ClMoneyGreen** (приставка **Cl** представляет собой сокращение от английского слова **Color** и должна в языке Object Pascal обязательно предшествовать названию цвета). Для изменения содержания заголовка формы используется уже знакомое нам свойство **Caption**. Вместо имеющегося по умолчанию неинформативного значения «Form1» вводим значение «Работа с цветом формы» и для подтверждения нажимаем **Enter**.

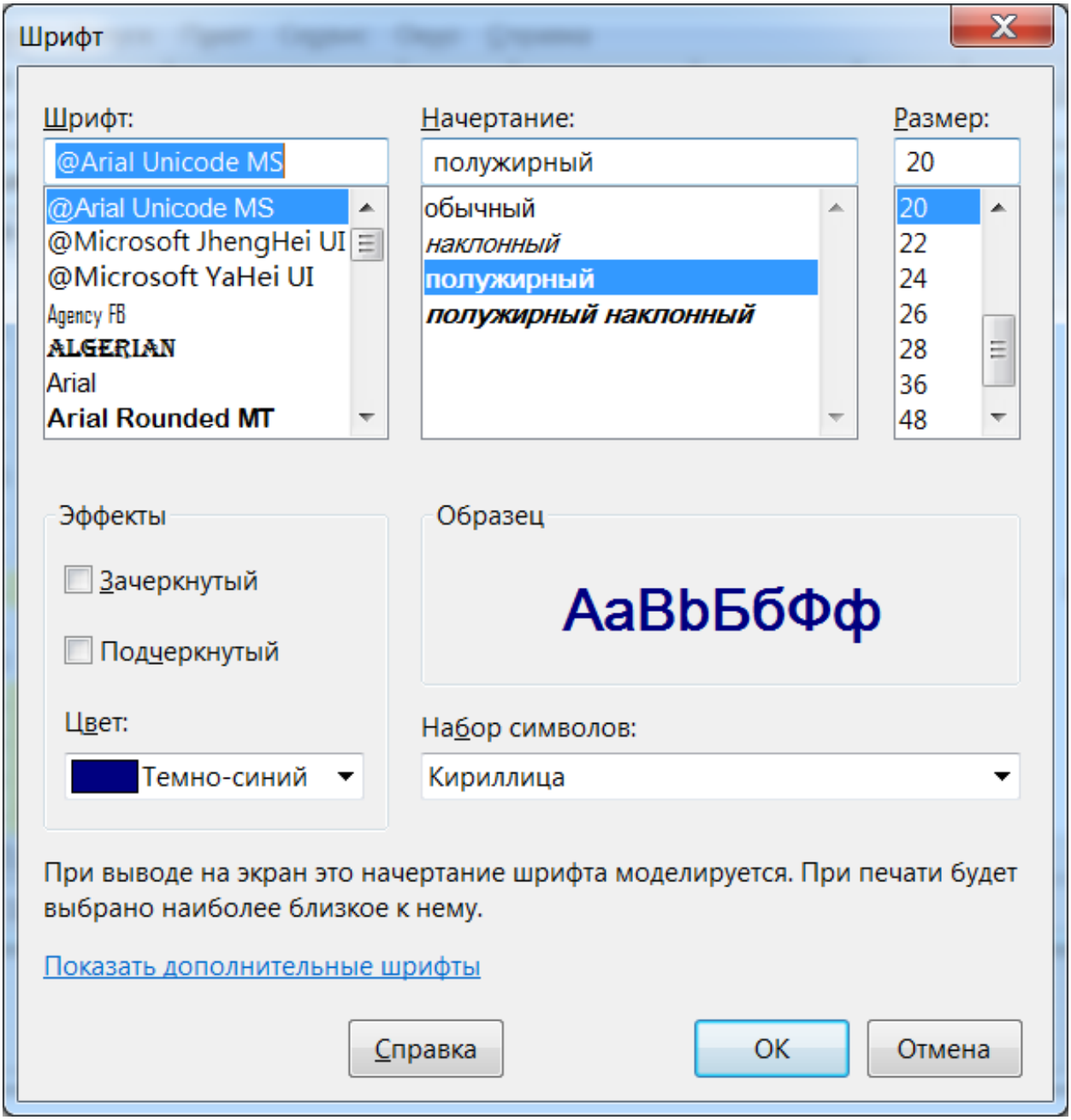

Рис. 1.10. Диалоговое окно настройки свойств шрифта

Следующими объектами, которые создаются в приложении, являются 8 переключателей, каждый из которых окрашивает форму в определенный цвет: красный, желтый, зеленый, синий, серебристый, розовый, бирюзовый или лайм. Для оформления этих переключателей создается специальная рамка. Заготовка для рамки также находится на стандартной палитре компонентов и выглядит следующим образом:  . Созданный объект называется **GroupBox1**. В верхней части рамки имеется одноименный заголовок. Меняем содержание заголовка в Инспекторе объектов с помощью свойства **Caption.** Теперь вместо «GroupBox1» там будет написано: «Выберите цвет формы».

Далее внутри рамки помещаем переключатели. Заготовка для переключателя также находится на стандартной палитре компонентов и выглядит так:  . Первый из создаваемых переключателей получает имя **RadioButton1**. Второй переключатель располагается ниже первого и приобретает имя **RadionButton2,** и так далее вплоть до последнего с именем **RadioButton8**. С помощью свойства **Caption** меняем для первого переключателя содержание надписи, расположенной справа от него, с RadioButton1 на «Красный», для второго - с RadioButton2 на «Желтый», и так далее для каждого переключателя из группы вплоть до последнего восьмого, который снабжается надписью «Лайм».

Затем на форме создаются две экранные кнопки. Заготовка для экранной кнопки также находится на стандартной палитре компонентов и имеет вид: 

. Кнопки размещаем в правой нижней части формы. Первая кнопка получает имя **Button1**. У кнопок также имеется свойство **Caption**. С помощью этого свойства через инспектор объектов меняем содержание надписи на кнопке с Button1 на «Сведения об авторе». С помощью свойства **Font** увеличиваем шрифт надписи и делаем его полужирным. Вторая кнопка получает имя **Button2**. Через свойство Caption меняем надпись на кнопке на слово «Выход», поскольку данная кнопка будет закрывать главную форму.

Последним элементом пользовательского интерфейса является надпись, содержащая сведения о разработчике приложения. Эта надпись имеет две особенности. Во-первых, эта надпись по длине не помещается внутри формы. Поэтому создадим не обычную надпись, а многострочное текстовое поле. Заготовка для такого объекта также находится на стандартной палитре компонентов и имеет следующий вид:  . Создаваемый на экранной форме объект получает название **Memo1**.

Теперь ничто не препятствует тому, чтобы разместить информацию об авторе в три строки. Для этого в инспекторе объектов находим у поля **Memo1** свойство **Lines**. Справа от названия свойства расположена кнопка-построитель. Щелчок по этой кнопке открывает дополнительное диалоговое

окно, в котором можно набирать текст, подобно тому как это делается в стандартном приложении Блокнот операционной системы Windows (см. рис. 1.11). Ввод текстовой информации завершается щелчком по расположенной в нижней части диалогового окна экранной кнопке **Ok**.

Во-вторых, данный объект (**Memo1**) по умолчанию, т.е. сразу после запуска программы на выполнение, должен быть невидим. Для того чтобы выполнить это условие, необходимо вновь обратиться к **Инспектору объектов** и на вкладке свойств найти свойство под названием **Visible**. Данное свойство относится к логическим (булевым) величинам, которые могут принимать только одно из двух значений **True** или **False**, т.е. **Истина** или **Ложь**. Снимая установленный справа от названия свойства флажок, устанавливаем тем самым значение **False**. Теперь многострочная надпись в рабочем режиме приложения не будет видна до выполнения определенных действий. На этом работу над пользовательским интерфейсом приложения закончена. Внешний вид приложения приведен на рис.1.12.

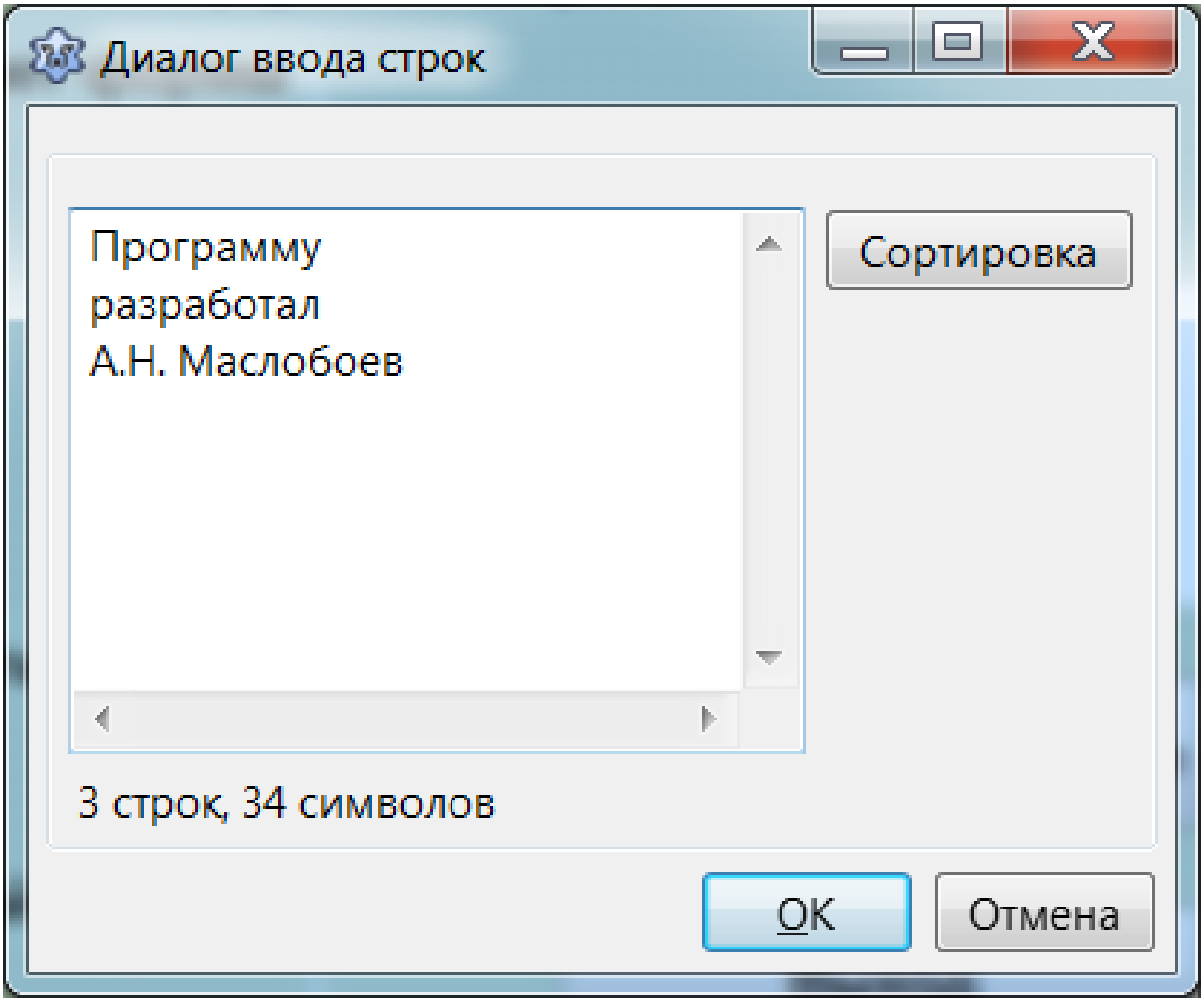

Рис. 1.11. Диалоговое окно, используемое для ввода текста в многострочное текстовое окно

Далее можно приступать к написанию программного кода для приложения. Начнем с кода для экранной кнопки **Button2** (кнопка «**Выход**»). Дважды щелкаем мышью на объекте. В результате в окне кода автоматически создается заготовка для процедуры, описывающей реакцию созданного объекта на щелчок левой клавиши мыши. Такая заготовка содержит название процедуры **TForm1.Button2Click**, а также служебные слова **begin** и **end**, отмечающие начало и конец тела процедуры. Между этими словами вписываем команду **Close**, которая закрывает рабочий режим приложения и возвращает нас в режим конструирования. В итоге текст процедуры должен выглядеть следующим образом:

```
procedure TForm1.Button2Click(Sender: TObject);
begin
 Close;
end;
```

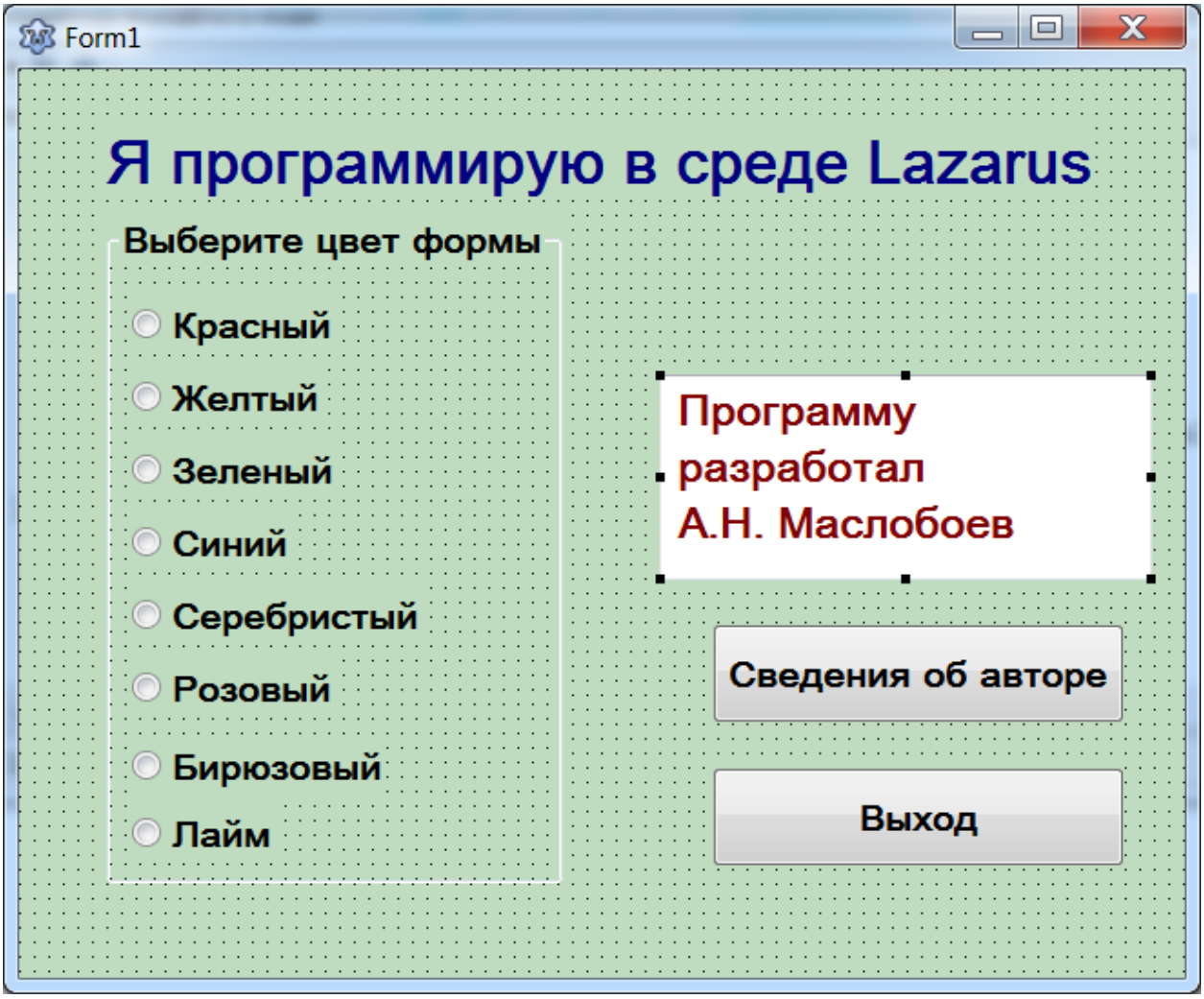

Рис. 1.12. Внешний вид пользовательского интерфейса приложения на стадии его разработки

Теперь переходим к программированию переключателей. Также дважды щелкаем мышью переключатель **RadioButton1** и получаем заготовку процедуры с заголовком и ключевыми словами **begin** и **end**. Между ними вписываем команду, которая меняет цвет формы на красный. Эта команда выглядит так:

```
Form1.Color:=clRed;
```

Рассмотрим эту команду более подробно. Она начинается с указания объекта, над которым выполняется действие. Это основная экранная форма **Form1**. Затем указывается название свойства, которое изменяется – **Color**. Имя объекта от имени его свойства отделяется точкой. Затем идет операция присваивания (двоеточие со знаком равенства). После знака операции указывается присваиваемое значение, т.е. в данном случае цвет. Напоминаем, что перед названием цвета в Object Pascal всегда ставится приставка Cl (сокращение от Color). В итоге процедура для первого переключателя будет записана так:

```
procedure TForm1.RadioButton1Change(Sender: TObject);
    begin
      Form1.Color:=clRed;
    end;
```

Аналогично будет выглядеть процедура для второго переключателя (только в ней название цвета поменяется с красного на желтый).

```
procedure TForm1.RadioButton2Change(Sender: TObject);
    begin
      Form1.Color:=clYellow;
    end;
```

Такие же процедуры необходимо написать и для остальных шести переключателей, не забывая при этом каждый раз менять название цвета.

Теперь осталось запрограммировать кнопку «**Сведения об авторе**». В отличие от предыдущих объектов, эта кнопка должна реагировать на два события. Щелчок левой кнопкой мыши должен выводить на экран окно со сведениями об авторе, а щелчок правой кнопкой скрывать эту информацию. Поэтому для кнопки **Button1** необходимо написать две процедуры. Первая создается таким же образом, как и все предыдущие, т.е. заготовка создается двойным щелчком мыши на объекте, а затем пишется код следующего вида:

```
Memo1.Visible:=True;
```

Этот оператор присваивает свойству **Visible** объекта **Memo1** значение **True**, т.е. делает данный объект видимым. Целиком соответствующая процедура выглядит так:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Memo1.Visible:=True;
end;
```

Для создания следующей процедуры необходимо будет обратиться к Инспектору объектов и активизировать в этом окне вкладку **События**. На вкладке следует найти событие **OnContexPopup** (это вызов контекстного меню правой клавишей мыши) и щелкнуть мышью на кнопке-построителе, расположенной справа от названия этого события. В результате создается заготовка для еще одной процедуры. В тело этой процедуры следует вписать оператор следующего вида:

```
Memo1.Visible:=False;
```

Этот оператор делает текстовое поле со сведениями об авторе снова невидимым. Полностью процедура выглядит так:

```
procedure TForm1.Button1ContextPopup(Sender:
TObject; MousePos: TPoint;  var Handled: Boolean);
begin
  Memo1.Visible:=False;
end;
```

Таким образом, написание программного кода для данного приложения можно считать завершенным. Ниже приводится полная запись кода, которая выглядит следующим образом:

```
procedure TForm1.Button2Click(Sender: TObject);
    begin
     Close;
    end;
procedure TForm1.Button1Click(Sender: TObject);
    begin
      Memo1.Visible:=True;
    end;
procedure  TForm1.Button1ContextPopup(Sender:  TObject;
MousePos: TPoint;
      var Handled: Boolean);
    begin
      Memo1.Visible:=False;
    end;

procedure TForm1.RadioButton1Change(Sender: TObject);
    begin
      Form1.Color:=clRed;
    end;
```

```
procedure TForm1.RadioButton2Change(Sender: TObject);
    begin
      Form1.Color:=clYellow;
    end;

procedure TForm1.RadioButton3Change(Sender: TObject);
    begin
      Form1.Color:=clGreen;
    end;

procedure TForm1.RadioButton4Change(Sender: TObject);
    begin
      Form1.Color:=clBlue;
    end;

procedure TForm1.RadioButton5Change(Sender: TObject);
    begin
      Form1.Color:=clSilver;
    end;

procedure TForm1.RadioButton6Change(Sender: TObject);
    begin
      Form1.Color:=clFuchsia;
    end;

procedure TForm1.RadioButton7Change(Sender: TObject);
    begin
      Form1.Color:=clAqua;
    end;

