

82.9
П58

В.Б. Попов

Delphi

ДЛЯ ШКОЛЬНИКОВ



В.Б.Попов

Delphi

для школьников

Учебно-методическое пособие



МОСКВА
"ФИНАНСЫ И СТАТИСТИКА"



ИЗДАТЕЛЬСКИЙ ДОМ
"ИНФРА-М"

2010

РЕЦЕНЗЕНТ:

В.П. Грибанов,
кандидат экономических наук, доцент, профессор кафедры
«Математическое обеспечение
и администрирование информационных систем»
Московского государственного университета
экономики, статистики, информатики (МЭСИ)

Попов В.Б.

П58 Delphi для школьников: учебно-методическое пособие / В.Б. Попов. –
М.: Финансы и статистика; ИНФРА-М, 2010. – 320 с.:ил.
ISBN 978-5-279-03470-3 (Финансы и статистика)
ISBN 978-5-16-004256-5 (ИНФРА-М)

Последовательно излагаются основные принципы структурного и объектно-ориентированного программирования на языке Objekt Pascal в интегрированной среде программирования Delphi. Изложение материала дается на основе примеров рабочих программ. Для проверки усвоения теоретического материала в конце каждой главы имеются контрольные вопросы. Выполнение заданий по разработке приложений поможет сформировать прочные навыки программирования.

Для учащихся общеобразовательных, средних и высших учебных заведений, может быть использовано для самообразования.

П 2404000000 - 007
010(01) - 2010 без объявл.

УДК 004.438 Delphi
ББК 32.973.26-018.1я72

ISBN 978-5-279-03470-3
ISBN 978-5-16-004256-5

© Попов В.Б., 2010
© Издательство «Финансы и статистика», 2010

ПРЕДИСЛОВИЕ

Уважаемые читатели, это издание является логическим продолжением книги "Turbo Pascal для школьников"¹. Пособие, которое вы держите в руках, представляет собой курс по изучению основ объектно-ориентированного программирования и созданию простых приложений Windows в среде визуального программирования Delphi. Языком, на котором разрабатываются приложения в среде Delphi, является Object Pascal. Это прежний Pascal, поскольку все его основные конструкции сохранены (они подробно описаны в названной выше книге); коренному преобразованию подверглась лишь важнейшая составная часть языка – модель объектов, что и рассматривается в данном пособии.

Глава 1 книги посвящена изучению основных понятий объектно-ориентированного программирования, глава 2 – освоению интегрированной среды разработки Delphi 6, глава 3 – созданию приложений для обработки строк, массивов и файлов. В главе 4 вы познакомитесь с созданием мультимедийных приложений. В главе 5 подробно разобраны примеры создания простых приложений: многооконного текстового редактора, приложений для работы с базами данных, а также консольного приложения и приложения, взаимодействующего с внешними программами.

В последней главе 6 дается пример разработки приложения «Школьная библиотека». Разработка приложения представлена в виде 14 упражнений. Их последовательное выполнение позволит вам узнать порядок создания Windows-приложения, обслуживающего базу данных MS Access и взаимодействующего с MS Word и MS Excel, а также научит использовать реестр Windows для хранения параметров приложения, Active Data Objects (ADO) – для связи приложения с базами данных MS Access, компонент TPageControl – для создания многостраничных панелей с закладками и предоставит возможность создать реально необходимый продукт для вашего учебного заведения.

Материалы приложения раскроют для вас возможности интегрированной среды разработки программ Delphi 6. В конце книги представлен словарь терминов.

В пособии содержится большое количество подробно разобранных примеров программ², которые компилируются без ошибок и корректно исполняются в среде визуального программирования Delphi, начиная с версии 6.

¹ См.: Попов В.Б. Turbo Pascal для школьников. – М.: Финансы и статистика; ИНФРА-М, 2010.

² Все приведенные примеры программ можно найти в Интернете по адресу: <http://www.vb-popov.narod.ru>.

В конце каждой главы подводятся итоги того, чему, по предположению автора, должны научиться учащиеся, а также приводятся вопросы, которые можно использовать для проверки успешности усвоения теоретического материала, и задания для самостоятельной работы.

При пользовании этим пособием надо учитывать, что каждая его глава является самостоятельным разделом, поэтому при изучении учебного материала возможны различные подходы. Начинающим программистам рекомендуем последовательно изучать теоретический материал каждой главы и только затем переходить к выполнению практических упражнений. Читателям, которые знают эту теорию, следует повторить синтаксические описания (структур данных, операторов, процедур и функций) и проверить прочность знаний, ответив на контрольные вопросы, а затем перейти к выполнению заданий практической части.

Очевидно, что данное пособие не может охватить все детали использования среды визуального программирования Delphi, компонентов VCL (Visual Component Library – библиотека визуальных компонентов для программирования под Windows), поэтому в конце книги приводится список литературы и ссылок на информационные ресурсы в Интернете, которые могут расширить ваши знания по программированию.

Во все общеобразовательные учебные заведения России в рамках проекта «Обеспечение лицензионной поддержки стандартного базового пакета программного обеспечения для использования в общеобразовательных учреждениях Российской Федерации» поставлена среда визуального программирования Delphi 2009. Ознакомительную версию Delphi 2010 можно загрузить с сайта производителя (<https://downloads.embarcadero.com/free/delphi>).

Следует отметить, что при подготовке рукописи данной книги для разработки программ автором использовалась среда программирования Delphi 2009.

Выражаю глубокую признательность всем коллегам, которые помогли выходу данного пособия. Все замечания и пожелания по поводу данного пособия будут приняты автором с благодарностью по адресу: vb-popov@yandex.ru.

ВВЕДЕНИЕ

В развитии языков программирования выделяются два основных направления: процедурное и непроцедурное. В процедурных языках программа явно описывает действия, которые необходимо выполнить, а результат задается способом получения его при помощи некоторой процедуры — определенной последовательности действий. Основными средствами, применяемыми в этих языках, являются величины (в том числе и табличные) присваивания, циклы, процедуры. При построении процедурной программы необходимо ясно представлять, какие действия будут производиться при ее выполнении и в какой последовательности. Среди процедурных языков можно, в свою очередь, выделить структурные и операционные языки. В структурных языках одним оператором записываются целые алгоритмические структуры: ветвления, циклы. В операционных языках для этого используется несколько операций. Широкое распространение получили структурные языки Паскаль, Си, Ада, ПЛ/1 и операционные языки Фортран, Бейсик, Фокал.

Непроцедурное (декларативное) программирование появилось в начале 1970-х гг., но его развитие началось в 1980-е гг. в связи с проектом по созданию компьютеров пятого поколения, целью которого явилась подготовка почвы для создания интеллектуальных машин. К непроцедурному программированию относятся функциональные и логические языки. В функциональных языках программа описывает вычисление некоторой функции. Обычно эта функция задается как композиция других, более простых, те в свою очередь разбиваются на еще более простые и т. д. Один из основных элементов в функциональных языках — рекурсия, т. е. вычисление значения функции через значение этой же функции от других элементов. Наиболее распространенными среди функциональных языков являются Лисп и Рефал. Лисп — язык обработки списков — давно и активно применяется в системах искусственного интеллекта. Рефал — единственный из распространенных в мире языков программирования, разработанный в нашей стране; построенный на алгоритмах Маркова он удобен для обработки текстов. Промежуточное положение занимает язык Лого, который содержит средства и процедурного, и функционального программирования. На начальном уровне он похож на классический процедурный язык, а при решении сложных задач обработки данных на первый план выходят функциональные методы.

Функциональная программа, как и процедурная, описывает действия, которые надо совершить для достижения результата (вызов функции —

это тоже действие), но построение функциональных программ требует скорее математического, чем алгоритмического мышления.

В логической традиции программа вообще не описывает действий. Она задает данные и соотношения между ними. После этого можно задавать вопросы. Машина перебирает известные (заданные в программе) данные и находит ответ на вопрос. Порядок перебора не описывается в программе, а неявно задается самим языком. Классическим языком логического программирования считается Пролог, хотя он и содержит некоторые средства управления перебором, т. е. процедурные элементы. Построение логической программы вообще не требует алгоритмического мышления. Здесь нет динамики, отсутствует описание действий: программа описывает статические отношения объектов, а динамика находится в механизме перебора и скрыта от программиста.

Можно выделить еще один класс языков программирования – объектно-ориентированные языки сверхвысокого уровня. На таких языках подробно не описывают последовательность действий для решения задачи, хотя они содержат элементы процедурного программирования. Объектно-ориентированные языки, благодаря богатому пользовательскому интерфейсу, предлагают человеку решить задачу в удобной для него форме. Примером такого языка может служить язык программирования визуального общения Smalltalk. Трудно провести четкую границу между системами программирования сверхвысокого уровня и прикладным программным обеспечением. Как те, так и другие системы позволяют работать с ними неквалифицированному пользователю, не являющемуся программистом.

Интегрированные среды разработки программ. Трудоемкость создания сложных компьютерных программ и разнообразие средств, используемых в процессе разработки программы для написания кода, компиляции и отладки, привели в 80-х гг. XX в. к осознанию необходимости интеграции этих средств. В те годы фирма Borland International, Inc (США) разработала интегрированную систему Turbo Pascal (integration – объединение отдельных элементов в единое целое), так как она объединяет в себе возможности ранее разрозненных средств, используемых при разработке программ: редактора текстов, компилятора, компоновщика и отладчика. Часто ее кратко называют IDE (Integrated Development Environment – интегрированная среда разработки).

Переход к визуальному программированию. В конце XX в. стандартом интерфейса пользователя компьютерных программ становится графический интерфейс (GUI).

Примечание. GUI (graphical user interface) – графический интерфейс пользователя, использующий окна (windows), значки (icons) и мышь (mouse). Взаимо-

действие пользователя с программой средствами GUI основано на интуитивно понятных принципах, что обеспечивает продуктивное использование компьютера даже неподготовленным пользователем.

GUI предлагает более сложное и дружелюбное окружение пользователя, чем командно-управляемый интерфейс DOS. Однако при разработке приложений с GUI возникали большие трудности программирования, потому что для отображения объектов интерфейса пользователя (меню, окон, кнопок и т. п.) требовалось написание фрагмента кода программы, создающего и настраивающего объект «по месту». Увидеть же закодированные объекты можно было только в ходе исполнения программы. При таком подходе достижение того, чтобы объекты выглядели и вели себя заданным образом, было утомительным процессом, требующим неоднократных исправлений кода с последующим запуском программы и просмотром полученного результата.

В свое время система программирования Turbo Pascal была одной из ведущих систем программирования для операционной системы MS-DOS, но в 1990-е гг. ведущей операционной системой для персональных компьютеров стала OS Windows, и возник вопрос о создании мощной и в то же время достаточно простой и удобной для работы среды создания приложений Windows. С целью преодоления трудностей на этапе создания интерфейса пользователя в конце XX в. широкое распространение получило визуальное программирование. Средства визуального программирования предоставляют программисту возможность наглядным образом конструировать интерфейс, изображая объекты интерфейса на экране на этапе конструирования элементов интерфейса программы с автоматической генерацией соответствующего кода. После того как объект помещен в форму среды визуального программирования, все его атрибуты сразу отображаются в виде кода, который соответствует объекту как единице, исполняемой в ходе работы программы. Благодаря средствам визуальной разработки можно работать с объектами, держа их перед глазами и получая результаты практически сразу. Способность видеть объекты такими, какими они появляются в ходе исполнения программы, снимает необходимость проведения множества операций вручную.

Одним из популярных средств разработки приложений с использованием методов визуального программирования является среда программирования Delphi. Языком, на котором разрабатываются приложения в Delphi, является Object Pascal. Это прежний Pascal, поскольку все его основные конструкции сохранены. Важнейшая составная часть языка — модель объектов — подверглась коренному преобразованию. Причем Delphi, начиная с 4-й версии, наряду с приложениями, использующими

графический пользовательский интерфейс, позволяет создавать консольные приложения.

Примечание. Консольные приложения — это особый вид Windows-приложений: такое приложение не только имеет полный доступ к функциям Win API, но и обеспечивает графический интерфейс и выполняется в текстовом режиме.

Развитие средств разработки приложений идет необычайно высокими темпами. Среда программирования Delphi-2009, которая использовалась автором для разработки программ, обладает развитым языком программирования, быстрым компилятором, обширной библиотекой компонентов, полным набором инструментов редактирования, рефакторинга и отладки. Благодаря этому разработчик получает все необходимое для быстрого и удобного создания приложений любых типов на платформе Windows.

ГЛАВА 1

Введение в объектно-ориентированное программирование

В этой главе вы познакомитесь с основными понятиями объектно-ориентированного программирования.

После изучения главы вы будете знать:

- Основные понятия объектно-ориентированного программирования.
- Порядок использования объектов при визуальном проектировании интерфейса.
- Общую организацию программы в Delphi.
- Особенности архитектуры Windows-приложения, управляемого событиями.
- Основы единого графического интерфейса.

Основные понятия объектно-ориентированного программирования

Объектно-ориентированное программирование (ООП) зародилось в языках программирования Pascal, Ада, Smalltalk, C++. До появления ООП технология создания компьютерных программ базировалась на процедурном программировании, в котором основой программ являлись функции и процедуры, т.е. действия. Как вы имели возможность убедиться в первой части этой книги, программист определял, какие действия и вычисления нужны для решения поставленной задачи, затем описывал эти действия в виде процедур и функций и объединял их в программу. Созданная таким образом компьютерная программа отличалась четким алгоритмом работы – последовательностью действий, направленных на достижение поставленной цели.

Объектно-ориентированное программирование представляет собой отличный от процедурного способ программирования, который напоминает процесс человеческого мышления. В объектно-ориентированном программировании главной отправной точкой при проектировании программы является не процедура, не действие, а объект. Такой подход представляется довольно естественным, поскольку в реальном мире мы имеем дело именно с объектами, взаимодействующими друг с другом. Объектно-ориентированное программирование базируется на трех основных принципах: наследовании, инкапсуляции и полиморфизме. Программа, построенная в соответствии с этими принципами, – это не последовательность операторов, не некий жесткий алгоритм, а совокупность объектов и способов их взаимодействия. Обмен информацией между объектами происходит посредством сообщений. С точки зрения ООП

взаимодействие пользователя с программой – это тоже взаимодействие двух объектов: программы и человека, которые обмениваются друг с другом определенными сообщениями.

Что такое объекты? Весь окружающий нас мир состоит из объектов, предметов живой и неживой природы, которые представляются как единое целое, а отдельные части объектов вступают в сложное взаимодействие друг с другом. При структурном подходе программист обычно разделяет описываемый объект на составные части (структурирует его), стараясь описать свойства отдельных частей, не вдаваясь в подробности их взаимодействия, что, вообще говоря, не является лучшим способом программирования.

Назовем объектом понятие, абстракцию или любой предмет с четко очерченными границами, имеющий смысл в контексте рассматриваемой прикладной проблемы. Например, объектами являются форточка, яблоко, Петр Сидоров, дело № 7461, банковский счет и т. п. Все объекты могут быть отличны один от другого: пусть у нас есть два яблока, имеющие одинаковые цвет, форму, вес и вкус; все равно это – два яблока, а не одно, в чем легко убедиться, съев одно из них (другое останется). Между объектами можно установить отношение тождества: объекты, удовлетворяющие этому отношению, одинаковы (тождественны), как вышеупомянутые яблоки.

Объекты представляют собой высший уровень абстракции данных. Объект можно разделить на части, но тогда он перестанет быть объектом. Отношения частей к целому и взаимоотношения между отдельными частями становятся понятнее тогда, когда части объединяются в единое целое. Этот важный процесс называется *инкапсуляцией*.

Не менее важным является и тот факт, что объекты могут наследовать характеристики и поведение других объектов, называемых *порождающими*, или *родительскими*, объектами (или «предками»). С введением понятия наследования происходит качественный скачок, поскольку наличие механизма наследования является самым существенным различием между обычным программированием на языке Pascal и объектно-ориентированным программированием в Delphi.

Классы объектов. В Object Pascal наряду с понятием объекта широко используется понятие класса. Классом называют особую структуру, которая может иметь в своем составе поля, методы и свойства. Класс выступает в качестве объектного типа данных, а объект – это конкретный, существующий в памяти компьютера, экземпляр класса. Например, два яблока из предыдущего примера принадлежат одному и тому же классу объектов (именно с этим связана их одинаковость). Цвет, форма, вес и вкус яблока – это его атрибуты, совокупность атрибутов и их значений (например, красное, овальное, стограммовое, кисло-сладкое) характеризуют объект. Объединение объектов в классы определяется семантикой (смыслом). Например, объекты «конюшня» и «лошадь» могут иметь одинаковые атрибуты: цену и возраст. При этом они могут относиться к одному классу, если рассматриваются в задаче просто как товар, либо к разным классам, что более естественно. Объединение объектов в классы позволяет ввести в задачу абстракцию и рассмотреть ее в более общей постановке. Класс

имеет имя (например, «лошадь»), которое относится ко всем объектам этого класса. Кроме того, в классе вводятся имена атрибутов, которые определены для объектов. В этом смысле описание класса аналогично описанию типа структуры (записи); при этом каждый объект имеет тот же смысл, что и экземпляр структуры (переменная или константа соответствующего типа).

Классы имеют поля (как тип данных `Record`), свойства (напоминающие поля, но имеющие дополнительные описатели, определяющие механизмы записи и считывания данных, что позволяет повысить строгость декларирования внутренней структуры класса) и методы (подпрограммы, которые обрабатывают поля и свойства класса).

Базовым классом для всех объектов в Delphi является класс `TObject`. Он инкапсулирует основные функции, свойственные всем объектам Delphi. Все классы в Delphi являются прямыми или косвенными наследниками `TObject`. Прямое наследование используется только при объявлении простых классов, объекты которых не являются компонентами, не могут присваиваться друг другу и не участвуют в операциях обмена с потоками. Большинство составляют классы, являющиеся косвенными наследниками `TObject` и прямыми наследниками промежуточных классов.

Примечание. Если при объявлении нового типа объектов не указывается класс-предок, то предком нового класса считается класс `TObject`.

Иерархия объектов класса. Обычно, классифицируя некоторый объект, мы задаем следующие вопросы: чем этот объект похож на другие объекты общего класса и чем он отличается от других объектов? Каждый конкретный класс имеет свои особенности поведения и характеристик, определяющих этот класс. Например, класс геометрических фигур можно разделить на два подкласса: плоские фигуры и объемные фигуры. Плоские фигуры могут иметь вершины и не иметь их. Плоскими фигурами, не имеющими вершин, являются окружности и эллипсы (рис. 1.1).

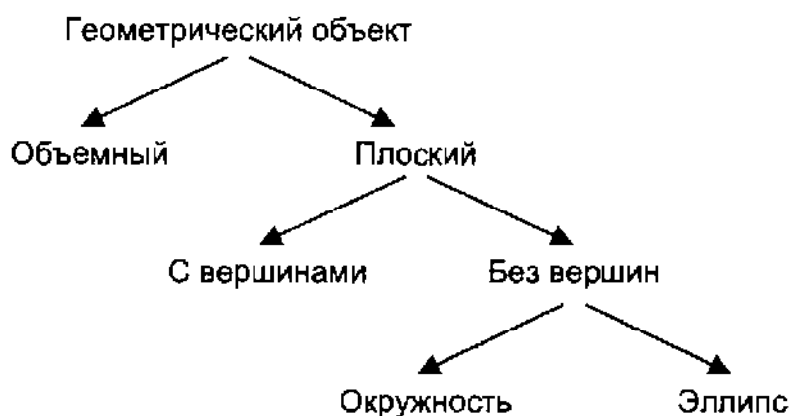


Рис. 1.1. Фрагмент графического представления иерархии объектов класса

Задавая себе приведенные вопросы, программист продвигается от вершины иерархического дерева класса и проходит по дочерним подклассам. Наивысший уровень – самый общий, а вопросы здесь самые простые, например являются ли

фигуры плоскими или объемными. Каждый последующий уровень является более специфическим, чем предыдущий, и менее общим. На самом последнем уровне программист определит цвет, стиль заполнения, радиус окружности и другие конкретные детали фигур.

При использовании ООП следует помнить, что если характеристика объекта определена, то все категории, расположенные ниже данного определения, тоже будут содержать эту характеристику. Поэтому если определена окружность, то нет необходимости узнавать, сколько у нее вершин, так как она относится к подклассу фигур, не имеющих вершин. Объектно-ориентированное программирование является наилучшим инструментом для построения иерархических деревьев структур данных. Одной из важных особенностей, которые объектно-ориентированное программирование добавляет к традиционным языкам типа Pascal, является механизм, с помощью которого типы данных могут наследовать характеристики более простых, т.е. более общих типов. Этим механизмом является наследование.

Наследование. В терминах языка Pascal объект наиболее схож с типом `Record`¹, который является структурированным типом для объединения нескольких связанных элементов под одним именем.

Предположим, что требуется написать программу, подсчитывающую размер стипендий и заработной платы в институте или университете. Переменная `TPerson`, содержащая данные об именах студентов или сотрудников, дате и размере выплаты, могла бы выглядеть следующим образом:

```
TPerson = Record
    Name : String[30];
    Date : String[10];
    Rate : Real;
end;
```

Каждое значение, присвоенное переменной `TPerson`, является экземпляром типа `record`.

Примечание. В дальнейшем термин «экземпляр» будет часто использоваться при описании программ, созданных объектно-ориентированным методом.

`TPerson` представляет два уровня абстракции. Можно рассматривать поля `Name`, `Date` и `Rate` по отдельности, а когда речь идет о полях, работающих одновременно для описания конкретного человека, можно рассматривать их совокупность как `TPerson`.

Предположим, что программа должна учитывать выплаты денег студентам, преподавателям и сотрудникам кафедры. В каждой группе выплаты производятся особым способом. Можно создать другой тип записи для каждой группы.

¹ Примеры использования типа `Record` рассмотрены в гл.8 книги В.Б. Попова «Turbo Pascal для школьников» – М.: Финансы и статистика; ИНФРА-М, 2010.

Например, для получения данных о том, сколько денег должен получить студент, необходимо в первую очередь знать его средний балл. Можно построить запись TStudent вида

```
TStudent = Record
    Name : String[30];
    Date : String[10];
    Rate : Real;
    Ball : Real;
end;
```

Однако можно сохранить тип TPerson путем создания поля Student типа TPerson внутри типа TStudent:

```
TStudent = Record
    Student : TPerson;
    Ball : Real;
end;
```

Такая конструкция удобна и проста, поэтому постоянно используется в программировании. Но она не учитывает специфику данных, обрабатываемых в программе. Необходимо учесть, что выплата денег студентам отличается от выплат другим лицам. Естественно, что студент, как и преподаватель или другой служащий кафедры, имеет имя, фамилию, год рождения и ему полагается определенная сумма денег.

Для студента тип TStudent должен содержать все поля, которые имеются в записи TPerson, при этом тип TStudent является типом-потомком для типа TPerson. TStudent наследует все, что принадлежит TPerson, и, кроме того, содержит новые поля, которые делают TStudent уникальным.

Процесс, с помощью которого один тип наследует характеристики другого типа, называется *наследованием*. В Object Pascal все классы являются потомками класса TObject. В данном примере наследующий тип (TStudent) называется порожденным типом, или потомком, а тип, который наследуется (TPerson), называется *родительским типом*, или *предком*.

Итак, мы познакомились с новой категорией структуры данных, связанной с записями и имеющей механизм наследования. Типы данных в этой новой категории определяются с помощью зарезервированного слова *object*. Объектный тип может быть определен как полный, самостоятельный тип, подобно описанию записей в языке Pascal, но может определяться и как потомок существующего типа объекта указанием после зарезервированного слова *object* имени родительского типа, заключенного в скобки.

В приводимом примере два связанных типа объектов могли бы определяться следующим образом:

```
type
TPerson = Object
    Name : String[30];
    Date : String[10];
    Rate : Real;
end;
```

Здесь TPerson является родительским типом, а TStudent – дочерним. Фактически процесс наследования может продолжаться сколь угодно долго. Для дочернего типа TStudent при необходимости можно определить еще один дочерний тип. Все типы, наследующие тип TPerson, называются его *дочерними типами*, но TStudent является непосредственным дочерним типом для TPerson, а TPerson является непосредственным родителем типа TStudent.

Основное различие между переменными типа object и record состоит в том, что использование зарезервированного слова object при создании объектов в Pascal позволяет не указывать явно поля Name, Date и Rate типа TPerson в типе TStudent, при этом тип TStudent будет содержать эти поля благодаря свойству наследования.

Поля объектов. К полю объекта можно обратиться, как и к полю обычной записи, с помощью оператора with либо используя префикс с именем объекта, например:

```
Student.Ball := 4.5;
with Student do begin
    Name := 'Иванов Николай Петрович';
    Date := '25-06-1995';
end;
```

Даже если поля Name, Date и Rate не являются частью описания типа TStudent, благодаря тому, что они наследуются от типа TPerson, на них можно ссылаться как на описанные в TStudent:

```
Student.Name := 'Николай Иванов';
```

Как правило, объект – это сложная конструкция, поведение и свойства составных частей которой находятся во взаимодействии. К полям объекта можно обратиться непосредственно, но лучше этого избегать. ООП требует, чтобы поля объектов были исключены из исходного кода, насколько это возможно. Поля метода можно объявить скрытыми, ограничив возможность доступа к ним пределами модуля, в котором они определены. Для обеспечения надежности функционирования программы применяется правило инкапсуляции: исключение прямого доступа к полям объекта посредством непосредственных операций чтения и обновления их содержимого; доступ к полям объекта осуществляется посредством вызова соответствующих методов.

Наиболее рациональным способом получения доступа к полям данных в языке Pascal является использование процедур или функций, описанных

Операции и методы

Функция (или преобразование), которую можно применять к объектам данного класса, называется *операцией*. Примеры операций: проверить, снять, поместить (для объектов класса «счет»), открыть для чтения, читать, закрыть (для объектов класса «файл») и т. п.

Примечание. Все объекты одного класса используют один и тот же экземпляр каждой операции (т.е. увеличение количества объектов некоторого класса не приводит к увеличению количества загруженного программного кода). Объект, из которого вызвана операция, передается ей в качестве неявного аргумента (параметра).

Одна и та же операция может, вообще говоря, применяться к объектам разных классов. Такая операция называется *полиморфной*, так как она может иметь разные формы для разных классов. Например, для объектов классов «вектор» и «комплексное число» можно определить операцию «+»; эта операция будет полиморфной, так как для сложения векторов и сложения комплексных чисел выполняются разные действия.

Методы. Инициализация полей объектов. Обычно при работе с записями возникает проблема инициализации полей записи. Предположим, имеется следующая структура:

```
TPerson = Object
    Name : String[30];
    Date : String[10];
    Rate : Real;
end;
```

Для присвоения полям Name, Date и Rate начальных значений можно использовать оператор with, но при необходимости инициализировать более одной записи типа TPerson придется использовать большое число операторов with, которые будут выполнять одни и те же действия. Поэтому естественным является создание инициализирующей процедуры, которая обобщает применение оператора with к любому экземпляру типа TPerson, передаваемому в качестве параметра:

```
procedure Init(var Person:TPerson; Nm, Dt: String; Rt: Real);
begin
    with Person do begin
        Name := Nm;
        Date := Dt;
        Rate := Rt;
    end;
end;
```

Процедура Init, включенная в объект специально для обслуживания типа TPerson, называется методом, т.е. метод – это процедура или функция, включенная в объект таким образом, что экземпляр данного типа становится доступным для нее изнутри. В определение типа включается только заголовок

метода. При определении метода он дополнительно идентифицируется именем типа. Поля и методы являются двумя составными частями новой структуры, называемой объектом. С учетом сказанного объект TPerson можно описать следующим образом:

```

type
  TPerson = Object
    Name : String[30];
    Date : String[10];
    Rate : Real;
    procedure Init(Nm,Dt: String; Rt: Real);
end;
procedure TPerson.Init(Nm,Dt:String; Rt:Real);
begin
    Name:=Nm; {Поле Name объекта TStudent}
    Date:=Dt; {Поле Date объекта TStudent}
    Rate:=Rt; {Поле Rate объекта TStudent}
end;
```

Таким образом, каждой операции соответствует метод – реализация этой операции для объектов данного класса. Можно сказать, что операция – это спецификация метода, а метод – реализация операции. Например, в классе «файл» может быть определена операция печати (print). Эта операция может быть реализована разными методами: печать двоичного файла; печать текстового файла и др. Логически эти методы выполняют одну и ту же операцию, хотя реализуются они разными фрагментами кода.

Каждая операция имеет один неявный аргумент – объект, к которому она применяется. Кроме того, операция может иметь и другие аргументы, иначе называемые *параметрами*. Эти дополнительные аргументы параметризуют операцию, но не связаны с выбором метода. Метод связан только с классом и объектом.

Теперь для инициализации экземпляра типа TStudent достаточно просто вызвать его метод:

```

var
  Person : TPerson;
  Person.Init('Николай Иванов' , '25-06-1995 ' , 400000);
```

Определение методов. Процесс определения методов объектов напоминает создание модулей в Turbo Pascal. Внутри объекта метод определяется заголовком процедуры или функции, действующей как метод:

```

type
  TPerson = Object
```

```
Name : String[30];  
Date : String[10];  
Rate : Real;  
procedure Init(Nm,Dt: String; Rt: Real);  
function GetName : String;  
function GetDate : String;  
function GetRate : Real;
```

```
end;
```

Поля данных должны быть объявлены перед объявлением методов. Как и в интерфейсной части модуля, описание методов внутри объекта только указывает действия, но не определяет, каким образом они будут выполнены. Сами методы описываются вне определения объекта как отдельная процедура или функция. При определении метода его имени должно предшествовать имя типа объекта, которому принадлежит данный метод, за которым следует точка:

```
procedure TPerson.Init(Nm, Dt: String; Rt: Real);  
begin  
    Name:=Nm;  
    Date:=Dt;  
    Rate:=Rt;  
end;  
function TPerson.GetName: String;  
begin  
    GetName :=Name;  
end;  
function TPerson.GetDate: String;  
begin  
    GetDate :=Date;  
end;  
function TPerson.GetRate: Real;  
begin  
    GetRate :=Rate;  
end;
```

В программе при необходимости можно определить уже существующую функцию, например GetName, не связывая ее с типом TPerson.

Свойства объекта. Совокупность данных и методов их чтения и записи называется *свойством*. Свойства объектов можно устанавливать в процессе проектирования, а также можно изменять программно во время выполнения программы. В процессе проектирования приложения в среде программирования Delphi можно просматривать значения некоторых из этих данных в окне Инспектора объектов и изменять эти значения.

События и их обработка

Средой взаимодействия объектов являются сообщения, генерируемые в результате наступления различных событий.

Примечание. Событие – это воздействие на объект. Список событий, на которые реагирует конкретный объект, можно посмотреть в Инспекторе объектов на странице событий (Events).

События наступают в результате действий пользователя – перемещения курсора мыши, нажатия кнопок мыши или клавиш на клавиатуре, а также в результате работы самих объектов. В каждом объекте определено множество событий, на которые он может реагировать. В конкретных экземплярах объекта могут быть определены обработчики каких-то из этих событий, которые и определяют реакцию данного экземпляра объекта. К написанию этих обработчиков и сводится основное программирование при разработке графического интерфейса пользователя с помощью Delphi.

Итак, можно определить объект как совокупность свойств и методов, а также событий, на которые он может реагировать. Внешнее управление объектом осуществляется через обработчики событий. Эти обработчики обращаются к методам и свойствам объекта. Начальные значения данных объекта могут задаваться также в процессе проектирования – установкой различных свойств. В результате выполнения методов объекта могут генерироваться новые события, воспринимаемые другими объектами программы или пользователем.

Создание и уничтожение объектов

Конечно, программа, спроектированная на основе ООП, – это не просто какая-то предопределенная совокупность объектов. В процессе работы объекты могут создаваться и уничтожаться. Таким образом, структура программы является динамическим образованием, меняющимся в процессе выполнения. Основная цель создания и уничтожения объектов – рациональное использование ресурсов компьютера, прежде всего памяти. С целью организации динамического распределения памяти во все объекты заложены методы их создания – конструкторы и уничтожения – деструкторы. Конструкторы объектов, которые изначально должны присутствовать в приложении, срабатывают при запуске программы. Деструкторы всех объектов, имеющих в данный момент в приложении, срабатывают при завершении его работы. Но нередко и в процессе выполнения приложения новые объекты (например, новые окна документов) динамически создаются и уничтожаются с помощью их конструкторов и деструкторов.

Добавлять объекты в программу можно двумя способами: вручную, включая в нее соответствующие операторы, или визуальным программированием, используя заготовки – компоненты.

Использование объектов при визуальном проектировании интерфейса

Как было сказано ранее, для преодоления трудностей, возникающих на этапе создания интерфейса пользователя, в конце XX в. широкое распространение получило визуальное программирование, вооружившее программиста наглядными средствами конструирования интерфейса. Рассмотрим процесс визуального программирования на примере Delphi.

Работа производится в интегрированной среде разработки (ИСР, или Integrated development environment, IDE) Delphi. Среда предоставляет программисту формы, на которых размещаются компоненты. На форму с помощью мыши помещаются значки компонентов, имеющихся в библиотеке Delphi.

С помощью простых манипуляций мышью можно изменять размеры и расположение этих компонентов. При этом в процессе проектирования можно постоянно видеть результат – изображение формы и расположенных на ней компонентов. Отпадает необходимость в запуске приложения для проверки того, как выглядит тот или иной компонент окна, и последующего изменения программного кода для подбора наиболее удачных размеров окна и других компонентов.

Результаты проектирования можно видеть, даже не компилируя программу, немедленно после выполнения операции по изменению размеров и положения компонента с помощью мыши. Но достоинства визуального программирования не сводятся только к этому. Самое главное заключается в том, что во время проектирования формы и размещения на ней компонентов редактор кода Delphi автоматически генерирует код программы, включая в нее соответствующие фрагменты, описывающие данный компонент. А затем в соответствующих диалоговых окнах пользователь может изменить заданные по умолчанию значения свойств этих компонентов и при необходимости написать обработчики событий, т.е. проектирование сводится фактически к размещению компонентов на форме, заданию некоторых свойств этих компонентов и написанию при необходимости обработчиков событий.

Компоненты могут быть визуальными, видимыми при работе приложения, и не визуальными, выполняющими некоторые служебные функции. Визуальные компоненты немедленно отображаются на экране в процессе проектирования в таком же виде, в каком их увидит пользователь во время выполнения приложения. Это позволяет очень легко выбрать место их расположения и дизайн – форму, размер, оформление, текст, цвет и т.д. Невизуальные компоненты в процессе проектирования отображаются на форме в виде значка, но пользователю во время выполнения они не видны, хотя и выполняют очень полезную работу.

В библиотеки визуальных компонентов Delphi включено множество типов компонентов, и их число очень быстро растет от версии к версии.

Введение в Object Pascal

Возможности объектно-ориентированного проектирования в Delphi базируются на свойствах языка Object Pascal. Но ориентация приложений Delphi на Object Pascal несколько не сужает возможностей разработчика, так как приложения Delphi могут использовать разработки, сделанные с использованием других языков, включая C++ и даже ассемблер. Далее мы остановимся на том, как с помощью справочной системы Delphi получить подсказку по языку Object Pascal.

Нововведения в Object Pascal. Основные элементы языка Object Pascal ведут свое происхождение от Turbo Pascal и должны быть вам знакомы, поэтому рассмотрим некоторые новые элементы Object Pascal, появившиеся в Delphi версии 6.

Дополнительно к типам данных, имеющихся в Turbo Pascal, в Object Pascal введен новый тип данных `variant`. В переменных типа `variant` могут храниться данные любых типов, за исключением записей, множеств, статических массивов, файлов, классов, ссылок на классы, указателей и `Int64`. Иначе говоря, переменные типа `variant` могут хранить все, кроме структур, указателей и `Int64`. В переменных типа `variant` могут храниться объекты COM и CORBA, а также динамические массивы и массивы `variant`. В Object Pascal Delphi 6 добавлена возможность создания собственных типов `variant`, допускающих перегрузку операций.

В Object Pascal добавлено множество новых функций, подробные сведения о которых можно получить в справочной системе Delphi, выбрав в меню Help команду Delphi Help>Object Pascal Reference.

Общая организация программы в языке Delphi. Программа, создаваемая в среде Delphi в процессе проектирования приложения, основана на модульном принципе. Головная программа получается предельно простой и короткой. Она состоит из объявления списка используемых модулей и нескольких операторов, создающих объекты для необходимых форм и запускающих приложение на выполнение.

Модульность очень важна для создания надежных и относительно легко модифицируемых и сопровождаемых приложений. Четкое соблюдение принципов модульности в сочетании с принципом скрытия информации позволяет модифицировать любые модули, не затрагивая при этом остальных модулей и головную программу.

Все объекты компонентов размещаются в объектах – формах. Для каждой формы, проектируемой в приложении, Delphi создает отдельный модуль. Именно в модулях и осуществляется программирование задачи. В обработчиках событий объектов – форм и компонентов – размещаются описания алгоритмов, которые в основном сводятся к обработке информации, содержащейся в свойствах одних объектов, и заданию по результатам этой обработки свойств других объектов.

Структура файла головной программы приложения Delphi. В процессе проектирования приложения Delphi автоматически создает код головной

программы и отдельных модулей. В модули программист вводит свой код, создавая обработчики различных событий. Но, как правило, не приходится модифицировать головную программу и даже просматривать ее текст. Только в исключительных случаях бывает необходимо что-то изменить в тексте головной программы, сгенерированном Delphi. Головной файл приложения Delphi имеет следующую структуру:

```
program <имя>;  
    {Объявления подключаемых модулей, а также объявления  
    локальных для головного файла типов, классов, констант, переменных,  
    описания локальных функций и переменных}  
  
begin  
    {Операторы тела программы}  
end.
```

В Delphi головной файл обычно не содержит ничего, кроме операторов инициализации приложения, создания форм и запуска приложения. Вся логика программы помещается в файлы модулей.

Типичная головная программа приложения имеет следующий вид:

```
program Project1;  
uses  
    Forms,  
    Unit1 in 'Unit1.pas' {Form1}, Unit2 in 'Unit2.pas' {Form2};  
    {$R *.res}  
    {Здесь вы можете поместить описания констант,  
    переменных, функций, процедур, доступных для использования только  
    в пределах данного файла}  
begin  
    Application.Initialize;  
    Application.CreateForm(TForm1, Form1);  
    Application.CreateForm(TForm2, Form2);  
    Application.Run;  
end.
```

Программа начинается с ключевого слова `program`, после которого указывается имя программы. Оно совпадает с именем файла, в котором был сохранен проект. Это же имя присваивается исполняемому файлу приложения. По умолчанию используется имя `Project1`.

Примечание. Всегда сохраняйте проект под осмысленным именем, изменяя тем самым имя проекта, заданное Delphi по умолчанию. Иначе очень скоро вы запутаетесь в программах `Project1`, находящихся в различных каталогах.

После заголовка в тексте программы располагается предложение

```
uses
```

```
  Forms,
```

```
  Unit1 in 'Unit1.pas' {Form1}, Unit2 in 'Unit2.pas' {Form2};
```

В этом предложении перечисляются модули, загружаемые программой. Первый модуль Forms является системным, а следующие – модулями разработанных программистом форм. Данный пример показывает, что в проекте были созданы две формы с именами Form1 и Form2 в модулях с именами Unit1 и Unit2. Заключенные в фигурные скобки названия форм – это комментарии.

Предложение в головном файле программы может также содержать после имени модуля ключевое слово in, вслед за которым указывается имя файла, в котором содержится модуль, например:

```
uses Unit1 in 'Unit1.pas', Unit2 in 'c:\examples\Unit2.pas';
```

Ключевое слово in используется в случаях, когда необходимо указать Delphi, где следует искать соответствующие файлы.

Примечание. В предложениях uses, включаемых в модули, использование in не допускается.

Следующая строка текста – {\$R *.RES} – представляет собой директиву компилятора, связывающую с исполняемым модулем файлы ресурсов. Она указывает файлы ресурсов (.DFM, .RES), которые должны быть включены в исполняемый модуль или библиотеку. Эти файлы должны быть файлами ресурсов Windows. По умолчанию для файлов ресурсов используется расширение .RES. В процессе компоновки программы или библиотеки файлы, перечисленные в директивах {\$R}, копируются в исполняемый модуль.

Первый оператор в теле программы Application.Initialize; инициализирует приложение, следующие за ним операторы Application.CreateForm(TForm1, Form1); и Application.CreateForm(TForm2, Form2); создают объекты форм Form1 и Form2, последний оператор Application.Run; начинает выполнение приложения.

Если требуется добавить какой-то текст в головную программу, то следует ввести описания необходимых констант, переменных, функций и процедур в место программы, отмеченное соответствующим комментарием в приведенном ранее тексте. Кроме того, можно добавить или изменить операторы в теле программы, например, произвести настройки при запуске приложения на выполнение (скажем, настроить формы на русский язык) или сделать запрос пользовательского ввода и в зависимости от ответа создавать или не создавать те или иные формы. Например, если требуется, чтобы вторая форма приложения Form2 создавалась только в случае, если при запуске приложения в командной строке указана опция Y, то следует заменить приведенный ранее оператор Application.CreateForm(TForm2, Form2); оператором

```
if (ParamStr(1) = 'Y')
```

```
  then Application.CreateForm(TForm2, Form2);
```

Этот оператор анализирует первый параметр командной строки. Если приложение Project1 запущено командой Project1 Y, то будет создана форма Form2. В остальных случаях этой формы не будет.

Примечание. Можно добавить в головной файл и другие операторы, функции, процедуры. Однако делать это не рекомендуется, поскольку такие действия противоречат принципу модульности. Все необходимые в начале выполнения процедуры и функции настройки следует помещать в отдельный модуль без формы.

Обратите внимание на то, что все определенные в головном файле приложения константы, переменные, процедуры, функции, типы будут доступны только в пределах этого головного файла и недоступны в модулях.

Общая структура модуля. Каждый модуль в общем случае имеет следующую структуру:

```
unit <имя модуля>;
interface // Открытый интерфейс модуля
    {Сюда могут помещаться списки подключаемых
модулей, объявления типов, констант, переменных, функций и процедур,
к которым будет доступ из других модулей}
implementation // Реализация модуля
    {Сюда могут помещаться списки подключаемых модулей,
объявления типов, констант, переменных, к которым не будет доступа
из других модулей. Тут же должны быть реализации всех объявленных в
разделе interface функций и процедур, а также могут быть реализации
любых дополнительных, не объявленных ранее функций и процедур}
    initialization {не обязательный}
        <Операторы, выполняемые один раз при первом обращении
к модулю>
    finalization {не обязательный}
        <Операторы, выполняемые при любом завершении работы
модуля>
end.
```

Рассмотрим теперь, как выглядят тексты модулей. Далее приведен текст модуля с пустой формой. Подробные комментарии в этом тексте поясняют, куда и что в этот код можно добавлять.

```
unit Unit1;
interface // Открытый интерфейс модуля
uses    {Список подключаемых модулей}
    Windows, Messages, SysUtils, Variants, Classes, Graphics,
Controls,
    Forms, Dialogs;
type {Объявление класса формы}
```



```

TForm1 = class(TForm)
private // Закрытый раздел класса
    { Private declarations }
    {Объявления переменных, функций и процедур, включаемых в класс
формы, но недоступных для других модулей}
public // Открытый раздел класса
    { Public declarations }
    {Объявления переменных, функций и процедур, включаемых в класс
формы и доступных для других модулей}
end;
var
    Form1: TForm1;
    {Объявления типов, констант, переменных, функций и процедур, к
которым будет доступ из других модулей, но которые не включаются
в класс формы}
implementation // Реализация модуля
    {Сюда могут помещаться предложения uses, объявления типов,
констант, переменных, к которым не будет доступа из других
модулей. Тут же должны быть реализации всех объявленных в разделе
interface функций и процедур, а также могут быть реализации любых
дополнительных, не объявленных ранее функций и процедур}
    {$R *.dfm}
end.

```

Модуль начинается с ключевого слова `unit`, после которого указывается имя модуля. Оно совпадает с именем файла, в котором был сохранен модуль. По умолчанию для первого модуля используется имя `Unit1`, для второго – `Unit2` и т.д.

Примечание. Если в вашем приложении несколько модулей, сохраняйте их файлы с осмысленными именами, изменяя тем самым имена модулей, заданные Delphi по умолчанию.

Текст модуля состоит из двух основных разделов: `interface` – открытый интерфейс модуля и `implementation` – реализация модуля. Все, что помещается непосредственно в раздел `interface` (типы, переменные, константы, функции, процедуры), может быть использовано другими модулями программы. Все, что помещается в раздел `implementation`, является внутренним делом модуля. Внешние модули не могут видеть типы, переменные, константы, функции и процедуры, размещенные в разделе реализации.

В разделе `interface` после предложения `uses`, содержащего список подключаемых модулей, располагается заготовка объявления класса формы, подготовленная Delphi. Имя класса формы – `TForm1`. Класс содержит два раздела: `private` – закрытый раздел класса и `public` – открытый раздел класса. То, что

объявлено в разделе `public`, будет доступно для других классов и модулей. То, что объявлено в разделе `private`, доступно только в пределах данного модуля.

За объявлением класса формы следуют строки

```
var
```

```
Form1: TForm1;
```

Таким образом объявляется переменная `Form1` класса `TForm1`.

Затем следует пока пустой раздел реализации `implementation`, в котором содержится только директива компилятора `{ $R *.dfm }`, обеспечивающая компоновку файлов ресурсов форм.

Примечание. Директиву `{ $R *.dfm }` нельзя удалять из текста модуля, так как в противном случае загрузочный модуль не будет создан и будет сгенерировано исключение `EResNotFound`.

Кроме разделов `interface` и `implementation`, можно добавить в модуль еще два раздела: `initialization` и `finalization`.

Области видимости и доступ к объектам, переменным и функциям модуля

Структура модуля, содержащего объекты и процедуры. Рассмотрим текст кода модуля, обрабатывающего размещенные на форме два компонента: кнопку `Button1` типа `TButton` и метку `Label1` типа `TLabel`. Кроме того, в модуле реализован обработчик события, связанного со щелчком мышью на кнопке, и в разных местах модуля добавлены переменные и функции, чтобы можно было видеть, как получить к ним доступ:

```
unit Unit1;
interface
uses
    Windows, Messages, SysUtils, Classes, Graphics, Controls,
    Forms, Dialogs, StdCtrls;
type
    TForm1 = class(TForm)
        Label1: TLabel;
        Button1: TButton;
        procedure Button1Click(Sender: TObject);
    private
        procedure F1(Ch:char); {Процедура F1 доступна только в
данном модуле}
    public
        {Переменная Ch1 и процедура F2 доступны для объектов любых
классов и для других модулей, но со ссылкой на объект}
```

```

    Ch1:char;
    procedure F2 (Ch:char);
end;
var
    Form1: TForm1;
    {Переменная Ch2 и процедура F3 доступны для объектов любых
классов и для других модулей}
    Ch2:char;
    procedure F3 (Ch:char);
implementation
{$R *.DFM}
uses unit2;
    {Переменная Ch3 и процедура F4 доступны только внутри данного
модуля}
    var Ch3:char;
    procedure F4 (Ch:char);
begin

    Form1.Label1.Caption:=Form1.Label1.Caption+Ch+Form1.Ch1;
end;
procedure TForm1.F1 (Ch:char);
begin
    Label1.Caption:=Label1.Caption+Ch+Ch1;
end;
procedure TForm1.F2 (Ch:char);
begin
    Label1.Caption:=Label1.Caption+Ch+Ch1;
end;
procedure F3 (Ch:char);
begin

Form1.Label1.Caption:=Form1.Label1.Caption+Ch+Form1.Ch1;
end;
procedure TForm1.Button1Click(Sender: TObject);
    {Переменная Ch4 и процедура F5 доступны только внутри данной
процедуры}
    var Ch4:char;
    procedure F5 (Ch:char);
begin
    Label1.Caption :=Label1.Caption+Ch+Ch1;
end;

```

```
begin
  Ch1:='-';
  Ch2:='A';
  Ch3:='B';
  Ch4:='C';
  Label1.Caption:='';
  F1(Ch1);
  F2(Ch2);
  F3(Ch3);
  F4(Ch4);
  F5('D');
  Label1.Font.Color:=clRed;
end;
end.
```

Структура программы является стандартной, но в текст внесены некоторые добавления. В описании класса можно видеть строки

```
Label1: TLabel;
Button1: TButton;
procedure Button1Click(Sender: TObject);
```

Первые две были вставлены редактором кода Delphi автоматически, как только разработчик приложения разместил на форме метку и кнопку. Эти строки означают, что в форме имеются объекты соответствующих типов. Третью из приведенных строк редактор кода Delphi добавил в тот момент, когда разработчик приложения начал реализовывать обработку события OnClick – щелчка на кнопке Button1. Одновременно с записью в описание класса объявления процедуры Delphi занесла в раздел implementation заготовку соответствующей процедуры:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
end;
```

Здесь между резервированными словами begin и end; можно разместить текст обработчика события.

Помимо этой процедуры, в коде присутствует еще несколько одинаковых процедур F1 – F5 и несколько переменных символьного типа Ch1 – Ch4. Рассмотрим работу процедур F1 – F5. У компонента – метки типа TLabel имеется свойство Caption – надпись на метке. Каждая из процедур F1 – F5 берет значение этой надписи, добавляет к ней символ, переданный в нее в качестве параметра Ch, затем добавляет символ, хранящийся в переменной Ch1, и возвращает надпись с этими добавлениями обратно в метку.

Обработчик щелчка мышью по кнопке – процедура TForm1.Button1Click – задает символьным переменным Ch1 – Ch4 значения символов «←», A, B и C,

затем очищает свойство Caption метки Label1, занося в него пустую строку, а затем поочередно обращается к процедурам F1 – F5, передавая в них в качестве параметров различные символы. В заключение надпись метки окрашивается в красный цвет. Для этого используется свойство Font – шрифт объекта Label1. Это свойство само является объектом, имеющим свойство Color – цвет. Значение этого свойства изменяет последний оператор процедуры TForm1.Button1Click.

Доступ к свойствам и методам объектов. Для получения из программы доступа к свойствам и методам объекта следует указать ссылку на него в следующем формате:

<имя объекта>. <имя свойства>

За именем объекта следует без пробела символ точки, а затем тоже без пробела – имя свойства. Например, ссылка на свойство Caption метки Label1 осуществляется при помощи записи Label1.Caption.

Иногда свойство объекта, в свою очередь, является объектом. Тогда при обращении к этому свойству указывается вся цепочка предшествующих объектов, разделенных точкой. Например, метки имеют свойство Font – шрифт, которое, в свою очередь, является объектом. У этого объекта имеется множество свойств, в частности свойство Color – цвет шрифта. Чтобы сослаться на цвет шрифта метки Label1, надо записать Label1.Font.Color, что означает «свойство Color объекта Font, принадлежащего объекту Label1». Аналогичная нотация с точкой используется и для доступа к методам объекта. Например, для метки, как и для большинства других объектов, определен метод Free, который уничтожает метку и освобождает занимаемую ею память. Если в какой-то момент потребовалось уничтожить метку Label1 в приложении, то следует использовать оператор

Label1.Free;

Примечание. Имейте в виду, что если по ошибке далее в программе будет выполняться оператор, обращающийся к свойствам или методам уничтоженного объекта, то это вызовет прерывание программы, поскольку этого объекта уже не будет в памяти. Обращение к объекту, удаленному из памяти методами Destroy или Free, вызывает ошибку и аварийное завершение выполнения программы.

Различия между переменными, функциями и процедурами, включенными и не включенными в описание класса. Теперь посмотрим, чем различаются константы, переменные, функции и процедуры, включенные и не включенные в описание класса. В приведенном ранее примере переменная Ch1 и процедуры F1 и F2 включены в описание класса, а переменные Ch2, Ch3 и процедуры F3 и F4 объявлены вне класса. Различие в их использовании будет заключаться в следующем.

В описании процедур, объявленных в классе, в разделе implementation к имени процедуры должна добавляться ссылка на класс. В нашем примере имена процедур F1, F2 и Button1Click в описании этих процедур в разделе implementation заменяются именами TForm1.F1, TForm1.F2 и TForm1.Button1Click.

К переменным и процедурам, описанным вне класса, обращение происходит просто по их именам, а к переменным и процедурам, описанным в классе, – с указанием имени объекта класса.

Если в приложении создается только один объект данного класса (в нашем примере – только один объект формы класса TForm1), то различие в основном чисто внешнее.

Обращение к переменным и процедурам из процедур, описанных вне класса, выполняется следующим образом: к переменным и процедурам, описанным вне класса, обращение происходит просто по их именам, а к переменным и процедурам, описанным в классе, – с указанием имени объекта класса. Поэтому в нашем примере в процедурах F3 и F4 обращение к переменной Ch1 имеет вид TForm1.Ch1. По той же причине и обращение к свойству Caption объекта Label1 в этих процедурах имеет вид TForm1.Label1.Caption. Внешние по отношению к классу процедуры могут получить доступ к переменным и процедурам, объявленным в классе, только через ссылку на объект TForm1.

Примечание. Необходимость добавления к именам процедур, описанных в классе, и ссылок на класс объясняется просто. Вне класса можно описать другую процедуру с тем же именем, например F1, что и у процедуры класса. Тогда из процедур, не описанных в классе, можно будет ссылаться на обе эти процедуры F1, только на одну из них непосредственно по имени F1, а на другую – через объект класса – TForm1.F1. Благодаря этому при описании процедур вне класса можно даже не знать имен всех процедур, описанных в классе, а сам этот класс может быть описан в другом модуле, текст которого недоступен, но путаницы при этом не возникнет.

Все эти ссылки на объект не требуются для процедур, объявленных в классе. Поэтому в процедурах TForm1.F1, TForm1.F2 и TForm1.Button1Click ссылки на переменную Ch1 и на объект Label1 не содержат дополнительных ссылок на объект формы.

Если в приложении создается несколько объектов одного класса, например несколько форм класса TForm1 (как мы увидим, это часто делается в приложениях с интерфейсом MDI), то проявляются более принципиальные различия между переменными, описанными внутри и вне класса. Переменные, описанные вне класса (в нашем примере это Ch2 и Ch3), так и остаются в одном экземпляре. А переменные, описанные в классе (в нашем примере Ch1), тиражируются столько раз, сколько объектов этого класса будет создано, т.е. в каждом объекте класса TForm1 будет своя переменная Ch1, и все они никак не будут связаны друг с другом. Таким образом, в переменную, описанную внутри класса, можно заносить какую-то информацию, индивидуальную для каждого объекта данного класса. А переменная, описанная в модуле вне описания класса, может хранить только одно значение.

Области видимости переменных и функций. Теперь остановимся на вопросе об областях видимости элементов программы – констант, переменных, функций и процедур, т.е. о связи места их объявления в программе и места их использования. Частично мы уже затрагивали этот вопрос в предыдущем разделе, не упоминая о самом понятии области видимости.

Видимость отдельных элементов модуля, описанного в разделе, поясняется подробными комментариями в тексте этого модуля.

Элементы, объявленные в разделе `interface` модуля вне описания типа, видимы и доступны внутри данного модуля и из внешних модулей. В рассмотренном примере это относится к переменной `Ch2` и процедуре `F3`.

Элементы, объявленные в разделе `implementation` модуля, видимы и доступны внутри данного модуля, но недоступны из внешних модулей. В рассмотренном примере это относится к переменной `Ch3` и процедуре `F4`.

Элементы, объявленные в классе в разделе `private`, видимы и доступны только внутри данного модуля. При этом из процедур, объявленных внутри класса, к ним можно обращаться непосредственно по имени, а из других процедур — только со ссылкой на объект данного класса. В рассмотренном примере это относится к процедуре `F1`. Если в модуле описано несколько классов, то объекты этих классов взаимно видят элементы, описанные в их разделах `private`.

Элементы, объявленные в классе в разделе `public`, видимы и доступны для объектов любых классов и для других модулей. При этом из объектов того же класса к ним можно обращаться непосредственно по имени, а из других объектов и процедур — только со ссылкой на объект данного класса. В рассмотренном примере это относится к переменной `Ch1` и процедуре `F2`.

В классах, помимо обсуждавшихся ранее, могут быть еще разделы `protected` (защищенные). Элементы, объявленные в классе в разделе `protected`, видимы и доступны для любых объектов внутри данного модуля, а также для объектов классов — наследников данного класса в других модулях. Объекты из других модулей, классы которых не являются наследниками данного класса, не видят защищенных элементов.

Элементы, объявленные внутри другой процедуры (в рассмотренном примере это переменная `Ch4` и процедура `F5`, описанные внутри процедуры `TForm1.Button1Click`), являются локальными, т.е. они видимы и доступны только внутри данной процедуры или внутри процедур, вложенных в данную. При этом время жизни переменных, объявленных внутри процедуры, определяется временем выполнения этой процедуры. Так, переменная `Ch4` в нашем примере создается в момент вызова процедуры `TForm1.Button1Click` и уничтожается при завершении работы этой процедуры. Она видима в самой процедуре `TForm1.Button1Click`, и ее может видеть процедура `F5`. Вложенная процедура `F5` доступна только из процедуры `TForm1.Button1Click`, в которой она описана.

Передача параметров в функции. В Object Pascal параметры в процедуры и функции могут передаваться в основном двумя способами: по значению и по ссылке. То, что вы видели в рассмотренном ранее примере, — это передача параметра по значению. В этом случае в заголовке процедуры указываются имя параметра и его тип, например:

```
procedure TForm1.F2(Ch: char);
```

Здесь `Ch` — это локальное имя формального параметра, используемое только внутри данной процедуры. При вызове этой процедуры, который может иметь вид

```
F2(Ch2);
```

в памяти создается временная переменная с именем Ch и в нее копируется значение аргумента Ch2. После этого связь между Ch и Ch2 разрывается. Внутри процедуры можно изменять значение Ch, но это никак не отразится на значении внешней переменной Ch2, указанной в вызове функции в качестве аргумента.

Такая передача параметров по значению имеет свои достоинства и недостатки. Достоинство заключается в том, что процедура не может «испортить» переданный в нее аргумент. Это важно для надежной работы приложения и позволяет разрабатывать процедуру, не задумываясь о том, не использованы ли где-то в программе те же имена параметров. Но у передачи параметра по значению имеется и ряд недостатков. Во-первых, процедура не может изменить значение аргумента, а иногда это очень желательно. Во-вторых, копирование значения аргумента требует дополнительных затрат времени и памяти для хранения копии. И в-третьих, после окончания работы процедуры временная переменная, хранившая значение параметра, уничтожается. Поэтому ее нельзя использовать, например, для накопления какой-то информации.

Для реализации второго способа передачи информации – по ссылке – перед именем параметра в заголовке функции должно быть указано ключевое слово `var`, например:

```
procedure TForm1.F2(var Ch:char) ;
```

В этом случае не происходит копирования значения аргумента в локальную временную переменную в процедуре. Процедура реально работает не с параметром, а со ссылкой – указателем на место хранения аргумента в памяти. Любые изменения параметра Ch, произведенные в процедуре, в действительности относятся не к этому параметру, а к тому аргументу, который передан при вызове процедуры. Таким образом, ключевое слово `var` позволяет возвращать информацию из процедуры в вызвавшую ее внешнюю процедуру.

Дальнейшее знакомство с языком объектно-ориентированного программирования Object Pascal мы продолжим в ходе изучения возможностей среды визуального объектно-ориентированного программирования Delphi 6 при проектировании приложений.

Приложения Windows

Среда разработки программ Delphi ориентирована прежде всего на создание программ для Windows. Приложение Windows является специальным типом программы для персонального компьютера, которая должна:

- иметь исполняемый файл специального формата;
- в общем случае работать в прямоугольном окне на экране;
- быть способной работать одновременно с другими приложениями Windows, включая свои собственные версии;
- уметь осуществлять связь и обмен данными с другими приложениями Windows.

Управляемая событиями архитектура Windows-приложения. Среда Windows основана на архитектуре, управляемой событиями. Это означает, что любой осуществляемый пользователем ввод рассматривается как событие. Состоит ли это событие в нажатии кнопки мыши или в нажатии клавиши на клавиатуре, для Windows это является событием, для которого генерируется соответствующее сообщение.

Независимая от аппаратуры графика. Windows унифицирует процесс вывода информации на экран и принтер и сводит его к единому модулю, называемому *интерфейсом графического устройства* (GDI), который обеспечивает общий интерфейс для всех программ Windows. Windows предлагает драйверы для большинства стандартных графических адаптеров и принтеров, поэтому приложение Windows будет работать без каких-либо изменений на большинстве имеющихся в мире аппаратных средств.

Многозадачный режим. Windows разрешает пользователю одновременно запускать несколько приложений, устраняя необходимость в написании программ, которые после прерывания выполнения остаются резидентными. В Windows не просто реализован многозадачный режим работы, но и обеспечивается ряд возможностей межпроцессорного обмена, таких, как буфер для временного хранения изображения и динамический обмен данными.

В Windows работа нескольких приложений организована таким образом, что каждое приложение вместо полного экрана использует одно или несколько окон. С точки зрения программиста, это означает, что программа выводит текст или графики не в конкретное место экрана, а в окно – некоторую область, ограниченную рамкой. Также приложение должно разделять имеющуюся свободную память и другие приложения.

Управление памятью. В типичном сеансе работы с Windows несколько приложений запускаются на выполнение и заканчивают свою работу по несколько раз, но при этом не происходит загрузки в память каждого приложения после предыдущего, так как в этом случае в системе скоро закончилась бы свободная память. Вместо этого Windows временно перемещает большую часть памяти приложения либо в другую область памяти, либо на диск, чтобы обеспечить работу других приложений или самой операционной системы. Таким образом, приложение Windows должно соответствовать правилам Windows по динамическому управлению памятью и не использовать непосредственного доступа к конкретным адресам памяти. Одно из основных преимуществ менеджера памяти Windows состоит в возможности разделения скомпилированного кода между приложениями. Например, если пользователь запустит два экземпляра одного и того же приложения, то эти два приложения будут использовать один скомпилированный код в памяти. Аналогично приложение может динамически загрузить модуль библиотеки, который будет совместно использоваться различными приложениями.

Ресурсы. Описания интерфейсов устройств приложения Windows для пользователя: меню, блоков диалога, курсоров, значков, растровых изображений, строк и клавиш акселераторов – называются ресурсами. В Windows имеется средство обработки этих описаний за пределами исходного кода приложения.

Ресурсы приложения объединяются с откомпилированным исполняемым файлом до запуска приложения. Для уменьшения используемой памяти приложение загружает свои ресурсы в память только тогда, когда они необходимы.

Выделение спецификации ресурса из исходного кода дает дополнительное преимущество: внешний вид приложения может быть изменен без модификации исходного кода программы. В действительности для модификации ресурсов приложения нет необходимости даже иметь его исходный код. Это значительно упрощает процесс настройки и локализации существующих приложений Windows.

В Delphi предлагается специальное средство для создания и адаптации ресурсов – Image Editor.

Многодокументный интерфейс. Многодокументный интерфейс (Multiple Document Interface, MDI) – это набор пользовательских соглашений относительно создания окон, которые содержат внутри себя дочерние окна. Примером MDI является интегрированная среда разработки Turbo Pascal 7.0.

Примечание. Среда Delphi следует спецификации Single Document Interface – SDI (однодокументный интерфейс) и состоит из нескольких отдельно расположенных окон. В Delphi вы можете одновременно работать только над одним проектом.

Как вы уже могли заметить, программирование в среде Windows является довольно сложной задачей. Эта задача существенно облегчается ООП, позволяющему разработчику приложения сосредоточиться на функциях приложения, а не на его форме. Используя объекты для представления сложных структур, подобных окнам, программа на Delphi может объединять свои операции и хранение данных.

Объектно-ориентированное программирование создает среду, в которой программист использует объекты для представления сложных элементов интерфейса пользователя программы Windows. Например, окно является объектом. Объектно-ориентированное окно и типы объектов приложения управляют обработкой сообщений, которая требуется от программы, что в значительной степени упрощает взаимодействие программиста и пользователя. В действительности объектами являются не только окна, но и блоки диалога и управления (блоки списков и клавиши).

Автоматизация реакции системы в виде сообщений. Кроме инструкций для среды Windows о необходимости проведения определенных действий, большинство приложений должно иметь возможность реагировать на сотни сообщений Windows, которые являются результатом действий пользователя (например, щелчок мышью), а также на сообщения от других приложений и прочих источников. Обработка сообщений и корректная реакция на них служат определяющим фактором правильной работы программы.

Основы единого графического интерфейса. Так как при разработке собственных проектов нам придется реализовывать средства взаимодействия пользователя с приложением, рассмотрим основные элементы, из которых состоит интерфейс пользователя, работающего в среде Microsoft Windows.

Базовым элементом интерфейса является окно. Окна составляют основу пользовательского интерфейса Microsoft Windows и являются основным средством взаимодействия пользователя с прикладной программой. Большинство остальных элементов интерфейса представляют собой частные случаи

окна. **Окно** – это прямоугольная область экрана, выделяемая прикладной программой для осуществления операций ввода-вывода. Окна могут быть перекрывающимися (cascade) и заполняющими экран (tile). На рис. 1.2 показаны основные экранные компоненты приложения Windows.

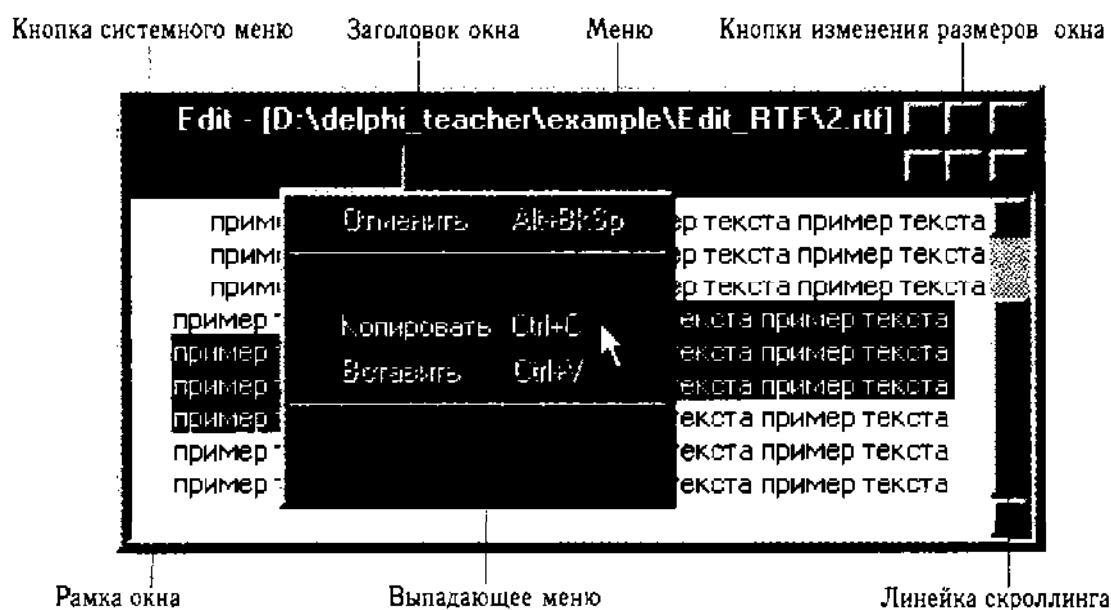


Рис. 1.2. Экранные компоненты приложения Windows

Заголовок окна – это специальная область, обычно содержащая строку текста и располагающаяся в верхней части окна. Заголовок может содержать кнопку системного меню и кнопки управления размерами окна. Системное меню – это меню, наличие которого в окне обозначается специальной кнопкой. В этом меню содержатся команды для перемещения, изменения размеров окна или его закрытия. **Рамка** – прямоугольная окантовка окна, используемая для указания границ окна и их изменения. **Меню** – набор команд, из которых можно сделать выбор с помощью клавиатуры или мыши. Результатом выбора команды меню может быть либо выполнение какого-либо действия, либо появление вложенного меню, также содержащего команды. **Полоса прокрутки** – это вертикальная или горизонтальная полоса с передвигающимся бегунком посередине и кнопками-стрелками по краям. Полосы прокрутки предназначены для позиционирования той части изображения, которая должна быть видна в окне. Окно может содержать в себе дочерние окна. При закрытии основного окна закрываются и его дочерние окна.

Специальный тип окна, используемый для отображения и ввода информации пользователем, называется **диалоговым окном** (рис. 1.3). В большинстве прикладных программ диалоговые окна связаны с определенными командами меню. Особенностью диалоговых окон является отсутствие кнопок изменения размеров. Обычно диалоговые окна содержат одну или несколько кнопок, позволяющих пользователю выполнить определенные действия. Часто в состав панели диалога включаются комбинированные списки, строки редактирования, группы кнопок и другие элементы управления.

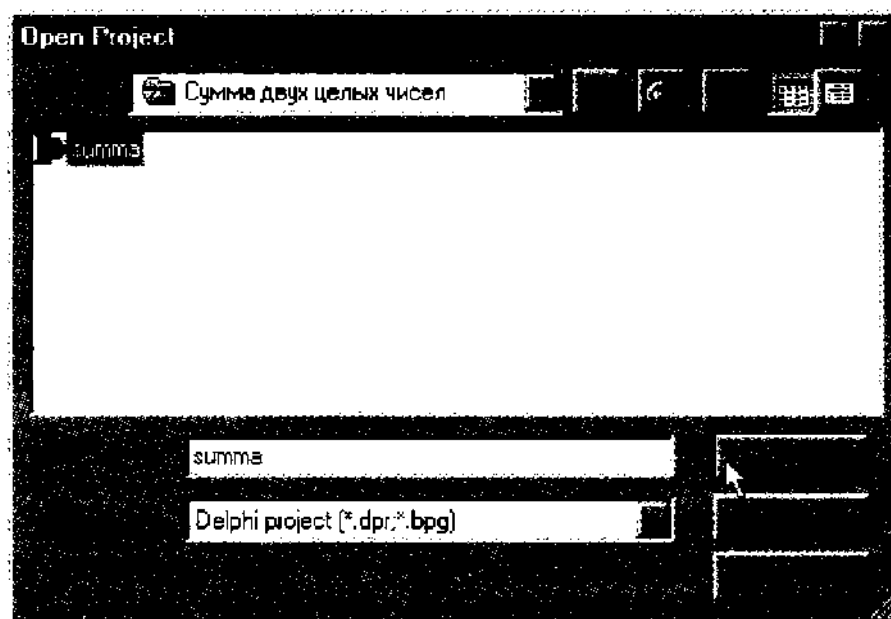


Рис. 1.3. Диалоговое окно

Диалоговые окна бывают *модальными* и *немодальными*. Наиболее часто используются модальные окна. Пользователь, находящийся в модальном окне, для продолжения работы с другими окнами должен нажать на кнопку и закрыть это окно. Эти окна не позволяют пользователю работать с другими окнами, созданными приложением, породившим диалоговое окно, но разрешают переключаться на работу с другими приложениями. Модальные окна отличаются отсутствием полос прокрутки, фиксированным размером и невозможностью минимизации (свертывания в значки). В особых случаях, представляющих угрозу для системы и требующих немедленной реакции оператора, могут использоваться системные модальные окна. Эти окна блокируют работу пользователя с другими окнами вплоть до своего закрытия.

Немодальные диалоговые окна не требуют закрытия для продолжения работы, и пользователь может во время работы с ними свободно переключаться на любое приложение. Например, диалоговое окно Find, позволяющее искать строки символов, может быть немодальным.

Частным случаем модального диалогового окна является *окно сообщений* (рис. 1.4), используемое для вывода информации типа сообщения о статусе операции или об ошибке. Помимо сообщения такая панель может содержать одну или несколько кнопок, позволяющих пользователю обозначить свою реакцию на выводимое на панель сообщение. Чаще всего используются кнопки OK, Cancel, Retry.

Многие окна содержат списки команд, называемые *меню*. Различают *строчное* (горизонтальное) и *вложенное* (вертикальное) меню. Глубина вложенности является довольно большой величиной и позволяет создавать *комплексные меню*. Обычно выбор команды меню приводит либо к выполнению каких-либо действий, либо к появлению вложенного меню или панели диалога.

Для управления окном часто используются кнопки. *Кнопка* – это элемент управления прямоугольной формы, имеющий название и при нажатии посылающий



Рис. 1.4. Окно сообщения

соответствующее сообщение. Обычно одна из букв названия кнопки выделяется специальным образом и служит для имитации нажатия кнопки с клавиатуры. Для выбора одного из нескольких элементов в диалоговом окне часто используются списки. Список – это набор элементов, один или несколько из которых могут быть выбраны. Типичным примером такого набора элементов может служить список имен файлов.

С остальными элементами окна мы познакомимся в процессе разработки приложений.

Архитектура Windows-программы. Программист, впервые взявшись за создание Windows-программы, обнаружит, что этот процесс в значительной степени отличается от процесса создания программы для среды DOS. Архитектура Windows, управляемая событиями, накладывает отпечаток и на структуру самой прикладной программы. Для сравнения представим программу стандартной архитектуры (рис. 1.5) и программу, управляемую событиями (рис. 1.6).

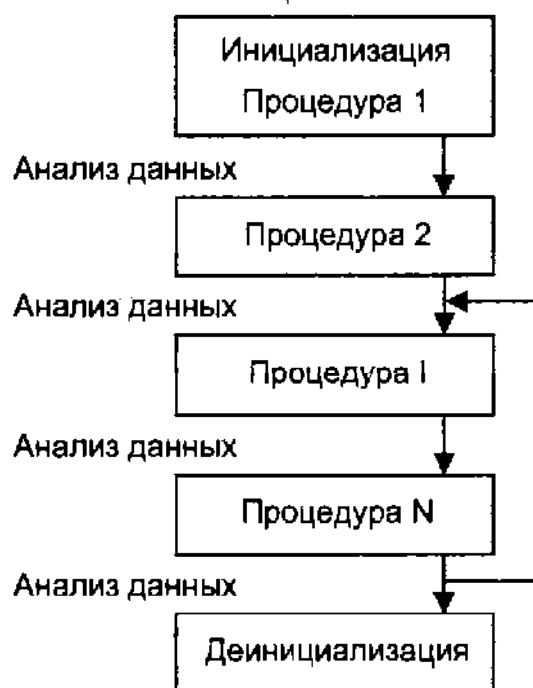


Рис. 1.5. Программа стандартной архитектуры

Как видно из рис. 1.5, в программе стандартной архитектуры каждая процедура обрабатывает некоторые данные, и в результате работы образуются новые данные; на их основе выбирается процедура, которая должна выполняться следующей. В начале и в конце программы определяются действия по инициализации и деинициализации.

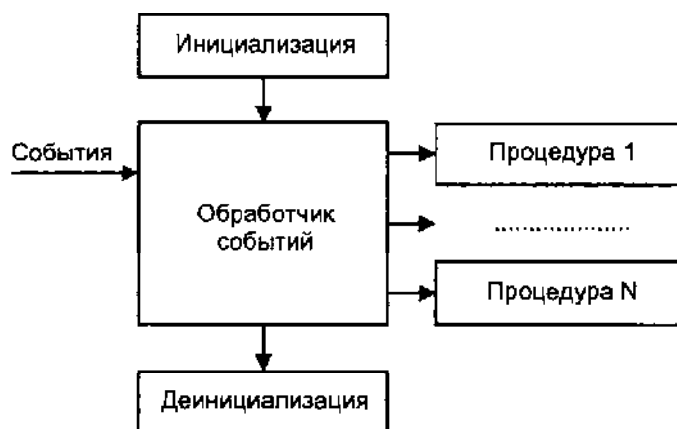


Рис. 1.6. Архитектура программы, управляемой событиями

Из рис. 1.6 видно, что программа, управляемая событиями, не имеет привычных переходов от процедуры к процедуре. В этом случае сразу после выполнения инициализации начинает работать обработчик событий, который, анализируя поступившее событие, передает управление процедуре-обработчику события. После обработки события управление возвращается системе. После этого обработчик готов к приему следующего события. Информация о наступившем событии передается в виде сообщения, которое раскрывает характер события и содержит информацию, необходимую для его обработки. По окончании работы программы производится деинициализация.

Таким образом, для программы работающей в среде Windows, необходимо создать ряд процедур:

- процедуру инициализации;
- процедуру – обработчика событий;
- диспетчер событий;
- процедуры – обработчики конкретных событий;
- процедуру деинициализации.

Вывод информации на экран и прием сообщений Windows-программа осуществляет посредством окон, поэтому, помимо работы с сообщениями, необходимо выполнять работу с окнами.

Что нового вы узнали?

В этой главе вы познакомились со следующими понятиями:

- объектно-ориентированное программирование;
- объект и класс;
- поля объекта;
- операции и методы объекта;
- свойства объекта;
- наследование, инкапсуляция и полиморфизм;
- события и сообщения;

- обработчик события;
- единый графический интерфейс;
- визуальное проектирование интерфейса программы;
- форма, модуль, компонент.

Изучив эту главу, вы узнали:

- отличительные особенности архитектуры Windows-программ;
- структуру модуля программы, назначение его разделов;
- форматы доступа к свойствам и методам объекта;
- чем различаются константы, переменные, функции и процедуры, включенные и не включенные в описание класса;
- основные элементы единого графического интерфейса.

Контрольные вопросы

1. В чем заключается основное различие процедурного и объектно-ориентированного программирования?
2. Что такое объект? Что общего имеет тип `object` с типом `record`? Чем они различаются?
3. Что такое класс?
4. Как можно обратиться к полю объекта?
5. В чем заключается наследование?
6. Что такое инкапсуляция и для чего она применяется?
7. В чем заключается полиморфизм операций?
8. Что такое метод? Каковы особенности описания методов?
9. Что такое свойство объекта, каким образом можно его изменять?
10. Что такое события? Каково назначение обработчика события?
11. Каково назначение сообщений?
12. Что такое конструктор, деструктор?
13. В чем заключаются преимущества визуального проектирования интерфейса?
14. Опишите структуру и назначение отдельных элементов головной программы приложения Delphi.
15. Каково назначение модуля в проекте приложения Delphi? Опишите назначение отдельных разделов модуля.
16. Как различается доступность объектов, описанных в разделах `interface` и `implementation` модуля?
17. Как указать ссылку на свойства и методы объекта в тексте программы?
18. Каковы различия в передаче параметров в процедуры и функции по значению и по ссылке?

ГЛАВА 2

Интегрированная среда разработки Delphi 6

В этой главе вы познакомитесь с использованием интегрированной среды Delphi 6 для создания, компиляции и отладки приложения.

После изучения главы вы будете знать:

- Назначение интегрированной среды разработки Delphi 6.
- Порядок использования интегрированной среды для создания, компиляции и отладки приложения.
- Порядок создания формы с размещением визуальных компонентов.
- Порядок использования редактора кода при создании обработчика события.
- Порядок использования редактора изображений Image Editor для редактирования иконки приложения.

Возможности Delphi

Delphi – это потомок среды программирования Turbo Pascal. Название среды разработки приложений Delphi произошло от названия города в Древней Греции, где находился знаменитый Дельфийский оракул.

Примечание. Дельфийский оракул – храм Аполлона в городе Дельфы, жрецы которого давали предсказания.

Система визуального объектно-ориентированного проектирования Delphi позволяет:

- создавать законченные приложения для Windows самой различной направленности: от чисто вычислительных и логических до использующих графику и мультимедиа;
- быстро создавать профессионально выглядящий оконный интерфейс для любых приложений, написанных на любом языке; интерфейс удовлетворяет всем требованиям Windows и автоматически настраивается на ту систему, которая установлена на компьютере пользователя, поскольку использует функции, процедуры и библиотеки Windows;
- создавать свои динамически присоединяемые библиотеки (DLL) компонентов, форм, функций, которые затем можно использовать из других языков программирования;
- создавать мощные системы работы с локальными и удаленными базами данных любых типов;

- формировать и печатать сложные отчеты, включающие таблицы, графики и т. п.;
- создавать справочные системы (файлы .hlp), как для своих приложений, так и для любых других, с которыми можно работать не только из приложений, но и просто из Windows;
- создавать профессиональные программы установки для приложений Windows, учитывающие всю специфику и все требования операционной системы.

Delphi – быстро развивающаяся система. Первая версия Delphi 1.0 была выпущена в феврале 1995 г. В 1996 г. вышла версия Delphi 2, в 1997 г. – Delphi 3, в 1998 г. – Delphi 4, в 1999 г. – Delphi 5, в 2001 г. – Delphi 6, в 2002 г. – Delphi 7, в 2003 г. – Delphi 8 и т.д. Все версии, начиная с Delphi 2.0, рассчитаны на разработку 32-разрядных приложений, т.е. приложений для операционных систем Windows 95/98 и Windows NT.

Примечание. Вместе с шестой версией Delphi вышла совместимая с ним по языку и библиотекам среда Kylix, предназначенная для компиляции программ под операционную систему Linux. Версия 8 способна генерировать байт-код исключительно для платформы .NET. Последующие версии (обозначаемые годами выхода, а не порядковыми номерами) могут создавать как приложения Win32, так и байт-код для платформы .NET.

Когда готовилась рукопись книги, для разработки программ с использованием методов визуального программирования готовилась к выпуску среда программирования Delphi 2010. Интегрированная среда разработки Delphi 2010 имеет новые возможности:

- функцию IDE Insight, позволяющую обращаться ко всем возможностям, параметрам и компонентам интегрированной среды разработки, не тратя время на их поиск в меню и диалоговых окнах;
- средство Code Formatter, упрощающее унификацию внешнего вида кода;
- усовершенствованные функции поиска и повторного открытия файлов, позволяющие быстро находить нужные сведения.

Особенностью данной версии является уменьшение объемов ввода данных с клавиатуры за счет того, что Delphi 2010 позволяет создавать использующие сенсорный ввод данных графические интерфейсы и приложения для сенсорных панелей.

Общее описание среды

Интегрированная среда разработки (ИСР) Delphi 6 – это среда, в которой есть все необходимое для проектирования, запуска и тестирования создаваемых приложений: редактор исходного кода, отладчик, инструментальные панели, редактор изображений, инструментарий баз данных. Такая интеграция обеспечивает разработчика приложений набором инструментов, гармонично дополняющих друг друга и облегчающих процесс создания современных приложений.

Для запуска Delphi следует выбрать в Главном меню Windows команды Пуск ► Программы ► Borland Delphi 6 ► Delphi 6 или щелкнуть по ярлыку программы Delphi 6 на рабочем столе. После запуска Delphi на экране компьютера появляется основное окно интегрированной среды разработки (рис. 2.1).

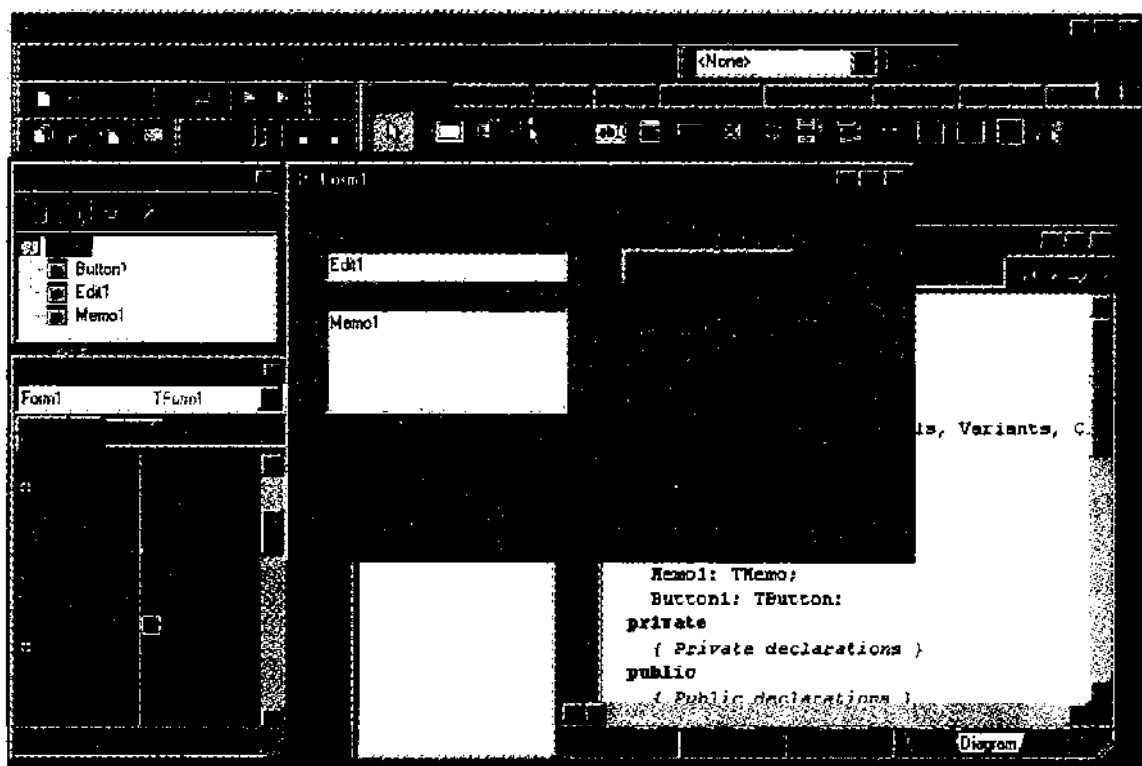


Рис. 2.1. Вид окна интегрированной среды разработки Delphi 6

В верхней части окна ИСР отображается полоса главного меню. Ниже полосы главного меню расположены две инструментальные панели. Левая панель (состоящая, в свою очередь, из трех панелей) содержит два ряда кнопок, дублирующих некоторые наиболее часто используемые команды меню (рис. 2.2).



Рис. 2.2. Панель кнопок, дублирующих команды меню, с всплывающей подсказкой

Правая панель содержит панель библиотеки визуальных компонентов (Visual Component Library – VCL). В дальнейшем для краткости будем называть библиотеку визуальных компонентов просто палитрой. Палитра компонентов содержит ряд страниц, закладки которых видны в ее верхней части (рис. 2.3).

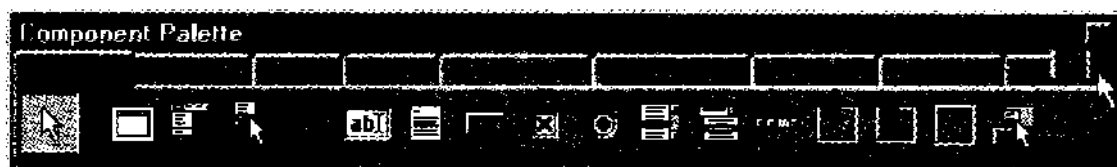


Рис. 2.3. Палитра компонентов

Правее полосы главного меню располагается еще одна небольшая инструментальная панель, содержащая раскрывающийся список и две кнопки (рис. 2.4).



Рис. 2.4. Панель выбора и сохранения различных конфигураций ИСР

Эта панель служит для сохранения и выбора различных конфигураций окна ИСР, которые вы сами можете создавать и запоминать.

Под палитрой компонентов располагается окно формы, на которой размещены три добавленных компонента (рис. 2.5).

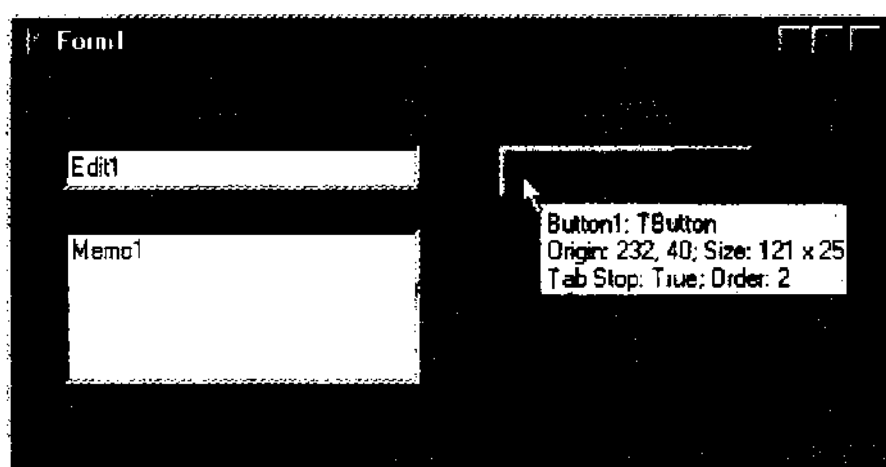


Рис. 2.5. Окно формы с размещенными компонентами

Форма, на которой размещаются другие компоненты, является основой почти всех приложений Delphi.

В начале создания нового проекта форма не содержит компонентов. В некоторых случаях при разработке какого-то модуля форма может оказаться вообще ненужной.

Форму можно понимать как типичное окно Windows. Она обладает теми же свойствами, что и другие окна Windows: имеет управляющее меню в верхнем левом углу, полосу заголовка, занимающую верхнюю часть, кнопки разворачивания, свертывания и закрытия окна в верхнем правом углу. Используя Инспектор объектов, можно изменить вид окна: убрать какие-то кнопки или всю полосу заголовка, сделать его окном с неизменяемыми размерами и т. п.

Во время проектирования форма покрыта сеткой из точек. В узлах этой сетки размещаются компоненты, которые разработчик помещает на форму. Во время выполнения приложения эта сетка не видна.

После того как на форме размещены какие-либо компоненты, можно получить о них контекстную справку. Для этого следует выделить интересующий компонент и нажать клавишу F1. Если щелкнуть на самой форме и нажать F1, то будет показана справка о классе формы (рис. 2.6).

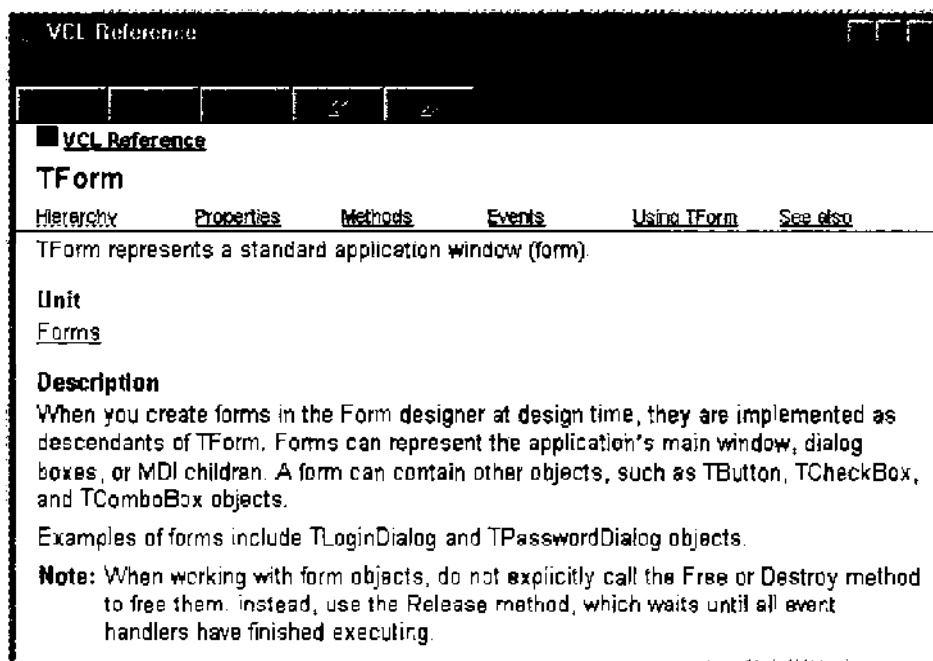


Рис. 2.6. Окно Справки о форме

В основном поле окна слева находится окно Object Inspector (Инспектор объектов), с помощью которого в дальнейшем можно будет задать свойства компонентов и обработчики событий. На рис. 2.7 в окне Инспектора объектов отображается информация о свойствах формы Form1.

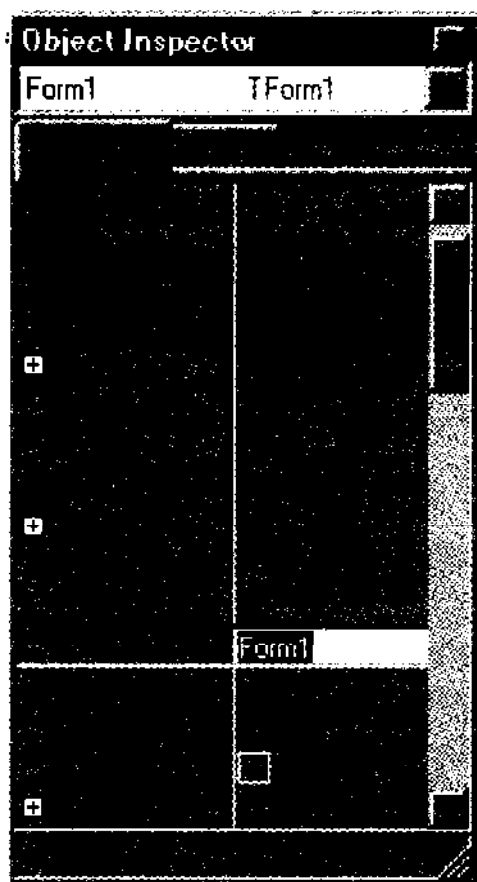


Рис. 2.7. Окно Инспектора объектов

Выше окна Object Inspector (см. рис. 2.1) расположено окно Object TreeView (Дерево объектов) (рис. 2.8), отображающее иерархию компонентов приложения с точки зрения их принадлежности друг другу. В Дереве объектов надо щелкать на дочерние компоненты и перетаскивать их на другие контейнеры, при этом изменения синхронно отображаются в редакторе форм.

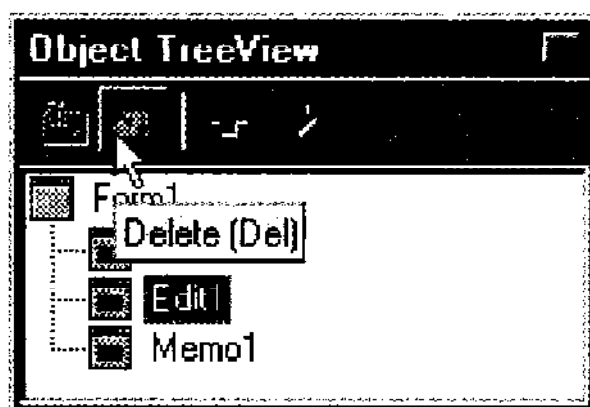


Рис. 2.8. Окно Дерева объектов

Примечание. Отметим, что компоненты, ключевые свойства которых не определены, отмечаются в дереве объектов вопросительными знаками, сразу же привлекающими внимание. Кроме того, Дерево объектов отображает и компоненты, создаваемые неявно.

Ниже окна формы расположено окно Code Editor (Редактор кода). Обычно при первом взгляде на экран оно не видно, так как его размер равен размеру формы и окно редактора кода практически полностью перекрывается окном формы (рис. 2.9).

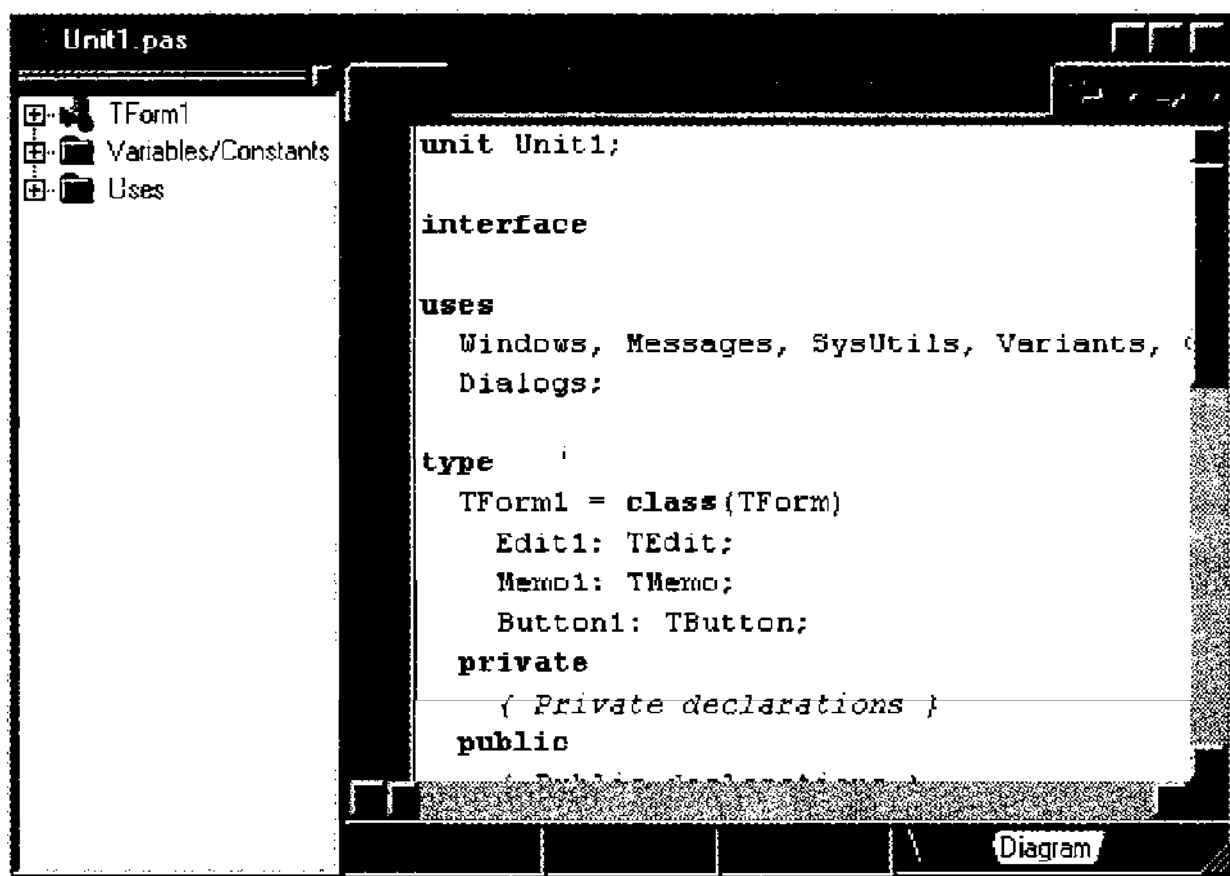


Рис. 2.9. Окно Редактор кода

На рис. 2.1 это окно немного сдвинуто вправо и вниз и выглядывает из-под окна формы.

Подробное описание элементов окна интегрированной системы программирования Delphi 6 можно найти в приложении 2.

Одним из способов работы с ИСП является выбор команды из меню. Выбрать команду меню можно любым из стандартных способов: нажатием F10, Alt в сочетании с «горячей клавишей» или щелчком мыши по соответствующему пункту меню. Можно уточнить назначение пункта меню, используя справочную систему Delphi. Для получения справки Delphi о пункте меню следует выбрать интересующий вас пункт меню, например команду Run в меню Run, и нажать клавишу F1. После этого откроется окно Delphi Help, содержащее справочную информацию (рис. 2.10).

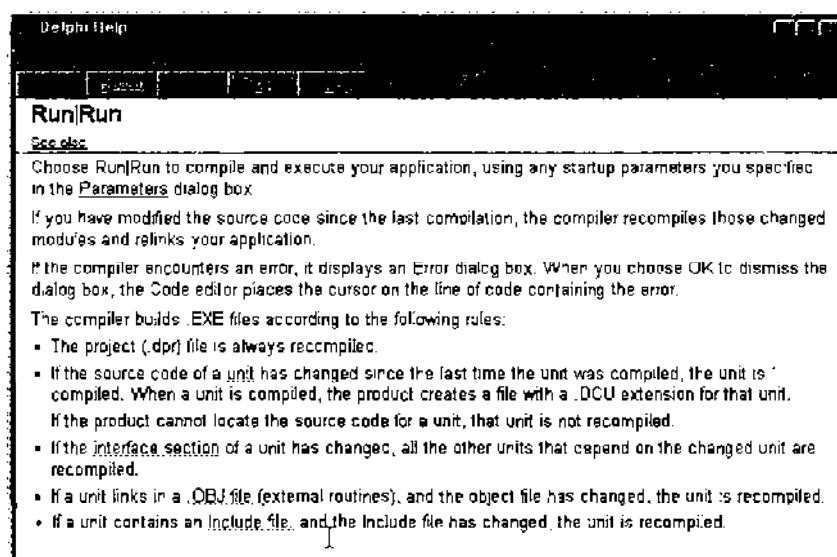


Рис. 2.10. Окно Delphi Help со справочной информацией о пункте меню

Создание, компиляция и отладка простого приложения

Основные операции в интегрированной среде разработки Delphi рассмотрим на примере создания, компиляции и отладки простого приложения.

Упражнение 2.1. В главе 1¹ была рассмотрена задача вычисления суммы двух целых чисел. Создадим в ИСР Delphi приложение, которое обеспечивает ввод двух целых чисел, по щелчку на кнопке с символом «=» вычисляет их сумму и выводит результат.

Создание формы с размещением визуальных компонентов. После запуска ИСР Delphi создайте новый проект при помощи команды **File ► New ► Application**. В результате создания проекта приложения в окне Delphi будет раскрыто окно формы, на которой можно размещать визуальные компоненты проекта. Сохраните новый проект, для чего выберите команду **File ► Save Project As**. В окне **Save Unit1 As** создайте новую папку для файлов создаваемого проекта, например «Сумма двух целых чисел» (рис. 2.11).

Затем в окне **Save Unit1 As** откройте созданную папку и нажмите кнопку **Сохранить**. После сохранения файла модуля **Unit1.pas** откроется окно **Save Project As**. Задайте имя файла проекта, например «summa», и нажмите кнопку **Сохранить** (рис. 2.12).

Измените свойства формы. Для изменения размеров формы захватите угол окна формы и, не отпуская левую кнопку мыши, перемещайте мышь, задавая требуемый размер формы. Измените надпись в заголовке формы **Form1** с помощью Инспектора объектов. Инспектор объектов обеспечивает простой и удобный интерфейс для изменения свойств объектов Delphi и управления событиями, на которые реагирует объект. Окно Инспектора объектов состоит из двух страниц, каждую из которых можно использовать для определения поведения компонента.

¹ См.: Попов В.Б. Turbo Pascal для школьников. — М.: Финансы и статистика; ИНФРА-М, 2010.

Первая страница называется Properties (Свойства), вторая – Events (События). Над ними располагается раскрывающийся список всех компонентов, размещенных на форме. В нем можно выбрать тот компонент, свойства и события которого вас интересуют. Страница Properties Инспектора объектов (рис. 2.13) показывает свойства того объекта, который в данный момент выделен. Для переключения между страницами свойств и событий используются закладки Properties и Events, расположенные в верхней части окна Инспектора объектов.

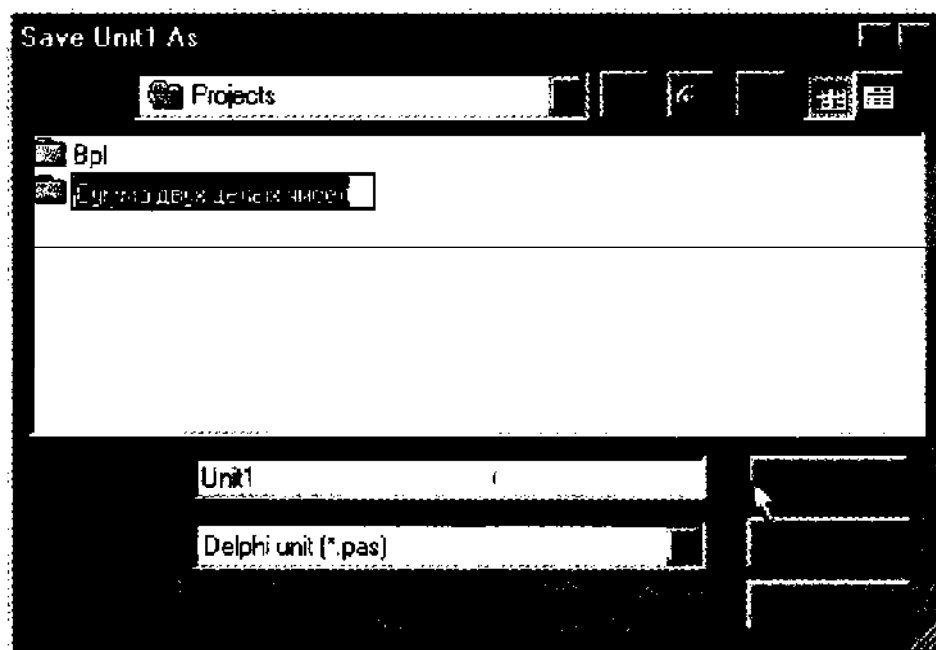


Рис. 2.11. Создание папки для нового проекта

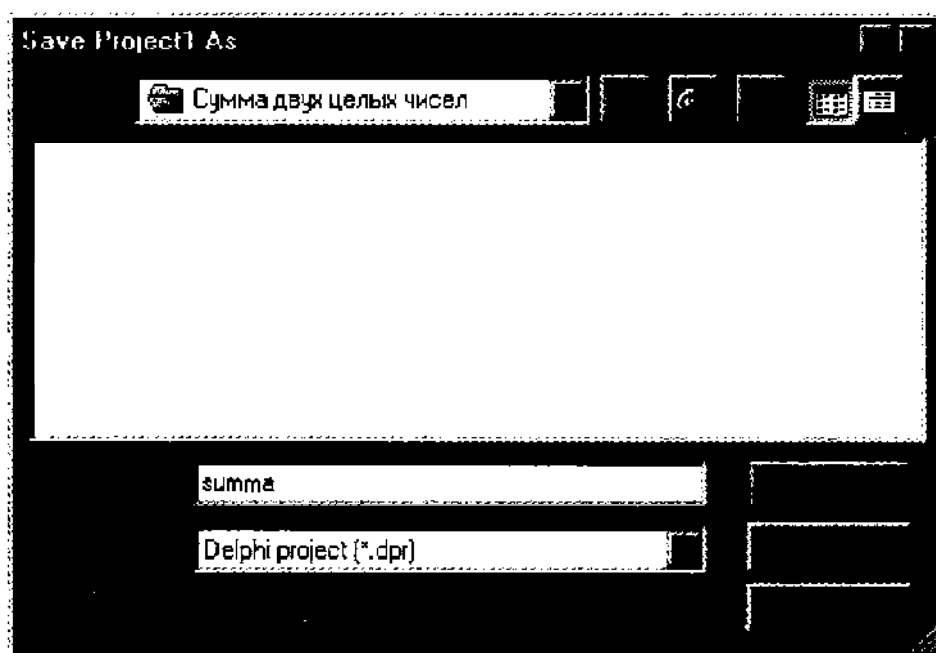


Рис. 2.12. Сохранение проекта

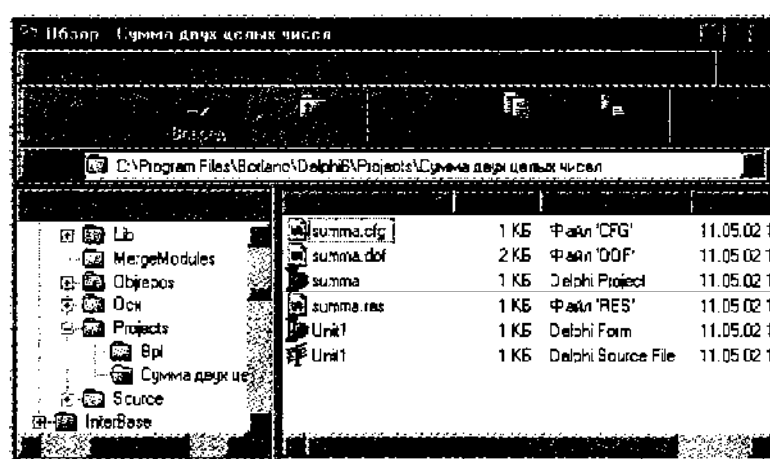
После щелчка на некоторых свойствах, например на свойстве Color, справа от имени свойства открывается окно раскрывающегося списка. Нажав в нем кнопку со стрелкой вниз, можно увидеть список возможных значений этого свойства. Список значений можно просмотреть с помощью бегунка. Например, если изменить значение свойства Color с принятого по умолчанию clBtnFace (цвет поверхности кнопок) на clWindow (цвет окна), поверхность формы изменит свой цвет.

Рядом с некоторыми свойствами можно заметить знак «плюс» (например, возле свойства Font). Это означает, что данное свойство является объектом, который, в свою очередь, имеет ряд свойств. После щелчка на «плюсе» или двойного щелчка на свойстве Font откроется таблица таких свойств, как Color (цвет), Height (высота), Name (имя шрифта) и др. Среди них есть свойство Style (стиль), около которого тоже имеется знак «плюс». Щелчок на нем или двойной щелчок на этом свойстве раскроет дополнительный список подсвойств, в котором можно, например, установить в значение True для свойства fsBold (жирный). Кстати, для смены True на False и обратно в подобных булевых свойствах не обязательно выбирать значение из раскрывающегося списка. Для его изменения достаточно двойного щелчка на значении свойства. После просмотра или изменения подсвойств можно опять сделать двойной щелчок на головном свойстве или щелчок на знаке «минус» около него, и список подсвойств будет закрыт.

Для изменения заголовка формы Form1 в окне Инспектора объектов откройте страницу Properties (Свойства), затем выберите свойство Caption (заголовок) и задайте новое значение этого свойства, например «Сумма двух целых чисел» (рис. 2.13).



а



б

Рис. 2.13. Изменение заголовка формы Form1:

а – изменение свойства Caption формы Form1; б – Caption «Сумма двух целых чисел»

Разместите на форме компоненты Edit1, Edit2, Edit3, Label1, Button1 (рис. 2.14).

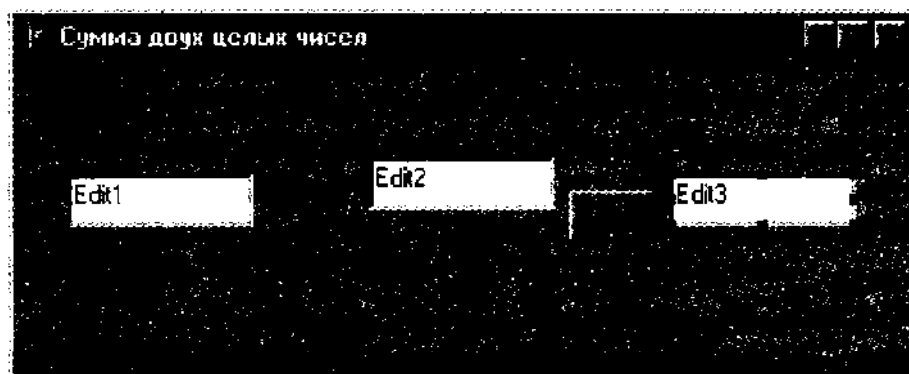


Рис. 2.14. Форма с размещенными на ней компонентами

Для размещения компонентов на форме, щелкнув на вкладке Standard, откройте палитру с требуемыми компонентами, затем, щелкнув на значке соответствующего компонента, например, Edit, щелкните в окне формы. Задайте положение и размер компонента при помощи мыши.

Примечание. Если вы забыли, на какой странице палитры расположен конкретный компонент, выберите команду View ► Component List, и на экране появится список компонентов в алфавитном порядке. Выбрав в окне Components нужный компонент, нажмите кнопку Add to form для размещения компонента на форме.

Есть и другой способ поместить компонент на форму – достаточно сделать двойной щелчок на значке компонента, и он будет автоматически помещен в центр формы. Если вы выбрали компонент, а затем решили не размещать его, достаточно нажать кнопку указателя. Это прервет процесс размещения компонента, и программа вернется в нормальный режим, в котором можно выбрать другой компонент или выполнить какую-либо команду.

Получение справки о назначении компонентов. Имена компонентов, соответствующих значкам на палитре, можно узнать из ярлычка, появляющегося, если задержать над этим значком курсор мыши.

Примечание. Имена на ярлычках выглядят так: MainMenu, Button и т.д. Однако в Delphi все имена классов в действительности начинаются с символа «Т», например: TMainMenu, TButton. Под такими именами во встроенной в Delphi справочной системе можно найти описания соответствующих компонентов.

Назначение компонентов можно уточнить, используя систему контекстной помощи Delphi. Для этого следует выделить нужную страницу в палитре компонентов и нажать клавишу F1. После этого откроется окно Delphi Help со справочной информацией. Например, окно справки о компонентах страницы Standard выглядит, как на рис. 2.15.

Для просмотра информации в окне используется полоса прокрутки. Если требуется просмотреть справку о конкретном компоненте, то следует выбрать нужную ссылку. Если выбрать в палитре компонент и нажать F1, то будет показана справка о типе данного компонента. Нажатие кнопки Заккрыть в окне Delphi Help приводит к закрытию окна со справочной информацией.

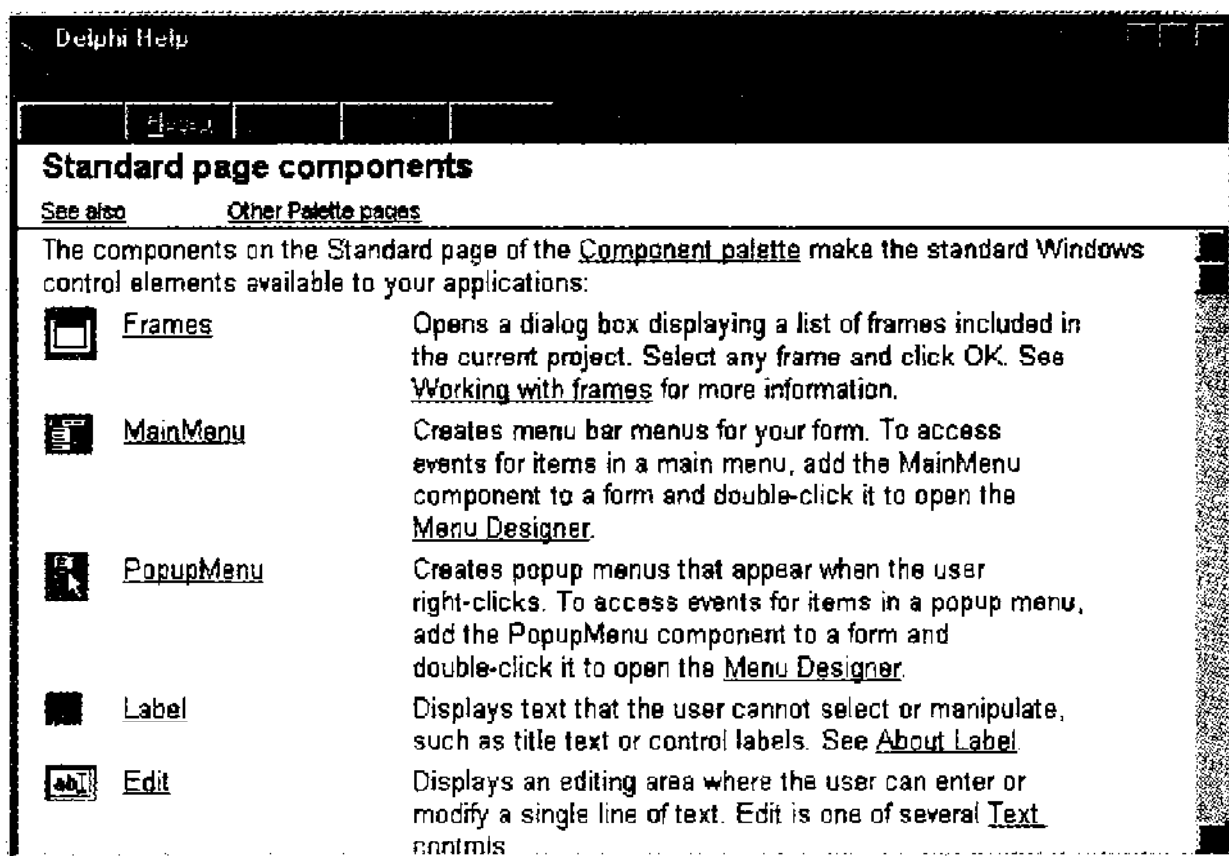


Рис. 2.15. Окно справочной информации о компонентах страницы Standard

Задайте свойство Caption компонента Label1 «+». Измените размер символов компонента Label1, для чего в окне Инспектора объектов выберите в списке объект Label1, затем на странице Properties разверните список свойств Font. В списке свойств Font выберите свойство Size и задайте для него значение 20 (рис. 2.16).

Аналогичным образом задайте свойство Caption компонента Button1 «=» и размер символов, равный 20.

Примечание. Помимо главного меню, в Delphi имеется система контекстных раскрывающихся меню, которые появляются при щелчке правой кнопкой мыши на каком-либо компоненте (рис. 2.17). Большинство разделов этих контекстных меню дублируют основные разделы главного меню.

Для выравнивания компонентов на форме при нажатой клавише Shift выделите компоненты и правой кнопкой мыши выберите в контекстном меню команду Position ► Align.

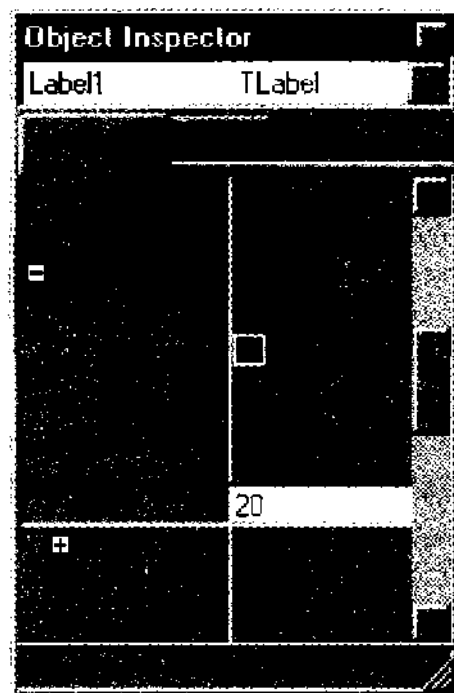


Рис. 2.16. Изменение значения свойства Font.Size

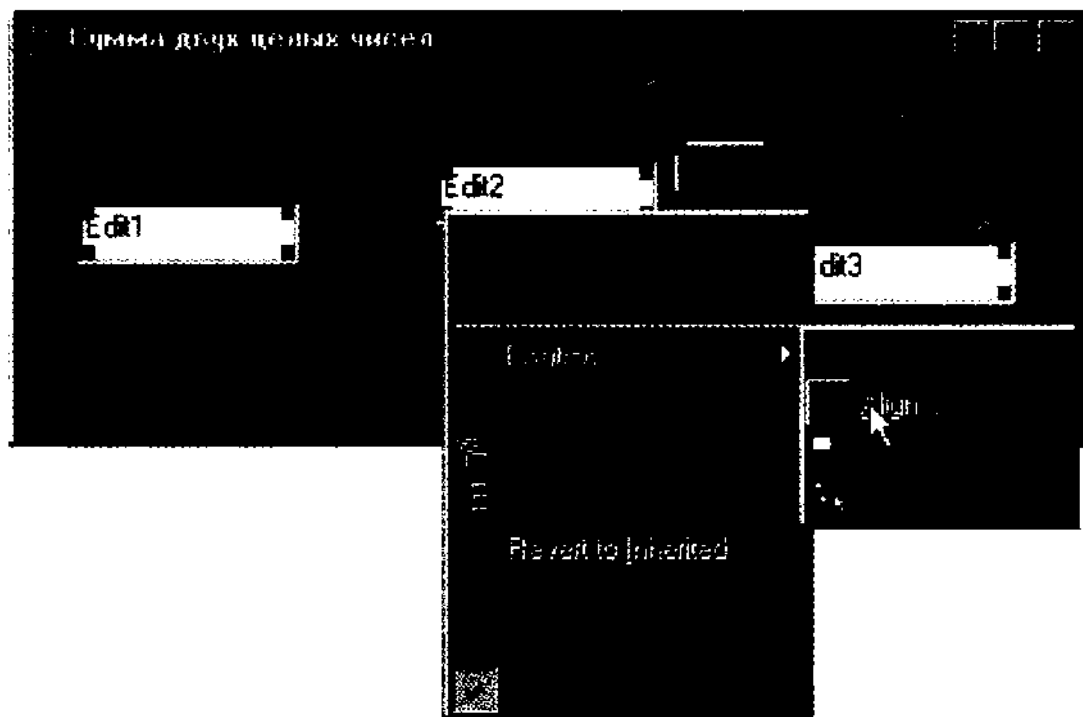


Рис. 2.17. Выравнивание компонентов

В окне Alignment (Выравнивание) выберите в панели Vertical (вертикальное) вариант Centers (по центру) и нажмите кнопку OK. Все компоненты, участвующие в операции выравнивания, будут выровнены по вертикали (рис. 2.18).

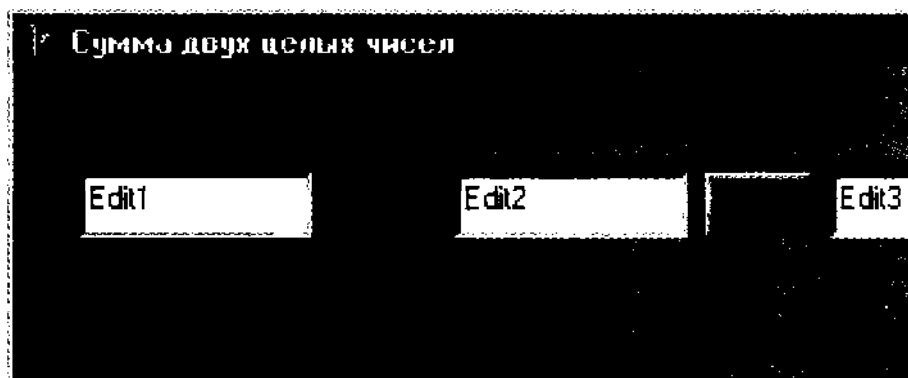


Рис. 2.18. Результат выравнивания компонентов на форме

Удалите текст «Edit1», «Edit2», «Edit3» в соответствующих компонентах. Для этого выберите объект в окне Инспектора объектов, на странице Properties выберите свойство Text и удалите текст (рис. 2.19).



Рис. 2.19. Удаление текста в Edit1.Text

Примечание. При работе с Инспектором объектов можно получить контекстную справку о свойствах или событиях. Для этого следует выделить в окне Инспектор объектов интересное свойство или событие и нажать клавишу F1, а затем просмотреть в окне Delphi Help справочную информацию.

Добавьте на форму три объекта Label, расположите их над объектами Edit1 – Edit3 и задайте их свойствам Caption значения «Слагаемое», «Слагаемое» и «Сумма». Нажатием F12 активизируйте окно редактора кода. Обратите внимание на то, что в разделе описания программного модуля ИСР были сгенерированы описание формы и размещенных на ней компонентов:

type

```
TForm1 = class(TForm)
```

```
    Edit1: TEdit;
```

```
    Edit2: TEdit;
```

```
    Edit3: TEdit;
```

```
    Label1: TLabel;
```

```
    Button1: TButton;
```

```
    Label2: TLabel;
```

```
    Label3: TLabel;
```

```
    Label4: TLabel;
```

Примечание. После того, как завершено тщательное размещение и выравнивание компонентов, их местоположение на форме можно зафиксировать, чтобы в дальнейшем случайно не сдвинуть какой-либо компонент. Для фиксации положения компонентов следует воспользоваться командой Edit ► Lock Controls. После этого блокируются даже вновь размещаемые на форме компоненты. Если в дальнейшем возникнет необходимость в изменении расположения компонентов, то нужно будет повторно выбрать команду Edit ► Lock Controls.

Сохраните изменения, внесенные в проект, нажатием кнопки Save All стандартной панели инструментов. Откомпилируйте созданный проект командой Project ► Compile summa (слово «summa» указывает имя проекта).

Для просмотра содержимого проекта Delphi откройте окно проводника и просмотрите папку проекта.

Как показано на рис. 2.20, проект Delphi состоит из форм, модулей, установок параметров проекта, ресурсов и т.д. Вся эта информация размещается в файлах. Многие из этих файлов автоматически создаются Delphi, когда вы строите приложение. Главной частью приложения является файл summa.dpr

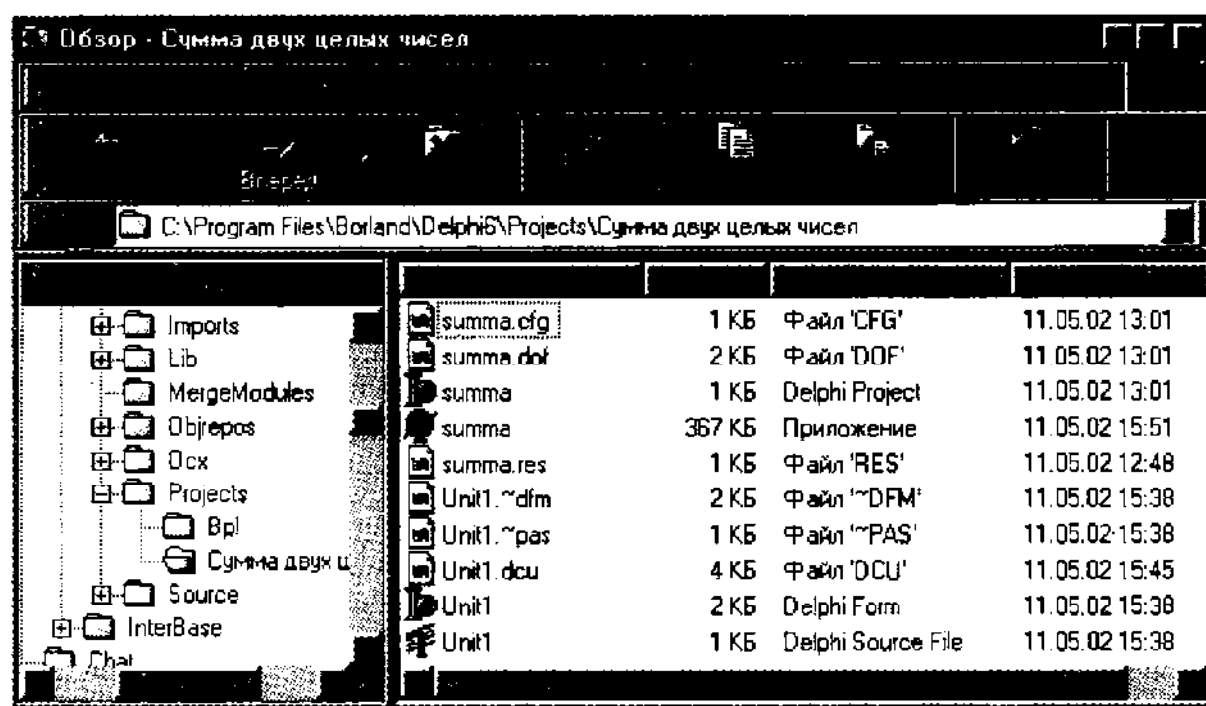


Рис. 2.20. Состав проекта в Delphi

– файл проекта, содержащий код на языке Object Pascal, с которого начинается выполнение программы и который обеспечивает инициализацию других модулей. Он создается и модифицируется Delphi автоматически в процессе разработки приложения. Имя, присваиваемое файлу проекта при его сохранении, становится именем исполняемого файла (в нашем примере – `summa.exe`). Другие файлы проекта:

- Unit1.pas – файл модуля;
- Unit1.dfm – файл формы;
- summa.dof – файл параметров проекта;
- summa.res – файл ресурсов;
- Unit1.dfm, Unit1.pas – файлы резервных копий.

Примечание. Обратите внимание на то, что файл модуля Unit1.pas был сгенерирован ИСР Delphi. Пока вам не пришлось вводить ни одного символа в текст программы на языке Object Pascal.

Список расширений файлов, создаваемых Delphi, выглядит следующим образом.

- DPR – файл проекта Delphi. Это главная программа приложения.
- ~DP – резервный файл DPR. Создается, если включено резервное сохранение.
- PAS – исходный код модуля или формы. Главная программа находится в файле DPR.
- ~PA – резервный файл PAS. Создается, если включено резервное сохранение.

- DFM – эти файлы связаны с файлами PAS. DFM – это двоичные файлы, в которых содержатся начальные данные для компонентов (это те свойства, которые программист устанавливает в Object Inspector). Файл DFM нельзя отредактировать текстовым редактором, но если открыть его в Delphi, то будет отображена текстовая версия его содержимого.

- ~DF – резервный файл DFM. Создается, если включено резервное сохранение.

- DCU – откомпилированный модуль, подобен файлу OBJ.

- DSM – файл с информацией, используемой внутренним отладчиком. Этот файл создается при каждой компиляции проекта.

- OPT – опции проекта. Это настройки компилятора и компоновщика для данного проекта. Их можно вызвать в меню Options/Project.

- RES – файл ресурсов Windows. Создается Delphi автоматически и требуется для компиляции. Пользоваться этим файлом не обязательно, но удалять его не следует.

- EXE – откомпилированный проект. Исполняемая Windows-программа.

Кроме того, в папке проекта могут находиться и другие файлы Windows, которые используются в проекте Delphi:

- файлы справки (.hlp);

- файлы изображений, или графические файлы (.wmf, .bmp, .ico).

После просмотра списка файлов проекта закройте окно Проводника и активизируйте окно Delphi.

Запустите программу на выполнение при помощи команды Run ► Run или нажатием кнопки Run в панели инструментов или клавиши F9. После этого на экране компьютера появится окно созданной вами формы с компонентами, в которые можно вводить значения. Обратите внимание на то, что окно формы имеет стандартные атрибуты окна Windows, его можно минимизировать, развернуть на весь экран, изменить его размеры, но приложение не будет выполнять каких-либо вычислений, так как выполняющий вычисления фрагмент программы не был создан. Завершите работу приложения любым из стандартных способов, например нажатием Alt+F4.

Создание кода – обработчика события. Для того чтобы приложение выполняло вычисления при щелчке по кнопке Button1 с изображением символа «=», следует написать код обработки этого события. Для создания кода обработки события следует воспользоваться Инспектором объектов. В окне Инспектора объектов на странице Events указаны все события, на которые может реагировать выбранный объект. Страница событий связана с редактором кода следующим образом: если дважды щелкнуть мышью справа от какого-либо пункта, то соответствующий данному событию код будет автоматически помещен в окно Редактор кода, а окно Редактор кода немедленно получит фокус, и вы сразу же получите возможность отредактировать код обработчика данного события. Например, если требуется выполнить определенные действия при щелчке левой кнопкой мыши по данному объекту, то следует выделить событие OnClick. Рядом с именем этого события откроется окно с раскрывающимся списком. Если в приложении уже были созданы какие-нибудь обработчики

событий и при событии `OnClick` требуется использовать один из них, можно выбрать необходимый обработчик из раскрывающегося списка.

Примечание. В Delphi версии 5 и старше в Инспектор объектов введена возможность фильтрации свойств и событий и возможность группировать их по категориям. Чтобы воспользоваться этими возможностями, следует щелкнуть в окне Инспектор объектов правой кнопкой мыши. Если выбрать в раскрывающемся меню раздел `View`, то будет показан список категорий свойств и событий. Около каждой категории имеется индикатор. Можно включить индикаторы только у некоторых категорий, и тогда в Инспекторе объектов будут отображаться события и свойства только указанных категорий. Выбор раздела `Toggle` переключает видимость разделов: видимые становятся невидимыми, и наоборот. Выбор раздела `All` позволяет сделать видимыми все свойства и события, а выбор раздела `None` позволяет сделать все события и свойства невидимыми. В нижней части окна Инспектора объектов указывается, сколько свойств или событий невидимы в данный момент.

Так как в приложении еще нет обработчика событий, требуется написать новый обработчик щелчка по кнопке `Button1` с изображением символа «=». Для этого следует выбрать в окне Инспектора объектов объект `Button1`, затем на странице События дважды щелкнуть на пустом поле списка в событии `OnClick`. После этого окно Редактора кода немедленно получит фокус. В этом окне в разделе `interface` находится запись процедуры обработчика события:

```
procedure TForm1.Button1Click(Sender: TObject);
```

В разделе `implementation` располагается текст заготовки этой процедуры:

```
procedure TForm1.Button1Click(Sender: TObject);
```

```
begin
```

```
...
```

```
end;
```

Курсор будет находиться в пустой строке между ключевыми словами `begin` и `end`. Этот код является заготовкой для обработчика события, которую автоматически создала ИСР Delphi. Вам остается только разместить необходимые операторы в пространстве между `begin` и `end`. Так как в задаче требуется выполнить сложение двух целых чисел, а результат их сложения также является целым числом, то в разделе описания переменных следует ввести следующее описание:

```
var
```

```
  a,b,c: integer; {2 слагаемых и сумма - целые числа}
```

Так как в приложении для ввода чисел-слагаемых используются окна редактирования `Edit1` и `Edit2`, то необходимо при помощи процедуры `StrToInt` преобразовать строки из этих окон в целые числа. Для вывода результата суммирования в окне редактирования `Edit3` нужно преобразовать число в строку

функцией `IntToStr`. Поэтому в основное тело процедуры обработки события следует ввести следующий текст:

```
a:=StrToInt(Edit1.text);  
b:=StrToInt(Edit2.text);  
c:=a+b;  
Edit3.text:=IntToStr(c);
```

Целиком процедура обработки события щелчка по кнопке `Button1` будет выглядеть следующим образом:

```
procedure TForm1.Button1Click(Sender: TObject);  
var  
    a,b,c: integer; { Два слагаемых и сумма - целые числа}  
begin  
    a:=StrToInt(Edit1.text); {Преобразование текстовой строки  
в целое число}  
    b:=StrToInt(Edit2.text);  
    c:=a+b; {Вычисление суммы}  
    Edit3.text:=IntToStr(c); {Преобразование целого числа в  
текстовую строку}  
end;
```

Примечание. В нижней части окна Редактора кода располагается строка состояния. В самой левой ее позиции отображается положение курсора: номер строки и колонки. Второй элемент строки – индикатор модификации. Если код был изменен с момента последнего сохранения его в файле, то в индикаторе модификации появляется слово «Modified», означающее, что код был изменен и изменения не сохранены на диске. Третий элемент строки состояния – индикатор режима вставки, показывающий, будут ли вводимые символы вставляться в текст или заменять его. Переключение режима Вставка/Замещение производится клавишей `Insert`.

В окно Редактора кода, как и в другие окна Delphi, встроена контекстная **справка**. Чтобы получить справку о каком-либо слове кода (ключевом слове, имени функции и т. п.), например `StrToInt`, следует установить курсор на это слово и нажать клавишу `F1`. После этого откроется окно `Delphi Help` со справочной информацией (рис. 2.21).

Закройте окно справки и сохраните изменения в проекте, нажав на кнопку `Save All` в панели инструментов ИСР.

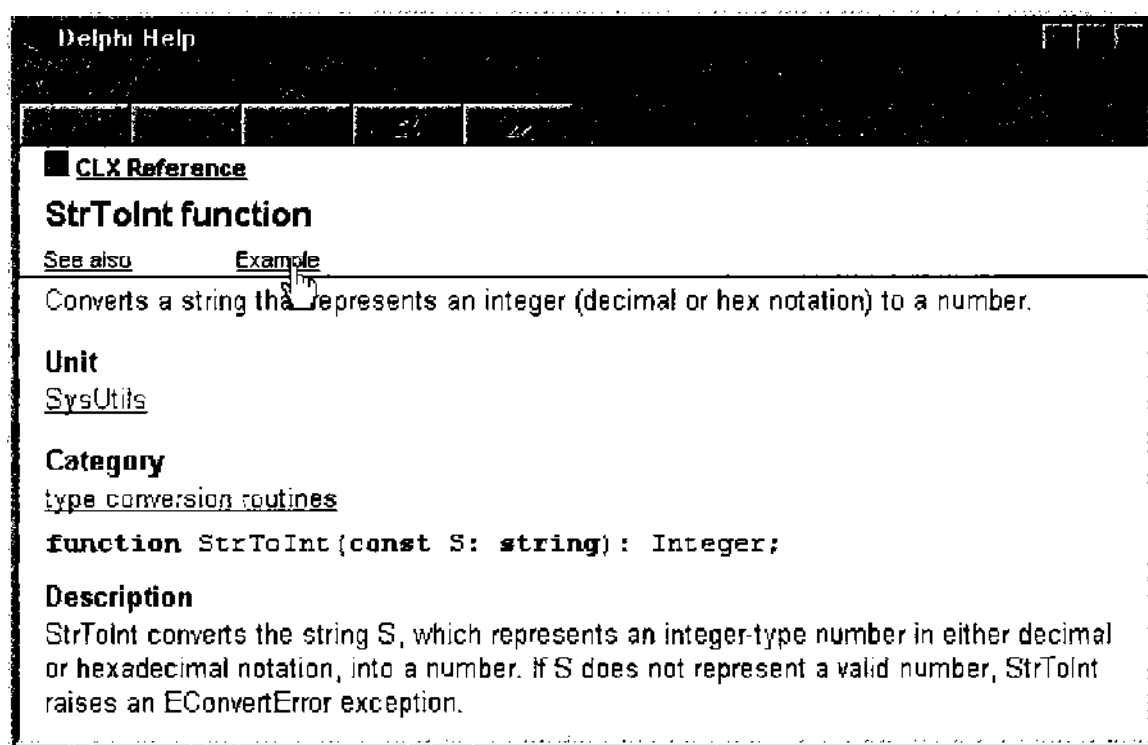


Рис. 2.21. Справка Delphi о функции StrToInt

Запустите приложение на выполнение нажатием кнопки Run на панели инструментов. В окне приложения введите значения слагаемых и, нажав кнопку Button1 с надписью «=», запустите процедуру обработки события – вычисление суммы. Окно созданного приложения будет выглядеть следующим образом (рис. 2.22).

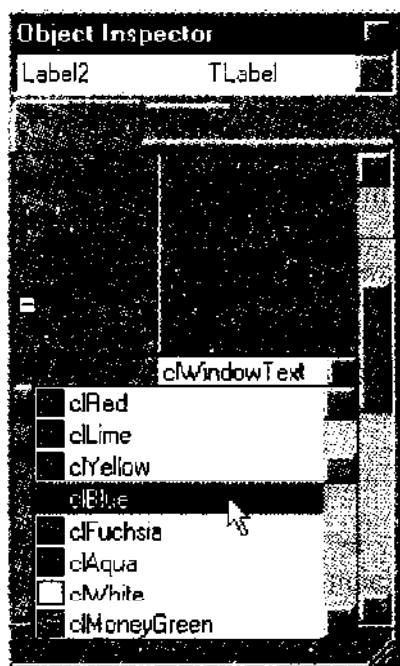


Рис. 2.22. Вид окна приложения

После проверки работы приложения закройте его окно.

Открытие ранее сохраненного проекта. Чтобы открыть сохраненный ранее проект, выберите в меню File команду Open Project. В окне Open Project выберите папку, файл проекта и нажмите кнопку Открыть. Можно открыть проект из списка редактированных в последнее время, выбрав команду File ► Reopen, а затем в списке последних редактировавшихся проектов выбрать нужный, например summa.

Изменение свойств визуальных компонентов

Упражнение 2.2. В проекте summa измените свойства компонентов Label2, Label3, Label4, задав для этих компонентов стиль шрифта «курсив», цвет шрифта «синий», размер символов «10 пунктов».

Для изменения свойств компонента Label2 в окне Инспектора объектов выберите компонент Label2. На странице Properties разверните группу свойств

Font двойным щелчком на свойстве Font. Откроется таблица свойств шрифта, содержащая свойства Color (цвет), Name (имя шрифта), Size (размер символа) и др. Щелкнув на свойстве Color, выберите в списке цвет clBlue (синий) – рис. 2.23.



Рис. 2.23. Определение свойства Font.Color

Примечание. Двойной щелчок в поле значений свойства Color открывает окно Цвет для выбора цвета символов. В этом окне можно выбрать или, щелкнув по кнопке Определить цвет, определить требуемый цвет в палитре, расположенной в правой части окна Цвет (рис. 2.24), и нажать кнопку ОК. Шестнадцатеричный код выбранного цвета, например \$00B8E61A, будет записан в поле значений свойства Font.Color.

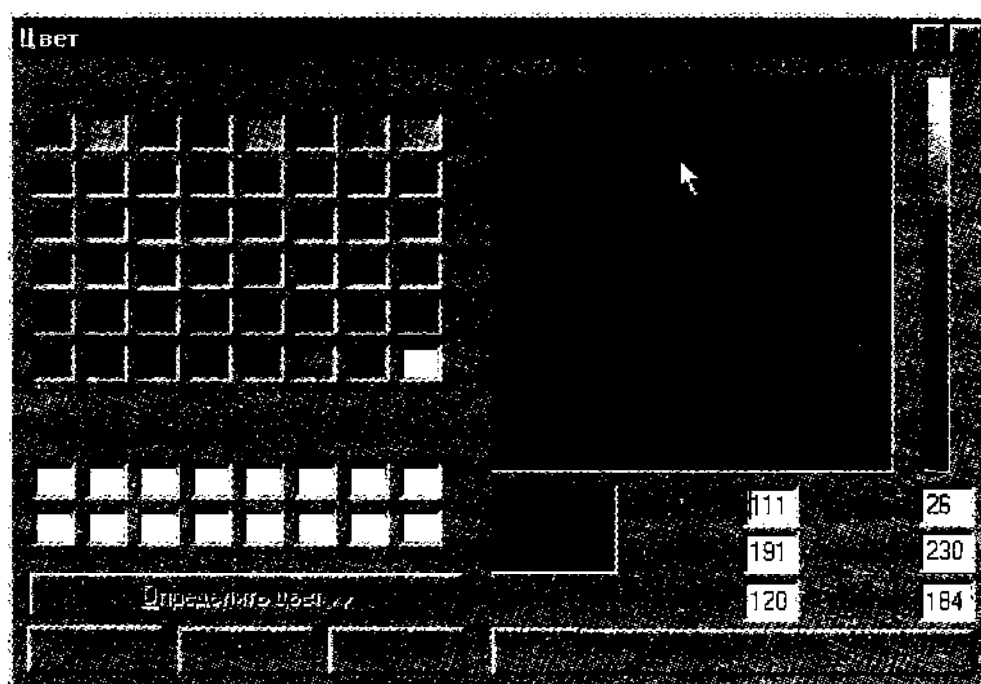


Рис. 2.24. Окно выбора цвета

Щелчком мышью в окне Инспектора объектов на свойстве Size активизируйте поле ввода значений и задайте размер символов 10 пунктов. Дважды щелкнув на свойстве Style (Стиль), раскройте дополнительный список подсвойств, в котором установите в True свойство fsItalic (Курсив). Обратите внимание на то, что на форме изменились цвет, размер символов и стиль начертания над-

писи Label2. Аналогичным образом измените цвет, размер символов и стиль начертания надписей Label3 и Label4.

Упражнение 2.3. Создайте приложение, реализующее ввод двух целых чисел, по щелчку на кнопке с символом « $\Rightarrow$ » вычисляющее результат операции вещественного деления и выводящее значение результата на экран.

Запустите ИСР Delphi и создайте новый проект командой File ► New ► Application. В результате создания проекта в окне Delphi будет раскрыто окно формы, на которой вы будете размещать визуальные компоненты проекта. Сохраните новый проект при помощи команды File ► Save Project As. В окне Save As создайте новую папку для файлов создаваемого проекта, например «Вещественное деление». Затем в окне Save As откройте созданную папку, задайте в поле Имя файла имя main и щелкните на кнопке Сохранить. После сохранения файла модуля main.pas откроется окно Save Project As, в котором следует задать имя файла проекта, например Delenie, а затем нажать кнопку Сохранить.

Измените свойства формы. Для изменения размеров формы захватите угол окна формы и, не отпуская левую кнопку мыши, перемещайте мышью, задавая требуемый размер формы. Измените надпись в заголовке формы Form1 с помощью Инспектора объектов. Для этого в окне Инспектора объектов выберите объект Form1, откройте страницу Properties (Свойства), затем выберите свойство Caption (Заголовок) и задайте новое значение этого свойства, например «Вещественное деление двух целых чисел».

Разместите на форме компоненты Edit1, Edit2, Edit3, Label1, Label2, Label3, Label4 и Button1 (рис. 2.25).

Удалите текст «Edit1», «Edit2», «Edit3» из соответствующих компонентов, для чего выберите объект в окне Инспектора объектов, на странице Properties выберите свойство Text и удалите текст (см. рис. 2.19).

Задайте для свойства Caption компонента Label1 значение «/». Измените размер символов компонента Label1, для чего в окне Инспектора объектов выберите в списке объект Label1, затем на странице Properties разверните список свойств Font. В списке свойств Font выберите свойство Size и задайте для него значение 20 (см. рис. 2.16). Аналогичным образом задайте для свойства Caption компонента Button1 значение « $\Rightarrow$ » и размер символов 20. Задайте для свойства Caption компонента Label2 значение «Делимое», для Label3 – «Делитель», для Label4 – «Частное». Зафиксируйте положение компонентов на форме, выбрав команду Edit ► Lock Controls. После редактирования свойств визуальных компонентов форма приложения будет выглядеть следующим образом (рис. 2.25):



Рис. 2.25. Вид формы приложения с визуальными компонентами

Нажав F12, откройте окно Редактор кода и просмотрите текст модуля, сгенерированный Delphi в процессе визуального проектирования формы. Обратите внимание на то, что в тексте программы еще не описаны переменные и нет процедуры вычисления операции вещественного деления.

Чтобы приложение выполняло вычисления при щелчке мышью по кнопке Button1 с изображением символа «=», следует написать код обработки этого события. Для создания кода обработчика события следует воспользоваться помощью Инспектора объектов. Выберите в его окне объект Button1, затем на странице События дважды щелкните мышью на пустом поле списка в событии OnClick. После этого окно Редактор кода немедленно получит фокус, в котором в разделе interface появится запись процедуры обработчика события:

```
procedure TForm1.Button1Click(Sender: TObject);
```

В разделе implementation появится текст заготовки этой процедуры:

```
procedure TForm1.Button1Click(Sender: TObject);  
begin  
...  
end;
```

Курсор будет расположен в пустой строке между ключевыми словами begin и end. Этот код является заготовкой обработчика события, которое будет выполняться при щелчке мышью по кнопке Button1. Остается только разместить в промежутке между begin и end необходимые операторы. Так как в задаче необходимо выполнить деление двух целых чисел, а результат их вещественного деления всегда будет вещественным числом, то в разделе описания переменных следует ввести следующее описание:

```
var  
    a,b : integer; {2 операнда - целые числа}  
    c : real;      {Частное - вещественное число}
```

В тело процедуры обработки события введите следующий текст:

```
    a:=StrToInt(Edit1.text); {Преобразование текстовой строки  
в целое число}  
    b:=StrToInt(Edit2.text);  
    c:=a / b; {Выполнение операции вещественного деления и  
присваивание результата переменной c}  
    Edit3.text:=FloatToStrF(c,ffGeneral,7,4);  
{Преобразование вещественного числа в текстовую строку}
```

Примечание. Поскольку свойства text объектов Edit1 и Edit2 имеют строковые значения, для их преобразования в целые числа используется стандартная функция Object Pascal – StrToInt. Так как свойству Edit3.text нужно присвоить значение вещественного частного, то для преобразования вещественного числа в текстовую строку используется функция FloatToStrF.

Для получения справочной информации о синтаксисе функции FloatToStrF отметьте курсором название функции и нажмите F1. В окне справки описано назначение параметров этой функции. Откомпилируйте и запустите программу на выполнение при помощи команды Run ► Run или щелчка мышью по кнопке Run в панели инструментов или нажатием клавиши F9. После этого на экране компьютера появится окно созданной вами формы с компонентами Edit1, Edit2, в которые можно вводить значения (рис. 2.26).



Рис. 2.26. Окно приложения, выполняющего операцию вещественного деления

При щелчке по кнопке Button1 «=» обработчик этого события будет выполнять операцию вещественного деления двух чисел и выводить частное в окне Edit3.

Завершите работу приложения любым из стандартных способов, например щелчком мышью по кнопке Заккрыть в правом верхнем углу окна.

Упражнение 2.4. Дополните программу обработчика события таким образом, чтобы перед операцией деления выполнялась проверка делителя на равенство нулю. В случае равенства делителя нулю вместо результата деления в окне Edit3 должно отображаться сообщение «На ноль делить нельзя!».

Вставьте в процедуру обработчика события TForm1.Button1Click оператор if then else, а в качестве условия выполнения операции вещественного деления задайте $b \neq 0$. Фрагмент программы, проверяющий условие отличия делителя от нуля и выполняющий деление, если условие выполняется, может быть записан следующим образом:

```
if b<>0 then {Проверка отличия делителя от 0}
begin
  c:=a / b; {Выполнение операции вещественного деления и
присваивание результата переменной c}
  Edit3.text:=FloatToStrF(c,ffGeneral,7,4);
  {Преобразование вещественного числа в текстовую строку}
end
else {Вывод сообщения}
  Edit3.text:='На ноль делить нельзя!'
end;
```

Сохраните текст модуля под именем main1.pas, воспользовавшись командой File ► Save As, затем в окне Save As задайте в поле Имя файла имя main1 и щелкните мышью по кнопке Сохранить. Сохраните измененный проект под именем delenie1 в той же папке при помощи команды File ► Save Project As,

затем в окне Save As задайте в поле Имя файла имя delenie1 и щелкните мышью по кнопке Сохранить.

При помощи команды View ► Units или клавиш Ctrl+F12 в окне View Unit выберите программу delenie1 и отредактируйте ее текст, изменив описание раздела uses следующим образом:

```
uses
  Forms,
  main1 in 'main1.pas' {Form1};
```

Откомпилируйте модифицированную программу командой Project ► Compile. Проверьте работу приложения, задавая в качестве делителя разные значения.

Упражнение 2.5. Выполните отладку приложения в пошаговом режиме. Для этого командой View ► Debug Windows ► Watches откройте окно просмотра значений переменных и выражений. Для включения переменной или выражения в окно просмотра выделите в окне Редактор кода переменную или выражение и нажмите Ctrl+F5.

Примечание. Среда разработки Delphi 6 поддерживает операции перетаскивания во время отладки. Например, из Редактора кода можно перенести выражение в окно Watch List, после чего это выражение останется в соответствующем списке.

Для копирования выражения $b \neq 0$ из окна Редактор кода в окно Watch List выделите в тексте модуля main1 выражение $b \neq 0$ и переместите его мышью в окно Watch List. Список переменных и выражений для просмотра в окне может выглядеть, как показано на рис. 2.27.

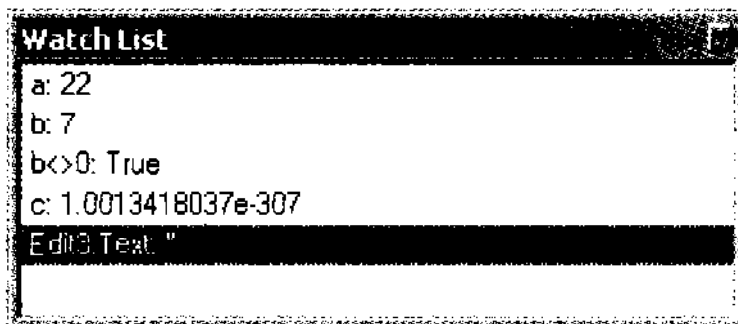


Рис. 2.27. Список переменных и выражений в окне Watch List

Выполните приложение в пошаговом режиме при помощи последовательных нажатий клавиши F7, наблюдая в окне Watch List за изменением значений переменных и выражений. Обратите внимание на то, что в случае ввода нуля в качестве делителя значение выражения $b \neq 0$ в окне Watch List становится равным False, а Edit3.text присваивается значение «На ноль делить нельзя!».

Упражнение 2.6. Чтобы сообщение «На ноль делить нельзя!» выводилось красным цветом, дополните код следующим образом:

```
begin
  Edit3.Font.Color:=clRed; {Установить красный цвет шрифта}
  Edit3.text:='На ноль делить нельзя!';
end;
```

Откомпилируйте и запустите на выполнение модифицированную программу. Проверьте ее работу, задавая в качестве делителя различные значения. Обратите внимание на то, что в случае ввода в качестве делителя нуля сообщение «На ноль делить нельзя!» в окне Edit3 выводится красным цветом. Причем если цвет шрифта был изменен, то он так и остается красным, даже если при следующей итерации делитель не будет равен нулю. Чтобы восстановить черный цвет шрифта в окне Edit3, дополните код модуля main1 перед оператором if then else следующей строкой:

```
Edit3.Font.Color:=clBlack; {Установить черный цвет шрифта}
```

Откомпилируйте и запустите приложение на выполнение. Проверьте его работу, задавая в качестве делителя разные значения. Убедитесь, что процедура обработчика события нажатия на кнопку Button1 правильно обрабатывает все варианты значений делителя и изменяет цвет шрифта в окне Edit3. Обратите внимание на то, что текст сообщения «На ноль делить нельзя!» отображается в окне Edit3 не целиком. Закройте окно приложения.

Упражнение 2.7. Измените программу таким образом, чтобы ширина окна Edit3 изменялась в ходе выполнения программы и при выводе сообщения «На ноль делить нельзя!» ширина окна Edit3 обеспечивала бы отображение всего текста сообщения.

Переключитесь в окно Инспектора объектов, выберите в списке объектов Edit3 и проверьте значение свойства Width (ширина). В нашем примере оно равно 81 пикселю. Для того чтобы ширина окна Edit3 изменялась в ходе выполнения программы и обеспечивала отображение всего текста сообщения «На ноль делить нельзя!», отредактируйте код модуля main1, вставив перед оператором вывода сообщения Edit3.text:='На ноль делить нельзя!' строку

```
Edit3.Width:=130; {Увеличение ширины Edit3}
```

Чтобы восстановить значение ширины окна Edit3, заданное при проектировании формы, введите перед оператором if then else следующую строку:

```
Edit3.Width:=81; {Восстановить первоначальную ширину Edit3}
```

Откомпилируйте и запустите на выполнение отредактированный проект. Проверьте работу программы, задавая в качестве делителя разные значения. Убедитесь, что процедура обработчика события нажатия на кнопку Button1 правильно обрабатывает все варианты значений делителя и изменяет цвет шрифта и ширину окна Edit3 при выводе сообщения «На ноль делить нельзя!» (рис. 2.28).



Рис. 2.28. Вид окна приложения с изменяющимся по ширине окном Edit3

Обратите внимание на то, что в случае корректного ввода делителя, не равного нулю, ширина окна Edit3 уменьшается до значения, заданного при проектировании формы. Закройте окно приложения и сохраните внесенные в проект изменения, щелкнув мышью по кнопке Save All в панели инструментов Delphi.

Упражнение 2.8. Измените программу таким образом, чтобы текст «На ноль делить нельзя!» выводился в отдельном окне сообщения.

Примечание. Для вывода текста в отдельном окне сообщения используйте вызов процедуры ShowMessage(const Msg: string), которая отображает модальное окно сообщения с кнопкой ОК. Текст сообщения задайте параметром Msg, в нашем примере – «На ноль делить нельзя!».

Чтобы не внести изменения в проект, созданный в предыдущих заданиях, сохраните проект под новым именем delenie2, а текст программы – под именем main2.

Отредактируйте текст тела модуля main2 следующим образом:

```
begin
    a:=StrToInt(Edit1.text);  {Преобразование текстовой строки
в целое число}
    b:=StrToInt(Edit2.text);
    Edit3.text:='';
    if b=0 then ShowMessage('На ноль делить нельзя!')
                                {Вывод текста в окне сообщения}
    else                        {Делитель отличен от нуля}
        begin
            c:=a/b;             {Выполнение вещественного деления}
            Edit3.text:=FloatToStr(c);
            {Преобразование вещественного числа в текстовую строку}
        end;
end;
```

Выберите команду View ► Units, в окне View Unit выберите программу delenie2 и отредактируйте ее текст, изменив описание раздела uses следующим образом:

```
uses
    Forms,
    main2 in 'main2.pas' {Form1};
```

Сохраните проект. Откомпилируйте и запустите приложение на выполнение. Проверьте его работу, задавая в качестве делителя разные значения. Обратите внимание на то, что в случае ввода в качестве делителя нуля в модальном окне сообщения выводится сообщение «На ноль делить нельзя!» (рис. 2.29).

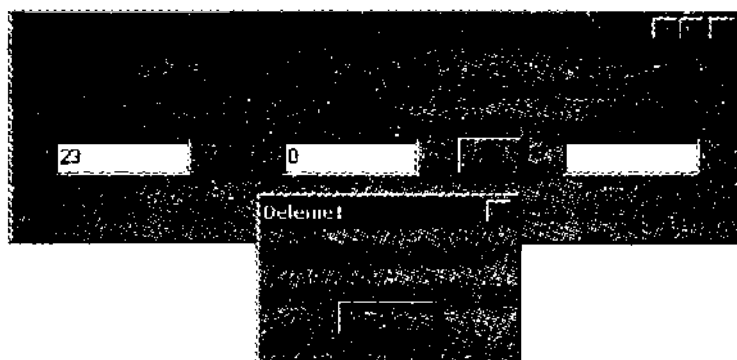


Рис. 2.29. Вывод сообщения в модальном окне

Для продолжения работы с приложением необходимо закрыть окно сообщения, щелкнув мышью по кнопке ОК.

Упражнение 2.9. Измените значок приложения Delenie2. Для вызова редактора изображений выберите в меню Delphi команду Tools ► Image Editor. Изучите окно Image Editor. Как видно на рис. 2.30, оно очень напоминает окно стандартного графического редактора Paint. В окне редактора изображений командой File ► Open откройте файл ресурсов приложения. В диалоговом окне Открытие файла выберите папку с проектом (в нашем примере – Вещественное деление), в поле Тип файла выберите вариант Resource/DCR files (*.res;*.dcr), в списке файлов выберите файл ресурсов приложения (в нашем примере – Delenie2.res) и нажмите кнопку Открыть.

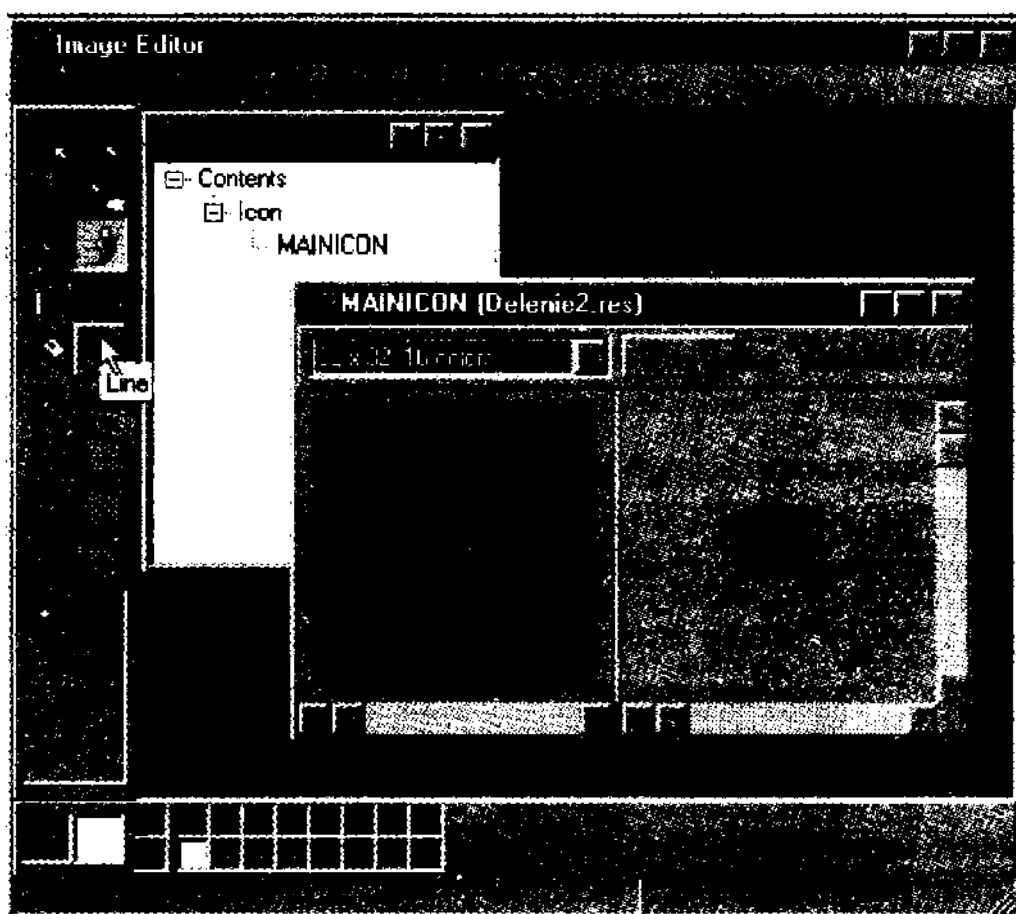


Рис. 2.30. Редактирование значка приложения в Image Editor

Как видно на рис. 2.30, в окне структуры файла ресурсов имеется стандартный узел Icon. Щелкнув мышью по символу + в строке Icon, разверните узел. Щелкнув мышью по вершине MAINICON, соответствующей стандартному значку приложения, откройте значок для редактирования.

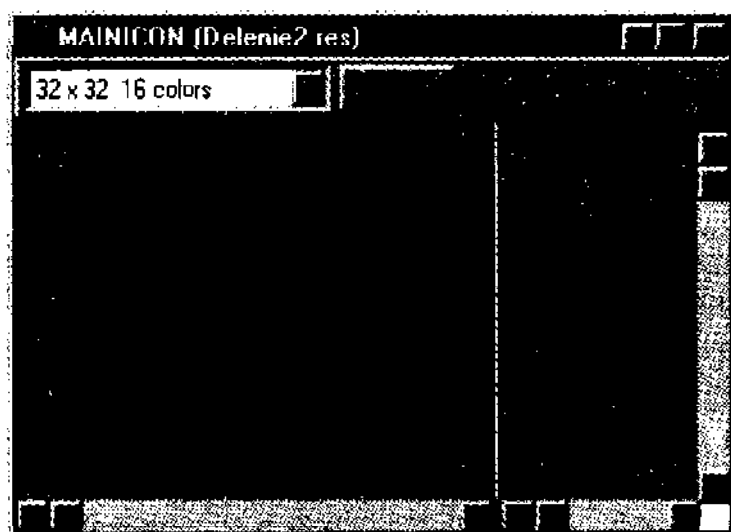


Рис. 2.31. Отредактированный значок

Закройте окно редактирования значка командой File ► Close. Сохраните изменения в файле ресурсов командой File ► Save. Закройте окно редактора Image Editor командой File ► Exit (рис. 2.31).

В ИСР Delphi откройте проект приложения Delenie2 при помощи команды File ► Open Project, в окне Open Project откройте нужную папку и, выбрав в ней файл проекта приложения, например Delenie2, нажмите кнопку Открыть. Откомпилируйте и запустите приложение на выполнение. Обратите внимание на то, что значок приложения изменился (рис. 2.32).



Рис. 2.32. Окно приложения с измененным значком

Упражнение 2.10. Создайте приложение, обеспечивающее ввод двух целых чисел и выполнение над ними арифметических операций сложения, вычитания, умножения и вещественного деления. Для выбора операции используйте переключатели, вывод сообщения об ошибке при вводе делителя, равного нулю, выполните в отдельном окне сообщений.

После запуска ИСР Delphi создайте новый проект командой File ► New ► Application. Сохраните новый проект командой File ► Save Project As. В окне Save Unit1 As создайте новую папку для файлов создаваемого проекта, например «Калькулятор». Затем откройте созданную папку в окне Save Unit1 As, в

поле Имя файла задайте имя модуля Main и нажмите кнопку Сохранить. После сохранения файла модуля main.pas откроется окно Save Project As. В этом окне в поле Имя файла задайте имя файла проекта, например Calculator, и нажмите кнопку Сохранить.

Задайте для свойства Form1.Caption значение «Калькулятор». Разместите на форме Form1 компоненты Edit1, Edit2 и Label1, Label2. Задайте для свойства Caption компонентов Label1 и Label2 значения «Операнд». При нажатой клавише Shift выделите компоненты Edit1, Edit2, Label1, Label2 и, вызвав контекстное меню, командой Position ► Align ► Horizontal ► Centers выровняйте их по горизонтали.

Для выбора одной из четырех арифметических операций над операндами используйте переключатели, размещенные на панели RadioGroup.

Примечание. Компонент RadioGroup позволяет отображать и редактировать поля с ограниченным количеством возможных значений.

Разместите на форме Form1 компонент RadioGroup1 из палитры Standard. Задайте для свойства Caption компонента RadioGroup1 значение «Операция». Так как количество переключателей в группе и надписи около них определяются свойством Items, выберите в Инспекторе объектов компонент RadioGroup1, а на странице свойств выберите свойство Items (Список элементов). В окне String List Editor введите список элементов – символов арифметических операций: +, −, *, / (рис. 2.33).

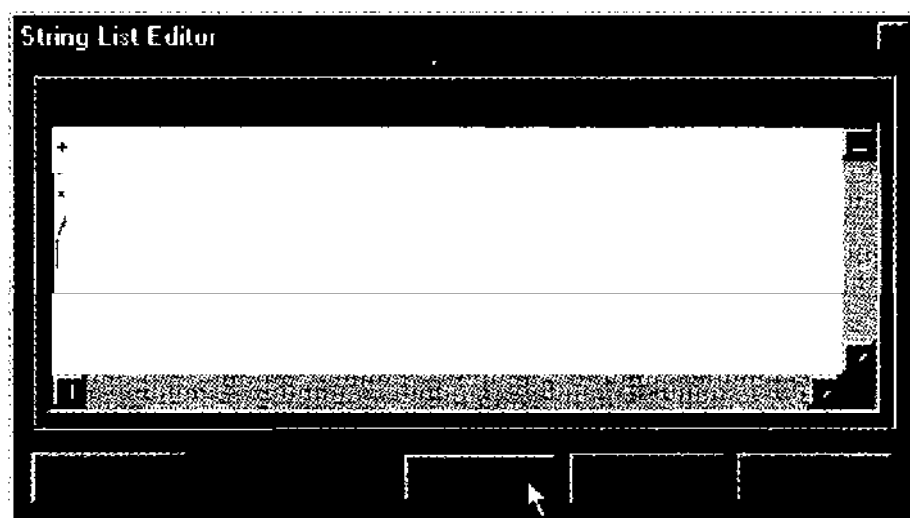


Рис. 2.33. Редактирование списка элементов в панели RadioGroup

Щелкнув мышью по кнопке ОК, завершите формирование списка арифметических операций.

В окне Инспектора объектов задайте для свойства RadioGroup.ItemIndex значение 0, чтобы сделать первую кнопку (сложение) выбранной по умолчанию.

Примечание. Определить во время выполнения, какая кнопка выбрана в данный момент, можно по индексу ItemIndex (0 означает первую кнопку, 1 – ни одна кнопка не выбрана).

Задайте размер символов компонента RadioGroup, установив для свойства RadioGroup.Font.Size значение 12 пунктов.

Разместите на форме Form1 кнопку Button1 и задайте для нее надпись «Вычислить». Разместите на форме Form1 компоненты Edit3, Label3 и задайте для свойства Label3.Caption значение «Результат». Удалите текст «Edit1», «Edit2», «Edit3» в соответствующих компонентах. Для этого выберите объект в окне Инспектора объектов, на странице Properties выберите свойство Text и удалите текст. Выровняйте компоненты Edit3 и Label3, отцентрировав их по горизонтали. Зафиксируйте положение компонентов на форме, выбрав в меню Delphi команду Edit ► Lock Controls. Форма будет выглядеть, как показано на рис. 2.34.

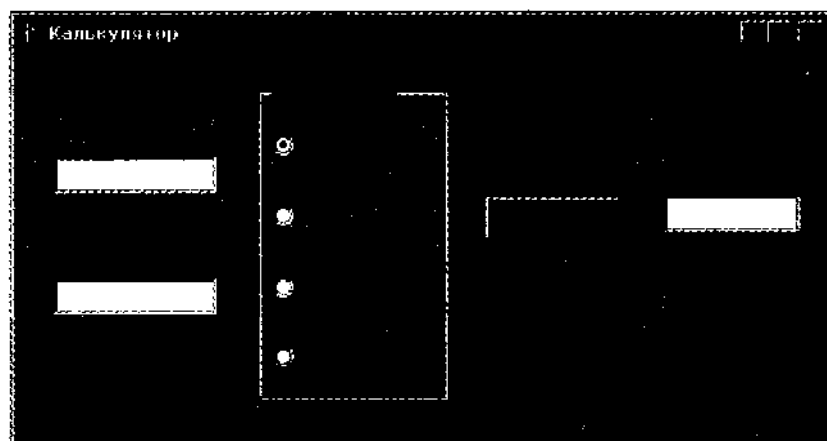


Рис. 2.34. Вид формы приложения с компонентами

Для обработки щелчка по кнопке Button1 с изображением символа «=» в окне Инспектора объектов выберите объект Button1, затем на странице События дважды щелкните мышью на пустом поле списка в событии OnClick. После этого в разделе interface модуля Main.pas появится процедура обработчика события `procedure TForm1.Button1Click(Sender: TObject);`, а в разделе implementation появится текст заготовки этой процедуры:

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