procedure TForm1.RadioButton8Change(Sender: TObject);
    begin
      Form1.Color:=clLime;
    end;
```

Теперь приложение можно запустить на выполнение. Для этого следует нажать экранную кнопку  , которая расположена на **Панели инструментов** в левом верхнем углу экрана, или нажать клавишу **F9**. Начинается процесс компиляции, который отображается в специальном диалоговом окне. Если в процессе компиляции никаких ошибок не обнаружено и все прошло успешно, то выводится соответствующее сообщение, показанное на рис. 1.13. При обнаружении же ошибок в программном коде следует их исправить и повторно произвести компиляцию, нажав ту же кнопку. При этом следует не забывать периодически сохранять на жесткий диск компьютера изменения, сделанные в программе.

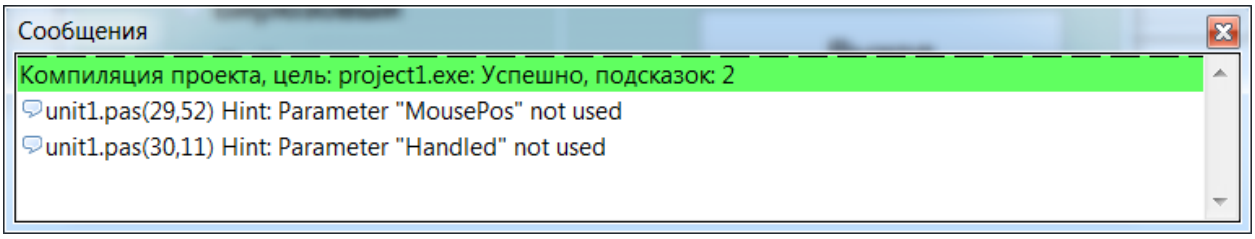

Рис. 1.13. Диалоговое окно с сообщением об успешной компиляции проекта

После благополучного выполнения компиляции на экране появляется окно приложения в рабочем режиме, о чем можно судить по отсутствию пунктирной сетки (см. рис. 1.14). Следует проверить все элементы интерфейса созданного приложения на предмет их работоспособности, после чего работа над данным проектом может быть завершена.

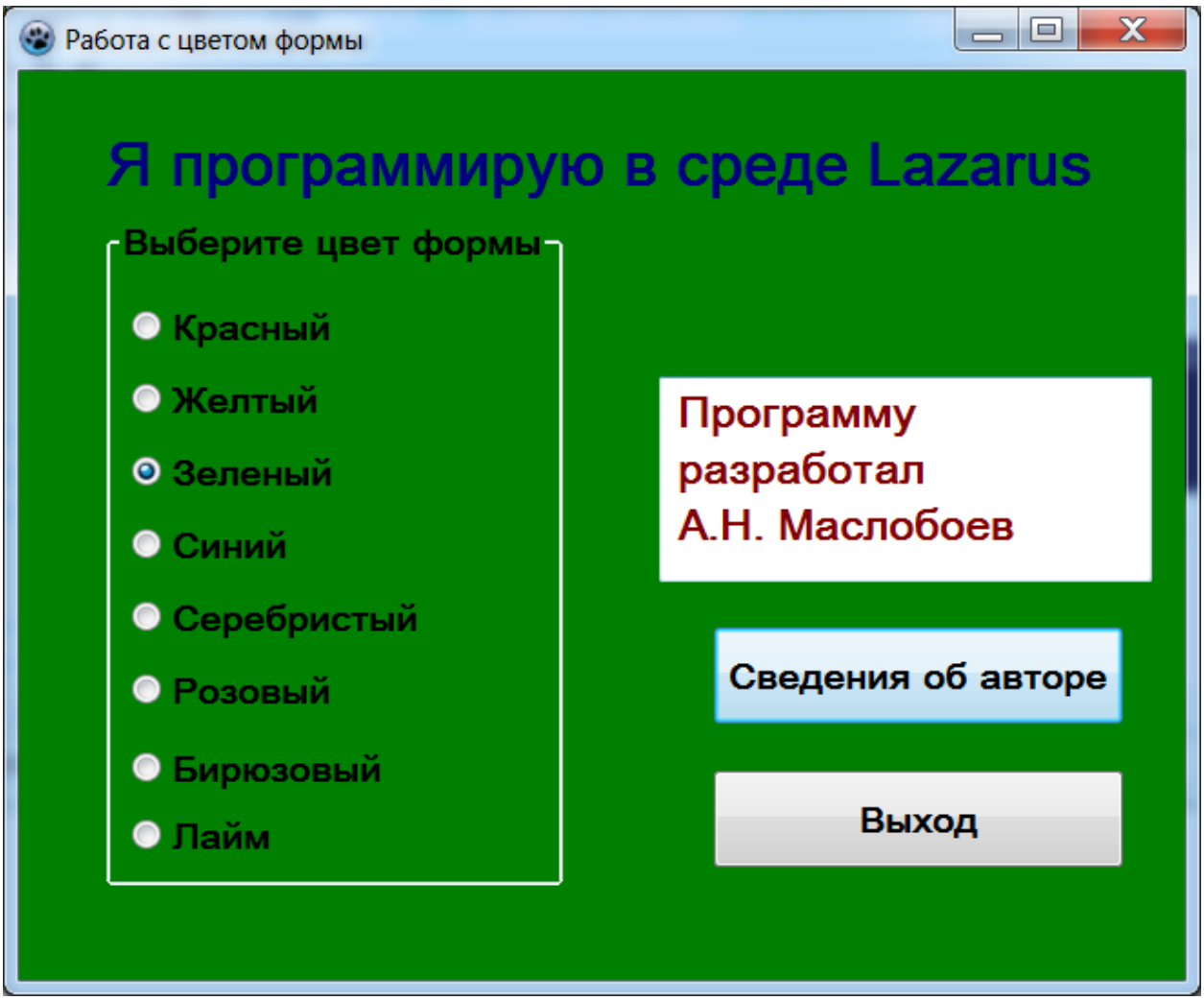

Рис. 1.14. Приложение «Colors» в рабочем режиме. Фоновый цвет формы – зеленый. Окно со сведениями об авторе выведено на экран.

Контрольные вопросы

1. Какой файл запускает среду Lazarus на выполнение?

2. Какая экранная кнопка позволяет открыть основной экран среды Lazarus?

3. Где размещаются создаваемые разработчиком элементы пользовательского интерфейса приложения?

4. Перечислите известные Вам элементы пользовательского интерфейса приложения.

5. Для чего используется пунктирная сетка, расположенная на экранной форме?

6. Какие основные вкладки имеются в инспекторе объектов среды Lazarus?

7. Какое свойство позволяет изменить характеристики шрифта, которым выполнена надпись на объекте?

8. Как называется кнопка, открывающая дополнительное диалоговое окно?

9. Какая приставка обязательно должна быть указана перед названием цвета в Object Pascal?

10. Как записывается в языке Object Pascal операция присваивания?

11. Как называется в инспекторе объектов Lazarus событие, соответствующее щелчку левой клавиши мыши на объекте?

12. Как называется в инспекторе объектов Lazarus событие, соответствующее щелчку правой клавиши мыши на объекте?

Глава 2. Линейные программы вычислительного характера

В данной главе рассматривается создание в среде Lazarus приложений, которые могут выполнять элементарные вычислительные операции. На примере этих приложений мы разберемся, как осуществлять в языке Object Pascal различные вычисления, а также производить преобразование данных разных типов.

2.1. Вычисления с использованием целочисленных переменных

В данном случае нашей целью будет создание программы, которая по заданной длине и ширине вычисляет площадь прямоугольника. Как обычно, создание программы начинается с того, что в диалоговом окне выбирается тип программы – приложение (см. главу 1). На экране появляется новая свободная форма. Затем создается новая папка **Rectangle**, в которой необходимо сохранить основные файлы разрабатываемого приложения – **project1.lpi** и **unit1.pas**.

Следующим шагом является создание пользовательского интерфейса приложения. С помощью свойства **Caption** изменяем надпись в строке заголовка основной экранной формы: вместо «**Form1**» там должно быть написано «Вычисление площади прямоугольника».

Затем на экранной форме создаются следующие элементы пользовательского интерфейса: три текстовых окна для ввода исходных данных и вывода результата; три поясняющих надписи к текстовым окнам; четыре экранные кнопки: «**Расчет**», «**Сброс**», «**Выход**», «**Сведения об авторе**»; многострочное текстовое окно типа **Memo**.

Поскольку создание надписей, экранных кнопок и окна типа «Memo» было рассмотрено в материалах предыдущей главы, мы здесь более подробно остановимся только на создании текстовых окон. В данном проекте в первое текстовое окно будет вводиться информация о длине прямоугольника, во втором – о его ширине, в третьем будут выводиться данные о площади прямоугольника, полученные в результате вычислений.

Заготовка для текстового окна находится на палитре компонентов **Standard** и имеет вид белого прямоугольника, в котором содержатся две строчные латинские буквы **a** и **b**, а также вертикальный текстовый курсор:  . Если подвести к этой заготовке курсор, то появится всплывающая подсказка, содержащая английское название данного компонента – **Edit**.

После щелчка на экранной форме появляется заготовка для первого текстового окна с именем **Edit1**. Далее производим начальную настройку свойств данного объекта. Для этого в Инспекторе объектов настраиваем

свойство **Font**. После щелчка на кнопке-построителе справа от названия объекта появляется уже знакомое нам диалоговое окно для настройки свойств шрифта объекта. Выбираем гарнитуру шрифта **Arial**, полужирное начертание и размер шрифта – 16 пунктов.

В текстовом окне по умолчанию отображается текст **Edit1**, но на момент начала работы программы окно должно быть чистым, чтобы в него можно было вводить исходные данные (в данном случае – это длина прямоугольника). За содержание текстового окна отвечает свойство **Text**. Находим в окне **Инспектора объектов** данное свойство, выделяем текст, находящийся в поле справа от названия свойства, и нажимаем клавишу **Delete**. Теперь окно будет свободным для ввода новой информации.

Далее аналогичным образом создаем и настраиваем текстовые окна **Edit2** для ввода ширины прямоугольника и **Edit3** для вывода его площади. Окно **Edit2** располагаем под окном **Edit1**, а **Edit3** под **Edit2**. Между **Edit2** и **Edit3** следует оставить интервал, для того чтобы в нем можно было поместить экранную кнопку «**Расчет**».

Слева от каждого из окон следует поместить поясняющую надпись, которая объясняет пользователю, какую именно информацию следует вводить в расположенное рядом с ней окно или какая информация будет выведена в соответствующем окне.

Так рядом с окном **Edit1** будет находиться надпись **Label1**. С помощью инспектора объектов меняем содержание свойства **Caption** для **Label1** на следующее: «Введите длину прямоугольника». Для надписи **Label2** свойство **Caption** меняем на «Введите ширину прямоугольника». Для надписи **Label3** в свойстве **Caption** следует прописать «Площадь прямоугольника равна».

Под окном **Edit3** располагаем экранную кнопку «**Сброс**», под ней экранные кнопки «**Об авторе**» и «**Выход**», знакомые нам по материалам, изложенным в главе 1. В правой нижней части окна располагаем объект **Memo1**. На этом работу над интерфейсом приложения можно считать завершенной. Внешний вид окна с пользовательским интерфейсом данного приложения приведен на рис. 2.1.

Следующим этапом в создании приложения является написание программного кода для всех четырех экранных кнопок, имеющихся на экранной форме. Начнем с кнопки «**Расчет**» (имя кнопки – **Button1**). Нажатие на эту кнопку позволяет получить на основании значений, введенных пользователем в текстовые окна **Edit1** и **Edit2**, величину площади прямоугольника, которая выводится в окно **Edit3**.

Заготовка соответствующей процедуры создается, как обычно, двойным щелчком мыши на кнопке **Button1**. Данная процедура отличается от тех, которые были рассмотрены в предыдущей главе, тем, что в ней

выполняются определенные вычисления. Для выполнения вычислений нам понадобятся переменные. По правилам языка Object Pascal все переменные, используемые в процедуре, должны быть предварительно описаны. Описание процедуры располагается между ее заголовком и телом процедуры, начинающимся с ключевого слова **Begin**.

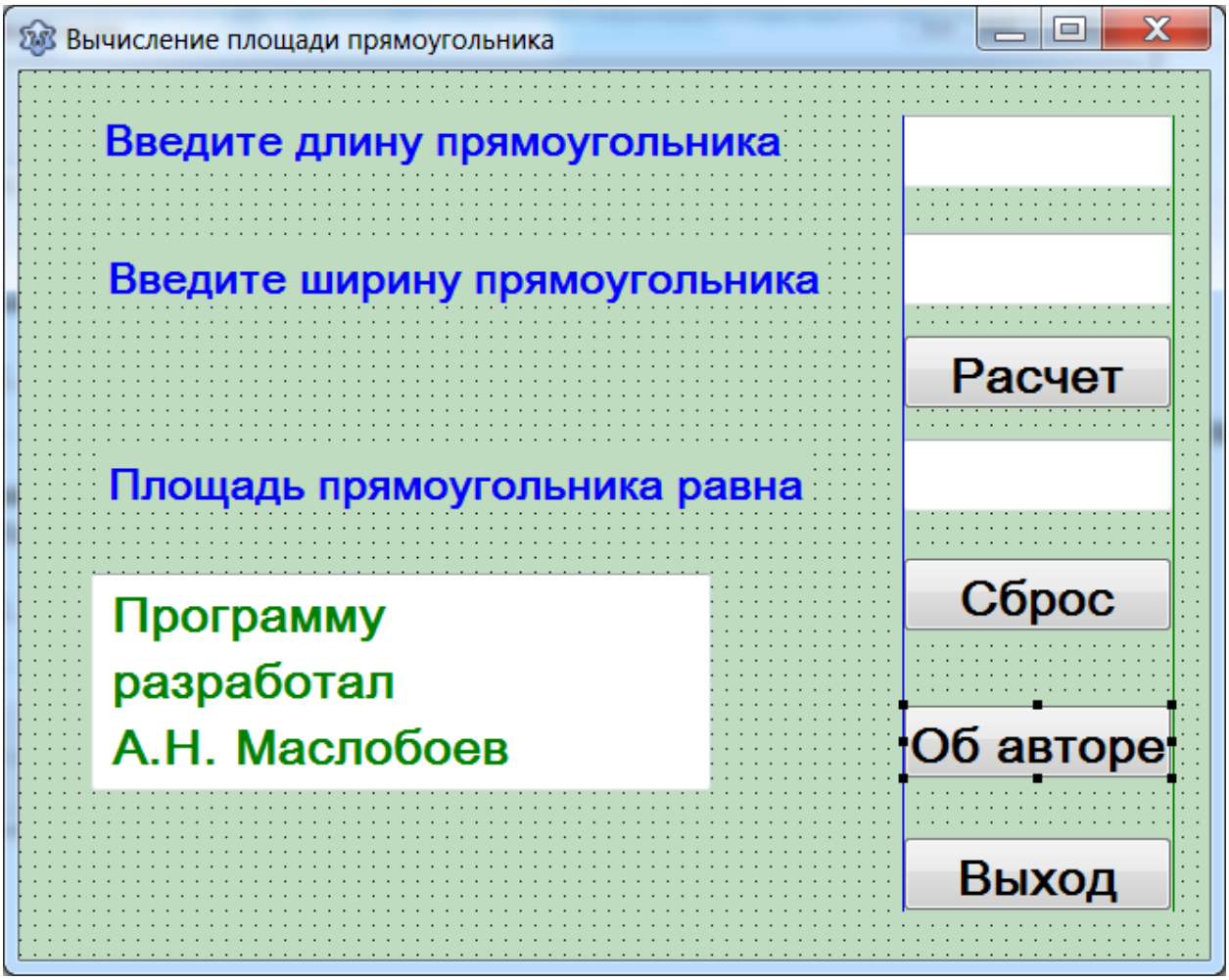

Рис. 2.1. Пользовательский интерфейс приложения «Вычисление площади прямоугольника»

В нашем проекте описание переменных в процедуре будет выглядеть следующим образом:

```
var a,b,s:integer;
```

где **var** – служебное слово, открывающее раздел описания переменных;

a,b,s – имена переменных;
integer – служебное слово, обозначающее тип переменных.

Рассмотрим данное описание более подробно. Раздел описания переменных всегда должен начинаться со служебного слова **var**. За ним после пробела указываются имена переменных, для обозначения которых могут использоваться латинские буквы и цифры, причем первым символом всегда должна быть буква. Для каждой переменной должно быть указано не только ее имя, но и ее тип. Имя переменной от названия ее типа отделяется двоеточием. Если в процедуре есть несколько переменных, относящихся к одному и тому же типу (как в нашем примере), то можно перечислить их через запятую, а затем после двоеточия указать общий для них тип.

В данной процедуре используются три переменных: **a** – длина прямоугольника, **b** – ширина прямоугольника, **s** – площадь прямоугольника. Все эти переменные относятся к одному и тому же целому типу, который в языке Object Pascal обозначается служебным словом **integer**.

Далее в теле процедуры нам необходимо для дальнейших вычислений получить ширину и длину прямоугольника в целочисленном виде. Но информация, которая вводится в текстовые окна, относится не к целочисленному, а к строковому типу. Поэтому данные, находящиеся в текстовых окнах **Edit1** и **Edit2,** следует вначале преобразовать в целые числа. Для этой цели в языке Object Pascal существует специальная функция **StrToInt**. Аргументом данной функции является строковая величина, а значением величина целочисленная. Аргумент должен быть заключен в круглые скобки и указан после имени функции. Затем значение функции присваивается целочисленной переменной. В процедуре эти действия записываются с помощью двух операторов присваивания.

```
a:=StrToInt(Edit1.Text);
b:=StrToInt(Edit2.Text);
```

Следует обратить внимание, что в таких операторах слева указывается имя переменной, которой присваивается новое значение (в нашем случае – это переменные **a** и **b**). Затем после имени переменной ставится знак операции присваивания – двоеточие со знаком равенства. После операции присваивания в правой части оператора указывается присваиваемое значение – результат выполнения функции преобразования. Для переменной **a** аргумент функции **StrToInt** берется из свойства **Text** первого текстового окна, для **b** – из аналогичного свойства второго окна. При этом сначала указывается имя объекта, а затем имя свойства, которое отделяется от имени объекта точкой.

Далее полученные значения переменных **a** и **b** можно использовать в следующем операторе присваивания. В этом операторе данные значения перемножаются и результат присваивается переменной s. Умножение в Object Pascal всегда обозначается символом «звездочка» и пропускать этот

символ между сомножителями нельзя. Данный оператор в процедуре выглядит следующим образом:

```
s:=a*b;
```

Затем осталось полученные значение преобразовать из целочисленной формы в строковую и вывести полученный результат в третье текстовое окно. Для обратного преобразования используется функция **IntToStr**, а сам оператор выглядит следующим образом:

```
Edit3.Text:=IntToStr(s);
```

Теперь работа над процедурой, описывающей работу кнопки «Расчет», завершена. Полный текст процедуры приводится ниже:

```
procedure TForm1.Button1Click(Sender: TObject);
var a,b,s:integer;
begin
  a:=StrToInt(Edit1.Text);
  b:=StrToInt(Edit2.Text);
  s:=a*b;
  Edit3.Text:=IntToStr(s);
end;
```

Кнопка «**Сброс**» позволяет очистить все три текстовых окна одним щелчком мыши в том случае, если нужно повторно рассчитать площадь прямоугольника по новым исходным данным. Очистка производится путем присваивания свойству **Text** каждого из текстовых окон «пустого» значения. Это значение представляет собой пробел, заключенный в апострофы, а текст данной процедуры будет следующим:

```
procedure TForm1.Button2Click(Sender: TObject);
begin
  Edit1.Text:=' ';
  Edit2.Text:=' ';
  Edit3.Text:=' ';
end;
```

Процедуры для экранных кнопок «**Выход**» и «**Сведения об авторе**» записываются аналогично тому, как это было сделано в предыдущем проекте. Ниже приводится полный код программного модуля, созданного для данного приложения:

```
unit Unit1;
{$mode objfpc}{$H+}
interface
 Classes, SysUtils, FileUtil, Forms, Controls,
Graphics, Dialogs, StdCtrls, Types;
type
 { TForm1 }
  TForm1 = class(TForm)
    Button1: TButton;
    Button2: TButton;
    Button3: TButton;
    Button4: TButton;
    Edit1: TEdit;
    Edit2: TEdit;
    Edit3: TEdit;
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    Memo1: TMemo;
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure Button3Click(Sender: TObject);
    procedure  Button3ContextPopup(Sender:  TObject;
MousePos: TPoint;
      var Handled: Boolean);
    procedure Button4Click(Sender: TObject);
  private
    { private declarations }
  public
    { public declarations }
  end;
var
  Form1: TForm1;
```

```
implementation
{$R *.lfm}
{ TForm1 }
procedure TForm1.Button4Click(Sender: TObject);
begin
  Close;
end;
procedure TForm1.Button3Click(Sender: TObject);
begin
  Memo1.Visible:=True;
end;
procedure TForm1.Button1Click(Sender: TObject);
var a,b,s:integer;
begin
  a:=StrToInt(Edit1.Text);
  b:=StrToInt(Edit2.Text);
  s:=a*b;
  Edit3.Text:=IntToStr(s);
end;
procedure TForm1.Button2Click(Sender: TObject);
begin
  Edit1.Text:=' ';
  Edit2.Text:=' ';
  Edit3.Text:=' ';
end;
procedure         TForm1.Button3ContextPopup(Sender:
TObject; MousePos: TPoint;
  var Handled: Boolean);
begin
  Memo1.Visible:=False;
end;
end.
```

После завершения работы над программным кодом следует запустить приложение на выполнение с помощью экранной кнопки аналогично тому, как это было описано в главе 1, ввести данные в текстовые окна и проверить работу экранной кнопки «Расчет» и прочих элементов пользовательского интерфейса приложения.

На рис. 2-2 показан внешний вид данного приложения в процессе его выполнения.

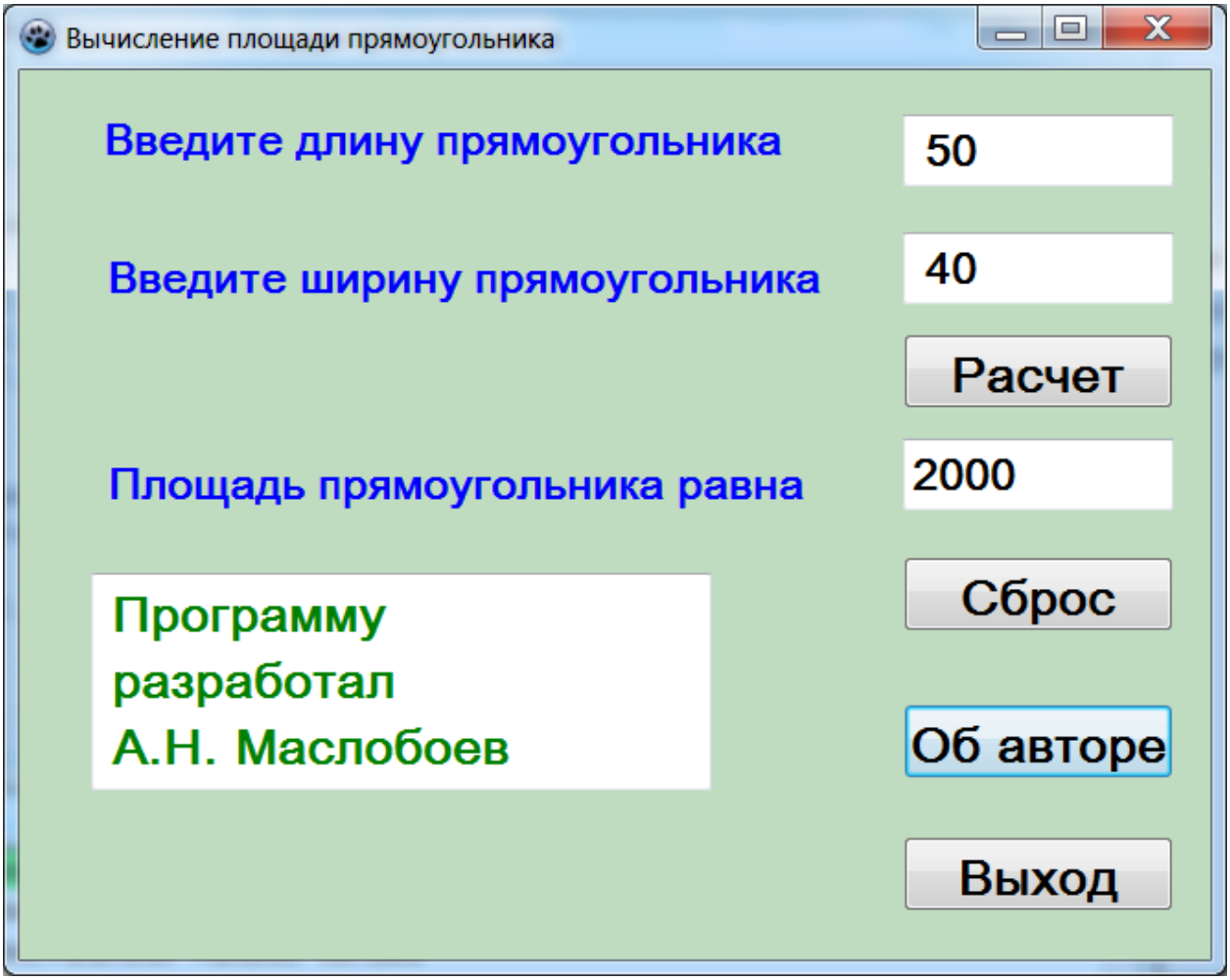

Рис. 2.2. Приложение «Вычисление площади прямоугольника» в рабочем режиме

2.2. Специальные операции для работы с целочисленными величинами

В языке Object Pascal, помимо четырех основных арифметических операций, (сложения, вычитания, умножения и деления), существуют еще две дополнительные арифметические операции, которые применяются только по отношению к целочисленным величинам.

Первая из этих операций – целочисленное деление. Алгоритм выполнения этой операции следующий. Вначале выполняется обычное деление. Затем берется получившееся частное и из него берется только

целая часть, а дробная отбрасывается. Оставшаяся целая часть - это результат выполнения данной операции. Для обозначения этой операции используется ключевое слово **div**.

Рассмотрим выполнение этой операции на следующем примере. Нужно найти результат операции

18 div 5

Первоначально производим обычное деление – 18/5. Полученный промежуточный результат равен 3,6. Отбрасываем 0,6 и получаем окончательный результат – число 3.

Используя аналогичный метод вычисления, получим:

22 div 4 =5

47 div 11=4

Вторая целочисленная операция обозначается ключевым словом **mod** – это нахождение остатка при целочисленном делении. Приведем примеры выполнения операции **mod**:

22 mod 4=2

47 mod 11=3

Как видно из приведенных примеров, при использовании тех же операндов, что и для операции **div**, операция **mod** дает иные результаты (операндами называются те величины, над которыми выполняются операции; в операции деления, как и в других арифметических операциях, присутствуют два операнда – делимое и делитель).

Рассмотрим использование вышеописанных операторов в следующем проекте. Необходимо разработать приложение, которое для любого трехзначного числа вычисляет сумму его цифр. Как и ранее, начинаем работу над приложением с создания его графического интерфейса.

Графический интерфейс данного приложения должен включать в себя следующие элементы: текстовое окно для ввода исходных данных, т.е. трехзначного числа, окно для вывода результата – суммы цифр этого числа, а также экранные кнопки. Кнопка «**Расчет**» необходима для выполнения необходимых вычислений. Кнопка «**Сброс**» позволяет очистить оба текстовых окна (с исходными данными и с результатом вычислений) в случае, если необходимо повторно произвести вычисления с другими исходными данными (трёхзначным числом). Кнопка «**Сведения об авторе**» делает видимым скрытое ранее многострочное окно, в котором указывается фамилия и инициалы программиста, разработавшего данное приложение. Кнопка «**Выход**» закрывает приложение после окончания его работы.

Новым элементом интерфейса, который присутствует в данном приложении, является иллюстрация, на которой изображен калькулятор. Это объект расположен в центральной части окна слева от кнопки «**Сброс**».

Для того чтобы создать иллюстрацию, необходимо в группе вкладок «Палитра компонентов» вместо открытой по умолчанию вкладки «**Standard**» щелчком мыши выбрать вкладку «**Additional**», содержащую дополнительные компоненты. На указанной вкладке нужно выбрать заготовку **TImage**. После щелчка мышью на основной экранной форме появляется штриховая рамка, обозначающая границы объекта **Image1**. Границы этого объекта можно менять произвольным образом, перетаскивая их мышью. Для вставки в созданную рамку изображения нужно в **Инспекторе объектов** найти для **Image1** свойство **Picture** (картинка). В правом поле этого свойства находится кнопка-построитель, на которой изображено многоточие. Щелчок мышью по этой кнопке открывает дополнительное диалоговое окно (рис. 2.3), в котором можно выбрать подходящее по смыслу изображение.

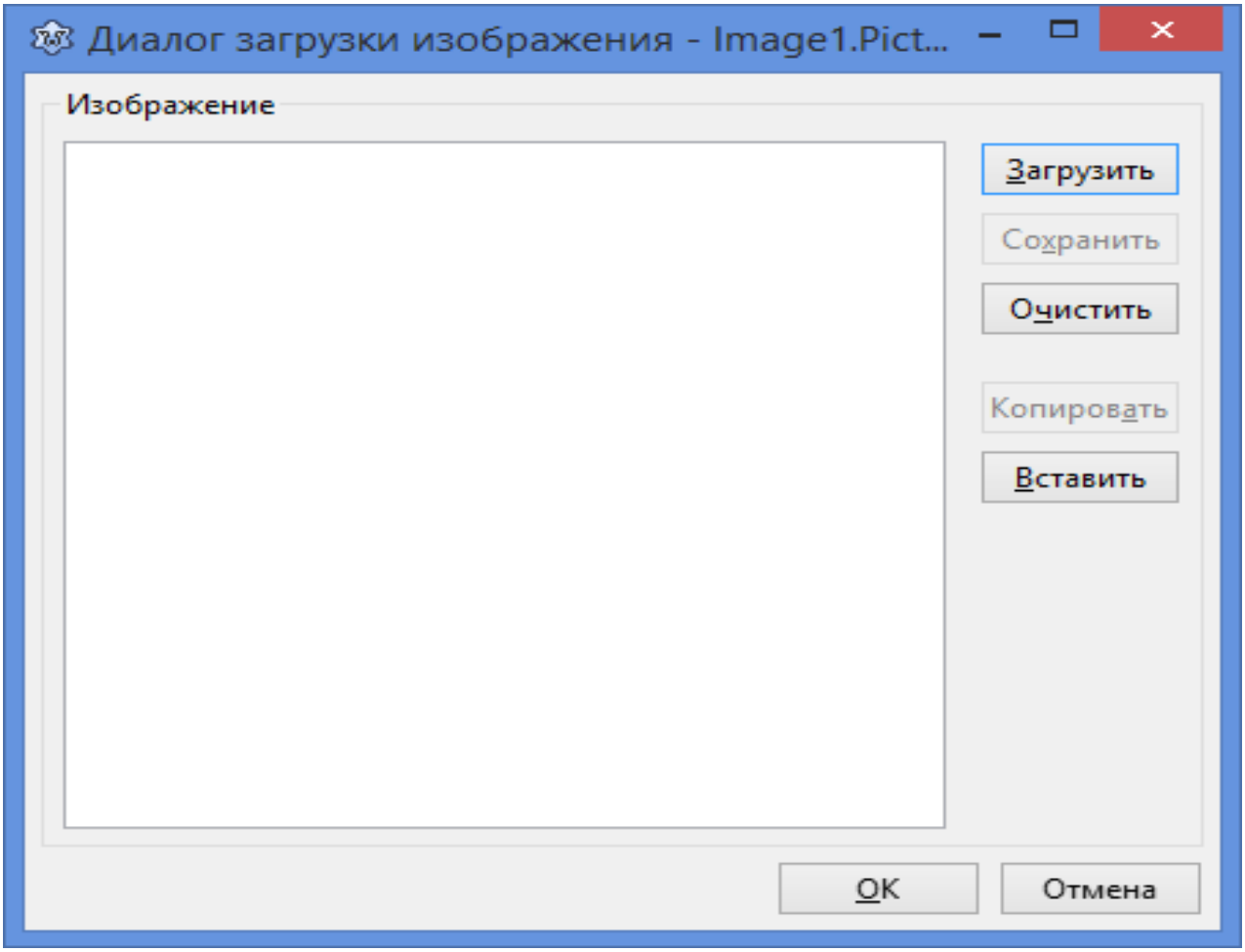

Рис. 2.3. Окно «**диалог загрузки изображения**», позволяющее выбрать изображение, которое будет загружено в объект **Image**

После щелчка на экранной кнопке «**Загрузить**» на экране компьютера появляется еще одно дополнительное диалоговое окно под названием «**Открыть файл изображения**». Это окно по своему внешнему виду аналогично окну программы «**Проводник**» операционной системы Windows. С помощью данного окна можно выбрать любое подходящее по смыслу изображение, имеющееся на компьютере разработчика. Оптимальным вариантом является размещение изображения непосредственно в папке создаваемого проекта. Файл изображения следует выделить и затем щелкнуть мышью на экранной кнопке «**Открыть**». После щелчка диалоговое окно «**Открыть файл изображения**» закрывается, а в окне «**Диалог загрузки изображения**» вместо белого поля появляется содержимое выбранного файла с изображением (см. рис. 2.4), показанное полностью или в виде фрагмента. После щелчка на экранной кнопке «Вставить» изображение помещается на основной экранной форме в пределах, установленных штриховой рамкой. Затем диалоговое окно следует закрыть щелчком на экранной кнопке «**Ok**».

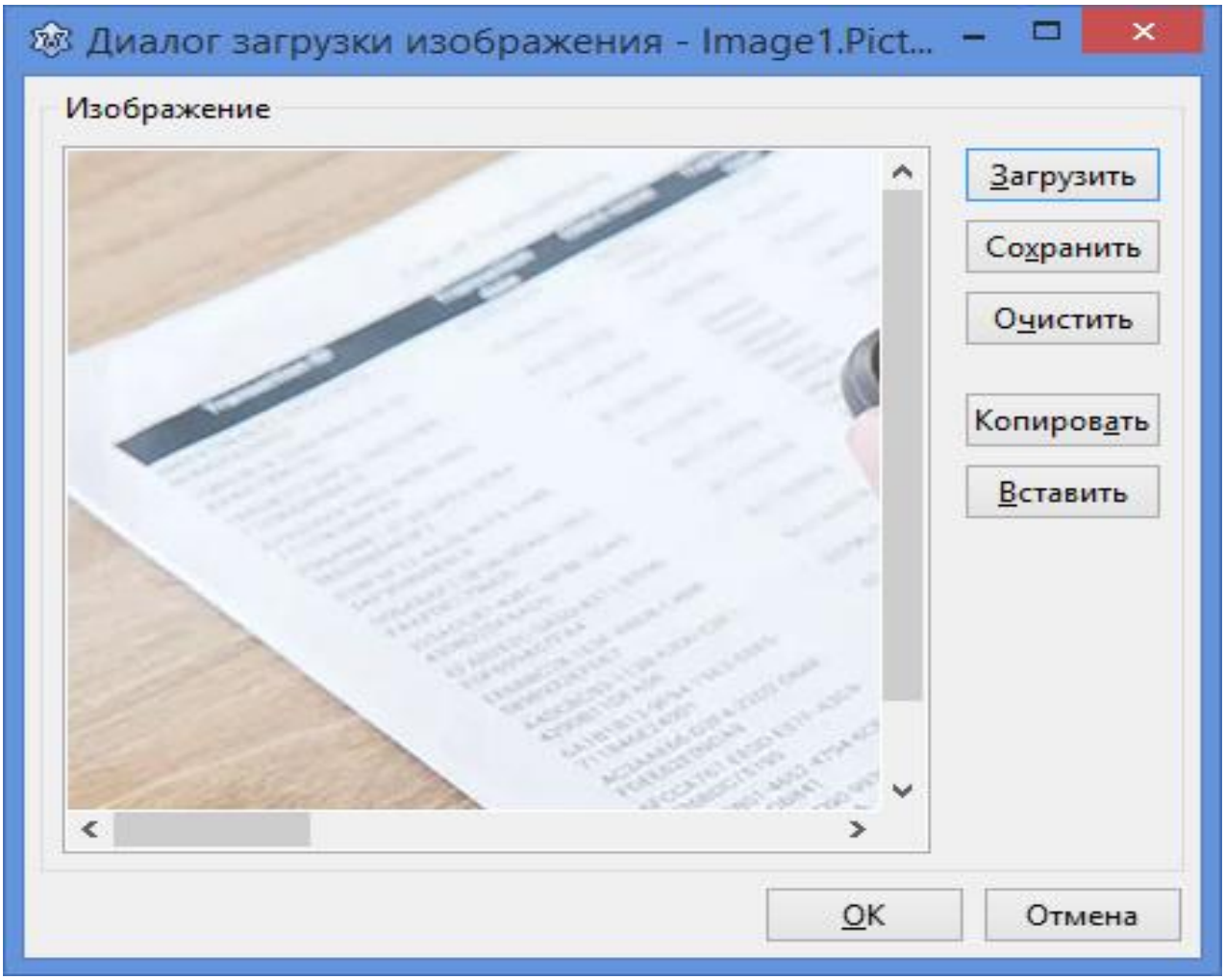

Рис. 2.4. Окно «**диалог загрузки изображения**» после выбора подходящего файла с изображением

Но при просмотре экранной формы видно, что изображение в рамке отображается не полностью, а только частично. Причиной данной ситуации является то, что реальные физические размеры изображения больше, чем отведенное для него место. Для устранения этой проблемы следует в окне Инспектора объектов для **`Image1`** найти свойство **`Stretch`**. Это свойство отвечает за «подгонку» изображения к размерам отведенной для него рамки. По умолчанию данное свойство имеет значение **`False`**. После установки флажка справа от названия свойства его значение меняется на **`True`**, и изображение на форме становится видимым полностью. При этом, естественно, размеры его уменьшаются таким образом, чтобы оно полностью поместилось в рамке.

Внешний вид пользовательского интерфейса приложения, полученный после всех вышеописанных действий, показан на рис. 2.5.

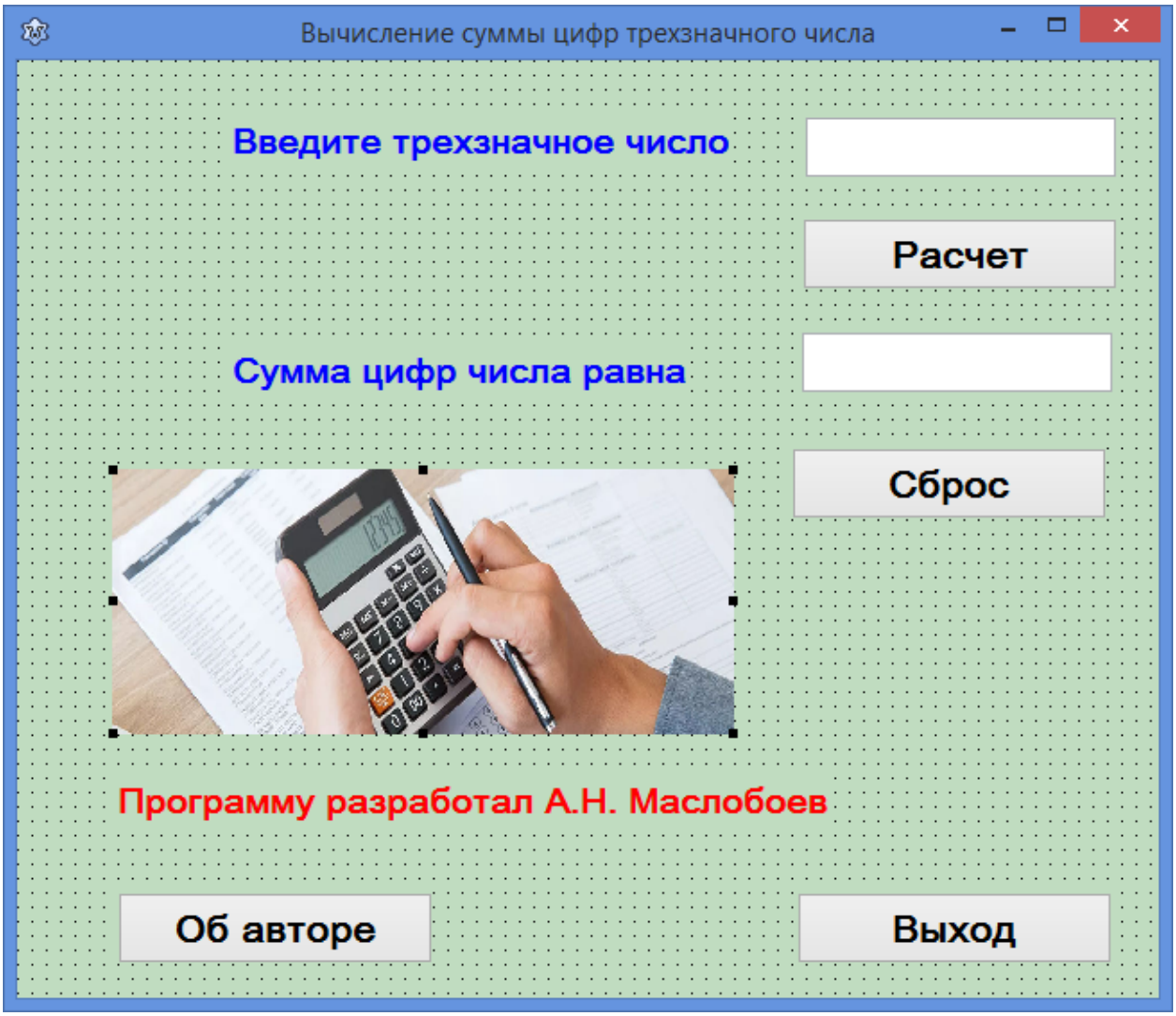

Рис. 2.5. Пользовательский интерфейс приложения «Вычисление суммы цифр трехзначного числа»

Далее переходим к написанию программного кода для данного приложения. Процедуры для экранных кнопок «**Выход**», «**Сброс**» и «**Об авторе**» пишутся так же, как и в предыдущих проектах, поэтому на них мы останавливаться не будем, а более подробно разберем программирование экранной кнопки «**Расчет**».

В начале данной процедуры после ее заголовка следует описать используемые в ней переменные. Раздел описания выглядит таким образом:

```
var n,sum,r1,r2,r3,t:integer;
```

Переменная **n** должна содержать вводимое трехзначное число. В переменной **sum** хранится искомый результат – сумма цифр введенного числа. Переменные **r1**, **r2** и **r3** – это соответственно старший, средний и младший разряды введенного числа, т.е. количество сотен, десятков и единиц в нем; **t** – вспомогательная переменная. Все указанные переменные относятся к целому типу.

Значение переменной **n** определяется с помощью уже знакомой нам стандартной функции **StrToInt**. Затем над этой величиной выполняется операция целочисленного деления на 100:

```
r1:=n div 100;
```

в результате получаем значение r1 – количество сотен в числе. Затем с помощью операции mod находим остаток от деления исходной величины на 100 и помещаем его в переменную t. Это будет суммарное количество десятков и единиц в числе:

```
t:=n mod 100;
```

Затем производим целочисленное деление переменной t на 10 и находим тем самым r2 – количество десятков в числе:

```
r2:=t div 10;
```

Наконец, найдя остаток от деления t на 10, мы определим r3 - количество единиц в исходном числе:

```
r3:=t mod 10;
```

Теперь осталось сложить все найденные значения разрядов числа и найти искомое значение переменной **sum**.

Полный текст процедуры Button1Click, соответствующей кнопке «Расчет», выглядит следующим образом:

```
procedure TForm1.Button1Click(Sender: TObject);
var n,sum,r1,r2,r3,t:integer;
begin
 n:=StrToInt(Edit1.Text);
 r1:=n div 100;
 t:=n mod 100;
```

```
    r2:=t div 10;
    r3:=t mod 10;
    sum:=r1+r2+r3;
    Edit2.Text:=IntToStr(sum);
  end;
```

Процедуры, которые соответствуют экранным кнопкам «Сброс», «Выход» и «Сведения об авторе», мы уже создавали при работе над приложением «Вычисление площади прямоугольника», поэтому подробно останавливаться на их создании нет необходимости.

Ниже приводится полный программный код приложения «Сумма цифр трехзначного числа»:

```
  unit Unit1;

  {$mode objfpc}{$H+}

  interface

  uses
    Classes, SysUtils, FileUtil, Forms, Controls,
Graphics, Dialogs, StdCtrls,
    ExtCtrls, Types;

  type

    { TForm1 }

    TForm1 = class(TForm)
      Button1: TButton;
      Button2: TButton;
      Button3: TButton;
      Button4: TButton;
      Edit1: TEdit;
      Edit2: TEdit;
      Image1: TImage;
      Label1: TLabel;
      Label2: TLabel;
      Label3: TLabel;
      procedure Button1Click(Sender: TObject);
      procedure Button2Click(Sender: TObject);
      procedure Button3Click(Sender: TObject);
      procedure Button4Click(Sender: TObject);
      procedure  Button4ContextPopup(Sender:  TObject;
MousePos: TPoint;
        var Handled: Boolean);
    private
```

```
    { private declarations }
  public
    { public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.lfm}

{ TForm1 }

procedure TForm1.Button3Click(Sender: TObject);
begin
  Close;
end;

procedure TForm1.Button4Click(Sender: TObject);
begin
  Label3.Visible:=True;
end;

procedure         TForm1.Button4ContextPopup(Sender:
TObject; MousePos: TPoint;
  var Handled: Boolean);
begin
  Label3.Visible:=False;
end;

procedure TForm1.Button1Click(Sender: TObject);
var n,sum,r1,r2,r3,t:integer;
begin
 n:=StrToInt(Edit1.Text);
 r1:=n div 100;
 t:=n mod 100;
 r2:=t div 10;
 r3:=t mod 10;
 sum:=r1+r2+r3;
 Edit2.Text:=IntToStr(sum);
end;
procedure TForm1.Button2Click(Sender: TObject);
begin
  Edit1.Text:=' ';
  Edit2.Text:=' ';
end;