```
...
```

```
end;
```

Для решения задачи проведения арифметических операций над двумя целыми операндами введите в текст процедуры TForm1.Button1Click следующее описание:

```
var  
    a,b : integer; { 2 операнда - целые числа}  
    c : real; {Результат арифметических операций}
```

Примечание. Тип переменной c — вещественное число — выбран из-за того, что в результате вещественного деления целых чисел результат будет числом вещественного типа.

Так как в процедуре обработки нажатия кнопки Button1 «Вычислить» должно быть четыре варианта реализации, по одному для каждой арифметической

операции, то следует записать выбранный вариант вычисления с помощью оператора case. Для получения подсказки Delphi по синтаксису оператора case введите слово case, укажите на него мышью и нажмите F1. В списке найденных разделов выберите раздел Case statements и щелкните мышью по кнопке Показать. Для просмотра информации используйте полосу прокрутки в окне Delphi Help. Для копирования примера кода программы в окне Delphi Help выделите текст в окне справки и, нажав правую кнопку мыши, вызовите контекстное меню, затем в контекстном меню выберите команду Копировать (рис. 2.35).

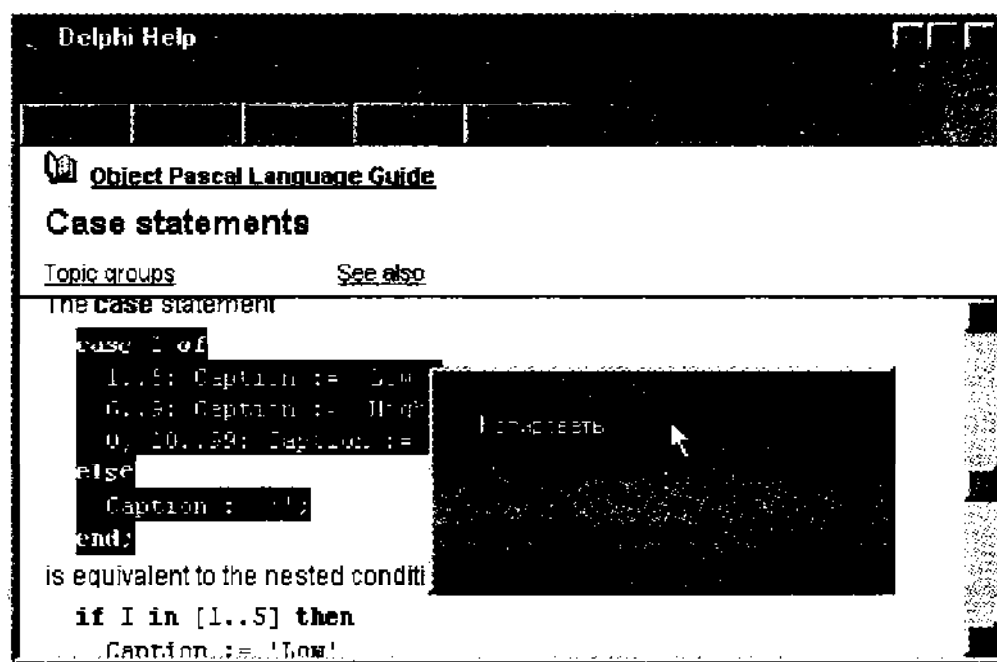


Рис. 2.35. Копирование примера кода программы в окне Delphi Help

В окне Редактор кода укажите в основном теле процедуры обработки события место вставки скопированного фрагмента и, вызвав контекстное меню, выберите команду Paste (Вставить).

В основном теле процедуры обработки события отредактируйте текст оператора case и введите текст остальных операторов следующим образом:

```
begin
  a:=StrToInt(Edit1.text); {Преобразование текстовой строки
                           в целое число}

  b:=StrToInt(Edit2.text);

  Edit3.text:=''; {Очистить от результата предыдущих
вычислений}

  {Выбор операции в зависимости от значения свойства
RadioGroup1.ItemIndex}
  Case RadioGroup1.ItemIndex of
    0 : c:=a+b; {Сложение}
    1 : c:=a-b; {Вычитание}
```

```

2 : c:=a*b; {Умножение}
3 : {деление} if b=0 then ShowMessage('На ноль делить
нельзя!')
else c:=a/b;
end;
if RadioGroup1.ItemIndex <> 3 then {Вывод результата
операций}
Edit3.text:=FloatToStrF(c,ffGeneral,10,4) {Преобразование
результата вещественного деления в текстовую строку}
else if b<>0 then
Edit3.text:=FloatToStrF(c,ffGeneral,10,4)
end;

```

Как видно из текста процедуры, в операторе case выбирается вариант вычислений, а вывод результата вычислений выполняется оператором, следующим после оператора case, причем для случая $b=0$ результат не вычисляется и в окне Edit3 не выводится, а выводится сообщение в окне сообщений.

Откомпилируйте и запустите приложение на выполнение. На рис. 2.36 показано, как будет выглядеть окно приложения.

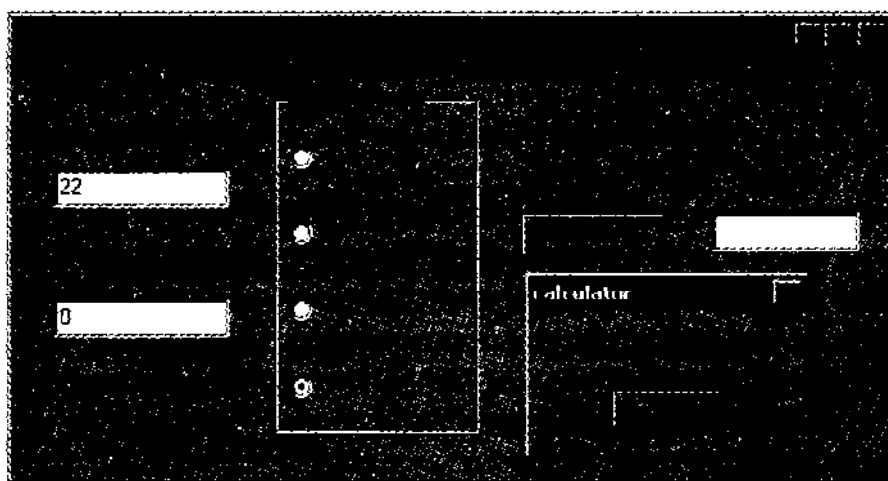


Рис. 2.36. Вид окна приложения с сообщением об ошибке ввода делителя

Проверьте работу приложения для случая, когда не заданы значения операндов. В этом случае выводится окно сообщения о некорректности значения операнда, закрыть которое можно нажатием кнопки ОК.

Упражнение 2.11. Закройте окно приложения и отредактируйте текст модуля таким образом, чтобы перед выполнением вычислений выполнялась проверка, заданы ли значения операндов. Если значения операндов не заданы, то следует вывести сообщение об этом в отдельном окне. Поместите перед оператором присваивания `a:=StrToInt(Edit1.text);` строку

```
if (Edit1.Text<>'') and (Edit2.Text<>'') then begin,
```

которая проверяет значения свойств Edit1.Text и Edit2.Text. Если эти значения не пустые, то вычисляется арифметическая операция, в противном случае управление передается следующему фрагменту программы, который нужно вставить перед последним оператором end;;

```
end
```

```
else ShowMessage('Не заданы значения');
```

Сохраните, откомпилируйте и запустите приложение на выполнение. Проверьте работу приложения для случая, когда не заданы значения операндов, и убедитесь, что в этом случае выводится сообщение в отдельном окне (рис. 12.37).

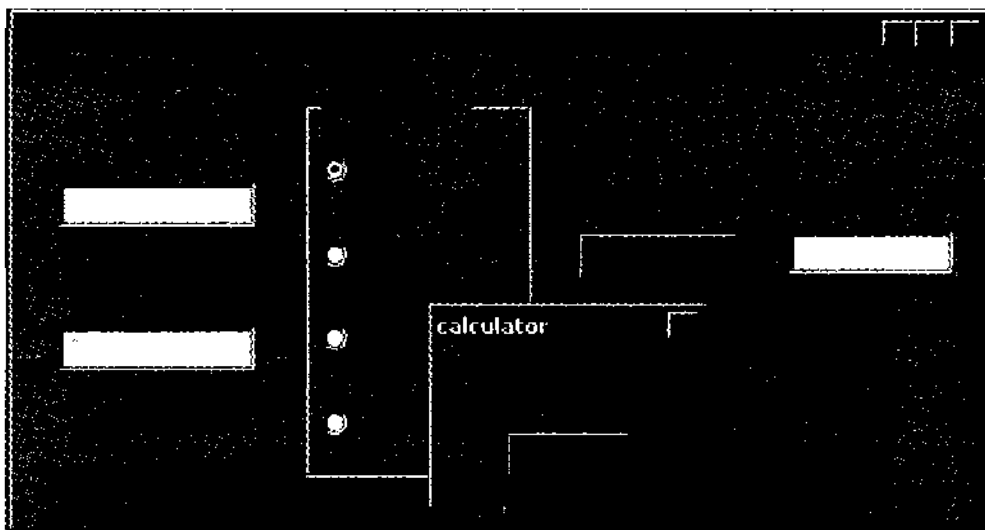


Рис. 2.37. Окно сообщения о некорректности значений операндов

Попробуйте ввести в качестве значений операндов не цифры, а символы, например «а» и «б». Щелкнув на кнопке Button1, убедитесь в том, что приложение завершается и выдаст сообщение об ошибке, которая возникает из-за того, что приложение пытается преобразовывать символы в число. Для предупреждения данной ошибки введем обработку события нажатия на клавишу в окне Edit1 и Edit2, чтобы запретить ввод любых символов, кроме цифр от 0 до 9 и знаков -, +.

Для создания процедуры обработчика события нажатия на клавишу в окне Edit1 выберите в окне Инспектора объектов компонент Edit1 и на странице Events (События) дважды щелкните левой кнопкой мыши на пустом поле списка в событии OnKeyPress. После этого окно Редактор кода немедленно получит фокус и в разделе interface появится запись процедуры обработчика события:

```
procedure Edit1KeyPress(Sender: TObject; var Key: Char);
```

Параметр Key в обработчике этого события соответствует символу нажатой клавиши.

Примечание. При этом различаются символы в верхнем и нижнем регистрах и символы кириллицы и латиницы. Клавиши, не имеющие соответствующих им кодов ASCII

(Shift, Alt, Ctrl и функциональные клавиши), не вызывают этого события. Поэтому нажатие таких комбинаций клавиш, как, например, Shift+A, генерирует только одно событие OnKeyPress, при котором параметр Key равен «A». Для того чтобы распознавать подобные комбинации клавиш, следует использовать обработчики событий OnKeyDown и OnKeyUp.

В разделе *implementation* появится текст заготовки этой процедуры:

```
procedure TForm1.Edit1KeyPress(Sender: TObject; var Key:
Char);
begin
...
end;
```

Вставьте в тело процедуры следующий оператор:

```
if not (Key in ['0'..'9', '+', '-']) then Key:=#0;
```

Действие оператора сводится к сравнению значения переменной Key с множеством значений ['0'..'9', '+', '-']. Если символ нажатой клавиши не входит во множество, то Key присваивается значение нулевого символа (#0). Таким образом, в окне Edit1 будет отображаться текст, состоящий из цифр и знаков «+» и «-».

Аналогичным образом создайте процедуру обработчика события нажатия на клавишу в окне Edit2. В итоге получится следующий текст модуля приложения:

```
unit main;
interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics,
  Controls, Forms,
  Dialogs, StdCtrls, ExtCtrls;
type
  TForm1 = class(TForm)
    Edit1: TEdit;
    Edit2: TEdit;
    Label1: TLabel;
    Label2: TLabel;
    RadioGroup1: TRadioGroup;
    Button1: TButton;
    Edit3: TEdit;
    Label3: TLabel;
  procedure Button1Click(Sender: TObject);
  procedure Edit1KeyPress(Sender: TObject; var Key: Char);
  procedure Edit2KeyPress(Sender: TObject; var Key: Char);
```



```

private
  { Private declarations }
public
  { Public declarations }
end;
var
  Form1: TForm1;
implementation
{$R *.dfm}
procedure TForm1.Button1Click(Sender: TObject);
    {Обработчик события щелчка на кнопке
Button1}
    var
      a,b : integer; {2 операнда - целые числа}
      c : real;      {Результат арифметических операций}
    begin
      if (Edit1.Text<>'') and (Edit2.Text<>'') then
        begin
          {Если значения заданы}
          a:=StrToInt(Edit1.text); {Преобразование текстовой
строки в целое число}
          b:=StrToInt(Edit2.text);
          Edit3.text:='';          {Очистить от результата
предыдущих вычислений}
          {Выбор операции в зависимости от значения свойства
RadioGroup1.ItemIndex}
          Case RadioGroup1.ItemIndex of
            0 : c:=a+b; {Сложение}
            1 : c:=a-b; {Вычитание}
            2 : c:=a*b; {Умножение}
            3 : {Деление} if b=0 then ShowMessage('На ноль делить
нельзя!')
                else c:=a/b;
          end;
          if RadioGroup1.ItemIndex <> 3 then {Вывод результата
операций}
            Edit3.text:=FloatToStrF(c,ffGeneral,10,4)
            Преобразование результата вещественного деления в текстовую
строку}
          else if b<>0 then
            Edit3.text:=FloatToStrF(c,ffGeneral,10,4)
          end
        end
      end;
    end;

```

```
    else ShowMessage('Не заданы значения');  
end;  
procedure TForm1.Edit1KeyPress(Sender: TObject; var Key:  
Char);  
begin    {Разрешить ввод в Edit1 только цифр}  
    if not (Key in ['0'..'9','+','-']) then Key:=#0;  
end;  
procedure TForm1.Edit2KeyPress(Sender: TObject; var Key:  
Char);  
begin    {Разрешить ввод в Edit2 только цифр}  
    if not (Key in ['0'..'9','+','-']) then Key:=#0;  
end;  
end.
```

Сохраните, откомпилируйте и запустите приложение на выполнение. Попробуйте ввести в качестве значений операндов не цифры, а символы и убедитесь, что созданные ранее обработчики событий OnKeyPress для Edit1 и Edit2 не позволяют пользователю ввести нецифровую информацию. Закройте окно приложения.

Примечание. При обработке таких исключительных ситуаций Delphi использует специальные объекты – исключения, характеризующие возникшую в программе исключительную ситуацию.

Что нового вы узнали?

Изучив эту главу, вы узнали:

- состав, назначение компонентов и основные возможности интегрированной среды разработки приложений Delphi 6;
- состав и назначение отдельных файлов проекта в Delphi;
- назначение и порядок изменения свойств визуальных компонентов Edit, Label, Button, RadioGroup;
- назначение и формат записи стандартных функций преобразования записи числа в текст и наоборот;
- назначение и формат вызова стандартной процедуры ShowMessage;
- назначение и порядок использования графического редактора Image Editor для редактирования файла ресурсов приложения.

Вы научились:

- использовать при проектировании приложения компоненты интегрированной среды разработки приложений Delphi 6: меню, палитру компонентов, Инспектор объектов, справочную систему и др.;
- создавать форму с размещением на ней визуальных и невидимых компонентов;
- использовать Инспектор объектов для изменения свойств объектов;

- использовать Редактор кода для создания процедур обработчиков событий;
- использовать Редактор изображений Image Editor для редактирования файлов ресурсов;
- создавать, компилировать и отлаживать приложения с графическим интерфейсом;
- выполнять операции с проектом Delphi: сохранять, открывать, добавлять формы.

Контрольные вопросы и задания

Вопросы

1. Что такое Delphi? Почему пакет получил такое название?
2. Какие компоненты входят в интегрированную среду разработки приложений Delphi?
3. Перечислите основные компоненты окна среды Delphi и укажите их назначение.
4. Как получить подсказку Delphi по элементам окна, компонентам проектируемого графического интерфейса?
5. Каково назначение страниц Properties и Events в окне Инспектора объектов?
6. Как разместить компонент на форме?
7. Какими способами можно изменять свойства компонента? Приведите примеры из текста.
8. Перечислите состав проекта Delphi. Опишите назначение различных файлов.
9. Каково назначение обработчиков событий? Каким образом можно инициировать создание процедуры-обработчика события?
10. Опишите порядок получения справки по синтаксису языка Object Pascal в ИСР Delphi.
11. Как перейти от редактирования кода в модуле к редактированию кода в основной программе?
12. Как открыть окно просмотра значений переменных и выражений Watch List? Как включить в это окно переменную или выражение?
13. Чем различается пошаговое выполнение программы в режимах Step Over (F8) и Trace Into (F7)?
14. Каково назначение процедуры ShowMessage? Что выступает в качестве параметра процедуры?
15. Какое инструментальное средство включено в интегрированную среду Delphi для редактирования файлов ресурсов?
16. Опишите назначение компонента RadioGroup. Как задать список элементов-переключателей в панели RadioGroup?
17. По какому свойству RadioGroup определяется выбранный переключатель? Как задать этому свойству значение?
18. Какие ситуации, возникающие во время выполнения приложения, относятся к исключительным? Каковы способы обработки исключительных ситуаций?

Задания

1. Откройте справку Delphi и изучите справочную информацию о компонентах палитры Standard.

2. В справке Delphi найдите и изучите информацию о числовых типах данных языка Object Pascal.

3. В справке Delphi найдите и изучите информацию об арифметических операциях языка Object Pascal.

4. Измените программу вычисления операции вещественного деления двух целых чисел, чтобы она вычисляла результат целочисленного деления и остаток от деления.

5. Создайте приложение, которое по заданному натуральному N вычисляло бы числа Фибоначчи, не превышающие заданного N, а также определяло бы количество найденных чисел и их сумму. Вариант вычислений должен определяться по включению соответствующего флажка (рис. 2.38). Список чисел Фибоначчи следует вывести в компоненте ListBox.

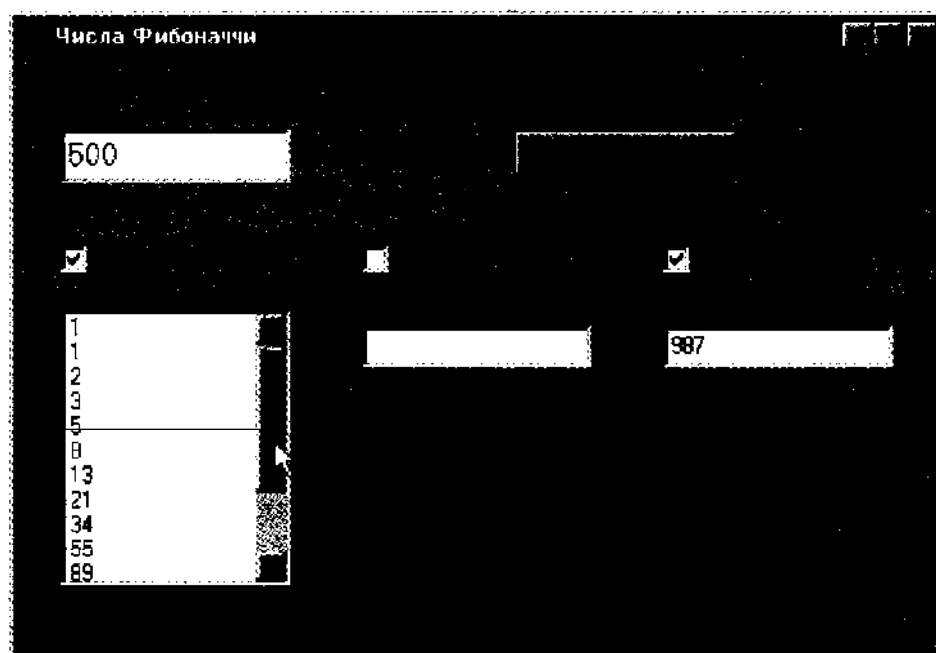


Рис. 2.38. Окно приложения, вычисляющего числа Фибоначчи

Примечание. Числа Фибоначчи – это числовая последовательность, где каждый последующий член ряда равен сумме двух предыдущих, т.е. 1, 1, 2, 3, 5, 8, 13, 21, 34, 55 и т.д.

6. В справке Delphi найдите и изучите информацию о функции Random в языке Object Pascal.

В справке Delphi найдите и изучите информацию об операторах повтора While, Repeat, For в языке Object Pascal.

Создайте приложение «Угадай число», в котором компьютер будет загадывать случайное целое число в диапазоне от 0 до 10, а пользователь должен угадать это число. Ошибка должна сопровождаться сообщением типа «Загаданное число меньше» или «Загаданное число больше», выводимым в отдельном окне. В случае правильного ответа следует вывести в отдельном окне сообщение «Вы угадали» и указать число попыток.

ГЛАВА 3

Приложения для обработки строк, массивов и файлов

В этой главе мы рассмотрим использование интегрированной среды программирования Delphi 6 при создании приложений для обработки структурированных типов данных: строк, массивов и файлов.

После изучения главы вы будете знать:

- Особенности обработки строк типа String.
- Программирование создания и обработки линейных массивов.
- Программирование сортировки массивов.
- Порядок использования компонента *StringGrid* для представления двумерных массивов.
- Порядок создания приложений для обработки файлов.

Обработка строк типа String

В языке Object Pascal Delphi 5–6 имеется четыре типа строк: String, ShortString, AnsiString, WideString. В справке Object Pascal вы можете изучить эти типы более подробно. Здесь же рассмотрим использование родового типа String.

Упражнение 3.1. Создайте приложение, предлагающее пользователю ввести строку, определяющее количество символов в ней и выводящее результат, согласуя окончания различных падежей слова «символ» с числом символов, например: «В тексте 1 символ», «В тексте 32 символа», «В тексте 47 символов».

Примечание. Для согласования окончания слова с количеством символов используйте оператор case.

Создайте форму и разместите на ней компоненты Edit1, Label1, Label2, Button1, как показано на рис. 3.1. Задайте значения свойств Label1.Caption – «Введите текст», Label2.Caption – «В тексте», Button1.Caption – «Вычислить». Удалите текст «Edit1» из соответствующего компонента. Выровняйте компоненты на форме.

Для вычисления количества символов во введенной строке и вывода результатов создайте процедуру обработчика события щелчка мышью по кнопке Button1. Выберите в окне Инспектора объектов объект Button1, затем на странице событий сделайте двойной щелчок мышью на пустом поле списка в

событии OnClick. После этого в окне Редактор кода будет сгенерирована заготовка процедуры обработчика события `procedure TForm1.Button1Click(Sender: TObject);`.

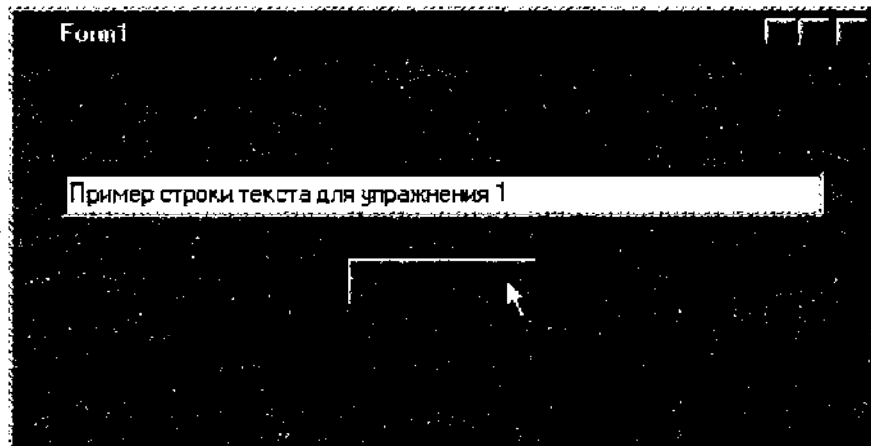


Рис. 3.1. Вид окна приложения, подсчитывающего число символов в тексте

Введите в текст процедуры следующее описание переменных:

```
var
  N : integer; {Последняя цифра}
  S : string; {Изменяемое слово «символ»}
```

Для ознакомления со справкой Object Pascal о процедурах и функциях обработки строк введите в окне Редактор кода слово `Length` и нажмите F1. В списке Найденные разделы выберите тему `Length function` и нажмите кнопку Показать. В окне справки, прочитав информацию о применении функции `Length`, щелкните мышью по ссылке `Example` для просмотра примера. При помощи кнопки Назад в панели инструментов окна Delphi Help вернитесь в предыдущее окно. Для просмотра справочной информации о других процедурах и функциях обработки строк щелкните по ссылке `See also`. Выбирая в списке Найденные разделы нужную процедуру или функцию, щелкайте мышью по кнопке Показать для просмотра справки. После изучения справочной информации закройте окно справки.

Введите в окне Редактор кода текст тела процедуры обработки текста:

```
begin
  N:=Length(Edit1.Text);           {Определить число символов}
  Label2.Caption:='В тексте ';
  if N>20 then N:=N mod 10;         {Выделить последнюю цифру}
  case N of                         {В зависимости от последней цифры}
    1 : S:=' символ';              {Изменять окончание слова}
    2..4 : S:=' символов';
    0,5..20 : S:=' символов';
  end;
  Label2.Caption:=Label2.Caption+
    IntToStr(Length(Edit1.Text))+S; {Вывести результат}
end;
```

Сохраните файлы проекта и программного модуля, откомпилируйте и запустите программу на выполнение. Проверьте работу приложения, задавая текст с разным количеством символов и проверяя правильность вывода результатов работы.

Упражнение 3.2. Создайте приложение, предлагающее пользователю ввести строку текста, а затем заменяющее символы в тексте и подсчитывающее количество замененных символов. Вариант замены символов должен определяться по положению соответствующего флажка CheckBox (рис. 3.2). Создайте форму, для свойства Caption задайте значение «Подсчет и замена символов». На форме разместите компоненты Edit1, Edit2, Edit3, CheckBox1, CheckBox2, CheckBox3, Label1, Label2, Label3, Button1. Задайте значения свойств: Label1.Caption – «Исходный текст на русском языке», Label2.Caption – «Измененный текст», Label3.Caption – «Число замен», CheckBox1.Caption – «Подсчитать число замен», CheckBox2.Caption – «Заменить «а» на «б»», CheckBox3.Caption – «Заменить пробелы на тире», Button1.Caption – «Выполнить». Удалите текст «Edit1», «Edit2», «Edit3» из соответствующих компонентов. Выровняйте компоненты на форме (см. рис. 3.2).

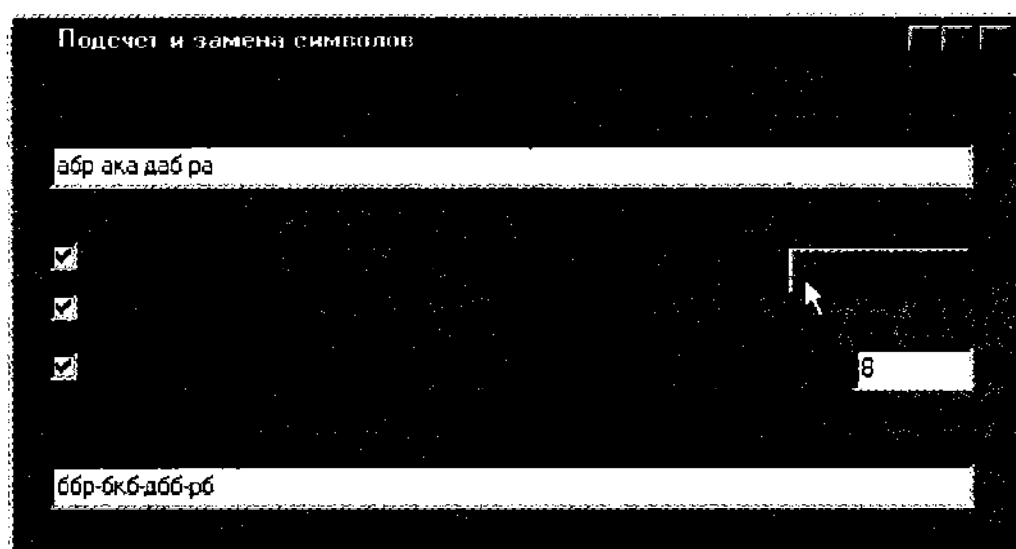


Рис. 3.2. Вид окна приложения, выполняющего замену символов в тексте

Замену символов в тексте и подсчет количества замененных символов опишите в процедуре обработчика события щелчка мышью по кнопке Button1. Для создания процедуры обработчика события выберите в окне Инспектора объектов объект Button1, затем на странице События произведите двойной щелчок на пустом поле списка в событии OnClick. После этого в окне Редактор кода в заготовку процедуры обработчика события procedure TForm1.Button1Click(Sender: TObject); введите следующее описание переменных:

```
var
  var S : string; {Строка текста}
      N : byte;    {Количество замен символов}
```

Примечание. Для поиска символа в строке используйте функцию Pos языка Object Pascal. Для программирования варианта замены символов используйте значение свой-

ства `Checked` компонента `CheckBox`. Если флажок `CheckBox1` установлен, то свойство `CheckBox1.Checked` приобретает значение `True`.

С учетом сказанного тело процедуры может быть записано следующим образом:

```
N:=0; {Обнулить число букв «а»}  
Edit3.Text:='';  
Edit2.Text:='';  
S:=Edit1.Text; {Присвоить S значение текстовой строки}  
if CheckBox2.Checked then {Если включен флажок CheckBox2}  
  while Pos('a', S) > 0 do {Если в строке найдена буква «а»}  
    begin  
      N:=N+1; {Увеличить счетчик замен на 1}  
      S[Pos('a', S)] := 'б'; {Заменить букву «а» буквой «б»}  
    end;  
  if CheckBox3.Checked then  
    while Pos(' ', S) > 0 do {Если в строке найден пробел}  
      begin  
        N:=N+1; {Увеличить счетчик замен на 1}  
        S[Pos(' ', S)] := '-'; {Заменить пробел символом «-»}  
      end;  
  if CheckBox1.Checked then Edit3.Text:=IntToStr(N);  
                                {Вывод числа замен символов}  
  Edit2.Text:=S;                {Вывод измененного текста}  
end;
```

Сохраните файлы проекта и программного модуля, откомпилируйте и запустите программу на выполнение. Проверьте работу приложения, задавая различный текст на русском языке с разным количеством букв «а» и пробелов. Проверьте правильность вывода результатов работы и закройте окно приложения.

Создание и обработка линейного массива

Интегрированная среда разработки Delphi обеспечивает программиста средствами создания приложений обработки массивов – структурированных типов данных, состоящих из фиксированного числа элементов, имеющих один и тот же тип.

Упражнение 3.3. Создайте приложение, которое предлагает пользователю задать массив линейного размера, заполняет этот массив случайными целыми числами, выводит список элементов массива, а затем по выбору пользователя определяет минимальный и максимальный элементы массива, сумму всех элементов и количество положительных элементов. Алгоритмы обработки массивов были подробно рассмотрены при изучении программирования на языке Turbo Pascal.

Создайте форму, для свойства `Caption` задайте значение «Создание и обработка массива». На форме расположите компоненты `Edit1` и `Edit2`, кнопку `Button1`, для свойства `Caption` кнопки задайте значение «Создать массив». Раз-

местите на форме панель `GroupBox1`, для свойства `Caption` которой задайте значение «Определить».

Примечание. Панель `GroupBox` – это контейнер с рамкой и надписью, объединяющий группу связанных органов управления, таких, как переключатели `RadioButton`, флажки `CheckBox` и т.д. `GroupBox` имеет встроенную рамку с надписью, которая обычно используется для выделения на форме группы функционально объединенных компонентов.

В панели `GroupBox1` разместите компоненты `CheckBox1`, `CheckBox2`, `CheckBox3` и `CheckBox4`, для свойств `Caption` которых задайте значения: «Минимальный элемент», «Максимальный элемент», «Сумма всех элементов», «Число положительных элементов» соответственно. Напротив них расположите компоненты `Edit3`, `Edit4`, `Edit5` и `Edit6`. Если компоненты `CheckBox1`, `CheckBox2`, `CheckBox3`, `CheckBox4` окажутся размещенными под панелью `GroupBox1` и не будут отображаться, то следует выделить панель `GroupBox1` и выбрать в контекстном меню команду `Control ► Send to Back` (Порядок ► На задний план). В нижней части формы разместите кнопку `Button1` и задайте значение свойства `Button1.Caption` – «Вычислить». Удалите текст «Edit1», «Edit2», «Edit3», «Edit4», «Edit5», «Edit6» из соответствующих компонентов. Выровняйте компоненты на форме, как показано на рис. 3.3. Зафиксируйте положение компонентов на форме, выбрав в меню Delphi команду `Edit ► Lock Controls`.

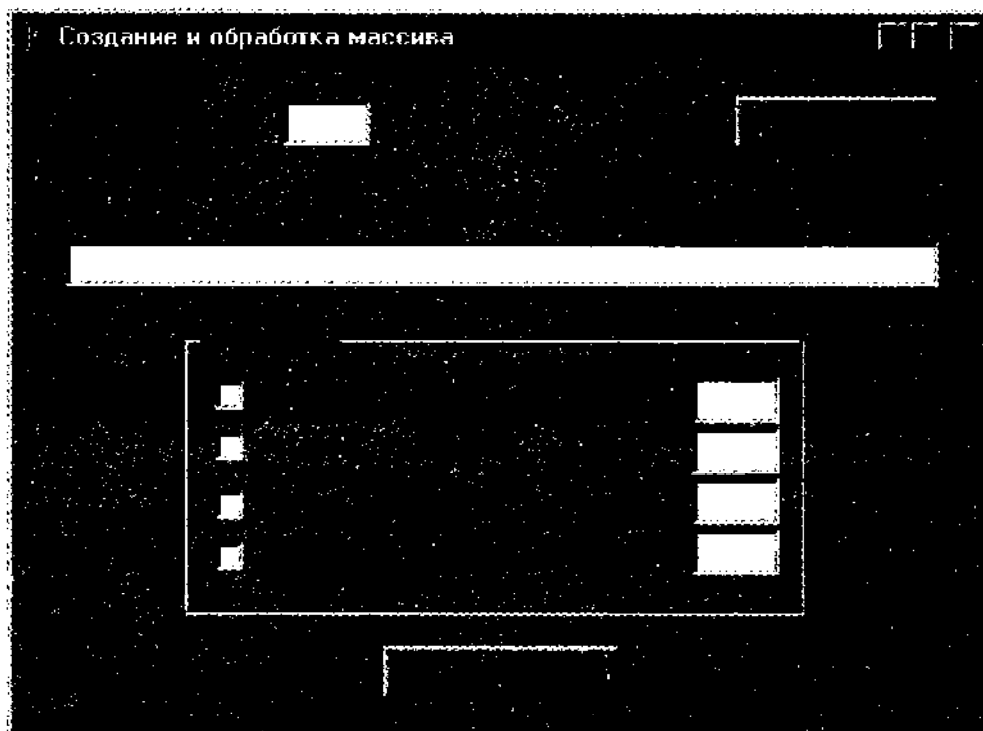


Рис. 3.3. Окно формы с компонентами

Сохраните файл проекта и программного модуля.

Прежде чем создавать обработчики событий щелчка мышью по кнопкам `Button1` и `Button2`, в разделе описания переменных опишите переменные целого типа `N` и `I`, где `N` – размер массива, а `I` – порядковый номер элемента массива, а также `M` – динамический массив целых чисел:

```
var
    Form1: TForm1;
    N, I : integer;
    M : array of integer; {Описание динамического массива
целых чисел}
```

Примечание. Динамические массивы введены в Object Pascal, начиная с версии Delphi 5. Они отличаются от обычных статических массивов тем, что для них заранее не объявляется длина – число элементов. Объявление динамического массива содержит только его имя и тип элементов (один из базовых типов). При объявлении динамического массива место под него не отводится. Прежде чем использовать массив, надо задать его размер процедурой `SetLength`.

Для предупреждения ввода в окно `Edit1` нечислового значения реализуем обработку события нажатия на клавишу в окне `Edit1`, чтобы запретить ввод любых символов, кроме цифр от 0 до 9. Для создания процедуры обработчика события нажатия на клавишу в окне `Edit1` выберите в окне Инспектора объектов компонент `Edit1` и на странице `Events` (События) дважды щелкните левой кнопкой мыши на пустом поле списка в событии `OnKeyPress`. После этого окно Редактор кода немедленно получит фокус и в разделе `interface` появится запись процедуры обработчика события:

```
procedure Edit1KeyPress(Sender: TObject; var Key: Char);
```

а в разделе `implementation` – текст заготовки этой процедуры:

```
procedure Edit1KeyPress(Sender: TObject; var Key: Char);
begin
    ...
end;
```

Вставьте в тело процедуры следующий оператор:

```
if not (Key in ['0'..'9']) then Key:=#0;
```

Действие этого оператора сводится к сравнению значения переменной `Key` с множеством значений `[0..9]`. Если символ нажатой клавиши не входит в это множество, то `Key` присваивается значение нулевого символа (`#0`). Таким образом, в окне `Edit1` будет отображаться текст, состоящий только из цифр.

Создание массива целых чисел опишите в процедуре обработчика события щелчка мышью по кнопке `Button1`. Для создания процедуры обработчика события выберите в окне Инспектора объектов объект `Button1`, затем на странице События сделайте двойной щелчок на пустом поле списка в событии `OnClick`. После этого отредактируйте заготовку процедуры обработчика события `procedure TForm1.Button1Click(Sender: TObject);` в окне Редактор кода следующим образом:

```

procedure TForm1.Button1Click(Sender: TObject);
begin
    Randomize;
    N:=StrToInt(Edit1.Text); {Число элементов массива}
    SetLength(M, N);         {Задать массиву M длину N}
    Edit2.Text:='';          {Очистить окно Edit2}
    for I := 0 to N-1 do {Заполнить массив случайными
значениями целых чисел}
        begin
            M[I] := Round(Sin(Random(100))*100);
            {Присвоить элементу массива случайное значение}
            Edit2.Text:=Edit2.Text+' '+IntToStr(M[I]);
            {Вывести элементы массива}
        end;
end;
end;

```

Примечание. Оператор Randomize; задает случайное начальное значение для функции Random.

Обработку массива опишите в процедуре обработчика события щелчка мышью по кнопке Button2. Для этого выберите в окне Инспектора объектов объект Button2, затем на странице События сделайте двойной щелчок на пустом поле списка в событии OnClick. После этого в окне Редактор кода в заготовку процедуры обработчика события procedure TForm1.Button2Click(Sender: TObject); в раздел описания локальных переменных поместите следующее описание:

```

var
    Max, Min, Sum, CountPlus : integer; {Результаты обработки
массива},

```

где Max – максимальный элемент массива, Min – минимальный элемент массива, Sum – сумма всех элементов массива, CountPlus – количество положительных элементов массива.

Обработку массива можно реализовать с помощью цикла For, в котором вычисление значения каждой из этих переменных записывается при помощи оператора if then, проверяющего условие CheckBox.Checked. Если свойство Checked имеет значение True, то вычисляется соответствующая переменная. В заключительной части процедуры можно разместить вывод результатов обработки массива. Текст процедуры обработки массива может быть записан следующим образом:

```

procedure TForm1.Button2Click(Sender: TObject); {обработка
массива}
var
    Max, Min, Sum, CountPlus : integer; {Результаты обработки
массива}

```

```
begin
  if CheckBox1.Checked then Min:=M[0]; {пусть 0-й элемент -
Min}
  Edit3.Text:='';
  if CheckBox2.Checked then Max:=M[0]; {Пусть 0-й элемент -
Max}
  Edit4.Text:='';
  Sum:=0;           {Обнулить значения Sum}
  Edit5.Text:='';
  CountPlus:=0;     {Обнулить значения суммы положительных
элементов}
  Edit6.Text:='';
  for I:=0 to N-1 do
    begin
      if CheckBox1.Checked then {Определить минимальный элемент
массива}
        if Min>M[I] then Min:=M[I];
      if CheckBox2.Checked then {Определить максимальный
элемент массива}
        if Max<M[I] then Max:=M[I];
      if CheckBox3.Checked then {Суммировать элементы
массива}
        Sum:=Sum+M[I];
      if CheckBox4.Checked then {Суммировать положительные
элементы массива}
        if M[I]>0 then CountPlus:=CountPlus+1;
    end;
  {Вывести результаты обработки массива}
  if CheckBox1.Checked then Edit3.Text:=IntToStr(Min);
  if CheckBox2.Checked then Edit4.Text:=IntToStr(Max);
  if CheckBox3.Checked then Edit5.Text:=IntToStr(Sum);
  if CheckBox4.Checked then Edit6.Text:=IntToStr(CountPlus);
end;
```

Сохраните файлы проекта и программного модуля, откомпилируйте и запустите программу на выполнение. Задавая различные количества элементов массива и щелкая мышью по кнопке Создать массив, убедитесь в правильной работе процедуры защиты от ввода нечисловых данных в Edit1 и генерации массива случайных целых чисел.

Выбирая варианты обработки массива установкой соответствующих флажков CheckBox (рис. 3.4) и щелкая мышью по кнопке Вычислить, убедитесь в правильности работы процедуры обработки массива.

Закройте окно приложения.

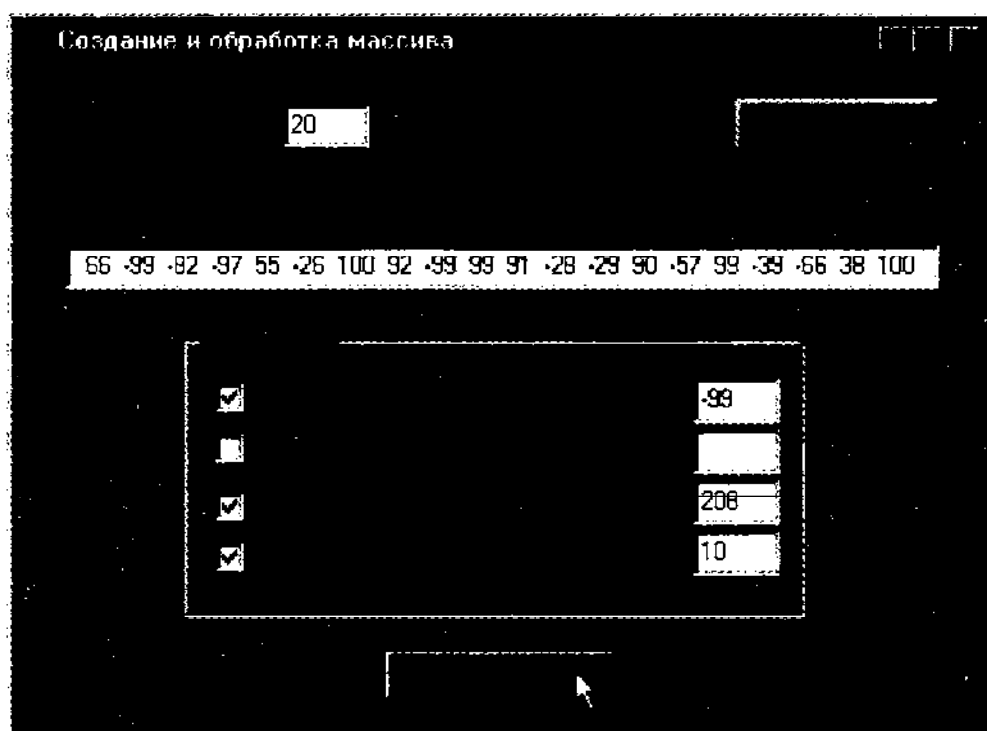


Рис. 3.4. Окно приложения создания и обработки линейного массива

Линейная сортировка массива

Средства визуального программирования ИСР Delphi позволяют создать приложение, обеспечивающее наглядное отражение результатов сортировки массивов.

Упражнение 3.4. Создайте приложение, которое предлагает пользователю ввести размер массива и создать массив случайных целых чисел, а затем, используя переключатели, указать порядок сортировки (по неубыванию, по невозрастанию), выполнить сортировку и просмотреть отсортированный массив.

Создайте форму, для свойства Caption задайте значение «Линейная сортировка массива». На форме разместите компоненты Edit1, Edit2 и Edit3, а также кнопку Button1, для свойства Caption которой задайте значение «Создать массив». Удалите текст «Edit1», «Edit2», «Edit3» из соответствующих компонентов. Разместите на форме компоненты Label1, Label2 и задайте для их свойств Caption значения «Число элементов» и «Исходный массив» соответственно.

Ниже объекта Edit3 на форме разместите панель RadioGroup1, для свойства Caption которой задайте значение «Порядок сортировки». Для выбора порядка сортировки задайте в панели RadioGroup1 два переключателя и подписи к ним. Выбрав в Инспекторе объектов компонент RadioGroup1, на странице свойств выберите свойство Items, затем в окне String List Editor введите список элементов: по невозрастанию, по неубыванию – и нажмите ОК. Справа от панели RadioGroup1 разместите кнопку Button2, для свойства Caption которой задайте значение «Отсортировать».

В нижней части формы разместите объект Edit3 для вывода отсортированного массива. Над Edit3 разместите объект Label3, для свойства Caption которого задайте значение «Отсортированный массив». Выровняйте компоненты на форме, как показано на рис. 3.5. Зафиксируйте положение компонентов на форме, выбрав в меню Delphi команду Edit ► Lock Controls.

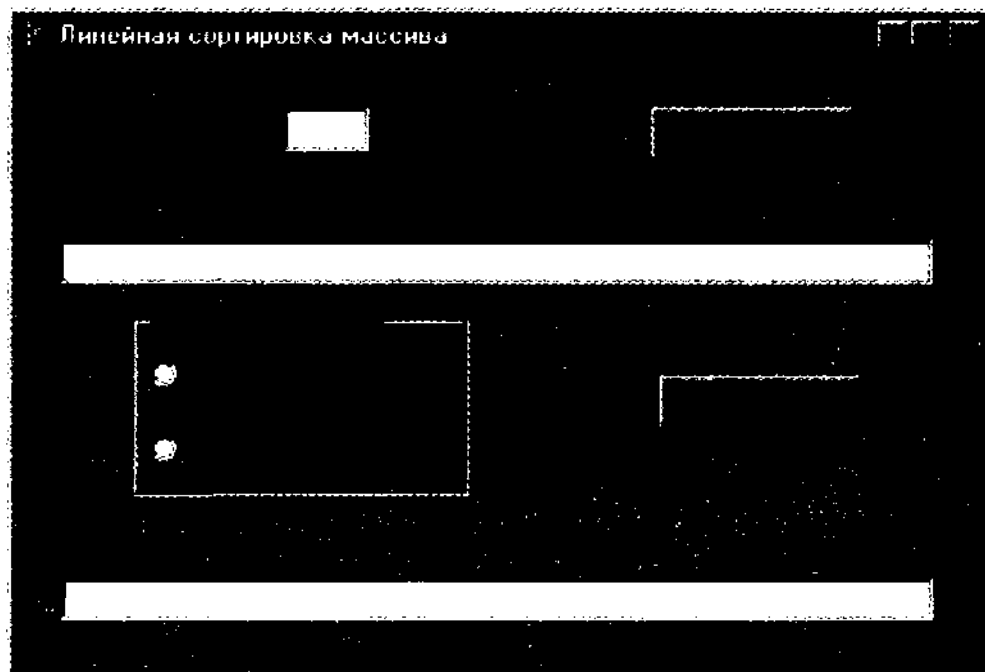


Рис. 3.5. Окно формы приложения линейной сортировки массива

Сохраните файл проекта и программного модуля. Как и в предыдущем примере, прежде чем создавать обработчики событий щелчка мышью по кнопкам Button1 и Button2, опишите глобальные переменные целого типа N и I, где N – размер массива, а I – порядковый номер элемента массива, а также M – динамический массив целых чисел:

```
var
  Form1: TForm1;
  N, I : integer;
  M : array of integer; {Описание динамического массива целых
чисел}
```

Для предупреждения ввода в окно Edit1 нечислового значения введите обработку события нажатия клавиши в окне Edit1, чтобы запретить ввод любых символов, кроме цифр от 0 до 9. Для создания процедуры обработчика события нажатия клавиши в окне Edit1 выберите в окне Инспектора объектов компонент Edit1 и на странице Events (События) дважды щелкните левой кнопкой мыши на пустом поле списка в событии OnKeyPress. После этого в текст процедуры обработчика события добавьте следующий оператор: `if not (Key in ['0'..'9']) then Key:=#0;`. Полный текст процедуры обработчика события будет выглядеть следующим образом:

```

procedure Edit1KeyPress(Sender: TObject; var Key: Char);
begin
    if not (Key in ['0'..'9']) then Key:=#0;
end;

```

Создание массива целых чисел опишите в процедуре обработчика события щелчка мышью по кнопке Button1. Для создания процедуры обработчика события выберите в окне Инспектора объектов объект Button1, затем на странице События сделайте двойной щелчок на пустом поле списка в событии OnClick. После этого в окне Редактор кода в заготовку процедуры обработчика события введите следующий текст:

```

procedure TForm1.Button1Click(Sender: TObject);
begin
    Randomize;
    N:=StrToInt(Edit1.Text); {Число элементов массива}
    SetLength(M, N); {Задать динамическому массиву M длину N}
    Edit2.Text:='';
    for I := 0 to N-1 do      {Заполнить массив случайными значениями
целых чисел}
        begin
            M[I]:= Round(Sin(Random(100))*100);
            {Присвоить элементу массива случайное значение}
            Edit2.Text:=Edit2.Text+' '+IntToStr(M[I]);
                                {Вывести элементы массива}
        end;
    end;

```

Обработка события нажатия кнопки Button2 «Вычислить» начинается с сортировки массива, которую можно записать с помощью оператора цикла **for**. Для хранения номера элемента массива, сравниваемого с текущим (имеющим номер I), введем переменную целого типа J. Для запоминания элемента массива при обмене в процессе сортировки введем переменную Tmp.

Линейная сортировка массива выполняется перестановкой элементов в массиве при соблюдении следующих условий: если в неотсортированной части массива найден элемент с номером J, больший, чем элемент с номером I (для сортировки по невозрастанию), или меньший, чем элемент с номером I (для сортировки по неубыванию). Поэтому можно записать следующий оператор **if then else** с составным условием:

```

if (RadioGroup1.ItemIndex=0) and (M[I]<M[J])
or (RadioGroup1.ItemIndex=1) and (M[I]>M[J]) then

```

Перестановка элементов массива осуществляется с использованием третьей переменной:

```
Tmp := M[I]; {Запомнить на время значение M[I]}
M[I] := M[J];
M[J] := Tmp;
```

Вывод отсортированного массива в окне Edit3 можно записать оператором

```
for I:=0 to N-1 do
  Edit3.Text:=Edit3.Text+' '+IntToStr(M[I]);
```

В целом текст процедуры сортировки и вывода отсортированного массива будет выглядеть следующим образом:

```
procedure TForm1.Button2Click(Sender: TObject);
var
  J,Tmp : integer;
begin
  Edit3.Text:='';
  for I := 0 to N - 2 do {Изменять размер неотсортированной
    части массива}
    for J:=I+1 to N-1 do {Сравниваем поочередно I-й элемент
      неотсортированной части массива со всеми от (I+1)-го до конца}
      begin {Выбор операции в зависимости от значения
        свойства RadioGroup1.ItemIndex}
        if (RadioGroup1.ItemIndex=0) and (M[I]<M[J])
          or (RadioGroup1.ItemIndex=1) and (M[I]>M[J]) then
          {Если в неотсортированной части массива нашли J-й элемент,
            больший, чем I-й (для сортировки по невозрастанию), или меньший,
            чем I-й (для сортировки по убыванию)}
            begin {Обменять местами элементы массива}
              Tmp := M[I]; {Запомнить на время значение M[I]}
              M[I] := M[J];
              M[J] := Tmp;
            end;
          end;
      for I:=0 to N-1 do {Вывести отсортированный массив}
        Edit3.Text:=Edit3.Text+' '+IntToStr(M[I]);
      end;
```

Сохраните файлы проекта и программного модуля, откомпилируйте и запустите программу на выполнение. Задавая различные количества элементов

массива и щелкая мышью по кнопке Создать массив, создавайте линейный массив целых чисел. Выбирая при помощи переключателей в панели RadioGroup1 вариант сортировки и щелкая мышью по кнопке Отсортировать (рис. 3.6), убедитесь в правильной работе процедуры сортировки и вывода отсортированного массива.

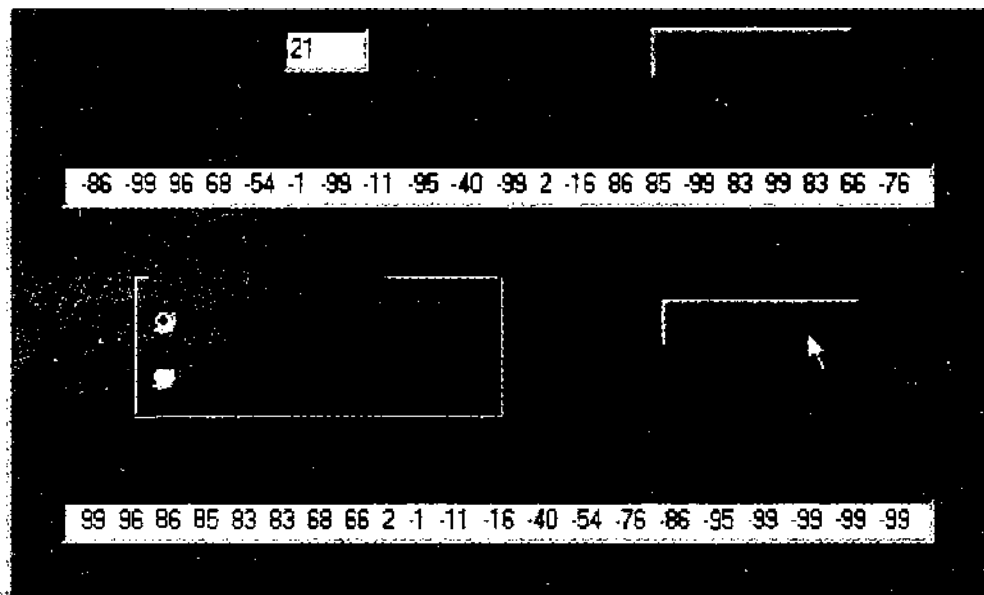


Рис. 3.6. Окно приложения линейной сортировки массива

После окончания проверки работы приложения закройте его окно.

Использование компонента **StringGrid** для представления двумерных массивов

Упражнение 3.5. Создайте приложение, которое выводит двумерный массив случайных целых чисел в объекте **StringGrid** и определяет минимальный и максимальный элементы, а также сумму элементов массива, расположенных на главной диагонали.

Создайте форму, свойству **Caption** которой присвойте значение «Обработка двумерного массива». Выберите в палитре компонентов страницы **Additional** компонент **StringGrid** и расположите его в левом верхнем углу формы. Задайте для свойств **ColCount** (количество столбцов) и **RowCount** (количество строк) значения 6. Задайте для свойств **FixedCols** и **FixedRows** (количество фиксированных, не прокручиваемых столбцов и строк, используемых для размещения номеров столбцов и строк).

Примечание. Компонент **StringGrid** представляет собой таблицу, содержащую строки. Таблица может иметь полосы прокрутки, причем заданное число первых строк и столбцов может быть фиксированным и не подвергаться прокрутке. Таким образом, можно задать заголовки столбцов и строк, постоянно присутствующие в окне компонента. Каждой ячейке таблицы может быть поставлен в соответствие некоторый объект. Обратите внимание, что при обращении к ячейке таблицы **StringGrid** первый индекс означает номер столбца, а второй – номер строки, тогда как в **Pascal** – наоборот.

Справа от объекта StringGrid разместите кнопку Button1 и задайте для ее свойства Caption значение «Заполнить». Ниже объекта StringGrid расположите панель GroupBox1 и присвойте ее свойству Caption значение «Определить». На этой панели разместите компоненты CheckBox1, CheckBox2, CheckBox3 и присвойте их свойствам Caption значения «Минимальный элемент», «Максимальный элемент», «Сумма элементов главной диагонали» соответственно. Справа от компонентов CheckBox1, CheckBox2, CheckBox3 разместите компоненты Edit1, Edit2, Edit3 и удалите текст «Edit1», «Edit2», «Edit3» из соответствующих компонентов. Правее панели GroupBox1 расположите кнопку Button2 и задайте для ее свойства Caption значение «Вычислить». Выворняйте компоненты на форме, как показано на рис. 3.7, и зафиксируйте их положение на форме.

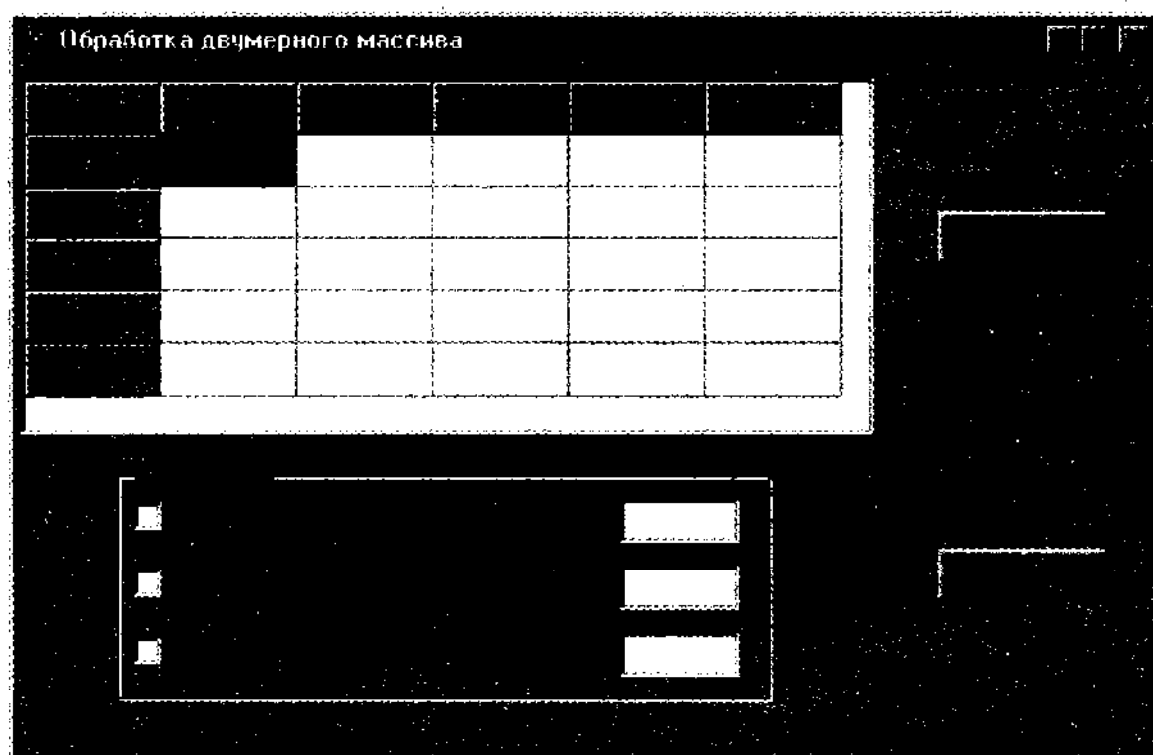


Рис. 3.7. Окно формы приложения обработки двумерного массива

Сохраните файл проекта и программного модуля. Как и в предыдущем примере, прежде чем создавать обработчики событий щелчка мышью по кнопкам Button1 и Button2, следует добавить в раздел описания переменных данного модуля целочисленные переменные I и J, предназначенные для хранения индексов массива (I – номер столбца, J – номер строки):

```
var  
    Form1: TForm1;  
    I, J : integer;
```

Для получения подсказки Delphi по объекту StringGrid1, указав объект, нажмите F1. В окне Delphi Help просмотрите общую информацию о назначении объекта. Щелкнув мышью по ссылке Properties (свойства), откройте окно со

списком свойств объекта и, выбирая нужные свойства, например Cells, ColCount и т. п., просмотрите справочную информацию. Для возврата к предыдущему экрану справки воспользуйтесь кнопкой Назад в панели инструментов окна Delphi Help. Щелкая мышью по ссылкам Methods (методы) и Events (События), просмотрите список методов и событий объекта. Для просмотра примеров следует щелкнуть мышью по ссылке Example. Завершив просмотр справочной информации, закройте окно Delphi Help.

Примечание. Обратите внимание на использование оператора with .. do для обращения к элементам объекта. При изучении языка Turbo Pascal вы использовали этот оператор для сокращения записи при обращении к полям записей. Теперь вы будете применять его для сокращения записи при обращении к свойствам и методам объекта. Это делается для того, чтобы избежать повторных ссылок на объект в последующих операторах. В операторе, следующем за ключевым словом do, можно не указывать ссылки на поля, свойства и методы объекта. При этом каждый идентификатор в операторе, совпадающий с именем поля, свойства, метода объекта, трактуется как относящийся к этому объекту, и к нему неявно добавляется ссылка на этот объект. Например, вместо того чтобы, обращаясь к ячейке объекта StringGrid1, писать StringGrid1.Cells[I,J], удобнее использовать оператор with StringGrid1 do, в теле которого можно неоднократно обращаться к Cells[I,J], не упоминая имени объекта StringGrid1.

Создайте процедуру обработчика события щелчка мышью по кнопке Button1, в которой сначала будет выполнена операция вывода номеров строк и столбцов, а затем ячейки StringGrid1 будут заполнены случайными целыми числами. Для этого выберите в окне Инспектора объектов объект Button1 и на странице События сделайте двойной щелчок на пустом поле списка в событии OnClick. После этого отредактируйте заготовку процедуры обработчика этого события следующим образом:

```
procedure TForm1.Button1Click(Sender: TObject);
                                {Заполнение массива}
begin
    Randomize;
    with StringGrid1 do {Вывести номера строк в 0-м столбце и
столбцов в 0-й строке}
    begin
        I:=0; {Столбец 0}
        for J:= 1 to RowCount - 1 do {Вывести номера строк}
            Cells[I,J] := IntToStr(J);
        J:=0; {Строка 0}
        for I:= 1 to ColCount - 1 do {Вывести номера столбцов}
            Cells[I,J] := IntToStr(I);
        end;
    with StringGrid1 do {Вывести в таблице элементы двумерного
массива}
        for I := 1 to ColCount - 1 do
```

```

    for J:= 1 to RowCount - 1 do
        begin
            Cells[I,J] := IntToStr(Round(Sin(Random(100))
*100));
        end;
    end;

```

Для создания процедуры обработки массива выберите в окне Инспектора объектов объект Button2 и на странице События сделайте двойной щелчок на пустом поле списка в событии OnClick. После этого отредактируйте заготовку процедуры обработчика этого события следующим образом:

```

procedure TForm1.Button2Click(Sender: TObject); {Обработка
массива}
var
    Min, Max, Sum : Integer; {Локальные переменные - результаты
обработки массива}
begin
    if Checkbox1.Checked then {Определение Min-элемента}
        with StringGrid1 do
            begin
                Min:=StrToInt(Cells[1,1]); {Пусть это Min-элемент}
                for I := 1 to ColCount - 1 do
                    for J:= 1 to RowCount - 1 do
                        if StrToInt(Cells[I,J])<Min then
                            Min:=StrToInt(Cells[I,J]);
                Edit1.Text:=IntToStr(Min);
            end
        else Edit1.Text:='';
    if Checkbox2.Checked then {Определение Max-элемента}
        with StringGrid1 do
            begin
                Max:=StrToInt(Cells[1,1]); {Пусть это Max-элемент}
                for I := 1 to ColCount - 1 do
                    for J:= 1 to RowCount - 1 do
                        if StrToInt(Cells[I,J])>Max then
                            Max:=StrToInt(Cells[I,J]);
                Edit2.Text:=IntToStr(Max);
            end
        else Edit2.Text:='';
    if Checkbox3.Checked then {Вычисление Sum}
        with StringGrid1 do

```

```

begin
    Sum:=0; {Обнулить сумму перед подсчетом}
    for I := 1 to ColCount - 1 do
        Sum:=Sum+StrToInt(Cells[I,I]);
        Edit3.Text:=IntToStr(Sum);
    end else Edit3.Text:='';
end;

```

Как видно из текста процедуры, в ней имеется три фрагмента, каждый из которых выполняет вычисления, если свойство `Checked` соответствующего флажка (`Checkbox1`, `Checkbox2` или `Checkbox3`) имеет значение `True`. Операторы типа `Edit1.Text:=""`, `Edit2.Text:=""`, `Edit3.Text:=""` обеспечивают очистку соответствующего окна `Edit`, если вычисление не проводилось. Для обращения к элементу массива, расположенному на главной диагонали `StringGrid`, указываются одинаковые номера строки и столбца, `Cells[I,I]`.

Сохраните файлы проекта и программного модуля, откомпилируйте и запустите программу на выполнение. Щелкнув мышью по кнопке `Заполнить`, проверьте заполнение надписей номеров строк и столбцов, а также заполнение объекта `StringGrid` значениями элементов двумерного массива. Как показано на рис. 3.8, устанавливая флажки `Checkbox1..3` и щелкая мышью по кнопке `Вычислить`, убедитесь, что приложение правильно обрабатывает двумерный массив: определяет минимальное и максимальное значения, вычисляет сумму элементов на главной диагонали.

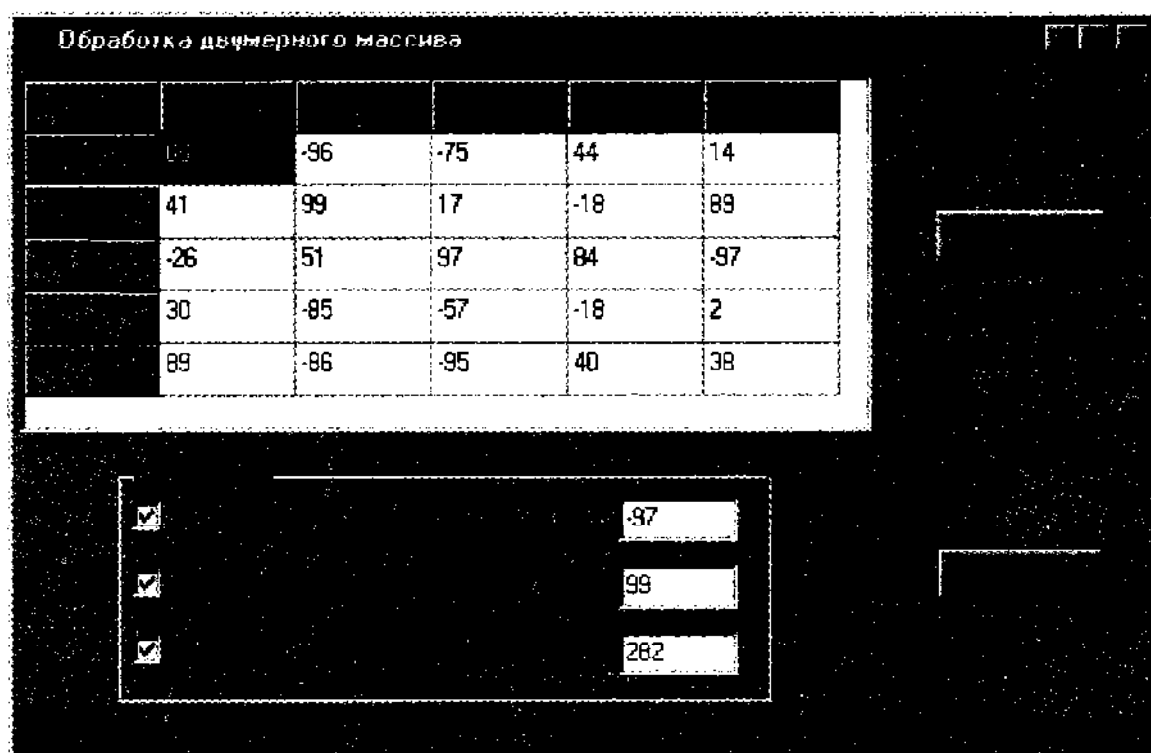


Рис. 3.8. Окно приложения обработки двумерного массива

После проверки работы приложения закройте его окно.

Упражнение 3.6. Создайте приложение, выводящее в окно двухмерный массив – таблицу успеваемости по нескольким школьным предметам с оценками за 1-ю – 4-ю четверти и за год, причем годовая оценка должна вычисляться как среднее арифметическое всех четвертных оценок по данному предмету. Используйте закрашку ячеек таблицы разными цветами, которые могут определяться как на стадии разработки формы приложения, так и во время работы приложения в зависимости от значения элемента массива.

Создайте форму, свойству Caption которой присвойте значение «Табель успеваемости». Выберите в палитре компонентов страницы Additional компонент StringGrid и разместите его в левом верхнем углу формы. В окне Инспектора объектов выберите компонент StringGrid1 и задайте для его свойства Align (выравнивание) значение alClient (вся клиентская область формы). При этом компонент StringGrid1 должен занимать всю клиентскую область формы и во время выполнения приложения его размеры должны изменяться при изменении размеров формы. Задайте для свойств ColCount (количество столбцов) и RowCount (количество строк) значения 6. Задайте для свойства FixedRows (количество фиксированных, не прокручиваемых строк, используемых для размещения номеров столбцов и строк) значение 1. Сохраните файл проекта и программного модуля.

Создание табеля с оценками опишите в процедуре обработчика события создания формы Form1. Для этого выберите в окне Инспектора объектов объект Form1 и на странице События сделайте двойной щелчок мышью на пустом поле списка в событии OnCreate, наступающем при создании формы. После этого в окне Редактор кода будет сгенерирована заготовка процедуры обработчика события procedure TForm1.FormCreate(Sender: TObject);.

Вставьте в текст модуля main перед разделом implementation описание раздела констант и переменных, задающих постоянные параметры массива и столбцов StringGrid1:

```
const
  COLWIDTHS : array[0..5] of integer=(85,70,70,70,70,60);
                                     {Ширина столбцов таблицы}
  COLTITLES : array[0..5] of string=('Предмет','1-я
четверть','2-я четверть','3-я четверть','4-я
четверть','годовая');
                                     {Шапка таблицы}
  COLALIGNS : array[0..5] of byte=(1,2,2,2,2,2); {Вариант
выравнивания}
var
  Form1: TForm1;                    {Объект Form1}
  ColAll: integer;                  {Ширина курсора}
```

Отредактируйте заготовку процедуры обработчика этого события следующим образом:

```

procedure TForm1.FormCreate(Sender: TObject);
var I,J:integer;
    K : real;
begin
    for i:=0 to 5 do
        begin
            Inc(ColAll,COLWIDTHS[i]+1); {Подсчет общей ширины для
курсора}
            StringGrid1.ColWidths[i] := COLWIDTHS[i]; {Установка
ширины колонок}
            StringGrid1.Cells[ i,0 ] := COLTITLES[i]; {Установка
шапки таблицы}
        end;
    with StringGrid1 do begin {Массив с данными успеваемости}
        Cells[ 0, 1]:='Русский язык';
        Cells[ 0, 2]:='Математика';
        Cells[ 0, 3]:='История';
        Cells[ 0, 4]:='География';
        Cells[ 0, 5]:='Музыка';
        {Оценки за 1-ю четверть - 2-й столбец}
        Cells[ 1, 1]:='5';
        Cells[ 1, 2]:='4';
        Cells[ 1, 3]:='4';
        Cells[ 1, 4]:='4';
        Cells[ 1, 5]:='5';
        {Оценки за 2-ю четверть - 3-й столбец}
        Cells[ 2, 1]:='4';
        Cells[ 2, 2]:='2';
        Cells[ 2, 3]:='4';
        Cells[ 2, 4]:='5';
        Cells[ 2, 5]:='3';
        {Оценки за 3-ю четверть - 3-й столбец}
        Cells[ 3, 1]:='5';
        Cells[ 3, 2]:='2';
        Cells[ 3, 3]:='3';
        Cells[ 3, 4]:='4';
        Cells[ 3, 5]:='5';
        {Оценки за 4-ю четверть - 4-й столбец}
    end;
end;

```

```

Cells[ 4, 1]:='4';
Cells[ 4, 2]:='3';
Cells[ 4, 3]:='4';
Cells[ 4, 4]:='5';
Cells[ 4, 5]:='5';

        {5-й столбец: вычисление оценок за год
        как среднего арифметического оценок
        за четыре учебные четверти}
for I:=1 to 5 do
begin
    K:=0;
    for J:=1 to 4 do
        K:=K+StrToInt(Cells[J,I]);
    K:=K/4;        {Вычисление среднего балла за год}
    Cells[J,I]:=FloatToStr(K);
end;
end;
end;

```

Как видно из текста процедуры, сначала задаются параметры «шапки» таблицы, затем задаются элементы массива: названия предметов и оценки за 1-ю – 4-ю четверти. В заключительном фрагменте процедуры вычисляется оценка за год как среднее арифметическое оценок по предмету за все четверти.

Для раскраски ячеек StringGrid1 создайте процедуру обработчика события OnDrawCell, для чего выберите в окне Инспектора объектов объект StringGrid и на странице Свойства сделайте двойной щелчок мышью на пустом поле списка в событии OnDrawCell. После этого отредактируйте заготовку процедуры обработчика этого события следующим образом:

```

procedure TForm1.StringGrid1DrawCell(Sender: TObject; Col,
Row: Integer; Rect: TRect; State: TGridDrawState);
var
    CurrCell:String;                {Текущая ячейка}
begin
    with TStringGrid(sender).Canvas,Rect do
    begin
        CurrCell:=TStringGrid(sender).Cells[Col,Row];
        Bottom:=Bottom+1;
        Font.Color:=clBlack;        {Цвет шрифта}
    end;
end;

```



```

        Brush.Color:=clWhite;      {Цвет закрашки}
    if Row<1 then begin           {Если нулевая строка, то рисуем
шапку таблицы}
        Brush.Color:=clBtnFace;  {Цвет закрашки - текущий цвет
кнопок}
        Bottom:=Bottom-1;
        DrawEdge(Handle, Rect, BDR_RAISEDINNER, BF_TOPRIGHT or
            BF_MIDDLE or BF_BOTTOMLEFT);
        InflateRect(Rect,-2,-2);
        TextRect(Rect,Left-2+((COLWIDTHS[Col]-
            TextWidth(CurrCell))shr 1), Top-2,CurrCell);
        exit;
    end;
    case Col of {Закраска ячеек в разных столбцах}
        0: brush.Color:=clBlue; {Нулевой столбец синий}
        1..4: brush.Color:=clAqua; {Первый - четвертый столбцы
- голубые}
        5: {Цвет закрашки пятого столбца и цвет шрифта
символов зависит от значения ячейки}
            if StrToFloat(CurrCell)<3 then
                begin
                    brush.Color:=clRed; {Если оценка <3, то цвет
закраски красный}
                    Font.Color:=clAqua; {Цвет шрифта символов
голубой}
                end
            else
                if StrToFloat(CurrCell)<4 then
                    begin
                        brush.Color:=clYellow; {Если оценка <4,то
цвет закрашки желтый}
                        Font.Color:=clOlive; {Цвет шрифта символов
оливковый}
                    end
                else

```

```

begin
    brush.Color:=clGreen;      {Иначе - цвет
закраски зеленый}
    Font.Color:=clYellow;      {Цвет шрифта
символов желтый}
end;

end;
{Выравнивание значений в столбцах(COLALIGNS)}
case COLALIGNS[Col] of
    1: {влево} TextRect(Rect,Left,Top,CurrCell);
    2: {по центру} TextRect(rect,Left+((COLWIDTHS[Col]-
        TextWidth(CurrCell)) shr 1),Top,CurrCell);
end;

{Рисование курсора выделения строки}
if (Row=TStringGrid(sender).Row) then
begin
    Left:=0;                  {Левая граница}
    Bottom:=Bottom-1;         {Верхняя граница}
    Right:=ColAll;            {Правая граница}
    Brush.Color:=clBlack;      {Цвет}
    TStringGrid(sender).Canvas.FrameRect(Rect); {Курсор в
таблице}
end;
end;
end;

```

Изучите текст процедуры и комментарии к ее фрагментам. Сохраните файлы модуля и проекта, откомпилируйте и запустите приложение. Окно приложения будет выглядеть, как на рис. 3.9.

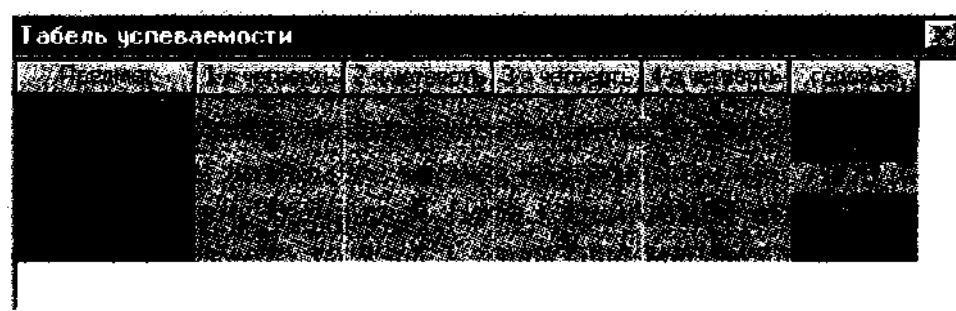


Рис. 3.9. Окно приложения «Табель успеваемости»

Проверьте работу приложения. Закройте окно приложения и задайте другие значения элементов массива, например, изменив оценки за 4-ю четверть, как показано далее:

```
Cells[ 4, 1] := '5';  
Cells[ 4, 2] := '4';  
Cells[ 4, 3] := '3';  
Cells[ 4, 4] := '4';  
Cells[ 4, 5] := '3';
```

Сохраните изменения в проекте, перекомпилируйте и запустите приложение. Обратите внимание на то, что с изменением значения среднего балла за год по предметам поменялся цвет закрашки и шрифта символов текста.

Закройте окно приложения.

Ввод и обработка элементов массива с использованием StringGrid

Для отображения таблиц текстовой информации используется визуальный компонент StringGrid, который представляет собой таблицу, содержащую столбцы и строки.

Упражнение 3.7. Создайте приложение с компонентом StringGrid, обеспечивающее пользователю возможность ввода значений элементов двумерного массива в ячейки и вычисляющее сумму элементов массива.

Создайте форму, для свойства Caption задайте значение «Ввод и суммирование элементов массива». Выберите в палитре компонентов страницы Additional компонент StringGrid и расположите его в центре формы. В окне Инспектор объектов выберите компонент StringGrid1 и присвойте его свойству Align (выравнивание) значение alNone (не изменять). При этом компонент StringGrid1 остается там, куда он был помещен во время проектирования, и размеры его не изменяются. Задайте для свойств ColCount и RowCount значения 5. Задайте для свойств FixedCols и FixedRows (количество фиксированных, не прокручиваемых столбцов и строк, используемых для размещения номеров столбцов и строк) значение 1. На странице свойств Инспектора объектов разверните список свойства StringGrid1.Options и задайте для его подсвойств goFixedVertLine, goFixedHorzLine, goVertLine, goHorzLine, goRangeSelect, goDrawFocusSelected и goEditing значение True.

В левом нижнем углу формы разместите кнопку Button1 и присвойте ее свойству Caption значение «Вычислить». В правом нижнем углу формы расположите объект Edit1 для отображения суммы элементов массива. Над объектом StringGrid1 разместите компонент Label1 и задайте для его свойства Caption значение «Введите элементы массива». Слева от объекта Edit1 расположите компонент Label2 и задайте для его свойства Caption значение «Сумма элементов». Удалите текст «Edit1» из окна Edit1. Выводите компоненты, как показано на рис. 3.10, и зафиксируйте их положение на форме.

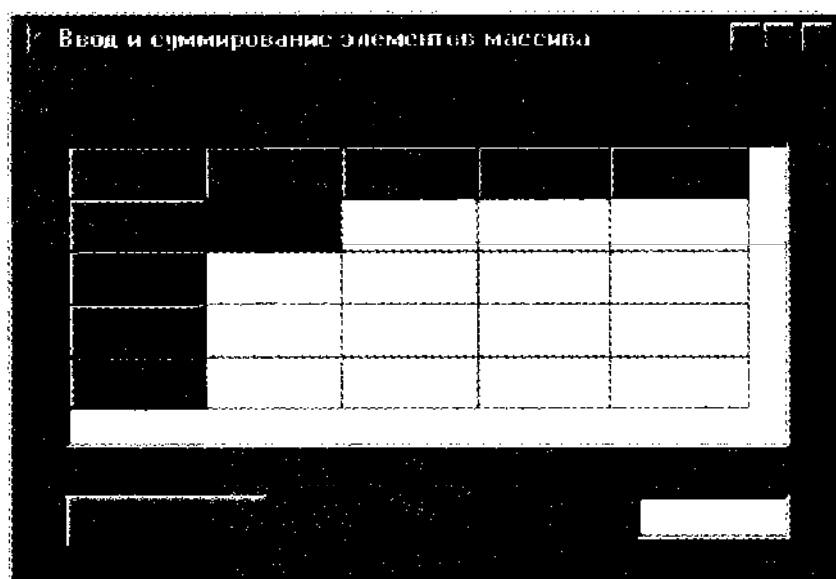


Рис. 3.10. Окно формы приложения ввода и суммирования элементов массива

Сохраните файл проекта и программного модуля.

Чтобы обеспечить пользователю возможность ввода значений элементов двумерного массива в ячейки `StringGrid1`, создайте процедуру обработчика события `OnEnter` для объекта `StringGrid1`. Для этого выберите в окне Инспектор объектов объект `StringGrid1` и на странице События сделайте двойной щелчок на пустом поле списка в событии `OnEnter`, наступающем в момент получения элементом фокуса. После этого в окне Редактор кода будет сгенерирована заготовка процедуры обработчика события `procedure TForm1.FormCreate(Sender: TObject);`.

Так как эта процедура сводится к выводу в окне приложения таблицы `StringGrid1` подписи номеров строк и столбцов, то текст этой процедуры можно записать следующим образом:

```

procedure TForm1.FormCreate(Sender: TObject);
var
    I,J: Integer;           {Номера столбцов и строк}
begin
    with StringGrid1 do      {Вывести номера строк в 0-м столбце
и столбцов в 0-й строке}
        begin
            I:=0;             {Столбец 0}
            for J:= 1 to RowCount - 1 do
                Cells[I,J] := IntToStr(J);
            J:=0;              {Строка 0}
            for I:= 1 to ColCount - 1 do
                Cells[I,J] := IntToStr(I);
            end;
        end;
end;

```

Для предупреждения ввода в окно ячейки StringGrid1 нечислового значения введите обработку нажатия клавиши, чтобы запретить ввод любых символов, кроме цифр от 0 до 9 и знаков «-» и «+». Для создания процедуры обработчика события нажатия клавиши выберите в окне Инспектор объектов компонент StringGrid1 и на странице Events (События) дважды щелкните левой кнопкой мыши на пустом поле списка в событии OnKeyPress. После этого окно Редактор кода немедленно получит фокус, и в разделе interface появится запись процедуры обработчика события: `procedure TForm1.StringGrid1KeyPress(Sender: TObject; var Key: Char);`.

Вставьте в текст заготовки этой процедуры оператор

```
if not (Key in ['0'..'9', '-', '+']) then Key:=#0;
```

Сохраните, откомпилируйте и запустите приложение на выполнение. Как вы увидите, приложение обеспечивает ввод числовых значений элементов массива в ячейки StringGrid1, но не вычисляет их сумму при щелчке мышью по кнопке Button1, так как процедура обработки этого действия еще не создана.

Для подсчета элементов введенного массива следует записать в виде процедуры обработчик события щелчка мышью по кнопке Button1. Для генерации кода процедуры обработчика события выберите в окне Инспектор объектов объект Button1 и на странице События сделайте двойной щелчок на пустом поле списка в событии OnClick, наступающем в момент получения элементом фокуса. После этого в окне Редактор кода будет сгенерирована заготовка процедуры обработчика события `procedure TForm1.Button1Click(Sender: TObject);`, в которую нужно добавить следующие операторы:

```
procedure TForm1.Button1Click(Sender: TObject);
    {Подсчет суммы элементов}
```

```
var
```

```
    I,J,Sum : integer;
```

```
begin
```

```
    with StringGrid1 do
```

```
    begin
```

```
        Sum:=0; {Обнулить сумму перед суммированием элементов}
```

```
        Edit1.Text:='';
```

```
        for J:=1 to RowCount - 1 do
```

```
            for I := 1 to ColCount - 1 do
```

```
                if Cells[J,I]<>' ' then
```

```
            {Если ячейка заполнена, то суммировать ее значение}
```

```
                Sum:=Sum+StrToInt(Cells[J,I]);
```

```
            Edit1.Text:=IntToStr(Sum);
```

```
            {Вывод результата
```

```
суммирования}
```

```
        end;
```

```
    end;
```

Сохраните, откомпилируйте и запустите на выполнение созданное приложение. Теперь оно обеспечивает не только ввод числовых значений элементов массива в ячейки StringGrid1, но и вычисляет их сумму при щелчке на кнопке Button1 (рис. 3.11).

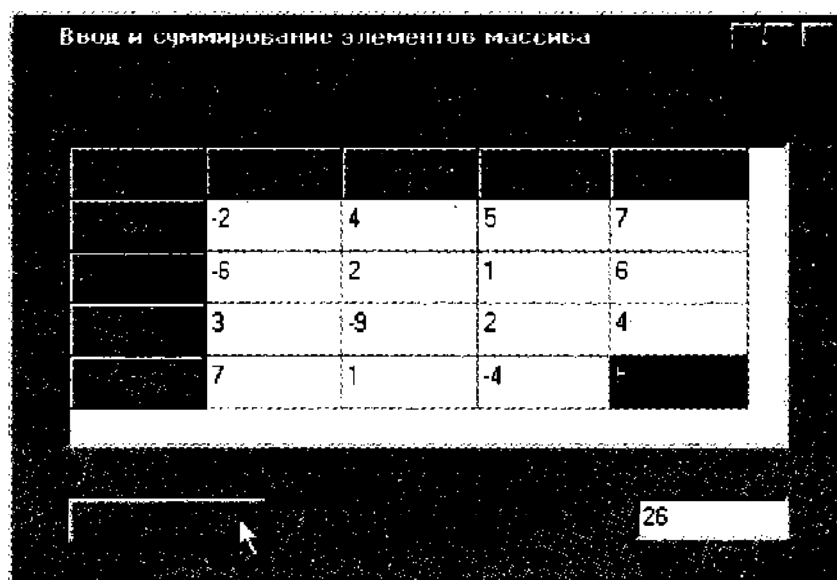


Рис. 3.11. Окно приложения ввода и суммирования элементов массива

Закройте окно приложения.

Обработка файлов

Упражнение 3.8. Создайте приложение, которое создает файл text1.txt и записывает в него текст, введенный пользователем в окно Edit, после чего закрывает файл.

Создайте форму и задайте для ее свойства Caption значение «Создание файла и вывод в него текста из Edit». Разместите на форме компоненты Edit1, Label1 и Button1 (рис. 13.12). Задайте значения для свойств Label1.Caption – «Введите текст», Button1.Caption – «Сохранить». Удалите текст «Edit1» из соответствующего компонента. Выровняйте компоненты и зафиксируйте их положение на форме.

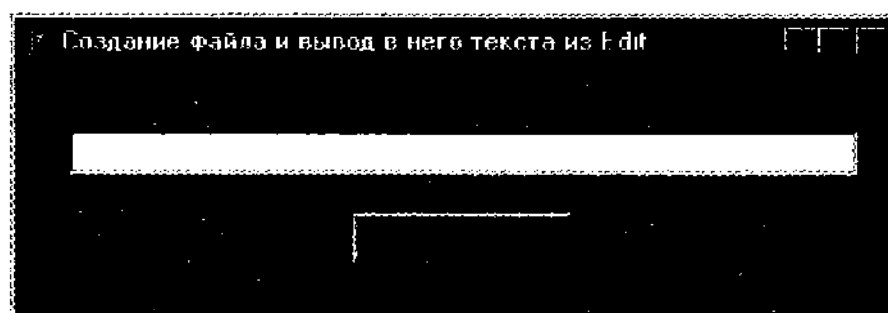


Рис. 3.12. Окно формы

Сохраните файлы модуля (под именем `main`) и проекта (под именем `TextEditFile`) в папке Обработка текстовых файлов. Откомпилируйте приложение и запустите его на выполнение. Убедитесь в том, что приложение позволяет пользователю вводить текст в окно `Edit1`. Для того чтобы на диске создавался текстовый файл и текст из окна `Edit1` записывался в него, создайте процедуру обработчика события щелчка мышью по кнопке `Button1` «Сохранить» и в ней реализуйте создание нового файла и запись в него текста. Для создания процедуры обработчика щелчка мышью по кнопке `Button1` выберите в окне Инспектор объектов объект `Button1`, затем на странице событий дважды щелкните мышью на пустом поле списка в событии `OnClick`. После этого в окне Редактор кода будет сгенерирована заготовка процедуры обработчика события `procedure TForm1.Button1Click(Sender: TObject);`.

Дополните текст процедуры следующим кодом:

```
var
    f:TextFile;                                {Описание файловой
переменной}
begin
    AssignFile(f, 'text1.txt');    {Связь файловой переменной
с файлом}
    Rewrite(f);                    {Создать новый файл}
    Writeln(f, Edit1.Text);        {Записать в файл}
    CloseFile(f); {Закрыть файл}
end;
```

Как видно из текста процедуры, знакомого вам по аналогичной теме из книги об использовании среды программирования Turbo Pascal, в начале процедуры файловая переменная `f` связывается с именем файла `text1.txt`, а затем на диске создается новый файл с этим именем. После этого записывается значение свойства `Text` объекта `Edit1`, и файл закрывается (рис. 3.13). Сохраните файл модуля под именем `Main`, файл проекта – под именем `TextEditFile1`, откомпилируйте программу и запустите ее на выполнение.



Рис. 3.13. Окно приложения, сохраняющего текст из `Edit1` в файле

В окне приложения введите текст (см. рис. 3.13) и щелкните мышью по кнопке Сохранить. Закройте окно приложения и откройте в окне Проводник Windows папку Обработка текстовых файлов, в которой сохранены файлы проекта. В списке файлов этой папки вы увидите и вновь созданный файл text1.txt (рис. 3.14).

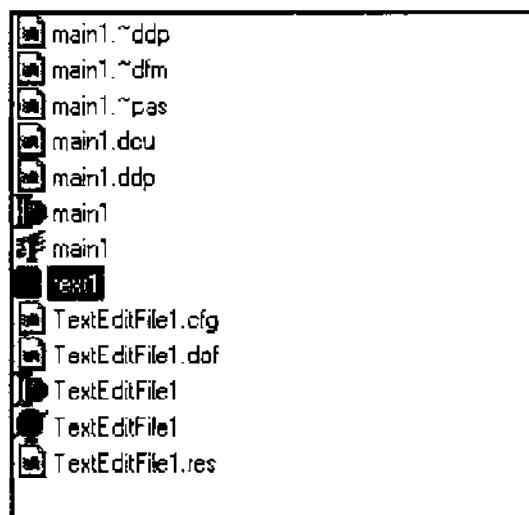


Рис. 3.14. Файл text1.txt в папке проекта

Дважды щелкните левой кнопкой мыши на имени файла text1.txt. Будет открыто окно редактора Блокнот с этим файлом. Убедитесь, что это – тот самый текст, который вы только что вводили в окне приложения. Закройте окна редактора Блокнот и Проводник Windows.

Упражнение 3.9. Создайте приложение, открывающее текстовый файл для чтения и считывающее из него текст в окно Мемо. Перед открытием файла следует проверить его наличие; в случае отсутствия файла должно выводиться соответствующее сообщение.

Примечание. Многострочное окно редактирования Мемо используется для ввода, отображения и редактирования многострочных текстов. Свойство Lines, доступное как во время проектирования, так и во время выполнения, имеет большое количество свойств и методов типа Strings, которые обычно используются для формирования и редактирования текста. Весь текст содержится в свойстве Text.

Создайте форму и задайте для ее свойства Caption значение «Чтение текста из файла в окно Мемо». На форме Form1 разместите компоненты Memo1, Label1 и Button1 (см. рис. 3.12), задайте значения для свойств Label1.Caption – «Текст из файла», Button1.Caption – «Прочитать текст из файла». Для удаления текста «Memo1» из окна компонента Memo1 выберите в окне Инспектор объектов объект Memo1, затем на странице Свойства дважды щелкните мышью на поле значения Strings свойства Lines для формирования и редактирования текста. После этого в окне String List Editor удалите текст «Memo1» (рис. 3.15) и щелкните мышью по кнопке ОК.

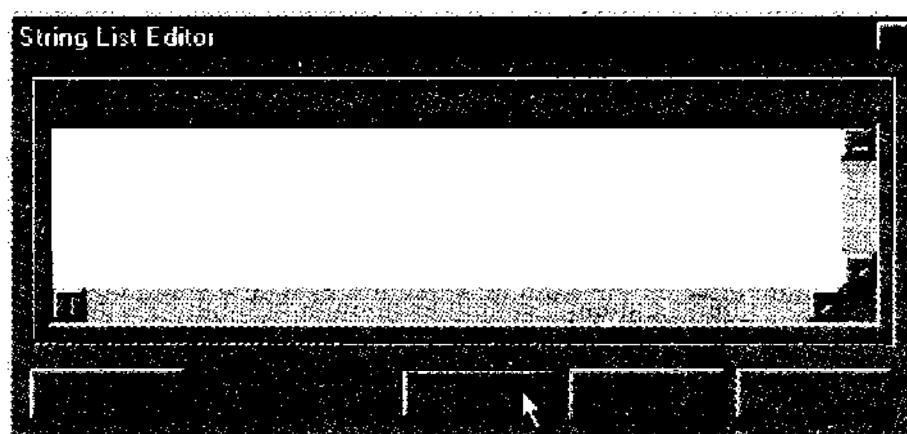


Рис. 3.15. Редактирование текста в окне Memo1

Для обеспечения возможности просмотра в окне Memo1 длинных текстов с использованием вертикальной полосы прокрутки в окне Инспектор объектов выберите объект Memo1 и на странице Свойства установите для свойства ScrollBars значение ssVertical. Выровняйте компоненты и зафиксируйте их положение на форме.

Выбрав в окне Инспектор объектов объект Button1, на странице событий сделайте двойной щелчок на пустом поле списка в событии OnClick. После этого в окне Редактор кода будет сгенерирована заготовка процедуры обработчика события. Чтобы эта процедура выполняла открытие текстового файла и выводила текст в окно Memo1, отредактируйте текст процедуры следующим образом:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  F:TextFile; {Описание файловой переменной}
  Ch : Char;   {Описание переменной, в которую будет считываться
символ из файла}
begin
  AssignFile(F,'text1.txt'); {Связь файловой переменной с
файлом на диске}
  {$I-} {Директива компилятора: отключить проверку ошибок
ввода-вывода}
  Reset(F); {Открыть файл для чтения}
  {$I+}
  if IOResult = 0 then begin {Если операция «Открыть файл»
выполнена успешно}
    while not Eof(f) do      {Пока не конец файла}
    begin
      Read(F,Ch); {Прочитать из файла символ}
      Memo1.Text:=Memo1.Text+Ch; {Вывести символ в поле
Memo1}
    end;
  end;
```

```
CloseFile(F); {Закреть файл}
end
else          {IOResult<>0 - операция «Открыть файл» не
выполнена}
  ShowMessage('Нет такого файла');
end;
```

Как видно из текста процедуры, после связывания файловой переменной с именем файла при помощи специальной директивы компилятора отключается контроль ошибок ввода-вывода для того, чтобы проверить функцией `IOResult` успешность операции. В случае успешности операции открытия файла на чтение производится чтение файла по одному символу и вывод его в окно `Memo1`. Условием прекращения чтения из файла является достижение конца файла (`Eof(f)=True`). После этого файл закрывается. Если открыть файл не удастся, то `IOResult` возвращает значение, отличное от нуля, при этом в отдельном окне выводится соответствующее сообщение.

Примечание. Директивы компилятора `SI` позволяют включить или отключить автоматический контроль результата вызова процедур ввода-вывода `Object Pascal`. Если используется директива `{SI+}`, то при возвращении процедурой ввода-вывода ненулевого значения генерируется исключение `EInOutError` и в его свойство `errorcode` заносится код ошибки. Таким образом, при использовании директивы `{SI+}` операции ввода-вывода располагаются в блоке `try...except`, имеющем обработчик исключения `EInOutError`. Если такого блока нет, то обработка производится методом `TApplication.HandleException`. Если используется директива `{SI-}`, то исключение не генерируется. В этом случае можно проверить наличие ошибки, обратившись к функции `IOResult`. Эта функция возвращает код ошибки, который можно проанализировать.

Сохраните файл модуля под именем `main1`, а файл проекта – под именем `TextMemoFile1` в папке Обработка текстовых файлов. Откомпилируйте и запустите приложение. Щелкнув на кнопке Прочитать текст из файла, убедитесь, что текст из файла считывается в окно `Memo1` (рис. 3.16).

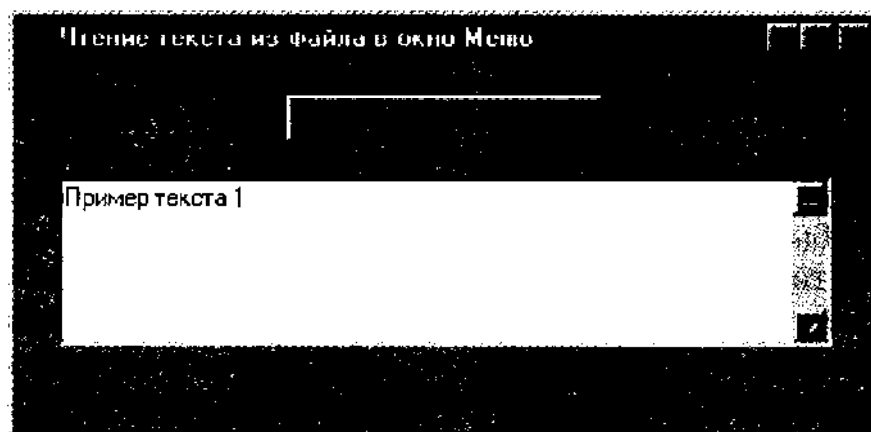


Рис. 3.16. Вид окна приложения чтения текста из файла в окно `Memo`

Закройте окно приложения.

Упражнение 3.10. Создайте приложение, открывающее текстовый файл для дополнения и затем добавляющее в него введенный текст.

Создайте форму и задайте для ее свойства Caption значение «Добавление текста в файл». На форме Form1 разместите компоненты Button1, Memo1, Label1 и, как показано на рис. 3.17, присвойте значения свойствам Label1. Caption – «Текст из файла», Button1.Caption – «Прочитать текст из файла». Под Label1 разместите Label2 и задайте для свойства Label2.Caption значение «Добавляемый текст». Ниже Label2 разместите на форме объект Edit1. Под объектом Edit1 расположите кнопку Button2 и задайте для свойства Button1. Caption значение «Добавить текст в файл». Удалите текст из окон компонентов Memo1, Edit1. Для обеспечения возможности просмотра в окне Memo1 длинных текстов с использованием вертикальной полосы прокрутки в окне Инспектора объектов выберите объект Memo1 и на странице Свойства установите для свойства ScrollBars значение ssVertical. Выровняйте компоненты и зафиксируйте их положение на форме (см. рис. 3.17).

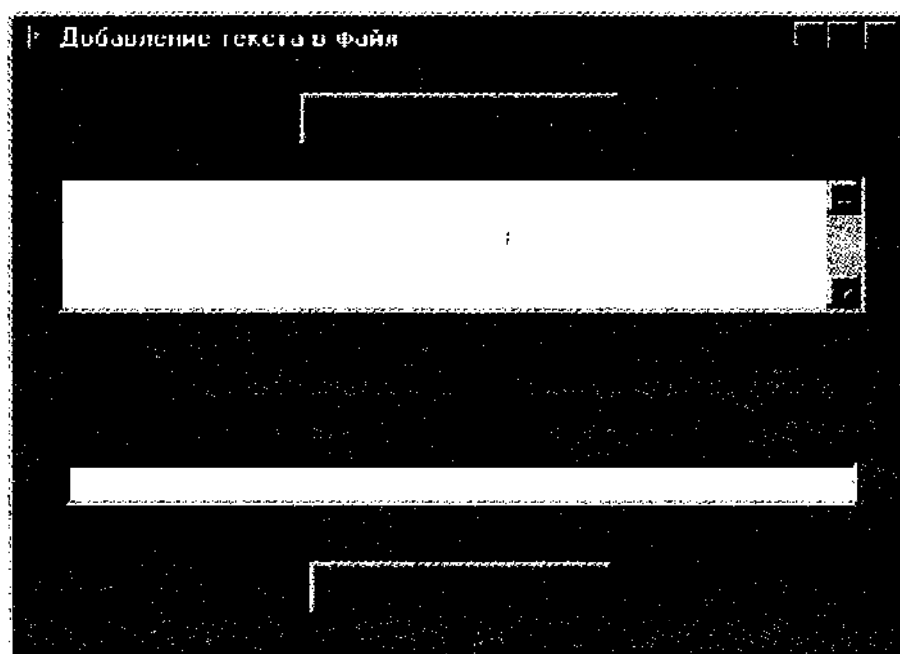


Рис. 3.17. Окно формы

Создайте обработчики нажатий кнопок Прочитать текст из файла и Добавить текст в файл. Процедура обработки события нажатия кнопки Прочитать текст из файла была описана в упражнении 3.9. Для создания процедуры обработчика события нажатия кнопки Добавить текст в файл выберите в окне Инспектор объектов объект Button2, на странице События сделайте двойной щелчок на пустом поле списка в событии OnClick. После этого в окне Редактор кода будет сгенерирована заготовка процедуры обработчика события. Чтобы эта процедура открывала существующий текстовый файл и добавляла в него текст, являющийся значением свойства Text объекта Edit1, отредактируйте текст процедуры следующим образом:

```
procedure TForm1.Button2Click(Sender: TObject);  
var
```

```
f:TextFile; {Описание файловой переменной}
begin
  AssignFile(f,'text1.txt');
  Append(f); {Открыть существующий файл для добавления в его
конец}
  Writeln(f>Edit1.Text); {Записать в файл значение свойства
Text объекта Edit1}
  CloseFile(f); {Закрыть файл}
end;
```

Сохраните файл модуля под именем Main2, а файл проекта – под именем TextMemoFile2 в папке Обработка текстовых файлов. Откомпилируйте и запустите приложение. Щелкнув мышью по кнопке Прочитать текст из файла, убедитесь, что текст из файла на диске считывается в окно Memo1 (рис. 3.18).

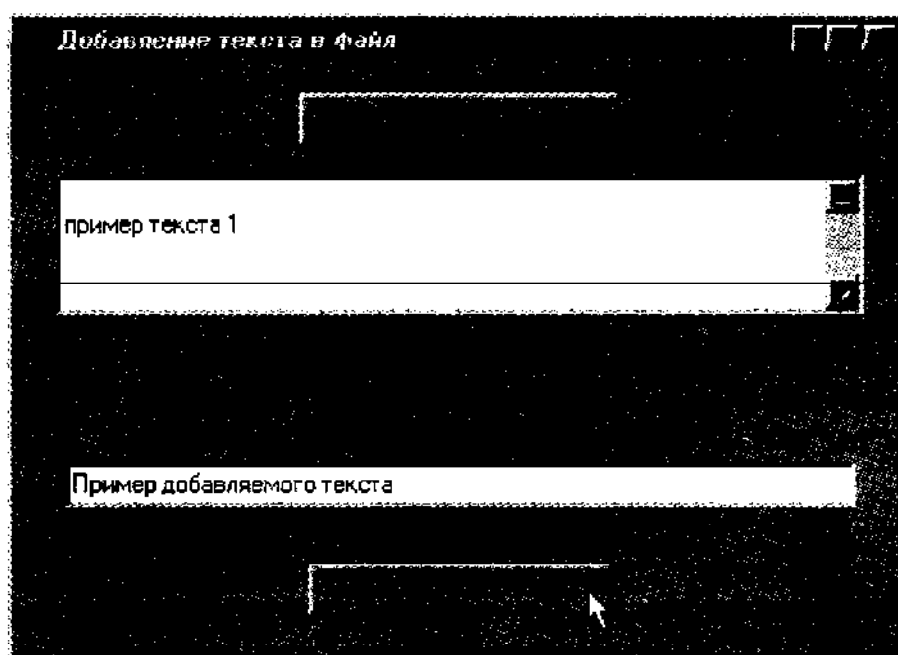


Рис. 3.18. Окно приложения, добавляющего текст в файл

Введите текст в окно Edit1 и щелкните мышью по кнопке Добавить текст в файл. Щелкнув по кнопке Прочитать текст из файла, просмотрите текст, прочитанный из файла, и убедитесь, что при повторном считывании из файла считывается текст, включающий фрагмент, только что добавленный вами. Закройте окно приложения.

Упражнение 3.11. Создайте приложение, формирующее текстовый файл и сохраняющее в нем текст из Мемо, используя метод SaveDialog для выбора папки и задания имени и расширения текстового файла.

Создайте форму, на которой расположите компоненты Memo1, Label1 и Button1. Задайте для свойства Form1.Caption значение «Пример с использованием

SaveDialog», для Label1.Caption – значение «Введите текст», для Button1.Caption – значение «Сохранить». Удалите текст «Memo1» из окна Memo1. Выровняйте и зафиксируйте визуальные компоненты на форме (рис. 3.19). Выберите в палитре компонентов страницу Dialogs и поместите на форму компонент SaveDialog. Так как он не является визуальным компонентом, его можно поместить в любое место формы. Задайте для свойства SaveDialog1.Title значение «Сохранить текстовый файл», которое будет отображаться в заголовке диалогового окна сохранения файла.

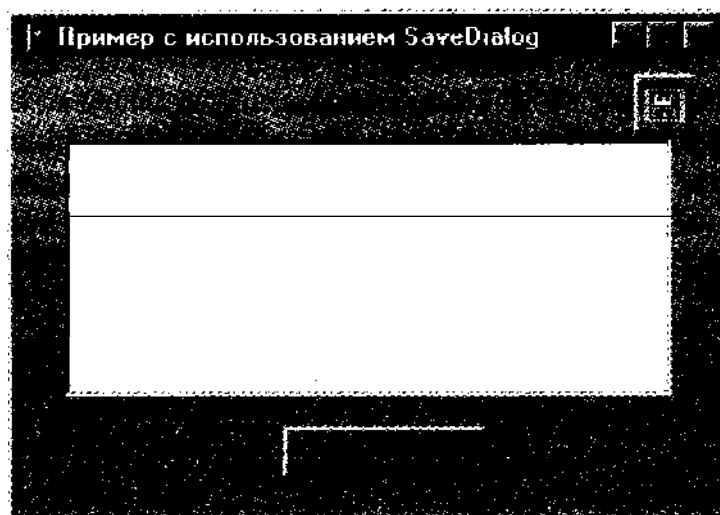


Рис. 3.19. Форма приложения

Чтобы при сохранении файла в окне диалога обеспечить выбор типа файла, выберите в окне Инспектор объектов объект SaveDialog1, на странице Свойства сделайте двойной щелчок в списке значений свойства Filter. В окне Filter Editor задайте фильтры для выбора типа и расширения файла (рис. 3.20).

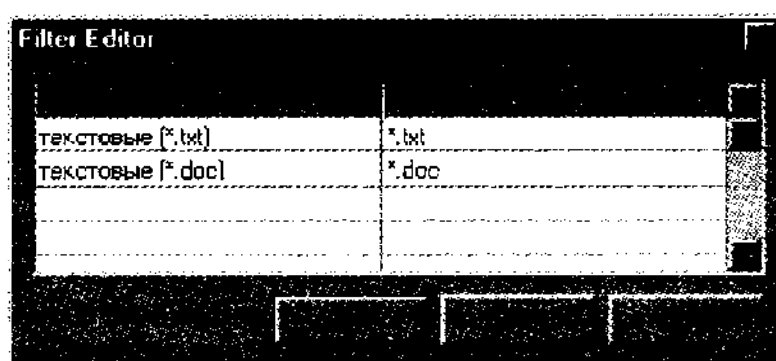


Рис. 3.20. Редактирование фильтров SaveDialog

Щелкнув по кнопке ОК, закройте окно Filter Editor. Для установки расширения *.txt по умолчанию задайте значение 1 для свойства SaveDialog1.FilterIndex.

Примечание. Варианты расширения файла и значение расширения по умолчанию можно указать в тексте процедуры записи файла, например, следующим образом: `SaveDialog1.Filter:="Text files (*.txt)|*.txt|doc files(*.doc)|*.doc";`

`SaveDialog1.FilterIndex := 1.`

Для создания процедуры обработчика события щелчка мышью по кнопке Сохранить выберите в окне Инспектор объектов объект Button1 и на странице События дважды щелкните мышью на пустом поле списка в событии OnClick. После этого в окне Редактор кода будет сгенерирована заготовка процедуры обработчика события `procedure TForm1.Button1Click(Sender: TObject);`.

Введите в раздел описания переменных модуля строковую переменную FName для хранения имени файла:

```
var
```

```
    Form1: TForm1;
```

```
    FName: string;
```

Отредактируйте текст процедуры следующим образом:

```
procedure TForm1.Button1Click(Sender: TObject);
```

```
begin
```

```
    FName:='Text1';
```

```
    SaveDialog1.FileName := FName; {Присвоить свойству FileName  
значение из переменной FName}
```

```
    if SaveDialog1.Execute then begin
```

```
        FName:=SaveDialog1.FileName;    {Открыть диалог и запомнить  
новое имя файла}
```

```
        case SaveDialog1.FilterIndex of {Изменить расширение  
файла}
```

```
            1 : FName:=FName+'.txt';
```

```
            2 : FName:=FName+'.doc';
```

```
        end;
```

```
        Memo1.Lines.SaveToFile(FName); {Записать в файл содержимое  
из свойства Lines объекта Memo}
```

```
    end;
```

```
end;
```

Как видно из текста процедуры, сначала переменной FName присваивается значение «Text1», затем это значение присваивается свойству SaveDialog1.FileName. Приложение открывает диалоговое окно «Сохранить текстовый файл», в котором задается имя файла. Имя файла из свойства SaveDialog1.FileName запоминается в переменной FName. При помощи оператора case реализуется выбор расширения файла в зависимости от значения свойства SaveDialog1.FilterIndex. В заключительной части процедуры оператор Memo1.Lines.SaveToFile(FName); используется для записи в файл содержимого свойства Lines объекта Memo1.

Сохраните, откомпилируйте и запустите на выполнение созданное приложение. Введите текст в окне приложения и щелкните мышью по кнопке Сохранить. Будет открыто диалоговое окно Сохранить текстовый файл (рис. 3.21).

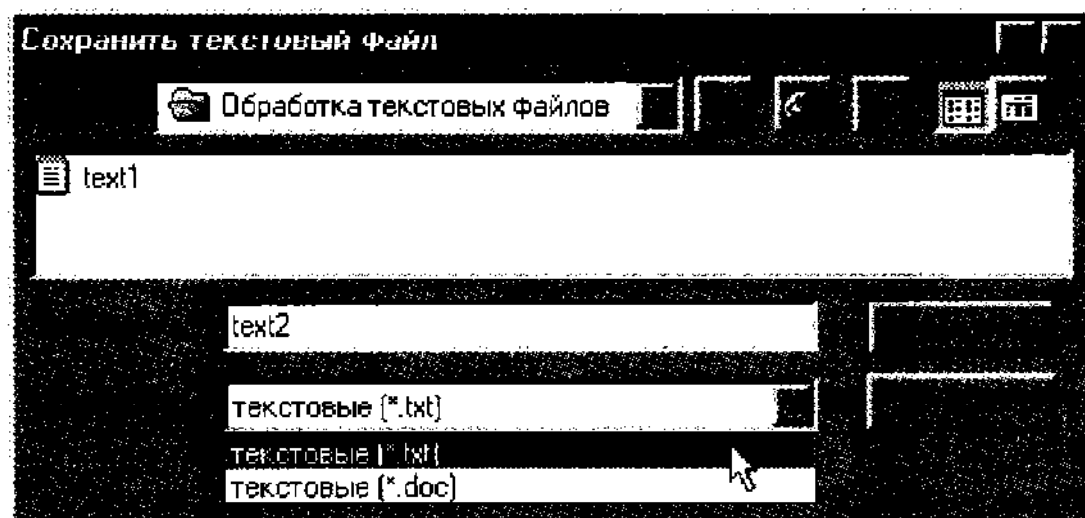


Рис. 3.21. Диалоговое окно сохранения файла

Изменив имя файла и выбрав тип файла, нажмите кнопку Сохранить. Открыв в окне Проводника папку, в которой был сохранен файл (в нашем примере папка называется «Обработка текстовых файлов»), убедитесь, что файл с указанным вами именем в ней есть. Дважды щелкнув мышью по этому файлу, откройте его в окне редактора Блокнот и убедитесь, что это файл, созданный при проверке работы приложения. Закройте окна редактора Блокнот и приложения Пример с использованием SaveDialog.

Упражнение 3.12. Создайте приложение, которое открывает текстовый файл с использованием метода OpenFileDialog, считывает текст из него в объект Memo, затем сохраняет измененный текст в файле с использованием метода SaveDialog и выводит текст на печать, используя метод PrintDialog.

Создайте форму и разместите на ней компоненты Memo1, Button1, Button2 и Button3. Задайте для свойства Form1.Caption значение «Пример с OpenFileDialog, SaveDialog, PrintDialog», для Button1.Caption – «Открыть», для Button2.Caption – «Сохранить», для Button3.Caption – «Печатать». Удалите текст «Memo1» из окна Memo1. Для обеспечения возможности просмотра в окне Memo1 длинных текстов с использованием вертикальной полосы прокрутки установите для свойства Memo1.ScrollBars значение ssVertical. Выровняйте и зафиксируйте визуальные компоненты на форме, как показано на рис. 3.22. Выберите в палитре компонентов страницу Dialogs и поместите на форму компоненты OpenFileDialog, SaveDialog и PrintDialog. Так как они не являются визуальными компонентами, то их можно поместить в любое место формы. Задайте для свойства SaveDialog1.Title значение «Сохранить текстовый файл», а для OpenFileDialog1.Title – «Открыть текстовый файл». Эти текстовые строки будут отображаться в заголовках диалоговых окон сохранения и открытия файла.



Рис. 3.22. Окно Form1 с компонентами приложения

Чтобы реализовать выбор типа файла при открытии файла в окне диалога, выберите в окне Инспектор объектов объект `OpenDialog1`, на странице Свойства дважды щелкните мышью по списку значений свойства `Filter`. В окне `Filter Editor` задайте фильтры для выбора типа и расширения файла (рис. 3.23).

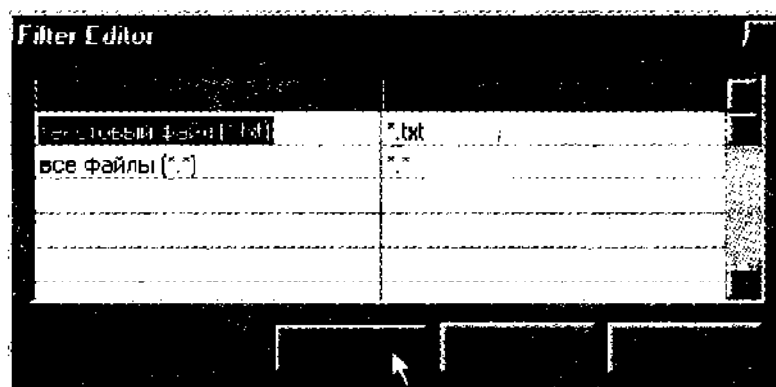


Рис. 3.23. Редактирование фильтров OpenFileDialog

Чтобы установить в качестве расширения файла по умолчанию первый вариант (*.txt), задайте для свойства `OpenDialog1.FilterIndex` значение 1. Отредактируйте фильтры `SaveDialog1`, как описано в предыдущем упражнении. Чтобы в диалоговом окне Печать включить возможность выбора диапазона печатаемых страниц и печати выделенного фрагмента, задайте для свойств `PrintDialog1.Options.poPageNums` и `PrintDialog1.Options.poSelection` значение `True`.

Примечание. Чтобы можно было использовать принтер, добавьте в список модулей `uses` модуль `PRINTERS`. В этом модуле имеется переменная `Printer:TPrinter`, что избавляет от необходимости описывать свою. Модуль `PRINTERS` позволяет выводить данные на печать и управлять процессом печати.

После этого раздел описания модулей будет выглядеть следующим образом:

```
uses
```

```
Windows, Messages, SysUtils, Classes, Graphics, Controls,  
Forms, Dialogs, StdCtrls, Printers;
```


Отредактируйте раздел описания переменных модуля следующим образом:

```
var
  Form1: TForm1; {Описание формы}
  FName : string; {Переменная, в которой хранится имя файла}
  F: TextFile; {Файловая переменная}

  S: string; {Переменная для хранения данных, прочитанных из
файла}
```

Процедура обработчика щелчка мышью по кнопке Button2 (Сохранить) создается так, как описано в предыдущем упражнении. Чтобы файл можно было сохранять с другим расширением, отредактируйте фрагмент, описывающий выбор типа файла, добавив строки удаления прежнего расширения файла:

```
case SaveDialog1.FilterIndex of {Изменить расширение файла}
  1 : if Copy(FName, Pos('.', FName), 4) <> '.txt' then begin
      Delete(FName, Pos('.', FName), 4); {Удалить расширение
файла}

      FName:=FName+'.txt';
    end;
  2 : if Copy(FName, Pos('.', FName), 4) <> '.doc' then begin
      Delete(FName, Pos('.', FName), 4);
      FName:=FName+'.doc';
    end;
end;
```

Для создания процедуры обработчика щелчка мышью по кнопке Открыть выберите в окне Инспектор объектов объект Button1 и на странице События сделайте двойной щелчок на пустом поле списка в событии OnClick. После этого в окне Редактор кода будет сгенерирована заготовка процедуры обработчика события `procedure TForm1.Button1Click(Sender: TObject);`.

Отредактируйте текст процедуры следующим образом:

```
procedure TForm1.Button1Click(Sender: TObject); {Открыть
файл}
begin
  if OpenFileDialog1.Execute then
    begin
      AssignFile(F, OpenFileDialog1.FileName); {Выбор файла в
диалоговом окне}

      FName:=OpenDialog1.FileName; {Имя выбранного файла}
      Reset(F); {Открыть существующий файл}
```

```

      Readln(F, S);    {Прочитать содержимое файла в переменную S}
      Memo1.Text := S;    {Присвоить значение S свойству Text
объекта Memo1}
      CloseFile(F);
    end;
  end;

```

Как видно из текста процедуры, в начале процедуры открывается диалоговое окно, в котором выбирается файл (FName:=OpenDialog1.FileName;), затем имя файла связывается с файловой переменной (AssignFile(F,OpenDialog1.FileName;)), и файл открывается для чтения (Reset(F;)). После этого содержимое открытого файла считывается в переменную S (Readln(F, S;)) и присваивается свойству Text объекта Memo1 (Memo1.Text:= S;). После завершения чтения содержимого файл закрывается (CloseFile(F;)).

Для создания процедуры обработчика щелчка мышью по кнопке Печатать выберите в окне Инспектор объектов объект Button3 и на странице События сделайте двойной щелчок на пустом поле списка в событии OnClick. После этого в окне Редактор кода будет сгенерирована заготовка процедуры обработчика события procedure TForm1.Button3Click(Sender: TObject);, которую нужно отредактировать следующим образом:

```

procedure TForm1.Button3Click(Sender: TObject);
begin
  if PrintDialog1.Execute then
  begin
    AssignPrn(F); {Объявить файловую переменную F -
принтер}
    Rewrite(F);           {Открыть доступ к принтеру}
    Writeln(F,Memo1.Text); {Вывести текст из Memo1 на
принтер}
    System.CloseFile(F);  {Закрыть доступ к принтеру}
  end;
end;

```

Сохраните, откомпилируйте и запустите на выполнение созданное приложение. Щелкните мышью по кнопке Открыть, в диалоговом окне Открыть текстовый файл выберите папку, задайте тип файла и выберите текстовый файл, после чего нажмите кнопку Открыть. Отредактируйте текст в окне приложения и нажмите кнопку Сохранить. После этого в диалоговом окне Сохранить текстовый файл выберите в поле Тип файла расширение для сохраняемого файла, задайте его имя и щелкните мышью по кнопке Сохранить. Для вывода считанного из файла текста на печать щелкните мышью по кнопке Печатать. В диалоговом окне Печать выберите принтер, укажите число копий, диапазон

печатаемых страниц и нажмите ОК. Открыв в окне Проводник папку, в которой был сохранен файл, убедитесь, что файл с указанным вами именем в ней есть. Убедитесь в том, что текст файла напечатан на выбранном вами принтере, и закройте окно приложения.

Что нового вы узнали?

В этой главе вы познакомились с понятием «динамический массив».

Вы узнали назначение и порядок изменения свойств визуальных компонентов GroupBox, CheckBox, StringGrid, Memo, SaveDialog, OpenFileDialog, PrintDialog.

Вы научились создавать приложения с графическим интерфейсом для обработки структурированных типов данных: строк, массивов и файлов.

Контрольные вопросы и задания

Вопросы

1. Опишите четыре типа строк в языке Object Pascal Delphi 5–6: String, ShortString, AnsiString, WideString.
2. Опишите назначение процедур и функций обработки строк в стиле Pascal: Concat, Copy, Delete, FloatToStr, Insert, Length, Pos, StrToInt.
3. Опишите различия статических и динамических массивов. Каков порядок описания и применения динамических массивов?
4. Опишите назначение операций и функций для массивов: операции присваивания, функций Length, High, Low.
5. Каково назначение и в чем специфика следующих функций для массивов: MaxIntValue, MinIntValue, MaxValue, MinValue, Sum?
6. Какие функции применимы только к символьным массивам?
7. Каково назначение визуального компонента GroupBox? Как задать надпись в этом компоненте?
8. Чем отличаются компоненты RadioGroup и GroupBox? Как задать список элементов в объекте RadioGroup?
9. Для чего и каким образом фиксируется положение компонентов на форме?
10. Каково назначение визуального компонента StringGrid? Что содержат его свойства Align, ColCount, FixedRows?
11. С какой целью используется оператор with .. do для обращения к элементам объекта?
12. Опишите назначение процедур и функций файлового ввода-вывода: Append, AssignFile, CloseFile, Eof, IOResult, Read, Reset, Rewrite, Seek, Write.
13. Каково назначение директивы компилятора {\$I-}?
14. Опишите назначение компонентов OpenFileDialog, SaveDialog, PrintDialog страницы Dialogs палитры компонентов. Каковы особенности их размещения на форме приложения?
15. Каково назначение свойства SaveDialog1.Filter? Какими способами можно изменять его значение?

Задания

1. Измените значок приложения, созданного при выполнении упражнения 3.1, создав, например, значок с изображением буквы «Т» в белом круге.

2. Создайте приложение, предлагающее пользователю ввести строку, а затем определяющее, является ли введенная строка перевертышем.

Примечание. Перевертышем (палиндромом) назовем текст, одинаково читающийся слева направо и справа налево, например «шалаш».

3. Создайте приложение, которое предлагает пользователю ввести строку, а затем выводит обращенную строку текста, например: «Привет» – «тевирП».

4. Дополните приложение обработки линейного массива, созданное при выполнении упражнения 13.3, таким образом, чтобы пользователь мог определить диапазон изменений значений элементов массива.

5. Измените приложение, созданное при выполнении упражнения 13.8, таким образом, чтобы в создаваемый текстовый файл выводился текст – значение свойства `text` компонента `Мето`.

6. Создайте приложение, которое выполняет следующие операции с текстовым файлом: открывает текстовый файл с расширением `*.txt` или `*.doc`, указанный пользователем в диалоговом окне, затем изменяет текст, удаляя из него последовательность символов, заключенную в фигурные скобки, и сохраняет файл на диске с новыми именем и расширением, которые задает пользователь в диалоговом окне.

ГЛАВА 4

Приложения с мультимедиа

В этой главе мы рассмотрим технологию использования интегрированной среды программирования Delphi 6 при создании приложений, использующих средства мультимедиа.

После изучения главы вы будете знать:

- Особенности обработки графики при рисовании по пикселям, пером, кистью.
- Создание мультипликации программированием движений объекта.
- Порядок создания приложений, воспроизводящих звук и видеоклипы.
- Процедуры и функции воспроизведения аудио- и видеоинформации.
- Порядок использования конструктора меню для создания меню Windows-программы.

Интегрированная среда программирования Delphi обеспечивает программисту возможность создания приложений, использующих средства мультимедиа.

Мультимедиа – это интеграция технологий, позволяющая объединить в компьютере практически все виды информационных сообщений: текст, графику, анимацию, аудио- и видеосообщения – и обеспечить активное взаимодействие человека с этими данными в реальном масштабе времени.

Рассмотрим примеры использования графики, анимации, аудио и видео в приложениях, созданных в среде Delphi.

Канва и пиксели

Многие компоненты в Delphi имеют свойство *Canvas* (канва, холст), позволяющее рисовать или отображать готовые изображения в определенной области компонента. Свойство *Canvas* типа *TCanvas* используется для рисования пером (Pen) и кистью (Brush), для модификации изображения и наложения друг на друга нескольких изображений. Класс *TCanvas* является основой графической подсистемы Delphi. Канва обеспечивает:

- загрузку и хранение графических изображений;
- создание новых и изменение имеющихся изображений с помощью пера, кисти, шрифта;
- рисование и закраску различных фигур, линий, текстов;
- комбинирование различных изображений.

Канва реализует свойства и методы, существенно упрощающие работу с графикой в Delphi. С ними можно подробно ознакомиться при помощи спра-

вочной системы Delphi. Каждая точка канвы имеет координаты X и Y. Началом координат канвы, как и везде в Delphi, является ее левый верхний угол. Координата X возрастает при перемещении слева направо, а координата Y – при перемещении сверху вниз. Координаты измеряются в пикселях. **Пиксель** – это наименьший элемент поверхности рисунка, с которым можно манипулировать. Важнейшее свойство пикселя – его цвет. Для описания цвета используется тип TColor. Рисовать на канве можно разными способами, например, окрашивая пиксели канвы.

Рисование на канве по пикселям. Для рисования на канве пикселями используется свойство канвы Pixels. Это свойство задает двухмерный массив, который отвечает за цвета канвы. Например, Canvas.Pixels[15,25] соответствует цвету пикселя, 15-го слева и 25-го сверху. С массивом пикселей можно обращаться, как с любым свойством: изменять цвет, присваивать пикселю новое значение или определять его цвет по хранящемуся в нем значению. Например, при помощи операторов Canvas.Pixels[15,25] := 0 или Canvas.Pixels[15,25] := clBlack для пикселя задается черный цвет.

Упражнение 4.1. Создайте приложение, которое строит пикселями график функции $y = 3\sin(x)\cos(x/6)$.

Создайте форму, ее свойству Caption присвойте значение «График функции». В окне Инспектор объектов задайте для свойства Form1.ClientHeight (высота окна формы) значение 300, для свойства Form1.ClientWidth (ширина окна формы) – значение 500, для свойства Form1.Color – значение clWhite (белый).

Для создания процедуры обработчика события OnPaint (перерисовать изображение) выберите в окне Инспектор объектов объект Form1 и на странице Events дважды щелкните мышью на пустом поле списка в событии OnPaint. После этого в окне Редактор кода будет сгенерирована заготовка процедуры обработчика события procedure TForm1.FormPaint(Sender: TObject);.

Введите в текст процедуры вызов процедуры построения графика функции DrawGraph:

```
procedure TForm1.FormPaint(Sender: TObject);  
begin  
    DrawGraph; {Вызов процедуры построения графика функции}  
end;
```

Так как при программировании процедуры построения графика функции нам придется использовать для рисования Canvas (холст, канва), окрашивая пиксели канвы, изучите справочную информацию о свойствах Canvas и Pixels. Поскольку перед построением графика придется чертить прямые линии – оси координат и выводить текст на изображении, следует также изучить справку Delphi о методах MoveTo, LineTo и TextOut.

В связи с тем, что при построении графика функции необходимо задавать целочисленные значения координат окрашиваемого пикселя, а результат вычисления функции является вещественным числом, следует ознакомиться со

справочной информацией Delphi, касающейся функции Round, округляющей вещественные числа до ближайшего целого числа.

Откройте окно Редактор кода и введите текст процедуры построения графика функции:

```

procedure DrawGraph; // Построение графика функции
var
    x1,x2 : real; {Границы изменения аргумента функции}
    y1,y2 : real; {Границы изменения значения функции}
    x : real;      {Аргумент функции}
    y : real;      {Значение функции в точке x}
    dx : real;     {Приращение аргумента}
    l,b : integer; {Левый нижний угол области вывода графика}
    w,h : integer; {Ширина и высота области вывода графика}
    mx,my : real;  {Масштаб по осям X и Y}
    x0,y0 : integer; {Точка - начало координат}
    Function f(x:real):real; {Функция, график которой надо
построить}
begin
    f:= 3*Sin(x)*Cos(x/6);
end;
begin
        {Область вывода графика на форме}
    l:=10;      {X - координата левого верхнего угла}
    b:=Form1.ClientHeight-20; {Y - координата левого верхнего
угла}
    h:=Form1.ClientHeight-40; // Высота
    w:=Form1.Width-40; // Ширина
    x1:=0; // Нижняя граница диапазона аргумента
    x2:=25; // Верхняя граница диапазона аргумента
    dx:=0.01; // Шаг аргумента
    {Определение максимального и минимального значений функции на
отрезке [x1,x2] для масштабирования}
    y1:=f(x1); {Минимальное значение функции}
    y2:=f(x1); {Максимальное значение функции}
    x:=x1;
    {Вычислить значения функции для x<=x2 и определить
максимальное и минимальное значения для масштабирования}

```

```

repeat
  y:=f(x);
  if y<y1 then y1:=y;      {Если текущее значение y меньше
минимального, то y1:=y}
  if y>y2 then y2:=y;      {Если текущее значение y больше
минимального, то y2:=y}
  x:=x+dx;                  {Перейти к следующему значению x}
until (x>=x2);              {Как только x>=x2, прекратить}
my:=h/abs(y2-y1);           // Масштаб по оси Y
mx:=w/abs(x2-x1);           // Масштаб по оси X
                             {Задать начало координат}
x0:=1;
y0:=b-Abs(Round(y1*my));
with Form1.Canvas do        {Рисование графика}
begin                        {Начертить оси координат}
  MoveTo(1,b); LineTo(1,b-h); {Ось x}
  MoveTo(x0,y0); LineTo(x0+w,y0); {Ось y}
  TextOut(1+5,b-h,FloatToStrF(y2,ffGeneral,6,3));
                             {Вывести максимальное значение}
  TextOut(x0+w,y0,'x');      // Подписать ось x
  TextOut(1+5,b,FloatToStrF(y1,ffGeneral,6,3));
                             {Вывести минимальное значение}
  TextOut(1-7,b-h-5,'y');    // Подписать ось y
                             {Построение графика}
  x:=x1;
  repeat                    {Повторять рисование точки}
    y:=f(x);
    Form1.Canvas.Pixels[x0+Round(x*mx),y0-
Round(y*my)]:=clRed;
    {clRed - красный цвет нарисованного графика}
    x:=x+dx;
  until (x>=x2);            {Прекратить, как только x>=x2}
end;
end;

```

Изучите текст процедуры построения графика функции DrawGraph. Сохраните, откомпилируйте и запустите приложение на выполнение. Убедитесь, что в окне приложения построен график заданной функции (рис. 4.1).

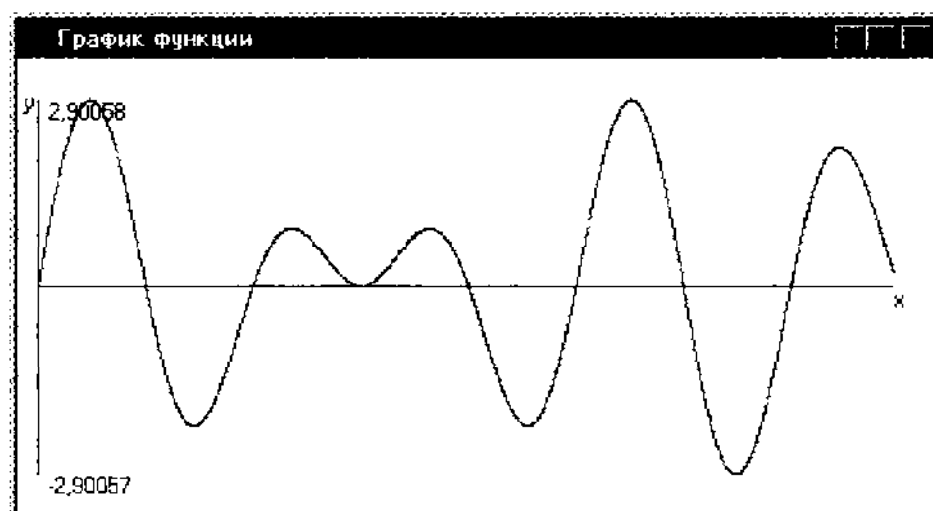


Рис. 4.1. Окно приложения с графиком функции

Если вы попытаетесь изменить размеры окна, то увидите, что график искажается, так как он не перерисовывается в соответствии с новыми размерами окна.

Для того чтобы изображение перерисовывалось при изменении размеров формы, следует создать обработчик события `OnResize`, определяющий новые параметры клиентской области и вызывающий процедуру построения графика функции `DrawGraph`. Эта процедура может выглядеть следующим образом:

```
procedure TForm1.FormResize(Sender: TObject);
begin
    with Form1 do
        Canvas.FillRect(Rect(0,0,ClientWidth,ClientHeight));
        {Заполнение прямоугольника канвы, указанного параметром Rect, где
        ClientWidth,ClientHeight - ширина и высота клиентской области в
        пикселях}
        DrawGraph;
    end;
```

Сохраните измененное приложение, откомпилируйте его и проверьте его работу. Убедитесь, что в окне приложения выполняется построение графика заданной функции, причем при изменении размеров окна происходит корректное перерисовывание графика в новом масштабе.

Упражнение 4.2. Измените приложение таким образом, чтобы оно выполняло построение графика функции $y = \sqrt{x}\sin(x)$, причем цвет графика должен быть синим.

Для изменения выражения функции отредактируйте текст функции $f(x)$ следующим образом:

```

Function f(x:real):real; {функция, график которой надо
построить}
begin
    f:=Sqrt(x)*Sin(x);

end;

```

Для изменения цвета графика измените значение свойства Form1.Canvas.Pixels с красного (clRed) на синий (clBlue) в тексте фрагмента процедуры DrawGraph, рисующего точки на канве:

```

repeat {Рисовать точки}
    y:=f(x);
    Form1.Canvas.Pixels[x0+Round(x*mx),
    y0-Round(y*my)]:=clBlue; {clBlue - синий цвет}
    x:=x+dx;
until (x>=x2); {Прекратить, как только x>=x2}

```

Сохраните измененное приложение, откомпилируйте его и проверьте его работу. Убедитесь, что в окне приложения построен график заданной функции, причем цвет пикселей графика синий.

Рисование пером. У канвы имеется свойство Pen (перо), которое определяет атрибуты пера, используемого для рисования линий и фигур. Свойство Pen, в свою очередь, имеет ряд подсвойств. Одно из них – уже известное вам свойство Color – определяет цвет, которым наносится рисунок. Второе свойство – Width (ширина линии) – определяет ширину линии, которая задается в пикселях. По умолчанию ширина равна 1 пикселю. Свойство Style определяет вид линии (сплошная, штриховая, пунктирная и т. п.). Все стили со штрихами и пунктирами доступны только при Width = 1, в противном случае линии этих стилей рисуются как сплошные.

Для определения в координатах канвы текущей позиции пера используется свойство канвы PenPos типа TPoint. Перемещение пера без прорисовки линии, т.е. изменение PenPos, производится при помощи метода канвы MoveTo(X,Y). Здесь X и Y – координаты точки, в которую перемещается перо. Эта текущая точка становится исходной, из нее при помощи метода LineTo(X,Y) можно провести линию в точку с координатами (X, Y). При этом текущая точка перемещается в конечную точку линии, и новый вызов LineTo будет проводить линию из этой новой текущей точки.

Упражнение 4.3. Создайте приложение, которое по щелчку мышью по кнопке выполняет рисование пером графика функции $y = \sin(x)\cos(x/6)$ на интервале $X_{\min} = 0, \dots, X_{\max} = 4\pi$. Задайте толщину пера 2 пикселя, цвет линии clBlue (синий).

Создайте форму и задайте для ее свойства Caption значение «Рисование графика функции пером». Поместите в верхнюю часть формы компонент Image1

из палитры Additional. Ниже Image1 на форме расположите компонент Button1 из палитры Standard и задайте для его свойства Caption значение «Построить график».

Рисование графика функции пером опишите в процедуре обработчика щелчка мышью по кнопке Button1. Для этого, выбрав в окне Инспектор объектов Button1, на странице Events дважды щелкните на пустом поле списка в событии OnClick.

Так как в процедуре обработчика события необходимо обрабатывать свойства Pen, Font, а также использовать методы LineTo, MoveTo, TextOut, вам следует изучить соответствующие разделы справочной системы Delphi.

После этого в окне Редактор кода отредактируйте текст процедуры обработчика события TForm1.Button1Click следующим образом:

```
procedure TForm1.Button1Click(Sender: TObject);
var
    X,Y : real; {Значения аргумента и функции}
    PX,PY : longint; {Координаты пикселей, соответствующих X, Y}
begin
    with Image1.Canvas do
        begin
            Pen.Width:=1; {Задать толщину пера для рисования осей
координат}
            Pen.Color:=clBlack; {Задать черный цвет линии}
            {Начертить оси координат: начало координат в точке 5,Image1.
Height div 2}
            MoveTo(0,Image1.Height div 2);
            LineTo(Image1.Width,Image1.Height div 2); {Ось x}
            MoveTo(5,0);
            LineTo(5,Image1.Height); {Ось y}
            {Рисовать график от начала координат на всю ширину Image1}
            Pen.Width:=2; {Задать толщину пера для рисования
графика}
            Pen.Color:=clBlue; {Задать синий цвет линии}
            MoveTo(5,Image1.Height div 2); {Установить перо в начало
координат}
            for PX:=5 to Image1.Width do
```

```

begin
    X:=PX*4*Pi/Image1.Width; {Вычисление значения
аргумента X для масштабирования графика в соответствии с шириной
Image1}

    Y:=Sin(x)*Cos(x/6); {Вычисление значения функции}
    PY:=Trunc(Image1.Height-(Y+1)*Image1.Height/2);
    {Вычисление координаты пикселя, соответствующей значению
функции Y}

    LineTo(PX,PY);          {Рисование линии}
end;

Font.Size:=18;             {Задать размер шрифта в пунктах}
TextOut(180,20,'y=Sin(x)*Cos(x/6)'); {Подписать график}
end;
end;

```

Изучите текст процедуры и комментарии. Сохраните текст приложения, откомпилируйте его и проверьте его работу. Щелкнув мышью по кнопке Построить график, убедитесь, что в окне приложения нарисован график заданной функции, причем цвет пикселей – синий, а толщина линии составляет 2 пикселя (рис. 4.2).

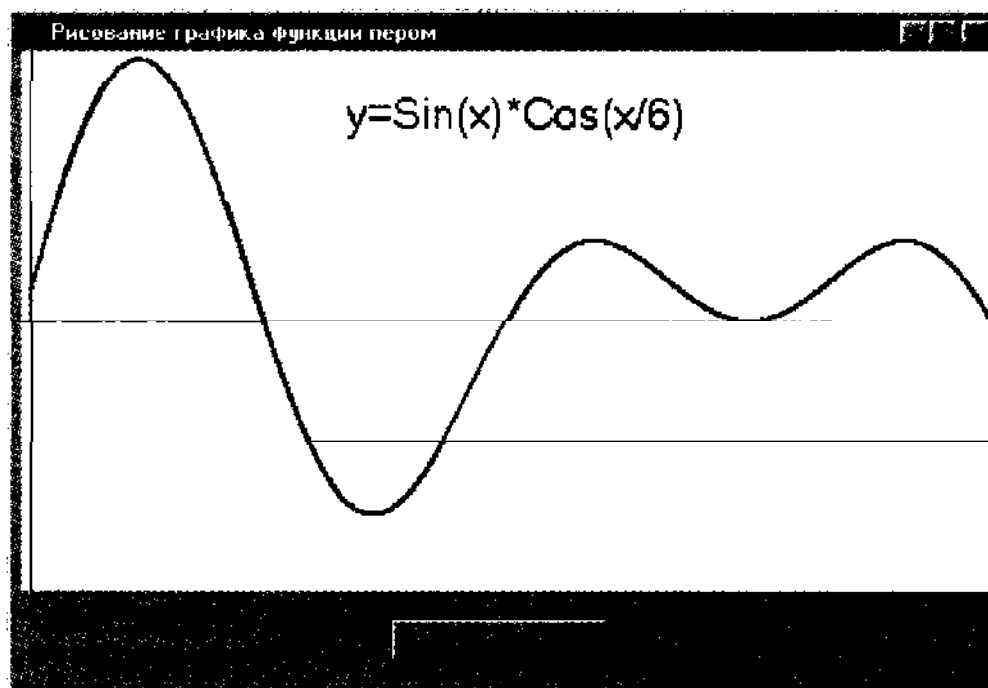


Рис. 4.2. Рисование графика функции пером

Перо может рисовать не только прямые линии, но и геометрические фигуры. Для рисования фигур пером используются следующие методы.

Метод	Действие
Arc	Дуга окружности или эллипса
Chord	Замкнутая фигура, ограниченная дугой окружности (или эллипса) и хордой
Ellipse	Окружность или эллипс
Pie	Сектор окружности или эллипса
PolyBezier	Кусочная кривая третьего порядка с точным отображением первой и последней точек; число точек должно быть кратно 3
PolyBezierTo	Кусочная кривая третьего порядка с точным отображением последней точки; число точек должно быть на 1 меньше числа, кратного 3
Polygon	Замкнутая фигура с кусочно-линейной границей
Polyline	Кусочно-линейная кривая
Rectangle	Прямоугольник
RoundRect	Прямоугольник со скругленными углами

Упражнение 4.4. Создайте приложение, которое рисует пером домик.

Создайте форму и задайте для ее свойства Caption значение «Рисунок домика». Поместите в верхнюю часть формы компонент Image1 из палитры Additional. Ниже Image1 на форме расположите компонент Button1 из палитры Standard и задайте для его свойства Caption значение «Нарисовать».

Рисование пером графика функции опишите в процедуре обработчика щелчка мышью по кнопке Button1. Для этого, выбрав в окне Инспектор объектов Button1, на странице Events дважды щелкните на пустом поле списка в событии OnClick.

Так как в процедуре обработчика события необходимо обрабатывать свойства Pen, Font, а также использовать методы LineTo, MoveTo, TextOut, Arc, Chord, Ellipse, Pie, PolyBezier, PolyBezierTo, Polygon, Polyline, Rectangle, RoundRect, вам следует ознакомиться с соответствующими разделами справочной системы Delphi.

После этого в окне Редактор кода отредактируйте текст процедуры обработчика события TForm1.Button1Click следующим образом:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
    with Image1.Canvas do
    begin
        Pen.Width:=1;           {Задать толщину пера для рисования}
        Pen.Color:=clBlack;      {Задать черный цвет линии}
        MoveTo(0,200);
        LineTo(Image1.Width,200); {Начертить поверхность земли}
```

```
MoveTo(100,200);  
Rectangle(100,100,220,200); {Стены}  
MoveTo(100,200);  
Polyline([Point(100,100),Point(160,50),Point(220,100)]); {Крыша}  
Ellipse(140,70, 180,90);    {Окошко на чердаке}  
Rectangle(115,130,150,200); {Дверь}  
Rectangle(165,130,205,165); {Окно}  
Font.Size:=16;    {Задать размер шрифта в пунктах}  
TextOut(140,220,'Домик');    {Подписать рисунок}  
end;  
end;
```

Изучите текст процедуры и комментарии. Сохраните текст приложения, откомпилируйте его и проверьте его работу. Щелкнув мышью по кнопке Нарисовать, убедитесь, что в окне приложения нарисован домик (рис. 4.3).



Рис. 4.3. Окно приложения с нарисованным домиком

Рисование кистью. У канвы имеется свойство `Brush` – кисть. Оно определяет фон и заполнение замкнутых фигур на канве. Свойство `Brush`, в свою очередь, имеет ряд подсвойств:

`Color` – задает цвет кисти, по умолчанию – `clWhite` (белый);

`Handle` – дескриптор кисти окна, определяющий доступ к дескриптору объекта GDI Windows;

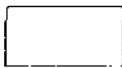
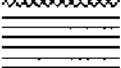
`Style` – определяет шаблон заполнения (штриховку);

Bitmap – указатель на внешнюю матрицу растрового изображения, используемую в качестве шаблона заполнения. Свойство кисти Bitmap определяет нестандартное заполнение заданным шаблоном, который задается матрицей размером 8×8 разрядов. Если для кисти задан шаблон Bitmap, то заполнение производится именно этим шаблоном, независимо от значения свойства Style. Шаблон Bitmap может создаваться в процессе выполнения приложения или, например, загружаться из файла, как в приведенном далее примере:

```
var
  Bitmap: TBitmap;
begin
  Bitmap := TBitmap.Create; {Создание объекта}
  try
    Bitmap.LoadFromFile('MyBitmap.bmp'); {Загрузка в объект
    Bitmap матрицы из файла}
    Image1.Canvas.Brush.Bitmap := Bitmap; {Использование
    шаблона для заполнения фигур на канве Image1}
    ...
  finally
    Image1.Canvas.Brush.Bitmap := nil; {Обнуление указателя}
    Bitmap.Free; {Освобождение памяти}
  end;
end;
```

В этом примере создается объект Bitmap типа TBitmap и в него загружается матрица из файла с именем MyBitmap.bmp. Затем свойству Image1.Canvas.Brush.Bitmap присваивается указатель на этот объект. После этого загруженный шаблон можно использовать для заполнения фигур на канве Image1. В конце фрагмента кода свойству Bitmap присваивается значение nil, после чего заполнение опять начинает определяться свойством Style. Затем объект Bitmap уничтожается, чтобы освободить занимаемую им память.

Свойство Style может принимать следующие значения.

Значение	Шаблон	Значение	Шаблон
bsSolid		bsCross	
bsClear		bsDiagCross	
bsBDiagonal		bsHorizontal	
bsFDiagonal		bsVertical	

Например, следующие операторы изменяют цвет и стиль заполнения объекта Image1.Canvas – канвы компонента Image1:

```
Image1.Canvas.Brush.Style := bsCross;  
Image1.Canvas.Brush.Color := clRed;  
Image1.Canvas.FillRect(Rect(0,0,Image1.Width,Image1.Height));
```

Последний из приведенных операторов заполняет при помощи метода FillRect всю поверхность канвы. Метод FillRect позволяет нарисовать заполненные фигуры. Он объявляется как

```
procedure FillRect(const Rect: TRect);
```

и позволяет заполнить заданным стилем или шаблоном прямоугольную область, заданную параметром Rect. Этот параметр имеет тип TRect. Для его задания проще всего использовать функцию Rect(X1,Y1,X2,Y2), возвращающую структуру Rect с координатами углов, заданных параметрами (X1, Y1) и (X2, Y2).

Функцию FillRect удобно использовать для стирания изображения. Например, оператор

```
with Image1 do Canvas.FillRect(Rect(0,0,Width,Height));
```

очищает всю площадь канвы компонента Image1.

Кисть участвует в заполнении фигур не только с помощью этой функции. Все перечисленные ранее методы рисования замкнутых фигур могут использоваться в сочетании с кистью. Это относится к методам Chord, Ellipse, Pie, Polygon и к прочим методам, рассмотренным при изучении рисования пером.

При работе с кистью используется также метод FloodFill, который заполняет замкнутую область на канве. Этот метод определен следующим образом:

```
type TFillStyle = (fsSurface, fsBorder) ;  
procedure FloodFill(X, Y: Integer; Color: TColor; FillStyle:  
TFillStyle);
```

Точка с координатами X и Y является произвольной внутренней точкой заполняемой области, которая может иметь произвольную форму. Граница этой области определяется сочетанием параметров Color и FillStyle. Параметр Color указывает цвет, который используется при определении границы заполняемой области, а параметр FillStyle определяет, как именно по этому цвету определяется граница. Если FillStyle = fsSurface, то заполняется область, окрашенная цветом Color, а на других цветах кисть останавливается. Если FillStyle = fsBorder, то наоборот, заполняется область, окрашенная любыми цветами, не равными Color, а на цвете Color кисть останавливается.

Для определения области закрашивания можно использовать координаты и цвет одного из пикселей, расположенных внутри области (если FillStyle = fsSurface) или снаружи нее (если FillStyle = fsBorder). Например, приведенные далее операторы закрашивают белым цветом на канве компонента Image1 все пиксели, прилегающие к пикселю с координатами (X, Y) и имеющие тот же цвет, что и этот пиксель:


```

with Image1.Canvas do
  begin
    Brush.Color:=clWhite;
    FloodFill(X,Y,Pixels[X,Y],fsSurface);
  end;

```

Приведенные далее операторы закрашивают белым цветом на канве компонента Image1 все пиксели, прилегающие к пикселю с координатами (X, Y) и имеющие цвет, отличный от черного. По достижении черной границы области закрашка прекращается:

```

with Image1.Canvas do
  begin
    Brush.Color:=clWhite;
    FloodFill(X,Y,clBlack, fsBorder);
  end;

```

Имеется еще один метод канвы, связанный с кистью. Это метод FrameRect. Он рисует на канве текущей кистью прямоугольную рамку, не закрашивая ее. Синтаксис метода FrameRect: procedure FrameRect(const Rect: TRect);. Параметр Rect определяет позицию и размеры прямоугольной рамки. Толщина рамки составляет 1 пиксель. Область внутри рамки не заполняется. Метод FrameRect отличается от рассмотренного ранее метода Rectangle тем, что рамка рисуется цветом кисти (в методе Rectangle – цветом пера) и область не закрашивается (в методе Rectangle она закрашивается).

Например, оператор

```

with Image1.Canvas do
  begin
    Brush.Color := clBlack;
    FrameRect(Rect(10,10,100,100));
  end;

```

рисует на канве компонента Image1 черную рамку.

При работе с кистью имеется возможность использовать метод TextRect для более красивого вывода текста. Например, чтобы нарисовать в заданном месте канвы компонента Image1 красный прямоугольник и внутри него по центру вывести текст, введенный пользователем в окно редактирования Edit1, можно создать следующий код:

```

var st:string;
    X1,Y1,X2,Y2:integer;
begin
  st:=Edit1.Text;
  X1:=100;      Y1:=100;      X2:=200;      Y2:=150;      {Координаты
прямоугольника}

```

```

with Image1.Canvas do
begin
  Brush.Color:=clRed; {Задать красный цвет кисти}
  {Вывести текст выровненным по центру прямоугольника}
  TextRect (Rect(X1,Y1,X2,Y2) ,
            X1+(X2-X1-TextWidth(st)) div 2,
            Y1+(Y2-Y1-TextHeight(st)) div 2,st);

end;
end;

```

Примечание. Если текст оказывается длиннее ширины прямоугольника, то он усекается. В данном примере будет видна только середина текста, так как текст выровнен по центру.

Если в приведенном примере заменить оператор TextRect на

```

TextRect (Rect(X1-5,Y1-5, X1+TextWidth(st)+5,
              Y1+TextHeight(st)+5), X1, Y1, st);

```

то текст будет выводиться полностью в красной прямоугольной области, на 5 пикселей отступающей во все стороны от текста.

Упражнение 4.5. Создайте простой графический редактор, который дает пользователю возможность загрузить рисунок, используя команду меню Файл ► Открыть, выбрать цвет и рисовать кистью, а также отменить операцию редактирования при помощи команды Правка ► Отмена.

Создайте форму и задайте для ее свойства Caption значение «Графический редактор». Поместите в нижнюю левую часть формы два компонента типа TImage, придайте им квадратную форму, например 17 × 17. Это будут окна основного и вспомогательного цветов. Присвойте этим компонентам имена Image1 и Image2.

Поместите в верхнюю правую часть формы еще один компонент, задайте его размеры такими, чтобы он занимал основную часть формы, оставив в левой части формы место для инструментальной панели размером, например 240 × 240 пунктов. Это будет холст для рисования. Присвойте этому компоненту имя Image3.

Перенесите на форму еще один компонент типа TImage и расположите его в нижней части формы правее первых двух на одном с ними уровне. Это будет палитра цветов. Сделайте ее высоту такой же, как и у первых двух компонентов (17 пунктов), а длину – примерно в 10 раз большей (160 пунктов). Присвойте этому компоненту имя Image4.

Разместите в верхнем левом углу формы кнопку типа TSpeedButton. Эта кнопка будет соответствовать кисти – типичному инструменту графических редакторов. В окне Инспектор объектов выберите объект SpeedButton1 и присвойте его свойству Name значение SbBrush. Чтобы обеспечить кнопке возможность фиксироваться в нажатом и ненажатом состояниях, установите для свойства GroupIndex кнопки SbBrush значение 1, а для свойства AllowAllUp – значение

True. Чтобы отобразить на кнопке рисунок кисти, загрузите в свойство Glyph значок кисти. Для этого в окне Инспектор объектов выберите объект SbBrush и на странице Properties дважды щелкните мышью в поле значения свойства Glyph, затем в окне Picture Editor щелкните мышью по кнопке Load и в окне Load Picture выберите соответствующий файл формата *.bmp и щелкните мышью по кнопке Открыть (рис. 4.4).

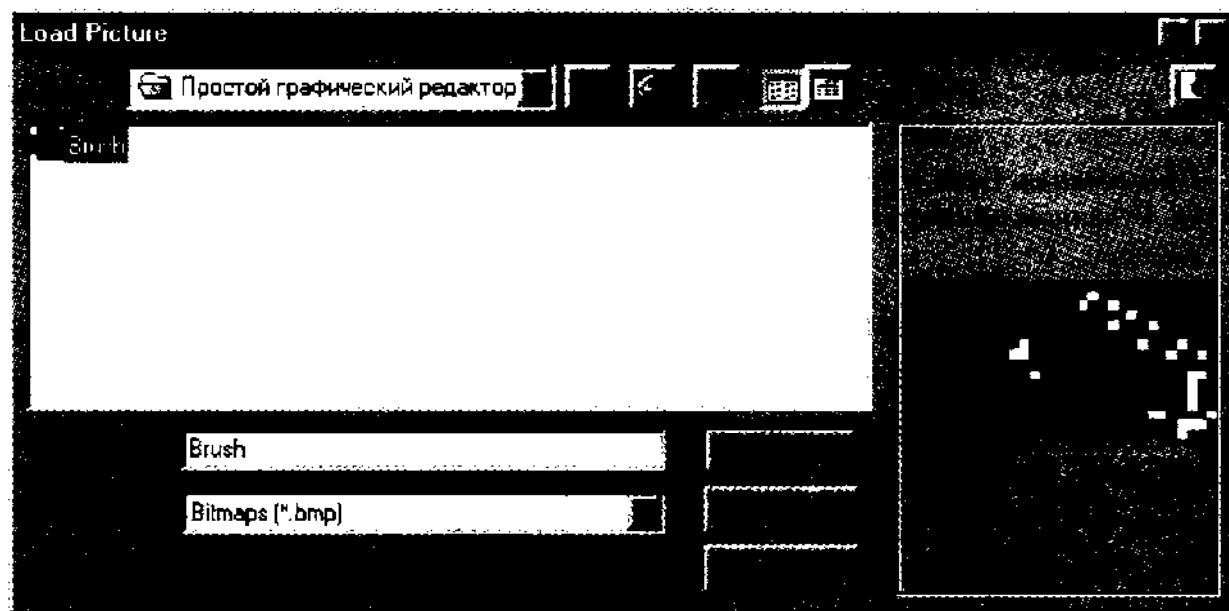


Рис. 4.4. Выбор графического файла для отображения на кнопке

Щелкнув мышью по кнопке ОК в окне Picture Editor, завершите выбор графического файла для отображения на кнопке.

Поместите на форму ниже кнопки SbBrush еще одну кнопку типа TSpeedButton и назовите ее SbColor. Установите для свойства SbColor.GroupIndex значение 1 (это гарантирует, что только одна из кнопок, SbBrush или SbColor, может быть нажата), а для свойства AllowAllUp – значение True. Загрузите в свойство Glyph значок (например, файл one2one.bmp).

Чтобы обеспечить использование диалога открытия файла, поместите на форму диалог OpenPictureDialog.

Примечание. Если вы работаете с Delphi версии 1–2, то вам следует поместить компонент OpenFileDialog и задать для его свойства Filter значение (bitmaps |*.bmp| все файлы|*.*).

Поместите на форму компонент типа TMainMenu. Свойства и методы компонента MainMenu обеспечивают объединение меню главной и вспомогательной форм и связь с меню OLE-контейнера.

Свойство Items содержит массив разделов меню типа TMenuItem, обладающих собственными свойствами, методами и событиями. Свойство Caption обозначает заголовок раздела, свойство Name – имя объекта раздела, свойство ShortCut определяет клавиши оперативного доступа к разделу. Свойство Default определяет, является ли данный раздел разделом по умолчанию для

своего подменю, т.е. разделом, выполняемым при двойном щелчке мышью по родительскому разделу. Свойство Break используется в длинных меню для того, чтобы разбить список разделов на несколько столбцов. Свойство Checked, установленное в True, указывает, что в разделе меню будет отображаться маркер флажка, показывающий, что данный раздел выбран. Еще одним свойством, позволяющим вводить маркеры в разделы меню, является RadioItem. Это свойство, будучи установленным в True, определяет, что данный раздел должен работать в режиме переключателя совместно с другими разделами, имеющими то же значение свойства GroupIndex. Свойства Enabled (доступен) и Visible (видимый) могут быть установлены для каждого раздела во время проектирования или программно во время выполнения.

Основное событие раздела меню – OnClick – возникает при щелчке мышью по разделу или при нажатии клавиш оперативного доступа.

Примечание. Начиная с версии 4, в Delphi предусмотрена возможность ввода в разделы меню изображений. Для этого предназначены свойства разделов BitMap и ImageIndex. Первое из них позволяет непосредственно ввести изображение в раздел, выбрав его из файла. Второе позволяет указать индекс изображения, хранящегося во внешнем компоненте TImageList. Указание на этот компонент можно задать в свойстве Images компонента TMainMenu.

Команды контекстного меню конструктора Create Submenu позволяют добавить в выделенный раздел подменю. Например, чтобы вставить подменю, нужно указать позицию и в контекстном меню выбрать команду Insert (рис. 4.5).

Используя конструктор меню, создайте меню приложения, показанное на рис. 4.5.

Примечание. Для ввода разделителя между пунктами подменю вставьте этот пункт командой Insert контекстного меню и задайте для его свойства Caption значение «—».

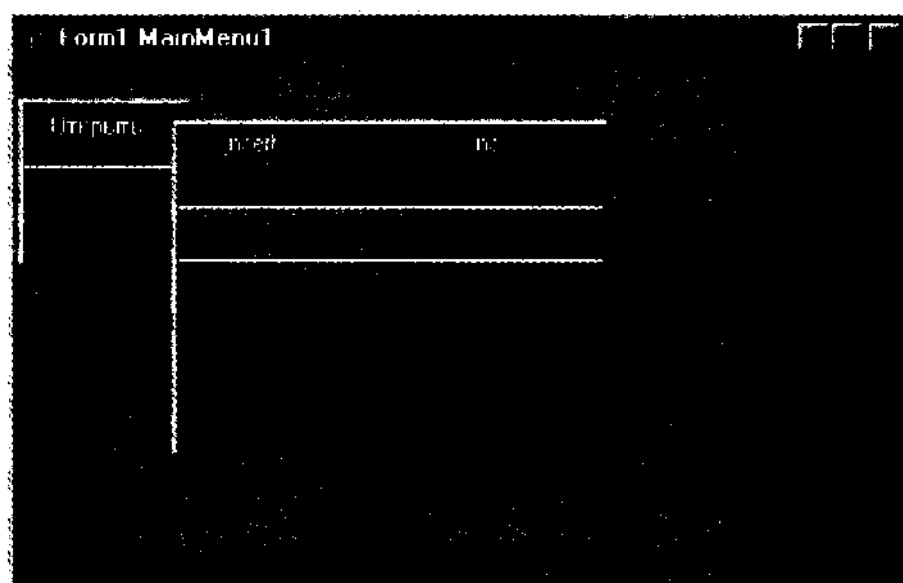


Рис. 4.5. Конструирование меню приложения

В меню создайте раздел Файл с подразделами Открыть и Выход. Конструктор меню присваивает пунктам меню по умолчанию имена N1, N2, N3 и N4. Чтобы имена пунктов меню в текстах процедур-обработчиков событий были понятнее, поменяйте их названия, изменив значения свойств Name соответствующих объектов. При этом нельзя пользоваться зарезервированными словами Object Pascal. Например, задайте для свойств Name пунктов меню значения «OpenFile» и «Exit». Для разделителя пунктов меню N3 свойство Name изменять не нужно, так оно не будет использоваться. Создайте в главном меню еще один раздел Правка с подразделом Отменить и назовите этот подраздел EditUndo. После этого состав меню будет представлен в окне Object TreeView (рис. 4.6).

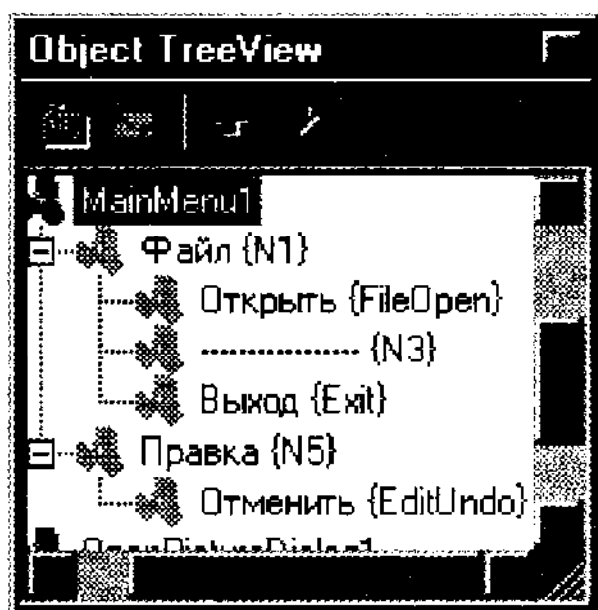


Рис. 4.6. Состав меню простого графического редактора

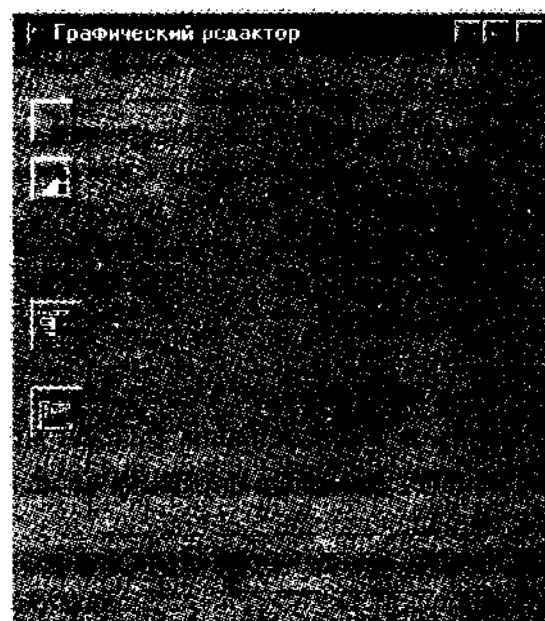


Рис. 4.7. Вид формы приложения «Графический редактор»

В результате размещения компонентов и конструирования меню получится форма приложения примерно такая, как показано на рис. 4.7.

Сохраните файлы программного модуля и проекта в папке «Простой графический редактор». Для обеспечения работы приложения отредактируйте исходный текст модуля, добавив в него описания переменных и процедур-обработчиков событий.

Чтобы сгенерировать процедуру обработчика события при создании формы приложения, выберите в окне Инспектор объектов объект Form1 и на странице Events дважды щелкните на пустом поле списка в событии OnCreate. Добавьте в раздел implementation объявление переменной