end.
```

На рис. 2.6. показано окно приложения «Вычисление суммы цифр трехзначного числа» в процессе его выполнения

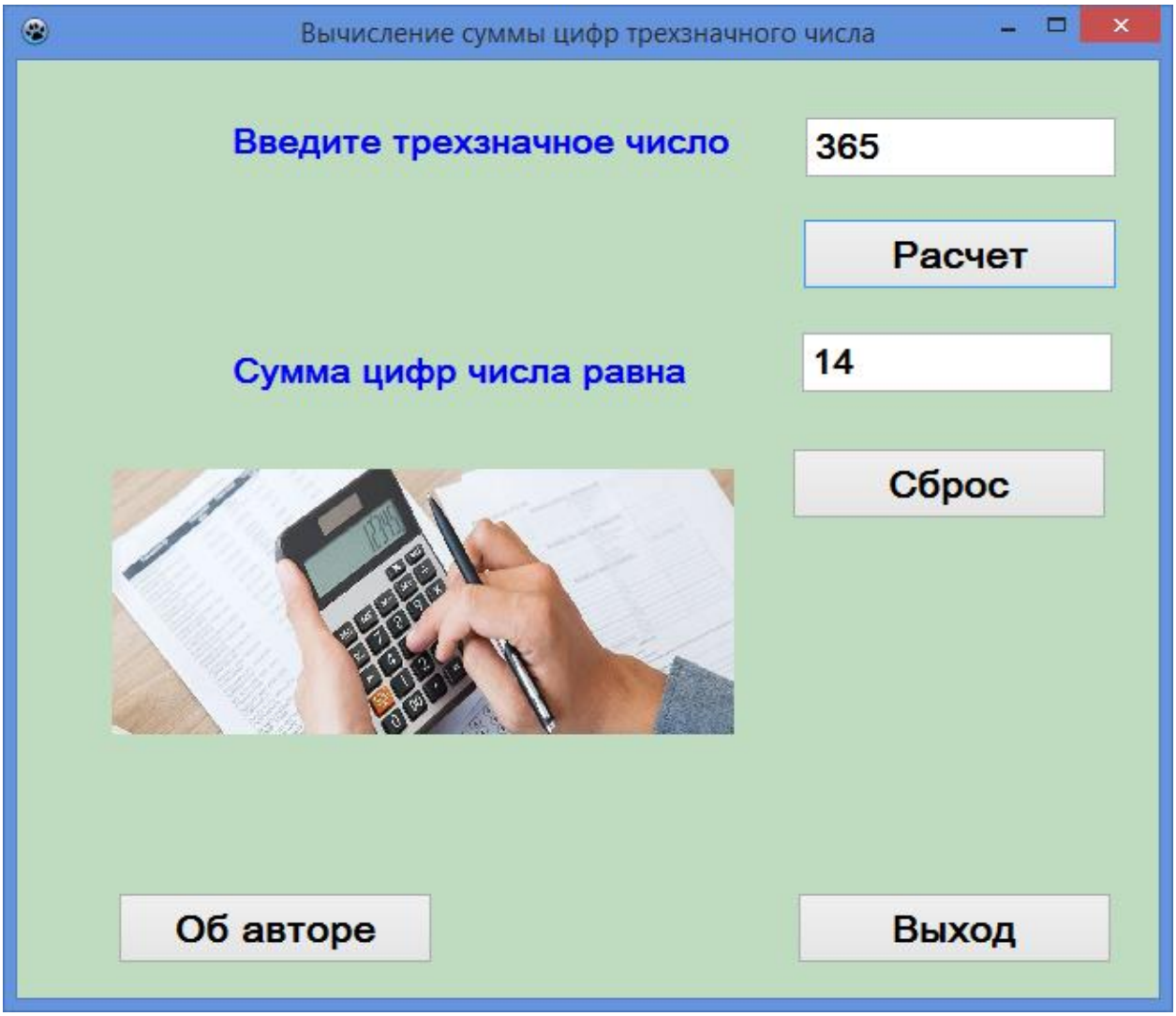

Рис. 2.6. Приложение «Вычисление суммы цифр трехзначного числа» в процессе его выполнения

2.3. Вычисления с использованием вещественных чисел

В большинстве случаев при вычислениях научно-технического характера нельзя обойтись только целыми величинами. Например, результатом операции деления в Object Pascal всегда является вещественная величина, даже если делимое и делитель относятся к целым величинам. Также вещественным всегда является результат, если хотя бы одна из величин, участвующих в операции, является вещественной. Например, вещественный результат получается при сложении целого и вещественного чисел, при умножении целого числа на вещественное, при вычитании из целого числа вещественного. Описание же вещественных переменных и

вывод вещественных величин имеет определенные особенности, которые мы рассмотрим ниже.

Во-первых, как уже отмечалось, для описания вещественных величин необходимо использовать специальный тип данных **real**. Используемые в программе целые и вещественные величины следует описывать отдельно друг от друга, хотя они и располагаются в одном разделе программы. Эти описания должны быть отделены друг от друга точкой с запятой.

Во-вторых, при выводе вещественных величин следует учитывать, что вещественное число в Object Pascal может быть представлено в двух различных видах. Эти виды называются: число с фиксированной точкой и число с плавающей точкой.

Представление числа с фиксированной точкой похоже на привычное представление вещественного числа в математике, только для отделения целой части числа от дробной используется не запятая, а точка. Этот способ представления вещественных чисел в большинстве случаев оказывается более удобным и наглядным.

Но по умолчанию Object Pascal выводит вещественные числа с плавающей точкой. В этом случае вещественное число представляется по следующей формуле:

M*10^{P},

где **M** – мантисса числа (т.е. значимая часть числа);

P – целое число со знаком, которое называется порядком.

Поскольку при работе в консольном режиме информация может выводиться на экран компьютера только в одну строку (без надстрочных и подстрочных индексов), то на экране запись числа с плавающей точкой будет выглядеть таким образом:

MEP

Буква **E** в этой записи заменяет число 10.

Поэтому, например, запись на экране компьютера **1.237E2** расшифровывается как 1,237*10^{2} или 123,7.

Если имеется необходимость преобразовать запись вещественного числа с плавающей точкой к виду «с фиксированной точкой», следует задать формат выводимого значения. Для этого в **Object Pascal** существует специальная стандартная функция **FloatToStrF**, использование которой будет рассмотрено далее в этом разделе пособия.

Рассмотрим использование вещественных переменных на примере создания приложения, которое по заданному радиусу основания вычисляет объем шара и площадь его поверхности.

Работу над приложением «**Вычисление параметров шара**» начинаем с создания пользовательского интерфейса. Интерфейс данного приложения включает следующие элементы:

- три текстовых окна – один для ввода исходных данных и два для вывода результатов;
- три поясняющие надписи к этим текстовым окнам;
- четыре экранные кнопки, используемые для расчета, сброса данных в окнах, выхода из окна и вывода сведений об авторе приложения;
- одна (скрытая) надпись, содержащая сведения об авторе.

Ввод данных производится в текстовое окно **Edit1**. Для вывода используются окна **Edit2** и **Edit3**. Пояснения к текстовым окнам содержатся в надписях **Label1**, **Label2** и **Label3**. В скрытой надписи **Label4** имеются сведения об авторе. Кнопка «Расчет» (**Button1**) находит искомые результаты по введенным в верхнее окно данным. Кнопка «Сброс» (**Button2**) позволяет очистить сразу все три текстовых окна, если нужно выполнить повторный расчет с новыми исходными данными. Кнопка «Выход» (**Button3**) закрывает приложение. Кнопка «Об авторе» (**Button4**) делает видимой надпись со сведениями об авторе (при нажатии левой кнопки мыши) или снова скрывает ее (при нажатии правой кнопки).

Пользовательский интерфейс приложения показан на рис. 2.7.

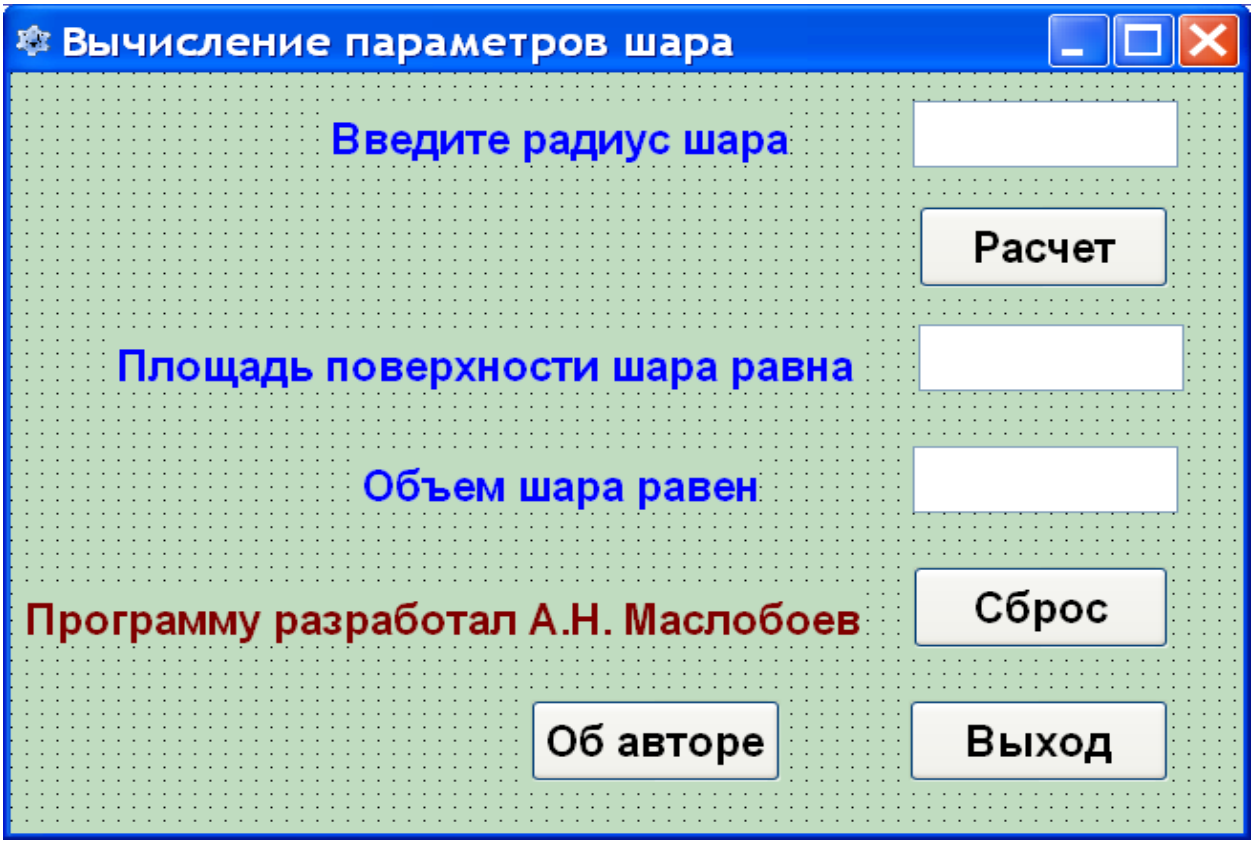

Рис. 2.7. Пользовательский интерфейс приложения «Вычисление параметров шара»

Написание программного кода начинаем с процедуры **Button1.Click** для кнопки «**Расчет**». В разделе описаний процедуры после ключевого слова **var** описываются три переменные: переменная **r** (радиус шара) является целочисленной, а переменные **s** (площадь шара) и **v** (объем шара) описываются как переменные вещественного типа. Результаты вычислений **s** и **v** могут относиться только к вещественному типу, ввиду того, что используемые при вычислениях стереометрические формулы содержат число π, которое, как известно, является бесконечной дробью. Поэтому вводим в процедуру следующую запись:

```
var r:integer;
    s,v:real;
```

Раздел операторов процедуры **Button1.Click** начинается с ввода единственного исходного данного - радиуса шара **r**. Эта величина берется из текстового окна **Edit1** и преобразуется к целочисленному виду:

```
r:=StrToInt(Edit1.Text);
```

Затем производится вычисление объема и площади шара по следующим формулам:

$V = \frac{4}{3}\pi r^3$ — объем шара;

$S = 4\pi r^2$ — площадь шара.

В программе запись может производиться только в одну строку, поэтому операция деления, присутствующая в первой формуле, обозначается символом /. Поскольку операция деления должна быть выполнена в первую очередь, она заключается в круглые скобки. Число π, присутствующее в формуле, определяется с помощью стандартной функции **pi**, имеющейся в языке Object Pascal. Так как операция возведения в степень в Object Pascal отсутствует, ее приходится заменять множественным умножением. С учетом вышеизложенного формула вычисления объема шара будет записана в программе в виде следующего оператора:

```
v:=(4/3)*pi*r*r*r;
```

Формула вычисления площади шара, записанная по аналогичным правилам, выглядит в программе так:

```
s:=4*pi*r*r;
```

Теперь полученные результаты нужно преобразовать из вещественной формы в строковую. Для этого используем стандартную функцию языка Object Pascal **FloatToStr**. Площадь поверхности выводится в текстовое окно **Edit2**, а объем в окно **Edit3**:

```
Edit2.Text:=FloatToStr(s);
Edit3.Text:=FloatToStr(v);
```

На этом работа над процедурой **Button1.Click** завершена. Полный текст этой процедуры выглядит так:

```
procedure TForm1.Button1Click(Sender: TObject);
var r:integer;
    s,v:real;
begin
     r:=StrToInt(Edit1.Text);
     s:=4*pi*r*r;
     v:=(4/3)*pi*r*r*r;
     Edit2.Text:=FloatToStr(s);
     Edit3.Text:=FloatToStr(v);
end;
```

Процедуры для кнопок «**Сброс**», «**Выход**» и «**Сведения об авторе**» записываются так же, как в предыдущих проектах, поэтому на них подробно останавливаться не будем. Ниже приводится полный текст программного модуля для данного приложения:

```
unit Unit1;

{$mode objfpc}{$H+}

interface

uses
  Classes, SysUtils, FileUtil, Forms, Controls,
Graphics, Dialogs, StdCtrls, Types;

type

  { TForm1 }

  TForm1 = class(TForm)
    Button1: TButton;
    Button2: TButton;
    Button3: TButton;
    Button4: TButton;
    Edit1: TEdit;
    Edit2: TEdit;
```

```
        Edit3: TEdit;
        Label1: TLabel;
        Label2: TLabel;
        Label3: TLabel;
        Label4: TLabel;
        procedure Button1Click(Sender: TObject);
        procedure Button2Click(Sender: TObject);
        procedure Button3Click(Sender: TObject);
        procedure Button4Click(Sender: TObject);
        procedure  Button4ContextPopup(Sender:  TObject;
MousePos: TPoint;
          var Handled: Boolean);
      private
        { private declarations }
      public
        { public declarations }
      end;
    var
      Form1: TForm1;
    implementation
    {$R *.lfm}
    { TForm1 }
    procedure TForm1.Button1Click(Sender: TObject);
    var r:integer;
        s,v:real;
    begin
         r:=StrToInt(Edit1.Text);
         s:=4*pi*r*r;
         v:=(4/3)*pi*r*r*r;
         Edit2.Text:=FloatToStr(s);
         Edit3.Text:=FloatToStr(v);
    end;
    procedure TForm1.Button2Click(Sender: TObject);
    begin
      Edit1.Text:=' ';
      Edit2.Text:=' ';
      Edit3.Text:=' ';
    end;
    procedure TForm1.Button3Click(Sender: TObject);
    begin
      Close;
    end;
```

```
    procedure TForm1.Button4Click(Sender: TObject);
    begin
      Label4.Visible:=True;
    end;

    procedure TForm1.Button4ContextPopup(Sender:
TObject; MousePos: TPoint;
      var Handled: Boolean);
    begin
      Label4.Visible:=False;
    end;

    end.
```

Результат работы данного приложения можно увидеть на рис. 2.8 .

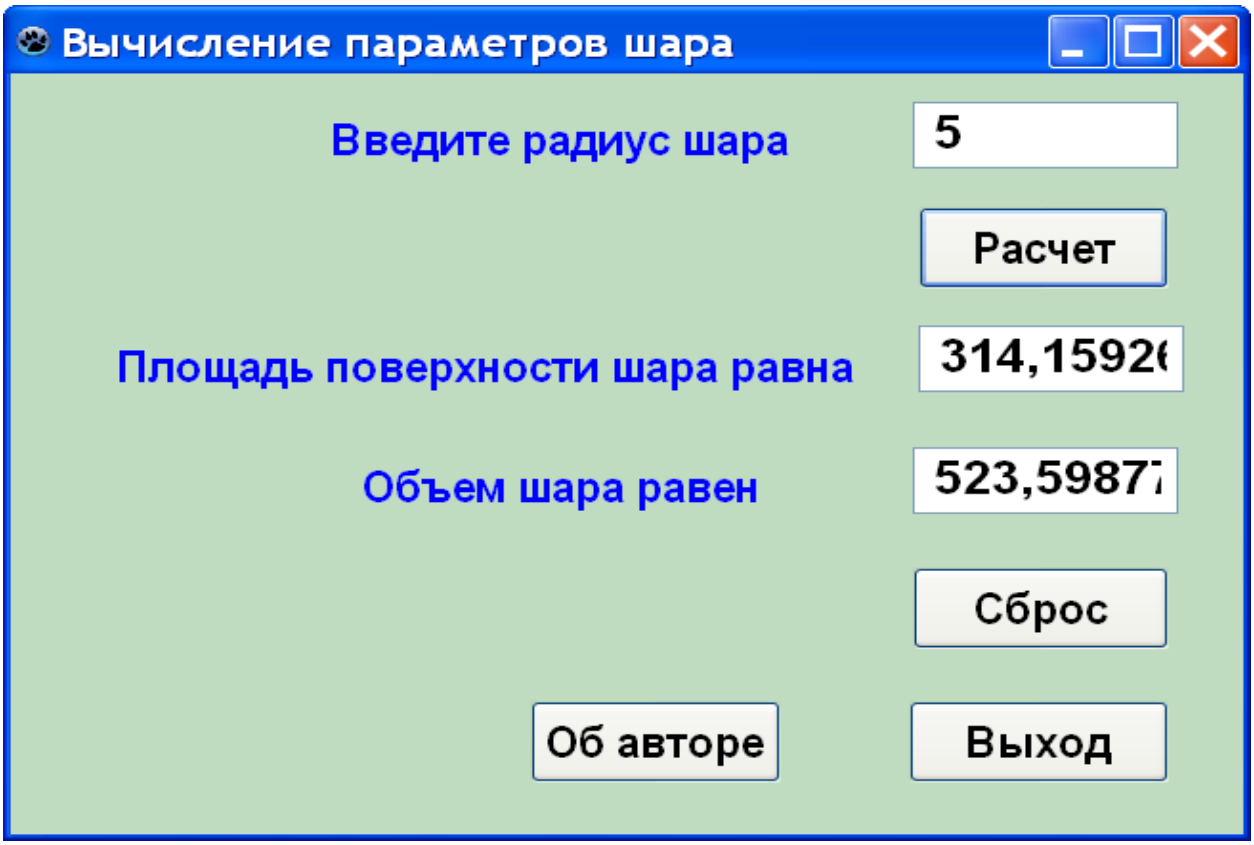

Рис. 2.8. Приложение «Вычисление параметров шара» в процессе выполнения

Из приведенного рис 2.8 видно, что вычисленные значения площади и объема шара выводятся с большим количеством цифр в дробной части. Эти значения даже полностью не помещаются в окнах **Edit2** и **Edit3**. Для того чтобы отрегулировать количество символов в выводимых результатах, в процедуре **Button1.Click** используем вместо функции **FloatToStr** другую стандартную функцию **FloatToStrF**. В этой функции в качестве параметров через запятую перечисляются:

а) имя выводимой переменной;

б) формат ее вывода (в данном случае используется формат ffFixed – с фиксированным количеством символом;

в) общее количество символов;

г) количество символов в дробной части.

Измененная процедура выглядит так:

```
procedure TForm1.Button1Click(Sender: TObject);
var r:integer;
    s,v:real;
begin
     r:=StrToInt(Edit1.Text);
     s:=4*pi*sqr(r);
     v:=(4/3)*pi*r*r*r;
     Edit2.Text:=FloatToStrF(s,ffFixed,8,3);
     Edit3.Text:=FloatToStrF(v,ffFixed,8,3);
end;
```

На рис.2.9 показаны результаты работы приложения после изменений, внесенных в вышеуказанную процедуру.

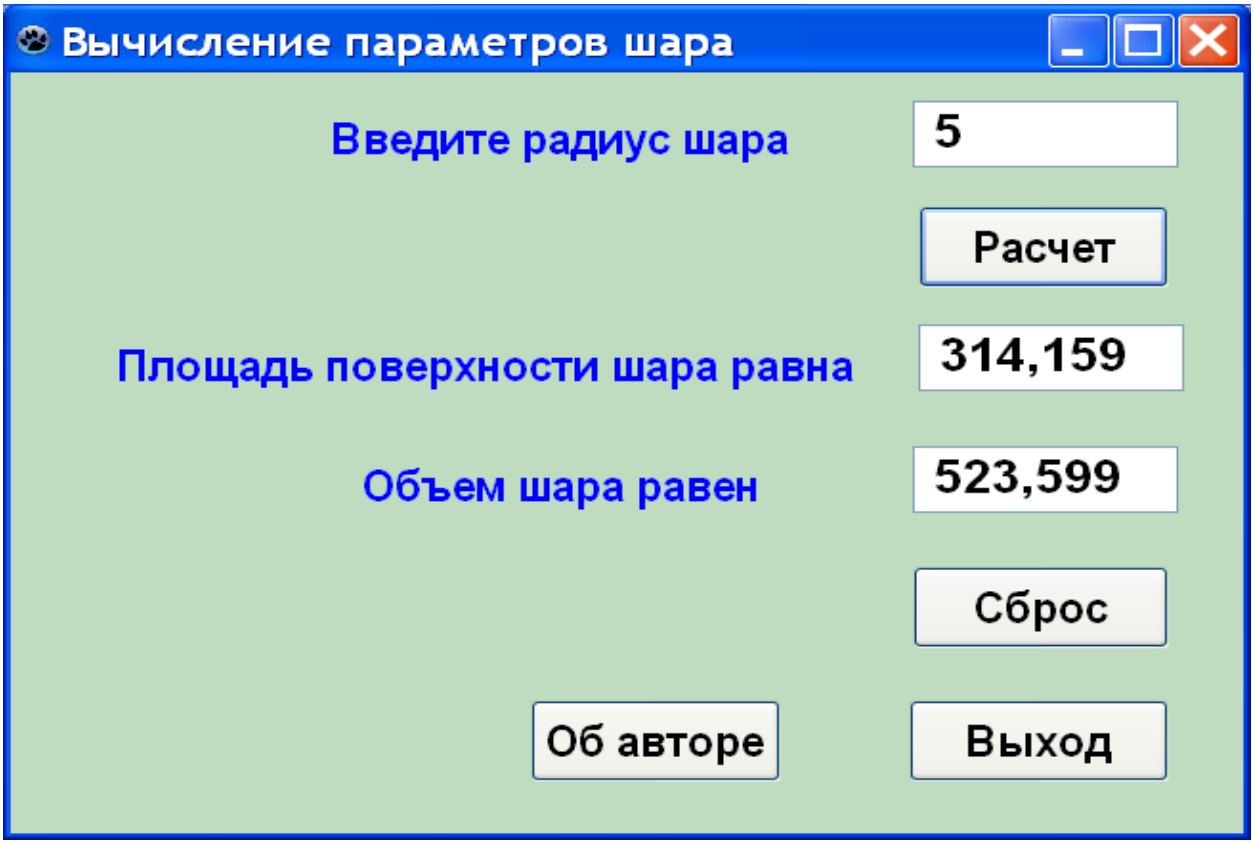

Рис 2.9. Приложение «Вычисление параметров шара» в рабочем режиме после модификации программного кода

Контрольные вопросы

1. Укажите основные правила наименования переменных в Object Pascal?

2. С какого ключевого слова начинается раздел описания переменных в Object Pascal?

3. Какое ключевое слово используется для описания переменных целого типа в Object Pascal?

4. Какое служебное слово используется для описания переменных вещественного типа в Object Pascal?

5. Какой символ в Object Pascal отделяет имя переменной от названия ее типа?

6. Можно ли в Object Pascal описать сразу группу переменных, относящихся к одному и тому же типу?

7. Каким образом в Object Pascal обозначается операция присваивания?

8. Каким ключевым словом в Object Pascal обозначается операция целочисленного деления?

9. Каким ключевым словом в Object Pascal обозначается операция нахождения остатка при делении?

10. Каким образом можно изменить приоритет при вычислении выражения в операторе присваивания?

11. Укажите название стандартной функции, которая применяется в Object Pascal для преобразования строковой величины в целочисленную.

12. Укажите название стандартной функции, которая применяется в Object Pascal для преобразования строковой величины в вещественную.

Задания для самостоятельной работы:

1. Разработать приложение, которое по длине, ширине и высоте параллелепипеда вычисляет его объем и общую площадь поверхности.

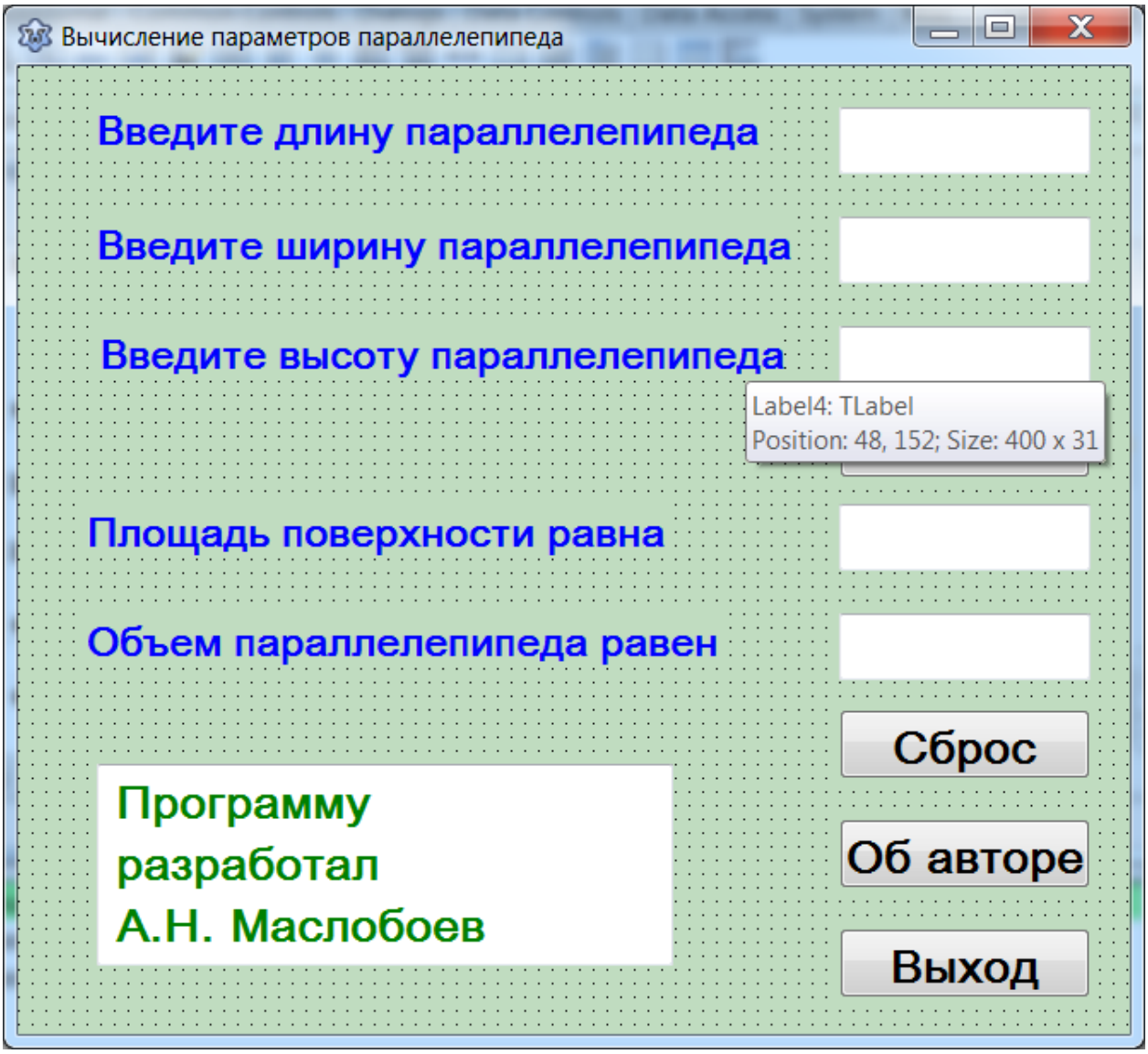

Рис. 2.10. Примерный вид интерфейса приложения «Вычисление параметров параллелепипеда»

2. Разработать приложение, которое по заданной длине, ширине и высоте цилиндра вычисляет его объем и общую площадь его поверхности.

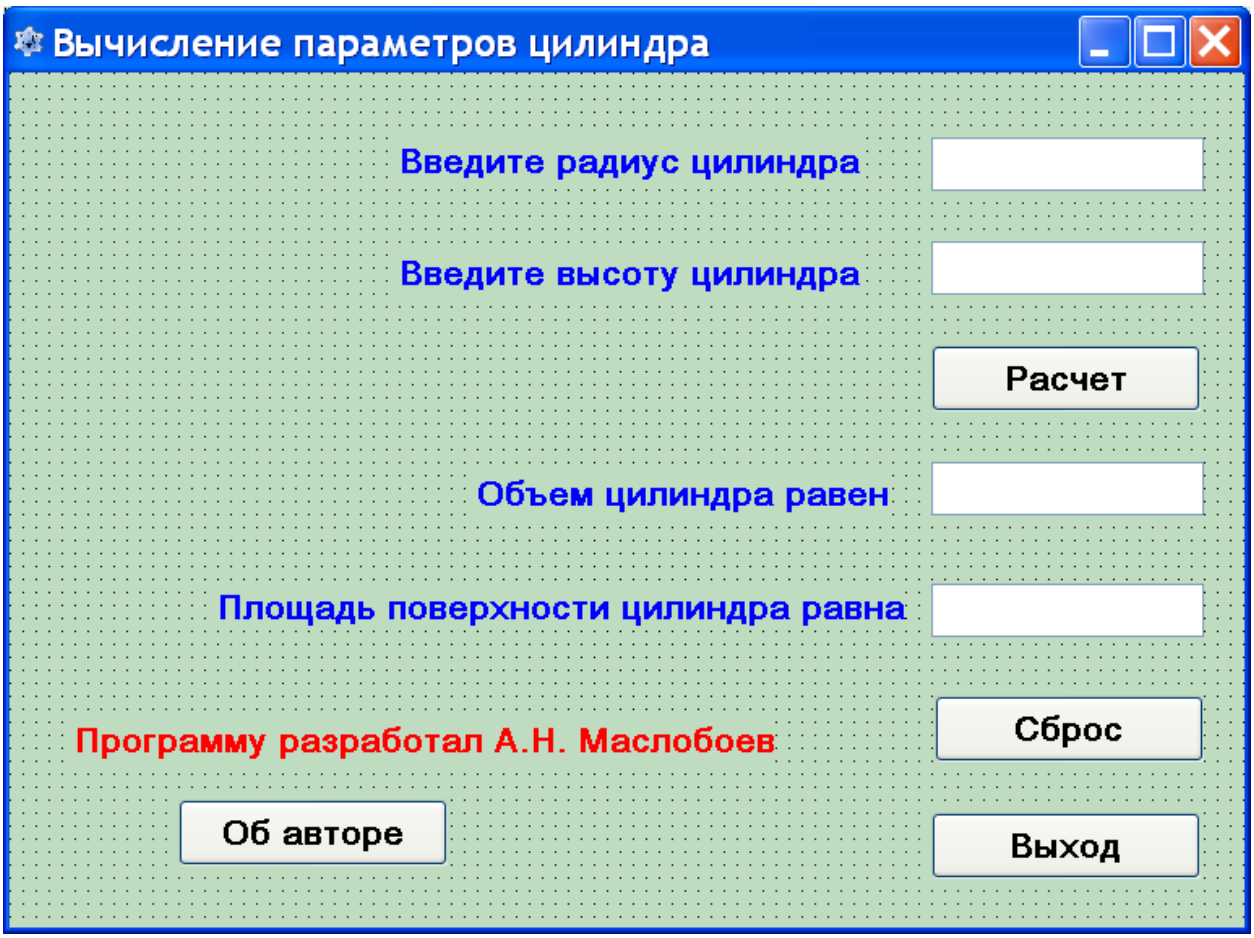

Рис. 2.11. Примерный вид интерфейса приложения «Вычисление параметров цилиндра»

Глава 3. Использование операторов выбора

Задачи вычислительного характера, рассмотренные в предыдущей главе данного пособия, относятся к программам линейной структуры. В таких программах все операторы выполняются последовательно друг за другом.

Но при решении многих поставленных перед программистом задач часто возникает ситуация, когда нужно выбирать один из двух или нескольких возможных вариантов дальнейшего хода программы. В этом случае используется структура, называемая ветвлением. В программах с ветвлением в зависимости от значений вводимых исходных данных может работать одна из двух или нескольких ветвей. Для этой цели в языке Object Pascal используются два вида операторов: условный оператор и оператор множественного выбора.

3.1. Условный оператор

Условный оператор в языке Object Pascal используется для решения задач, в которых требуется выбрать один вариант действий из двух возможных. Общая структура оператора следующая: в начале оператора находится заголовок, который содержит некоторое условие. В зависимости от выполнения или невыполнения этого условия и производится выбор соответствующего варианта.

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 операции сравнения различных величин. В языке Паскаль используется шесть таких операций:

< меньше > больше = равно

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

Написание последних трех операций отличается от принятого в математике, где соответственно используются обозначения $\leq$, $\geq$ и $\neq$, но так как клавиши с подобными обозначениями отсутствуют на клавиатуре компьютера, то используются вышеуказанные сочетания из двух символов.

Результатом проверки является логическая величина, которая может принимать только два значения: «**ИСТИНА**» или «**ЛОЖЬ**». В Object Pascal для этих значений используются англоязычные названия **TRUE** и **FALSE**.

В случае истинности проверяемого соотношения (логическая величина имеет значение **TRUE**) выполняется один вариант действий, а в случае ложности (значение логической величины **FALSE**) – другой вариант. Такой

выбор дальнейшего хода действий реализуется с помощью условного оператора **if**…**then**…**else**.

Общий вид данного оператора 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 идет проверка, выполняется ли условие, находящееся после слова **if**. Если это условие истинно, то осуществляется первый вариант действий, в противном случае, если условие ложно – второй, записанный после служебного слова **else**. В самом простом случае действие, осуществляемое в каждом из вариантов, состоит из одного оператора.

Тот вид условного оператора, который был описан нами выше, 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.

Достаточно часто в процессе разработки программ возникает необходимость разместить в одной или в 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 и другие виды операторов. Вложенным условным оператором называется условный оператор, входящий в качестве составной части в другой условный оператор. Количество простых операторов, входящих в составной оператор, не ограничено.

Операторы, входящие в составной оператор, друг от друга отделяются точками с запятой. После слова **end** в данном случае точка с запятой не ставится (если только это слово не находится в конце всего условного оператора), так как **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.

Рассмотрим применение условного оператора на следующем примере: требуется составить программу решения стандартного квадратного уравнения вида $ax^2+bx+c=0$. Как известно, решение данного уравнения зависит от величины, называемой дискриминантом. Если дискриминант неотрицателен, то уравнение имеет решения в действительных числах. В противном случае (если дискриминант отрицателен) уравнение не имеет решения в действительных числах.

Разрабатываемое нами приложение должно по заданным коэффициентам либо выводить корни данного уравнения, либо выводить информацию об отсутствии решения. Как обычно, создаем в Lazarus новое приложение и сохраняем его в отдельной папке. Затем начинаем работу над пользовательским интерфейсом.

Этот интерфейс включает такие элементы как три текстовых окна для ввода коэффициентов, три поясняющие надписи к каждому из окон, экранная кнопка «**Расчет**», которая производит необходимые вычисления, два текстовых окна для вывода корней уравнения и поясняющая надпись к ним, экранная кнопка «**Сброс**» для очистки всех текстовых окон и кнопка «**Выход**». Пользовательский интерфейс данного приложения приведен ниже на рис. 3-1.

Окна для ввода исходных данных по умолчанию получают названия **Edit1**, **Edit2** и **Edit3**. Поясняющие надписи имеют названия **Label1**, **Label2** и **Label3**. Экранная кнопка «**Расчет**» получает название **Button1**. Окна для вывода корней уравнения имеют названия **Edit4** и **Edit5**. Поясняющая надпись к ним получила название **Label4**. Экранные кнопки «**Сброс**» и «**Выход**» получают соответственно имена **Button2** и **Button3**.

Окна **Edit4** и **Edit5**, а также надпись **Label4** делаем невидимыми. Для этого в **Инспекторе объектов** для каждого из указанных объектов выбираем свойство **Visible** и устанавливаем для него значение **False**, снимая флажок, находящийся справа от названия свойства.

После выполнения начальной настройки свойств объектов приступаем к написанию программного кода для данного приложения. Начнем с программирования кнопки «**Расчет**». Дважды щелкнув клавишей мыши по кнопке, создаем новую процедуру **Button1.Click**. После заголовка процедуры описываем; **x1** и **x2** – корни уравнения. Все переменные относятся к вещественному типу, а соответствующее описание выглядит так:

```
var a,b,c,d,x1,x2:real;
```

Затем необходимо получить в численном виде значения коэффициентов, для чего необходимо их преобразовать из строкового значения в вещественное с помощью функции **StrToFloat.** Эти преобразования выполняются с помощью следующих операторов:

```
a:=StrToFloat(Edit1.Text);
b:=StrToFloat(Edit2.Text);
c:=StrToFloat(Edit3.Text);
```

Следующим шагом является вычисление дискриминанта:

```
d:=sqr(b)-4*a*c;
```

В этом операторе для возведения числа в квадрат используется функция **sqr**, которая является стандартной для языка Object Pascal. Затем значение дискриминанта используется в условном операторе выбора. Если оно больше или равно нулю, то вычисляются корни уравнения **x1** и **x2**.

Строго говоря, при нулевом значении дискриминанта корень получается один, но для упрощения решения задачи будем считать, что в данном случае имеются два корня уравнения, которые равны между собой. При вычислении корней используется еще одна стандартная функция **sqrt** (не путать с **sqr**) – извлечение квадратного корня. Вычисление выполняется следующими операторами:

```
x1:=(-b-sqrt(d))/(2*a);
x2:=(-b+sqrt(d))/(2*a);
```

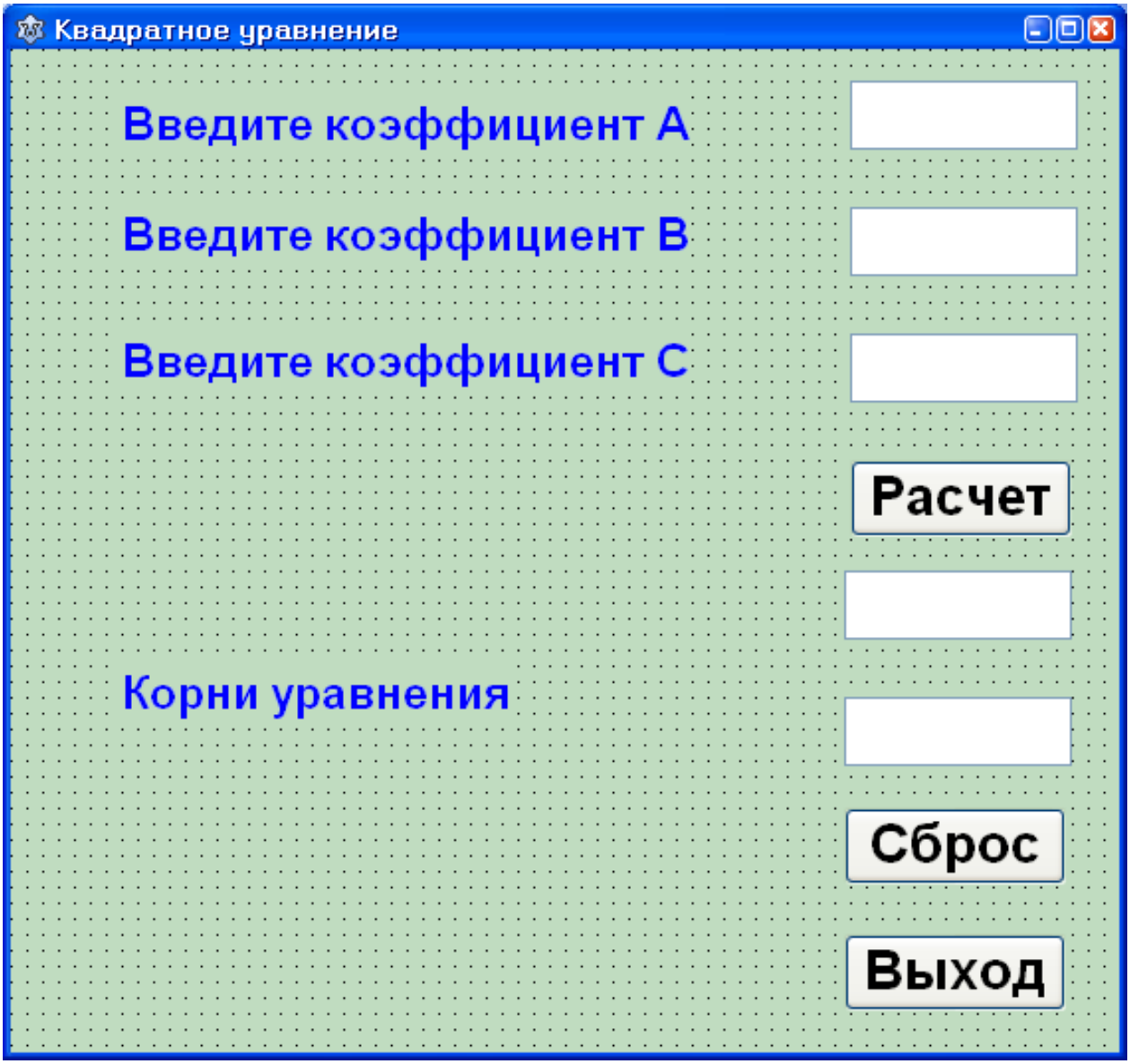

Рис. 3.1. Пользовательский интерфейс приложения «Решение квадратного уравнения»

Полученные значения преобразуются из вещественной формы в строковую с помощью функции **FloatToStr** и выводятся в окнах **Edit4** и **Edit5** с помощью следующих операторов:

```
Edit4.Text:=FloatToStr(x1);
Edit5.Text:=FloatToStr(x2);
```

Окна **Edit4** и **Edit5** делаются видимыми, так же, как и надпись **Label4**. Цвет надписи меняется на зеленый, а содержанием ее становится фраза: «Корни уравнения». Все вышеперечисленные действия находятся в условном операторе после слова **then** и объединяются с помощью операторных скобок.

После слова **else** находится другой вариант. Он содержит следующие действия: надпись **Label4** становится видимой, меняет цвет шрифта на красный, содержанием надписи становится фраза «Уравнение не имеет решения». Делать видимыми окна **Edit4** и **Edit5** в данном случае нет необходимости, поскольку выводить в них нечего. Ниже приводится весь текст условного оператора, структуру которого мы рассмотрели выше:

```
if d>=0 then
  begin
  Label4.Caption:='Корни уравнения';
  Label4.Font.Color:=clGreen;
  Label4.Visible:=True;
  x1:=(-b-sqrt(d))/(2*a);
  x2:=(-b+sqrt(d))/(2*a);
  Edit4.Text:=FloatToStr(x1);
  Edit5.Text:=FloatToStr(x2);
  Edit4.Visible:=True;
  Edit5.Visible:=True;
  end
  else
  begin
  Label4.Caption:='Уравнение не имеет решения';
  Label4.Font.Color:=clRed;
  Label4.Visible:=True;
  end;
```

Далее программируем экранную кнопку «Сброс» (**Button2**), которая очищает все текстовые окна, имеющиеся в приложении, а также делает невидимыми четвертое и пятое текстовые окна, и надпись **Label4**. Текст тела процедуры для этой кнопки следующий:

```
Label4.Visible:=False;
Edit1.Text:='';
Edit2.Text:='';
Edit3.Text:='';
Edit4.Text:='';
Edit5.Text:='';
```

```
Edit4.Visible:=False;
Edit5.Visible:=False;
```

Наконец программируем кнопку **Выход** (**Button3**), аналогично тому, как это было сделано в предыдущих программах. Ниже приводится полный текст программного модуля для приложения «Решение квадратного уравнения».

```
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
  Label4.Caption:='Корни уравнения';
  Label4.Font.Color:=clGreen;
  Label4.Visible:=True;
  x1:=(-b-sqrt(d))/(2*a);
  x2:=(-b+sqrt(d))/(2*a);
  Edit4.Text:=FloatToStr(x1);
  Edit5.Text:=FloatToStr(x2);
  Edit4.Visible:=True;
  Edit5.Visible:=True;
  end
  else
  begin
  Label4.Caption:='Уравнение не имеет решения';
  Label4.Font.Color:=clRed;
  Label4.Visible:=True;
  end;
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
  Label4.Visible:=False;
  Edit1.Text:='';
  Edit2.Text:='';
  Edit3.Text:='';
  Edit4.Text:='';
  Edit5.Text:='';
  Edit4.Visible:=False;
  Edit5.Visible:=False;
end;
```

```
procedure TForm1.Button3Click(Sender: TObject);
begin
  Close;
end;
```

После завершения работы над программным кодом следует запустить приложение на выполнение и протестировать его при различных исходных данных. На рис.3.2 показана копия экрана приложения, находящегося в рабочем режиме при положительном значении дискриминанта, а на рис.3.3 при отрицательном дискриминанте.

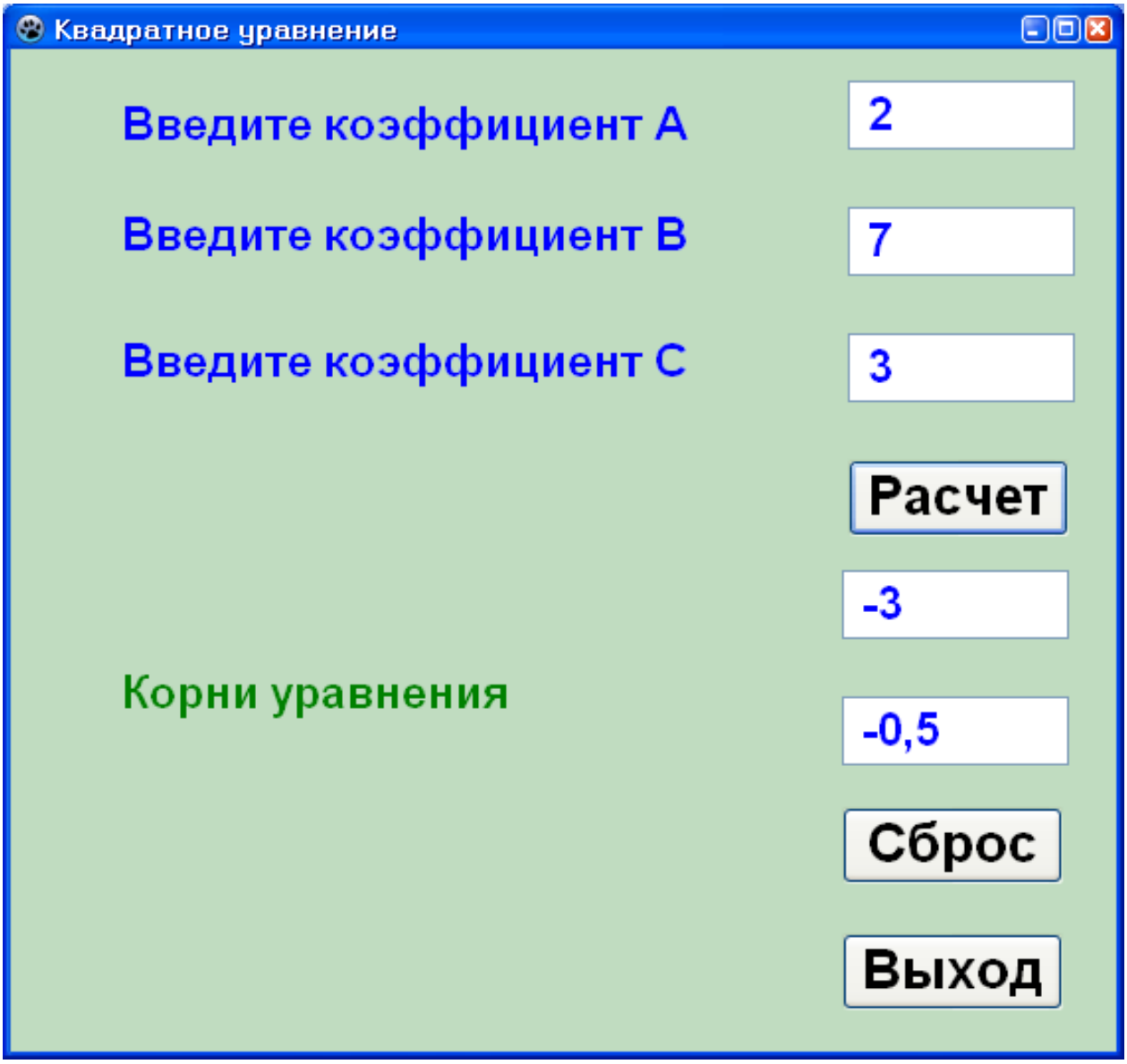

Рис. 3.2. Приложение «Решение квадратного уравнения» в рабочем режиме при положительном значении дискриминанта уравнения

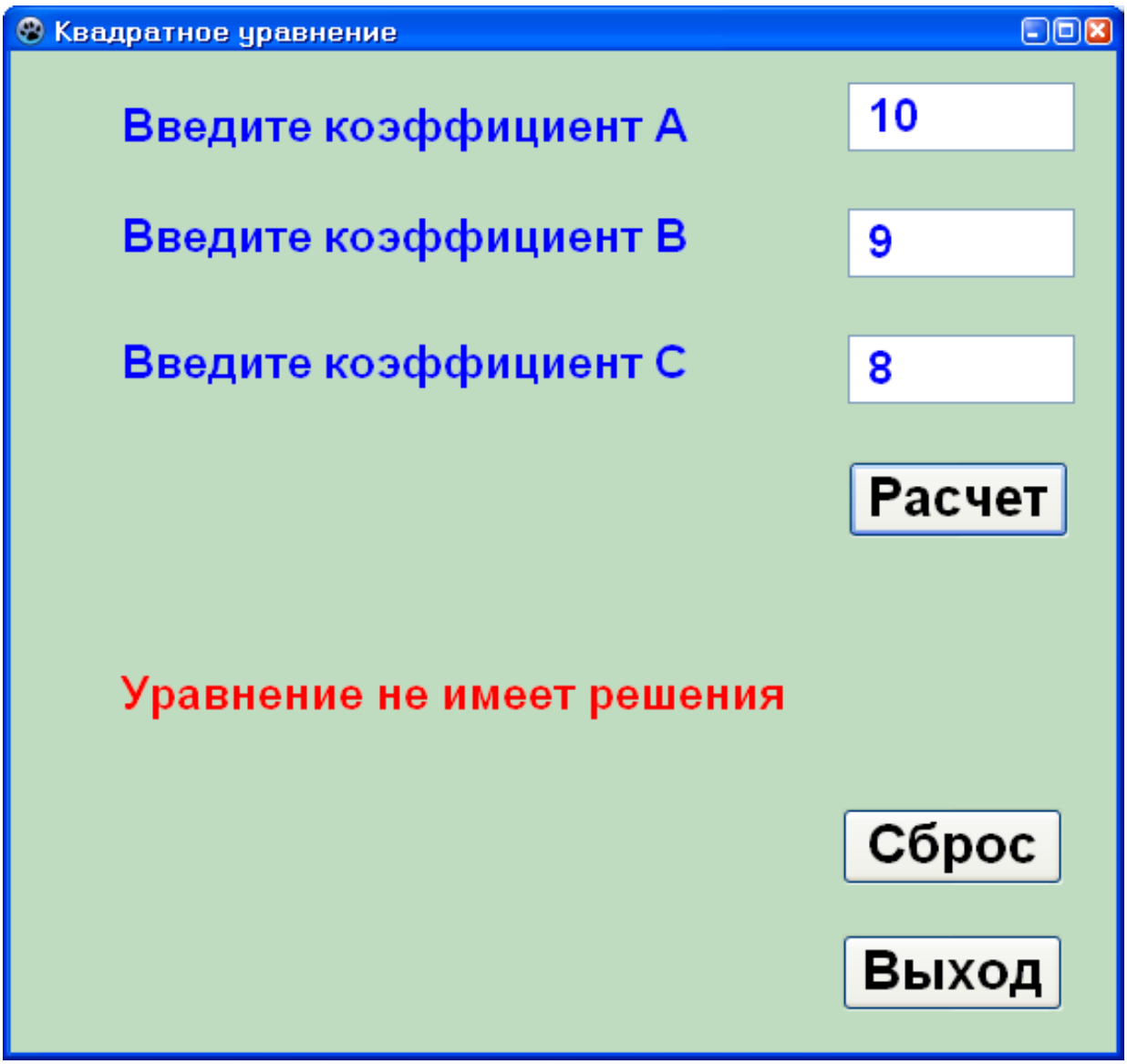

Рис. 3.3. Приложение «Решение квадратного уравнения» в рабочем режиме при отрицательном значении дискриминанта уравнения

3.2. Оператор множественного выбора

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. используют 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 в результате может привести к

появлению в программе ошибок. Поэтому в языке Object Паскаль предусмотрена другая конструкция, которая позволяет осуществлять выбор из множества возможных вариантов и в то же время является более простой и наглядной, чем вышеописанная. Эта конструкция реализуется с помощью оператора множественного выбора **case.**

Общий вид данного оператора следующий:

```
case селектор of
Значение1: Вариант1;
Значение2: Вариант2;
.....................
Значение n: Вариант n;
else  Вариант n+1;
end;
```

где **case, of , else** – ключевые слова языка Object Pascal.

Словосочетание **case**…**of**, находящееся в заголовке оператора, переводится на русский как «в случае если». Между этими словами находится переменная-селектор. Селектором называется переменная целого или иного порядкового типа. Эта переменная может принимать ряд различных значений.

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 переменная-селектор принимает это значение. Этот вариант указывается после двоеточия, отделяющего его от значения, и представляет собой простой или составной оператор. Составной оператор обязательно должен ограничиваться операторными скобками, т.е. ключевыми словами **begin** и **end**.

Если переменная-селектор не принимает ни одного из перечисленных в операторе множественного выбора значений, то выполняется альтернативный вариант, который указан после служебного слова **else**. Обратите внимание на то, что в отличие от условного оператора **if**…**then** в операторе множественного выбора перед вариантом с **else** ставится точка с запятой. Заканчивается оператор множественного выбора служебным словом end, после которого ставится точка с запятой.

Следует отметить, что каждому из вариантов, описанных в операторе множественного выбора, может соответствовать не одно значение переменной селектора, а целый ряд значений. Этот ряд значений может задаваться двумя способами:

а) в виде значений, перечисляемых перед соответствующим вариантом через запятую:

Значение1, Значение 2,… Значение k: вариант;

б) если значения образуют непрерывный диапазон, можно указать только начальное значение, две горизонтальные точки и конечное значение. Затем после двоеточия нужно указать общий для них вариант:

Значение 1 .. Значение k: вариант;

Существует также сокращенная форма записи оператора множественного выбора, отличие которой от полной заключается в том, что в ней отсутствует служебное слово **else** и соответствующий ему вариант действий. Соответственно, изменяется механизм работы оператора. Если переменная-селектор не принимает ни одного из перечисленных в операторе множественного выбора значений, то данный оператор не выполняет никаких действий и управление передается следующему за ним оператору.

Рассмотрим применение оператора множественного выбора на следующем примере: требуется составить программу, которая по введенному номеру дня недели определяет, является ли данный день будним днем, субботой или воскресеньем. В случае, если пользователь ввел номер дня недели, которого не существует (например, 8 или 9), программа должна вывести на экран сообщение об ошибке.

Как обычно, создаем в Lazarus новое приложение и сохраняем его в отдельной папке Week. Затем начинаем работу над пользовательским интерфейсом.

Этот интерфейс включает такие элементы как текстовое окно для ввода номера дня недели, поясняющая надпись к нему, экранная кнопка «**Ввод**», вторая надпись, выводящая пользователю информацию о дне недели, кнопка «**Сброс**» для очистки текстового окна и кнопка «**Выход**». Внешний вид интерфейса данного приложения приведен ниже на рис. 3-4.

Окно для ввода исходных данных по умолчанию получает название **Edit1**. Поясняющая надпись к этому окну имеет названия **Label1**. Экранная кнопка «Ввод» получает название **Button1**. Надпись, содержащая информацию о дне недели, имеет название **Label2**. Экранные кнопки «**Сброс**» и «**Выход**» получают соответственно имена **Button2** и **Button3**.

Надпись **Label2** делаем невидимой. Для этого в Инспекторе объектов для данного объекта выбираем свойство **Visible** и устанавливаем для него значение **False**, снимая флажок, находящийся справа от названия свойства.

После выполнения начальной настройки свойств объектов приступаем к написанию программного кода для данного приложения. Начнем с программирования кнопки «**Ввод**». Дважды щелкнув клавишей мыши по

кнопке, создаем новую процедуру **Button1.Click**. После заголовка процедуры описываем переменную-селектор **N**, относящуюся к целому типу:

```
var N:integer;
```

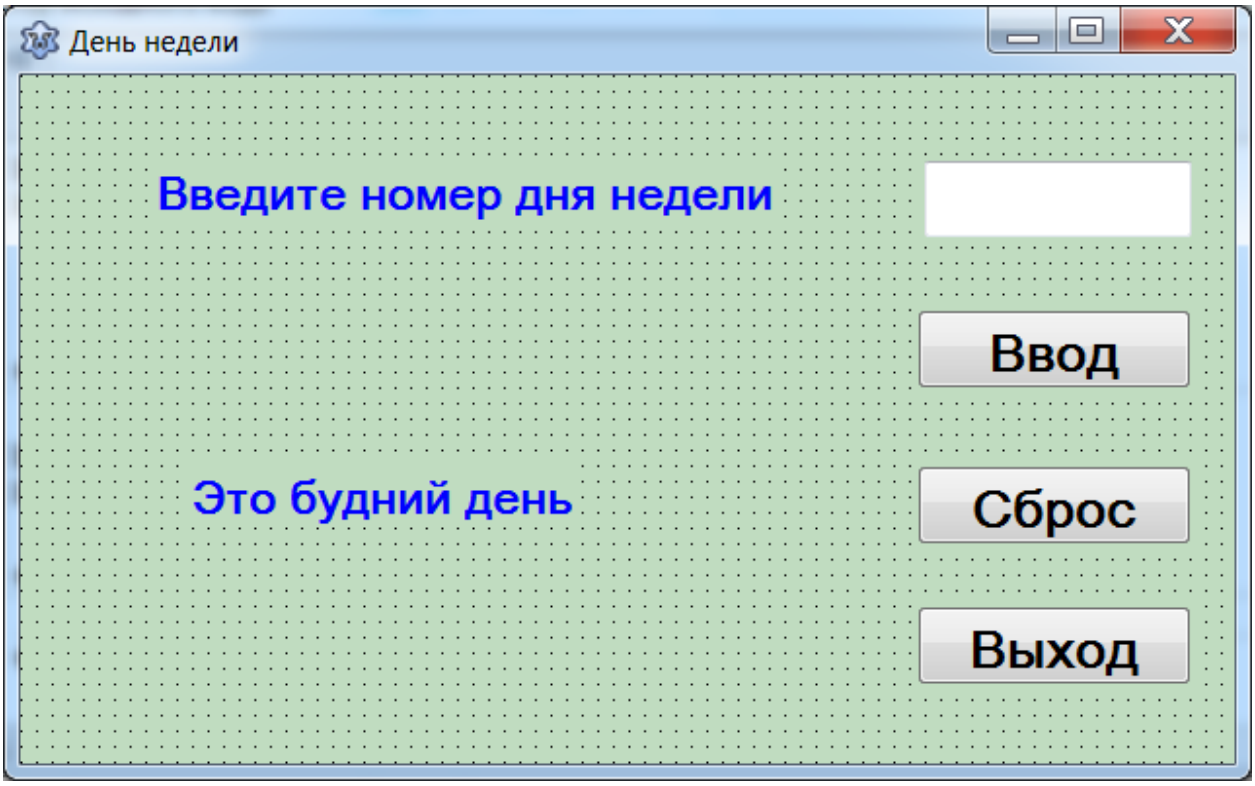

Рис. 3.4. Пользовательский интерфейс приложения «День недели»

Затем значение переменной **N** необходимо получить в целочисленном виде, для чего необходимо преобразовать значение введенное в окно **Edit1** из строкового в целочисленное с помощью функции **StrToInt**. Это преобразование выполняется с помощью следующих операторов:

```
N:=StrToInt(Edit1.Text);
```

Следующим шагом является использование этой переменной в операторе множественного выбора. В начале пишем заголовок оператора:

```
Case N of
```

Затем рассматриваем возможные значения переменной-селектора. В нашей задаче возможные следующие варианты:

а) номер дня недели от 1 до 5: в этом случае синим цветом будет выводиться сообщение, что день является будним.

Соответствующая ветвь оператора множественного выбора выглядит так:

```
1..5:
```

```
begin
Label2.Font.Color:=clBlue;
Label2.Caption:='Это будний день';
end;
```

поскольку номера будних дней недели образуют непрерывный диапазон, то мы указываем только начальное значение 1 и конечное 5, а между ними ставим две горизонтальные точки. Можно было бы перечислить эти значения через запятую и таким образом:

```
1,2,3,4,5:
```

Это тоже допустимый, правильный способ задания значений переменной-селектора. Если же группа значений переменной не образует непрерывного диапазона, то последний приведенный способ является единственно возможным.

б) номер дня недели – 6: зеленым цветом будет выводиться сообщение о том, что это день – суббота.

Соответствующая ветвь оператора множественного выбора выглядит таким образом:

```
6:
      begin
      Label2.Font.Color:=clGreen;
      Label2.Caption:='Это суббота';
      end;
```

в) номер дня недели - 7: красным цветом будет выведена информация о том, что это – воскресенье

Соответствующая ветвь оператора

```
7:
      begin
      Label2.Font.Color:=clRed;
      Label2.Caption:='Это воскресенье';
      end;
```

г) пользователь ввел несуществующий в действительности номер дня недели: черным цветом должно быть выведено сообщение об ошибке.

Соответствующая ветвь оператора

```
else
      begin
      Label2.Font.Color:=clBlack;
      Label2.Caption:='Ошибка ввода';
      end;
```

Таким образом, в целом весь оператор множественного выбора будет выглядеть так:

```
case N of
1..5:
     begin
     Label2.Font.Color:=clBlue;
     Label2.Caption:='Это будний день';
     end;
6:
     begin
     Label2.Font.Color:=clGreen;
     Label2.Caption:='Это суббота';
     end;
7:
     begin
     Label2.Font.Color:=clRed;
     Label2.Caption:='Это воскресенье';
     end;
else
     begin
     Label2.Font.Color:=clBlack;
     Label2.Caption:='Ошибка ввода';
     end;
end;
```

Последний оператор процедуры Button1.Click делает видимой надпись Label2 с информацией о дне недели:

```
Label2.Visible:=True;
```

Далее программируем экранную кнопку «**Сброс**» (**Button2**), которая очищает текстовое окно, имеющееся в приложении, а также делает невидимой надпись **Label2** с очисткой ее содержимого:

```
Edit1.Text:='';
Label2.Visible:=False;
Label2.Caption:='';
```

Наконец программируем кнопку **Выход (Button3)**, аналогично тому, как это было сделано в предыдущих программах.

Ниже приводится полный текст программного модуля для приложения «День недели».

```
procedure TForm1.Button1Click(Sender: TObject);
var N:integer;
begin
```

```
N:=StrToInt(Edit1.Text);
case N of
1..5:
     begin
     Label2.Font.Color:=clBlue;
     Label2.Caption:='Это будний день';
     end;
6:
     begin
     Label2.Font.Color:=clGreen;
     Label2.Caption:='Это суббота';
     end;
7:
     begin
     Label2.Font.Color:=clRed;
     Label2.Caption:='Это воскресенье';
     end;
else
     begin
     Label2.Font.Color:=clBlack;
     Label2.Caption:='Ошибка ввода';
     end;
end;
Label2.Visible:=True;
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
  Edit1.Text:='';
  Label2.Visible:=False;
  Label2.Caption:='';
end;

procedure TForm1.Button3Click(Sender: TObject);
begin
  Close;
end;
```

После завершения работы над программным кодом следует запустить приложение на выполнение и протестировать его при различных исходных данных. На рис. 3.5 показана копия экрана приложения, находящегося в рабочем режиме при правильном вводе исходных данных, а на рис. 3.6 при ошибочном вводе.

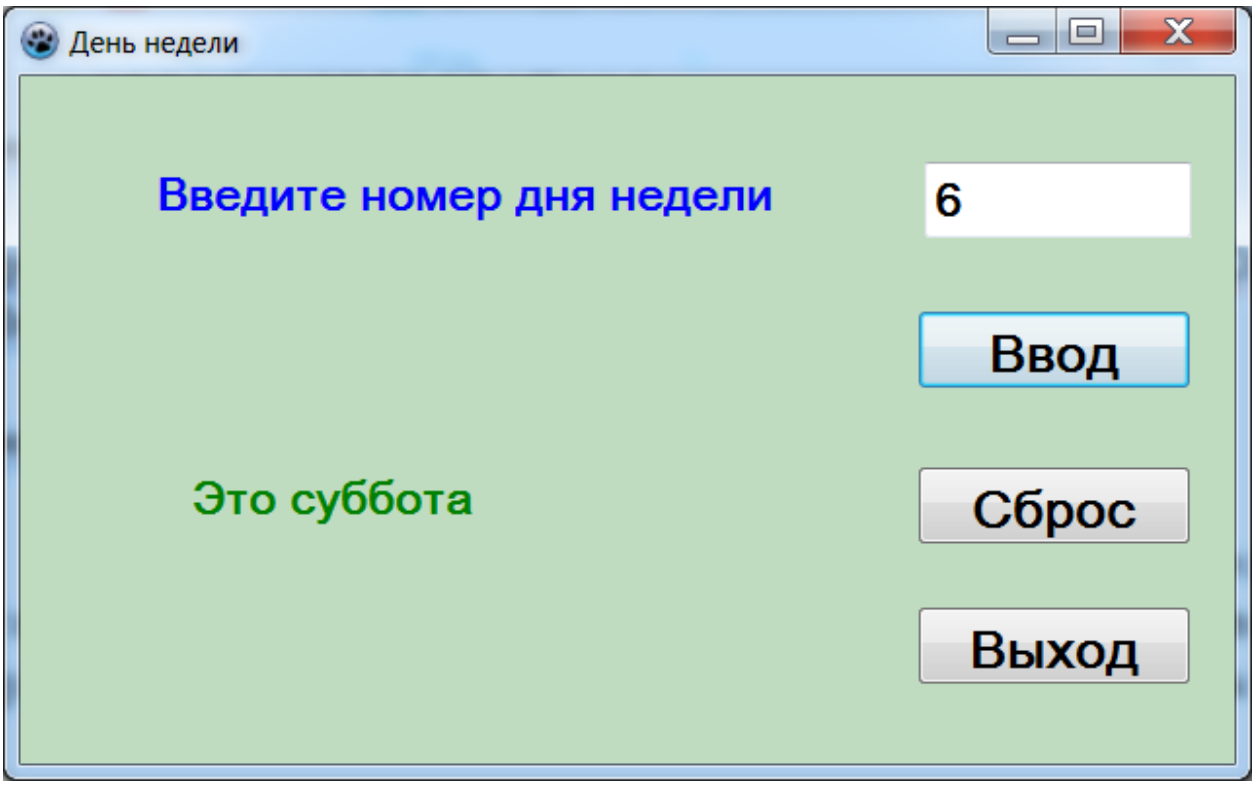

Рис. 3.5. Копия экрана приложения «День недели» в рабочем режиме при правильном вводе исходных данных

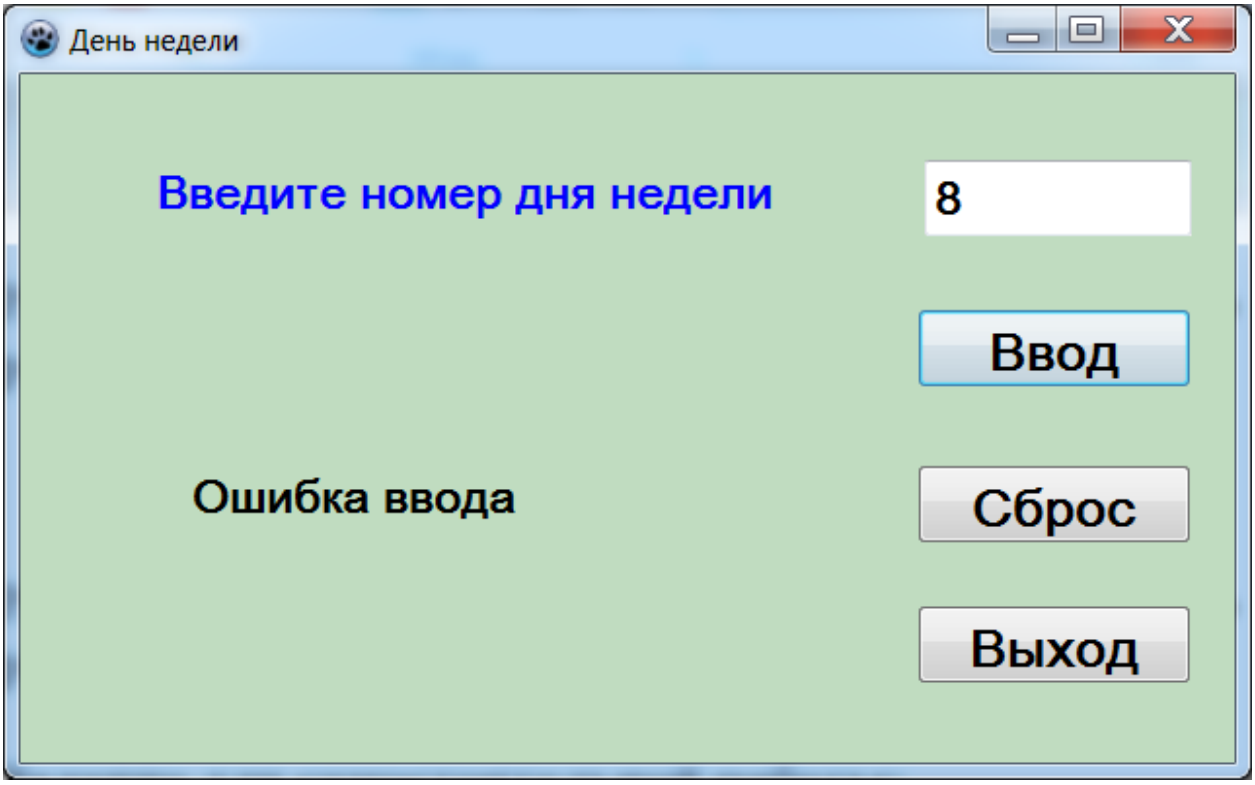

Рис. 3.6. Копия экрана приложения «День недели» в рабочем режиме при ошибочном вводе исходных данных

Контрольные вопросы

1. Какие операции сравнения могут использоваться в условном операторе?

2. С какого ключевого слова всегда начинается условный оператор в Object Pascal?

3. В чем состоит различие между полной и сокращенной формой условного оператора?

4. Каким образом можно определить границы составного оператора в Object Pascal?

5. Когда целесообразно использовать в программе оператор множественного выбора?

6. К каким типам может относиться переменная-селектор в операторе множественного выбора?

7. С какого ключевого слова всегда начинается оператор множественного выбора?

8. Каким ключевым словом всегда заканчивается оператор множественного выбора?

9. Что происходит в операторе множественного выбора, если переменная-селектор не принимает ни одного из перечисленных в нем значений?

10. В каком операторе (оператор сравнения или оператор множественного выбора) перед ключевым словом else ставится точка с запятой?

11. Какой символ отделяет в операторе множественного выбора значение или группу значение переменной селектора от соответствующего ей варианта?

12. Каким образом в условном операторе можно выполнить совместную проверку нескольких условий?

Задание для самостоятельной работы

Компьютерная фирма предоставляет своим клиентам скидку на приобретаемую ими продукцию в зависимости от ее стоимости. Если клиент приобретает продукцию на сумму от 5000 руб до 14999 руб, скидка составляет 3 % от исходной стоимости заказа. Если стоимость заказа более 15000 руб, то скидка составляет 5 %. При сумме заказа менее 5000 руб скидка отсутствует.

Разработать приложение, которое определяет величину скидки и итоговую стоимость заказа, т.е. исходную стоимость за вычетом скидки.

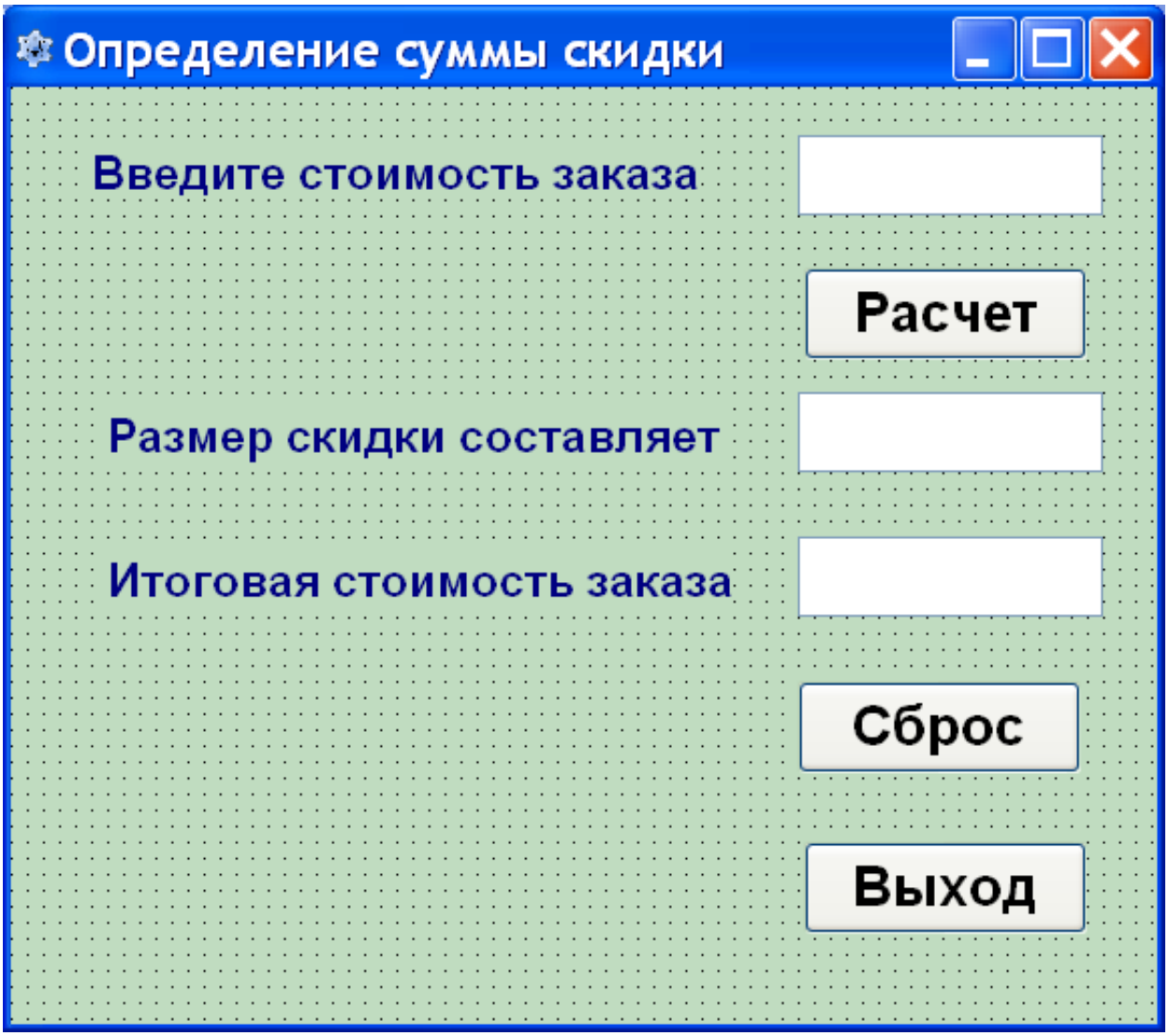

Рис. 3.7. Примерный вид пользовательского интерфейса приложения «Определение суммы скидки»

Глава 4. Использование операторов цикла

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 отличаются лишь значениями некоторых переменных или вводимых исходных данных.

Естественно, что возникает вопрос: нельзя ли использовать в таких случаях какую-либо структуру, которая бы позволяла облегчить и сделать менее трудоемким решение задачи путем автоматического повторения данного оператора или группы операторов с соответствующим изменением значений переменной или переменных. Такая структура в языке Паскаль существует и называется циклом.

В общем виде цикл состоит из двух основных блоков:

1. Заголовок цикла. В заголовке цикла содержится некоторое условие, которое определяет число повторений цикла.

2. Тело цикла. Телом цикла называется простой или составной оператор (как известно, 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 могут быть операторы ввода и вывода, операторы присваивания, условные операторы, а также другие операторы цикла. Оператор цикла, находящийся в теле другого циклического оператора, называется вложенным.

При работе цикла постоянно производится проверка содержащегося в заголовке условия. В зависимости от истинности условия и вида цикла либо очередной раз выполняются операторы тела цикла, либо цикл завершается. Если выполняются операторы тела цикла, то по завершении этого процесса вновь проверяется условие в заголовке и все начинается сначала. Если же работа цикла завершается, то управление программой переходит к оператору, расположенному в программе вслед за циклом.

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 приводило бы к завершению работы цикла.

Циклы могут использоваться для решения самых разнообразных задач, и, соответственно, структура самих циклов может несколько различаться. Всего в языке Паскаль используется три разновидности циклов:

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

Каждый из этих видов цикла имеет свои особенности и область применения, которые мы и рассмотрим ниже.

В том случае, если количество повторений цикла заранее определить невозможно, в Object Pascal используют следующие виды циклов: цикл с постусловием и цикл с предусловием.

Общий вид оператора цикла с постусловием следующий:

```
repeat
тело цикла
until условие;
```

где **repeat** и **until** – служебные слова. Слово **repeat** в переводе с английского означает «повторять», **until** – «до тех пор». Цикл работает следующим образом: вначале выполняется оператор или группа операторов, входящих в тело цикла. Затем проверяют, 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**, является ложным.

Такой цикл называют циклом с постусловием (латинская приставка «пост» означает «после»). Это 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.

Общий вид оператора цикла с предусловием следующий:

```
while   условие   do
тело цикла;
```

где **while** и **do** – служебные слова, **while** в переводе с английского 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 и 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.

Общий вид оператора цикла с заранее заданным количеством повторений 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. 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, но допускаются и некоторые другие типы. Категорически нельзя использовать в качестве счетчика переменные вещественного типа;

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

тело цикла - в качестве тела цикла выступает обычный оператор или составной оператор, в который входит несколько других.

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.

Приведенный вид записи цикла **for** не является единственным. Цикл 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.

Проектом, который мы будем разрабатывать в среде Lazarus с использованием циклических структур, является проект «Вычисление суммы банковского вклада». 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.

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 вышесказанного, можно составить общее представление об интерфейсе программы в среде Lazarus. Интерфейс данной программы должен включать в себя следующие элементы:

- три текстовых окна, используемых для ввода исходных данных;
- поясняющие надписи к данным окнам;
- две экранные кнопки одна – для вычисления итоговой суммы вклада («**Расчет**»), а вторая – для очистки всех текстовых окон («**Сброс**»);
- текстовое окно для вывода полученных результатов;
- поясняющая надпись к этому окну;
- иллюстрация к проекту;
- экранная кнопка, 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 «Вычисление суммы банковского вклад». Затем создаем на форме текстовые окна **Edit1**, **Edit2** и **Edit3** для ввода исходных данных. Слева от каждого из этих окон должна быть сделана надпись, что и реализовано в виде объектов **Label1**, **Memo1** и **Label2**. Для окна **Edit2** создано поле **Memo1**, чтобы поясняющую надпись можно было расположить в 2 строки.

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 на форме расположено окно **Edit4**, которое используется для вывода результатов. Слева от окна расположена пояснительная надпись, которая реализована в виде поля **Memo2**. Справа от окна находится небольшая надпись **Label3**, указывающая вид валюты - рубли.

В правой верхней части формы мы расположим иллюстрацию к проекту в виде растрового рисунка (объект **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 завершено. Интерфейс проекта вклад приведен на рис. 4.1.

Разработку программного кода начнем с процедуры **Button1Click** для экранной кнопки "Расчет". В начале описываем переменные:

sum - исходная сумма вклада;

n - срок действия договора по вкладу (в годах);

p - процент по вкладу;

i - переменная цикла;

itog - сумма, которая будет находиться на счете клиента по окончании срока действия договора.

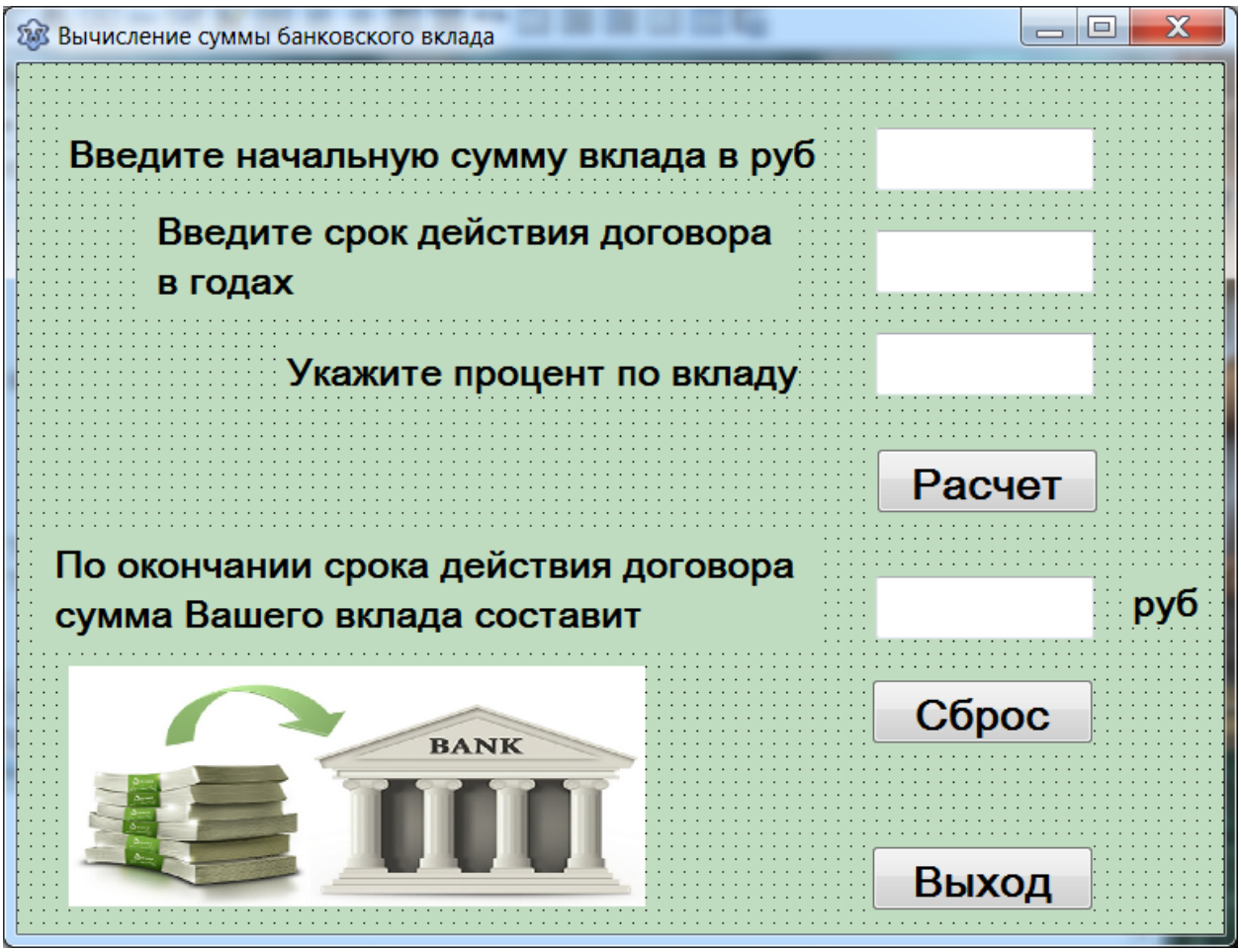

Рис. 4.1. Пользовательский интерфейс приложения «Вычисление суммы банковского вклада»

Переменные **sum,n,p,i** относятся к "длинному" целому типу, а **itog** - к вещественному, а само описание выглядит так:

```
var sum,n,p,i:longint;
itog:real;
```

Затем уже в теле процедуры преобразуем исходные данные, введенные в три текстовых окна из строковой формы в целочисленную:

```
sum:=StrToInt(Edit1.Text);
n:=StrToInt(Edit2.Text);
p:=StrToInt(Edit3.Text);
```

Переменной **itog** присваиваем начальное значение равное исходной сумме:

```
itog:=sum;
```

Далее значение переменной **itog** будет увеличиваться в ходе выполнения цикла.

Затем организуем цикл с заранее известным количеством повторений, в котором переменная цикла **i** будет изменяться от начального значения 1 до конечного **n**. Заголовок цикла:

```
for i:=1 to n do
```

Тело цикла здесь содержит только один оператор, в котором к текущей сумме вклада будет добавляться процент по вкладу. Тело цикла:

```
itog:=itog*(1+p/100);
```

Последний оператор процедуры преобразует полученное значение из вещественной формы в строковую, причем для преобразоваия используется функция **FloatToStrF**, которая позволяет также задать формат выводимого значения (в данном случае с точностью до 2 цифр после точки, т.е. в рублях и копейках).

```
Edit4.Text:=FloatToStrf(itog,ffGeneral,7,2);
```

В целом процедура **Button1Click** выглядит таким образом:

```
procedure TForm1.Button1Click(Sender: TObject);
 var sum,n,p,i:longint;
itog:real;
begin
  sum:=StrToInt(Edit1.Text);
  n:=StrToInt(Edit2.Text);
  p:=StrToInt(Edit3.Text);
  itog:=sum;
  for i:=1 to n do  itog:=itog*(1+p/100);
  Edit4.Text:=FloatToStrf(itog,ffGeneral,7,2);
end;
```

Процедуры **Button2Click** (для кнопки "**Сброс**") и **Button3Click** (для кнопки "**Выход**") программируются так же, как это было сделано в предыдущих проектах.

Ниже приводится полный текст программного модуля для данного приложения.

```
unit Unit1;

{$mode objfpc}{$H+}
  Interface
  uses
```

```
    Classes, SysUtils, FileUtil, Forms, Controls,
Graphics, Dialogs, StdCtrls, ExtCtrls;
  Type
    { TForm1 }
    TForm1 = class(TForm)
      Button1: TButton;
      Button2: TButton;
      Button3: TButton;
      Edit1: TEdit;
      Edit2: TEdit;
      Edit3: TEdit;
      Edit4: TEdit;
      Image1: TImage;
      Label1: TLabel;
      Label2: TLabel;
      Label3: TLabel;
      Memo1: TMemo;
      Memo2: TMemo;
       procedure Button1Click(Sender: TObject);
      procedure Button2Click(Sender: TObject);
      procedure Button3Click(Sender: TObject);
    private
      { private declarations }
    Public
      { public declarations }
    end;

  var
    Form1: TForm1;

  Implementation

  {$R *.lfm}

  { TForm1 }

  procedure TForm1.Button3Click(Sender: TObject);
  begin
  Close;
  end;

  procedure TForm1.Button2Click(Sender: TObject);
  begin
    Edit1.Text:='';
    Edit2.Text:='';
    Edit3.Text:='';
    Edit4.Text:='';
```

```
end;

procedure TForm1.Button1Click(Sender: TObject);
 var sum,n,p,i:longint;
     itog:real;
begin
  sum:=StrToInt(Edit1.Text);
  n:=StrToInt(Edit2.Text);
  p:=StrToInt(Edit3.Text);
  itog:=sum;
  for i:=1 to n do
  itog:=itog*(1+p/100);
  Edit4.Text:=FloatToStrf(itog,ffGeneral,7,2);
end;
end.
```

После окончания программирования запускаем приложение на выполнение. Возможный вид приложения во время работы показан на рис. 4.2.

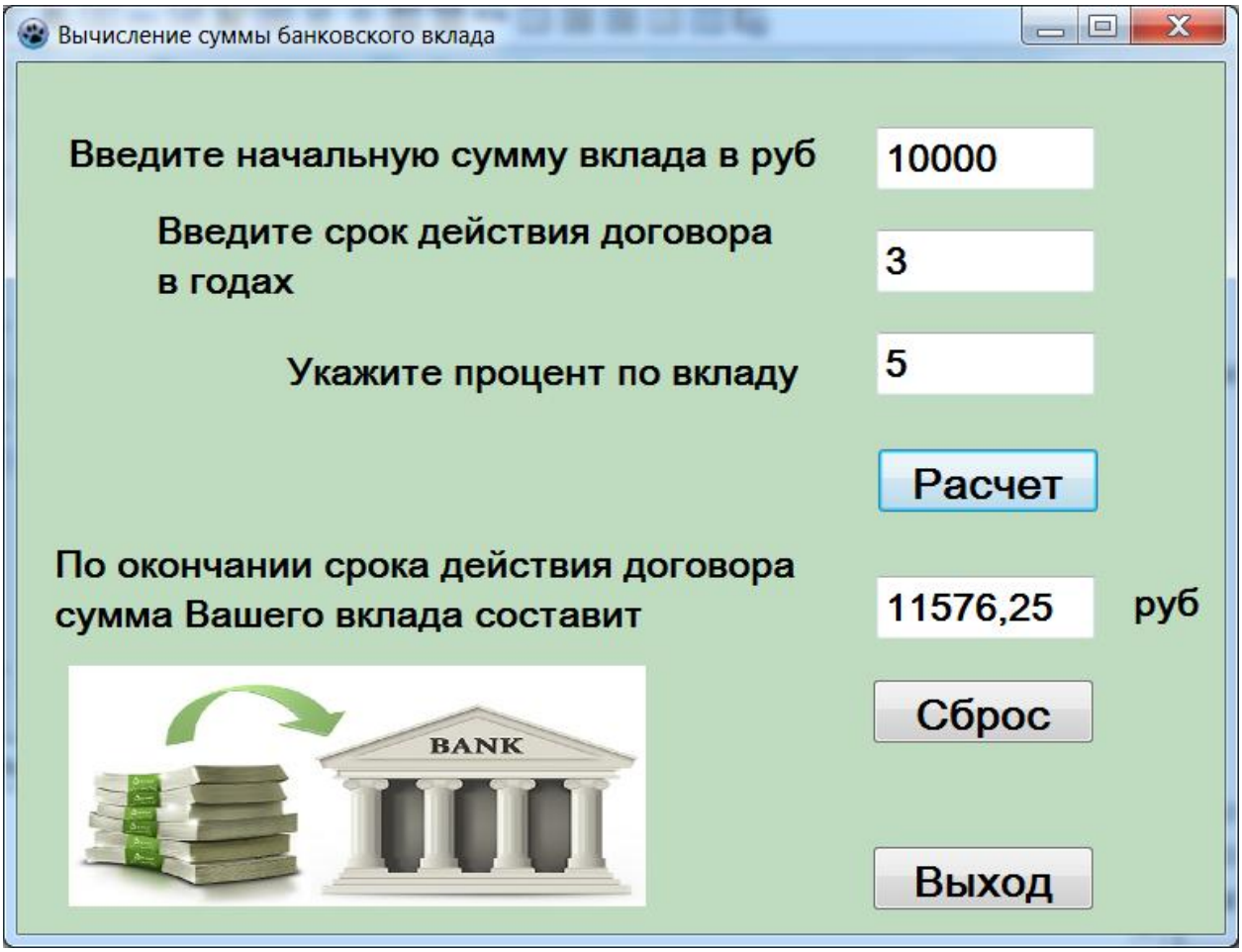

Рис. 4.2. Приложение «Вычисление суммы банковского вклада» в режиме выполнения

Контрольные вопросы

1. Перечислите основные виды циклов, используемые в Object Pascal.

2. С какого ключевого слова начинается цикл с заранее заданным количеством повторений?

3. К каким типам может относиться переменная цикла с заранее заданным количеством повторений?

4. Какие ключевые слова в цикле с заранее заданным количеством повторений определяют, происходит ли в цикле приращение значения переменной или уменьшение этого значения?

5. Что представляют собой в Object Pascal операторные скобки, задающие границы тела цикла?

6. В каком виде цикла необязательно использовать операторные скобки?

7. С какого ключевого слова начинается цикл с предусловием?

8. С какого ключевого слова начинается цикл с постусловием?

9. Может ли цикл с предусловием не выполниться ни разу?

10. Может ли цикл с постусловием не выполниться ни разу?

11. Истинным или ложным должно быть условие, заданное в цикле с постусловием, для завершения его работы?

12. Истинным или ложным должно быть условие, заданное в цикле с предусловием, для завершения его работы?

Задание для самостоятельной работы

Разработать приложение, которое для заданного натурального числа вычисляет его факториал. Примерный вид пользовательского интерфейса приложения приведен на рис. 4.3.

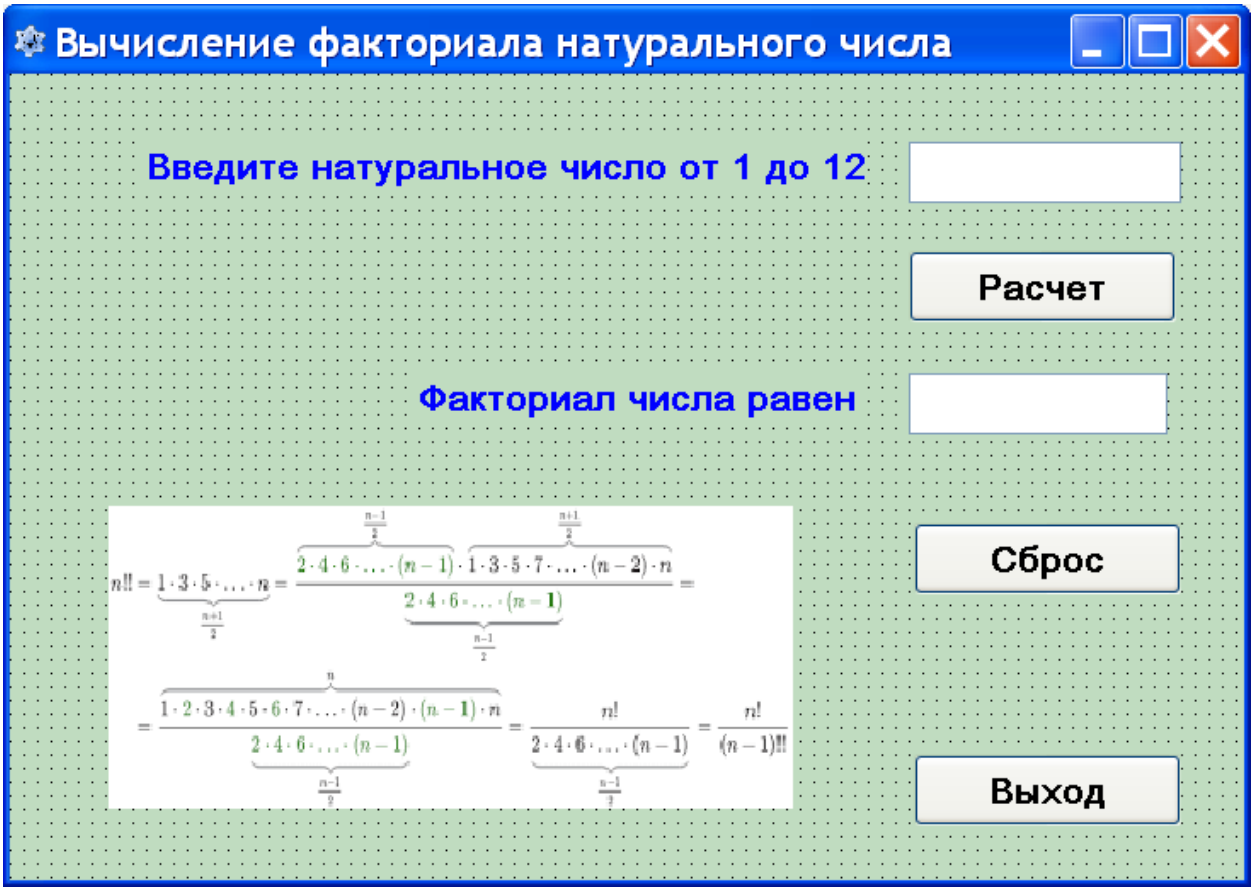

Рис. 4.3. Примерный вид пользовательского интерфейса приложения «Вычисление факториала натурального числа»

Глава 5. Создание авторских функций

При составлении различных программ часто возникает необходимость в выполнении таких операций, как возведение числа в квадрат, извлечение квадратного корня, определение модуля числа, вычисление тригонометрических функций, округление дробного числа до ближайшего целого, определение четности или нечетности целого числа и многих других.

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

Функция 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 круглых скобках. Есть небольшое количество функций, которые не имеют аргументов, но значение существует у любой функции.

Ниже приводится список наиболее часто используемых стандартных функций языка Object Pascal (некоторые из них уже знакомы по предыдущим главам данного пособия):

- **`abs(x)`** – используется для получения абсолютной величины числа;
- **`chr(k)`** – определяет символ по его коду;
- **`concat(s1,s2,…)`** – объединяет несколько строк в одну;
- **`copy(s,n,l)`** – создает копию части строки или части массива;
- **`cos(x)`** – вычисляет косинус числа;
- **`date`** – определяет текущую дату;
- **`exp(x)`** – вычисляет экспоненту числа;
- **`floattostr(x)`** – преобразует вещественное число в строку;
- **`inttostr(x)`** – преобразует целое число в строку;
- **`length(s)`** – определяет число элементов в массиве или строке;
- **`ln(x)`** – вычисляет натуральный логарифм числа;
- **`log10(x)`** – вычисляет десятичный логарифм числа;
- **`max(a,b)`** – определяет большее число из двух целых значений;
- **`min(a,b)`** – определяет меньшее число из двух целых значений;
- **`now`** – определяет текущую дату и время;
- **`odd(x)`** – проверяет, является ли целое число нечетным;
- **`ord(c)`** – определяет числовой код символа;
- **`pi`** – вычисляет значение числа Пи;
- **`random(x)`** – генерирует случайное целое число;
- **`round(x)`** – округляет вещественное число до целого числа;

- **sin(x)** – вычисляет синус числа;
- **sqr(x)** – вычисляет квадрат числа;
- **sqrt(x)** – вычисляет квадратный корень числа;
- **strToFloat(s)** – преобразует строку в вещественное число;
- **strToInt(s)** – преобразует строку в целое число;
- **tan(x)** – вычисляет тангенс числа;
- **time** – определяет текущее время;
- **trunc(x)** – определяет целую часть числа вещественного числа.

Язык Object Pascal имеет достаточно обширный набор стандартных функций (выше перечислена только некоторая часть этих функций). Но возможны ситуации, когда этого набора стандартных функций при разработке программы оказывается недостаточно. Например, в Object Pascal есть функция возведения числа в квадрат, но нет функции, которая бы возводила число в произвольную целую степень. Отсутствуют функции вычисления факториала натурального числа, перевода чисел из одной системы счисления в другую и ряд других полезных функций.

В этом случае программист должен самостоятельно разработать необходимую ему функцию. Такая вновь создаваемая функция называется авторской. Создание новой функции начинается с ее описания. Описание функции, как правило, находится в тексте программы после раздела описания констант и переменных и до начала ее основной части. Такую функцию часто называют функцией-подпрограммой, так как 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  локальных переменных;
begin
раздел операторов функции
end;
```

Рассмотрим элементы этой структуры более детально.