```
var Bitmap: TBitmap;
```

В этой переменной изображение будет сохраняться для того, чтобы его можно было восстановить командой Правка ► Отменить.

Отредактируйте текст процедуры обработчика следующим образом:

```
procedure TForm1.FormCreate(Sender: TObject);
var
    HW,i:integer; {HW - ширина отдельной ячейки в палитре цветов,
i - номер ячейки в палитре цветов}
begin
    Bitmap:=TBitmap.Create;
    Image1.Canvas.Brush.Color:=clBlack;    {Задать основной цвет
кисти}
    Image2.Canvas.Brush.Color:=clWhite;    {Задать вспомогательный
цвет кисти}
    {Заполнение окон основного и вспомогательного цветов}
    with Image1.Canvas do FillRect(Rect(0,0,Width,Height));
    with Image2.Canvas do FillRect(Rect(0,0,Width,Height));
    HW:=Image4.Width div 10; {Задание ширины ячейки палитры
цветов}
    {Закраска элементов палитры цветов}
    with Image4.Canvas do
        for i:=1 to 10 do
            begin
                case i of {в зависимости от i задать цвет}
                    1 : Brush.Color:=clBlack;    {черный}
                    2 : Brush.Color:=clAqua;    {голубой}
                    3 : Brush.Color:=clBlue;    {синий}
                    4 : Brush.Color:=clFuchsia; {сиреневый}
                    5 : Brush.Color:=clGreen;    {зеленый}
                    6 : Brush.Color:=clLime;    {лимонно-зеленый}
                    7 : Brush.Color:=clMaroon;  {темно-бордовый}
                    8 : Brush.Color:=clRed;     {красный}
                    9 : Brush.Color:=clYellow;  {желтый}
                    10 : Brush.Color:=clWhite;  {белый}
                end;
                Rectangle((i-1)*HW,0,i*HW,Height); {Заполнить выбранным
цветом прямоугольник-ячейку палитры цветов}
            end;
        {Рисование изображения креста на холсте - простейшего рисунка
в окне редактора}
        with Image3.Canvas do begin
            MoveTo(0,0); {Переместить перо в левый верхний угол
Image3}
```

```

    LineTo(Width,Height); {Провести линию в правый нижний угол
Image3}
    MoveTo(0,Height); {Переместить перо в левый нижний угол
Image3}
    LineTo(Width, 0); {Провести линию в правый верхний угол
Image3}
end;
Bitmap.Assign(Image3.Picture);
end;

```

Как видно из текста процедуры, сначала создается объект `Bitmap`, в котором будет храниться первоначальное изображение. Затем задаются свойства кисти окон основного (`Image1`) и вспомогательного (`Image2`) цветов: черный и белый. Окна закрашиваются соответствующим цветом с помощью функции `FillRect`. После этого формируется палитра цветов – для каждого элемента палитры задается свой цвет и элемент закрашивается этим цветом с помощью функции `Rectangle`. Затем на холсте `Image3` рисуется крест для демонстрации простого изображения. В конце процедуры нарисованное на канве изображение сохраняется в объекте `Bitmap` методом `Assign`.

Для завершения работы приложения создайте обработчик события формы `OnDestroy` и запишите в теле процедуры обработчика оператор `Bitmap.Free`., который освобождает память при закрытии приложения:

```

procedure TForm1.FormDestroy(Sender: TObject);
begin
    Bitmap.Free; {Освобождение памяти}
end;

```

Чтобы можно было загрузить изображение из файла, создайте обработчик выбора команды `Файл ▶ Открыть`, для чего в окне Инспектор объектов выберите объект `FileOpen` и на странице `Events` дважды щелкните мышью на пустом поле списка в событии `OnClick`. Отредактируйте текст процедуры следующим образом:

```

procedure TForm1.FileOpenClick(Sender: TObject);
begin
    if OpenPictureDialog1.Execute then
    begin
        Image3.Picture.LoadFromFile(OpenPictureDialog1.FileName);
        Bitmap.Assign(Image3.Picture);
    end;
end;

```

Как видно из текста процедуры, сначала пользователь с помощью `OpenPictureDialog1` выбирает файл изображения, а затем это изображение запоминается в `Bitmap`.

Чтобы можно было в процессе редактирования отменить операцию редактирования, создайте обработчик события выбора команды Правка ► Отменить, для чего в окне Инспектор объектов выберите объект EditUndo и на странице Events дважды щелкните на пустом поле списка в событии OnClick и вставьте в текст процедуры оператор `Image3.Picture.Assign(Bitmap);`, который восстанавливает на холсте изображение, ранее сохраненное в `Bitmap`. Текст процедуры обработчика отмены операции редактирования будет выглядеть следующим образом:

```
procedure TForm1.EditUndoClick(Sender: TObject);
begin
    Image3.Picture.Assign(Bitmap);
end;
```

Чтобы перед началом работы с очередным инструментом запомнить текущий вид изображения, создайте обработчик события OnClick для кнопок SbBrush и SbColor, а в текст процедуры добавьте оператор

```
if (Sender as TSpeedButton).Down then
    Bitmap.Assign(Image3.Picture);
```

Создайте обработчик события OnMouseDown для компонентов Image3 и Image4 и отредактируйте текст процедуры обработчика следующим образом:

```

procedure TForm1.Image3MouseDown(Sender: TObject; Button:
TMouseButton;
    Shift: TShiftState; X, Y: Integer);
begin
    if (Sender=Image4) or SbColor.Down then
    {Режим установки основного и вспомогательного цветов}
        begin
            if Button=mbLeft then
                with Image1.Canvas do
                    begin {Установка основного цвета}
                        Brush.Color:=(Sender as TImage).Canvas.Pixels[X,Y];
                        FillRect(Rect(0,0,Width,Height));
                    end
                else
                    with Image2.Canvas do
                        begin {Установка вспомогательного цвета}
                            Brush.Color:=(Sender as TImage).Canvas.Pixels[X,Y];
                            FillRect(Rect(0,0,Width,Height));
                        end
                    end
                end
            end
        end
    end
end

```



```

        end;

    end

    else if SbBrush.Down then
        with Image3.Canvas do
            begin {Режим закраски указанной области холста}
                if Button=mbLeft then
                    Brush.Color:=Image1.Canvas.Brush.Color
                else Brush.Color:=Image2.Canvas.Brush.Color;
                FloodFill(X,Y,Pixels[X,Y],fsSurface);
            end;
        end;
    end;
end;

```

Эта процедура устанавливает основной и вспомогательный цвета, а также выполняет функцию инструмента графического редактора – кисти. Если левая кнопка мыши нажата на палитре цветов Image4 или если выбрана кнопка SbColor (кнопка указателя цвета), то приложение находится в режиме установки цветов. При нажатой левой кнопке мыши цвет пикселя под курсором мыши воспроизводится в окне основного цвета, а при нажатии правой кнопки – в окне вспомогательного цвета.

Если кнопка мыши нажата на холсте Image3 и при этом выбрана кнопка SbBrush, то приложение находится в режиме закраски указанной области рисунка. В этом случае в зависимости от нажатой кнопки мыши выбирается основной или вспомогательный цвет и функцией FloodFill закрашивается область, координаты внутренней точки которой указаны курсором мыши, цветом пикселя, на который указывает курсор мыши.

Создайте процедуру обработки выбора команды меню Файл ► Выход, для чего в окне Инспектор объектов выберите объект Exit, на странице Events дважды щелкните мышью на пустом поле списка в событии OnClick и вставьте в текст процедуры оператор Close;, который закрывает окно приложения. Текст процедуры будет выглядеть следующим образом:

```

procedure TForm1.ExitClick(Sender: TObject);
begin
    Close; {Закреть окно приложения}
end;

```

Измените значок приложения. Для этого выберите в меню Delphi команду Project ► Options, в окне Project Options откройте вкладку Application и щелкните мышью по кнопке Load Icon. В окне Application Icon выберите папку и файл значка (например, Paint), и щелкните мышью по кнопке Открыть (рис. 4.8).

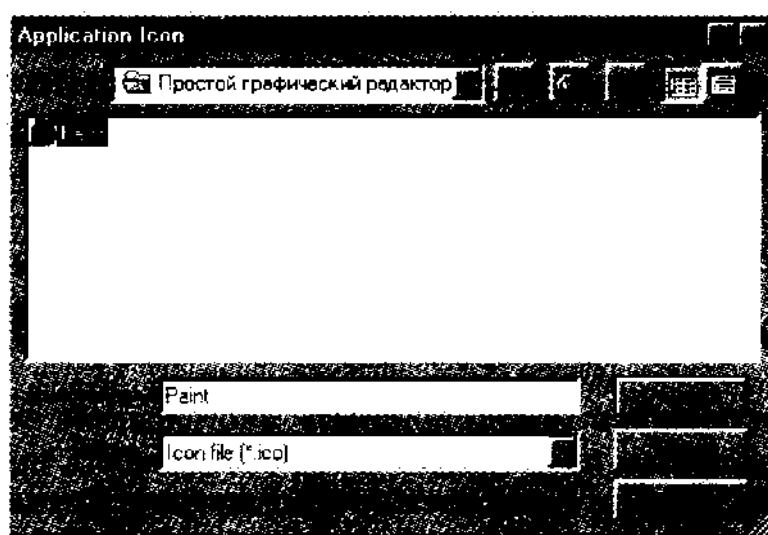
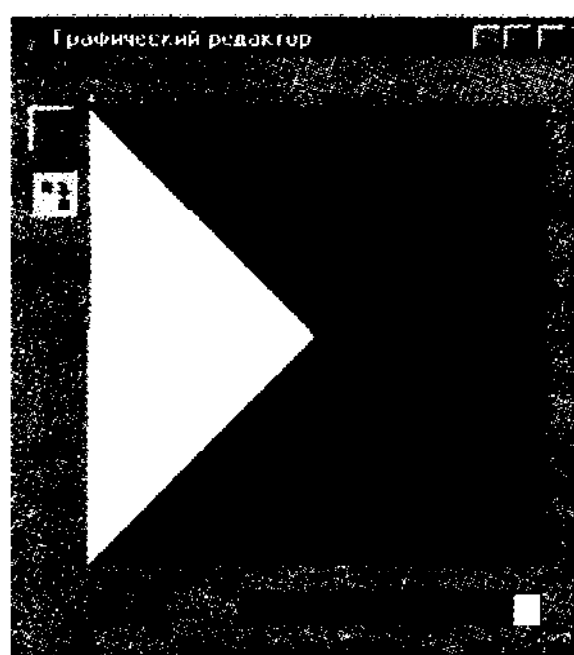


Рис. 4.8. Выбор файла значка

Щелкнув на кнопке ОК в окне Project Options, завершите изменение значка приложения. Сохраните текст приложения, откомпилируйте его и выполните. Проверьте установку цветов. При щелчке левой или правой кнопкой мыши по палитре цветов будет меняться соответственно основной или вспомогательный цвет. Если выбрать кисть, то при щелчке на холсте указанная курсором область должна закрашиваться основным или вспомогательным цветом (рис. 4.9,а). Если отпустить кнопку кисти в окне приложения и нажать кнопку определения цветов, то при щелчке мышью по холсту основной или вспомогательный цвет будет устанавливаться одинаковым с цветом холста под курсором мыши. Для загрузки изображения из файла выберите в меню редактора команду Файл ► Открыть, затем в диалоговом окне Открытие файла выберите папку и файл изображения и щелкните мышью по кнопке Открыть. На рис. 4.9,б показано окно приложения с загруженным из файла изображением.



а



б

Рис. 4.9. Окно графического редактора:

а – с тестовым рисунком; б – с изображением, загруженным из файла

Поэкспериментируйте с кистью и цветами. Если потребуется отказаться от изменений, внесенных в изображение, выберите команду Правка ▸ Отменить. При этом будет отменена последняя операция редактирования, и изображение вернется к тому виду, который оно имело до начала работы с последним инструментом.

Закройте окно приложения.

Мультипликация движения объекта

Под *мультипликацией* можно понимать многократное перерисовывание изображения. Мультфильм представляет собой совокупность множества кадров, каждый из которых немного отличается от предыдущего. Поэтому при быстром поочередном просмотре этих кадров возникает иллюзия движения. Рассмотрим создание простой мультипликации графическими средствами Delphi.

Упражнение 4.6. Создайте форму, ее свойству Caption задайте значение «Движущаяся окружность». Поместите на форму компонент Timer из палитры System. Для задания темпа смены кадров присвойте свойству Timer1.Interval значение 50 мс (0,05 с).

Для запуска анимации создайте обработчик события OnActivate, наступающего, когда форма получает или теряет фокус, например, при щелчке по ней мышью. Чтобы создать обработчик события, выберите в окне Инспектор объектов объект Form1 и на странице Events дважды щелкните мышью на пустом поле списка в событии OnActivate. После этого в окне Редактор кода будет сгенерирована заготовка процедуры обработчика события `procedure TForm1.FormActivate(Sender: TObject);`.

Добавьте в раздел описания переменных переменные целого типа `x` и `y` – координаты центра окружности и `dx` – приращение координаты `x` при движении окружности:

```
var
  Form1: TForm1;
  x, y: byte;      {Координаты центра окружности}
  dx: byte;        {Приращение координаты x при движении
окружности}
```

Отредактируйте текст процедуры следующим образом:

```
procedure TForm1.FormActivate(Sender: TObject);
begin
  x:=0;              {Начальная позиция рисунка}
  y:=30;
  dx:=3;             {Шаг изменения координаты x}
  Timer1.Interval:=50; {Период возникновения события OnTimer
- 0.5 с}
```

```
Form1.Canvas.Brush.Color:=Form1.Color; {Цвет окраски канвы}
end;
```

Создайте процедуру обработчика события OnTimer, для чего выберите в окне Инспектор объектов объект Timer1 и на странице Events дважды щелкните мышью на пустом поле списка в событии Timer1Timer. В тело процедуры добавьте вызов процедуры рисования Ris:

```
procedure TForm1.Timer1Timer(Sender: TObject);
begin
    Ris;
end;
```

Так как процедура должна быть описана до ее вызова, то выше этой процедуры в тексте программного модуля опишите процедуру рисования Ris:

```
procedure Ris; {Рисование}
begin
    Form1.Canvas.Pen.Color:=Form1.Color; {Задать перу цвет
формы}
    Form1.Canvas.Ellipse(x,y,x+20,y+20); {Стереть окружность -
нарисовать ее цветом формы}
    x:=x+dx; {Перейти в следующую позицию}
    Form1.Canvas.Pen.Color:=clBlack;{Задать перу черный цвет}
    Form1.Canvas.Ellipse(x,y,x+20,y+20); {Нарисовать окружность
на новом месте}
end;
```

Полный текст программного модуля будет выглядеть следующим образом:

```
unit main;
interface
uses
    Windows, Messages, SysUtils, Classes, Graphics, Controls,
Forms,
    Dialogs, ExtCtrls, StdCtrls;
type
    TForm1 = class(TForm)
        Timer1: TTimer;
        procedure Timer1Timer(Sender: TObject);
        procedure FormActivate(Sender: TObject);
    private
        { Private declarations }
    public
        { Public declarations }
    end;
```

```

var
  Form1: TForm1;
  x,y: byte;   {Координаты центра окружности}
  dx: byte;    {Приращение координаты x при движении
окружности}
implementation
  {$R *.DFM}
  procedure Ris;   {Рисование окружности}
  begin
    Form1.Canvas.Pen.Color:=Form1.Color; {Задать перу цвет
формы}
    Form1.Canvas.Ellipse(x,y,x+20,y+20); {Стереть окружность
- нарисовать ее цветом формы}
    x:=x+dx; {Перейти в следующую позицию}
    Form1.Canvas.Pen.Color:=clBlack;   {Задать перу черный
цвет}
    Form1.Canvas.Ellipse(x,y,x+20,y+20); {Нарисовать окружность
на новом месте}
  end;
  procedure TForm1.Timer1Timer(Sender: TObject);
  begin
    Ris; {Вызов процедуры рисования окружности}
  end;
  procedure TForm1.FormActivate(Sender: TObject);
  begin
    x:=0; {Начальная позиция рисунка}
    y:=30;
    dx:=3; {Шаг изменения координаты x}
    Timer1.Interval:=50; {Период возникновения события
OnTimer - 0.5 с}
    Form1.Canvas.Brush.Color:=Form1.Color;   {Цвет окраски
канвы}
  end;
end.

```

Сохраните текст приложения, откомпилируйте его и проверьте его работу. Убедитесь, что в окне приложения выполняется многократное перерисовывание окружности со смещением положения центра (рис. 4.10).

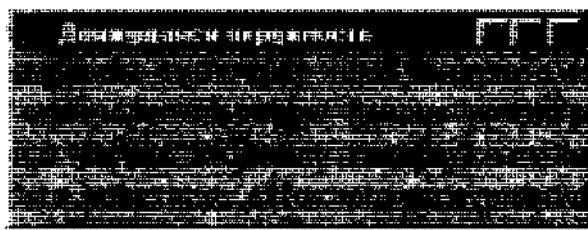


Рис. 4.10. Окно приложения с мультипликацией

Конечно, изображение окружности является очень простым и рисуется быстро, в случае сложных изображений время рисования может быть заметным и приводить к мерцанию рисунка. Чтобы избежать явления мерцания при мультипликации, используется буферизация изображения: рисование очередного кадра производится не на холсте, а на канве невидимого компонента типа *Bitmap*, а после того, как рисунок готов, он переносится на видимый холст методами *Draw* или *CopyRect*.

Упражнение 4.7. Создайте приложение-мультипликацию, в котором на фоне снежных гор летит самолет. В качестве значка приложения используйте рисунок самолета. Рисунки фона и самолета можно сделать самостоятельно или найти в Интернете (рис. 4.11).

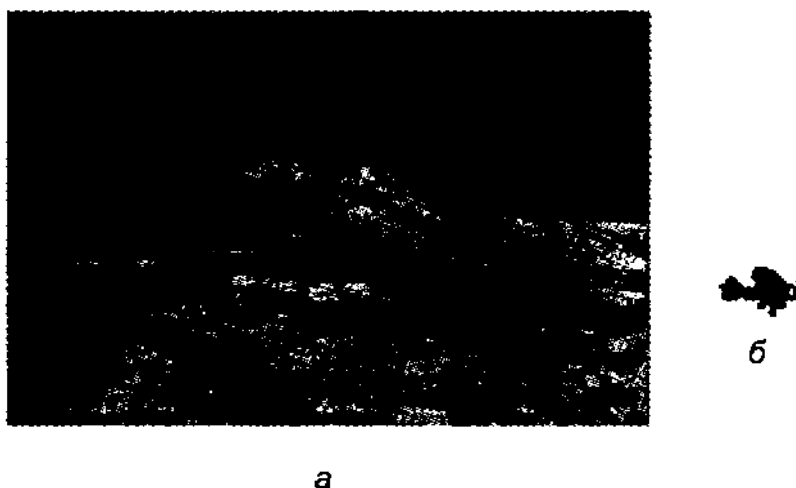


Рис. 4.11. Рисунки фона и самолета:
а – фон; б – самолет

Создайте форму, ее свойству *Caption* задайте значение «Полет самолета». Поместите на форму компонент *Timer* из палитры *System* и компонент *Image* из палитры *Additional*. Задайте свойству *Image1.Align* значение *alClient* (компонент занимает всю клиентскую область контейнера, и во время выполнения приложения его размеры изменяются при изменении размеров контейнера). Для задания темпа смены кадров установите свойству *Timer1.Interval* значение 100 мс (0,1 с). Сохраните файл проекта под именем *airplane*, файл модуля – под именем *main* в папке «Летающий самолет».

Для редактирования значка приложения запустите встроенный в Delphi графический редактор *Tools » Image Editor* и откройте файл ресурсов *airplane.res*,

затем в окне структуры файла ресурсов разверните стандартный узел Icon, щелкнув по символу + в строке Icon. Щелкнув по вершине MAINICON, соответствующей стандартному значку приложения, откройте значок для редактирования. Не закрывая окна редактирования значка приложения, откройте в окне Image Editor файл рисунка самолета, для чего выберите команду File ▶ Open, затем нужный файл (в нашем примере – airplane.bmp) и щелкните мышью по кнопке Открыть. Выделите весь рисунок самолета командой Edit ▶ Select All и скопируйте его в буфер обмена командой Edit ▶ Copy. Закройте окно с рисунком самолета и, установив в качестве текущего окно MAINICON (airplane.res), вставьте рисунок самолета из буфера обмена в окно редактирования значка командой Edit ▶ Paste. Отредактируйте полученное изображение (рис. 4.12).

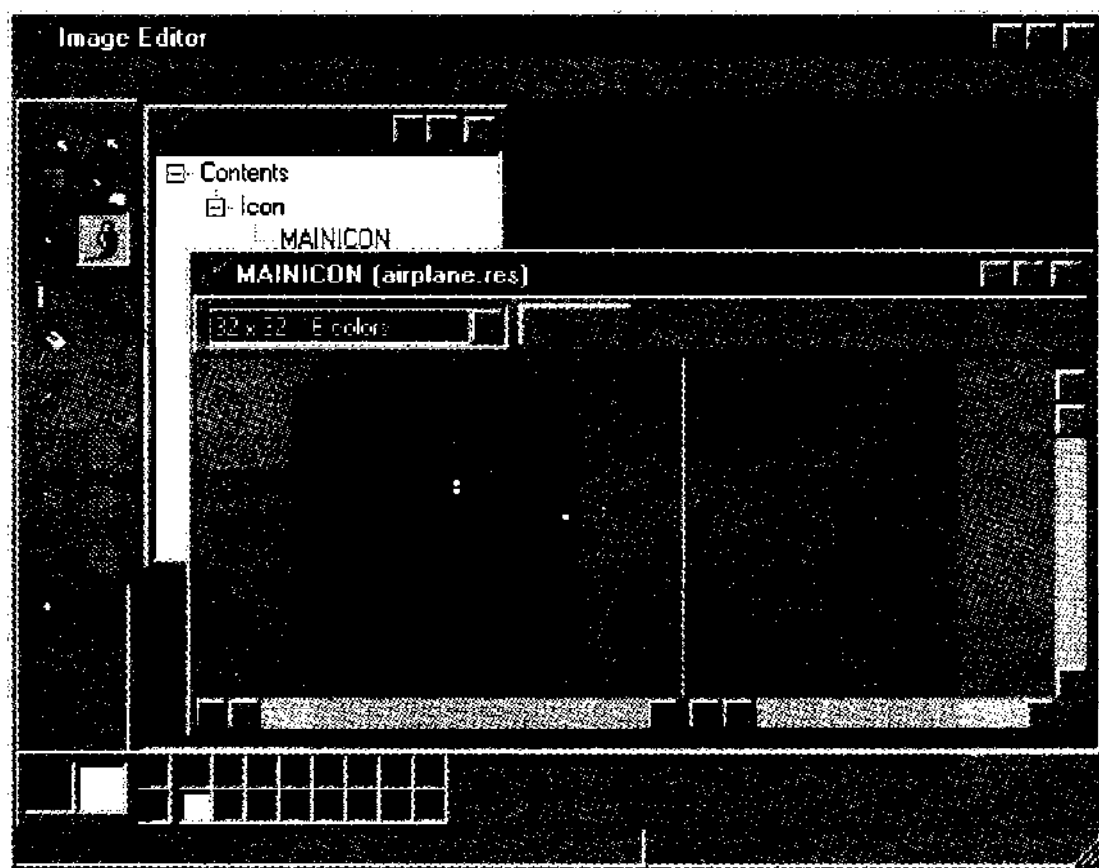


Рис. 4.12. Редактирование значка приложения в файле ресурсов

Закройте окно редактирования значка командой File ▶ Close. Сохраните изменения в файле ресурсов командой File ▶ Save. Закройте окно редактора Image Editor командой File ▶ Exit.

Добавьте в раздел переменных модуля описание переменных:

```
Back, bitmap, Buf : TBitmap; {Фон, картинка, буфер}
BackRect, bitmapRect, BufRect: TRect; {Область фона, картинки,
буфера}
x,y: integer; {Координаты левого верхнего угла картинки}
W,H: integer; {Размеры картинки (ширина, высота)}
```

Для запуска анимации создайте обработчик события OnActivate, наступающего, когда форма получает или теряет фокус, например, при щелчке по ней мышью. Чтобы создать обработчик события, выберите в окне Инспектор объектов объект Form1 и на странице Events дважды щелкните мышью на пустом поле списка в событии OnActivate. После этого в окне Редактор кода будет сгенерирована заготовка процедуры обработчика события procedure TForm1.FormActivate(Sender: TObject);.

Отредактируйте текст этой процедуры следующим образом:

```
procedure TForm1.FormActivate(Sender: TObject);
begin
    Back := TBitmap.Create;    // Фон
    bitmap := TBitmap.Create;  // Картинка
    Buf := TBitmap.Create;     // Буфер
    {Загрузить и вывести фон из файла Background.bmp}
    Back.LoadFromFile('Background.bmp');
    Form1.Image1.Canvas.Draw(0,0,Back);
    {Загрузить изображение самолета, которое будет двигаться}
    bitmap.LoadFromFile('aplane.bmp');
    bitmap.Transparent := true;
    bitmap.TransparentColor := bitmap.Canvas.Pixels[1,1];
    {Создать буфер для сохранения копии области фона, на которую
    накладывается картинка}
    W:= bitmap.Width;
    H:= bitmap.Height;
    Buf.Width:= W;
    Buf.Height:=H;
    Buf.Palette:=Back.Palette; {Обеспечить соответствие палитр}
    Buf.Canvas.CopyMode:=cmSrcCopy;
    BufRct:=Bounds(0,0,W,H);
    x:=-W;
    y:=20;
    BackRct:=Bounds(x,y,W,H);    {Определить сохраняемую область
    фона}
    Buf.Canvas.CopyRect(BufRct,Back.Canvas,BackRct); {Сохранить
    ее}
end;
```

Для получения эффекта мультипликации создайте процедуру обработчика события OnTimer, для чего выберите в окне Инспектор объектов объект Timer1 и на странице Events дважды щелкните мышью на пустом поле списка в событии Timer1Timer. Отредактируйте тело процедуры обработчика события следующим образом:

```
procedure TForm1.Timer1Timer(Sender: TObject);
begin
    Form1.Image1.Canvas.Draw(x,y,Buf);    {Восстановлением фона
    (из буфера) - удалить рисунок}
```



```

    x:=x+2; {Смещение самолета по оси x}
    if x>Form1.Image1.Width then x:=-W; {Если самолет «вылетел»
за пределы рисунка фона}
    BackRct:=Bounds(x,y,W,H); {Определить сохраняемую область
фона}
    Buf.Canvas.CopyRect(BufRct,Back.Canvas,BackRct); {Сохранить
ее копию}
    Form1.Image1.Canvas.Draw(x,y,bitmap);    Вывести    рисунок
самолета в новой позиции}
end;

```

Для освобождения памяти от переменных, в которых хранились значения фона, рисунка самолета и буфера с фрагментом фона, создайте процедуру обработчика события закрытия окна приложения. Для этого выберите в окне Инспектор объектов объект Form1 и на странице Events дважды щелкните мышью на пустом поле списка в событии OnClose. Вставьте в тело процедуры обработчика события вызов метода Free для освобождения памяти, динамически выделенной под объекты Back, bitmap и Buf:

```

procedure TForm1.FormClose(Sender: TObject; var Action:
TCloseAction);
begin
    Back.Free; {Освободить память}
    bitmap.Free;
    Buf.Free;
end;

```

Сохраните текст приложения, откомпилируйте его и проверьте его работу. Убедитесь, что в окне приложения на фоне выбранного рисунка перемещается изображение самолета (рис. 4.13).

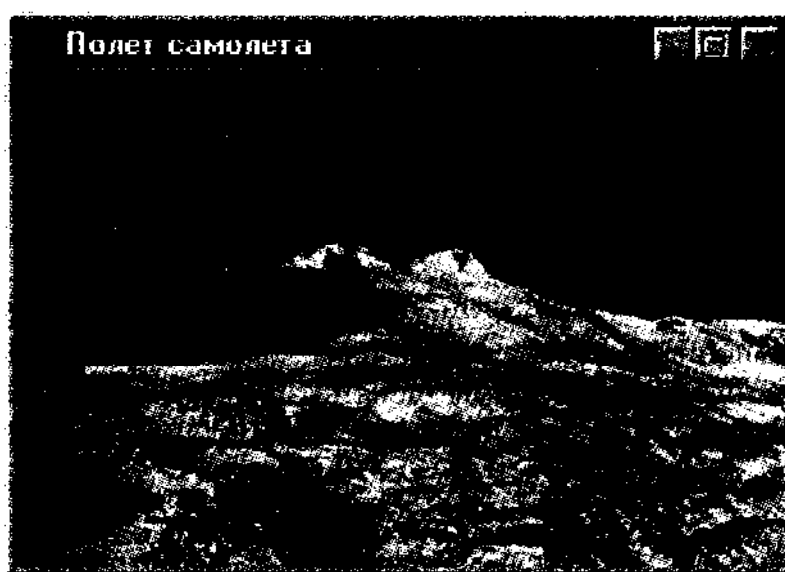


Рис. 4.13. Окно приложения с самолетом, летящим на фоне изображения

Воспроизведение звука и видеоклипов

Наиболее простым аудиофайлом является волновой файл формата .wav. Волновые и MIDI-файлы могут хранить только звук или музыку. Для хранения видеoinформации разработан ряд других форматов. Отметим среди них файлы AVI и MPEG. Большинство видеофайлов поддерживает также хранение звуковой дорожки, так что звук воспроизводится синхронно с изображением. В основу хранения видеoinформации заложено представление о том, что человеческий мозг интерпретирует быструю последовательность изображений, незначительно отличающихся друг от друга, как движение. Каждое из таких изображений называется *кадром*. Каждый следующий кадр несколько отличается от предыдущего. Чтобы мозг воспринимал смену кадров как плавное движение, желательно воспроизводить не менее 24 кадров в секунду. Более высокая частота не приводит к заметному росту качества, а при более низкой создается впечатление мерцания экрана. Если бы каждый кадр файла хранился в виде растрового изображения (а это несколько сотен килобайт), то для хранения всего файла потребовался бы огромный объем дисковой памяти. При такой простой схеме хранения на компакт-диск, например, можно было бы записать всего 72 с видеofilmа. Поэтому на практике для хранения видеofilmов используется сжатие видеоданных.

Процедуры воспроизведения звуков. Наиболее простой процедурой, управляющей звуком, является процедура Beep. Она не имеет параметров и воспроизводит стандартный звуковой сигнал, установленный в Windows, если компьютер имеет звуковую карту и задан стандартный сигнал (он устанавливается в Панели управления Windows после щелчка мышью по значку Звук). Если звуковой карты нет или стандартный сигнал не установлен, звук воспроизводится через динамик компьютера в виде короткого щелчка.

Более серьезной функцией является MessageBeep. Она определена как

```
Function MessageBeep (uType:word) :boolean;
```

Параметр uType определяет воспроизводимый звук как идентификатор раздела [sounds] реестра, в котором записаны звуки, сопровождающие те или иные события в Windows. С помощью приложения Звук в Панели управления Windows пользователь может удалить или установить соответствующие звуки. Для параметра uType определены следующие константы.

Значение	Звук
MB_ICONASTERISK	Звездочка
MB_ICONEXCLAMATION	Восклицание
MB_ICONHAND	Критическая ошибка
MB_ICONQUESTION	Вопрос
MB_OK	Стандартный звук

После инициализации воспроизведения звука функция MessageBeep возвращает управление в точку вызова, и звук воспроизводится асинхронно. Если функция MessageBeep не находит указанный тип звука, она пытается воспроиз-

вести стандартный звук. Если и он не установлен или если компьютер не снабжен звуковой картой, то звук воспроизводится через динамик компьютера.

При успешном завершении функция возвращает ненулевое значение (True). Если функция вернула нулевое значение, то получить информацию об ошибке можно с помощью вызова GetLastError.

Упражнение 4.8. Создайте приложение, которое позволит прослушать стандартные звуки Windows. Для выбора воспроизводимого звука используйте переключатели, размещенные в панели RadioGroup.

Создайте форму и задайте ее свойству Caption значение «Тестирование звука». Поместите на форму компоненты Button и RadioGroup (см. рис. 4.8). Задайте свойству Caption кнопки Button значение «Послушать». Задайте свойству Caption компонента RadioGroup1 значение «Вариант звучания». Так как количество переключателей в группе и подписей к ним определяется свойством Items, выберите в окне Инспектор объектов компонент RadioGroup1 и на странице Events – свойство Items (список элементов). В окне String List Editor введите список элементов: Веер, Звездочка, Восклицание, Критическая ошибка, Вопрос, Стандартный звук. Щелкнув мышью по кнопке ОК, завершите формирование списка вариантов прослушиваемых звуков Windows. Выровняйте и зафиксируйте компоненты на форме. После этого форма с размещенными на ней компонентами должна выглядеть так, как показано на рис. 4.14.

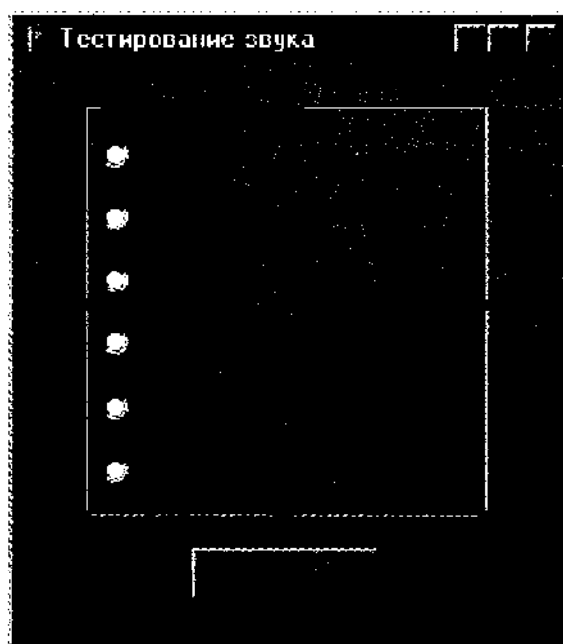


Рис. 4.14. Форма приложения для прослушивания стандартных звуков Windows

Для создания процедуры обработки события – щелчка мышью по кнопке Послушать – в окне Инспектор объектов выберите объект Button1, затем на странице Events дважды щелкните мышью на пустом поле списка в событии OnClick. После этого в окне Редактор кода будет сгенерирована заготовка процедуры обработчика события `procedure TForm1.Button1Click(Sender: TObject);`.

Так как в процедуре обработки щелчка на кнопке Button1 должно быть шесть вариантов реализации для разных звуков, то рационально записать выбор варианта звука с помощью оператора Case. Текст процедуры обработчика события может быть записан следующим образом:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Case RadioGroup1.ItemIndex of      {Выбор варианта звука}
    0 : Beep;                        {Стандартный звуковой сигнал}
    1 : MessageBeep(MB_ICONASTERISK); {Звездочка}
    2 : MessageBeep(MB_ICONEXCLAMATION); {Восклицание}
    3 : MessageBeep(MB_ICONHAND); {Критическая ошибка}
    4 : MessageBeep(MB_ICONQUESTION); {Вопрос}
    else MessageBeep(MB_OK);        {Стандартный звук Windows}
  end;
end;
```

Сохраните файлы проекта и модуля в папке «Стандартные звуки», а затем откомпилируйте и запустите приложение.

Примечание. Для успешной демонстрации работы приложения необходимо, чтобы на компьютере были установлены звуки, соответствующие различным типам сигналов Windows. Для их установки выберите команду Пуск ► Настройка ► Панель управления, а затем, дважды щелкнув мышью по значку Звук, откройте диалоговое окно Свойства: Звук.

В окне приложения Тестирование звука, выбирая варианты звучания с помощью переключателей и щелкая мышью по кнопке Послушать, убедитесь в правильной работе приложения.

Функция PlaySound. Функция PlaySound воспроизводит указанный волновой файл, или звук системного события, или звук из ресурса. Она определена следующим образом:

```
function PlaySound(pszSound:PChar; hmod:HINST;
                  fdwSound:Cardinal) :boolean;
```

Параметр pszSound представляет собой строку с нулевым символом в конце и определяет воспроизводимый звук. В зависимости от значения флага fdwSound (SND_FILENAME, SND_ALIAS или SND_RESOURCE) параметр pszSound может определять имя волнового файла, псевдоним системного события или идентификатор ресурса. Если ни один из этих флагов не указан, функция ищет в реестре Windows или в файле WIN.INI указанное имя звука. Если звук найден, то он воспроизводится. Если звук не найден, то параметр pszSound интерпретируется как имя файла.

Если параметр pszSound равен 0, то воспроизведение любого волнового файла прерывается. Для прерывания воспроизведения звука, не связанного с волновым файлом, следует указать SND_PURGE в параметре fdwSound.

Параметр hmod используется только при параметре fdwSound, равном SND_RESOURCE. В этом случае hmod является дескриптором исполняемого файла, содержащего ресурс, который должен загружаться. В противном случае значение hmod задается равным 0.

Параметр `fdwSound` задает флаги воспроизведения звука. Флаги могут комбинироваться друг с другом при помощи операции `or`. Наиболее важные для воспроизведения значения параметра `fdwSound` перечислены далее.

Параметр	Назначение
<code>SND_ASYNC</code>	Звук воспроизводится асинхронно, и функция <code>PlaySound</code> возвращает управление немедленно после начала воспроизведения. Чтобы прекратить асинхронное воспроизведение волнового файла, следует вызвать <code>PlaySound</code> с параметром <code>pszSound</code> , равным 0.
<code>SND_FILENAME</code>	Параметр <code>pszSound</code> является именем файла.
<code>SND_LOOP</code>	Воспроизведение звука постоянно повторяется, пока не будет вызвана функция <code>PlaySound</code> с параметром <code>pszSound</code> , равным 0. Одновременно должен быть указан флаг <code>SND_ASYNC</code> асинхронного воспроизведения звука.
<code>SND_NOSTOP</code>	Если заданный звук не может быть воспроизведен, поскольку ресурсы, необходимые для воспроизведения, заняты воспроизведением другого звука, функция <code>PlaySound</code> немедленно возвращает <code>False</code> , не воспроизводя заданный звук. Если этот флаг не указан, функция <code>PlaySound</code> пытается остановить воспроизведение другого звука, чтобы устройство могло быть использовано для воспроизведения нового звука.
<code>SND_NOWAIT</code>	Если драйвер занят, функция немедленно возвращает управление без воспроизведения заданного звука.
<code>SND_PURGE</code>	Останавливает воспроизведение любых звуков, вызванных в данной задаче. Если <code>pszSound</code> не равен 0, прекращается воспроизведение всех экземпляров указанного звука. Если <code>pszSound</code> равен 0, то прекращается воспроизведение всех звуков, связанных с данной задачей. Отдельно следует указать дескриптор для остановки событий <code>SND_RESOURCE</code> .
<code>SND_RESOURCE</code>	Параметр <code>pszSound</code> является идентификатором ресурса. Параметр <code>hmod</code> должен указывать на источник ресурса.
<code>SND_SYNC</code>	Синхронное воспроизведение звука события. Функция <code>PlaySound</code> возвращает управление только после окончания воспроизведения.

Примечание. Функция `PlaySound` – функция API Windows, параметры которой описаны в модуле `mmsystem`, поэтому для использования этой функции необходимо включить в раздел `uses` ссылку на `mmsystem`.

Звук, заданный параметром `pszSound`, должен помещаться в доступную память и подходить для установленного драйвера устройства воспроизведения волновых файлов. Функция `PlaySound` ищет файл звука в следующих каталогах: текущем, каталоге `Windows`, системном каталоге `Windows`, каталогах, перечисленных в переменной среды `PATH`, в списке каталогов, предоставляемых сетью.

Если указанный звук не найден, то функция `PlaySound` воспроизводит системный звук по умолчанию. Если функция не может найти и его, то звук не воспроизводится, а функция возвращает значение `False`.

Упражнение 4.9. Создайте приложение, которое позволит прослушивать музыкальные файлы, открывая их с помощью `OpenDialog`.

Создайте форму и задайте ее свойству `Caption` значение «Прослушивание звуковых файлов». Поместите в любое место формы компонент `OpenDialog` из палитры `Dialogs` и задайте для свойства `OpenDialog1.Title` значение «Открыть звуковой файл», которое будет отображаться в заголовке диалогового окна открытия файла. Задайте свойству `OpenDialog1.Filter` значение «звуковые файлы (*.wav)|*.wav». Поместите на форму компонент `Button` и задайте его свойству `Caption` значение «Прослушать». Для создания обработчика щелчка мышью по кнопке `Button1` в окне Инспектор объектов выберите объект `Button1`, затем на странице `Events` дважды щелкните мышью на пустом поле списка в событии `OnClick`. После этого в окне Редактор кода будет сгенерирована заготовка процедуры обработчика события

```
procedure TForm1.Button1Click(Sender: TObject);.
```

Так как в процедуре обработки нажатия кнопки `Button1` используется вызов функции `PlaySound`, то следует включить в раздел описания `Uses` вызов модуля `mmSystem`. Использование `OpenDialog1` для открытия звукового файла реализуется при помощи оператора

```
if OpenDialog1.Execute then  
    PlaySound(PChar(OpenDialog1.FileName), 0, SND_ASYNC);
```

где параметр `SND_ASYNC` указывает на асинхронное воспроизведение звука.

Сохраните файлы проекта и модуля в папке «Прослушивание звуковых файлов». Откомпилируйте и запустите приложение. Щелчком мышью по кнопке Прослушать в окне приложения откройте диалоговое окно Открыть звуковой файл, в котором выберите звуковой файл, например, в папке `C:\Windows\Media` (рис. 4.15).

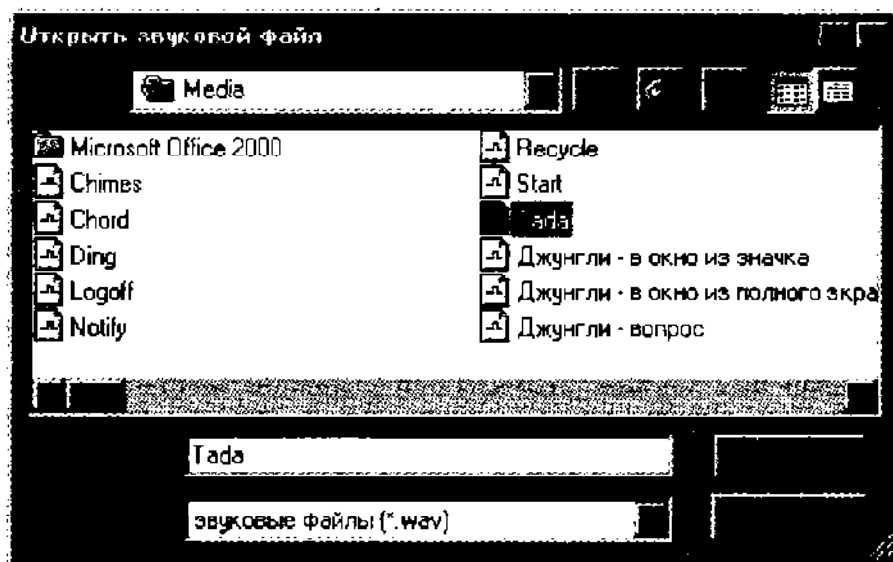


Рис. 4.15. Диалоговое окно открытия звукового файла

Прослушав звук, закройте окно приложения. Функция `PlaySound` позволяет воспроизводить и системные звуки, просто называя их псевдонимы. Например, оператор

```
PlaySound('SystemStart', 0, SND_ASYNC);
```

воспроизведет тот же звук, что и приведенный ранее оператор, указывавший имя файла и путь к нему. Оператор

```
PlaySound('C:\Windows\Media\Звук Microsoft.wav', 0, SND_ASYNC  
or SND_LOOP);
```

многократно асинхронно воспроизводит стандартный звук Microsoft, начиная воспроизведение снова и снова, как только оно заканчивается.

Примечание. Если вы ввели в свое приложение подобный оператор, то следует предусмотреть какую-нибудь кнопку, щелчок по которой прерывает воспроизведение заданием нового звука или выполнением оператора `PlaySound(0,0, SND_PURGE);`.

Компонент Animate. Для воспроизведения стандартных видеоклипов Windows (типа копирования файлов, поиска файлов и т. п.) и не сопровождаемых звуком видеофайлов формата *.avi (Audio Video Interleaved) используется компонент типа TAnimate. Эти файлы представляют собой последовательность кадров битовых матриц. Они могут содержать и звуковую дорожку, но компонент Animate способен воспроизводить только немые клипы AVI. Он работает только с файлами AVI, не использующими сжатие, или с клипами AVI, сжатыми с использованием технологии RLE (run-length encoding).

Во время проектирования в компонент Animate можно загрузить кадры клипа AVI, указав такие свойства, как FileName – имя файла или CommonAVI – стандартный клип Windows. Можно также просматривать клип по кадрам, щелкая по компоненту правой кнопкой мыши и выбирая разделы раскрывающегося меню Next Frame (следующий кадр) или Previous Frame (предыдущий кадр). Это позволяет выбирать фрагмент клипа и воспроизводить его методом Play.

В компоненте Animate предусмотрены события OnClose, OnOpen, OnStart и OnStop, генерируемые соответственно в моменты закрытия и открытия компонента, начала и окончания воспроизведения.

Упражнение 4.10. Создайте приложение, которое позволит проигрывать стандартные клипы Windows и видеоклипы формата *.avi, открывая их с помощью OpenFileDialog.

Создайте форму и задайте ее свойству Caption значение «Воспроизведение видеоклипов без звука». Поместите в любое место формы компонент OpenFileDialog из палитры Dialogs и задайте свойству OpenFileDialog1.Title значение «Открыть файл видеоклипа», которое будет отображаться в заголовке диалогового окна открытия файла. Задайте свойству OpenFileDialog1.Filter значение «Видеоклип (*.avi)|*.avi». Поместите на форму компоненты Button1, Button2 и Button3. Задайте для них имена BtWind, BtStop и BtFile соответственно. Задайте для свойств Caption этих кнопок значения «Клипы Windows», «Стоп» и «Клип из файла». Поместите на форму компонент Animate1 из палитры Win32 и задайте для свойства Visible значение False. Над компонентом Animate1 поместите компонент Label1 и задайте его свойству Caption пустое значение. Выровняйте и зафиксируйте компоненты на форме (рис. 4.16).



Рис. 4.16. Форма приложения для просмотра видеоклипов без звука

Создайте обработчик щелчка мышью по кнопке BtWind, обеспечивающий просмотр стандартных видеоклипов Windows. Для этого в окне Инспектор объектов выберите объект BtWind, затем на странице Events дважды щелкните мышью на пустом поле списка в событии OnClick. После этого в окне Редактор кода будет сгенерирована заготовка процедуры обработчика события `procedure TForm1.BtWindClick(Sender: TObject);`. Добавьте в раздел описания переменных модуля описание переменной `I` целого типа, она будет хранить значение номера стандартного видеоклипа Windows:

```
var
  Form1: TForm1;
  I : byte;
```

Отредактируйте текст процедуры `TForm1.BtWindClick` следующим образом:

```
procedure TForm1.BtWindClick(Sender: TObject);
begin
  Animat1.Visible:=True; {Сделать видимой область анимации}
  I:=1;                  {Указывает на первый клип Windows}
  Animat1.CommonAVI:=aviFindFolder; {Воспроизвести первый
стандартный клип - поиск в папке}
  Label1.Caption:='Клип Windows: поиск в папке';
  Animat1.Active:=True;  {Активизировать анимацию}
end;
```

Создайте процедуру обработчика щелчка мышью по кнопке BtStop и отредактируйте ее текст следующим образом:

```
procedure TForm1.BtStopClick(Sender: TObject);
begin
  Animat1.Stop; { Вызов процедуры смены стандартного видеоклипа
Windows}
end;
```


Процедуру смены стандартного видеоклипа Windows реализуйте в виде процедуры обработчика события окончания воспроизведения видеоклипа OnStop. Для этого в окне Инспектор объектов выберите компонент Animated и на странице Events дважды щелкните мышью на пустом поле списка в событии OnStop. После этого в окне Редактор кода будет сгенерирована заготовка процедуры обработчика события `procedure TForm1.AnimatedStop(Sender: TObject);`. Отредактируйте ее текст следующим образом:

```
procedure TForm1.AnimatedStop(Sender: TObject);
begin
  Inc(I); {Увеличить счетчик клипов}
  with Animated do begin
    case I of
      2: begin
          CommonAVI:=aviFindFile;
          Label1.Caption:='Клип Windows: поиск файла';
        end;
      3: begin
          CommonAVI:=aviFindComputer;
          Label1.Caption:='Клип Windows: поиск компьютера';
        end;
      4: begin
          CommonAVI:=aviCopyFiles;
          Label1.Caption:='Клип Windows: копирование
файлов';
        end;
      5: begin
          CommonAVI:=aviCopyFile;
          Label1.Caption:='Клип Windows: копирование файла';
        end;
      6: begin
          CommonAVI:=aviRecycleFile;
          Label1.Caption:='Клип Windows: удаление файла в
корзину';
        end;
      7: begin
          CommonAVI:=aviEmptyRecycle;
          Label1.Caption:='Клип Windows: очистка корзины';
        end;
```

```

      8: begin
          CommonAVI:=aviDeleteFile;
          Label1.Caption:='Клип Windows: удаление файла';
      end;
  end;
  if I<9 then Active:=True
  else
      begin
          Visible:=False;
          Label1.Caption:=''; {Очистить после показа
клипов}
      end;
  end;
end;
end;

```

Как видно из текста процедуры, выбор вида стандартного клипа описан с помощью оператора выбора case, причем значение селектора I изменяется от 2 до 8, так как для I=1 воспроизведение видеоклипа выполняется в процедуре обработчика щелчка мышью по кнопке BtWind. Каждый оператор, выполняемый для конкретного значения селектора I, является составным, например:

```

begin
    CommonAVI:=aviFindFile;
    Label1.Caption:='Клип Windows: поиск файла';
end;

```

где CommonAVI:=aviFindFile; определяет конкретный стандартный видеоклип Windows, а затем Label1.Caption:='Клип Windows: поиск файла'; выводит в окне приложения сообщение о том, какой клип воспроизводится.

Для воспроизведения видеоклипов из файлов формата *.avi создайте процедуру обработчика щелчка мышью по кнопке BtFile. Отредактируйте текст этой процедуры следующим образом:

```

procedure TForm1.BtFileClick(Sender: TObject);
var
    FName : string; {Переменная для хранения части имени
файла}
begin
    if OpenDialog1.Execute then
        with Animat1 do
            begin
                I:=9;

```

```

    FileName:=OpenDialog1.FileName;
    FName:=FileName;
    repeat {Удалить часть имени до '\'}
        Delete(FName,1,Pos('\',FName));
    until Pos('\',FName)=0;
    Visible:=True;
    Label1.Caption:='Клип из файла '+FName;
    Active:=True;
end;
end;

```

Как видно из текста процедуры, в разделе описания переменных введена переменная FName, в которую копируется значение из FileName, с тем, чтобы впоследствии «отрезать» от полного имени файла имя папки, в которой он находится. Для открытия файла видеоклипа используется стандартный диалог OpenFileDialog. Фрагмент программы

```

    repeat {удалить часть имени до '\'}
        Delete(FName,1,Pos('\',FName));
    until Pos('\',FName)=0;

```

удаляет из полного имени файла имя папки, в которой он находится.

Сохраните файлы проекта и модуля в папке «Воспроизведение немых видеоклипов». Откомпилируйте и запустите приложение. Щелкнув мышью по кнопке Клипы Windows в окне приложения, просмотрите воспроизведение стандартного видеоклипа. Для завершения просмотра текущего стандартного клипа Windows и перехода к следующему используйте щелчок мышью по кнопке Стоп. Просмотрите таким образом все клипы Windows. Щелкнув мышью по кнопке Клип из файла, в диалоговом окне Открыть файл видеоклипа выберите файл видеоклипа и щелкните мышью по кнопке Открыть для просмотра видеоклипа (рис. 4.17).

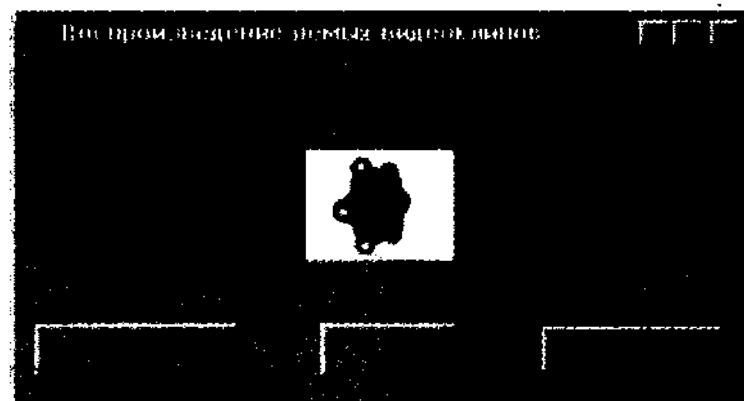


Рис. 4.17. Окно приложения с видеоклипом

Для остановки проигрывания видеоклипа щелкните мышью по кнопке Стоп. Закройте окно приложения.

Универсальный проигрыватель аудио- и видеоинформации MediaPlayer. Для проигрывания аудио- и видеоинформации в Delphi (начиная с Delphi 2) имеется универсальный проигрыватель аудио- и видеофайлов MediaPlayer. Этот компонент по умолчанию входит в состав палитры System и инкапсулирует интерфейс управления носителями (Media Control Interface, MCI) Windows 95 и Windows NT. MediaPlayer имеет ряд кнопок, управляющих такими устройствами мультимедиа, как CD-ROM и другие (ими можно также управлять мышью). Общий вид компонента представлен на рис. 4.18.



Рис. 4.18. Вид компонента MediaPlayer

Ниже представлено назначение кнопок MediaPlayer (слева направо).

Кнопка	Значение	Действие
Play	btPlay	Воспроизведение
Pause	btPause	Пауза воспроизведения или записи (если проигрыватель в момент щелчка уже находится в состоянии паузы, то воспроизведение или запись возобновляется)
Stop	btStop	Останов воспроизведения или записи
Next	btNext	Переход на следующий трек или на конец записи
Prev	btPrev	Переход на предыдущий трек или на начало записи
Step	btStep	Перемещение вперед на заданное число кадров
Back	btBack	Перемещение назад на заданное число кадров
Record	btRecord	Начало записи
Eject	btEject	Освобождение объекта, загруженного в устройство

Проигрыватель может управляться как кнопками, так и непосредственно при помощи соответствующих этим кнопкам методов (Play, Pause, Stop, Next, Previous, Step, Back, StartRecording, Eject). В этом случае сам компонент TMediaPlayer может быть сделан невидимым.

Тип устройства мультимедиа (dtWaveAudio или dtVideodisc) определяется свойством DeviceType. Если устройство хранит объект воспроизведения в файле, то имя файла задается свойством FileName. Если свойство DeviceType задано равным dtAutoSelect, то проигрыватель пытается определить тип устройства, исходя из расширения имени файла FileName. Если для свойства AutoOpen установлено значение True, то проигрыватель пытается открыть устройство, указанное свойством DeviceType, автоматически во время своего создания в процессе выполнения приложения.

Что нового вы узнали?

В этой главе вы познакомились со следующими понятиями:

- мультимедиа;
- канва, пиксель;
- мультипликация.

Изучив эту главу, вы узнали:

- назначение, возможности, свойства и методы класса TCanvas;
- назначение, свойства и методы компонентов Image, SpeedButton, OpenPictureDialog, MainMenu, Timer, Animate и MediaPlayer;
- порядок использования свойств Canvas.Pixels, Canvas.Pen, Canvas.Brush;
- назначение и порядок использования методов MoveTo, LineTo, TextOut, Arc, Chord, Ellipse, Pie, PolyBezier, PolyBezierTo, Polygon, Polyline, Rectangle, RoundRect, FillRect, FloodFill, FrameRect, TextRect;
- назначение и формат записи процедур и функций воспроизведения звука.

Вы научились:

- создавать приложения с программированием изменений свойств канвы;
- использовать конструктор меню для создания меню Windows-программы;
- создавать приложения с программированием воспроизведения звука и видеоклипов.

Контрольные вопросы и задания

Вопросы

1. Что такое мультимедиа?
2. Каково назначение свойства Canvas?
3. В чем разница между рисованием на канве по пикселям и пером?
4. Каково назначение методов MoveTo, LineTo, TextOut?
5. Как задать координаты и цвет пикселя канвы?
6. Как задаются толщина, цвет пера и стиль линии, которую оно рисует?
7. Каково назначение свойства канвы Brush (Кисть)? Какие подсвойства имеются у кисти?
8. Опишите назначение методов FillRect, FloodFill, FrameRect.
9. Каким образом можно задать изображение на кнопке?
10. Опишите назначение компонента MainMenu и порядок использования конструктора меню.
11. Что такое анимация? Какими способами можно создать в приложении Delphi анимированное изображение?

12. В чем различие звуковых файлов форматов *.wav и *.midi?
13. Сравните возможности функций Beep, MessageBeep и PlaySound.
14. Сравните назначение и возможности компонентов Animate и MediaPlayer.

Задания

1. Измените приложение, созданное при выполнении упражнения 4.1, таким образом, чтобы выполнялось построение графика функции $y = 2\sin(x) \exp(x/4)$.
2. Создайте приложение, которое при щелчке на кнопке Нарисовать рисует в окне приложения цветок ромашки.
3. Создайте приложение, в окне которого отображается анимация – движение мяча, брошенного под углом к горизонту.
4. Создайте приложение-мультипликацию, в котором на фоне дороги перемещается автомобиль (вариант: на фоне звездного неба перемещается космический корабль).
5. Создайте приложение, которое позволит прослушивать звуковые файлы формата *.wav, открывая их с помощью диалогового окна Открыть звуковой файл, вызываемого командой Файл ► Открыть. Для завершения работы приложения используйте команду Выход в главном меню.

ГЛАВА 5

Создание простых приложений

В этой главе мы рассмотрим технологию создания простых приложений в интегрированной среде программирования Delphi.

После изучения главы вы будете знать:

- Программирование взаимодействия Windows-приложения с внешними программами: непосредственный запуск внешней программы из приложения; запуск внешней программы, связанной с некоторым документом; обмен сообщениями с другими программами; внедрение и связывание документов, подготовленных внешними программами, с вашим приложением; динамический обмен данными между приложениями.
- Назначение и порядок использования функций CharToOem, WinExec, ShellExecute, GetSystemDirectory, GetWindowsDirectory, CreateProcess.
- Назначение, свойства и методы компонентов RichEdit, FontDialog, FindDialog, ReplaceDialog, ActionList, DataSource, Table, DBNavigator.
- Порядок разработки справочной системы Windows-приложения и интерфейс программы Help & Manual, используемой для создания справочных систем и руководств пользователя.
- Порядок создания базы данных и программирование обслуживания реляционной базы данных.

Консольное приложение

Наряду с приложениями, предоставляющими пользователю стандартный оконный интерфейс, Delphi позволяет создавать консольные приложения – приложения для DOS, в которых не используются средства графического интерфейса пользователя.

Упражнение 5.1. Создайте консольное приложение, которое предлагает ввести два целых числа, затем вычисляет их сумму и выводит на экран результат вычисления. Алгоритм решения этой задачи был рассмотрен в главе 1 книги "Turbo Pascal для школьников"¹.

Создать консольное приложение можно несколькими способами: удалив форму из проекта и работая непосредственно с головным файлом программы:

¹ См.: Попов В.Б. Turbo Pascal для школьников. – М.: Финансы и статистика; ИНФРА-М, 2010.

- выберите команду Project ► Options... и в окне Project Options на странице Linker установите опцию Generate Console Application;
- или включите в программу директиву компилятора {\$APPTYPE CONSOLE};
- выбрав команду File ► New ► Other, а затем в окне New Items на вкладке New выбрав значок Console Application (рис. 5.1) и щелкнув мышью по кнопке OK.

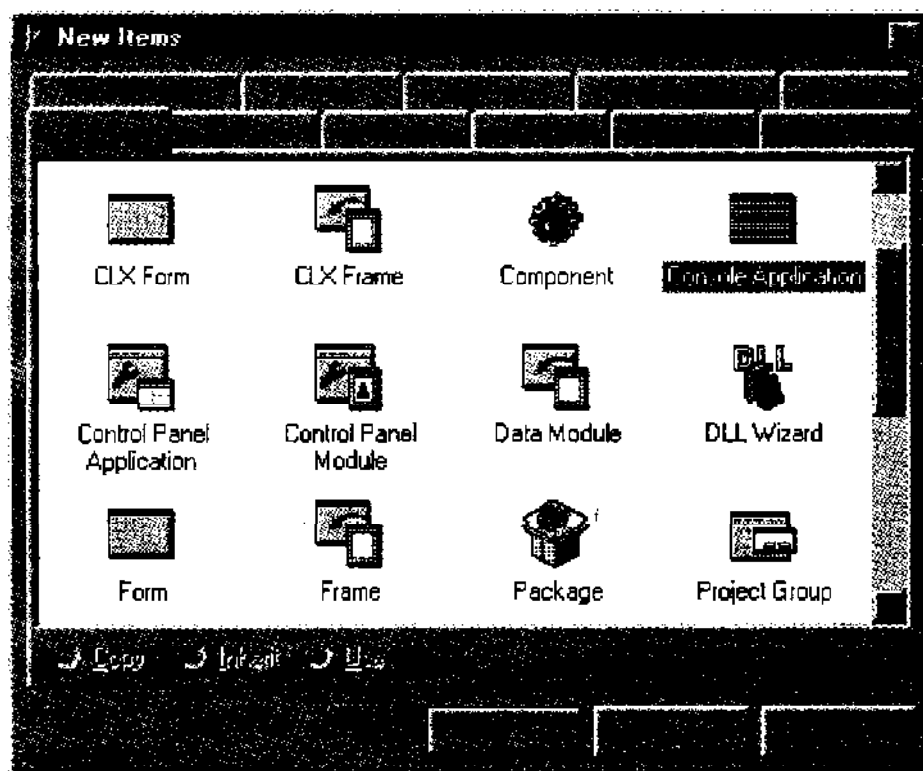


Рис. 5.1. Создание консольного приложения

После этого в окне интегрированной среды Delphi будет открыто окно проекта с шаблоном программы:

```
program Project1;  
{ $APPTYPE CONSOLE }  
uses  
    SysUtils;  
begin  
    { TODO -oUser -cConsole Main : Insert code here }  
end.
```

Как видно из текста программы, в него уже вставлена директива компилятора {\$APPTYPE CONSOLE}, указывающая на генерацию консольного приложения.

Примечание. Для правильного отображения русских шрифтов следует использовать функцию CharToOem из модуля Windows, которая преобразует текст из кодировки Windows 1251 (ANSI) в кодировку 866 (OEM).

Чтобы использовать функцию CharToOem, вставьте в раздел описания Uses модуль Windows. В раздел описания переменных поместите объявления целых переменных A, B и Summ, которые будут хранить значения слагаемых и суммы, а также переменной типа PChar – указатель на массив элементов типа Char (строка текста может быть представлена как массив символов):

```
program Project1;
    {$APPTYPE CONSOLE}      {Директива компилятора – создать
консольное приложение}
    uses
        SysUtils, Windows;
    var A,B, Summ : integer; {Слагаемые и сумма}
        TmpStr : PChar;     {Указатель на строку символов}
begin
    TmpStr := '';           // Инициализация TmpStr
    {Преобразование из кодировки 1251 (ANSI)
    в кодировку 866 (OEM)}
    CharToOem('Введите значения двух целых чисел', TmpStr);
    Write(TmpStr);          {Вывод сообщения в кодировке OEM}
    Readln(A,B); {Считывание с клавиатуры значений слагаемых}
    Summ:=A+B;              {Вычисление суммы}
    Writeln(Summ);          {Вывод результата вычислений}
    Readln; {Останов программы до нажатия Enter, чтобы пользователь
увидел результат}
end.
```

Сохраните проект в папке «Консольное приложение» под именем Summa. Откомпилируйте и запустите приложение на выполнение командой Run ▶ Run. Введите значения слагаемых и нажмите клавишу Enter (рис. 5.2).

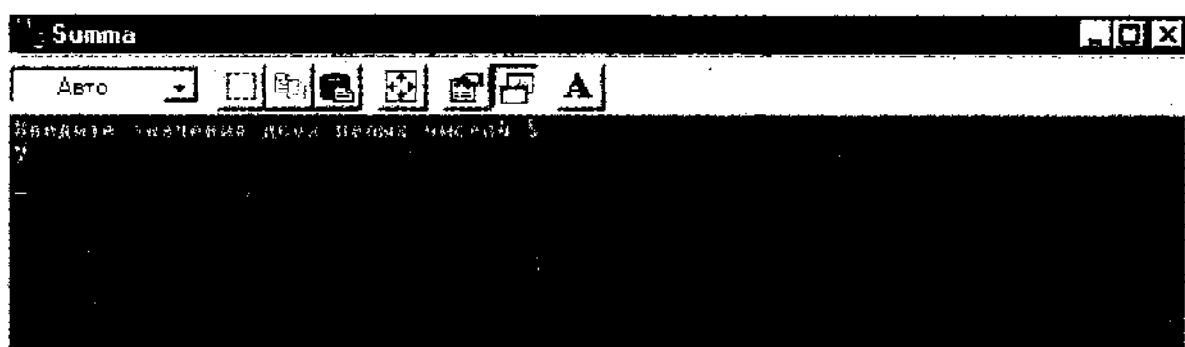


Рис. 5.2. Окно MS DOS, в котором выполняется консольное приложение Summa

После просмотра результата вычислений нажмите клавишу Enter, и окно MS DOS, в котором выполнялось консольное приложение, будет закрыто.

Попробуйте изменить текст программы, отключив использование функции CharToOem и организовав вывод сообщения оператором Write следующим образом:

```
{CharToOem('Введите значения двух целых чисел',TmpStr);}
Write('Введите значения двух целых чисел');

{Вывод сообщения в кодировке ANSI}
```

Откомпилируйте и запустите приложение на выполнение, нажав клавишу F9. Текст на экране консольного приложения в кодировке Windows 1251 (ANSI) будет напечатан непонятными символами (рис. 5.3).

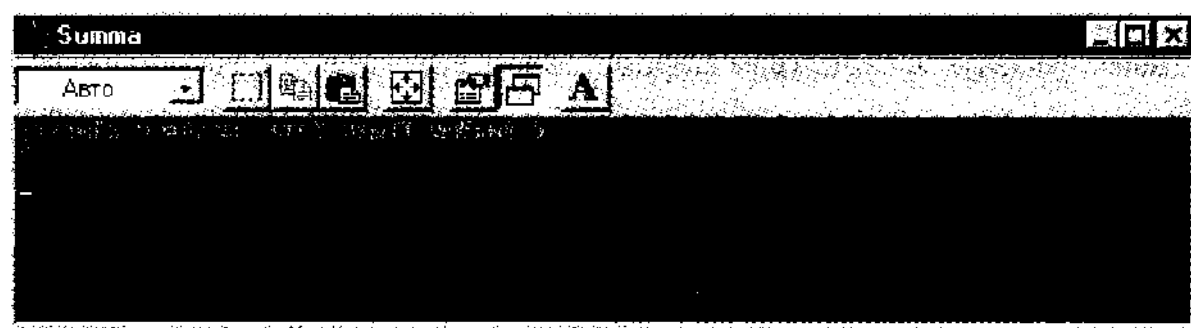


Рис. 5.3. Вид текста консольного приложения
в кодировке Windows 1251

Закройте окно MS DOS, в котором выполнялось консольное приложение, и отмените изменения, сделанные в программе, нажимая клавиши Ctrl+Z для отмены операции редактирования.

Многооконный текстовый редактор

Упражнение 5.2. Создайте многооконный текстовый редактор, который будет способен: редактировать текст в нескольких окнах, изменять шрифт текста, выравнивать строки в абзаце, находить и заменять фрагменты текста, выделять, копировать и вставлять фрагменты текста, сохранять текст и выводить его на печать.

Создание родительского окна в MDI-приложении. Запустите интегрированную среду разработки Delphi и создайте новое приложение командой File ► New ► Application. Переименуйте имя формы, выбрав в окне Инспектор объектов объект Form1 и заменив значение свойства Form1.Name на MainForm. Задайте свойству MainForm.Caption значение «Текстовый редактор». Чтобы окно редактора было родительским окном в MDI-приложении, задайте свойству MainForm.FormStyle значение fsMDIForm.

Примечание. MDI – это сокращение от Multiple Document Interface (интерфейс для одновременной работы с несколькими документами). В MDI-приложениях два или более окна могут быть активны одновременно. Наряду с MDI имеются приложения SDI (Single Document Interface, интерфейс для работы с одним документом). В SDI-приложениях в каждый момент времени может быть активным только одно окно. MDI-приложения являются удобным средством для одновременного вывода на экран текста или данных, хранящихся в разных файлах. Такую структуру построения окон можно использовать для редактирования текстов, открывая и выводя на экран одновременно несколько разных документов. Пример такой работы с файлами представляет собой программа Microsoft Word.

Сохраните файлы проекта и модуля в папке «Многооконный текстовый редактор» под именами Edit.dpr и main.pas. Для создания меню приложения поместите на форму компонент MainMenu из палитры Standard. Дважды щелкните мышью по компоненту MainMenu на форме, чтобы раскрыть конструктор меню окна.

Далее, пользуясь свойством Caption, задайте для меню следующую структуру.

Файл	Правка	Формат	Окна	Помощь
Создать	Отменить	Шрифт	Каскад	О программе
Открыть	Вырезать	Выравнивание влево	Горизонтально	
Сохранить	Копировать	По центру	Вертикально	
Сохранить как	Вставить	Вправо		
Печать	Найти			
Выход	Заменить			

Чтобы вызвать конструктор для размещенного в форме компонента главного меню (MainMenu), можно или дважды щелкнуть по нему мышью, или дважды щелкнуть мышью по свойству Items в Инспектор объектов.

Выбрать изменяемый пункт можно мышью, для изменения названия выберите в Инспектор объектов свойство Caption, для отделения секций меню линией используйте знак тире в свойстве Caption для нужного пункта меню.

Для изменения, добавления или удаления пунктов в конструкторе меню вызовите контекстное меню, используя правую клавишу мыши, и выберите нужное действие.

Проектирование дочерних окон приложения. Чтобы во время работы приложения все дочерние окна отображались списком в меню Окна, выберите пункт меню Окна и переименуйте его в окне Инспектор объектов в WindowMenu. Затем, выбрав в окне Инспектора объектов форму MainForm, в свойстве WindowMenu выберите из раскрывающегося списка пункт меню WindowMenu.

Спроектируйте окно дочернего документа, для чего добавьте в проект новую форму, выбрав команду File ► New ► Form. Сделайте окно Form2 меньше глав-

ного родительского окна и переименуйте свойство Name этого окна из Form2 в ChildForm. Задайте свойству дочернего окна MDI-приложения ChildForm.FormStyle значение fsMDIChild.

Поместите на форму ChildForm компонент RichEdit из палитры Win32. Чтобы компонент RichEdit занимал все окно ChildForm и во время выполнения приложения его размеры изменялись при изменении размеров окна, измените свойство RichEdit.Align на alClient.

Примечание. Компонент TRichEdit представляет собой многофункциональное средство редактирования текстов, позволяющее работать с форматом .rtf, т.е. выбирать различные атрибуты форматирования для разных фрагментов текста. В этом основное отличие TRichEdit от более простого компонента TMemo, в котором атрибуты форматирования одинаковы для всего текста. Окно редактирования снабжено многими функциями, свойственными большинству редакторов. Например, в нем предусмотрены типичные комбинации клавиш оперативного доступа:

Ctrl-C – копирование выделенного текста в буфер обмена (команда Copy);

Ctrl-X – вырезание выделенного текста в буфер обмена (команда Cut);

Ctrl-V – вставка текста из буфера обмена в позицию курсора (команда Paste);

Ctrl-Z – отмена последней команды редактирования.

Дважды щелкнув мышью в поле значений свойства RichEdit1.Lines, в окне String List Editor удалите текст и нажмите кнопку OK.

Сохраните новый модуль под именем Child.pas в папке «Многооконный текстовый редактор» и изменения в проекте под прежним именем Edit.dpr. Нажатием клавиши Shift-F12 выберите в окне View Form главную форму приложения MainForm и щелкните мышью по кнопке OK (рис. 5.4).

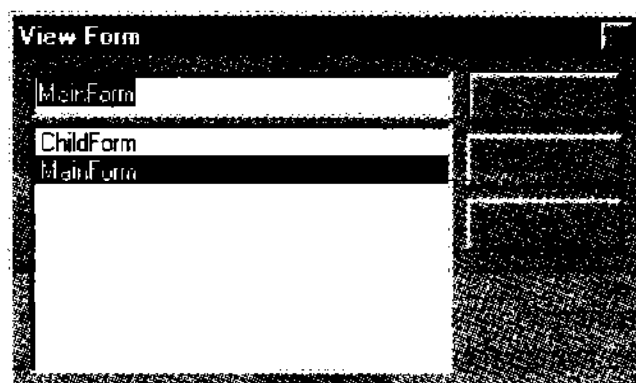


Рис. 5.4. Переход к форме MainForm

Поместите на главной форме MainForm компоненты OpenFileDialog, SaveDialog, FontDialog, PrintDialog, FindDialog, ReplaceDialog из палитры Dialogs, а также компонент ActionList из палитры Standard. Они понадобятся в дальнейшем при обработке событий в нашем приложении. Так как эти компоненты не являются визуальными, то их можно разместить в любом месте формы (рис. 5.5).

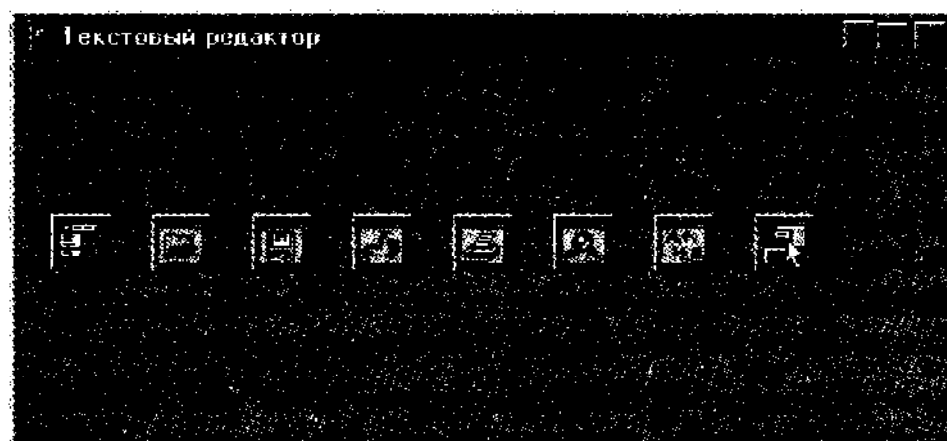


Рис. 5.5. Окно главной формы приложения с компонентами

Для создания дочернего окна документа в редакторе только при выборе меню **Файл** ▶ **Создать** отключите создание окна во время запуска приложения, которое задается Delphi по умолчанию. Для этого выберите в меню Delphi команду **Project** ▶ **Options...** В окне **Project Options** на вкладке **Forms** выберите форму **ChildForm** и, щелкнув мышью по кнопке со стрелкой, перенесите имя дочерней формы **ChildForm** из списка **Auto-create forms** (Формы, автоматически создающиеся при старте приложения) в список **Available forms** (Доступные формы) (рис. 5.6). Для завершения операции щелкните мышью по кнопке **OK**.

Задайте для приложения значок с изображением, например, блокнота с ручкой. Для этого присвойте соответствующее значение свойству **MainForm.Icon**, а также выберите этот значок при установке параметра **Application Icon** в окне **Project Options**.

Сохраните изменения в проекте командой **File** ▶ **Save All**.

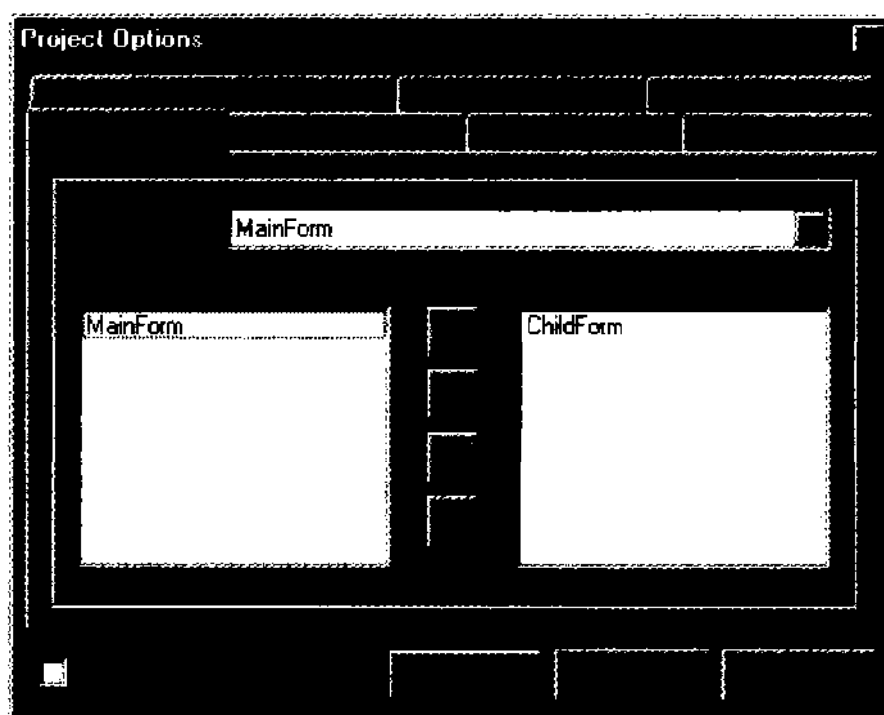


Рис. 5.6. Отключение создания окна **ChildForm** во время запуска приложения

Создание процедур обработчиков событий в текстовом редакторе. Создайте процедуры обработчиков событий в приложении.

Для открытия нового дочернего окна приложения создайте процедуру обработчика выбора команды Файл ► Создать. Для этого выберите на форме MainForm в меню Файл команду Создать. При этом в программном модуле Main будет сгенерирована процедура обработчика события, которую нужно отредактировать следующим образом:

```
procedure TMainForm.N2Click(Sender: TObject);  
begin  
    TChildForm.Create(Self); {Создать дочернее окно документа}  
end;
```

Так как в этом окне создается другое окно, то к модулю Main необходимо подключить модуль Child. Для этого следует выбрать команду File ► Use Unit..., в окне Use Unit указать модуль Child и щелкнуть по кнопке ОК.

Сохраните проект при помощи команды File ► Save All, откомпилируйте и запустите приложение на выполнение. Проверьте работу только что написанной процедуры обработчика события, выбирая в меню приложения команду Файл ► Создать. Убедитесь, что в окне приложения создается дочернее окно. Попробуйте закрыть дочернее окно ChildForm. Оно не закрывается, а минимизируется. Закройте окно приложения, щелкнув мышью по кнопке закрытия окна в правом верхнем углу окна.

Для того чтобы окно создаваемого документа имело название «Новый», создайте процедуру обработки события OnCreate для объекта ChildForm. Чтобы создать эту процедуру, откройте форму ChildForm и в окне Инспектор объектов на странице Events дважды щелкните мышью в поле значений события OnCreate, а затем добавьте оператор присваивания значения «Новый» свойству Caption создаваемого дочернего окна, как показано далее:

```
procedure TChildForm.FormCreate(Sender: TObject);  
begin  
    Caption:='Новый'; // Заголовок окна нового файла  
end;
```

Чтобы дочернее окно нормально закрывалось, перейдите в окно ChildForm (Shift-F12), затем выберите в окне Инспектор объектов объект ChildForm и на странице Events дважды щелкните мышью по пустому полю справа в строке события OnClose (Закрытие окна). Отредактируйте заготовку процедуры закрытия окна следующим образом:

```
procedure TChildForm.FormClose(Sender: TObject; var Action:  
TCloseAction);  
begin  
    Action := caFree; {Закреть форму и удалить из памяти}  
end;
```

Примечание. Здесь вносимый в процедуру параметр Action определяет поведение данной формы при закрытии. Он может принимать следующие значения:

- `saNone` – форма не предпринимает каких-либо действий;
- `saHide` – форма делается скрытой, но не удаляется из памяти. Поскольку мы создаем дочернюю форму «вручную» (программно), то должны при закрытии удалить ее из памяти;
- `saFree` – форма закрывается и удаляется из памяти;
- `saMinimize` – форма минимизируется. При использовании MDI это значение для дочерних окон устанавливается по умолчанию.

Для создания процедуры закрытия окна редактора выберите команду **Файл** ▶ **Выход** в списке объектов **Object TreeView** и на странице **Events** в окне **Инспектор объектов** дважды щелкните мышью в поле значений события **OnClick**. После этого добавьте в процедуру обработки этого события оператор **Close**, так что текст процедуры будет выглядеть следующим образом:

```
procedure TMainForm.N8Click(Sender: TObject);
begin
    Close; {закреть окно приложения}
end;
```

Сохраните, откомпилируйте и запустите приложение на выполнение. Открывая и закрывая дочерние окна, убедитесь в правильности работы процедуры обработчика закрытия дочернего окна. Выбрав команду **Файл** ▶ **Выход**, завершите работу приложения.

Создание процедуры открытия файла. Создавая процедуру открытия файла, учтите, что при выборе соответствующего пункта меню сначала открывается диалоговое окно выбора файла, при успешном выборе создается форма **ChildForm**, в которой в компонент **RichEdit1** загружается этот файл. Поэтому для организации открытия файла используйте компонент **OpenDialog**. Выделите объект в окне **Инспектор объектов** и задайте ему следующие свойства: для **OpenDialog1.Title** – значение «Открыть текстовый файл», а для свойства **OpenDialog1.Filter** – значения, как показано на рис. 5.7.

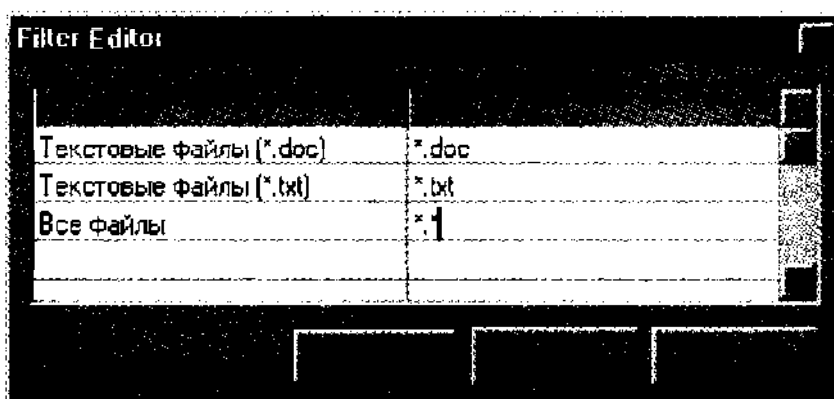


Рис. 5.7. Создание фильтра для **OpenDialog1**

Для создания процедуры открытия файла выберите в меню **Файл** на форме **MainForm** команду **Открыть**. После этого в окне редактора кода будет создана

заготовка процедуры реакции на событие выбора этого меню. Отредактируйте ее следующим образом:

```
procedure TMainForm.N3Click(Sender: TObject);
begin
    if OpenFileDialog1.Execute then      {Если файл выбран,
    то выполнять следующее}
        begin
            ChildForm := TChildForm.Create(Self);      {Создать
дочернее окно}
            ChildForm.RichEdit1.Lines.LoadFromFile(OpenDialog1.
FileName);
            {Загрузить в RichEdit1 Выбранный файл}
            ChildForm.Caption := OpenFileDialog1.FileName; {Установить
в заголовок дочернего окна название файла}
            ChildForm.RichEdit1.Tag:=0;    {Признак неизмененного
файла}
        end;
    end;
```

Сохраните изменения в проекте, откомпилируйте и запустите приложение на выполнение. Выбирая команду Файл ► Открыть, проверьте правильность выполнения процедуры открытия файла, открыв несколько файлов. Открытые окна располагаются в окне редактора каскадом. Выбрав в меню приложения пункт Окна, обратите внимание на то, что в меню Окна отображаются имена дочерних окон документов, а при выборе имени документа в меню Окна окно с этим документом становится активным и отображается на переднем плане.

Создание процедур обработки команд меню Окна. Для того чтобы изменить порядок отображения файлов в окне приложения, необходимо создать процедуры обработки команд меню Окна.

Откройте главную форму приложения и дважды щелкните мышью по компоненту ActionList1 или выберите пункт Action List Editor из контекстного меню. В окне Editing MainForm1.ActionList1 нажмите Ctrl+Ins или, щелкнув мышью по желтой кнопке в панели инструментов этого окна, выберите пункт New Standard Action из раскрывающегося меню для открытия окна Standard Action Classes со списком стандартных действий, которые можно привязать к пунктам меню или кнопкам. При нажатой клавише Shift или Ctrl выберите все стандартные действия, которые относятся к категории Window (рис. 5.8).

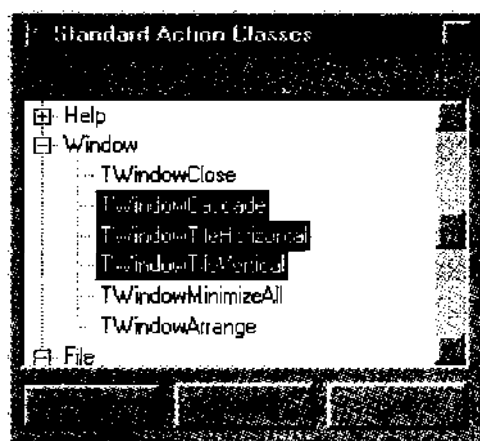


Рис. 5.8. Выбор списка стандартных действий управления состоянием окон

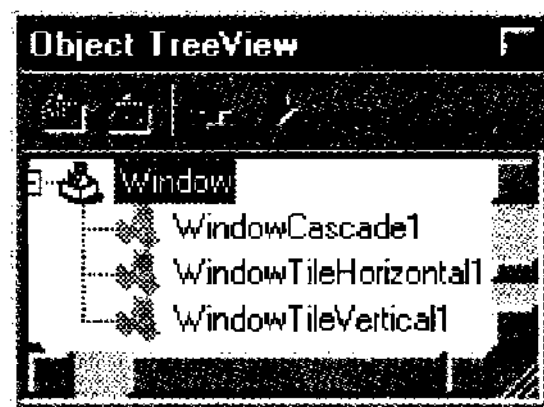


Рис. 5.9. Выбранные стандартные действия с окнами в окне Object TreeView

Завершите выбор стандартных действий, щелкнув мышью по кнопке ОК. После этого в списке действий появится категория Window, а в правой части окна будет показан список всех выбранных стандартных действий с окнами. Закройте окно `Editing MainForm1.ActionList1` и убедитесь в том, что список выбранных стандартных действий отображается в окне Object TreeView (рис. 5.9).

Укажите для каждой из команд меню Окна соответствующее стандартное действие. Выбрав в окне Object TreeView команду Каскад, в окне Инспектор объектов на странице Events откройте список значений стандартных событий Action и выберите в нем событие WindowCascade1 (рис. 5.10).

Так как по умолчанию названия всех стандартных действий написаны по-английски, то после этой операции название соответствующей команды меню поменяется на стандартное `&Cascade`. Чтобы изменить это название, на странице Properties выберите свойство Caption и замените его на `&Каскад`.

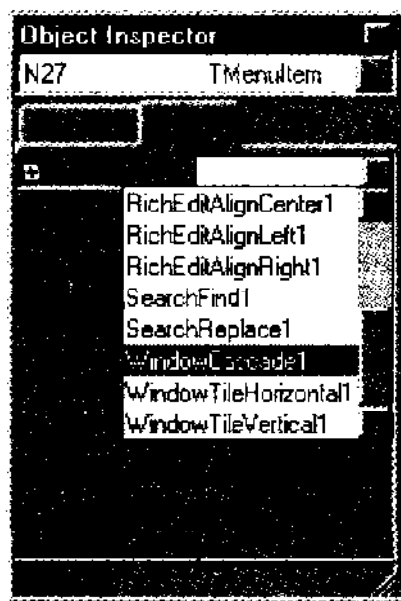


Рис. 5.10. Выбор стандартного события для команды меню

Примечание. Знак амперсанта (&) означает, что следующий за ним символ будет подчеркнут. Это свойство используется для быстрого выбора данного пункта меню с помощью оперативной клавиши, когда меню активизировано. Положение знака & указывает оперативную клавишу для выбора данной команды меню.

Аналогичным образом задайте стандартные действия для команд Горизонтально и Вертикально меню Окна, выбрав действия WindowTileHorizontal1 и WindowTileVertical1 соответственно, а затем отредактировав значения их свойств Caption.

Сохраните изменения в проекте, откомпилируйте и запустите приложение на выполнение. Выбором команды Файл ► Открыть откройте несколько файлов. Выбирая в меню Окна команды Каскад, Горизонтально или Вертикально, убедитесь в том, что окна дочерних документов располагаются в окне приложения в соответствии с выбранной командой.

Примечание. Обратите внимание на то, что если дочерних окон нет, то пункты меню Окна становятся недоступными.

Пример с подключением стандартных действий для обработки событий выбора команды в меню Окна наглядно иллюстрирует возможности создания приложения в визуальной объектно-ориентированной среде программирования Delphi. Вы не написали ни одной строки программного кода, а решили задачу, лишь изменяя некоторые свойства объектов в приложении.

Аналогичным образом подключите стандартные действия для обработки пунктов меню Правка (Отменить, Вырезать, Копировать, Вставить) и Формат ▸ Выравнивание (Влево, По центру, Вправо). При этом не забудьте о русификации свойства Caption.

Создание процедуры изменения шрифта. Для создания процедуры изменения шрифта выберите на главной форме приложения команду Формат ▸ Шрифт и отредактируйте заготовку процедуры обработчика этого события следующим образом:

```
procedure TMainForm.N21Click(Sender: TObject); {Изменение шрифта}
begin
    if FontDialog1.Execute then {Если пользователь выбрал шрифт}
    then (ActiveMDIChild as TChildForm).RichEdit1.Font :=
FontDialog1.Font;
    {Установить выбранный шрифт}
end;
```

Как видно из текста процедуры, сначала в диалоговом окне Выбор шрифта пользователь задает параметры шрифта, а затем, если эта операция выполнена успешно, выбранный шрифт присваивается свойству RichEdit1.Font окна документа.

Примечание. Строка (ActiveMDIChild as TChildForm) позволяет получить доступ из главного окна к активному дочернему окну. Эта строка равноценна ChildForm, выполняется она только для активного окна.

Сохраните изменения в проекте, откомпилируйте и запустите приложение на выполнение. Создав новый документ, введите текст и проверьте действие команды Формат ▸ Шрифт для изменения шрифта в окне документа.

Создание процедуры сохранения файла. Чтобы документ, открытый в окне редактора, сохранялся в файле на диске, задайте для свойства SaveDialog1.Filter значение (Текстовый файл (*.rtf) |*.rtf| Текстовый файл (*.txt) |*.txt) и создайте процедуры обработчика события выбора в меню команды Файл ▸ Сохранить как... Для создания процедуры сохранения файла выберите на главной форме приложения команду Файл ▸ Сохранить как... и отредактируйте заготовку процедуры обработчика этого события следующим образом:

```

procedure TMainForm.N6Click(Sender: TObject); {Сохранить
как...}
begin
    if MDIChildCount=0 then Exit; {Если нет MDI дочерних окон,
то выйти из процедуры}
    SaveDialog1.FileName := FName; {Присвоить свойству FileName
значение из FName}
    SaveDialog1.Title:='Сохранить файл как...'; {Заголовок
диалогового окна сохранения файла}
    if SaveDialog1.Execute then
        begin
            FName:=SaveDialog1.FileName; {Открыть диалог и запомнить
новое имя файла}
            Case SaveDialog1.FilterIndex of {Изменить расширение
файла}
                1 : FName:=ChangeFileExt(FName, '.rtf');
                2 : FName:=ChangeFileExt(FName, '.txt');
            end;
        end;
    (ActiveMDIChild as TChildForm).RichEdit1.Lines.
SaveToFile(FName);
        {Записать в файл содержимое из свойства
Lines объекта RichEdit активного окна}
    (ActiveMDIChild as TChildForm).Caption:=FName;
    {Заменить имя файла в заголовке активного окна}
    (ActiveMDIChild as TChildForm).RichEdit1.Tag:=0;
        {Изменения в файле сохранены}
end;

```

Как видно из текста процедуры, сначала проверяется, есть ли открытое окно документа. Если открытых окон документов нет, то процедура завершается. Если имеется открытое окно, то открывается диалоговое окно Сохранить файл как..., в котором задается вариант имени сохраняемого файла, взятый из переменной FName. В диалоговом окне пользователь может отредактировать имя и расширение файла. Если диалог завершается успешно, то запоминается присвоенное пользователем имя и расширение файла, а содержимое из свойства Lines объекта RichEdit1 активного окна записывается в файл. После сохранения файла обновляется значение имени файла в заголовке окна документа и свойству Tag объекта RichEdit1 активного окна присваивается значение 0.

Для создания процедуры сохранения документа по команде Файл ► Сохранить выберите в окне на главной форме приложения команду Файл ► Сохранить

и отредактируйте заготовку процедуры обработчика этого события следующим образом:

```
procedure TMainForm.N5Click(Sender: TObject); {Сохранить файл}
begin
    if MDIChildCount=0 then Exit;
    {Если нет MDI дочерних окон, то выйти из процедуры}
    if (ActiveMDIChild as TChildForm).Caption='Новый' then
    {Если документ имеет заголовок окна «Новый»}
        N6Click(Sender) {Вызвать процедуру «Сохранить как...»}
    else
        begin {Сохранить файл с именем, указанным в заголовке}
            (ActiveMDIChild as TChildForm).RichEdit1.Lines.
                SaveToFile((ActiveMDIChild as TChildForm).Caption);
            (ActiveMDIChild as TChildForm).RichEdit1.Tag:=0;
            {Изменения в файле сохранены}
        end;
end;
```

Как видно из текста процедуры, она выполняет операцию сохранения файла, если окно документа не является пустым. Если документ имеет название «Новый», то вызывается процедура обработчика события «Сохранить как...», текст которой был рассмотрен ранее.

Чтобы проверять при закрытии окна, редактировался ли текст, и, если он изменялся, предлагать сохранить файл, создайте процедуру обработки события закрытия окна документа. У каждого компонента в Delphi есть свойство Tag. Оно имеет тип Integer и является свойством, которым пользуются только программисты. Tag в программе ни на что не влияет и по умолчанию принимает нулевое значение. Как вы помните, при сохранении документа мы присваивали свойству Tag дочернего окна документа значение 0. Если документ в окне изменился, то нужно присвоить свойству Tag значение 1, чтобы указать на необходимость сохранения изменений при попытке закрыть окно.

Чтобы значение свойства Tag содержало информацию об изменениях в окне редактирования, создайте процедуру обработчика события OnChange для объекта RichEdit1 и включите в нее оператор присваивания RichEdit1.Tag:=1;

```
procedure TChildForm.RichEdit1Change(Sender: TObject);
begin
    RichEdit1.Tag:=1; {Текст изменен, должен быть сделан запрос
о сохранении}
end;
```

Для создания процедуры закрытия окна документа с сохранением изменений откройте форму ChildForm и в окне Инспектор объектов на странице Events дважды щелкните мышью в строке события OnCloseQuery. Отредактируйте заготовку процедуры следующим образом:

```

procedure TChildForm.FormCloseQuery(Sender: TObject; var
CanClose: Boolean);
    Var Res:Integer; {Переменная для хранения результата выбора
действия}
begin
    {Если не сохранять файл, то}
    if RichEdit1.Tag=0 then CanClose:=True {Можно закрыть
окно}
    else
        begin {Запрос о сохранении}
            Res:=Application.MessageBox('Текст изменен. Сохранить
изменения?', 'Вопрос', MB_YESNOCANCEL + MB_ICONQUESTION);
            if Res=IDYES then
                begin {Нажата кнопка «Да»}
                    ChildForm.BringToFront; {Расположить данное окно
выше всех (сделать активным)}
                    MainForm.Save(Sender); {Вызвать процедуру
«Сохранить»}
                    if RichEdit1.Tag=1 then CanClose:=False;
                    {Если пользователь не сохранил файл, то окно не
закрывать}
                end;
                if Res=IDNo then CanClose:=True;
                {Если «Нет», то можно закрыть}
                if Res=IDCANCEL then CanClose:=False;
                {Если «Отмена», то не закрывать}
            end;
        end;
end;

```

Как видно из текста процедуры, сначала проверяется значение свойства Tag. Если оно равно нулю, то документ в окне не изменялся, и можно закрыть окно. Если Tag=1, то в диалоговом окне отображается запрос о сохранении изменений, и в зависимости от выбора пользователя, определяемого значением параметра CanClose, выполняются следующие операции: сохранение документа, выход без сохранения или отказ от закрытия окна.

Создание процедуры вывода текста на печать. Для того чтобы редактор обеспечивал вывод текстов на печать, необходимо создать процедуру обра-

ботчика события выбора команды **Файл ► Печать**. Для этого откройте форму **MainForm** и выберите эту команду, а затем в окне Редактор кода отредактируйте процедуру печати следующим образом:

```
procedure TMainForm.Print(Sender: TObject); {Печатать текстовый
  файл}
begin
  if PrintDialog1.Execute then begin
    AssignPrn(F); {Объявить файловую переменную F – установить
    ассоциативную связь между файловой переменной F и принтером}
    Rewrite(F); {Создать и открыть файл печати}
    Writeln(F, (ActiveMDIChild as TChildForm).RichEdit1.Text);
    {Вывести текст из RichEdit1 на принтер}
    System.CloseFile(F); {Закрыть файл печати}
  end;
end;
```

Примечание. В раздел **Uses** необходимо добавить вызов модуля **Printers**. Сохраните измененный проект, откомпилируйте и запустите приложение на выполнение. Открыв текстовый файл, проверьте печать текста командой **Файл ► Печать**.

Создание процедур поиска и замены фрагментов текста. Создайте процедуры поиска и замены фрагментов текста. Чтобы найденный фрагмент текста выделялся, задайте для свойства **RichEdit1.HideSelection** значение **False**. Для хранения позиции начала поиска введите в раздел описания переменных объявление целой переменной **SPos**.

Для вызова диалога поиска выберите на форме **MainForm** команду **Правка ► Найти** и отредактируйте заготовку процедуры обработчика этого события следующим образом:

```
{Процедура обработки события OnClick пункта меню «Найти»}
procedure TMainForm.N20Click(Sender: TObject); { Вызов окна
  диалога поиска}
begin
  SPos:=(ActiveMDIChild as TChildForm).RichEdit1.SelStart;
  {Запомнить позицию курсора}
  with FindDialog1 do
    begin
      FindText:=(ActiveMDIChild as TChildForm).RichEdit1.
SelText;
      {Начальное значение искомого текста – выделенный текст}
      {позиционирование окна диалога поиска}
```

```

        Position:=Point (ChildForm.Left,ChildForm.Top+
                        ChildForm.RichEdit1.Top+ChildForm.RichEdit1.
Height);
        Options:=Options+[frHideUpDown,frHideWholeWord];
        {Удаление из окна диалога кнопок «Вверх», «Вниз»,
«Только слово целиком»}
        Execute;      {Выполнение поиска}
        end;
    end;

```

Для того чтобы при щелчке мышью по кнопке Найти далее в окне Поиск выполнялся поиск заданного фрагмента текста, создайте процедуру обработки события OnFind объекта FindDialog1 и отредактируйте ее следующим образом:

```

{Процедура обработки события OnFind объекта FindDialog1}
procedure TMainForm.FindDialog1Find(Sender: TObject);
begin
    with FindDialog1 do
        begin
            if frMatchCase in Options then      {Поиск с учетом
регистра}
                ChildForm.RichEdit1.SelStart:=Pos (FindText,
                Copy (ChildForm.RichEdit1.Lines.Text, SPos+1,
                Length (ChildForm.RichEdit1.Lines.Text))) +SPos-1
            else                                {Поиск без учета регистра}
                ChildForm.RichEdit1.SelStart:=Pos (AnsiLowerCase (FindText),
                AnsiLowerCase (Copy (ChildForm.RichEdit1.Lines.Text, SPos+1,
                Length (ChildForm.RichEdit1.Lines.Text)))) +SPos-1;
            if ChildForm.RichEdit1.SelStart>=SPos then
                begin {Выделение найденного текста}
                    ChildForm.RichEdit1.SelLength:=Length (FindText);
                    SPos:=ChildForm.RichEdit1.SelStart+
                    ChildForm.RichEdit1.SelLength+1;
                end
            else
                if MessageDlg ('Текст »'+FindText+'« не найден.
Продолжать диалог?', mtConfirmation, mbYesNoCancel, 0) <>mrYes
                then CloseDialog;
            end;
            ChildForm.RichEdit1.SetFocus;
        end;
    end;

```

Для вызова окна диалога замены выберите на форме MainForm команду Правка ► Заменить и отредактируйте заготовку процедуры обработчика этого события следующим образом:

```

{Процедура вызова окна диалога замены: обработка события OnClick
пункта меню «Заменить»}
procedure TMainForm.N19Click(Sender: TObject);
begin
    SPos:=ChildForm.RichEdit1.SelStart;      {Запомнить позицию
курсора}
    with ReplaceDialog1 do
    begin
        FindText:=ChildForm.RichEdit1.SelText; {Начальное значение
искомого текста - выделенный текст}
        {Позиционирование окна диалога поиска}
        Position:=Point(ChildForm.Left,ChildForm.Top+
ChildForm.RichEdit1.Top+ChildForm.RichEdit1.Height); Options:
=Options+[frHideUpDown,frHideWholeWord,frReplaceAll];
        {Удаление из диалога кнопок}
        Execute; {Выполнение поиска-замены}
    end;
end;

```

Чтобы при щелчке на кнопке Заменить в окне Замена выполнялся поиск фрагмента текста по образцу и замена его на заданный, создайте процедуру обработки события OnReplace объекта ReplaceDialog1 и отредактируйте ее следующим образом:

```

{Процедура обработки события OnReplace объекта
ReplaceDialog1}
procedure TMainForm.ReplaceDialog1Replace(Sender: TObject);
begin
    with TReplaceDialog(Sender) do
    begin
        SPos := Pos(FindText, ChildForm.RichEdit1.Lines.Text);
        if SPos > 0 then
        begin {Если есть искомый текст}
            ChildForm.RichEdit1.SelStart := SPos - 1;
            ChildForm.RichEdit1.SelLength := Length(FindText);
            ChildForm.RichEdit1.SelText := ReplaceText; {Замена
текста}
        end
        else MessageDlg('Текст »'+FindText+'« не найден.',
mtError,[mbOk],0);
    end;
end;

```


Сохраните текст программы, откомпилируйте его и проверьте работу процедур поиска и замены.

Создание окна модального типа для отображения информации о программе. Чтобы при выборе в меню приложения команды **Помощь ► О программе** вывести сведения о программе, создайте новое окно модального типа, открыв которое пользователь не сможет работать с другими окнами, пока не закроет окно сообщения **О программе**. Создайте новое окно командой **File ► New ► Form**. Установите размер новой формы и измените значение свойства **Name** на **About**. Задайте свойству **About.Caption** значение «О программе». Выбрав команду **File ► Save All**, сохраните новое окно под именем **AboutUnit**. Чтобы окно сообщения данных о программе открывалось посередине экрана, задайте свойству **About.Position** значение **poScreenCenter**. Чтобы пользователь не мог свернуть это окно или развернуть его на весь экран, задайте свойству **About.BorderStyle** значение **bsDialog**.

Поместите на форму **About** кнопку **TbitBtn** из палитры компонентов **Additional**. Чтобы окно **About** закрывалось при выборе пользователем этой кнопки без написания каких-либо процедур с вашей стороны, задайте свойству **About.Kind** значение **bkOK**. После этого кнопка приобретет новое название **OK** и на ней появится рисунок зеленой «галочки».

Разместите на форме **About** компоненты **Label1** и **Label2**. Задайте их свойствам **Caption** значения «Пример текстового редактора» и «© 2002 Версия 1.0» соответственно. Задайте шрифт в свойствах **Label1**.

Выровняйте и зафиксируйте компоненты на форме, как показано на рис. 5.11.



Рис. 5.11. Вид формы **About**

Создайте процедуру открытия окна **About** при выборе команды **Помощь ► О программе**. Для этого перейдите на форму **MainForm** и выберите команду **Помощь ► О программе**, а затем в заготовке процедуры введите оператор вызова окна сообщения модального типа **About.ShowModal**., так что текст процедуры будет выглядеть следующим образом:

```
procedure TMainForm.N31Click(Sender: TObject);
begin
    About.ShowModal;
end;
```

Чтобы подключить к проекту модуль AboutUnit, выберите команду File ► Use Unit и в окне Use Unit выберите модуль AboutUnit.

Сохраните изменения в проекте, откомпилируйте и запустите приложение на выполнение. Проверьте появление окна сообщения О программе при выборе команды Помощь ► О программе.

Итак, вы создали многооконный текстовый редактор, который, конечно, можно будет в дальнейшем усовершенствовать.

Взаимодействие приложения с внешними программами

Создавая приложения в ИСР Delphi, вы имеете возможность организовать следующие взаимодействия вашего приложения с внешними программами:

- непосредственный запуск внешней программы из приложения;
- запуск внешней программы, связанной с некоторым документом;
- обмен сообщениями с другими программами;
- внедрение и связывание документов, подготовленных внешними программами, с вашим приложением;
- динамический обмен данными между приложениями.

Функции выполнения внешних приложений. Рассмотрим пример, в котором из вашего приложения будут запускаться различные внешние программы. Для запуска внешних программ можно использовать функцию WinExec или ShellExecute.

Функция WinExec позволяет выполнить указанное приложение. Определенные функции: `function WinExec(CmdLine: PChar; CmdShow: integer): integer;`

Параметр CmdLine является указателем на строку с нулевым символом в конце, содержащую имя исполняемого файла и, если необходимо, параметры командной строки. Если имя указано без пути, то Windows ищет выполняемый файл в следующей последовательности:

- 1) в каталоге, из которого запущено приложение;
- 2) в текущем каталоге;
- 3) в системном каталоге Windows, имя которого возвращается функцией GetSystemDirectory;
- 4) в каталоге Windows, имя которого возвращается функцией GetWindowsDirectory;
- 5) в каталогах из списка в переменной окружения PATH.

Параметр CmdShow определяет форму представления окна запускаемого приложения Windows. Возможные значения этого параметра можно увидеть в разделе ShellExecute. Для приложений, не являющихся Windows-приложениями, файлов PIF и т. д., состояние окна определяет само приложение.

При успешном выполнении запуска приложения функция WinExec возвращает значение, большее 31.

Достоинством функции WinExec является ее совместимость с ранними версиями Windows. Собственно, для этого она и сохраняется в Win32, хотя для Win32 рекомендуется пользоваться функцией CreateProcess. При работе с Win32

функция WinExec завершает работу, если вызванное приложение вызывает функцию GetMessage или заканчивается выделенный лимит времени. Таким образом, ожидание можно прервать, предусмотрев в процессе, запущенном с помощью WinExec, вызов в нужный момент функции GetMessage.

В отличие от WinExec функция ShellExecute позволяет не только выполнить любое приложение Windows, но и открыть файл документа, что означает выполнение связанного с ним приложения и загрузку в него этого документа. Например, обычно с документами, имеющими расширение .doc, связана программа WinWord. В этом случае открыть файл, например, с именем file.doc означает запустить WinWord и передать ему в качестве параметра имя файла file.doc. Кроме реализации описанных возможностей, функция ShellExecute позволяет распечатать указанный файл или открыть указанную папку. Последнее означает, что будет запущена программа Проводник с открытой папкой. Определение функции:

```
function ShellExecute(Wnd: HWnd; Operation, FileName,
    Parameters, Directory: PChar; ShowCmd: Integer): THandle;
```

Важнейшие параметры функции имеют следующие значения.

Параметр	Описание
Wnd	Родительское окно, в котором отображаются сообщения запускаемого приложения
Operation	Указывает на строку с нулевым символом в конце, которая определяет выполняемую операцию. Эта строка может содержать текст «open» (открыть) или «print» (напечатать). Для Windows 95 и NT определено еще одно значение: «explore» (открыть папку). Если параметр Operation равен nil, то по умолчанию выполняется операция «open»
FileName	Указывает на строку с нулевым символом в конце, которая определяет имя открываемого файла
Parameters	Указывает на строку с нулевым символом в конце, которая определяет передаваемые в приложение параметры, если FileName определяет выполняемый файл. Если FileName указывает на строку, определяющую открываемый документ, то этот параметр равен nil
Directory	Указывает на строку с нулевым символом в конце, которая определяет каталог, используемый по умолчанию
ShowCmd	Определяет режим открытия указанного файла

Функция ShellExecute возвращает дескриптор открытого приложения или дескриптор сервера DDE-приложения. Если возвращаемое значение меньше 32 или равно 32, это указывает на ошибку.

Создание приложения с использованием функции ShellExecute

Упражнение 5.3. Создайте приложение, из которого, используя команды меню, можно вызвать внешние программы Калькулятор, текстовый процессор Microsoft Word или табличный процессор Microsoft Excel.

Создайте новый проект и задайте для свойства `Form1.Caption` значение «Пример запуска внешних программ». Поместите на форму компонент `MainMenu` из палитры компонентов `Standard`. Дважды щелкните мышью по компоненту `MainMenu` на форме, чтобы раскрыть конструктор меню окна. Далее, пользуясь свойством `Caption`, задайте меню следующего вида (рис. 5.12).

Закройте окно конструктора меню и, открыв окно Редактор кода, добавьте в раздел `uses` вызов модуля `ShellAPI`, в котором содержится функция запуска внешней программы `ShellExecute`, так как Delphi автоматически не включает этот модуль.

Откройте форму приложения и, выбрав команду меню `Запустить ► Выход`, создайте процедуру завершения работы приложения, записав в теле процедуры оператор `Close`; следующим образом:

```
procedure TForm1.N1Click(Sender: TObject);  
begin  
    Close; {Закрывает окно приложения}  
end;
```

Для создания процедуры обработки выбора команды меню `Запустить ► Калькулятор` выберите эту команду в меню, а затем отредактируйте текст процедуры-обработчика следующим образом:

```
procedure TForm1.N2Click(Sender: TObject); {Вызов  
калькулятора}  
begin  
    ShellExecute(Handle, 'open', 'Calc', nil, nil, SW_Restore);  
end;
```

Как видно из оператора вызова функции `ShellExecute`, параметр «open» указывает на операцию «Открыть» приложение с именем файла «Calc». Значение `Parameters=nil` указывает на то, что параметр `FileName` определяет исполняемый файл. Значение `Directory=nil` указывает на строку с нулевым символом в конце, которая определяет каталог, используемый по умолчанию. Параметр `SW_RESTORE` команды `ShowCmd` активизирует и отображает окно запускаемого приложения. Если это окно свернуто или развернуто, то оно восстанавливается до своих первоначальных размеров и отображается в первоначальной позиции.

Сохраните проект под именем `ExecProgram`, откомпилируйте приложение и запустите его на выполнение. Выбрав в меню вашего приложения команду `Запустить ► Калькулятор`, проверьте запуск внешней программы (рис.5.13).



Рис. 5.12. Вид меню приложения
в окне конструктора меню

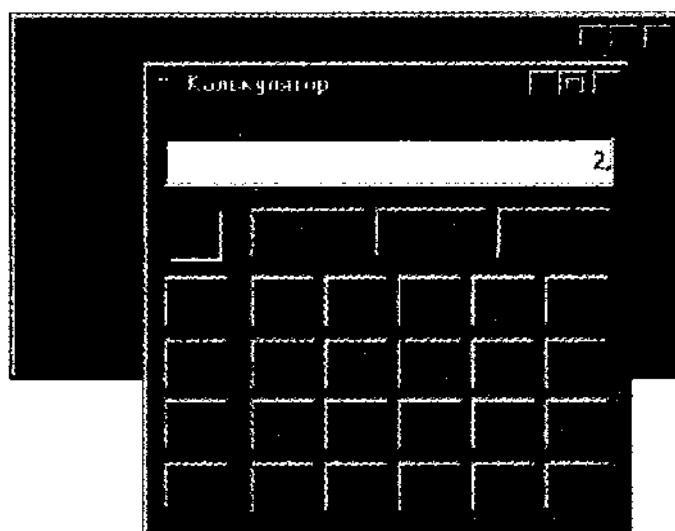


Рис. 5.13. Выполнение внешней программы,
вызванной из приложения

Закройте окно приложения Калькулятор и завершите работу приложения, выбрав команду Запустить ► Выход.

Создание приложения с использованием функции WinExec. Откройте форму приложения и, выбрав команду меню Запустить ► Winword, создайте процедуру запуска из вашего приложения текстового процессора Microsoft Word, отредактировав заготовку процедуры следующим образом:

```
procedure TForm1.Winword1Click(Sender: TObject);
begin
WinExec('C:\Program Files\Microsoft Office\Office\Winword.exe',
        SW_RESTORE);
end;
```

Для создания процедуры запуска из вашего приложения табличного процессора Microsoft Excel откройте форму приложения и, выбрав команду меню Запустить ► Excel, создайте заготовку процедуры, а затем вставьте в нее вызов функции WinExec, указав в качестве файла внешней программы файл Excel.exe с параметром SW_RESTORE (активизировать и отобразить окно запускаемого приложения) следующим образом:

```
procedure TForm1.Excel1Click(Sender: TObject);
begin
WinExec('C:\Program Files\Microsoft Office\Office\Excel.exe',
        SW_RESTORE);
end;
```

Сохраните изменения в проекте, откомпилируйте приложение и запустите его. Выбирая команды Запустить ► Калькулятор, Запустить ► Winword, Запустить ► Excel, проверьте правильность вызова внешних программ, затем закройте окна внешних программ и созданного вами приложения.

Приложение для работы с базами данных

Как было рассмотрено в главе 2, интегрированная среда разработки Delphi обеспечивает возможность создания приложений для работы с локальными и удаленными базами данных любых типов.

Упражнение 5.4. Создайте приложение, которое обслуживает реляционную базу данных «Памятники Московского Кремля», обеспечивая просмотр, добавление и удаление записей.

Примечание. Предполагается, что читатель знаком с основными понятиями реляционных баз данных.

Создание новой таблицы в Database Desktop. Для хранения информации о памятниках Московского Кремля создайте базу данных Paradox 7 с таблицей следующей структуры.

Имя поля	Тип	Размер	Назначение
Name	Символьный	30	Название памятника
Year	Символьный	15	Год создания
Author	Символьный	30	Авторы
Comment	Примечаний (Memo)		Краткое описание
Foto	Графический (Graphic)		Фотография памятника

Так как база данных в Paradox 7 – это папка, в которой расположены таблицы, то сначала создайте папку проекта (например, Kremlin), а затем в этой папке создайте папку (например, Data) для размещения в ней таблиц базы данных.

Чтобы запустить поставляемую в комплекте с Delphi программу Database Desktop, выберите в меню Delphi команду Tools ► Database Desktop или в главном меню Windows команду Пуск ► Программы ► Borland Delphi ► Database Desktop.

Для создания таблицы выберите в меню Database Desktop команду File ► New ► Table, а затем в окне Create Table (Создание таблицы) выберите тип таблицы Paradox 7 и щелкните мышью по кнопке OK.

В окне Create Paradox 7 Table опишите структуру данных. Щелкнув мышью в поле Field Name (Имя поля), введите имя поля, например Name. Завершите ввод имени поля, нажав клавишу Enter. После этого фокус ввода переместится в столбец Type (Тип поля). Щелкнув по клавише Space (Пробел) или щелкнув правой кнопкой мыши, откройте список типов данных и выберите из него

тип данных Alpha (Строка символов) и нажмите Enter. Нажатием клавиши Tab перейдите в столбец Size (Размер) и задайте максимальную длину строки символов, например 30. Нажав клавишу со стрелкой вниз, перейдите к описанию параметров следующего поля. Аналогичным образом опишите параметры полей Year, Author, Comment, Foto (рис. 5.14).

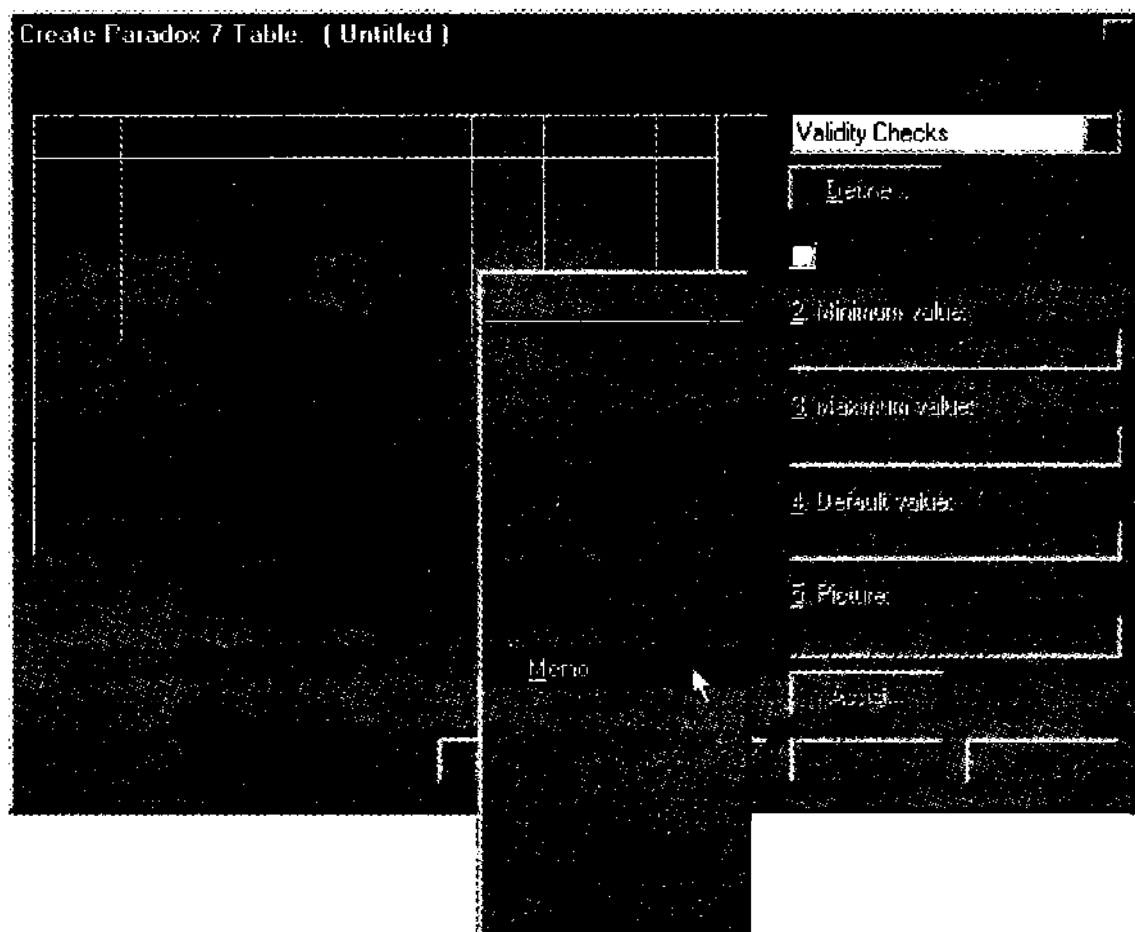


Рис. 5.14. Определение типа поля

Примечание. Для поля Comment задайте тип Memo – поле примечаний переменной длины. Чтобы иметь возможность хранить в поле Foto изображения, выберите для него тип Graphic.

В правой части окна можно задать дополнительные параметры, такие, как проверка правильности значений данных, таблица просмотра, целостность ссылок, вторичный индекс, пароль доступа, язык таблицы и др.

После описания структуры таблицы щелкните мышью по кнопке Save As и в окне Save Table As выберите в поле Сохранить в: папку для размещения таблиц базы данных, в нашем примере – Data. Затем в поле Имя файла: задайте имя файла таблицы, например Kreml. В раскрывающемся списке Alias оставьте неизменным псевдоним баз данных None (рис. 5.15). В дальнейшем мы рассмотрим изменение псевдонима базы данных.

Примечание. Псевдоним (alias) содержит всю информацию, необходимую для обеспечения доступа к базе данных.

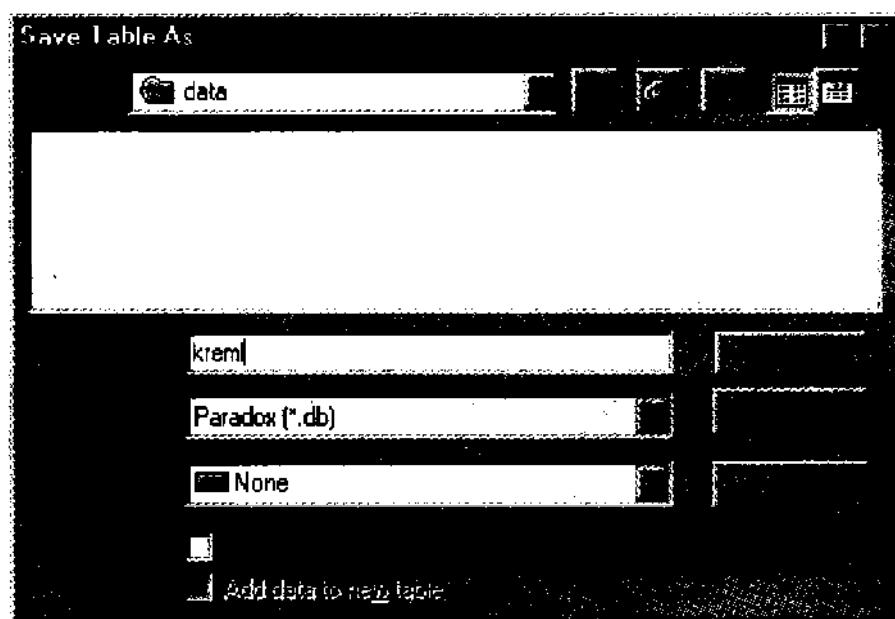


Рис. 5.15. Сохранение таблицы базы данных

Примечание. Создание базы данных и окна диалога сохранения других СУБД различаются с учетом особенностей различных СУБД.

Откройте созданную таблицу в окне Database Desktop командой File ► Open ► Table. Созданная таблица показана на рис. 5.16.

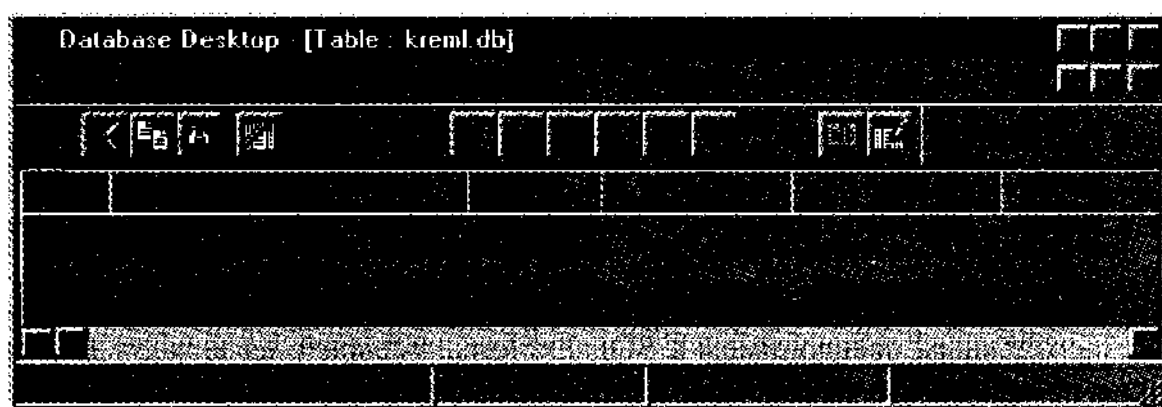


Рис. 5.16. Вид созданной таблицы в окне Database Desktop

Для ввода и редактирования данных в таблице можно выбрать в меню Database Desktop команду Table ► Edit или нажать клавишу F9. Установив курсор в нужную ячейку таблицы, введите значение поля. Нажатием клавиши Tab перейдите в следующее поле записи. Для добавления записи в таблицу выберите команду Record ► Insert. Остальные данные введите позднее в SQL Explorer.

Создание псевдонима базы данных. Прежде чем приступить к заполнению таблицы в SQL Explorer, создайте псевдоним базы данных, для чего запустите SQL Explorer, выбрав в меню Delphi команду Database ► Explore. Для создания псевдонима (alias) в меню SQL Explorer выберите команду Object ► New, затем в окне New Database Alias выберите тип драйвера STANDARD и щелкните мышью по кнопке OK. По умолчанию создаваемому псевдониму будет присвоено имя STANDARD1. Затем в правой части окна щелкните в строке PATH для выбора каталога с таблицами базы данных Paradox. В окне Select Directory выберите папку с файлами таблиц базы данных (рис. 5.17) и щелкните мышью по кнопке OK.

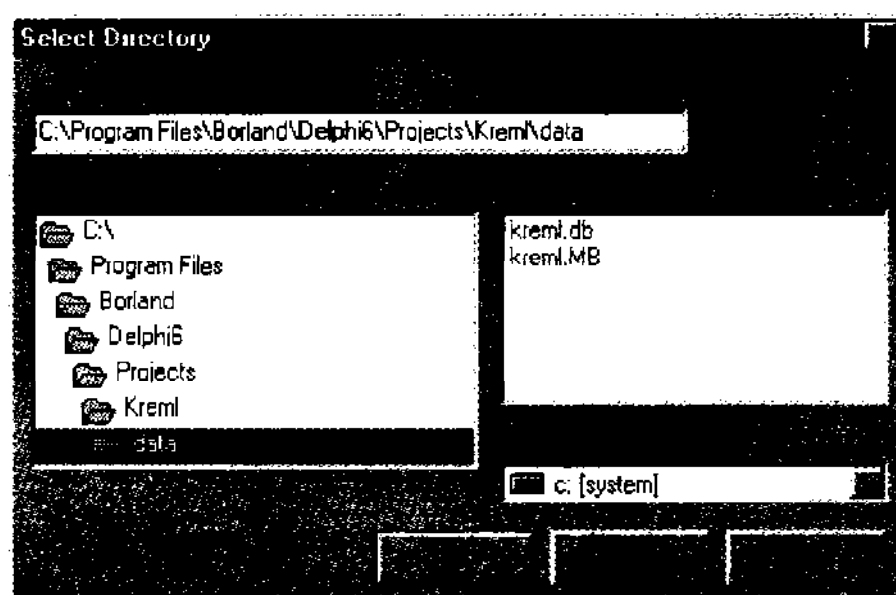


Рис. 5.17. Выбор каталога базы данных

Для сохранения созданного псевдонима базы данных выберите в меню SQL Explorer команду Object ► Save As, а затем в окне Save STANDARD1 as задайте имя псевдонима, например Monument, и щелкните мышью по кнопке OK.

Заполнение новой таблицы в SQL Explorer. Чтобы открыть таблицу базы данных с созданным псевдонимом в SQL Explorer, выберите на вкладке Databases в списке псевдонимов псевдоним Monument и, щелкнув мышью по символу «+», разверните список таблиц базы данных. Выбрав таблицу kreml.db, в правой части окна на вкладке Data введите данные (рис. 5.18).

Примечание. Для ввода данных в поля типа Memo и Graphics дважды щелкните левой кнопкой мыши, а затем введите в раскрытое окно текст или вставьте изображение из буфера обмена.

Завершите работу в SQL Explorer, выбрав в его меню команду Object ► Exit, и подтвердите сохранение данных, щелкнув мышью по кнопке Yes в окне сообщения Save all edits to SQL Explorer.

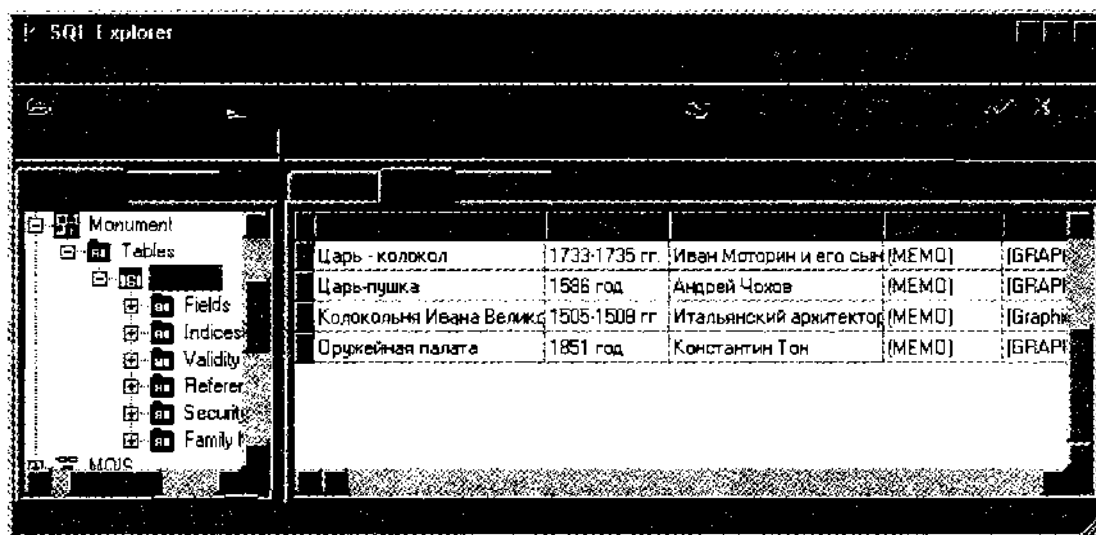


Рис. 5.18. Выбор базы данных и ввод данных в SQL Explorer

Создание формы приложения. Создайте форму и разместите на ней следующие компоненты: DataSource из палитры Data Access, Table из палитры BDE, DBNavigator из палитры DataControls.

Примечание. Компонент TDataSource представляет собой источник данных, который обеспечивает связь между набором данных и компонентами отображения и редактирования данных.

TTable – компонент набора данных, связанный с одной таблицей. Он обеспечивает прямой доступ к каждой записи и полю в одной указанной таблице базы данных.

TDBNavigator – компонент, предназначенный для перемещения по набору данных и редактирования данных. TDBNavigator имеет ряд кнопок, служащих для управления данными. Пользуясь свойством навигатора VisibleButtons, можно исключить любые ненужные в данном приложении кнопки.

Чтобы приложение с навигатором работало, надо установить его основное свойство DataSource – источник данных (имя компонента TDataSource).

Задайте для свойства Form1.Caption значение «Памятники Московского Кремля». Для связывания компонентов DataSource1, Table1 и DBNavigator1 в окне Инспектор объектов задайте следующие значения свойств:

```
DataSource1.Name = DataSource1;
DataSource1.DataSet = Table1;
Table1.DatabaseName = Monument (псевдоним базы данных)
Table1.TableName = kreml.db (имя базы данных);
Table1.Active = True (включить отображение данных);
DBNavigator1.Align = alBottom (отображать в нижней строке
окна);
DBNavigator1.DataSource = DataSource1.
```

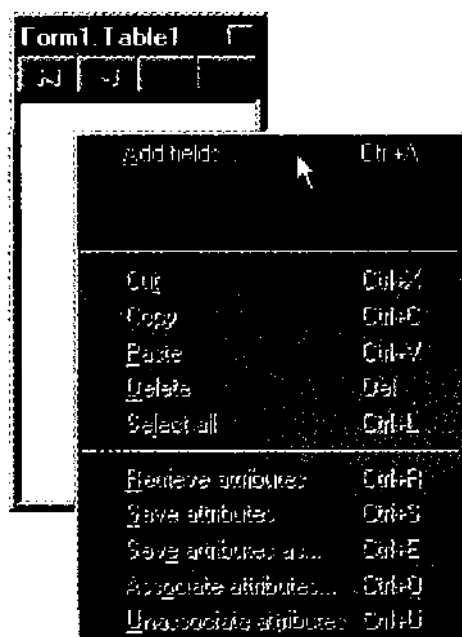


Рис. 5.19. Добавление полей записи базы данных в список отображаемых на форме

Для отображения полей записи базы данных в определенном месте формы, дважды щелкнув мышью по компоненту Table1, откройте окно Form1.Table1 со списком полей таблицы. Для отображения полей, щелкнув правой кнопкой мыши, в контекстном меню выберите Add fields (Добавить поля) – рис. 5.19.

Выбрав в списке Add fields поля, которые требуется отображать на форме, нажмите на кнопку ОК. Переместите каждое поле из списка в окне Form1.Table1 в определенное место на форме. Если для Table1.Active задано значение True, то в поле немедленно отображается значение соответствующего атрибута первой записи базы данных. Разместите и зафиксируйте положение визуальных компонентов с отображением полей записи базы данных, например, как показано на рис. 5.20.

Чтобы в окне с краткой информацией о памятнике можно было прочитать текст большой дли-

ны, используя вертикальную полосу прокрутки, для объекта DBMemo1 задайте свойство ScrollBars=ssVertical.

Используя свойство Caption объектов Label 1–5, измените тексты подписей к полям базы данных: вместо «Name» введите «Название», вместо «Year» – «Годы создания», вместо «Author» – «Авторы», вместо «Comment» – «Краткая справка», вместо «Foto» – «Фотография».

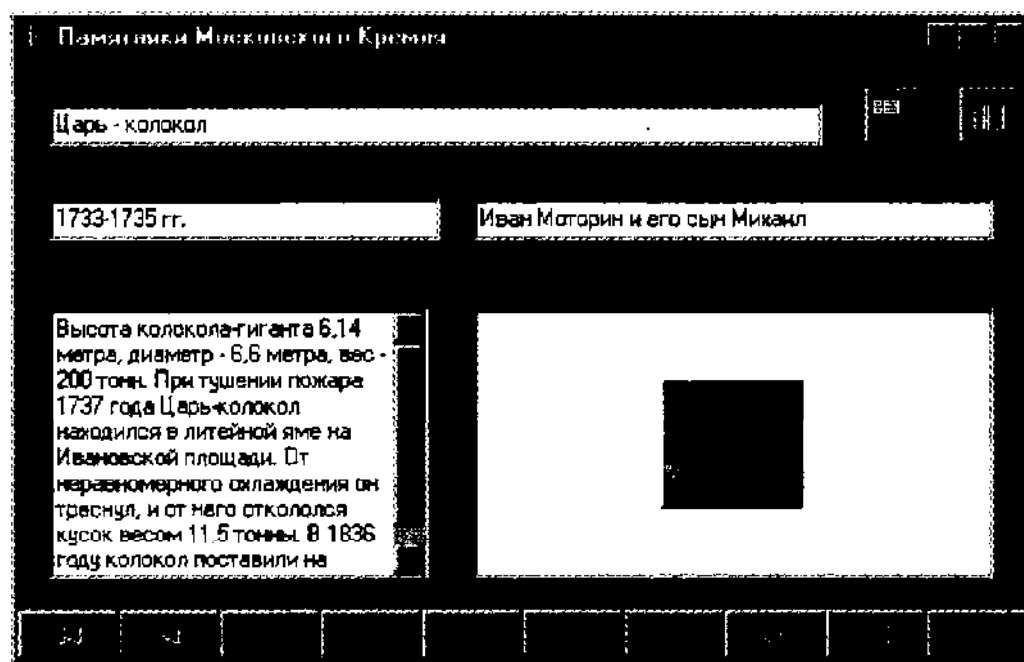


Рис. 5.20. Вид формы приложения для работы с базой данных

Сохраните проект, откомпилируйте его и запустите приложение. Щелкая мышью по кнопкам DBNavigator (рис. 5.21), проверьте переход к первой, предыдущей, следующей, последней записи. Щелкнув мышью по кнопке `nbInsert` со знаком `+`, вставьте новую запись перед текущей и введите данные, например, об Успенском соборе: «Успенский собор был построен в 1479 г. На протяжении шести столетий собор являлся государственным и культовым центром России: здесь поставляли великих князей, а удельные присягали им на верность, венчали на царство, короновали императоров. В Успенском соборе возводили в сан епископов, митрополитов и патриархов, оглашали государственные акты». Фотографию собора вы можете найти в Интернете.

Используя Image Editor, создайте файл значка приложения; для этого в меню Image Editor выберите команду `File ▶ New ▶ Icon File(*.ico)`. В окне `Icon Properties` (Свойства значка) выберите `Size: 32 × 32`, `Colors: 16 color`. Нарисуйте в редакторе значок, например изображение Царь-колокола. Сохраните созданный значок под именем `kreml.ico` в папке проекта. Закройте окно Image Editor.

Выбрав в окне Инспектор объектов объект `Form1`, щелчком в поле значений свойства `Icon` откройте окно `Picture Editor`. Щелкните мышью по кнопке `Load`, в окне `Load Picture` выберите требуемый файл значка и щелкните по кнопке `Открыть`. Просмотрев изображение значка, нажмите кнопку `ОК`. Вид значка на форме изменится. Чтобы изменить значок приложения, выберите в меню Delphi команду `Project ▶ Options`. На вкладке `Application`, щелкнув на кнопке `Load Icon`, загрузите созданный значок.

Откройте окно Редактор кода и убедитесь, что в процессе проектирования формы в окне Редактор кода был сгенерирован код программы.

Чтобы соединение и разрыв соединения с базой данных выполнялись в начале и окончании работы приложения соответственно, задайте для свойства `Active` объекта `Table1` значение `False` и создайте процедуры обработки событий `OnCreate` и `OnDestroy` для объекта `Form1`. В тело процедуры обработчика события создания формы добавьте оператор `Table1.Active:= True;`, а в тело обработчика события уничтожения формы – оператор `Table1.Active:= False;`.

После этого текст программного модуля будет выглядеть следующим образом:

```
unit Main;  
  
interface  
  
uses  
  
    Windows, Messages, SysUtils, Variants, Classes, Graphics,  
Controls, Forms, Dialogs, DB, DBTables, ExtCtrls, DBCtrls,  
StdCtrls, Mask;  
  
type  
  
    TForm1 = class(TForm)
```

```
DBNavigator1: TDBNavigator;
Table1: TTable;
DataSource1: TDataSource;
Label1: TLabel;
DBEdit1: TDBEdit;
Label2: TLabel;
DBEdit2: TDBEdit;
Label3: TLabel;
DBEdit3: TDBEdit;
Label4: TLabel;
DBMemo1: TDBMemo;
Label5: TLabel;
DBImage1: TDBImage;
Table1Name: TStringField;
Table1Year: TStringField;
Table1Author: TStringField;
Table1Comment: TMemoField;
Table1Foto: TGraphicField;

procedure FormCreate(Sender: TObject);
procedure FormDestroy(Sender: TObject);

private
    { Private declarations }
public
    { Public declarations }

end;

var
    Form1: TForm1;

implementation

{$R *.dfm}
```

```
procedure TForm1.FormCreate(Sender: TObject); {Создание формы}
begin
    Table1.Active:= True;           {Соединение с базой данных}
end;

procedure TForm1.FormDestroy(Sender: TObject); {Уничтожение
формы}
begin
    Table1.Active:= False; {Разрыв соединения с базой данных}
end;
end.
```

Сохраните изменения в проекте, откомпилируйте и запустите приложение. Используя кнопки DBNavigator (рис. 5.21), проверьте работу приложения.

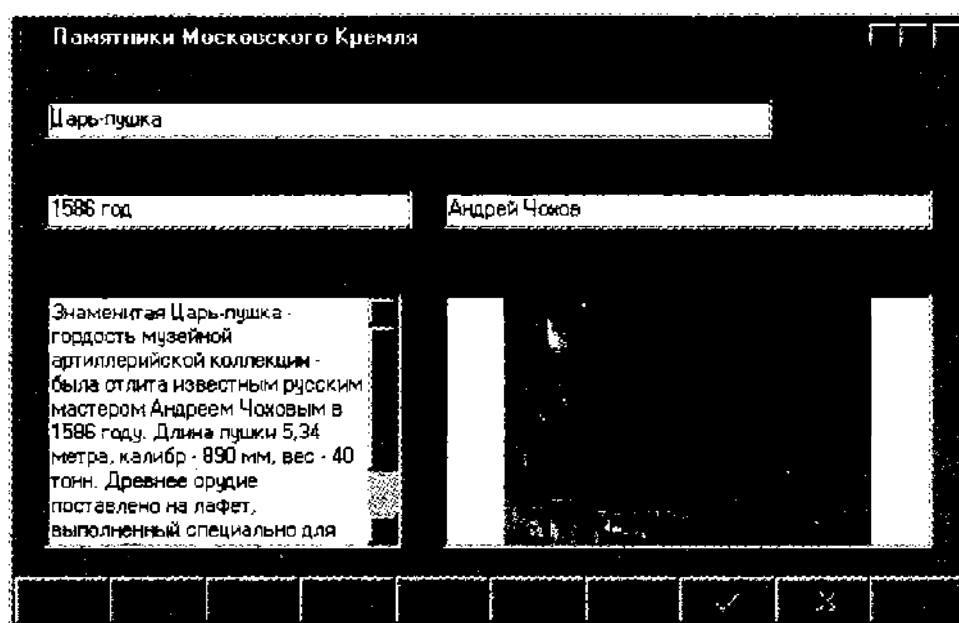


Рис. 5.21. Вид окна приложения для работы с базой данных

Конечно, созданное в данном упражнении простейшее приложение иллюстрирует лишь малую часть возможностей Delphi по разработке приложений для работы с базами данных.

Создание справочной системы

Каждое качественное приложение должно иметь справочную систему. Справка приложения может иметь самый различный вид – обыкновенный текстовый файл, файл в формате WinHelp, файл в формате PDF или файл в формате HTML Help.

Разработка справочной системы требует решения следующих основных задач.

1. Планирование системы справок. На этом этапе составляется перечень разделов справочной службы и необходимых перекрестных ссылок.

2. Создание текстовых файлов, содержащих описания справочных разделов. Текстовые файлы готовятся с помощью любого текстового редактора, поддерживающего расширенный текстовый формат RTF, например Microsoft Word. В них включаются специальные управляющие символы для создания перекрестных ссылок и подключения растровых изображений.

3. Разработка проектного файла, содержащего специальные команды для Help-компилятора. Проектный файл описывает структуру справочной службы в целом, в нем каждому разделу присваивается уникальный целочисленный идентификатор. Проектный файл служит основным управляющим документом для Help-компилятора. В Windows 32 он создается с помощью утилиты Microsoft Help Workshop и представляет собой ASCII-текстовый файл, содержащий несколько секций.

4. Редактирование файла содержания, которое активизируется при запуске HLP-файла, а также после щелчка по кнопке Содержание в окне справочной службы.

5. Компиляция Help-файлов. Для создания справочных руководств можно использовать компиляторы фирмы Microsoft, которые можно загрузить с сайта Microsoft. Microsoft Help Workshop требуется при создании справочных HLP-файлов для Windows, Microsoft HTML Help Workshop требуется при создании HTML Help (CHM-файлы).

6. Тестирование и отладка Help-службы.

7. Связь приложения с разделами Help-службы.

Одной из лучших программ для создания справочной системы является программа Help & Manual компании EC Software. Эта программа является средством WYSIWYG-типа и поддерживает все стандартные справочные форматы. После инсталляции Help & Manual интегрируется в ICP Delphi, что позволяет для ее вызова выбрать в меню ICP Delphi команду Tools ► Help & Manual Project Wizard. Интеграция Help & Manual в ICP Delphi позволяет создавать качественные справочные системы и руководства одновременно с проектированием приложения.

Создание нового проекта справки. Для создания нового проекта справки выберите в меню Help & Manual команду File ► New. В диалоговом окне New Help & Manual project выберите в блоке вариантов Create a new Help & Manual project пункт This is a new help project и нажмите кнопку Next, затем введите название и путь нового проекта справки. Выберите в блоке взаимоисключающих вариантов Template: Use the built-in default template – или в качестве шаблона уже существующий проект Help & Manual, но для первого проекта лучше эту возможность не использовать. Нажав кнопку Next, перейдите к следующему шагу работы мастера. В окне быстрого старта Quick start можно отредактировать несколько заголовков разделов справки (рис. 5.22).

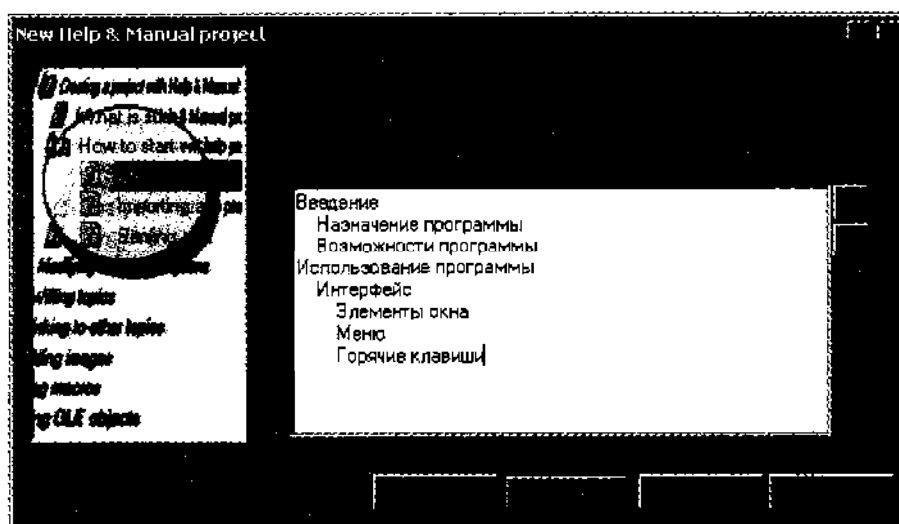


Рис. 5.22. Редактирование заголовков разделов справочной системы

Help & Manual создаст заготовку оглавления справки на их основе. Позже их всегда можно будет изменить. Для завершения процесса создания нового проекта нажмите кнопку Finish и подтвердите сохранение проекта. После этого в окне Help & Manual откроется проект справочной системы, в который вы можете внести текст и изображения.

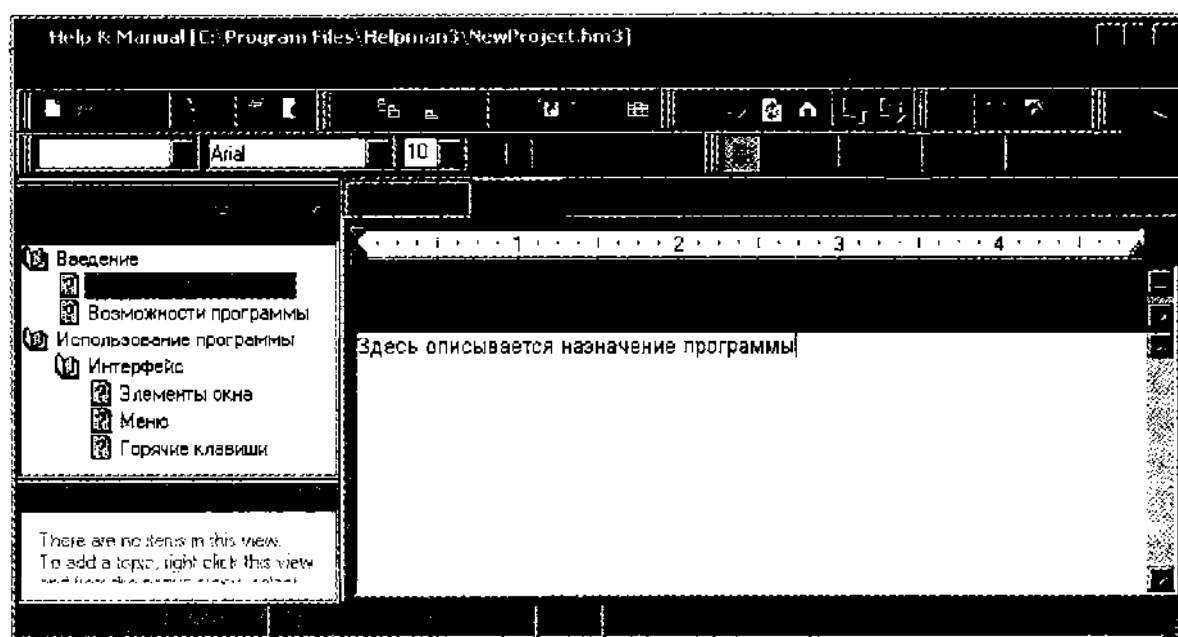


Рис. 5.23. Окно Help & Manual с проектом справочной системы

Элементы окна Help & Manual. Как показано на рис. 5.23, окно программы Help & Manual представляет собой трехконную рабочую область, содержащую инструментальные панели, оглавление и текстовый редактор. Инструментальные панели расположены в верхней части окна программы. В левой части окна расположена панель Оглавление (Table of Contents), на которой представлен древовидный список глав и разделов. Ниже панели Оглавление расположена панель Невидимые разделы (Invisible Topics), в которой

показан список разделов, требующихся для правильной работы справочной системы, но никогда не отображающихся непосредственно в ее главном окне. Большинство разделов, перечисленных здесь, — контекстно-зависимые подсказки.

В правой части окна расположена панель текстового редактора с двумя вкладками: *Topic Options* и *Help text*. При выборе раздела в оглавлении или в списке невидимых разделов данное окно становится доступным, и на вкладке *Help text* отображается текст раздела создаваемой справочной системы. Вкладка *Topic Options* (Параметры раздела) обеспечивает доступ к некоторым параметрам настройки выбранного раздела. В поле *Topic ID* определяется алфавитно-цифровой идентификатор раздела справки. В поле *Help Window* задается тип окна раздела. Например, тип *Main* является встроенным и представляет собой стандартное окно средства просмотра справки. В поле *Keywords* вводятся ключевые слова, которые будут выводиться в виде отсортированного списка на вкладке предметного указателя *Index* готового справочного файла и использоваться пользователем при поиске нужных разделов по ключевым словам. Поле *A-Key words* используется для определения ассоциативных ключевых слов, т.е. ключевых слов, которые никогда не появляются в поисковом предметном указателе файла справки. В поле *Help Context Number* задается уникальный числовой идентификатор раздела, который используется для вызова контекстно-зависимой помощи. В списке *Builds which include this topic* определяется, какой компилятор выходных форматов поддерживает данный раздел. Использование программы *Help & Manual* облегчается мощной системой контекстно-зависимой помощи.

Изменение параметров проекта справочной системы. При создании нового проекта *Help & Manual* назначает всем его параметрам и настройкам значения по умолчанию. В частности, языком справочного проекта по умолчанию является английский. Правильный выбор языка обеспечивает корректную сортировку предметного указателя (*Index*) ключевых слов в компилируемом справочном файле. При написании справочных файлов на кириллице необходимо также изменить набор символов, иначе шрифты в созданной справочной системе будут отображаться неправильно.

Для изменения параметров и набора символов проекта справки выберите пункт главного меню *Project ► Project Properties*. Раскрыв раздел *Common Settings*, выберите подраздел *Title and Copyright*, в котором задайте заголовок проекта, принадлежность авторских прав, раздел по умолчанию и краткое описание проекта для внутреннего использования. Открыв подраздел *Language and Character Sets*, выберите в поле *Language of help file* вариант *Russian*, чтобы задать в качестве языка проекта по умолчанию русский язык. Щелкнув на кнопке *Apply*, сохраните внесенные в проект изменения. Щелкнув на кнопке *OK*, закройте окно *Project Properties*.

Компиляция справочного файла. Для компиляции справочного файла выберите в меню *File ► Make Help File and Run...* В окне *Make Help File and Run...* выберите в списке *Output Format* вариант *Winhelp*, в поле *Output File* задайте имя файла справки и щелкните на кнопке *OK* для начала компиляции

проекта. После окончания компиляции в окне Compiler Messages будет выведено сообщение о результатах компиляции справки и откроется окно справки (рис. 5.24).

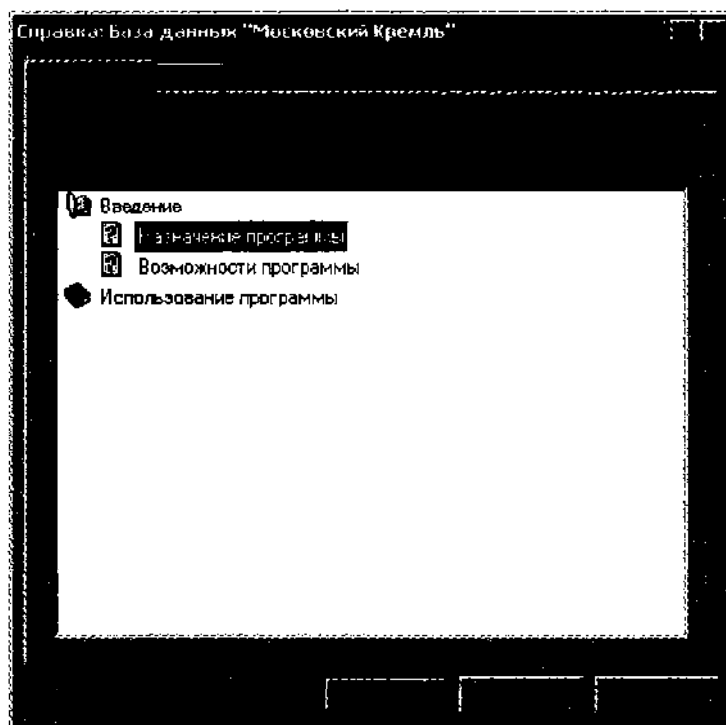


Рис. 5.24. Окно справки приложения

Используя Help & Manual, вы можете отредактировать содержание справки.

Подключение справки к приложению. Для подключения справки к приложению выберите в меню ИСР Delphi команду Project ► Options, затем в окне Project Options откройте вкладку Application и в поле Help file укажите путь к файлу справки. Щелкнув на кнопке ОК, завершите определение пути к файлу справки.

Организация контекстной справки. Для организации вызова контекстной справки откройте проект справки в окне Help & Manual, затем откройте вкладку Topic options. Выбирая в списке разделов справки нужный раздел (рис 5.25), в поле Help Context Number введите уникальный номер, в поле Keywords введите одно или более ключевых слов (для каждого используется отдельная строка) для образования при компиляции справочного файла предметного указателя (Index) справки. Сохранив изменения в проекте, откомпилируйте его.

Для подключения контекстной справки к конкретному компоненту приложения укажите компонент на форме приложения, а в окне Инспектор объектов дважды в строке свойства HelpContext укажите номер раздела справки и щелкните по кнопке с тремя точками. После этого в окне HelpContext Property Editor укажите в поле Help&Manual Project File имя файла проекта справки, выберите в группе Action вариант Connect to Existing Topic (Подключить к существующему разделу справки) и в списке выберите раздел справки (см. рис. 5.25).

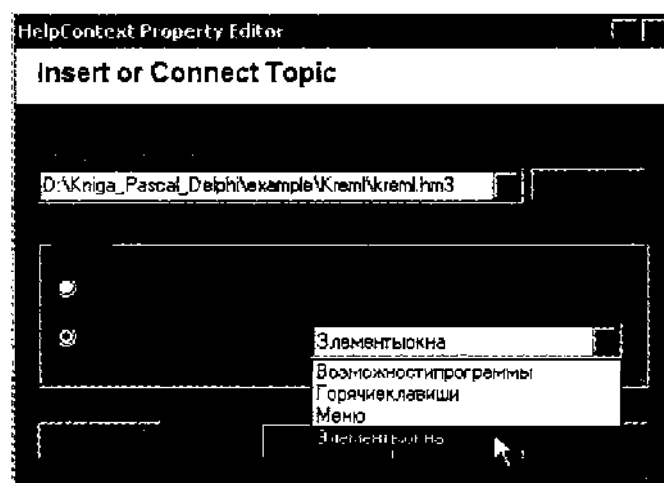


Рис. 5.25. Подключение контекстной справки

Щелкнув по кнопке OK, откройте окно Help & Manual, отредактируйте содержание справки данного раздела, сохраните и откомпилируйте проект справки. Теперь для вызова контекстной справки при запуском приложении нужно будет указать компонент, для которого имеется раздел справки, и нажать клавишу F1.

Итак, уважаемый читатель, мы с вами сделали первые шаги в освоении одной из современных сред разработки приложений. Успехов вам в дальнейшем совершенствовании знаний возможностей Delphi и приобретении практических навыков разработки приложений!

Что нового вы узнали?

В этой главе вы познакомились с понятием «консольное приложение».

Изучив эту главу, вы узнали:

- назначение и порядок использования функций CharToOem, WinExec, ShellExecute, GetSystemDirectory, GetWindowsDirectory, CreateProcess;
- назначение, свойства и методы компонентов RichEdit, FontDialog, FindDialog, ReplaceDialog, ActionList, DataSource, Table, DBNavigator;
- порядок разработки справочной системы Windows-приложения и интерфейс программы Help & Manual, используемой для создания справочных систем и руководств пользователя.

Вы научились:

- использовать интегрированную среду Delphi для создания многооконного текстового редактора;
- программировать следующие взаимодействия Windows-приложения с внешними программами: непосредственный запуск внешней программы из приложения, запуск внешней программы, связанной с некоторым документом, обмен сообщениями с другими программами, внедрение и связывание документов, подготовленных внешними программами, с вашим приложением, динамический обмен данными между приложениями;
- использовать поставляемую в комплекте с Delphi программу Database Desktop для создания таблиц базы данных;

- использовать поставляемую в комплекте с Delphi программу SQL Explorer для создания псевдонима базы данных и заполнения таблиц базы данных;
- использовать интегрированную среду Delphi для создания приложения, обслуживающего реляционную базу данных, обеспечивая просмотр, добавление и удаление записей;
- использовать программу Help & Manual для разработки справочной системы, имеющей контекстный характер.

Задания

1. Создайте приложение Таймер, которое по заданному интервалу времени выполняет обратный отсчет и по его окончании выводит на экран сообщение «Время истекло», а также звуковой сигнал.

2. Создайте приложение с отображением окна заставки при запуске приложения.

3. Создайте приложение Автомат школьных звонков, которое обеспечивает ввод данных о времени начала и окончания уроков в учебном заведении, длительности перемен и автоматически выдает звуковые сигналы – звонки на урок и с урока, а на экране приложения отображает текущее время и состояние: номер урока или перемены.

4. Создайте приложение Таблица Менделеева, которое отображает периодическую таблицу химических элементов и по щелчку на ячейке химического элемента выводит в окне модального типа подробную информацию о нем.

ГЛАВА 6

Создание приложения «Школьная библиотека»

В этой главе мы рассмотрим технологию разработки проекта с программированием взаимодействия Windows-приложения с базой данных Microsoft Access, документами Microsoft Word и Microsoft Excel на примере создания приложения «Школьная библиотека». Предполагается, что читатель знаком с основными понятиями реляционных баз данных и использованием Microsoft Access для создания базы данных.

После изучения главы вы будете знать:

- Порядок создания Windows-приложения, обслуживающего базу данных Microsoft Access и взаимодействующего с MS Word, MS Excel.
- Использование реестра Windows для хранения параметров приложения.
- Использование компонентов для связи с базами данных через Active Data Objects (ADO).
- Использование компонента TPageControl для создания многостраничных панелей с закладками.

Создание базы данных Microsoft Access

Упражнение 6.1. Создайте базу данных Library, состоящую из двух таблиц с данными о читателях и книгах.

Пусть таблица с данными о читателях называется «Reader» и имеет следующую структуру:

Название поля	Назначение	Тип данных	Размер поля (формат)
Num	Номер	Числовой	Длинное целое
Fam	Фамилия	Текстовый	50
Name	Имя	Текстовый	50
MidName	Отчество	Текстовый	50
Teacher	Учитель	Логический	Да/Нет
Class	Класс	Текстовый	10
Adress	Адрес	Текстовый	50
Tel	Телефон	Текстовый	20
Books	Книги	Текстовый	50

Пусть таблица с данными о книгах называется «Books» и имеет следующую структуру:

Название поля	Назначение	Тип данных	Размер поля (формат)
Num	Инвентарный номер	Числовой	Длинное целое
Name	Название книги	Текстовый	50
Author	Автор	Текстовый	50
Publishing	Издательство	Текстовый	50
Year	Год издания	Числовой	Длинное целое
Pages	Количество страниц	Числовой	Длинное целое
Price	Цена	Числовой	Одинарное с плавающей точкой
Cupboard	Место хранения	Текстовый	50
Comment	Аннотация	Поле MEMO	
ReaderNo	Номер читателя	Числовой	Длинное целое

Выполните следующие действия.

1. Для создания базы данных запустите Microsoft Access и, выбрав в меню Файл команду Создать, выберите в области задач вариант Новая база данных. В окне Файл новой базы данных выберите папку, в которой будет создаваться новая база данных, например C:\Library\Data, в поле Имя файла задайте имя базы данных, например Library, и щелкните на кнопке Создать.

2. Для создания таблицы «Reader» с данными о читателях выберите в списке вкладок в левой части окна базы данных вкладку Таблица. После этого в окне базы данных будут отображены ярлыки вариантов создания таблицы: в режиме конструктора, с помощью мастера и путем ввода данных. Дважды щелкнув мышкой по строке «Создание таблицы в режиме конструктора», откройте окно таблицы в режиме конструктора.

В верхней части окна находится создаваемый или модифицируемый макет таблицы, который представляет собой просто список полей с указанием имени поля, типа данных и описания.

3. В столбце Имя поля введите имя поля, а в следующем столбце укажите тип данных для этого поля. Тип данных можно выбрать из раскрывающегося списка. Как только курсор оказывается в столбце Тип данных, в нижней части окна возникает бланк Свойства поля (характеристики данного поля). Он представляет собой перечень свойств (слева – название свойства, справа – значение этого свойства) с окном подсказки по каждому свойству. Перечень свойств меняется в зависимости от типа данных, который в текущий момент отображается в столбце Тип данных. Щелкнув мышью на поле значения в бланке свойств, вы можете изменить это значение (в рамках допустимого для этого типа данных).

Большинство значений принимается системой по умолчанию, многие свойства можно изменить самостоятельно. Некоторые значения можно выбрать из раскрывающегося списка, как показано на рис.6.1.

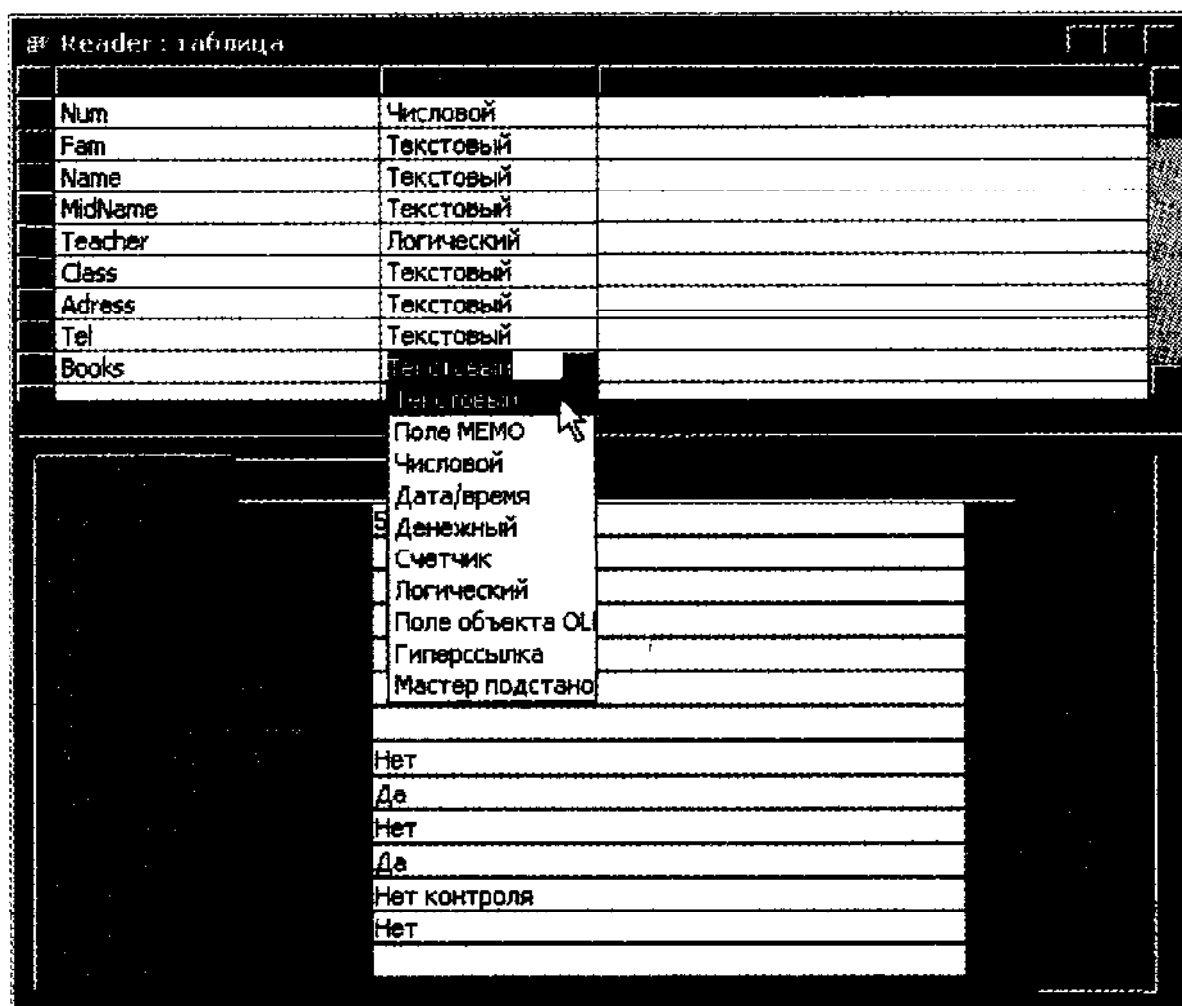


Рис. 6.1. Определение структуры таблицы

Примечание. При выборе значения свойства принципиально важно следовать следующим рекомендациям. Для текстового и числового поля надо указать размер поля, причем для текста – это допустимая длина значения (например, 20 или 40 символов), а для числа – формат представления в компьютере (байт, целое (два байта), длинное целое и т. д.). Для поля Дата/время обязательно надо указать формат, чтобы система знала, как обрабатывать вводимые данные. Например, если выбрать Краткий формат даты, система будет ожидать от вас ввода именно даты (в русской версии – ДД.ММ.ГГГГ), а если выбрать Краткий формат времени, в этом поле придется набирать ЧЧ:ММ (часы и минуты).