Общий вид заголовка функции следующий:

```
function имя_функции(параметры функции:тип_функции;
```

где **function** – служебное слово, означающее «функция». Имена функциям даются по тем же правилам, что и имена переменных. 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 имеет имя **beta**, в ней используются 3 параметра: **x** и **y** – целого типа; **z** – вещественного, а значение самой функции является вещественным.

Следующей частью функции является раздел описания локальных переменных. Локальными называются переменные, которые могут использоваться только внутри данной функции. Те переменные, которые используются в основной части программы, называются глобальными. Раздел описания локальных переменных выглядит так же, как и для глобальных переменных. Он начинается с ключевого слова **var**, за которым следует список локальных переменных. Для каждой переменной нужно указать ее имя и после двоеточия – тип переменной. Если локальные переменные в функции не используются, то соответствующий раздел функции можно опустить.

Последним разделом описания функции является раздел операторов, который также называют телом функции. Тело функции представляет собой оператор или группу операторов, с помощью которых вычисляется значение функции. Последним оператором в теле функции является оператор, который присваивает функции новое значение. Тело функции обязательно заключается в операторные скобки, в качестве которых используются ключевые слова **begin** и **end**. После слова end ставится точка с запятой, так как это конец описания функции, а не конец программы.

Для того чтобы использовать авторскую функцию в основной части программы, необходимо произвести вызов функции (обращение к функции). Обращение к функции не является самостоятельным оператором, а входит в качестве составной части в какой-либо оператор.

Обращение к функции включает в себя имя функции и следующий за ним список параметров, заключенный в круглые скобки. Параметры, указанные в обращении к функции, называются фактическими параметрами, а параметры, указанные в описании функции – формальными. Фактические параметры должны быть того же типа, что и формальные. При обращении к функции формальные параметры заменяются на соответствующие им фактические параметры. Затем с использованием фактических параметров вычисляется значение функции, и это вычисленное значение возвращается в основную часть программы.

Рассмотрим создание авторской функции на примере разработки приложения, которое подсчитывает число сочетаний из **n** по **k** по следующей формуле:

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

В данной формуле символ «восклицательный знак» означает факториал числа. Факториалом числа **n** называется произведение всех натуральных чисел от 1 до **n**.

При разработке приложения необходимо создать авторскую функцию, которая вычисляет факториал заданного натурального числа.

Разработку приложения начнем, как обычно, с создания папки (в данном случае она получила имя Combinations), в которой будут храниться файлы проекта – **project1** и **unit1**.

Следующим этапом является создание пользовательского интерфейса проекта и настройка свойств объектов, имеющихся в приложении. На основной экранной форме разместим следующие элементы: два текстовых окна для ввода исходных данных – значений **n** и **k**, поясняющие надписи к данным текстовым окнам, текстовое окно для вывода результата – числа сочетаний, поясняющая надпись к этому текстовому окну, три экранные кнопки: «**Расчет**» для выполнения необходимых вычислений, «**Сброс**» для очистки всех текстовых окон в случае необходимости повторных вычислений и «**Выход**» для закрытия окна приложения.

Текстовые окна для ввода исходных данных получают названия соответственно **Edit1** и **Edit2**. Для этих окон в Инспекторе объектов с помощью свойства **Font** задаем размер шрифта 16 пунктов и полужирное начертание. Затем для каждого из окон очищаем поле, расположенное справа от свойства **Text**. Это делается для того, чтобы в начале работы программы эти окна были пустыми, и в них можно было вводить исходные данные. Аналогичным образом задаем характеристики шрифта и очищаем свойство «**Text**» для текстового окна **Edit3**, в котором будут отображаться результаты вычислений. Надписи, поясняющие содержимое текстовых окон, получают соответственно наименования **Label1**, **Label2** и **Label3**. Для этих надписей также задаем размер шрифта 16 и его полужирное начертание.

Экранная кнопка «**Расчет**», используемая для выполнения расчетов получает имя **Button1**, кнопка «**Сброс**» - **Button2**, кнопка «**Выход**» - **Button3**. Для каждой экранной кнопки также производим начальную настройку шрифта, делая его размер равным 16 и начертание полужирным. На этом работа над созданием пользовательского интерфейса приложения завершена.

На рис. 5.1 показан пользовательский интерфейс данного приложения.

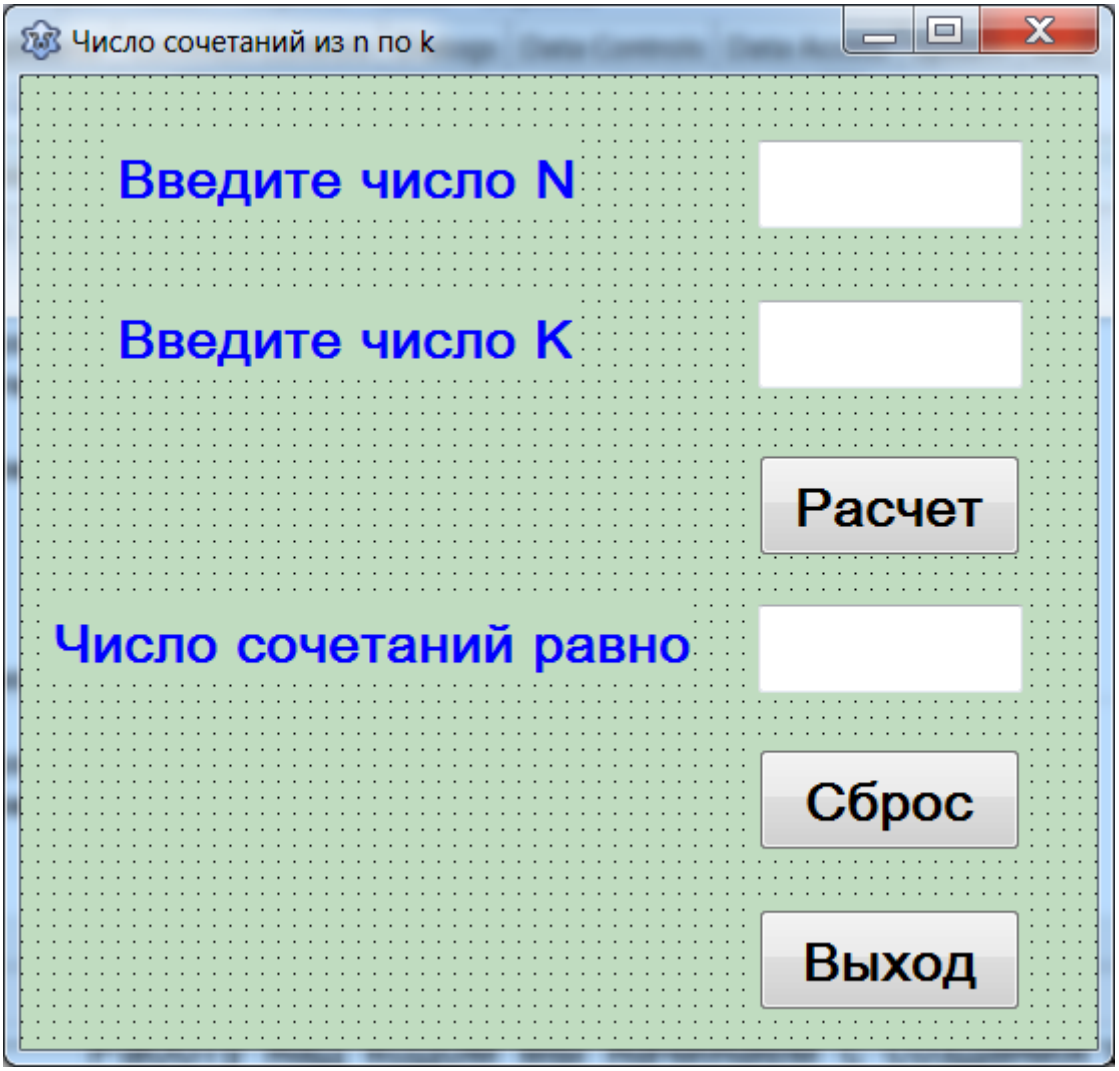

Рис. 5.1. Пользовательский интерфейс приложения «Определение числа сочетаний из n по k»

Следующим этапом является написание программного кода приложения.

Работу над кодом мы начинаем с создания описания функции, которая вычисляет факториал натурального числа. Данное описание мы размещаем в начале раздела программного модуля **implementation** после управляющего комментария. Это описание выглядит следующим образом:

```
function faktor(t:integer):longint;
var i:integer;
f:longint;
begin
f:=1;
for i:=1 to t do
f:=f*i;
faktor:=f;
end;
```

Рассмотрим составные части этого описания. Оно начинается с заголовка функции, в котором указывается имя функции – **faktor**. Затем в круглых скобках указывается параметр функции – натуральное число **t**, которое описывается как переменная типа **integer**. После круглых скобок указывается тип значения самой функции – **longint**.

Затем идет описание локальных переменных, которые используются внутри функции. Таких переменных в данной функции две. Это переменная **f**, которая содержит значение вычисляемого факториала и переменная цикла **i.**

Следующим разделом функции является тело функции, которое находится между ключевыми словами **begin** и **end**. Первым оператором является оператор присваивания. В результате его выполнения переменная **f** получает начальное значение, равное единице. Затем в цикле с заранее известным числом повторений производится вычисления значения факториала путем последовательного умножения начального значения f на все натуральные числа от 1 до **t** включительно. В последнем операторе тела функции полученное значение переменной **f** (то есть значение факториала) присваивается самой функции.

Затем создаем процедуру **Button1Click** для экранной кнопки «**Расчет**». Ниже приводится программный код этой процедуры.

```
procedure TForm1.Button1Click(Sender: TObject);
var n,k:integer;
    c:real;
begin
   n:=StrToInt(Edit1.Text);
   k:=StrToInt(Edit2.Text);
   c:=faktor(n)/(faktor(n-k)*faktor(k));
   Edit3.Text:=FloatToStr(c);
end;
```

В начале данной процедуры производится преобразование данных, вводимых в текстовые окна **Edit1** и **Edit2**, из текстовой формы в целочисленную, то есть значения переменных **n** и **k**.

Затем происходит вычисление значения переменной **c** (числа сочетаний из **n** по **k**) с использованием приведенной выше формулы. В этой формуле трижды происходит вычисление факториала различных величин. Каждое из этих вычислений осуществляется с помощью запрограммированной ранее функции **faktor**. Каждый раз при очередном вызове функции формальный параметр **t** заменяется на фактический **n**, **k** и **n-k**. В последнем случае в качестве параметра указывается не одна переменная, а выражение, содержащее две переменные. В этом случае при вызове функции сначала

вычисляется значение выражения, а затем полученный результат используется как параметр функции.

Затем происходит вычисление значения функции, и это значение возвращается в вызывающую процедуру **Button1.Click** для использования в вычислениях. Далее полученное таким образом значение переменной с преобразуется из вещественной формы в строковую и выводится в текстовое окно **Edit3**.

Процедуры **Button2Click** для кнопки "**Сброс**" и **Button3Click** для кнопки "**Выход**" программируются аналогично тому, как это было сделано в предыдущих проектах. Ниже приводится полный программный код данного приложения.

```
unit Unit1;

{$mode objfpc}{$H+}

interface
uses
  Classes, SysUtils, FileUtil, Forms, Controls,
Graphics, Dialogs, StdCtrls;

Type

  { TForm1 }

  TForm1 = class(TForm)
    Button1: TButton;
    Button2: TButton;
    Button3: TButton;
    Edit1: TEdit;
    Edit2: TEdit;
    Edit3: TEdit;
    Label1: TLabel;
    Label2: TLabel;
    Label3: 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 *.lfm}

{ TForm1 }

function faktor(t:integer):longint;
var i:integer;
    f:longint;
begin
  f:=1;
  for i:=1 to t do
  f:=f*i;
  faktor:=f;
end;

procedure TForm1.Button3Click(Sender: TObject);
begin
  Close;
end;

procedure TForm1.Button1Click(Sender: TObject);
var n,k:integer;
    c:real;
begin
   n:=StrToInt(Edit1.Text);
   k:=StrToInt(Edit2.Text);
   c:=faktor(n)/(faktor(n-k)*faktor(k));
   Edit3.Text:=FloatToStr(c);
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
  Edit1.Text:='';
  Edit2.Text:='';
  Edit3.Text:='';
end;

end.
```

После завершения написания программного кода и сохранения всех изменений в приложении его можно запускать на выполнение. На рис. 5.2 показан экран приложения в рабочем режиме.

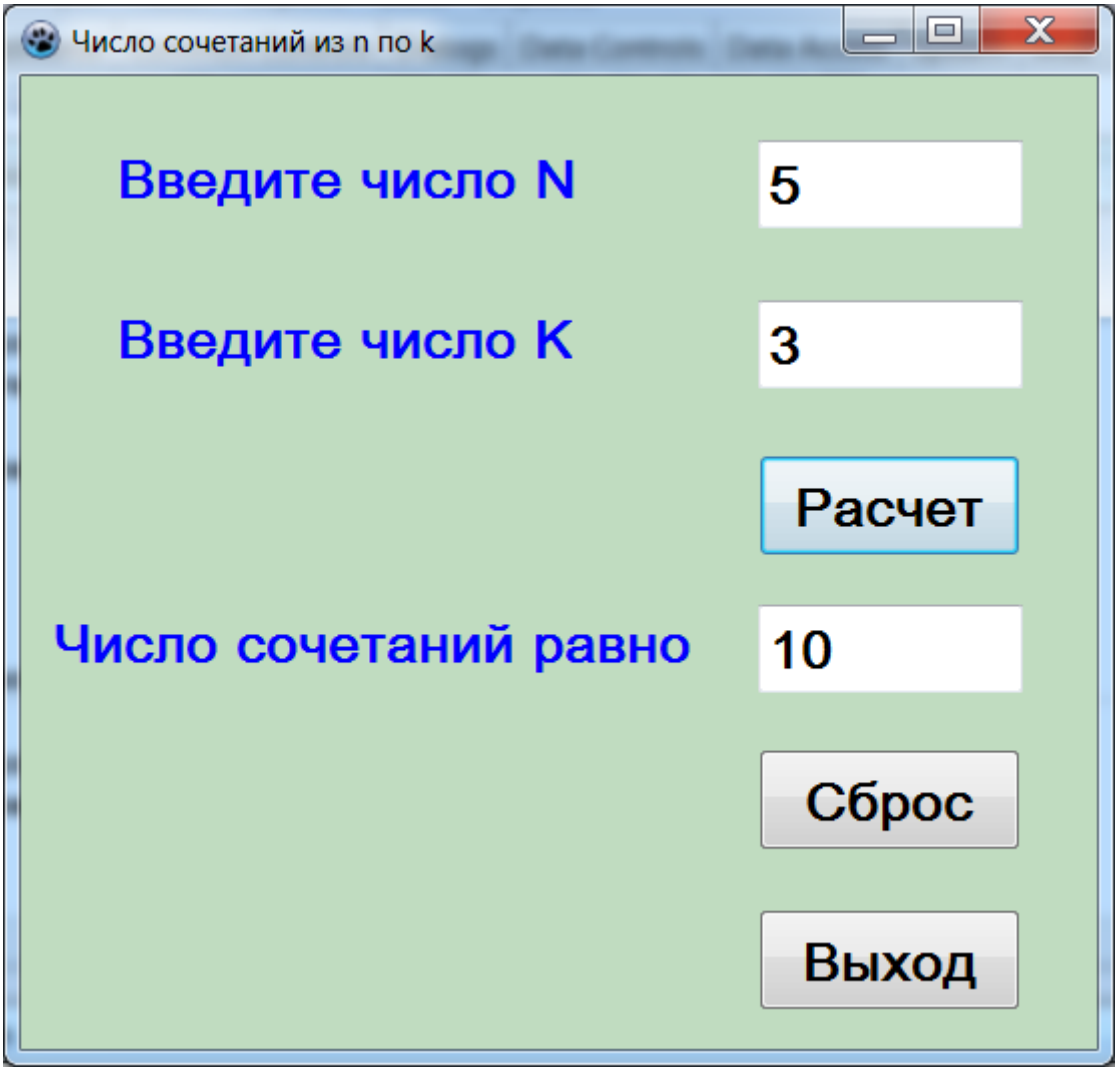

Рис. 5.2. Экран приложения «число сочетаний из n по k» в рабочем режиме

Контрольные вопросы

1. В чем заключается основное различие между стандартными и авторскими функциями и процедурами?

2. Из каких основных элементов состоит описание функции в Object Pascal?

3. Из каких основных элементов состоит описание процедуры в Object Pascal?

4. В чем заключается различие между локальными и глобальными переменными?

5. Что представляют собой операторные скобки, в которые заключается тело функции?

6. Может ли существовать функция, у которой отсутствуют параметры?

7. Может ли существовать процедура, у которой отсутствует значение?

8. В чем состоит основное различие между параметрами-значениями и параметрами-переменными?

9. Укажите основные правила наименования авторских функций и процедур.

10. Из каких основных элементов состоит вызов функции?

11. Что представляют собой формальные параметры функции?

12. Что представляют собой фактические параметры функции?

Задание для самостоятельной работы

Разработать приложение, которое вычисляет значение следующего выражения:

$$(a^x+b^y)^z$$

В приложении необходимо создать авторскую функцию для возведения числа в произвольную степень.

Примерный вид пользовательского интерфейса приложения приведен на рис. 5.3.

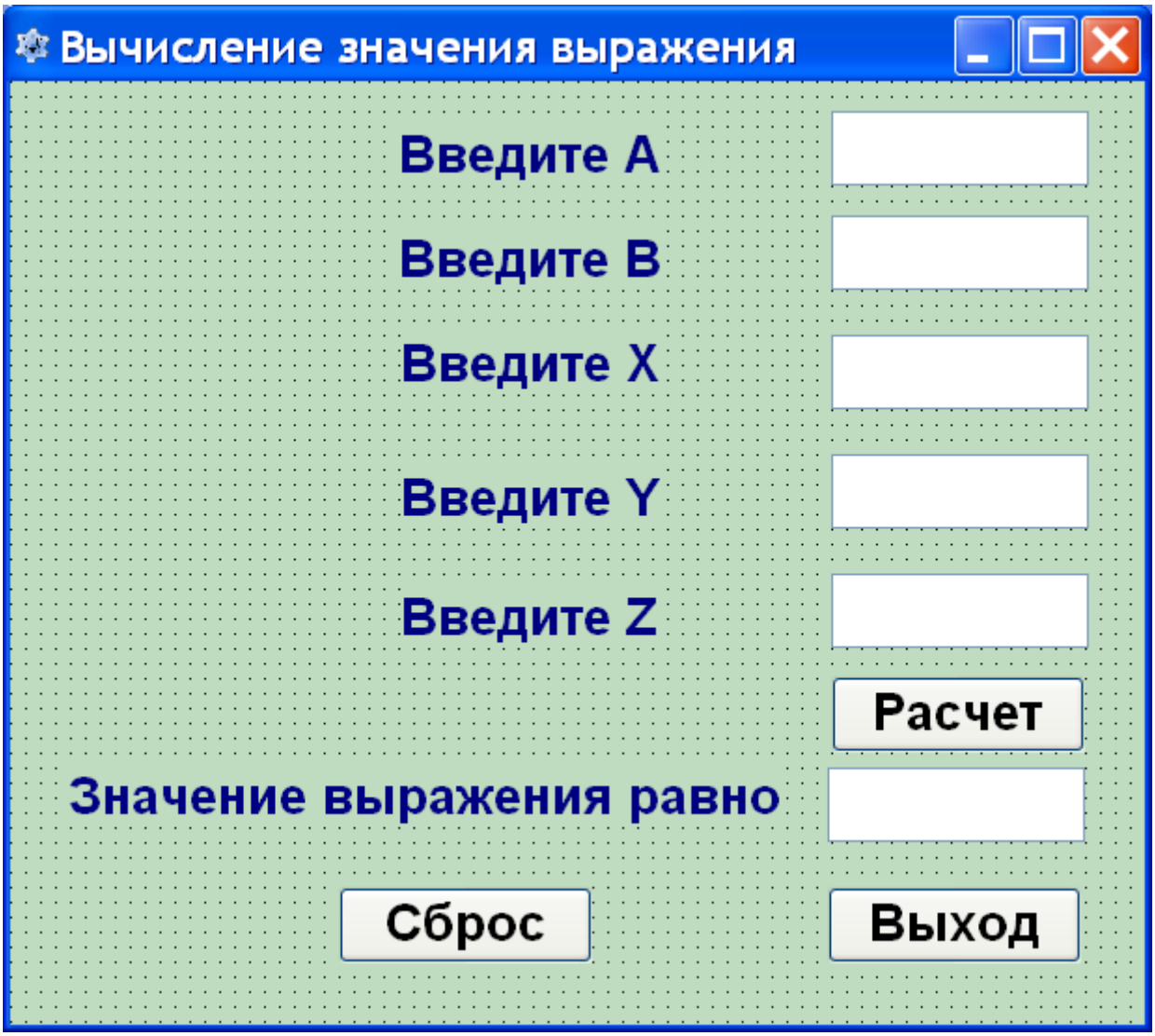

Рис. 5.3. Примерный вид пользовательского интерфейса приложения «Вычисление значения выражения»

Заключение

В данном учебном пособии были рассмотрены основные, базовые принципы программирования в среде Lazarus на языке Object Pascal. На конкретных примерах показан процесс создания приложений от разработки пользовательского интерфейса и написания программного кода до тестирования приложений.

Для лучшего усвоения материала в конце каждой главы приведены задания для самостоятельного выполнения и контрольные вопросы. Выполнение этих заданий и поиск ответов на вопросы позволит как закрепить теоретические знания, так и получить практические навыки в разработке полноценных приложений в среде Lazarus.

Автор данного пособия надеется, что освоение обучающимися изложенных в нем материалов станет необходимым этапом в формировании компетенций специалистов в области информационных технологий и программирования.

Библиографический список

1. Алексеев Е. Р., Чеснокова О. В., Кучер Т. В. Free Pascal и Lazarus. – Саратов: Профобразование, 2017.

2. Ачкасов В. Ю. Программирование на Lazarus. – Москва: Интернет-Университет Информационных Технологий (ИНТУИТ), 2016.

3. Новиков П. В. Объектно-ориентированное программирование. – Саратов: Вузовское образование, 2017.

4. Пестриков В.М., Маслобоев А.Н. Delphi на примерах. – СПб.: БХВ-Петербург, 2005.

5. Пестриков В.М., Маслобоев А.Н., Федоров О.К. Основы программирования в системе Borland Delphi: учебное пособие / СПбГТУРП, СПб., 2004.

6. Пестриков В.М., Маслобоев А.Н. Программирование на языке Object Pascal: учебно- методическое пособие. М-во образования и науки РФ. – СПб.: СПбГТУРП, 2014.

Учебное издание

Артур Николаевич Маслобоев

Языки и методы программирования

Основы программирования в среде Lazarus

Учебное пособие

Редактор и корректор В.А. Басова

Техн. редактор Л.Я. Титова Темплан 2020 г., поз.128

Подп. к печати 24.12.2020 г. Формат 60x84/16. Бумага тип. № 1.

Печать офсетная. Объем 5,5 печ.л.; 5,5 уч.-изд. л., Тираж 50 экз. Изд. № 128. Цена «С». Заказ

Ризограф Высшей школы технологии и энергетики СПбГУПТД, 198095, СПб., ул. Ивана Черных, 4.