В качестве значения свойства «Условие на значение» вы можете указать логическое выражение, которое должно принимать значение True («Истина») при вводе данных в это поле. В следующем свойстве можно записать произвольное сообщение об ошибке, которое будет выдано системой, например: «Это значение поля недопустимо». В свойстве «Обязательное поле» можно указать «Да» (пустые значения не допускаются) или «Нет» (пустые значения допускаются).

Итак, следуя вышеприведенным рекомендациям, определите поля таблицы Num, Fam, Name, MidName, Teacher, Class, Adress, Tel, Books.

4. Действуя аналогично п. 2–3, создайте таблицу с данными о книгах «Books».

5. Завершите работу Microsoft Access, сохранив таблицы базы данных Library.

Создание приложения выбора режима работы

Упражнение 6.2. Создайте новое приложение, задайте форме Form1 заголовков «Выбор режима работы», разместите на форме компоненты RadioGroup, Label, Button и Edit, как показано на рис. 6.3. Создайте процедуры обработки событий показа формы, щелкнув по кнопкам, и проверки правильности ввода пароля для режима Библиотекарь. Сохраните проект под именем Library-School в папке C:\Library. Выполните следующие действия.

1. Для создания новой формы выберите в меню File команду New-Form. Выбрав в окне Object TreeView объект Form1, в окне Инспектор объектов выберите свойство Form1.Caption и задайте ему значение «Выбор режима работы».

2. Задайте свойству RadioGroup1.Caption значение «Выберите режим».

3. Чтобы задать компоненту RadioGroup два значения режимов работы Библиотекарь и Читатель, щелкните в окне Инспектор объектов на кнопке с тремя точками в поле значения RadioGroup1.Items и в окне String List Editor введите текст режимов, как показано на рис. 6.2.

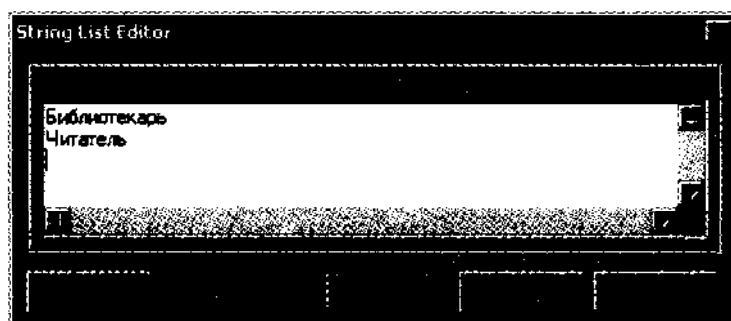


Рис. 6.2. Ввод значений режимов работы

4. Задайте значения заголовков кнопок Button1 – «Вход», Button2 – «Отменить». После этих операций форма Выбор режимов работы будет выглядеть, как показано на рис. 6.3.

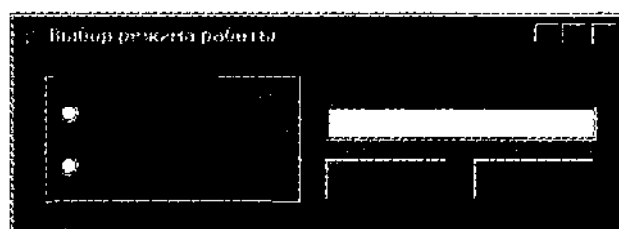


Рис. 6.3. Форма Form1 «Выбор режима работы»

5. Сохраните новый проект, выбрав в меню File команду Save Project As, а затем в окне Save Unit1 As (Сохранить модуль как) выберите папку, например C:\Library, и щелкните по кнопке Сохранить для сохранения текста программного модуля. После этого в окне Save Project1 As в поле Имя файла задайте название проекта (например, Library-School) и щелкните по кнопке Сохранить.

6. Чтобы создать процедуру обработки события показа формы Form1, выберите в окне Инспектор объектов для объекта Form1 вкладку Events (События) и дважды щелкните в поле значений события OnShow. После этого отредактируйте заготовку процедуры следующим образом:

```
procedure TForm1.FormShow(Sender: TObject);
begin
    Radiogroup1.ItemIndex := -1; {Нет, задан определенный выбор}
    Edit1.Visible := false; {Невидимое поле}
    Label1.Visible := false; {Невидимый текст}
    Edit1.Text := ''; {Очистка поля}
end;
```

7. Чтобы создать программу обработки события щелчка на кнопке Отменить, дважды щелкнув по этой кнопке, откройте окно редактора кода и дополните заготовку процедуры обработчика события, так чтобы текст процедуры был следующим:

```
procedure TForm1.Button2Click(Sender: TObject);
begin
    Form1.Close;
end;
```

8. Чтобы создать процедуру обработки события выбора режима работы, дважды щелкните по компоненту RadioGroup1 и отредактируйте следующим образом заготовку процедуры:

```
procedure TForm1.RadioGroup1Click(Sender: TObject);
begin
    case Radiogroup1.ItemIndex of
    0: begin {Режим Библиотекарь}
        Edit1.Visible := true; {Отобразить поле ввода пароля}
        Label1.Visible := true;
        Label1.Caption := 'Пароль';
        end;
    1: begin {Режим Читатель}
        Edit1.Visible := false; {Скрыть поле ввода пароля}
```

```

    Label1.Visible := false;
    Edit1.Text := '';
end;
end;
end;
```

9. Для создания процедуры обработчика щелчка по кнопке Вход и проверки правильности введенного пароля в случае выбора режима «Библиотекарь» дважды щелкните на кнопке Вход, а затем дополните заготовку процедуры следующим образом:

```

procedure TForm1.Button1Click(Sender: TObject);
var
  f: TextFile;
  FileName: string;
  I,K,L: integer;
  PKey,RKey: string;
  C: Char;
begin
  if Radiogroup1.ItemIndex = -1 then ShowMessage('Пожалуйста,
выберите режим!!')else
  if Radiogroup1.ItemIndex = 1 then
    begin
      {Выбран режим Читатель}
      Edit1.Visible := false;
      Label1.Visible := false;
      Edit1.Text := '';
      Form2.Show;
      {Переход в режим Читатель}
    end
  else
    {Выбран режим Библиотекарь}
    begin
      Pass := TRegistry.Create;
      {Создание записи о пароле в
реестре Windows}
      Pass.RootKey := HKEY_LOCAL_MACHINE;
      if not Pass.OpenKey('\software\Library',false) then
        begin
          {Если пароль не записан в реестре Windows}
          Pas:= 'book';
          {Первоначальное значение пароля для режима
Библиотекарь}
          if Edit1.Text = Pas then
            begin
              Pass.CreateKey('\software\Library');
            end
          else
            begin
              Pass.SetValue('Password',Pas);
            end
          end
        end
      else
        begin
          Pass.SetValue('Password',Pas);
        end
      end
    end
  end
end

```

```
Pass.OpenKey('\software\Library',true);
Pas := '';
for I := 1 to Length(Edit1.Text) do {Кодирование пароля}
begin
  C := Edit1.Text[I];
  Pas := Pas + IntToStr(Ord(C));
end;
PKey := '';
for I := 1 to length(Pas) do
begin
  Val(Pas[I],K,L);
  if K > 5 then K := 4 - (9-K) - 1 else K := K + 4;
  PKey := PKey + IntToStr(K);
end;
Pass.WriteString('P',PKey); {Запись в реестр значения пароля}
end
else
  ShowMessage('Неправильный пароль! Вам не разрешено работать в
этом режиме. ');
end
else {Если пароль записан в реестре Windows}
begin
  Pas := '';
  for I := 1 to Length(Edit1.Text) do {Преобразовать его в
цифровой код}
begin
  C := Edit1.Text[I];
  Pas := Pas + IntToStr(Ord(C));
end;
PKey := '';
for I := 1 to length(Pas) do
begin
  Val(Pas[I],K,L);
  if K > 5 then K := 4 - (9-K) - 1 else K := K + 4;
  PKey := PKey + IntToStr(K);
end;
Pass.OpenKey('\software\Library',true);
RKey := Pass.ReadString('P');
  if PKey <> RKey then {Сравнение введенного пароля и
сохраненного в реестре}
```

```
    ShowMessage('Неправильный пароль!')
else
begin
    Form1.Hide;
    Edit1.Text := '';
    Form3.Show;    {Переход в режим Библиотекарь}
end;
end;
end;
end;
```

10. Сохраните изменения в проекте и модуле, щелкнув на кнопке Save All на стандартной панели инструментов интегрированной среды программирования Delphi.

Создание формы с компонентами доступа к базе данных MS ACCESS

Упражнение 6.3. Создайте форму и разместите на ней компоненты для доступа к базе данных Library и ее таблицам, как показано на рис. 6.4. Для этого выполните следующие действия.

1. Для создания новой формы выберите в меню File команду New-Form. В окне Инспектор объектов задайте свойству Name данной формы значение DataModule1.

2. Из палитры компонентов ADO (ActiveX Data Objects) разместите на форме DataModule1 компоненты ADOConnection1, ADOTable1 и ADOTable2.

Примечание. Палитра компонентов ADO содержит компоненты для связи с базами данных через Active Data Objects (ADO) – множество компонентов ActiveX, использующих для доступа к информации баз данных Microsoft OLEDB. Компонент ADOConnection используется для связи с набором данных ADO. Он обеспечивает доступ к хранилищу данных компонентам ADO, инкапсулирующим набор данных. Компонент ADOTable используется для связи с таблицей базы данных.

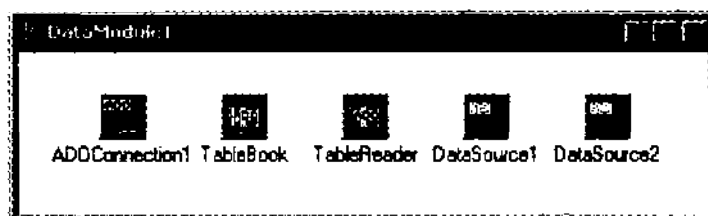


Рис. 6.4. Форма DataModule1

3. Для описания пути доступа к информации базы данных Library выберите объект ADOConnection1, в окне Инспектор объектов щелкните левой кнопкой мыши по кнопке с тремя точками в поле значений свойства Connection-

String. После этого в окне `DataModule1.ADOConnectionString` выберите опцию `Use Connection String` (Использование строки связи) и щелкните по кнопке `Build` (Строить).

В окне Свойства связи с данными, щелкнув на кнопке с тремя точками в поле 1, выберите или введите имя базы данных, откройте окно выбора базы данных. В окне Выберите базу данных Access, выбрав папку и файл базы данных, щелкните по кнопке Открыть.

Щелкнув по кнопке ОК в окне Свойства связи с данными, завершите формирование строки связи с базой данных. Щелкнув по кнопке ОК в окне `DataModule1.ADOConnectionString`, завершите редактирование свойства `ADOConnection1.ConnectionString`.

4. Выберите компонент `ADOTable1` и задайте его свойству `Name` значение `TableBook`. Для описания связи приложения с таблицей базы данных «Books» задайте свойству `Connection` значение `ADOConnection1`.

5. Выберите компонент `ADOTable2` и задайте его свойству `Name` значение `TableReader`. Для описания связи приложения с таблицей базы данных «Reader» задайте свойству `Connection` значение `ADOConnection1`.

6. Из палитры компонентов Data Access разместите на форме `DataModule1` компоненты `DataSource1` и `DataSource2`.

Примечание. Компонент `DataSource` используется для соединения наборов данных с компонентами, отображающими данные.

7. Для соединения набора данных с компонентами, отображающими данные в приложении, выбрав компонент `DataSource1`, задайте в окне Инспектор объектов его свойству `DataSet` значение `TableBook`. Действуя аналогично, задайте свойству `DataSource2.DataSet` значение `TableReader`.

После создания формы `DataModule1` и размещения на ней вышеуказанных компонентов в окне Редактор кода будет создан следующий код модуля:

```
unit Unit5;
interface
uses
    SysUtils, Classes, DB, ADODB;
type
    TDataModule1 = class(TDataModule)
        ADOConnection1: TADOConnection;
        DataSource1: TDataSource;
        TableBook: TADOTable;
        TableBookNum: TIntegerField;
        TableBookName: TWideStringField;
        TableBookAuthor: TWideStringField;
        TableBookPublishing: TWideStringField;
        TableBookYear: TIntegerField;
        TableBookPages: TIntegerField;
```

```
TableBookCupboard: TWideStringField;
TableBookComment: TMemoField;
TableReader: TADOTable;
TableReaderNum: TIntegerField;
TableReaderFam: TWideStringField;
TableReaderName: TWideStringField;
TableReaderMidName: TWideStringField;
TableReaderTeacher: TBooleanField;
TableReaderClass: TWideStringField;
TableReaderAdress: TWideStringField;
TableReaderTel: TWideStringField;
TableReaderBooks: TWideStringField;
TableBookReaderNo: TIntegerField;
DataSource2: TDataSource;
TableBookPrice: TFloatField;
private
  { Private declarations }
public
  { Public declarations }
end;
var
  DataModule1: TDataModule1;
implementation
  {$R *.dfm}
end.
```

8. Сохраните изменения в проекте и модуле, щелкнув по кнопке Save All на стандартной панели инструментов интегрированной среды программирования Delphi.

Создание формы и обработчиков поиска книг в режиме Читатель

Упражнение 6.4. Создайте форму, которую будет принимать окно приложения при поиске книг в режиме Читатель, и разместите на ней компоненты, как показано на рис. 6.5. Создайте процедуры обработки событий щелчков на кнопках Очистить, Найти. Для этого:

1. Выбрав в меню File команду New-Form, добавьте в проект форму.
2. Задайте свойству Caption значение «Режим читатель: поиск книги».
3. Из палитры компонентов Standard разместите на форме Form2 компоненты: Button1, Button2, Edit1, Edit2, Edit3, Edit4, Edit5, Edit6, Edit7, Memo, DBNavigator1.
4. Задайте свойства Caption объектов Button1, Button2 Очистить и Найти соответственно. Свойствам Text объектов Edit1, Edit2, Edit3, Edit4, Edit5, Edit6, Edit7 задайте пустые значения.

5. Чтобы задать пустое значение свойству Lines объекта Memo, дважды щелкните на кнопке с тремя точками в поле значения данного свойства и, очистив содержимое окна String List Editor, щелкните по кнопке OK.

6. Чтобы связать объект DBNavigator1 с таблицей «Book» базы данных, задайте его свойству DataSource значение DataModule1.DataSource1.

Примечание. DBNavigator – компонент, предназначенный для перемещения по набору данных и редактирования данных. DBNavigator имеет ряд кнопок, служащих для управления данными. Чтобы приложение с навигатором работало, надо установить его основное свойство DataSource – источник данных (имя компонента TDataSource).

7. В окне Инспектор объектов для объекта DBNavigator1, щелкнув на свойстве VisibleButtons, разверните список кнопок DBNavigator1 и задайте им следующие свойства: nbFirst (первая), nbProir (предыдущая), nbNext (следующая), nbLast (последняя) – True, остальные кнопки – False. После этого редактируемая форма будет иметь вид, как показано на рис. 6.5.

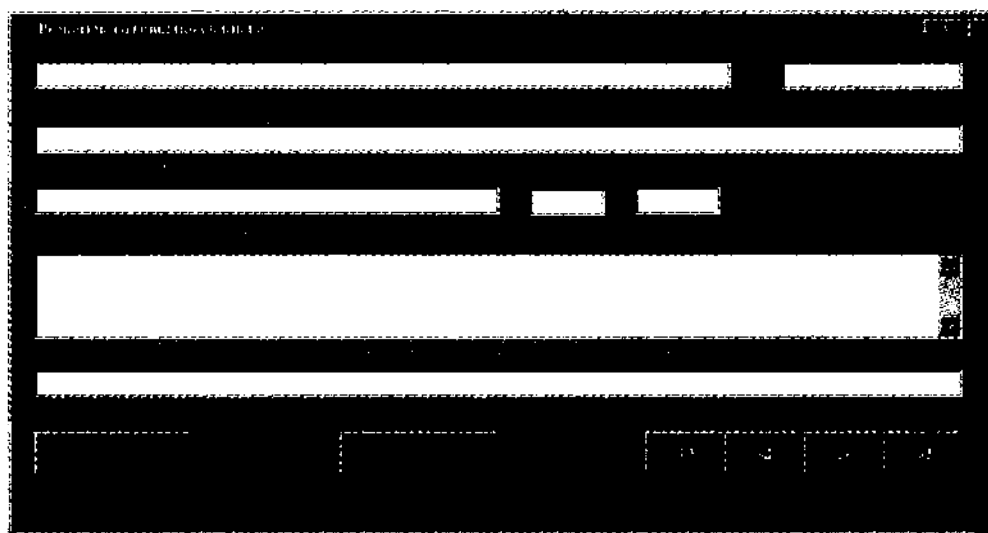


Рис. 6.5. Форма поиска книг в режиме «Читатель»

8. Для создания процедуры обработки события появления формы выберите в окне Инспектор объектов для объекта Form2 и, выбрав на вкладке Events событие OnShow, дважды щелкните в поле его значения. После этого дополните заготовку процедуры следующим образом:

```
procedure TForm2.FormShow(Sender: TObject);
begin
    Form1.Hide;                {Скрыть форму Form1}
    Edit1.Visible := false;    {Невидимые элементы формы}
    Edit5.Visible := false;
    Edit6.Visible := false;
    Edit7.Visible := false;
    Memo1.Visible := false;
    DBNavigator1.Visible := false;
```

```
    Edit2.Text := 'Автор';           {Начальные значения в полях}  
    Edit3.Text := 'Название';  
    Edit4.Text := 'Издательство';  
end;
```

Как видно из текста процедуры, в этой процедуре скрывается форма Form1, задаются невидимыми объекты Edit1, Edit5, Edit6, Edit7, Memo1, DBNavigator1 и в полях Edit2, Edit3 и Edit4 выводятся слова «Автор», «Название» и «Издательство» соответственно.

9. Для создания обработки события закрытия формы выберите в окне Инспектор объектов для объекта Form2 на вкладке Events событие OnClose и дважды щелкните в поле его значения. После этого дополните заготовку процедуры следующим образом:

```
procedure TForm2.FormClose(Sender: TObject; var Action:  
TCloseAction);  
begin  
    Form1.Show; {Показать форму Form1}  
end;
```

10. Для создания процедуры обработки события щелчка по кнопке Button1 с заголовком Очистить дважды щелкните на этой кнопке для генерации заготовки процедуры и после этого дополните текст процедуры следующим образом:

```
procedure TForm2.Button1Click(Sender: TObject);  
begin  
    Edit1.Visible := false;           {Невидимые элементы формы}  
    Edit5.Visible := false;  
    Edit6.Visible := false;  
    Edit7.Visible := false;  
    Memo1.Visible := false;  
    DBNavigator1.Visible := false;  
    Edit2.Text := 'Автор'; {Начальные значения в полях}  
    Edit3.Text := 'Название';  
    Edit4.Text := 'Издательство';  
end;
```

Как видно из текста процедуры, в этой процедуре задаются невидимыми объекты Edit1, Edit5, Edit6, Edit7, Memo1, DBNavigator1 и в полях Edit2, Edit3 и Edit4 выводятся слова «Автор», «Название» и «Издательство» соответственно.

11. Для создания процедуры обработки события щелчка по кнопке Button2 с заголовком Найти дважды щелкните на этой кнопке для генерации заготовки процедуры.

Допишите в раздел описания переменных данного модуля три переменные логического типа:


```
var
```

```
FlT, FlA, FlP: boolean;
```

Дополните текст процедуры следующим образом:

```
var
```

```
FlT, FlA, FlP: boolean;
```

Дополните текст процедуры следующим образом:

```
procedure TForm2.Button2Click(Sender: TObject);
```

```
begin
```

```
DataModule1.TableBook.Active := true;
```

```
DataModule1.TableBook.Filtered := true;
```

```
DataModule1.TableBook.Filter := '';
```

```
FlA := false; FlP := false; FlT := false;
```

```
If (Edit2.Text <> '') and (Edit2.Text <> 'Автор') then
```

```
FlA := true;
```

```
If (Edit3.Text <> '') and (Edit3.Text <> 'Название') then
```

```
FlT := true;
```

```
If (Edit4.Text <> '') and (Edit4.Text <> 'Издательство') then
```

```
FlP := true;
```

```
If FlA or FlT or FlP then begin
```

```
    If FlA then DataModule1.TableBook.Filter := 'Author LIKE ''' +  
    '*' + Edit2.Text + '*' + ''';
```

```
    If FlT then if DataModule1.TableBook.Filter <> '' then DataModule1.  
TableBook.Filter := DataModule1.TableBook.Filter + 'and Name LIKE  
''' + '*' + Edit3.Text + '*' + ''';
```

```
                                else DataModule1.TableBook.Filter :=  
'Name LIKE ''' + '*' + Edit3.Text + '*' + ''';
```

```
    If FlP then if DataModule1.TableBook.Filter <> '' then DataModule1.  
TableBook.Filter := DataModule1.TableBook.Filter + 'and Publishing  
LIKE ''' + '*' + Edit4.Text + '*' + ''';
```

```
                                else DataModule1.TableBook.Filter :=  
'Publishing LIKE ''' + '*' + Edit4.Text + '*' + ''';
```

```
    If DataModule1.TableBook.RecordCount = 0 then ShowMessage('В  
библиотеке нет таких книг!')
```

```
    else
```

```
        begin
```

```
            ShowMessage('Найдено ' + IntToStr(DataModule1.TableBook.  
RecordCount) + ' книг');
```

```
            DataModule1.TableBook.First;
```

```

        If DataModule1.TableBook.RecordCount > 1 then DBNavigator1.
Visible := true;
        Edit1.Visible := true;
        Edit5.Visible := true;
        Edit6.Visible := true;
        Edit7.Visible := true;
        Memo1.Visible := true;
        ShowBooks;      {Показать сведения о найденной книге}
        FlA := false; FlT := false; FlP := false;
    end;

end
else ShowMessage('Не заданы значения! Пожалуйста, введите автора
и/или название и/или издательство!');
end;

```

Как видно из текста процедуры, в начале процедуры активизируется таблица «Book», затем у объекта DataModule1.TableBook включается свойство Filtered и очищается фильтр поиска. Затем переменным FlA, FlP, FlT задаются значения false, обозначающие, что не задан образ поиска в соответствующем поле.

Затем формируется фильтр поиска, объединяющий сведения об авторе, названии книги и издательстве и выполняется проверка свойства RecordCount (количества записей, соответствующих фильтру поиска). Если оно равно 0, то выводится сообщение «В библиотеке нет таких книг!». Если DataModule1.TableBook.RecordCount > 0, то выводится сообщение «Найдено N книг». Если количество найденных книг больше 1, то становится видимым элемент DBNavigator1, отображаются все поля, в которых должны выводиться данные о найденной книге, а затем вызывается процедура показа сведений о найденной книге ShowBooks и переменным FlA, FlT, FlP задаются значения false.

12. Для создания процедуры показа сведений о найденной книге ShowBooks перейдите в окно Редактор кода и введите в раздел private заголовок процедуры

```
procedure ShowBooks;
```

После этого введите текст процедуры следующим образом:

```

procedure TForm2.ShowBooks;
begin
    Edit1.Text := DataModule1.TableBook.FieldValues['Num'];
    Edit2.Text := DataModule1.TableBook.FieldValues['Author'];
    Edit3.Text := DataModule1.TableBook.FieldValues['Name'];
    Edit4.Text := DataModule1.TableBook.FieldValues['Publishing'];
    Edit5.Text := DataModule1.TableBook.FieldValues['Year'];
    Edit6.Text := DataModule1.TableBook.FieldValues['Pages'];

```

```

If DataModule1.TableBook.FieldValues['ReaderNo'] = 0 then
  Edit7.Text := 'Книга имеется в библиотеке ' + DataModule1.
TableBook.FieldValues['Cupboard']
  else Edit7.Text := 'Книга на руках у читателя';
  Memo1.Text := DataModule1.TableBook.FieldValues['Comment'];
end;

```

Как видно из текста процедуры, в начале поля Edit1 – Edit6 и Memo1 заполняются соответствующими данными о книге. Затем в зависимости от значения поля ReaderNo найденной записи в таблице Books выводится сообщение. Если значение поля ReaderNo = 0, то выводится сообщение «Книга имеется в библиотеке», иначе выводится сообщение «Книга на руках у читателя».

13. Для создания процедуры обработки событий щелчков по кнопкам объекта DBNavigator1 дважды щелкните на данном объекте и дополните заготовку процедуры следующим образом:

```

procedure TForm2.DBNavigator1Click(Sender: TObject; Button:
TNavigateBtn);
begin
    {Перемещение по записям базы данных}
    case Button of
        nbFirst : begin
            DataModule1.TableBook.First;    {Первая запись}
            ShowBooks;
            end;
        nbPrior  : begin
            DataModule1.TableBook.Prior;    {Предыдущая}
            ShowBooks;
            end;
        nbNext   : begin
            DataModule1.TableBook.Next;      {Следующая}
            ShowBooks;
            end;
        nbLast   : begin
            DataModule1.TableBook.Last;      {Последняя}
            ShowBooks;
            end;
    end;
end;

```

Как видно из текста процедуры, в зависимости от кнопки DBNavigator1, по которой щелкает пользователь, оператор выбора case с помощью селектора Button задает значение свойства DataModule1.TableBook (First, Prior, Next, Last) и

вызывает процедуру ShowBooks, отображающую сведения о записи, соответствующей значению свойства DataModule1.TableBook (First, Prior, Next или Last).

14. Сохраните изменения в проекте и модуле, щелкнув на кнопке Save All на стандартной панели инструментов интегрированной среды программирования Delphi.

Создание формы и обработчиков обслуживания базы данных

Упражнение 6.5. Откройте проект Library-School и создайте форму Form3, на которой библиотекарь может выполнять обслуживание базы данных, изменять пароль и переходить к обработке данных о книгах и читателях. Выполните следующие действия.

1. Для добавления новой формы в проект выберите в меню File команду New-Form.

2. Задайте свойству Caption формы Form3 значение «Библиотекарь: выбор операции».

3. Из палитры компонентов Standard разместите на форме Form3 компоненты: GroupBox1 с кнопками Button1, Button2 и Button3, GroupBox2 с кнопками Button4, Button5, полями ввода текста Edit1, Edit2, Edit3, Label1, Label2 и Label3.

4. Задайте для компонента GroupBox1 свойство Caption «Обслуживание базы данных», а свойству Caption объектов Button1, Button2 и Button3 «Подключение БД», «Резервирование БД» и «Восстановление БД» соответственно. В компоненте GroupBox2 свойствам Text объектов Edit1, Edit2 и Edit3 задайте пустые значения, а кнопкам Button4 и Button5 – значения «Завершить» и «Применить» соответственно. Свойству Caption компонента GroupBox2 задайте значение «Смена пароля». Свойствам Caption объектов Label1, Label2 и Label3 задайте значения «Текущий пароль», «Новый пароль» и «Подтверждение» соответственно.

5. В верхней части формы разместите компонент SpeedButton1, которому задайте заголовок «Книги и читатели». Чтобы добавить на кнопку SpeedButton1 изображение, выбрав в окне Инспектор объектов SpeedButton1, щелкните по кнопке с тремя точками в поле свойства Clyph (Клип), затем в окне Picture Editor, показанном на рис. 6.6, щелкните по кнопке Load (Загрузить). Выбрав в окне Load Picture папку и файл картинки, щелкните по кнопке Открыть.

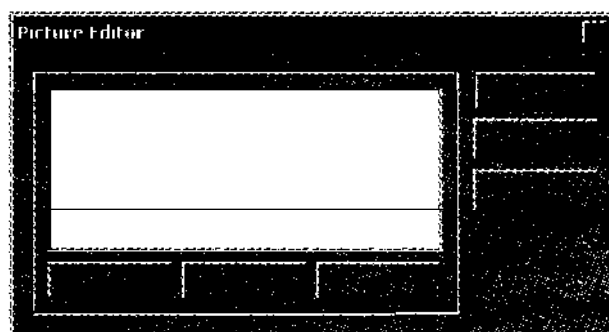


Рис. 6.6. Загрузка клипа для кнопки

После этого форма Form3 будет выглядеть так (рис. 6.7):

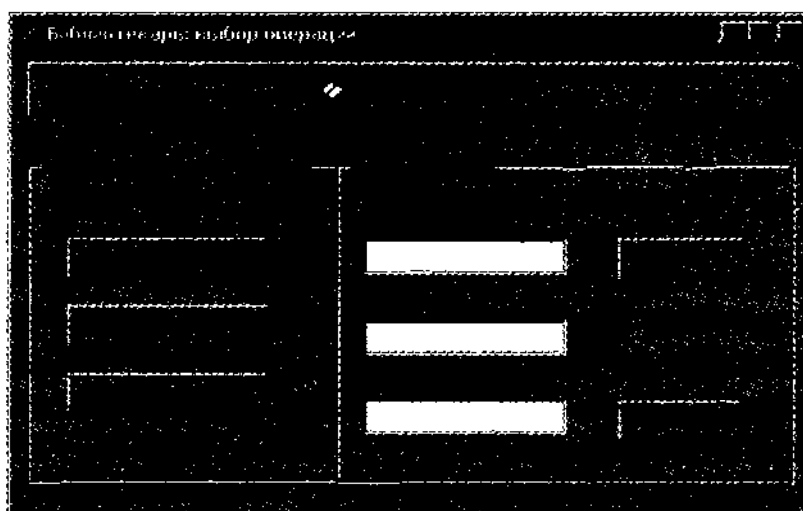


Рис. 6.7. Форма Form3 с элементами для выбора операции библиотекаря

6. Для создания процедуры обработки события появления формы выберите в окне Инспектор объектов объект Form3 и, выбрав на вкладке Events событие OnShow, дважды щелкните в поле его значения. После этого дополните заготовку процедуры следующим образом:

```
procedure TForm3.FormShow(Sender: TObject);
begin
    Button5.Visible := false; {Невидимая кнопка}
end;
```

7. Для создания процедуры обработки события щелчка на кнопке SpeedButton1 с заголовком «Книги и читатели» дважды щелкните на данной кнопке и дополните текст процедуры следующим образом:

```
procedure TForm3.SpeedButton1Click(Sender: TObject);
begin
    Form3.Hide;           {Скрыть форму Form3}
    Form4.Show;           {Отобразить форму Form4}
end;
```

8. В реальной практике работы с базой данных возможно изменение положения базы данных. Чтобы создать процедуру обслуживания щелчка по кнопке Button1 с надписью «Подключение БД», дважды щелкните на этой кнопке в форму Form3 и дополните заготовку процедуры следующим образом:

```
procedure TForm3.Button1Click(Sender: TObject);
begin
    DataModule1.ADOConnection1.Close;
    if EditConnectionString(DataModule1.ADOConnection1) then
    begin
```

```
with DataModule1
do
begin
  ADOConnection1.Connected := true;
  TableBook.Active := true;
  TableReader.Active := true;
end;
end;
end;
```

Как видно из текста процедуры, в начале процедуры закрывается существующий вариант соединения с базой данных. Затем вызывается диалог, в котором определяется свойство `ConnectionString`. Если свойство определено корректно, то выполняется соединение с базой данных и активизируются компоненты, связывающие приложение с таблицами базы данных.

9. Кнопка `Button2` с надписью «Резервирование БД» служит для копирования базы данных на диск. Дважды щелкнув на этой кнопке, создайте заготовку процедуры обработки события щелчка по данной кнопке и дополните его следующим образом:

```
procedure TForm3.Button2Click(Sender: TObject);
var
  Dir: string;
  FileName: string;
begin
  FileName := 'D:\Program Files\Borland\Delphi6\Projects\Library\
  DATA'; {Путь к файлам БД}
  Dir := 'c:\Temp'; {Путь к резервируемой копии БД}
  If not DirectoryExists(Dir) then if CreateDir(Dir) = true
  then ShowMessage('Папка успешно создана! =)')
  else ShowMessage('Папка не создана');
  if not WindowsCopyFile(FileName, Dir)
  then ShowMessage('Ошибка резервирования базы данных')
  else ShowMessage('Резервирование базы данных выполнено
  успешно!');
end;
```

Как видно из текста процедуры, в разделе переменных процедуры описываются две переменные строкового типа `Dir` и `FileName`. В них содержатся имена директории и файла базы данных.

В теле процедуры этим переменным присваиваются соответствующие значения, затем выполняется проверка существования папки для резервирования БД.

Если папки нет, то она создается и об этом выводится сообщение. Затем вызывается функция копирования файлов WindowsCopyFile из WindowsApi с передачей ей двух параметров FileName и Dir. Если WindowsCopyFile(FileName,Dir) возвращает результат Ложь, то выводится сообщение об ошибке резервирования БД, иначе – сообщение об успешности операции резервирования.

10. Для использования функции WindowsCopyFile запишите в списке используемых модулей вызов модуля ShellApi. Текст процедуры WindowsCopyFile может быть записан следующим образом:

```
function WindowsCopyFile(FromFile, ToDir: string): boolean;
var
  F: TShFileOpStruct;
begin
  F.Wnd := 0;
  F.wFunc := FO_COPY;
  FromFile := FromFile + #0;
  F.pFrom := pchar(FromFile);
  ToDir := ToDir + #0;
  F.pTo := pchar(ToDir);
  F.fFlags := FOF_ALLOWUNDO or FOF_NOCONFIRMATION;
  Result := ShFileOperation(F) = 0;
end;
```

11. Кнопка Button3 с надписью «Восстановление БД» служит для восстановления зарезервированной базы данных. Дважды щелкнув по этой кнопке, создайте заготовку процедуры обработки события щелчка по данной кнопке и дополните его следующим образом:

```
procedure TForm3.Button3Click(Sender: TObject);
var
  Dir: string;
  FileName: string;
begin
  Dir := 'D:\Program Files\Borland\Delphi6\Projects\Library\
  DATA'; {Путь к файлам БД}
  FileName := 'C:\Temp'; {Путь к резервной копии БД}
  If not DirectoryExists(Dir)
  then ShowMessage('Папка не создана, потому что нет
  резервированных копий.')
  else if not WindowsCopyFile(Dir,FileName)
  then ShowMessage('Произошла ошибка копирования
  файлов.')
  else ShowMessage('Восстановление БД успешно
  завершено.');
```

Как видно из текста процедуры, после инициализации значений строковых переменных Dir и FileName выполняется проверка существования директории Dir с файлами резервированной БД. Если этой директории нет, то выводится соответствующее сообщение и завершается выполнение процедуры. Если директория с резервированными файлами БД имеется, то вызывается вышеупомянутая функция копирования файлов WindowsCopyFile с передачей ей двух параметров FileName и Dir. Если WindowsCopyFile(FileName,Dir) возвращает результат Ложь, то выводится сообщение об ошибке копирования файлов БД, иначе – сообщение об успешности операции восстановления БД.

12. Проверьте действие созданных процедур. Запустив приложение щелчком по кнопке Run на панели инструментов, войдите в режим работы библиотекаря. Щелкнув в группе «Обслуживание базы данных» по кнопке «Подключение БД», откройте окно DataModule1.ADOConnectionString, показанное на рис. 6.8.

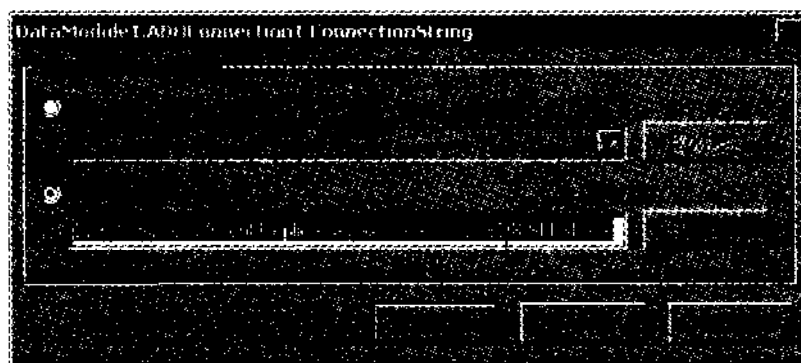


Рис. 6.8. Редактирование строки подключения базы данных

В этом окне щелкните по кнопке Build (Построить) и в окне Свойства связи с данными, на вкладке Provider (Поставщик данных) выберите в списке доступных ADO драйверов доступа к базам данных выделенный по умолчанию «Microsoft Jet 4.0 OLE DB Provider».

Примечание. Драйвер «Microsoft Jet OLE DB Provider» используется для доступа к базам данных MS Access. Такой драйвер обязательно устанавливается на компьютер при установке MS Office, а в последних версиях Windows он устанавливается по умолчанию.

На вкладке Подключение введите в поле 1. Имя (если надо, то и путь) к файлу базы данных. Чтобы легче было выбрать файл базы данных, можно щелкнуть по кнопке с точками справа от строки ввода. Помимо этого нужно заполнить следующие поля:

- 1) Имя пользователя (User name), можно оставить по умолчанию, если не задано иное при создании базы в MS Access;
- 2) Пароль (Password) – если база имеет пароль, то его необходимо указать;
- 3) Пустой пароль (Blank password) – если пароль не нужен, то здесь желательно поставить галочку;
- 4) Позволять сохранять пароль (Allow saving password).

Примечание. Если база данных будет располагаться в той же директории, что и исполняемый файл приложения, то путь указывать не надо. Всегда лучше хранить базы в одной директории с исполняемыми файлами. Если держать файлы отдельно от исполняемого, то придется указывать полный путь, а это может вызвать проблемы при переносе программы на другой компьютер.

13. После выбора базы данных щелкните в окне Свойства связи с данными по кнопке Test Connection (Проверить подключение), чтобы протестировать соединение. Если все указано правильно, то будет выведено сообщение «Проверка подключения выполнена».

Щелкните по кнопке ОК, чтобы закрыть окно Свойства связи с данными создания строки подключения, и еще раз ОК, чтобы закрыть окно DataModule1.ADOConnectionString. Подключение к БД завершено.

14. Щелкнув в группе «Обслуживание базы данных» по кнопке «Резервирование БД», прочтите сообщение об успешности резервирования базы данных. Открыв папку C:\Temp, убедитесь что в ней создана папка DATA с файлом базы данных.

15. Щелкнув в группе «Обслуживание базы данных» по кнопке «Восстановление БД», прочтите сообщение об успешности восстановления базы данных. Завершите работу приложения.

Чтобы библиотечарь имел возможность изменить пароль доступа к режиму, в компоненте GroupBox2 имеется несколько полей и кнопок. Рассмотрим пример, в котором пароль записывается в реестре Windows. Для использования процедур записи и чтения из реестра Windows в разделе Uses включите объявление об использовании модуля Registry, как показано в следующем примере:

uses

Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms, Dialogs, StdCtrls, Buttons, ShellApi, Registry;

16. Для создания процедуры обработки по кнопке Button4 с заголовком «Завершить» щелкните дважды по этой кнопке и допишите заготовку процедуры следующим образом:

```
procedure TForm3.Button4Click(Sender: TObject);
```

```
var
```

```
I,K,L: integer;
```

```
PKey,RKey: string;
```

```
C: Char;
```

```
begin
```

```
Pass := TRegistry.Create;
```

```
Pass.RootKey := HKEY_LOCAL_MACHINE;
```

```
if not Pass.OpenKey('\software\Library',false) then
```

```
begin
```

```
{Сделать проверку пароля}
```

```
Pas:= 'book';           {Начальное значение пароля}
if Edit1.Text = Pas then
begin
  Pass.CreateKey('\software\Library');
  Pass.OpenKey('\software\Library',true);
  Pas := '';
  for I := 1 to Length(Edit1.Text) do
  begin          {Преобразование пароля в цифровой код}
    C := Edit1.Text[I];
    Pas := Pas + IntToStr(Ord(C));
  end;
  PKey := '';
  for I := 1 to length(Pas) do
  begin
    Val(Pas[I],K,L);
    if K > 5 then K := 4 - (9-K) - 1 else K := K + 4;
    PKey := PKey + IntToStr(K);
  end;
  Pass.WriteString('P',PKey); {Запись в реестр}
  Button5.Visible := true;
end
else
begin
  ShowMessage('Неправильный пароль. Вам не разрешено работать в
этом режиме. ');
  Form3.Close;
end;
end
else
begin
  Pas := '';
  for I := 1 to Length(Edit1.Text) do
  begin
    C := Edit1.Text[I];
    Pas := Pas + IntToStr(Ord(C));
  end;
```

```

PKey := '';
for I := 1 to length(Pas) do
begin
  Val(Pas[I], K, L);
  if K > 5 then K := 4 - (9 - K) - 1 else K := K + 4;
  PKey := PKey + IntToStr(K);
end;
Pass.OpenKey('\software\Library', true);
RKey := Pass.ReadString('P'); {Прочитать значение пароля}
if PKey <> RKey then
begin
  ShowMessage('Неправильный пароль!');
  Form3.Close;
end
else Button5.Visible := true;
end;
end;

```

Как видно из текста процедуры, в разделе Var описаны переменные I, K, L, PKey, RKey, C. В начале процедуры задается ветвь реестра 'HKEY_LOCAL_MACHINE\Software\Library', в которой записан пароль. Если пароль в реестре не записан, то проверяется соответствие значения свойства Text поля Edit1 на равенство значению «book» – первоначального значения пароля. Если это так, то пароль преобразуется в цифровой код, этот код записывается в реестр и становится видимой кнопка Button5 с надписью «Применить». Если пароль, введенный в поле Edit1, неверный, то выводится сообщение «Неправильный пароль. Вам не разрешено работать в этом режиме» и закрывается форма Form3.

Если пароль был ранее записан в реестре Windows, то введенный в поле Edit1 пароль преобразуется в цифровой код и сравнивается со значением пароля, прочитанного из реестра. Если пароль не совпадает, то выводится сообщение «Неправильный пароль» и закрывается форма Form3, иначе – становится видимой кнопка Button5 с надписью «Применить».

17. Для создания процедуры обработки по кнопке Button5 с заголовком «Применить» щелкните дважды по этой кнопке и допишите заготовку процедуры следующим образом:

```

procedure TForm3.Button5Click(Sender: TObject);
var
  I, K, L: integer;
  PKey: string;
  C: Char;

```

```
begin
  if Edit2.Text <> Edit3.Text
  then ShowMessage('Неправильное подтверждение пароля.')
  else
    begin
      Pas := '';
      for I := 1 to Length(Edit2.Text) do
        begin
          C := Edit2.Text[I];
          Pas := Pas + IntToStr(Ord(C));
        end;
      PKey := '';
      for I := 1 to length(Pas) do
        begin
          Val(Pas[I], K, L);
          if K > 5 then K := 4 - (9-K) - 1 else K := K + 4;
          PKey := PKey + IntToStr(K);
        end;
      Pass.WriteString('P', PKey);
      ShowMessage('Пароль успешно изменен!');
    end;
end;
```

Как видно из текста процедуры, в начале тела процедуры описано сравнение свойств Text полей Edit2 и Edit3. Если они не равны, то выводится сообщение «Неправильное подтверждение пароля», иначе – измененный пароль преобразуется в цифровой код, записывается в реестр Windows, и на экран выводится сообщение «Пароль успешно изменен!».

18. Сохраните изменения в проекте и модуле и проверьте действие процедур обработки событий для объектов созданной формы Form3.

Создание формы графического интерфейса библиотекаря при обработке данных о книгах

Упражнение 6.6. Откройте проект Library-School и создайте в нем форму Form4, которая обеспечит графический интерфейс библиотекаря при обработке данных о книгах.

1. Для добавления новой формы в проект выберите в меню File команду New-Form. Задайте свойству Caption формы Form4 значение «Библиотекарь: операции с книгами и читателями».

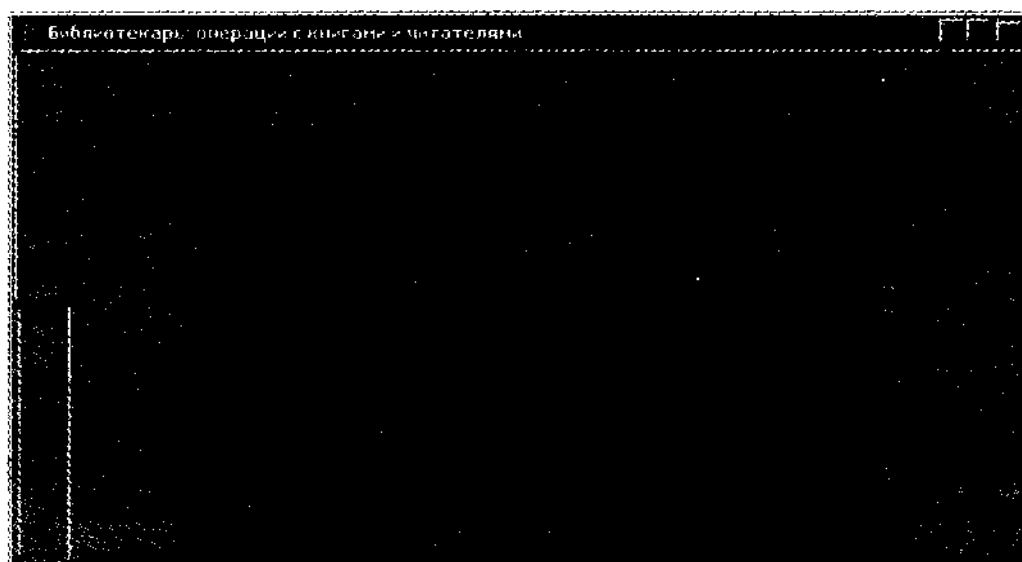


Рис. 6.10. Вид формы с двумя страницами «Книги» и «Читатели»

4. Чтобы на странице с ярлычком «Книги» создать несколько страниц для организации интерфейса библиотекаря при обработке данных в таблицах базы данных о книгах, открыв страницу «Книги», вставьте на нее компонент PageControl2 и на нем создайте четыре страницы с надписью на ярлыках закладок «Добавление книги», «Поиск книги», «Инвентаризация» и «Списание книги».

5. Для организации графического интерфейса библиотекаря в процессе добавления новой книги в таблицу базы данных «Books» откройте страницу TabSheet1 с надписью на ярлыке закладки «Добавление книги», расположите на ней компоненты Edit1, Edit2, Edit3, Edit4, Edit5, Edit6, Edit7, Edit8, Memo1, Button1, Button2 и задайте свойства Text компонентов Edit, Caption компонентов Button, Memo1. Lines, как показано на рис. 6.11.

Рис. 6.11. Вид страницы «Добавление книги»

6. Для организации графического интерфейса библиотекаря в процессе поиска книги в таблице базы данных «Books» откройте страницу TabSheet2 с надписью на ярлыке закладки «Поиск книги», расположите на ней компоненты Edit10, Edit11, Edit12, Edit13, Edit14, Edit15, Edit16, Edit17, Button5, Button6, DBNavigator1 и задайте свойства Text компонентов Edit, Caption компонентов Button, VisibleButtons компонента DBNavigator1, как показано на рис. 6.12.

7. Для организации графического интерфейса библиотекаря в процессе инвентаризации в таблице базы данных «Books» откройте страницу TabSheet3 с надписью на ярлыке закладки «Инвентаризация» и расположите на ней компоненты Button7, Button8 и QuickRep1. Задайте свойства Caption компонентов Button7 и Button8 «Просмотр и печать» и «Экспорт в Excel» соответственно.

Примечание. Инвентаризация книг – полный список книг в таблице базы данных. TQuickRep используется для введения в приложение средств печати отчетов QuickReport.

Рис. 6.12. Вид страницы «Поиск книги»

8. Чтобы подключить создаваемый отчет к таблице базы данных, раскройте в окне Инспектор объектов QuickRep1 свойство DataSet и задайте значение свойства Connection, равное ADOConnection1, свойство DataSet, равное DataModule1.TableBook, TableName, равное «Books».

9. Выбирая в списке Properties (Свойства), детально уточните свойства создаваемого отчета, например, как показано на рис. 6.13, включите отображение следующих разделов отчета: HasTitle (Заголовок отчета); HasColumnHeader (Заголовки столбцов), HasDetail (Детали отчета); HasPageFooter (Подвал страницы отчета), задав этим свойствам значение True.

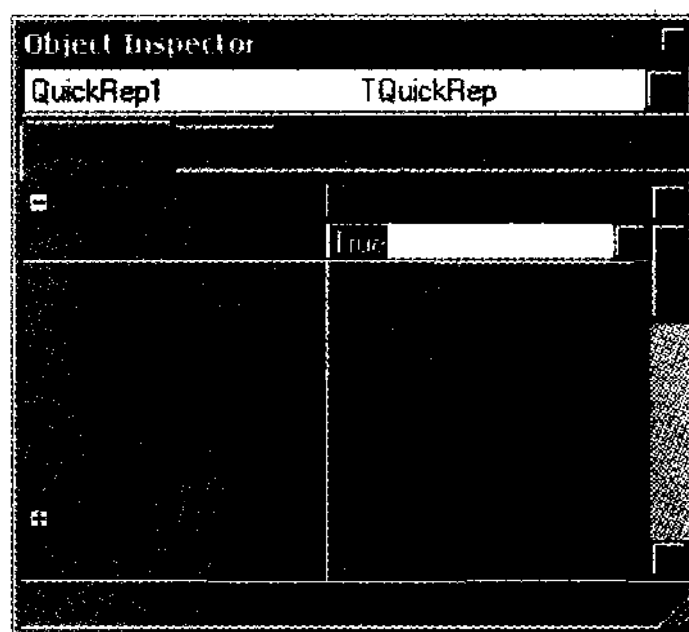


Рис. 6.13. Определение свойства QuickRep1.Bands

После этого в заготовке отчета будут отображены разделы, в которые нужно будет лишь добавить необходимые компоненты.

10. Чтобы задать заголовок отчета, поместите в разделе `QRBand1` компоненты `QRLabel1` и `QRSysData1`. Задайте свойству `QRLabel1.Caption` значение «Инвентаризация», а свойству `QRSysData1.Data` – `grsData`.

11. Чтобы задать шапку инвентаризационной ведомости, разместите в разделе `ColumnHeaderBand1` компоненты `QRLabel2`, `QRLabel3`, `QRLabel4`, `QRLabel5`, `QRLabel6`, `QRLabel7`. Задайте свойствам `Caption` этих объектов соответственно следующие значения: «Номер», «Название», «Автор», «Издательство», «Год», «Цена».

12. Для отображения в ведомости детальной информации об объектах таблицы базы данных «Books» разместите в разделе `DetailBand1` компоненты `QRDBText1`, `QRDBText2`, `QRDBText3`, `QRDBText4`, `QRDBText5` и `QRDBText6`.

13. Чтобы задать отображение поля записи из таблицы базы данных в компоненте `QRDBText1`, выбрав в окне Инспектор объектов объект `QRDBText1`, задайте свойство `DataSet` равным `DataModule1.TableBook`, а в свойстве `DataField` выберите в списке нужное поле записи данных, например `Num`.

Действуя аналогично, задайте отображение полей записи из таблицы базы данных в компонентах `QRDBText2`, `QRDBText3`, `QRDBText4`, `QRDBText5` и `QRDBText6`.

14. Для отображения в подвале каждой страницы отчета суммы стоимости книг разместите в разделе `PageFooter` компоненты `QRLabel8` и `QRExpr1`. Задайте свойству `QRLabel8.Caption` значение «Сумма».

15. Чтобы компонент `QRExpr1` автоматически суммировал и отображал в подвале каждой страницы сумму стоимости книг, выбрав в окне Инспектор объектов `QRExpr1`, дважды щелкните по кнопке с тремя точками в поле значений свойства `Expression` и в окне `Expression Wizard` задайте выражение `SUM(price)`, как показано на рис. 6.14.

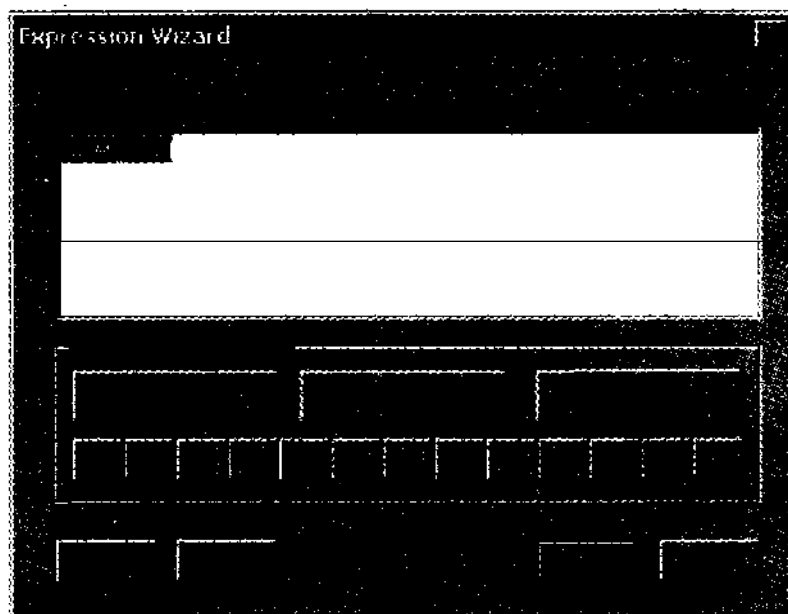


Рис. 6.14. Создание выражения

По окончании создания выражения щелчком по кнопке ОК закройте окно Expression Wizard. После этого страница «Инвентаризация» будет выглядеть, как на рис. 6.15.

16. Для организации графического интерфейса библиотекаря в процессе списания книги и удаления ее из таблицы базы данных «Books» откройте страницу TabSheet4 с надписью на ярлыке закладки «Списание книги», расположите на ней компоненты Edit9, Button3, Button4 и задайте свойства Text компонентов Edit, Caption компонентов Button, как показано на рис. 6.16.

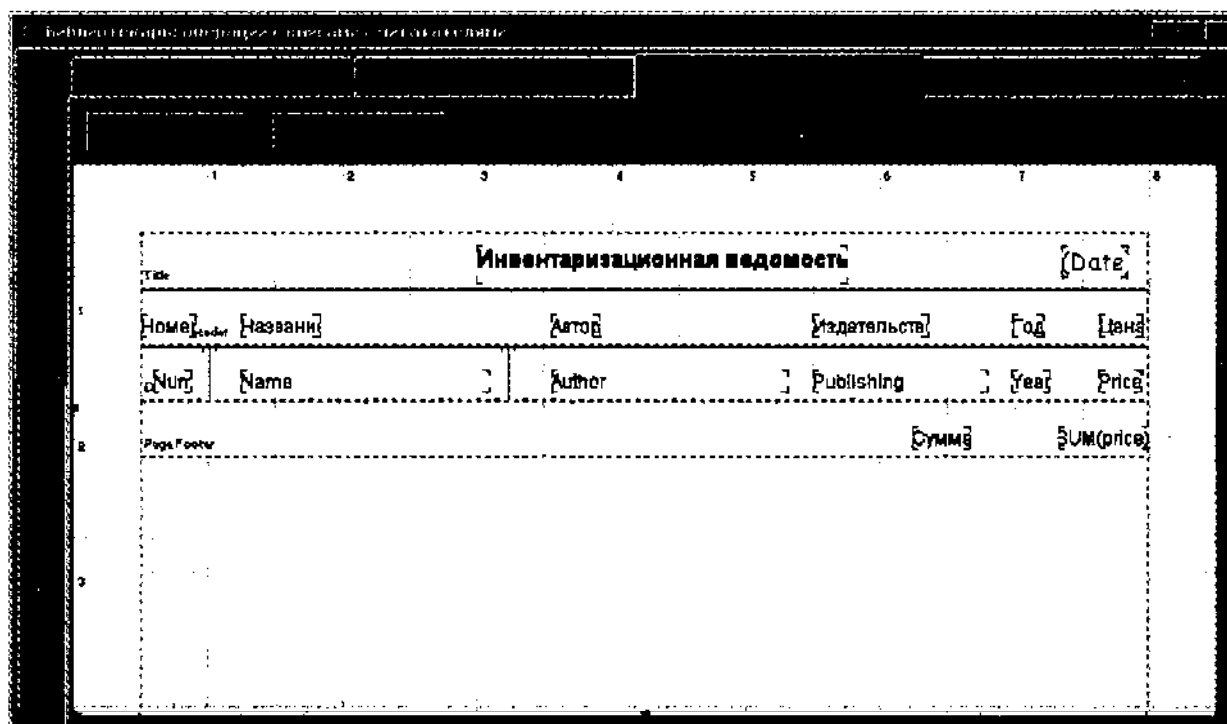


Рис. 6.15. Вид страницы «Инвентаризация»

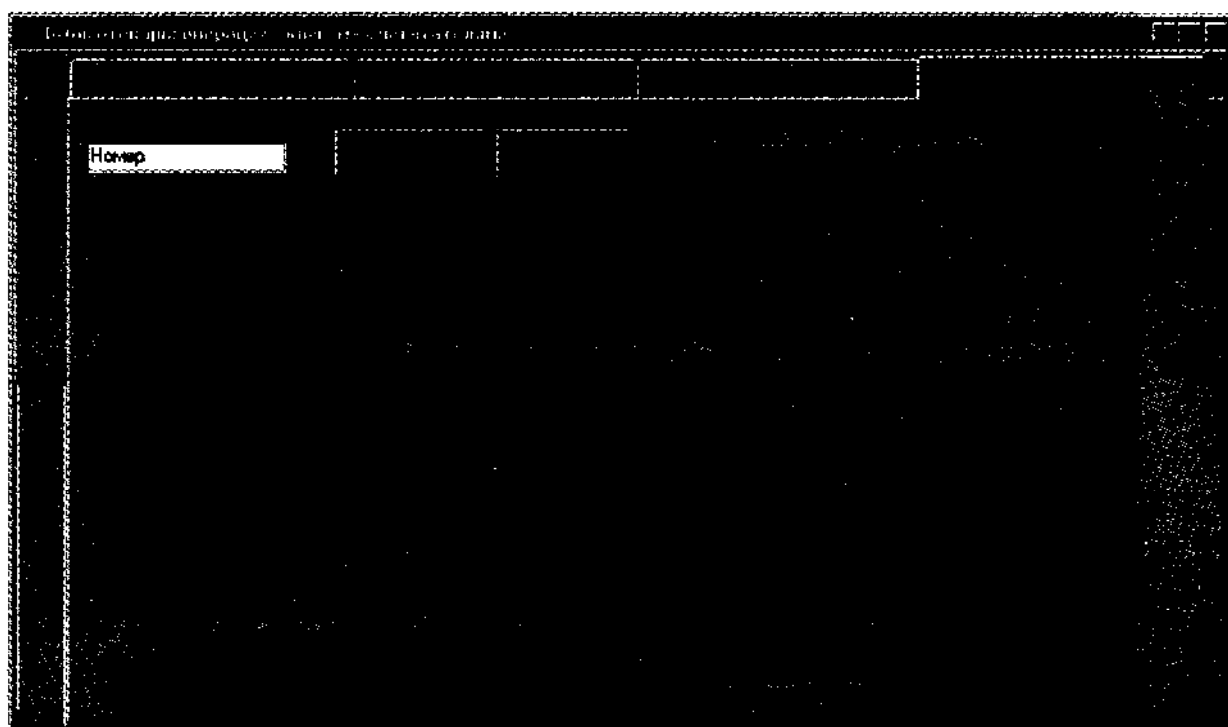


Рис. 6.16. Вид страницы «Списание книги»

17. Сохраните изменения в проекте и модуле.

Создание формы графического интерфейса библиотекаря при обработке данных о читателях

Упражнение 6.7. Откройте проект Library-School и на форме Form4 создайте на странице с ярлычком «Читатели» несколько страниц для организации интерфейса библиотекаря при обработке данных о читателях. Для этого выполните следующее.

1. Чтобы на странице с ярлычком «Читатели» создать несколько страниц для организации интерфейса библиотекаря при обработке данных в таблицах базы данных о читателях, открыв страницу «Читатели», вставьте на нее компонент PageControl3 и на нем создайте четыре страницы с надписью на ярлыках закладок «Добавление нового читателя», «Поиск/Удаление/Изъятие книг», «Выдача книг» и «Список читателей».

2. Для организации графического интерфейса библиотекаря в процессе добавления нового читателя в таблицу базы данных «Reader» откройте страницу TabSheet5 с надписью на ярлыке закладки «Добавление книги», расположите на ней компоненты Edit18, Edit19, Edit20, Edit21, Edit22, Edit23, CheckBox1, Button9, Button10 и задайте свойства Text компонентов Edit, Caption компонентов Button, как показано на рис. 6.17.

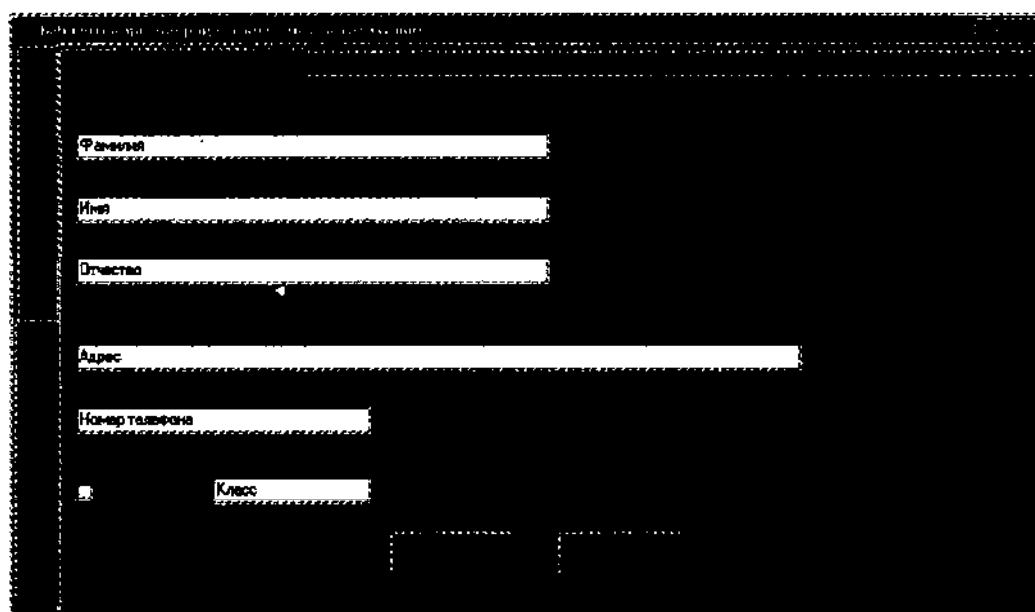


Рис. 6.17. Вид страницы «Добавление нового читателя».

3. Для организации графического интерфейса библиотекаря в процессе поиска читателя, изъятия книг у читателя и удаления читателя из таблицы базы данных «Reader» откройте страницу TabSheet6 с надписью на ярлыке закладки «Поиск / Удаление / Изъятие книг» и расположите на ней компоненты Edit24, Edit25, Edit26, Edit27, Edit28, Edit29, CheckBox2, Button11, Button12, Button13, DBNavigator2, GroupBox1 с компонентами ListBox1 и Button14. Задайте свойства Text компонентов Edit, Caption компонентов CheckBox2, Button11–Button14, GroupBox1, а также свойства VisibleButtons компонента DBNavigator1, как показано на рис. 6.18.



Рис. 6.18. Вид страницы «Поиск / Удаление / Изъятие книг»

4. Для организации графического интерфейса библиотекаря в процессе выдачи книг читателю откройте страницу TabSheet7 с надписью на ярлыке закладки «Выдача книг» и расположите на ней компоненты Edit30, Edit31, Edit32, Edit33, Edit34, Edit35, CheckBox3, GroupBox2 с компонентом ListBox2, GroupBox3

с компонентами Button15, Button16, Edit36, Edit37, Edit38, Edit39, Edit40. Задайте свойства Text компонентов Edit, Caption компонентов CheckBox3, GroupBox2, GroupBox3, Button15, Button16, как показано на рис. 6.19.



Рис. 6.19. Вид страницы «Выдача книг»

5. Для организации графического интерфейса библиотекаря в процессе выдачи книг читателю откройте страницу TabSheet8 с надписью на ярлыке закладки «Список читателей» и расположите на ней компоненты Button17, Button18, QuickRep2, WordApplication1 и WordDocument1. Задайте свойства Caption компонентов Button17 и Button18, как показано на рис. 6.20.

6. Используя в качестве подсказки операции, проделанные вами при создании отчета в п. 7–17 упражнения 6.6, сконструируйте форму отчета, показанную на рис. 6.20.

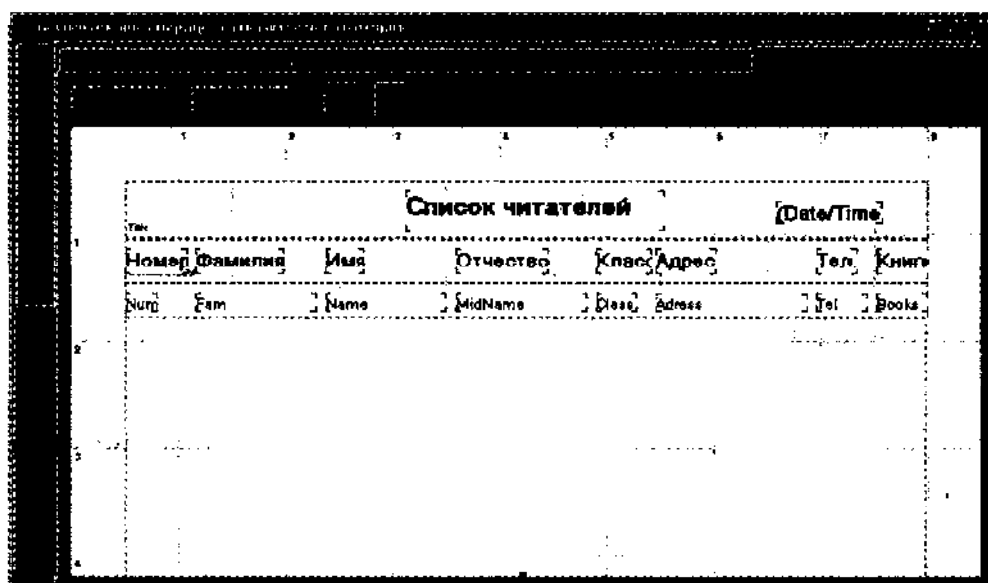


Рис. 6.20. Вид страницы «Список читателей»

7. Сохраните изменения в проекте и модуле.

Создание формы графического интерфейса библиотекаря при добавлении сведений о книге в таблицу базы данных

Упражнение 6.8. Создайте процедуры обработки событий при добавлении книги в таблицу базы данных. Для этого выполните следующее.

1. Откройте проект Library-School, затем командой View-Form выберите в списке форм Form4.

2. Откройте в форме Form4 страницу «Книги» на закладке «Добавление книги». Для создания процедуры обработки события щелчка по кнопке Button1 с надписью «Очистить» дважды щелкните по данной кнопке и дополните заготовку процедуры, созданную редактором кода Delphi, следующим образом:

```
procedure TForm4.Button1Click(Sender: TObject);  
begin  
    Form4.Edit2.Text := 'Автор';  
    Form4.Edit3.Text := 'Название';  
    Form4.Edit4.Text := 'Издательство';  
    Form4.Edit5.Text := 'Год';  
    Form4.Edit6.Text := 'Страниц';  
    Form4.Edit7.Text := 'Место хранения';  
    Form4.Memo1.Text := 'Аннотация';  
    NumPlus;  
end;
```

Как видно из текста процедуры, в ней задаются начальные значения полям для ввода данных о книге, а затем вызывается процедура NumPlus.

3. Для использования данной процедуры опишите ее заголовок в разделе Private следующим образом:

```
private  
    procedure NumPlus;
```

В раздел описания переменных данного модуля добавьте описание переменной NB длинного целого типа, которая будет задавать значение инвентарного номера добавляемой в базу данных книги.

```
var  
    NB: longint;
```

Текст процедуры можно записать следующим образом:

```
procedure TForm4.NumPlus;  
begin  
    DataModule1.TableBook.Active := true;  
    DataModule1.TableBook.Last;  
    if DataModule1.TableBook.FieldValues['Num'] < 1  
        then NB := 1  
        else NB := DataModule1.TableBook.FieldValues['Num'] + 1;  
    Form4.Edit1.Text := IntToStr(NB);  
    DataModule1.TableBook.Active := false;  
end;
```

В данной процедуре сначала определяется инвентарный номер последней книги в таблице «Books». Если он меньше 1, т.е. книг в базе данных нет, то NB задается значение 1, иначе – инвентарный номер последней книги увеличивается на 1 и выводится в поле Edit1.

4. Для создания процедуры обработки события щелчка по кнопке Button2 с надписью «Записать книгу» дважды щелкните по данной кнопке и дополните заготовку процедуры следующим образом:

```
procedure TForm4.Button2Click(Sender: TObject);  
begin  
    if (Edit1.Text <> '') and (Edit1.Text <> 'Номер')  
        and (Edit2.Text <> '') and (Edit2.Text <> 'Автор')  
        and (Edit3.Text <> '') and (Edit3.Text <> 'Название')  
        and (Edit4.Text <> '') and (Edit4.Text <> 'Издательство')  
        and (Edit5.Text <> '') and (Edit5.Text <> 'Год')  
        and (Edit6.Text <> '') and (Edit6.Text <> 'Страниц')  
        and (Edit7.Text <> '') and (Edit7.Text <> 'Место хранения')  
        and (Edit8.Text <> '') and (Edit8.Text <> 'Цена')  
    then begin  
        with DataModule1.TableBook do begin  
            Active := true;  
            Last;
```

```
Append;

FieldValues['Num'] := StrToInt(Edit1.Text);
FieldValues['Name'] := Edit3.Text;
FieldValues['Author'] := Edit2.Text;
FieldValues['Publishing'] := Edit4.Text;
FieldValues['Year'] := StrToInt(Edit5.Text);
FieldValues['Pages'] := StrToInt(Edit6.Text);
FieldValues['Price'] := StrToFloat(Edit8.Text);
FieldValues['Cupboard'] := Edit7.Text;
FieldValues['Comment'] := Memol.Text;
FieldValues['ReaderNo'] := 0;

Post;

Active := false;

end;

NumPlus;

ShowMessage('Книга записана');

Form4.Edit2.Text := 'Автор';
Form4.Edit3.Text := 'Название';
Form4.Edit4.Text := 'Издательство';
Form4.Edit5.Text := 'Год';
Form4.Edit6.Text := 'Страниц';
Form4.Edit7.Text := 'Место хранения';
Form4.Memol.Text := 'Аннотация';

end

else ShowMessage('Не заданы значения!');

end;
```

5. Для устранения ошибок ввода в поле Edit1 выберите в окне Инспектор объектов компонент Edit1, и, открыв вкладку Events, дважды щелкните в поле события OnKeyPress, а затем дополните заготовку процедуры следующим образом:

```
procedure TForm4.Edit1KeyPress(Sender: TObject; var Key: Char);  
begin  
    if not (Key in ['0' .. '9']) then Key := #0  
end;
```

6. Подключите эту процедуру к обработке событий нажатия клавиш в полях Edit5 и Edit6.

7. Для поля Edit8, в которое вводится цена книги, а она может быть и дробной, создайте отдельную процедуру, в тексте которой добавьте разрешенный символ «,». Текст данной процедуры может быть записан так:

```
procedure TForm4.Edit8KeyPress(Sender: TObject; var Key: Char);  
begin  
    if not (Key in ['0' .. '9', ',', '.']) then Key := #0;  
end;
```

8. Сохраните изменения в проекте и модуле. Запустив приложение, войдите в режим работы библиотекаря, выберите операции с книгами и читателями. На странице «Книги» откройте закладку «Добавление книги», как показано на рис. 6.21. Проверьте функционирование созданных в данном упражнении обработчиков событий. Добавьте несколько книг в базу данных и закройте приложение.

9. Открыв в среде MS Access базу данных Library, просмотрите таблицу «Books» и убедитесь в том, что книги действительно записаны. Закройте MS Access.

Рис. 6.21. Вид формы приложения на странице «Книги» на закладке «Добавление книги»

Создание формы графического интерфейса библиотекаря при добавлении сведений о читателе в таблицу базы данных

Упражнение 6.9. Создайте процедуры обработки событий при добавлении читателя в таблицу базы данных. Для этого выполните следующее.

1. Откройте проект Library-School и выберите в списке форм Form4. На странице «Читатели» откройте закладку «Добавление нового читателя».

2. Для создания процедуры обработки события щелчка по кнопке Button9 с надписью «Очистить» дважды щелкните по данной кнопке и дополните заготовку процедуры следующим образом:

```
procedure TForm4.Button9Click(Sender: TObject);
begin
    Edit18.Text := 'Фамилия';
    Edit19.Text := 'Имя';
    Edit20.Text := 'Отчество';
    Edit21.Text := 'Адрес';
    Edit22.Text := 'Номер телефона';
    CheckBox1.Checked := false;
    Edit23.Text := 'Класс';
end;
```

Как видно из текста процедуры, при щелчке по кнопке Button9 задаются начальные значения полей ввода данных о новом читателе.

3. Чтобы исключить отображение поля Edit23 для ввода названия класса в случае, когда читатель является не учащимся, создадим процедуру обработки события щелчка на флажке CheckBox1 с надписью «Учитель?». Дважды щелкните по компоненте CheckBox1 и дополните заготовку процедуры следующим образом:

```
procedure TForm4.CheckBox1Click(Sender: TObject);
begin
    If CheckBox1.Checked
    then
        begin
            Edit23.Text := '';
            Edit23.Visible := false;
        end
    else
        begin
            Edit23.Text := 'Класс';
            Edit23.Visible := true;
        end;
end;
end;
```



```

        else FieldValues['Class'] := Edit23.Text;
    Post;
    Active := false;
end;
NumReadPlus;
ShowMessage('Читатель добавлен');
Edit18.Text := 'Фамилия';
Edit19.Text := 'Имя';
Edit20.Text := 'Отчество';
Edit21.Text := 'Адрес';
Edit22.Text := 'Номер телефона';
CheckBox1.Checked := false;
Edit23.Text := 'Класс';
end
else ShowMessage('Не заданы значения!');
end;
```

В теле процедуры проверяется, заполнены ли поля формы значениями, отличающимися от начальных. Если это не так, то выводится сообщение «Не заданы значения», иначе – вызывается процедура NumReadPlus, которая увеличивает счетчик числа записанных в библиотеку читателей. Затем выполняется запись сведений о читателе из полей формы в таблицу базы данных, выводится сообщение «Читатель добавлен», и опять задаются начальные значения полей ввода данных о новом читателе.

5. Для создания процедуры NumReadPlus перейдите в окно Редактор кода и введите в раздел private заголовок процедуры

```
procedure NumReadPlus;
```

После этого введите текст процедуры следующим образом:

```

procedure TForm4.NumReadPlus;
begin
    DataModule1.TableReader.Active := true;
    DataModule1.TableReader.Last;
    if DataModule1.TableReader.FieldValues['Num'] < 1 then NB:=1
    else NB := DataModule1.TableReader.FieldValues['Num'] + 1;
    DataModule1.TableReader.Active := false;
end;
```

В теле процедуры устанавливается связь с таблицей «Reader» базы данных, указатель устанавливается на последнюю запись в таблице и проверяется значение поля FieldValues['Num'] – номер последнего читателя. Если читателей в таблице еще нет, то переменной NB присваивается 1, иначе – NB присваивается увеличенный на 1 номер последнего читателя.

Примечание. В п.3 упражнения 6.8 переменная NB была описана в разделе переменных данного модуля.

6. Сохраните изменения в проекте и модуле. Запустив приложение, войдите в режим работы библиотекаря, выберите операции с книгами и читателями. На странице «Читатели» откройте закладку «Добавление нового читателя». Проверьте функционирование созданных в данном упражнении обработчиков событий. Добавьте несколько читателей в базу данных и закройте приложение.

7. Открыв в среде MS Access базу данных Library, просмотрите таблицу «Reader» и убедитесь в том, что читатели действительно записаны. Закройте MS Access.

Создание процедур обработки событий при выполнении инвентаризации книг

Упражнение 6.10. Создайте процедуры обработки событий при выполнении инвентаризации книг в библиотеке. Для этого выполните следующие действия.

1. Откройте проект Library-School и, выбрав в списке форм Form4, на странице «Книги» откройте закладку «Инвентаризация». Как было показано на рис. 6.15, в окне приложения на данной странице имеются две кнопки, которыми может воспользоваться пользователь: Button7 с надписью «Просмотр и печать» и Button8 с надписью «Экспорт в Excel».

2. Создайте заготовку процедуры обработки события щелчка по кнопке Button7 с надписью «Просмотр и печать» и дополните ее следующим образом:

```
procedure TForm4.Button7Click(Sender: TObject);  
begin  
    DataModule1.TableBook.Active := true;  
    DataModule1.TableBook.Filter := '';  
    DataModule1.TableBook.First;  
    Form4.QuickRepl.PreviewModal;  
end;
```

В теле данной процедуры активизируется связь с таблицей «Books» базы данных, задается отображение всех записей, указатель устанавливается на первой записи таблицы и вызывается стандартная процедура PreviewModal для вывода сформированного отчета на экран и печати на бумаге.

3. Сохраните проект и проверьте действие созданной процедуры. Запустите приложение, войдите в режим «Библиотекарь: операции с книгами и читателями». Открыв страницу «Книги» на закладке «Инвентаризация», щелкните по кнопке «Просмотр и печать». Если текст процедуры TForm4.Button7Click был записан правильно, то на экране будет раскрыто окно Print Preview, в котором будет выведен отчет для просмотра перед печатью, как показано на рис. 6.22.

Print Preview

Инвентаризационная ведомость					05.01.2008
Номер	Название	Автор	Издательство	Год	Цена
10	Мультимедиа	Попов В.Б.	Финансы и статистика	2007	120
11	Паскаль для школьников	Попов	Финансы и статистика	1998	70
12	Большая юнга сказок	Шалаев	Слов о	2006	563
13	Война и мир	Толстой ЛН	Слов о	2005	968
14	Мцыри	Лермонтов	Малыш ок	1963	58
15	Сказки	Пушкин АС	Дрофа	2005	78
16	Сказка о царе Салтане	Пушкин АС	Дрофа	2001	456
17	Отцы и дети	Тургенев ИС	Просвещение	1996	321
Сумма					2634

Рис. 6.22. Просмотр инвентаризационной ведомости перед печатью

4. Создайте заготовку процедуры обработки события щелчка по кнопке Button8 с надписью «Экспорт в Excel» и дополните ее следующим образом:

```
procedure TForm4.Button8Click(Sender: TObject);
var
  XLApp, Sheet, Column: Variant;
  I, Index: integer;
  Sum: real;
begin
  XLApp := CreateOleObject('Excel.Application');
  XLApp.Visible := true;
  XLApp.Workbooks.Add(-4167);
  Sum := 0;
  XLApp.Workbooks[1].Worksheets[1].Name := 'Инвентаризация';
  Column := XLApp.Workbooks[1].Worksheets['Инвентаризация'].
  Columns;
  Column.Columns[1].ColumnWidth := 6;
  Column.Columns[2].ColumnWidth := 20;
  Column.Columns[3].ColumnWidth := 20;
```

```
Column.Columns[4].ColumnWidth := 16;
Column.Columns[5].ColumnWidth := 7;
Column.Columns[6].ColumnWidth := 10;
Column:=XLApp.Workbooks[1].Worksheets['Инвентаризация'].Rows;
Column.Rows[1].Font.Bold := true;
Column.Rows[1].Font.Color := clBlue;
Column.Rows[1].Font.Size := 16;
Column.Rows[1].Font.Name := 'Microsoft Sans Serif';
Sheet := XLApp.Workbooks[1].Worksheets['Инвентаризация'];
Sheet.Cells[1,2] := 'Инвентаризационная ведомость';
Sheet.Cells[1,5].Font.Color := clBlack;
Sheet.Cells[1,4].Font.Size := 12;
Sheet.Cells[1,4] := DateToStr(Date);
Column.Rows[3].Font.Bold := true;
Column.Rows[3].Font.Color := clBlack;
Sheet.Cells[3,1] := 'Номер';
Sheet.Cells[3,2] := 'Название';
Sheet.Cells[3,3] := 'Автор';
Sheet.Cells[3,4] := 'Издательство';
Sheet.Cells[3,5] := 'Год';
Sheet.Cells[3,6] := 'Цена';
index := 4;
with DataModule1.TableBook do begin
  Active := true;
  First;
  for I := 0 to RecordCount - 1 do
  begin
    Sheet.Cells[index,1] := Fields.Fields[0].AsString;
    Sheet.Cells[index,2] := Fields.Fields[1].AsString;
    Sheet.Cells[index,3] := Fields.Fields[2].AsString;
    Sheet.Cells[index,4] := Fields.Fields[3].AsString;
    Sheet.Cells[index,5] := Fields.Fields[4].AsString;
    Sheet.Cells[index,6] := Fields.Fields[5].AsString;
    Sum := Sum + StrToFloat(Fields.Fields[5].AsString);
    Inc(Index);
  Next;
end;
Active := false;
Inc(Index);
Column.Rows[index].Font.Bold := true;
Column.Rows[index].Font.Name := 'Microsoft Sans Serif';
Column.Rows[index].Font.Color := clGreen;
```

```

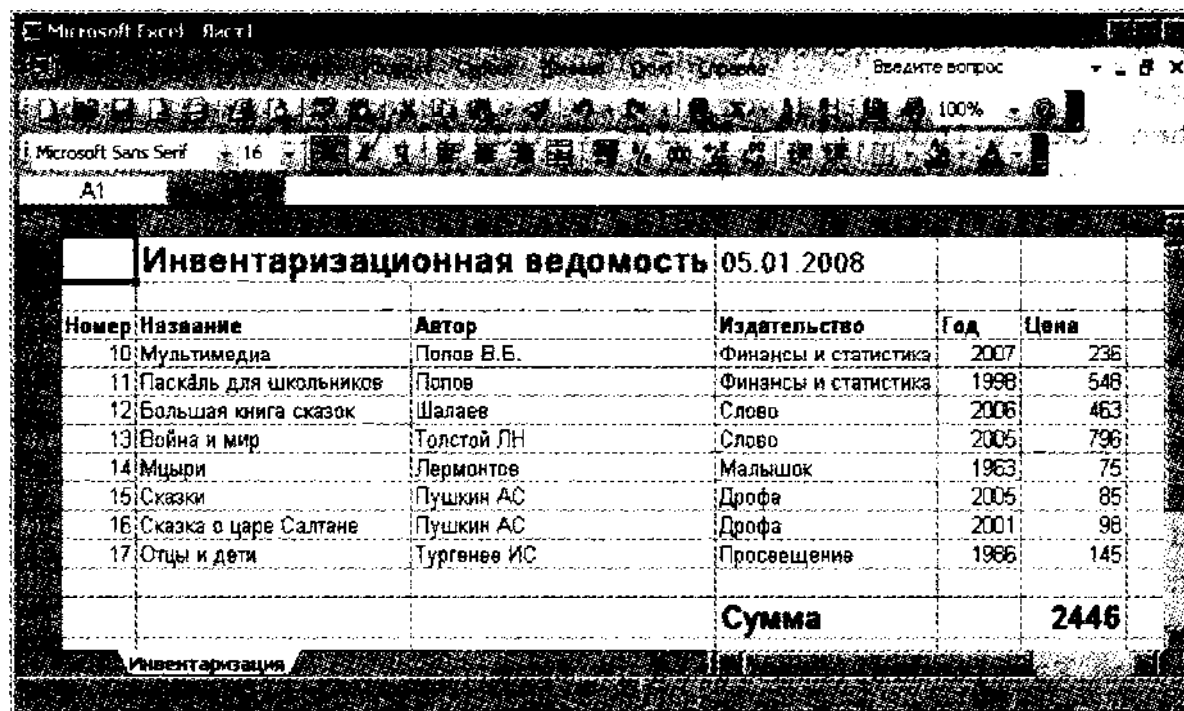
Column.Rows[index].Font.Size := 14;
Sheet.Cells[index,4] := 'Сумма';
Sheet.Cells[index,6] := Sum;
end;
end;

```

Как видно из текста процедуры, в разделе описания переменных введены шесть переменных. В начале процедуры создается переменная типа Variant XLApp, в которой записывается объект, созданный приложением MS Excel, и обнуляется значение переменной Sum, в которой будет накапливаться сумма стоимостей книг. Затем задается имя «Инвентаризация» листу рабочей книги MS Excel, задается ширина используемых колонок и выводятся в определенные ячейки установленным шрифтом текст заголовка таблицы и заголовки граф таблицы.

Оператор with DataModule1.TableBook заполняет ячейки листа «Инвентаризация» данными из таблицы «Books» базы данных. В заключительной части процедуры описано изменение шрифта и вывод в последней строке текста «Сумма» и суммированного значения всех стоимостей книг.

5. Сохраните проект и проверьте действие созданной процедуры. Запустите приложение, войдите в режим «Библиотекарь: операции с книгами и читателями». Открыв страницу «Книги» на закладке «Инвентаризация», щелкните по кнопке «Экспорт в Excel». Если текст процедуры TForm4.Button8Click был записан правильно, то на экране будет раскрыто окно MS Excel, в нем будет создана книга с листом «Инвентаризация», в котором будет выведена инвентаризационная ведомость, как показано на рис. 6.23.



The screenshot shows a Microsoft Excel window titled 'Лист1'. The active sheet is 'Инвентаризация'. The table contains the following data:

Инвентаризационная ведомость 05.01.2008					
Номер	Название	Автор	Издательство	Год	Цена
10	Мультимедиа	Полов В.Б.	Финансы и статистика	2007	236
11	Паскаль для школьников	Попов	Финансы и статистика	1998	548
12	Большая книга сказок	Шалаев	Слово	2006	463
13	Война и мир	Толстой ЛН	Слово	2005	796
14	Мцыри	Лермонтов	Малышок	1963	75
15	Сказки	Пушкин АС	Дрофа	2005	85
16	Сказка о царе Салтане	Пушкин АС	Дрофа	2001	98
17	Отцы и дети	Тургенев ИС	Просвещение	1986	145
Сумма					2446

Рис. 6.23. Просмотр инвентаризационной ведомости в формате MS Excel

6. Просмотрите инвентаризационную ведомость на листе «Инвентаризация» в книге MS Excel и убедитесь в том, что данные о книгах отображаются корректно. Закройте MS Excel.

Создание процедур обработки событий при выполнении списания книг

Упражнение 6.11. Создайте процедуры обработки событий при выполнении списания книг и удаления из базы данных библиотеки. Для этого выполните следующие действия.

1. Откройте проект Library-School и, выбрав в списке форм Form4, на странице «Книги» откройте закладку «Списание книг». Как было показано на рис. 16.16, в окне приложения на данной странице имеются две кнопки, которыми может воспользоваться пользователь: Button3 с надписью «Найти» и Button4 с надписью «Списать книгу».

2. Создайте заготовку процедуры обработки события щелчка по кнопке Button7 с надписью «Найти» и дополните ее следующим образом:

```
procedure TForm4.Button3Click(Sender: TObject);
begin
if (Edit9.Text <> '') and (Edit9.Text <> 'Номер')
then begin
    DataModule1.TableBook.Active := true;
    DataModule1.TableBook.Filtered := true;
    DataModule1.TableBook.Filter := 'Num =' + Edit9.Text;
    if DataModule1.TableBook.RecordCount = 0
    then ShowMessage('Книга не найдена!')
    else
        begin
            ShowMessage('Книга найдена');
            DataModule1.TableBook.Last;
            FlagDel := true;
            Button4.Visible := true;
        end;
    end
else ShowMessage('Пожалуйста, введите номер!');
end;
```

Тело процедуры представляет собой два вложенных оператора ветвления: внешний оператор ветвления проверяет, изменено ли значение свойства Text объекта Edit9. Если оно не изменено, то выводится сообщение «Пожалуйста,

введите номер», иначе – активизируется связь с таблицей «Books» базы данных, задается значение фильтра поиска. Если записи, соответствующей заданному фильтру, в таблице «Books» нет, то выводится сообщение «Книга не найдена», иначе – переменной FlagDel присваивается значение True, а кнопка Button4 с надписью «Списать книгу» становится видимой.

3. Для устранения ошибок ввода в поле Edit9 подключите к событию OnKeyPress процедуру Edit1KeyPress.

4. Создайте заготовку процедуры обработки события щелчка по кнопке Button4 с надписью «Списать книгу» и дополните ее следующим образом:

```
procedure TForm4.Button4Click(Sender: TObject);

var
  A: string;
begin
  If DataModule1.TableBook.FieldValues['ReaderNo'] = 0
  then begin
    FlagDel := true;
    DataModule1.TableBook.Delete;
    ShowMessage('Книга удалена');
    Button4.Visible := false;
    Edit9.Text := 'Номер';
  end
  else
  begin
    FIOReader(DataModule1.TableBook.FieldValues['ReaderNo']);
    A := 'Книга на руках. Читатель: ' + FIORead + ' Вы все еще хотите
удалить эту книгу?';
    if MessageDlg(A,mtConfirmation,[mbYes,mbNo],0) = mrNo
    then FlagDel := false
    else FlagDel := true;
    if not FlagDel
    then Button4.Visible := false
    else begin
      DataModule1.TableBook.Delete;
```

```
    ShowMessage('Книга удалена');  
    Button4.Visible := false;  
    Edit9.Text := 'Номер';  
end;  
end;  
DataModule1.TableBook.Filtered := false;  
end;
```

В разделе описания переменных процедуры описана переменная A строкового типа, которая служит для формирования текста сообщения.

В начале процедуры проверяется значение поля «ReaderNo» записи данных для выбранной книги. Если оно равно 0, то выполняется удаление книги из таблицы «Books», выводится сообщение «Книга удалена» и скрывается кнопка Button4 с надписью «Списать книгу». Если поле «ReaderNo» записи выбранной книги не равно 0, это означает, что книга выдана читателю.

Чтобы определить фамилию и инициалы читателя, вызывается процедура FIOReader, которую мы создадим ниже. Далее в переменной A формируется строка сообщения, которая выводится в диалоговом окне сообщения. В зависимости от выбора пользователя в диалоговом окне сообщения флагу FlagDel присваивается значение True или False. Если FlagDel имеет значение False, то книга не удаляется и кнопка Button4 с надписью «Списать книгу» становится невидимой, если иначе – выполняется удаление книги из таблицы «Books» базы данных, выводится сообщение «Книга удалена», и после этого кнопка Button4 с надписью «Списать книгу» становится невидимой.

5. Для создания процедуры FIOReader, которая по номеру читателя определяет его фамилию и инициалы, перейдите в окно Редактор кода и введите в раздел private заголовок процедуры:

```
procedure FIOReader(N: integer);
```

В раздел описания переменных введите переменную FIORead строкового типа:

```
var  
    FIORead: string;
```

После этого введите текст процедуры следующим образом:

```
procedure TForm4.FIOReader(N: integer);  
begin  
    with DataModule1.TableReader do begin  
        Active := true;
```

```

    Filtered := true;
    Filter := 'Num =' + IntToStr(N);
    First;

    FIORead := FieldValues['Fam'] + ' ' + FieldValues['Name'] + '
' + FieldValues['MidName'];
    if FieldValues['Teacher'] = true then FIORead := FIORead + '
учитель'
    else FIORead := FIORead + '    учащийся класса ' +
FieldValues['Class'];
    end;
end;
end;

```

В теле данной процедуры активизируется связь приложения с таблицей «Reader» базы данных, затем устанавливается фильтр по переданному в процедуру N номеру читателя и выполняется переход на первую отфильтрованную запись. После формирования фамилии и инициалов читателя проверяется его статус по значению логического поля «Teacher». Если читатель не является учащимся, то к строке FIORead приклеивается текст «учитель», если иначе – приклеивается строка «учащийся класса» и название класса.

6. Сохраните проект и проверьте действие созданной процедуры. Запустите приложение, войдите в режим «Библиотекарь: операции с книгами и читателями». Открыв страницу «Книги» на закладке «Списание книги», введите номер книги и щелкните по кнопке «Найти». Если книга с таким номером записана в таблице «Books» базы данных, то на экране будет выведено сообщение «Книга найдена» и отображена кнопка Button4 с надписью «Списать книгу».

Щелкнув по кнопке Button4, проверьте правильность действия процедуры списания книги и вывод сообщения о результате операции.

7. Закройте окно приложения. Открыв в среде MS Access базу данных Library, просмотрите таблицу «Books» и убедитесь в том, что книга с указанным вами номером действительно списана. Закройте MS Access.

Создание процедур обработки событий при поиске читателя, изъятии книг, исключении читателя из базы данных

Упражнение 6.12. Создайте процедуры обработки событий при поиске читателя, изъятии книг, а также исключении читателя из базы данных. Для этого выполните следующие действия.

1. Откройте проект Library-School и, выбрав в списке форм Form4, на странице «Читатели» откройте закладку «Поиск / Удаление / Изъятие книг». Как было показано на рис. 6.18, в окне приложения на данной странице имеются три

кнопки, которыми может воспользоваться пользователь: Button11 с надписью «Обновить», Button12 с надписью «Найти», Button13 с надписью «Удалить» и Button14 с надписью «Изъятие книг».

2. Создайте заготовку процедуры обработки события щелчка по кнопке Button11 с надписью «Обновить» и дополните ее следующим образом:

```
procedure TForm4.Button11Click(Sender: TObject);  
begin  
    Edit24.Text := 'Фамилия';  
    Edit25.Text := 'Имя';  
    Edit26.Text := 'Отчество';  
    Edit27.Text := 'Класс';  
    Edit27.Visible := true;  
    CheckBox2.Checked := false;  
    CheckBox2.Visible := true;  
    Edit28.Visible := false;  
    Edit29.Visible := false;  
    DBNavigator2.Visible := false;  
    DataModule1.TableBook.Filter := '';  
    ListBox1.Items.Clear;  
end;
```

Как видно из текста процедуры, в ней задаются начальные значения полей ввода, очищается фильтр поиска, а также скрываются не нужные в начальный момент компоненты.

3. Создайте заготовку процедуры обработки события щелчка по кнопке Button12 с надписью «Найти» и дополните ее следующим образом:

```
procedure TForm4.Button12Click(Sender: TObject);  
begin  
    DataModule1.TableReader.Filter := '';  
    FlLn := false; FlFn := false; FlMn := false; FlCl := false;  
    if (Edit24.Text <> '') and (Edit24.Text <> 'Фамилия')  
        then FlLn := true;  
    if (Edit25.Text <> '') and (Edit25.Text <> 'Имя')  
        then FlFn := true;
```

```

if (Edit26.Text <> '') and (Edit26.Text <> 'Отчество')
    then FlMn := true;
if CheckBox2.Checked
    then FlCl := true
    else if (Edit27.Text <> '') and (Edit27.Text <> 'Класс')
        then FlCl := true;
if FlLn or FlFn or FlMn or FlCl then
    begin
        with DataModule1.TableReader do begin
            Active := true;
            Filtered := true;
            If FlLn then Filter:='Fam LIKE '''+'*'+Edit24.Text+'*'+''';
            If FlFn then if Filter <> ''
                then Filter:=Filter+'and Name LIKE ''' + '*' +Edit25.Text +
                '*' +''';
            else Filter:= 'Name LIKE ''' + '*' + Edit25.Text + '*' +''';
            If FlMn then if Filter <> '' then Filter := Filter + 'and MidName
            LIKE ''' + '*' +Edit26.Text + '*' +''';
            else Filter:='MidName LIKE ''' + '*' + Edit26.Text + '*' +''';
            If FlCl and (not CheckBox2.Checked) then if Filter <> ''
                then Filter:=Filter+'and Class LIKE ''' +Edit27.Text +''';
                else Filter := 'Class LIKE ''' + Edit27.Text +''';
            If CheckBox2.Checked then if Filter <> ''
                then Filter := Filter + 'and Teacher = true'
                else Filter := 'Teacher = true';
            if RecordCount = 0 then ShowMessage('Читатель не найден!')
                else
                    begin
                        ShowMessage('Читатель найден');
                        First;

```

```
        if RecordCount > 1 then
Form4.DBNavigator2.Visible:=true;
        ShowRecReader;
        Edit24.Text := FieldValues['Fam'];
        FlLn:=false; FlFn:=false; FlMn:=false; FlCl:=false;
        end;
    end;
end
else Showmessage('Не заданы значения!');
end;
```

В начале процедуры выполняется очистка фильтра и проверяется корректность заполнения полей. Если образ поиска задан некорректно, то выводится сообщение «Не заданы значения!».

Если образ поиска задан корректно, то выполняется соединение с таблицей «Reader» базы данных. На основе значений свойств Text полей Edit24-Edit29 и CheckBox2.Checked создается фильтр и выполняется подсчет количества записей, удовлетворяющих заданному фильтру поиска. Если таких записей нет, то выводится сообщение «Читатель не найден!», иначе, если читателей больше одного, отображается компонент DBNavigator2 для управления просмотром нескольких записей. Затем вызывается процедура ShowRecReader, в поле Edit24 отображается фамилия найденного читателя и присваивается значение False логическим переменным FlLn, FlFn, FlMn, FlCl.

4. Для создания процедуры ShowRecReader, которая отображает данные о читателе, перейдите в окно Редактор кода и введите в раздел private заголовок процедуры:

```
procedure ShowRecReader;
```

После этого введите текст процедуры следующим образом:

```
procedure TForm4.ShowRecReader;
begin
    Button13.Visible := true;
    Edit28.Visible := true;
    Edit29.Visible := true;
    with DataModule1.TableReader do begin
        Edit24.Text := FieldValues['Fam'];
        Edit25.Text := FieldValues['Name'];
```

```

Edit26.Text := FieldValues['MidName'];
Edit28.Text := FieldValues['Adress'];
Edit29.Text := FieldValues['Tel'];
if FieldValues['Teacher'] = true then
begin
    CheckBox2.Checked := true;
    CheckBox2.Visible := true;
    Edit27.Visible := false;
end
else
begin
    CheckBox2.Visible := false;
    Edit27.Visible := true;
    Edit27.Text := FieldValues['Class'];
end;

ShowReadersBooks(FieldValues['Num']);

end;
end;

```

Как видно из текста данной процедуры, становятся видимыми компоненты Button13, Edit28 и Edit29, затем в соответствующие поля выводятся значения полей текущей записи о читателе. Если у читателя значение поля FieldValues['Teacher'] = True, то включается и становится видимым флажок CheckBox2, а поле для отображения названия класса становится невидимым.

Если у читателя значение поля FieldValues['Teacher'] = False, то выключается и становится невидимым флажок CheckBox2, а поле для отображения названия класса становится видимым, и в него выводится значение поля FieldValues['Class']. После этого вызывается процедура ShowReadersBooks, которая по номеру читателя отображает книги у него на руках.

5. Для создания процедуры обработки событий щелчков по кнопкам объекта DBNavigator2 дважды щелкните на данном объекте и дополните заготовку процедуры следующим образом:

```

procedure TForm4.DBNavigator2Click(Sender: TObject; Button:
TNavigateBtn);
begin
    case Button of

```

```
nbFirst : begin
    DataModule1.TableReader.First;
    ShowRecReader;
end;

nbPrior : begin
    DataModule1.TableReader.Prior;
    ShowRecReader;
end;

nbNext : begin
    DataModule1.TableReader.Next;
    ShowRecReader;
end;

nbLast : begin
    DataModule1.TableReader.Last;
    ShowRecReader;
end;

end;

end;
```

Как видно из текста этой процедуры, в зависимости от кнопки DBNavigator2, по которой щелкает пользователь, оператор выбора case с помощью селектора Button задает значение свойства DataModule1.TableBook (First, Prior, Next, Last) и вызывает процедуру ShowRecReader, отображающую сведения о записи, соответствующей значению свойства DataModule1.TableBook (First, Prior, Next или Last).

6. Для создания процедуры ShowReadersBooks, которая по номеру читателя отображает в списке ListBox1 книги у него на руках, перейдите в окно Редактор кода и введите в раздел private заголовок процедуры:

```
procedure ShowReadersBooks(N: integer);
```

После этого введите текст процедуры следующим образом:

```
procedure TForm4.ShowReadersBooks(N: integer);
var
I: integer;
```



```

begin
  ListBox1.Items.Clear;
  with DataModule1.TableBook do begin
    Active := true;
    Filtered := true;
    Filter := 'ReaderNo =' + IntToStr(N);
    First;
    for I := 1 to RecordCount do begin
      ListBox1.Items.Add(IntToStr(FieldValues['Num']) + ' '
        + FieldValues['Name'] + ' '
        + FieldValues['Author']);
    end;
  end;
end;
end;
end;

```

Как видно из текста процедуры, вначале выполняется очистка списка ListBox1, затем задается фильтр по номеру читателя и выполняется формирование списка ListBox1 с добавлением в него всех отфильтрованных записей.

7. Создайте заготовку процедуры обработки события щелчка по кнопке Button14 с надписью «Изъятие книг» и дополните ее следующим образом:

```

procedure TForm4.Button14Click(Sender: TObject);
var
  BBB,NS : string;
  NNN,LL : integer;
begin
  BBB := DataModule1.TableReader.FieldValues['Books'];
  LL := Pos(' ',
ListBox1.Items.Strings[ListBox1.ItemIndex]);
  NS :=
Copy(ListBox1.Items.Strings[ListBox1.ItemIndex],1,LL);
  NNN := Pos('#' + NS + '#',BBB);
  Delete(BBB,NNN,LL + 2);
  ListBox1.Items.Delete(ListBox1.ItemIndex);

```

```

DataModule1.TableBook.Edit;
DataModule1.TableReader.Edit;
DataModule1.TableBook.Filter := 'Num =' + NS;
DataModule1.TableBook.First;
DataModule1.TableBook.FieldValues['ReaderNo'] := 0;
DataModule1.TableReader.FieldValues['Books'] := BBB;
DataModule1.TableBook.Post;
DataModule1.TableReader.Post;
DataModule1.TableBook.Filter := '';
end;

```

Как видно из текста процедуры, в разделе описания переменных описаны четыре переменные, используемые во время работы процедуры для хранения временной информации.

Примечание. Знак # вводится программой в строку с инвентарными номерами книг, выданных читателю, чтобы отделить один номер от другого.

В начале процедуры из поля «Books» записи данного читателя в таблице «Reader» базы данных выделяется строка BBB, в которой хранится строка с инвентарными номерами книг, выданных читателю. Затем определяется LL – позиция пробела после номера книги в списке книг, которые читатель не сдал в библиотеку. Исходя из значения LL, из записи книги в списке книг, которые читатель не сдал в библиотеку, получается NS – инвентарный номер книги в формате строки. После этого вычисляется NNN – позиция записи строки '#' + NS + '#' в строке BBB, затем удаляется найденный фрагмент строки BBB, соответствующая запись в списке книг, выданных читателю, и открываются для редактирования таблицы «Books» и «Reader».

Из таблицы «Reader» в записи данного читателя оператором DataModule1.TableReader.FieldValues['Books'] := BBB удаляется запись об изъятной книге, а в таблицу «Books» в записи этой книги присваивается значение 0 полю DataModule1.TableBook.FieldValues['ReaderNo']. Затем применяются изменения в базе данных и очищается фильтр поиска книги по номеру читателя.

8. Создайте заготовку процедуры обработки события щелчка по кнопке Button13 с надписью «Удалить» и дополните ее следующим образом:

```

procedure TForm4.Button13Click(Sender: TObject);
begin
  with DataModule1.TableReader do begin
    If FieldValues['Books'] <> '' then

```

```

        ShowMessage('Читатель не сдал некоторые книги: ' +
FieldValues['Books'])
    else
        begin
            Delete;
            ShowMessage('Читатель удален');
            Button13.Visible := false;
            Filtered := false;
            Filter := '';
        end;
    end;
Edit24.Text := 'Фамилия';
Edit25.Text := 'Имя';
Edit26.Text := 'Отчество';
Edit27.Text := 'Класс';
CheckBox2.Visible := true;
CheckBox2.Checked := false;
Edit28.Visible := false;
Edit29.Visible := false;
DBNavigator2.Visible := false;
end;
```

В данной процедуре сначала проверяется значение поля «Books» текущей записи в таблице «Reader». Если оно не пустое, то выводится сообщение «Читатель не сдал некоторые книги» и перечисляются инвентарные номера книг на руках у читателя. Если книг на руках у читателя нет, то запись этого читателя удаляется из таблицы «Reader», выводится сообщение «Читатель удален», и выполняется очистка фильтра поиска данного читателя. *

В заключительной части процедуры присваиваются начальные значения полям ввода при поиске читателя и скрывается компонент DBNavigator2.

9. Сохраните проект и проверьте действие созданной процедуры. Запустите приложение, войдите в режим «Библиотекарь: операции с книгами и читателями». Открыв страницу «Читатели» на закладке «Поиск / Удаление / Изъятие книг», введите данные о читателе и щелкните по кнопке «Найти». Если такой читатель записан в таблице «Reader» базы данных, то на экране будет выведено сообщение «Читатель найден», отображена кнопка Button13 с надписью «Удалить», а если у читателя имеются книги на руках, то в списке «Книги у читателя» будут отображены краткие сведения о книгах: инвентарный номер, название и автор, как показано на рис. 6.24.

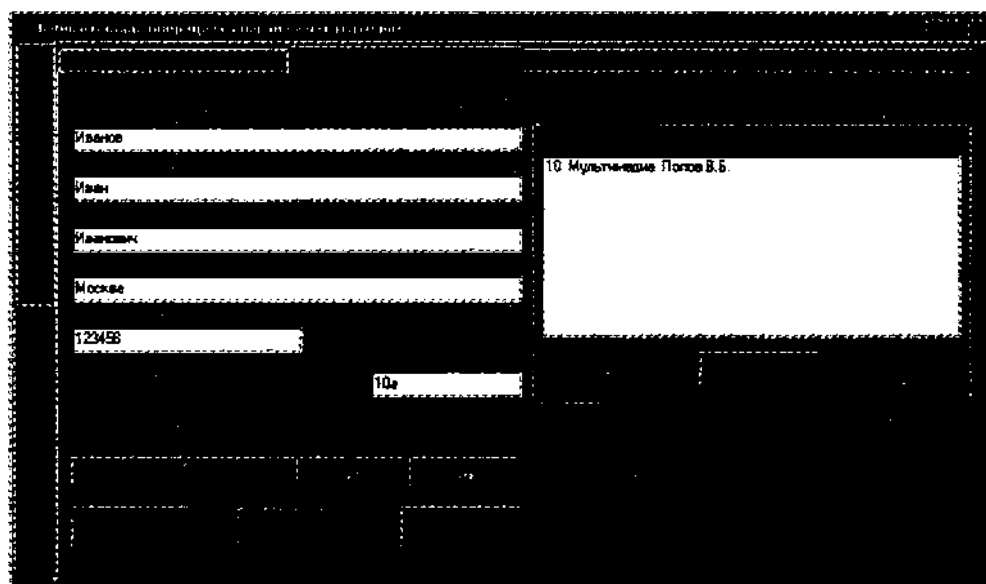


Рис. 6.24. Поиск читателя, изъятие книг и удаление читателя из списков

10. Щелкнув по кнопке Button14 с надписью «Изъятие книг», проверьте правильность действия процедуры изъятия книги у читателя при возврате ее в библиотеку и вывод сообщения о результате операции.

11. Закройте окно приложения. Открыв в среде MS Access базу данных Library, просмотрите таблицу «Reader» и убедитесь в том, что запись о найденном вами читателе действительно удалена из таблицы. Закройте MS Access.

Создание процедур обработки событий при выдаче книг читателю

Упражнение 6.13. Создайте процедуры обработки событий при выдаче книг читателю. Для этого выполните следующие действия.

1. Откройте проект Library-School и, выбрав в списке форм Form4, на странице «Читатели» откройте закладку «Выдача книг».

2. Чтобы создать заготовку процедуры обработки события перехода на закладку «Выдача книг», выберите в окне Инспектор объектов объект TabSheet7 (страница «Читатель», открытая на закладке «Выдача книг»). Открыв вкладку Events, выберите событие OnEnter и дважды щелкните на поле обработчика данного события. После этого дополните заготовку процедуры, заготовленную редактором кода Delphi, следующим образом:

```
procedure TForm4.TabSheet7Enter(Sender: TObject);
begin
  with DataModule1.TableReader do begin
    If Filter <> '' then begin
      Edit30.Text := FieldValues['Fam'];
```

```

    Edit31.Text := FieldValues['Name'];
    Edit32.Text := FieldValues['MidName'];
    Edit33.Text := FieldValues['Adress'];
    Edit34.Text := FieldValues['Tel'];
    if FieldValues['Teacher'] = true then begin
        CheckBox3.Checked := true;
        CheckBox3.Visible := true;
        Edit35.Visible := false;
    end
else
    begin
        CheckBox3.Visible := false;
        Edit35.Visible := true;
        Edit35.Text := FieldValues['Class'];
    end;
    ShowReadersBooks2(FieldValues['Num']);
end
else
    ShowMessage('Переход к поиску читателей');
end;
end;

```

Как видно из текста процедуры, в начале процедуры выполняется проверка значения фильтра поиска, если он пустой, выводится сообщение «Переход к поиску читателей». Если фильтр поиска не пустой, т.е. образ поиска читателя уже задан, то поля ввода заполняются данными о читателе, с помощью процедуры ShowReadersBooks2 выводится список книг на руках у читателя.

3. Создайте заготовку процедуры обработки события щелчка по кнопке Button15 с надписью «Поиск», описывающую поиск книги для выдачи читателю и дополните ее следующим образом:

```

procedure TForm4.Button15Click(Sender: TObject);
var
    FlB: boolean;
    I: integer;

```

```
FB: string;
begin
If Edit36.Text = ''
then ShowMessage('Пожалуйста, введите номер книги')
else
begin
with DataModule1.TableBook do begin
Active := true;
Filtered := true;
Filter := 'Num = ' + Edit36.Text;
if RecordCount = 0 then begin
ShowMessage('Жаль, но нет книги с таким номером');
Edit36.Text := '';
end
else begin
ShowMessage('Книга найдена');
Edit37.Text := FieldValues['Name'];
Edit38.Text := FieldValues['Author'];
Edit39.Text := FieldValues['Publishing'];
Edit40.Text := FieldValues['Year'];
If FieldValues['ReaderNo'] > 0 then begin
ShowMessage('Эта книга на руках');
Edit36.Text := '';
Edit37.Text := '';
Edit38.Text := '';
Edit39.Text := '';
Edit40.Text := '';
end
else begin {Нет ли аналогичной книги на руках у читателя
(чтобы не выдать ему две одинаковые книги)}
FlB := true;
```


Как видно из текста процедуры, после описания локальных переменных проверяется, не является ли пустым Edit36 – поле ввода номера искомой книги. Если свойство Edit36.Text = "", то выводится сообщение «Пожалуйста, введите номер книги».

Если номер искомой книги задан, то активизируется связь с таблицей «Books» базы данных, задается фильтр поиска книги по ее инвентарному номеру. Если книги с таким номером нет, то выводится сообщение «Жаль, но нет книги с таким номером», иначе – выводится сообщение «Книга найдена», и в полях ввода Edit37, Edit38, Edit39, Edit40 выводятся сведения: название, автор, издательство и год издания.

Если значение поля «ReaderNo» записи данной книги в таблице «Books» больше 0, то выводится сообщение «Эта книга на руках», и очищаются поля Edit36, Edit37, Edit38, Edit39 и Edit40.

Если книга в библиотеке имеется, то выполняется проверка, нет ли аналогичной книги на руках у читателя, чтобы не выдать ему две одинаковые книги. Если аналогичная книга уже на руках у данного читателя, то выводится сообщение «Аналогичная книга имеется на руках у читателя».

Если препятствий для выдачи книги нет, то выводится сообщение «Для того чтобы выдать книгу, щелкните по кнопке «Выдать книгу», и эта кнопка отобразится в окне приложения.

4. Создайте заготовку процедуры обработки события щелчка по кнопке Button16 с надписью «Выдать книгу» и дополните ее следующим образом:

```
procedure TForm4.Button16Click(Sender: TObject);  
  
begin  
    DataModule1.TableBook.Edit;  
  
    DataModule1.TableReader.Edit;  
  
    DataModule1.TableBook.FieldValues['ReaderNo'] := DataModule1.  
TableReader.FieldValues['Num'];  
  
    DataModule1.TableBook.Post;  
  
    with DataModule1.TableReader do begin  
        FieldValues['Books'] := FieldValues['Books'] + '#' +  
IntToStr(FieldValues['Num']) + '#';  
  
        Post;  
  
        ShowReadersBooks2(FieldValues['Num']);  
    end;  
  
    Edit36.Text:=''; Edit37.Text:=''; Edit38.Text:='';
```



```

Edit39.Text := ''; Edit40.Text := '';
Button16.Visible := false;
end;

```

В данной процедуре открываются таблицы «Books» и «Reader» базы данных для редактирования. В поле «ReaderNo» записи выбранной книги записывается номер читателя, затем в поле «Books» записи данного читателя добавляется инвентарный номер выданной книги, после этого вызывается процедура ShowReadersBooks2, которой в качестве параметра передается номер читателя.

В заключительной части процедуры очищаются поля Edit36, Edit37, Edit38, Edit39 и Edit40 и становится невидимой кнопка Button16 с надписью «Выдать книгу».

5. Для создания процедуры ShowReadersBooks2, которая после добавления книги читателю отображает в списке ListBox2 по номеру читателя список книг у него на руках, перейдите в окно Редактор кода и введите в раздел private заголовков процедуры:

```

procedure ShowReadersBooks2(N: integer);

```

После этого введите текст процедуры следующим образом:

```

procedure TForm4.ShowReadersBooks2(N: integer);
var
  I: integer;
begin
  ListBox2.Items.Clear;
  with DataModule1.TableBook do begin
    Active := true;
    Filtered := true;
    Filter := 'ReaderNo =' + IntToStr(N);
    First;
    for I := 1 to RecordCount do begin
      ListBox2.Items.Add(IntToStr(FieldValues['Num']) + ' '
        + FieldValues['Name'] + ' '
        + FieldValues['Author']);
      Next;
    end;
  end;
end;
end;

```

Как видно из текста процедуры, вначале выполняется очистка списка ListBox2, затем задается фильтр по номеру читателя и выполняется формирование обновленного в связи с выдачей книги списка ListBox2.

6. Сохраните проект и проверьте действие созданных процедур. Запустите приложение, войдите в режим «Библиотекарь: операции с книгами и читателями». Открыв страницу «Читатели» на закладке «Поиск / Удаление / Изъятие книг», введите данные о читателе и щелкните по кнопке «Найти». Если читатель будет найден, щелкните по закладке «Выдача книг». Обратите внимание, что при переходе на эту закладку будут переданы сведения о читателе.

7. Введите в поле ввода Edit36 инвентарный номер выдаваемой книги и щелкните по кнопке Button15 с надписью «Поиск». После этого будет выполнен поиск книги в таблице «Books» базы данных. Если поиск будет успешным, то, как показано на рис. 6.25, в соответствующие поля раздела «Выдаваемая книга» будут выведены сведения о книге, а на экран будет выведено сообщение.

The screenshot displays a graphical user interface for a library application. On the left, there is a vertical stack of text boxes for entering reader information: 'Иванов', 'Иван', 'Иванович', 'Москва', and '123456'. Below these is a label 'Library_school'. To the right of the reader information is a large rectangular area containing the text '10 Мультимедиа Попов В.Б.'. At the bottom of the interface, there is a section for book details with several text boxes: '12', 'Большая книга сказок', 'Шалаев', 'Слово', and '2006'.

Рис. 6.25. Поиск книги для выдачи читателю

8. Щелкнув по кнопке Button16 с надписью «Выдать книгу», проверьте, как обновляется список книг на руках у читателя.

9. Закройте окно приложения. Открыв в среде MS Access базу данных Library, просмотрите таблицу «Reader» и убедитесь в том, что запись о выданной вами читателю книге действительно добавлена в поле «Books» записи данного читателя. Просмотрите таблицу «Books» и убедитесь в том, что в поле «ReaderNo» записи выбранной книги записан номер читателя, которому она выдана. Закройте MS Access.

Создание процедур обработки событий при выводе списка читателей

Упражнение 6.14. Создайте процедуры обработки событий при выводе списка читателей, а также выводе в файл MS Word списка должников библиотеки. Для этого выполните следующие действия.

1. Откройте проект Library-School и, выбрав в списке форм Form4, на странице «Читатели» откройте закладку «Список читателей».

2. Создайте заготовку процедуры обработки события щелчка по кнопке Button17 с надписью «Все читатели» и дополните ее следующим образом:

```
procedure TForm4.Button17Click(Sender: TObject);  
  
begin  
  
    DataModule1.TableReader.Active := true;  
  
    DataModule1.TableReader.First;  
  
    Form4.QuickRep2.PreviewModal;  
  
end;
```

В теле данной процедуры активизируется связь с таблицей «Reader» базы данных, указатель устанавливается на первой записи таблицы, и вызывается стандартная процедура PreviewModal для вывода сформированного отчета на экран и печати на бумаге.

3. Создайте заготовку процедуры обработки события щелчка по кнопке Button18 с надписью «Должники» и дополните ее следующим образом:

```
procedure TForm4.Button18Click(Sender: TObject);  
  
var  
  
    MSWord: Variant;  
  
    I: integer;  
  
begin  
  
    MSWord := CreateOleObject('Word.Application');  
  
    MSWord.Visible := true;  
  
    MSWord.Documents.Add;  
  
    MSWord.Selection.Font.Size := 16;  
  
    MSWord.Selection.Font.Name := 'Microsoft Sans Serif';  
  
    MSWord.Selection.Font.Bold := true;
```

```
MSWord.Selection.TypeText('Должники' + DateToStr(Date));
MSWord.Selection.TypeText(#13);
MSWord.Selection.Font.Size := 12;
MSWord.Selection.Font.Bold := false;
DataModule1.TableBook.Active := true;
DataModule1.TableReader.Active := true;
DataModule1.TableBook.Filter := '';
DataModule1.TableReader.Filter := '';
DataModule1.TableReader.First;
While not DataModule1.TableReader.Eof do begin
    If (DataModule1.TableReader.FieldValues['books'] <> Null) and
(length(DataModule1.TableReader.FieldValues['books']) > 0) then
        begin
            MSWord.Selection.TypeText('-----
-----' #13);

            MSWord.Selection.TypeText('Дорогой читатель ');
MSWord.Selection.TypeText(DataModule1.TableReader.Fields.
Fields[1].AsString +
' ' + DataModule1.TableReader.Fields.Fields[2].AsString +
' ' + DataModule1.TableReader.Fields.Fields[3].AsString);
            MSWord.Selection.TypeText(#13);

            If DataModule1.TableReader.FieldValues['Teacher'] = false
then
                MSWord.Selection.TypeText('Учащийся ' + DataModule1.
TableReader.Fields.Fields[5].AsString + ' Класс');

            MSWord.Selection.TypeText(#13);
MSWord.Selection.TypeText(DataModule1.TableReader.Fields.
Fields[6].AsString +
' ' + DataModule1.TableReader.Fields.Fields[7].AsString);
            MSWord.Selection.TypeText(#13);
```

```
MSWord.Selection.TypeText('Вы должны библиотеке следующие  
книги:');  
MSWord.Selection.TypeText(#13);  
DataModule1.TableBook.Filter := 'ReaderNo = ' +  
IntToStr(DataModule1.TableReader.FieldValues['Num']);  
DataModule1.TableBook.First;  
For I := 1 to DataModule1.TableBook.RecordCount do  
begin  
MSWord.Selection.TypeText('Инвентарный номер: ' + DataModule1.  
TableBook.Fields.Fields[0].AsString);  
MSWord.Selection.TypeText(#13);  
MSWord.Selection.TypeText('Название: ' + DataModule1.  
TableBook.Fields.Fields[1].AsString);  
MSWord.Selection.TypeText(#13);  
MSWord.Selection.TypeText('Автор: ' + DataModule1.TableBook.  
Fields.Fields[2].AsString);  
MSWord.Selection.TypeText(#13);  
MSWord.Selection.TypeText('Издательство: ' + DataModule1.  
TableBook.Fields.Fields[3].AsString);  
MSWord.Selection.TypeText(#13);  
MSWord.Selection.TypeText('Год: ' + DataModule1.TableBook.  
Fields.Fields[4].AsString);  
MSWord.Selection.TypeText(#13);  
MSWord.Selection.TypeText('Страниц: ' + DataModule1.TableBook.  
Fields.Fields[5].AsString);  
MSWord.Selection.TypeText(#13);  
MSWord.Selection.TypeText('Цена: ' + DataModule1.TableBook.  
Fields.Fields[9].AsString);  
MSWord.Selection.TypeText(#13);
```

```
DataModule1.TableBook.Next;  
  
end;  
  
DataModule1.TableBook.Filter := '';  
  
MSWord.Selection.TypeText('Пожалуйста, сдайте книги до ' +  
DateToStr(Date + 7));  
  
MSWord.Selection.TypeText(#13);  
  
MSWord.Selection.TypeText(' ' + DateToStr(Date) +  
'Администрация библиотеки');  
  
MSWord.Selection.TypeText(#13#13);  
  
end;  
  
DataModule1.TableReader.Next;  
  
end;  
  
MSWord.ActiveDocument.SaveAs('readers.doc'); {Запись файла в  
папку "Мои документы"}  
  
end;
```

Как видно из текста процедуры, в разделе описания переменных введены 6 переменных. В начале процедуры создается переменная типа Variant MSWord, в которой записывается объект, созданный приложением MS Word. Затем задается шрифт и выводятся заголовок документа «Должники» и дата создания документа.

Оператор While not DataModule1.TableReader.Eof do, просматривая всю таблицу «Reader», выбирает записи, в которых поле «books» имеет непустое значение, т.е. у этих читателей не все книги сданы. Для каждого из читателей-должников создается текст, в котором после обращения «Дорогой читатель» выводятся фамилия, имя, отчество читателя и его домашний адрес. Далее выводится текст «Вы должны библиотеке следующие книги:» и перечисляются не сданные читателем книги с указанием инвентарного номера, названия, автора, издательства, года издания, количества страниц и цены.

В заключительной части каждого извещения читателю-должнику печатается напоминание «Пожалуйста, сдайте книги до» и автоматически указывается срок.

В конце каждого извещения проставляется дата высылки извещения и подпись «Администрация библиотеки».

В конце процедуры оператор MSWord.ActiveDocument.SaveAs('readers.doc') выполняет сохранение файла под именем «readers.doc» в папку «Мои документы».

4. Сохраните проект и проверьте действие созданных процедур. Запустите приложение, войдите в режим «Библиотекарь: операции с книгами и читателями». Открыв страницу «Читатели» на закладке «Список читателей», щелкните по

кнопке Button17 с надписью «Все читатели». После этого откроется окно Print Preview, показанное на рис. 6.26, со списком всех читателей, записанных в таблицу «Reader» базы данных библиотеки.

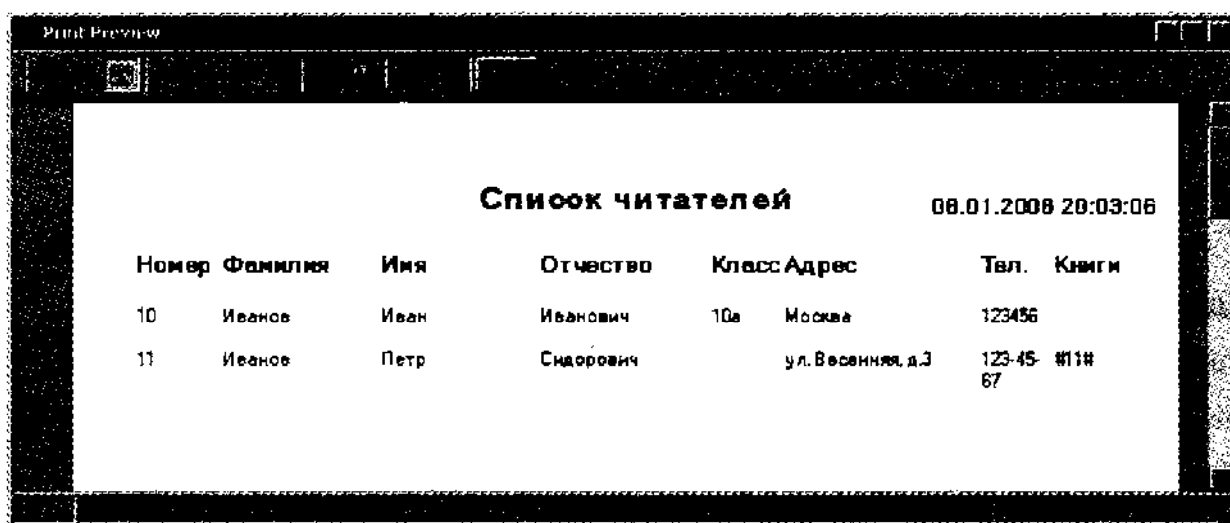


Рис. 6.26. Окно Print Preview со списком всех читателей

5. Открыв страницу «Читатели» на закладке «Список читателей», щелкните по кнопке Button18 с надписью «Должники». После этого откроется окно MS Word, показанное на рис. 6.27. В окне MS Word будет создан документ «readers.doc», сохраненный в папке «Мои документы» и содержащий извещения всем читателям-должникам.

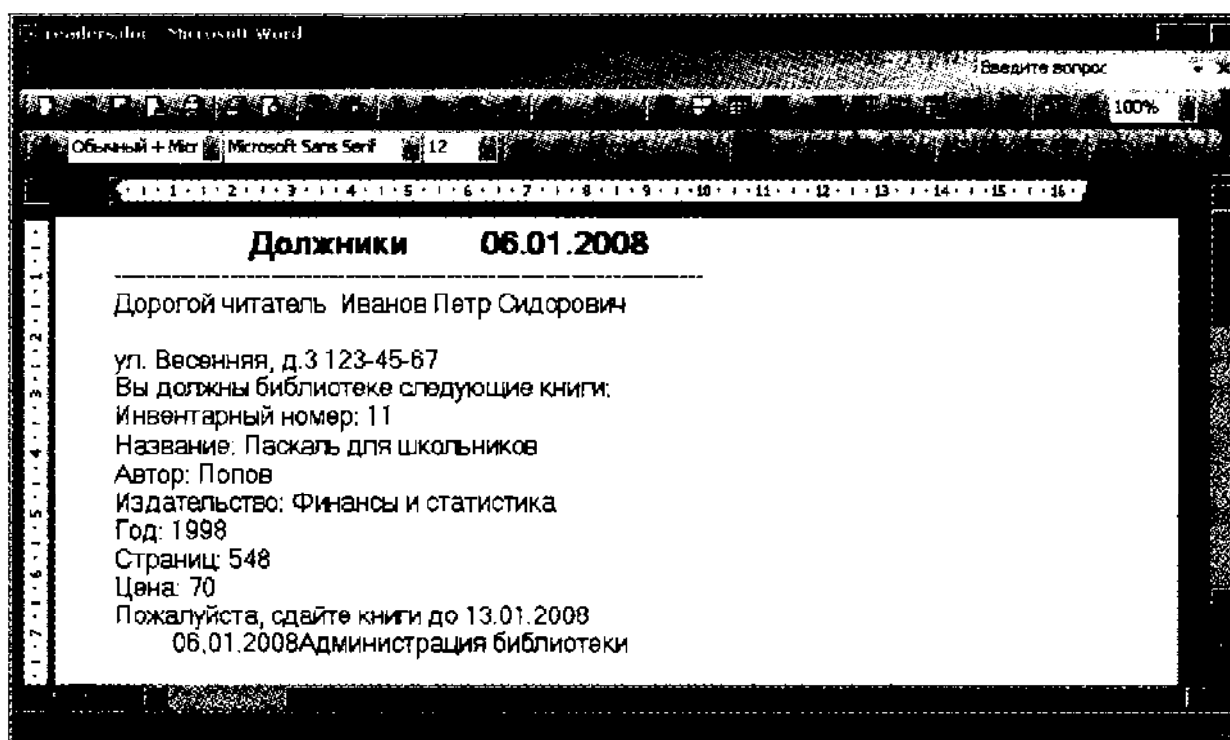


Рис. 6.27. Окно MS Word со списком читателей-должников

6. Закройте приложение Library-School и MS Word. Открыв папку «Мои документы», убедитесь в том, что в ней имеется сохраненный файл «readers.doc», содержащий извещения всем читателям-должникам.

* * *

Итак, уважаемый читатель, мы с вами сделали первые шаги в освоении одной из современных сред разработки приложений. Желаю вам успехов в дальнейшем совершенствовании знаний возможностей Delphi и приобретении практических навыков разработки приложений!

Задания

1. Создать процедуры обработки событий в режиме Библиотекарь: операции с книгами и читателями для компонентов на странице «Книги», открытой на закладке «Поиск книги».

Указание. Использовать опыт создания аналогичных процедур для режима Читатель при выполнении упражнения 6.3.

2. Описать процедуры установки начальных значений полей ввода при событии открытия каждой закладки страниц «Книги» и «Читатели».

3. Добавить в процедуру добавления нового читателя проверку на совпадение следующих сведений «Фамилия, имя, отчество, класс, адрес, телефон». Ввести дополнительное подтверждение библиотекаря для принятия решения о записи такого читателя.

4. Создать в форме Form3 в разделе «Обслуживание базы данных» кнопку «Окончание учебного года», по щелчку на которой будет выполняться перевод учащихся из класса в следующий класс по окончании учебного года.

5. Доработать интерфейс резервирования базы данных и восстановления базы данных с использованием компонента OpenDialog.

6. Создайте справочную систему для приложения Library-School.

Указание. Использовать опыт создания справочной системы в главе 5.

СЛОВАРЬ ТЕРМИНОВ

А

Адрес — идентификатор места в памяти ЭВМ, где хранится или куда может быть записана информация.

Активный диалог — режим взаимодействия пользователя и программной системы, который характеризуется равноправием его участников. Обычно для организации активного диалога используются директивные (командные) языки или языки, близкие к естественным.

Активное окно — окно, с которым в настоящий момент работает или может работать пользователь.

Актуализация данных — приведение данных в соответствие с состоянием отображаемых объектов предметной области. Актуализация реализуется посредством операций добавления, исключения и редактирования записей.

Алгоритм (Algorithm) — четко определенная последовательность действий для достижения поставленной цели за конечное число шагов.

Алфавит — фиксированный для каждого алгоритмического языка набор основных символов, из которых должен состоять любой текст на этом языке. Использование никаких других символов в тексте не допускаются.

Американский стандартный код обмена информацией (ASCII) — стандартная схема кодирования текстовой информации, при которой каждый текстовый или управляющий символ представляется в виде семиразрядного двоичного кода.

Анимация — создание впечатления движения объекта на экране дисплея. В основе анимации лежит быстрая смена последовательно смещаемых друг относительно друга образов.

Аппаратные средства (Hardware) — механические, электрические, электронные узлы и компоненты компьютера.

Аппаратные средства мультимедиа — основные средства: компьютер с высокопроизводительным процессором и памятью большого объема, манипуляторами и мультимедиа-монитором со встроенными стереодинамиками; специальные средства: приводы CD-ROM, TV-тюнеры и карты видеозахвата, графические ускорители, платы видеовоспроизведения, звуковые платы, акустические системы и др.

Аппаратное прерывание — прерывание, причиной которого является событие, произошедшее в аппаратных средствах.

Архитектура компьютера — логическая организация, структура и ресурсы компьютера, которые может использовать программист. Архитектура определяет принципы действия, информационные связи и взаимное соединение основных логических узлов компьютера.

Архитектура клиент – сервер — способ организации взаимодействия программ или компонентов программы, подразумевающий наличие программы или компонента программы, называемого сервером, и одной или нескольких других программ или компонентов, называемых клиентами.

Ассемблер — алгоритмический язык низкого уровня, основной состав операторов которого базируется на машинных командах.

Ассемблер — транслятор с языка ассемблера в машинные команды.

Ассоциативная память — безадресная память, поиск информации в которой производится по ее содержанию (ассоциативному признаку).

Атрибут (Attribute) — в базах данных — имя или структура поля записи. Атрибут характеризует размер или тип информации, содержащейся в поле.

Атрибут данных (Data attribute) — параметр данных, относящийся к их структурным свойствам, используемый для указания контекста данных или придания им смыслового значения.

Б

База данных (БД) (Database) — совокупность связанных данных, организованных по определенным правилам, предусматривающим общие принципы их описания, хранения и манипулирования ими. БД является информационной моделью предметной области. Для создания и ведения БД (обновления информации и обеспечения доступа к ней пользователей) используется набор языковых и программных средств, называемый системой управления БД (СУБД).

Байт — набор из стандартного числа (обычно 8) битов (двоичных единиц), используемый в качестве единицы количества информации при ее передаче, хранении и обработке на ЭВМ. В международных системах кодирования данных байт представляет код одного отображаемого (печатного) или управляющего символа.

Бегунок — небольшой прямоугольник на полосе прокрутки, используемый для перемещения по странице.

Бесконечный цикл — цикл, который не может завершиться нормальным образом.

Бета-версия (Beta version) — версия программного продукта, предшествующая выпуску коммерческого программного продукта. Предоставляется на льготных условиях с целью обкатки и выявления ошибок в новой системе.

Бета-тестирование — пробная эксплуатация программного продукта перед его выпуском на рынок.

Библиотека этапа выполнения (Run-time library) — библиотека процедур, обеспечивающих работу прикладной программы.

Библиотека пользователя — составленное программистом и определенным образом организованное личное собрание программ, подпрограмм, процедур, макроопределений, текстов и наборов данных.

Библиотека стандартных подпрограмм (Library) — совокупность подпрограмм, написанных на одном из языков программирования и удовлетворяющих единым требованиям к структуре, организации их входных и выходных данных и описаниям подпрограмм. Обычно эта библиотека хранится в виде файла во

внешней памяти ЭВМ в рамках той или иной файловой системы, обеспечивающей автоматизированный доступ к отдельным алгоритмам и программам.

Бит — двоичная цифра, принимающая значения 0 или 1. Минимальная единица измерения количества передаваемой или хранимой информации.

Блок (Block) — в алгоритмических языках — программная единица, включающая описания объектов и исполняемые операторы (подпрограмма, функция).

Блок-схема (Flowchart) — в программировании — графическое представление программы или алгоритма с использованием стандартных графических элементов (прямоугольников, ромбов, трапеций и др.), обозначающих команды, действия, данные и т. п.

Буфер обмена — средство операционной системы Windows, предназначенное для обмена данными между приложениями.

В

Ввод данных — процесс записи данных в память компьютера с помощью устройства ввода.

Верификация программы — доказательство правильности программы, т. е. соответствия программы ее спецификации.

Версия — вариант программного продукта или языка программирования.

Вертикальное меню — меню с вертикальным расположением пунктов (один под другим).

Ветвление программы — предусмотренная в программе возможность выполнения тех или иных действий в зависимости от результатов проверки некоторого условия.

Вещественное число — в информатике — тип данных, содержащий числа, записанные с десятичной точкой и/или с десятичным порядком.

Видеопамять — память, предназначенная для записи, хранения и считывания данных, определяющих изображение на экране дисплея.

Видеосистема компьютера — совокупность трех компонентов: монитора, видеоадаптера и драйверов видеосистемы.

Виртуальная машина (Virtual machine) — условная вычислительная система, компоненты которой программным путем формируются или моделируются на основе ресурсов реального компьютера. На одном реальном компьютере может быть построено несколько виртуальных машин, каждая из которых выполняет свою программу и не препятствует работе другой виртуальной машины.

Виртуальная память (Virtual memory) — механизм управления памятью вычислительной системы, позволяющий программе использовать память, размер которой больше реальной оперативной памяти, имеющейся в компьютере. Физическое расположение виртуальной памяти на реальных носителях может не совпадать с логической адресацией данных в прикладной программе. Преобразование логических адресов программы в физические адреса запоминающих устройств обеспечивается аппаратными средствами и операционной системой.

Вложенный цикл — цикл, содержащийся в теле другого цикла.

Внедренный объект — объект, созданный средствами одного приложения и помещенный в документ другого приложения. Пример: диаграмма Microsoft Excel, вставленная в документ Microsoft Word.

Внешнее устройство ЭВМ — периферийное устройство для внешней обработки информации, в отличие от преобразований информации, выполняемых в ЭВМ.

Внешняя память — память, к содержимому которой можно обратиться только с помощью операций ввода-вывода. Реализуется при помощи внешних запоминающих устройств.

Воспроизведение программы или базы данных — согласно законодательству РФ — изготовление одного или более экземпляров программы для ЭВМ или базы данных в любой материальной форме, а также их запись в память ЭВМ.

Входные данные — данные, вводимые в вычислительную систему через устройства ввода для обработки или хранения.

Выходные данные — данные, поступающие из ЭВМ на устройства вывода в результате выполнения программы.

Выражение — конструкция на языке программирования, предназначенная для выполнения вычислений. Выражение может состоять из констант, переменных, указателей функций, объединенных знаками операций. Различают арифметические, логические и символьные выражения.

Выравнивание (Alignment) — способ размещения текста на странице: по правому краю, по левому краю, центрирование.

Г

Главная программа, основная программа — программа, выполняемая первой и управляющая вызовом подпрограмм.

Глобальная переменная (Global variable) — переменная, описание которой вынесено за пределы текущего блока программы. Глобальные переменные описываются в блоке, охватывающем текущий блок.

Графическая информация — сведения или данные, представленные в виде схем, эскизов, изображений, графиков, диаграмм, символов.

Графический адаптер — адаптер дисплея, поддерживающий вывод графической информации.

Графический режим — режим работы адаптера дисплея, обеспечивающий вывод графической информации.

Графическое меню — меню, пункты которого изображаются на экране дисплея в виде кнопок с нарисованными на них значками.

Графический интерфейс пользователя — интерфейс пользователя, основанный на средствах машинной графики. Он предполагает взаимодействие человека с компьютером в форме диалога и использованием ввода и вывода на экран дисплея графической информации, управления программами с помощью кнопок, меню, окон, экранных панелей и других элементов управления, а также выделением цветом определенных частей изображения или текста и т. п. В графическом интерфейсе пользователя используются мышь, световое перо, цифровой планшет

графического ввода и другие средства ведения диалога с компьютером и формирования графических изображений.

Графический объект — модель объекта реального мира, представленная в виде графического изображения.

Д

Данные (Data) — 1. Сведения, полученные путем измерения, наблюдения, логических или арифметических операций, представленные в форме, пригодной для постоянного хранения, передачи и автоматизированной обработки. 2. Информация, подготовленная для печати, хранения и обработки в вычислительной машине, т. е. представленная в символьной (цифровой) форме. Примерами данных являются закодированные для ввода или уже введенные в компьютер текст, речь, изображение, таблицы всевозможных величин и т. п. 3. В контексте отдельной программы или пакета программ термин «данные» означает все обрабатываемые программой объекты, отличные от ее команд.

Двоичная система счисления — позиционная система счисления с основанием, равным 2. Широко применяется для внутреннего представления данных в ЭВМ.

Двоичное число — число, представленное в двоичной системе счисления.

Двоичное представление — представление информации в двоичном коде. Двоичное представление является основным способом представления данных в памяти ЭВМ.

Двоичный код — код, символами которого являются 0 и 1.

Двойной щелчок мышью — действие, заключающееся в двукратном быстром нажатии и отпускании кнопки неподвижной мыши.

Двухмерный массив — матрица. Массив с двумя измерениями (строками и столбцами).

Декомпилирование программы — согласно законодательству РФ — технический прием, включающий преобразование объектного кода в исходный текст в целях изучения структуры и кодирования программы для ЭВМ.

Демонстрационная программа — программа, демонстрирующая интерфейс пользователя с программным продуктом либо возможности программного продукта.

Демонстрационная база данных — база данных небольшого объема, представляющая собой фрагмент реальной БД и предназначенная для демонстрации возможностей СУБД или исходной БД.

Дерево файлов — графическое изображение структуры каталогов, подкаталогов и файлов на диске, отражающее расположение файлов в подкаталогах и каталогах и подкаталогов — в каталогах.

Диалог — двусторонний обмен информацией и управляющими сигналами между человеком и компьютером в форме вопросов и ответов в темпе, удобном для человека. Пользователь ведет диалог при помощи клавиатуры, мыши или микрофона.

Диалоговая система — 1. Система программ и аппаратных средств, обеспечивающая диалоговый режим взаимодействия пользователя с исполняемой

программой. 2. Программная система, управляемая пользователем в диалоговом режиме.

Диалоговый режим (Interactive mode) — способ взаимодействия пользователя или оператора с ЭВМ, при котором происходит непосредственный двусторонний обмен информацией, командами или инструкциями между человеком и ЭВМ. Диалоговый режим подразумевает такую скорость обработки данных, которая не сказывается на технологии действий пользователя. Различают активные и пассивные диалоговые режимы.

Динамическая область памяти — область оперативной памяти, предназначенная для размещения прикладных программ на время их выполнения. После завершения программы динамическая область памяти освобождается для загрузки другой программы.

Динамическое распределение памяти (Dynamic allocation) — управление ресурсами памяти, предполагающее выделение и освобождение памяти во время выполнения программы.

Директива — предложение или команда, управляющие вычислительной системой или ее компонентами. Директива содержит указания системе о необходимости выполнения определенных действий. Директивы могут вводиться оператором ЭВМ или пользователем, а также содержаться в программе.

Дистрибутив — программные продукты в виде, поставляемом производителем (чаще всего на компакт-дисках). Дистрибутивный диск, как правило, содержит саму программу и инсталлятор для установки программы на жестком диске и настройки ее параметров. Иногда применяются дистрибутивные дискеты.

Длина — 1. Размер области памяти, отводимой для хранения объекта (программы, блока данных, записи, машинного слова и т. д.). Измеряется в битах, байтах и т. п. 2. Число элементов, образующих объект. Например, длина блока данных — это и размер в байтах области памяти, отведенной блоку данных, и число машинных слов в блоке; длина файла — это и количество байтов, занимаемых файлом в памяти, и число блоков в файле; длина дорожки — это и информационная емкость дорожки диска, и число размещенных на ней секторов.

Документация к программному продукту, программная документация — совокупность документов, описывающих назначение, структуру и применение программного продукта. Документация предназначена для облегчения использования программного продукта и включает как печатные документы (описание, руководства, учебники, справочники и т. п.), так и электронные документы (выводимые на экран и распечатываемые тексты, а также обучающие программы, средства оперативной справки и т. п.).

Документографическая база данных — база данных, содержащая библиографические описания документов и/или их рефераты.

Доступ — 1. Процедура установления связи с запоминающим устройством и размещенным на нем файлом для чтения, записи или изменения данных. 2. Возможность считать, записать или изменить данные. Обычно в этом случае указывается особенность или способ доступа.

Драйвер устройства (Device driver) — программа, написанная специально для конкретного периферийного устройства с целью обеспечить управление этим устройством со стороны операционной системы.

Драйвер-русификатор — драйвер, поддерживающий ввод в память компьютера и вывод на экран дисплея букв русского алфавита (кириллицы). Обычно является резидентной программой, которая активизируется нажатием либо специальной клавиши, либо сочетания клавиш, что служит командой смены латинского алфавита на русский. При работе в текстовом режиме в знакогенератор адаптера дисплея загружаются изображения соответствующих клавиш клавиатуры.

Дружественный интерфейс — интерфейс, обеспечивающий пользователю максимально удобное взаимодействие с программой или вычислительной системой, не требующее специального обучения. Это наглядные, простые и понятные для него изображения на экране: значки, кнопки, меню, подсказки, звуковое сопровождение и т. п.

Ж

Жизненный цикл программы — совокупность взаимосвязанных и последовательных этапов существования программы. Согласно стандарту ISO/IEC 12207, существуют следующие этапы жизненного цикла программы:

- анализ предметной области и формулировка требований к программе;
- проектирование структуры программы;
- реализация программы в кодах;
- сопровождение программы;
- отказ от использования программы.

З

Завершение программы — последняя фаза обработки программы операционной системой, в ходе которой освобождаются использовавшиеся программой ресурсы, выводятся результаты, очищается выделенная программе оперативная память. Завершение программы может быть нормальным (после успешного выполнения программы) и аварийным.

Зависание — 1. Состояние вычислительной системы, при котором она перестает выдавать результаты и реагировать на запросы извне, например на нажатие пользователем IBM PC клавиш Ctrl и Break или на другие внешние прерывания. Зависание может быть вызвано аппаратным сбоем, ошибками в программном обеспечении или попыткой обращения прикладной программы к устройству, не подключенному к центральному процессору. 2. Состояние, характеризующееся аномально большим временем ожидания задачи или процесса при обращении к какому-либо ресурсу.

Загрузка — считывание данных из одного запоминающего устройства в другое, обычно обладающее большей оперативностью. Чаще всего этот термин обозначает копирование программ и данных из внешнего запоминающего устройства в оперативную память.

Загрузка программы — считывание программы из внешнего запоминающего устройства в оперативную память, настройка адресов и, возможно, запуск. Загрузка программы выполняется под управлением загрузчика — специальной программы, обычно входящей в состав операционной системы.

Загрузочный модуль — программный модуль в виде, пригодном для загрузки и выполнения. Загрузочный модуль получается из объектного модуля после редактирования связей и представляет собой программу в виде последовательности машинных команд, имеющую принятый для данной операционной системы формат.

Задание — совокупность программ и их данных, обрабатываемая операционной системой как единое целое. Содержание задания определяется пользователем в виде последовательности управляющих операторов, определяющих выполняемые программы и используемые ими данные.

Задача — 1. Задание вычислительной системе, представленное в виде программы или части программы и данных, которое операционная система рассматривает как единое целое при распределении ресурсов. В ряде операционных систем понятие «задача» совпадает с понятием «процесс», в других — с понятием «задание». 2. Процесс выполнения этого задания.

Закомментировать — временно отменить действие одного или нескольких операторов программы, превратив их в комментарий. Для этого достаточно ограничить эти операторы в соответствии с синтаксисом комментария, предусмотренным используемым языком программирования. Например, в языке Pascal нужно заключить оператор или последовательность операторов в скобки {...} или (*...*). Таким приемом часто пользуются для локализации ошибки при отладке программы. Чтобы восстановить первоначальный вид программы, достаточно убрать ограничители.

Закраска — в компьютерной графике — изменение цвета пикселей, расположенных внутри изображенного на экране графического объекта.

Заккрытие файла — операция завершения работы программы с файлом. При ее выполнении все связанные с файлом области буферной памяти очищаются, а информация о произведенных изменениях данных заносится во внешнюю память (обычно на диск).

Заливка — в компьютерной графике — однотонная закрашка изображенного на экране плоского графического объекта, ограниченного четким контуром.

Запись — 1. Процесс запоминания (фиксирования) данных в запоминающей среде или на носителе данных (writing). 2. Результат процесса записи (record). 3. Единица обмена данными между программой и внешней памятью, т. е. совокупность данных, совместно пересылаемых на периферийные устройства или с периферийных устройств одним оператором ввода-вывода. Поэтому файлы, хранимые во внешнем запоминающем устройстве, часто представляют собой набор записи.

Запрос — обращение с целью получения ответа. Это может быть сигнал, с помощью которого операционная система запрашивает периферийные устройства об их готовности к обмену данными, либо обращение пользователя к базе данных, содержащее задание на поиск, чтение и обработку нужной информации, либо обращение программы к устройству ввода с целью получения данных.

Запуск программы — загрузка программы в оперативную память и передача управления команде программы, выполняемой первой.

Зарезервированное слово — в языках программирования — слово, смысл которого зафиксирован правилами языка и по которому транслятор (или человек) распознает основные языковые конструкции. Их использование в качестве идентификаторов не разрешается, поскольку они являются зарезервированными (служебными) словами языка.

Знак — отдельный символ алфавита или элемент системы символов, используемой в вычислительной системе.

Знак арифметической операции — знак, определяющий арифметическую операцию. Как правило, это знаки «+», «-», «*», «/».

Значение — смысл или величина, содержащаяся в элементе данных.

Значение по умолчанию — стандартное значение, присваиваемое параметру автоматически, если пользователь не задаст ему иное значение.

Значок — условное изображение программы, операции или информационного объекта на экране. Обычно значки отображаются на кнопках или обозначают пункты меню.

И

Идентификатор — символическое имя переменной или подпрограммы, которое однозначно идентифицирует их в программе.

Изменение размера окна — многие виды окон допускают изменение своих размеров пользователем. Чтобы изменить размер окна, нужно подвести указатель мыши к границе окна, так чтобы он принял соответствующий вид, нажать левую кнопку мыши и перетащить мышью границу окна в нужном направлении.

Импорт — операция переноса данных из одной системы, программы или документа в другую систему, программу или документ.

Индекс — число целого типа или арифметическое выражение, принимающее целочисленное значение, приписываемые элементу массива или другой конструкции данных, для идентификации этого элемента.

Индикатор — 1. Элемент данных, отражающий текущее состояние устройства или данных. Например, поле данных, в которое системная или прикладная программа помещает код ошибки, называют индикатором ошибки. 2. Элемент графического интерфейса пользователя, показывающий ход выполнения какого-либо процесса.

Инициализация — приведение программы или устройства в состояние готовности к использованию. Инициализация программы заключается в задании начальных значений или программных переменных (адресов, счетчиков, переключателей, указателей и т. п.) перед выполнением программы.

Инсталляция, установка — установка программного продукта на компьютер. Инсталляция обычно выполняется под управлением инсталлятора — программы, которая приводит состав и структуру устанавливаемого программного изделия в соответствие с конфигурацией компьютера, а также настраивает программные параметры согласно типу имеющейся операционной системы, классам решаемых

задач и режимам работы. Таким образом, инсталляция делает программный продукт пригодным для использования в данной вычислительной системе и готовым решать определенный класс задач в определенном режиме работы.

Инструкция — 1. В некоторых языках программирования — то же, что команда, предложение или оператор. 2. Документ, указывающий и определяющий порядок работы при инсталляции и эксплуатации программного продукта.

Инструментальные средства — программные средства (сервисные программы, пакеты прикладных программ, оболочки), предназначенные для автоматизации создания, редактирования и отладки различных программных продуктов: программ, баз данных и знаний, электронных таблиц, текстов, графических изображений и т. д. Например, текстовый редактор, позволяющий набирать текст программы, отладчик, облегчающий программисту выполнение отладки разрабатываемой программы, библиотека программ, облегчающая разработку пользовательского интерфейса программы, и т. д.

Интегрированная среда — система, включающая все необходимые пользователю программные средства и обеспечивающая единообразное взаимодействие с ними. Примером интегрированной среды может служить среда разработки программ Delphi вместе с набором приложений, обеспечивающим потребности программистов при создании программ.

Интерпретатор — транслятор (программа или устройство), анализирующий команды или операторы исходной программы, написанной на алгоритмическом языке высокого уровня, и немедленно выполняющий их. Таким образом, интерпретатор одновременно и транслирует, и выполняет заданную программу, делая это покомандно или пооператорно.

Интерпретация — метод выполнения программы, при котором каждый отдельно взятый оператор транслируется и сразу выполняется, после чего осуществляется переход к следующему оператору. Интерпретация выполняется с помощью программных или аппаратных средств, называемых интерпретаторами. Достоинством интерпретации является возможность пошагового прослеживания и выполнения программы, недостатком — относительно малая скорость выполнения.

Интерфейс (Interface) — программное или аппаратное обеспечение коммуникации между компьютером и его пользователем или между двумя устройствами.

Интерфейс пользователя (User interface) — интерфейс, определяющий процессы взаимодействия пользователя с информационным ресурсом.

Информация (Information) — согласно законодательству РФ — сведения о лицах, предметах, фактах, событиях, явлениях и процессах независимо от формы их представления. Информация уменьшает степень неопределенности, неполноту знаний о лицах, предметах, событиях и т. д.

Информационный продукт (Information product) — документированная информация, подготовленная в соответствии с потребностями пользователей и представленная в форме товара. Информационными продуктами являются программные продукты, базы и банки данных и другая информация.

Исключение — прием программирования, используемый при отладке программ и для повышения надежности программ. Если программисту известно,

что при правильной работе программы должны выполняться некоторые условия, то невыполнение этих условий свидетельствует о неправильной работе программы.

Исполняемый оператор — оператор в программе, которому соответствует одна или несколько последовательных операций, составляющих алгоритм решения задачи. Например, оператор присваивания или оператор цикла.

Исполняемый файл — файл, готовый к выполнению операционной системой. В IBM PC исполняемые файлы имеют расширения *exe*, *com* и *bat*. Файлы с расширениями *exe* и *com* — это программы. Файлы с расширением *bat* — командные файлы.

Испытание программы, тестирование программы — проверка программы или ее составной части путем реального выполнения специально подобранных контрольных задач. Испытание программы является важным этапом разработки программы и представляет собой проведение испытательных прогонов программы с целью убедиться, что она действительно решает ту задачу, для которой предназначена, и выдает правильный ответ при любых условиях.

Исходные данные — данные, необходимые для решения задачи. Исходные данные подготавливаются заранее и используются программой в ходе ее выполнения. Исходные данные вводятся с помощью операторов ввода программы или пользователем в диалоговом режиме. При этом они становятся входными данными. Исходные данные для работы подпрограммы передаются ей в основном через механизм формальных и фактических параметров.

Исходный код — текст программы на алгоритмическом языке. В компьютере исходный текст либо непосредственно выполняется интерпретатором, либо предварительно переводится компилятором в стандартный загрузочный код, способный многократно исполняться в определенной вычислительной среде.

Итерационный цикл — оператор цикла, для которого число повторений тела цикла заранее неизвестно. В итерационном цикле на каждом шаге вычислений происходят последовательное приближение и проверка условия достижения искомого результата. Выход из итерационного цикла осуществляется в случае выполнения заданного условия.

К

Каталог, папка — 1) в широком смысле — перечень (оглавление) наборов данных, расположенных на внешнем носителе. В оглавлении обычно указываются имя набора данных, его размер, дата создания и местоположение. 2) оглавление файлов, доступное пользователю посредством интерпретатора команд операционной системы. Обычно система файловых каталогов имеет иерархическую структуру.

Килобайт — единица измерения объема информации, численно равная 1024 байтам.

Клавиша оперативного вызова, «горячие» клавиши — клавиша или сочетание нескольких клавиш, одновременное нажатие на которые вызывает определенные действия программы или вычислительной системы.

Ключевое слово — слово или сочетание слов естественного языка, отражающее содержание текста.

Ключ — элемент данных, выражающий местоположение записи или группы в файле.

Кнопка — в интерактивных системах — элемент управления графического интерфейса пользователя, выбор которого с помощью указателя мыши вызывает определенное действие системы.

Код — 1. Система символов и однозначных правил их интерпретации, используемая для представления информации в виде данных. В современных компьютерах широкое распространение получил двоичный код. В нем каждая буква, цифра, знак препинания, пробел или какой-либо другой символ однозначно представляются (кодируются) комбинацией двоичных цифр — нулей и единиц. 2. Программа, записанная на языке программирования. В таком смысле термин «код» применяется для того, чтобы подчеркнуть отличие готовой программы от ее блок-схемы или наброска, выполненного на некотором условном языке.

Кодирование символов — представление набора символов в виде последовательности цифр. Наиболее часто используемыми символами являются буквы, знаки, цифры и т. п., которые представляются двоичными числами.

Колонка, столбец — вертикальный ряд объектов одной природы. Обычно колонки образуют элементы данных или их позиции, расположенные на экране, бумаге или другом носителе вертикально — друг под другом. Например, столбец элементов матрицы, колонки ячеек электронной таблицы.

Команда машинная — описание элементарной операции, которую должен выполнить компьютер. Команды хранятся в ячейках памяти в двоичном коде. В общем случае команда содержит код выполняемой операции, указания по определению операндов (или их адресов) и указания по размещению получаемого результата. Результат выполнения команды вырабатывается по точно определенным правилам, заложенным в конструкцию компьютера. В зависимости от количества операндов различают одноадресные, двухадресные, трехадресные и переменнo-адресные команды.

Команда ввода — машинная команда, определяющая ввод данных с внешнего устройства в основную память.

Команда возврата — команда перехода, предписывающая выход из подпрограммы и возврат в вызывающую программу.

Команда вывода — машинная команда, определяющая вывод данных из основной памяти во внешнее устройство.

Команда операционной системы — предписание, инициирующее выполнение операционной системой определенных действий, например создание нового каталога или печать файла. Команда операционной системы, как правило, состоит из имени команды и, возможно, параметров, содержащих необходимую уточняющую информацию.

Команда паузы — команда, предписывающая временное прекращение выполнения программы.

Команда прерывания — команда, вызывающая прерывание.

Командная строка — строка экрана дисплея, предназначенная для записи вводимых пользователем команд операционной системы и запуска программ с клавиатуры.

Комментарий — ограниченный специальным образом фрагмент программы, предназначенный для чтения человеком и игнорируемый при трансляции. Комментарий служит для включения в текст программы пояснений, облегчающих человеку ее чтение и анализ. В каждом языке программирования имеется свой синтаксис комментария.

Коммерческий программный продукт — программный продукт, изготовленный для продажи.

Компилятор (программа) — транслятор, выполняющий компиляцию программных модулей. В отличие от интерпретатора компилятор только преобразует программу, составленную на языке программирования высокого уровня, в программу на машинном языке или языке, близком к машинному, не участвуя в ее исполнении. На вход компилятора поступает исходный модуль, который после компиляции преобразуется в объектный модуль.

Компиляция — трансляция программы или отдельного программного модуля, написанного на языке программирования высокого уровня (исходная программа, исходный модуль) в программу или модуль на машинном языке или языке, близком к машинному (объектная программа, объектный модуль). В процессе компиляции программа преобразуется в промежуточную форму, к которой впоследствии необходимо присоединить библиотечные средства, содержащие стандартные подпрограммы и процедуры, а если нужно, то можно добавить любые другие модули, написанные самим пользователем, скомпилированные в объектные модули, возможно, с иных языков высокого уровня.

Компонент программы — часть программы, которая структурно отделена от остальных частей и может использоваться в нескольких программах.

Компоновка — процесс сборки загрузочного модуля из полученных в результате раздельной компиляции объектных модулей с автоматическим поиском и присоединением библиотечных подпрограмм и процедур. В процессе компоновки программа собирается в единое целое.

Компьютер — программируемое электронное устройство, способное обрабатывать данные и производить вычисления, а также выполнять другие задачи манипулирования символами.

Конвертор — программа для перекодирования данных из одного машинного кода в другой или из одного формата в другой.

Конец файла — символ, указывающий конец текстового файла.

Константа — элемент данных, присутствующий в тексте программы и не меняющий своего значения при многократном ее использовании.

Контекстное меню — меню, которое содержит команды, применимые в данном контексте. Например, во многих приложениях Windows, если выделить какой-либо объект и щелкнуть правой кнопкой мыши, то появится контекстное меню, содержащее команды, применимые к выделенному объекту в его текущем состоянии.

Контекстно-зависимая справка — справочная система, которая выдает пользователю справочную информацию с учетом текущего контекста его работы. На-

пример, если выделен некоторый объект, то при нажатии клавиши F1 выводится список разделов справки, имеющих отношение к работе с объектами данного типа.

Л

Линейный алгоритм — набор инструкций, выполняемых последовательно.

Листинг — печатный документ, формируемый вычислительной системой и содержащий данные о ходе и результатах разных этапов выполнения задачи. Например, листинг, формируемый на этапе трансляции, обычно содержит текст исходной программы и результаты трансляции (сведения об ошибках или характеристики полученного объектного модуля).

Логическая переменная — переменная, принимающая только логические значения: «истина» или «ложь», которые в ЭВМ могут быть представлены в виде 1 и 0. В языках программирования эти значения обычно обозначаются 1, true, T, или 0, false, F соответственно. Логические переменные вводятся в программу путем описания переменной, в которой указываются идентификатор (имя) переменной и ключевое слово, определяющее логический тип. В качестве ключевых слов в описаниях логических переменных применяются: logical — в Фортране и Boolean — в Паскале.

Логическая запись — совокупность записей взаимосвязанных элементов данных, рассматриваемая в логическом плане как единое целое. Одна логическая запись может состоять из нескольких физических или быть частью одной физической записи.

Логический диск — область хранения информации на жестком диске, обозначаемая латинскими буквами C, B, E и т. д. Разбиение на логические диски условно и служит для удобства работы с жестким диском.

Логический тип, булевский тип — тип данных, принимающих только логическое значение «истина» или «ложь», над которыми производятся логические операции.

Логическое выражение, булево выражение — совокупность одной или нескольких логических переменных, логических функций и отношений, соединенных знаками логических операций и скобками.

Локальная база данных — база данных, размещенная на одном или нескольких носителях на одном компьютере.

Локальная переменная — переменная с ограниченной областью определения в программе. Локальная переменная существует только в том модуле или блоке, где она описана, а вне его является недоступной.

М

Макрокоманда, макрос — 1. Выражение в тексте программы, вместо которого подставляется текст, заданный макроопределением. Макроопределения часто используются в макроассемблерах. 2. Последовательность команд, запускаемая одним нажатием клавиши на клавиатуре или кнопки на экране дисплея.

Маска — шаблонная комбинация разрядов или символов, используемая для анализа данных. Обычно с помощью маски проводятся выделения, сравнения или исключения отдельных частей данных.

Массив — именованная последовательность однотипных элементов, число которых фиксировано. Каждый элемент массива однозначно определяется именем массива и набором целых чисел, называемых индексами массива.

Машинный язык — совокупность машинных команд, отличающихся количеством операндов в команде, назначением информации, задаваемой в операндах, и набором операций, которые может выполнить компьютер.

Машинная команда — команда, которая может быть распознана и выполнена центральным процессором, т. е. команда, входящая в систему команд конкретной ЭВМ.

Машинная программа — программа на машинном языке. Машинная программа является последовательностью машинных команд и может быть распознана и выполнена центральным процессором.

Машинное время — время, затрачиваемое ЭВМ на выполнение определенного комплекса вычислений.

Машинное слово — последовательность битов или символов определенной длины, воспринимаемая памятью, арифметическим устройством или устройством управления ЭВМ как единое целое, имеющее определенное содержание.

Машинный сбой, сбой — кратковременный отказ или ошибка в работе аппаратных средств вычислительной системы.

Мегабайт — единица количества информации; 1 Мбайт = 1024 Кбайт = 1 048 576 байт.

Меню (Menu) — изображаемый на экране дисплея список режимов, команд или вариантов ответа, предлагаемых пользователю для выбора. Предлагаемые варианты называются пунктами меню. Они определяют последующие действия системы. Если один из пунктов меню выделен другим цветом (цветовым маркером), значит, меню активно. Меню является одним из основных элементов графического интерфейса пользователя и одним из средств реализации интерактивного режима взаимодействия пользователя с вычислительной системой.

Метка — 1. Целое число без знака или идентификатор, приписанный оператору программы и используемый в других частях программы для обращения к этому оператору. Метка обычно ставится перед оператором и либо отделяется от него двоеточием, как в Паскале и Си, либо точкой. Оператор, снабженный меткой, называют помеченным оператором. 2. Физическая запись на внешнем носителе данных, определяющая начало или конец файла или тома.

Модификация программы или базы данных — согласно законодательству РФ — любые изменения программы для ЭВМ или базы данных, не являющиеся адаптацией.

Мультимедиа — программные и аппаратные средства, обеспечивающие воспроизведение видеоинформации с соответствующим звуковым сопровождением.

Мусор — ненужные, не подлежащие дальнейшему использованию данные в памяти ЭВМ. Мусор напрасно занимает место в памяти, поэтому ее следует регулярно очищать от мусора. Эту процедуру называют сборкой мусора.

Н

Навигатор — программа поиска записей в базе данных или в информационных массивах.

Наследование — понятие объектно-ориентированного программирования, которое состоит в том, что класс, определяемый на основе другого класса, наследует все или некоторые свойства и методы родительского класса.

Неактивная программа — загруженная в оперативную память программа, которая в текущий момент не управляет центральным процессором. Например, ожидающая обращения резидентная программа.

Неактивное окно — окно, не используемое пользователем в настоящий момент. Неактивное окно может быть полностью или частично перекрыто другими окнами. Чтобы сделать его активным, как правило, достаточно навести на него указатель мыши и нажать левую кнопку.

О

Оболочка (Shell) — программа, создаваемая для упрощения работы со сложными программными системами. Оболочка преобразует неудобный командный пользовательский интерфейс в дружелюбный графический интерфейс или интерфейс типа меню. Обычно оболочка реализуется в виде отдельной программы.

Обработка данных — выполнение определенных действий над данными по заданной программе.

Обработка данных в реальном масштабе времени — обработка данных, протекающая с такой же скоростью, что и моделируемые события.

Обрабатывающая программа — программа, обрабатывающая данные или другие программы, например текстовый редактор, компилятор, редактор связей, загрузчик.

Обработка прерываний — стандартные действия, выполняемые операционной системой или аппаратными средствами при возникновении прерываний.

Объект — предмет или явление, которому можно присвоить название.

Объектно-ориентированное программирование (ООП) — наиболее популярная в настоящее время методология программирования, являющаяся развитием структурного программирования. Центральной идеей ООП является инкапсуляция, т. е. структурирование программы на модули особого вида, объединяющее данные и процедуры их обработки, причем внутренние данные модуля могут быть обработаны только предусмотренными для этого процедурами.

Объектный модуль — программный модуль, являющийся результатом компиляции исходного модуля. Объектный модуль представляет собой последовательность машинных команд, готовую к объединению с другими объектными модулями с помощью редактора связей (компоновщика).

Объектографическая база данных — фактографическая база данных, содержащая расширенный набор данных о сложных объектах предметной области.

Одномерный массив — массив с одним индексом.

Окно — выделенная часть экрана дисплея, с которой программа или пользователь работает как с отдельным независимым экраном, размеры и расположение которого пользователь может изменять. Различают несколько типов окон: окно приложения, окно документа, диалоговое окно.

Оператор — 1. Допустимое в языке программирования высокого уровня предложение, задающее целостное законченное действие ЭВМ или представляющее набор описаний. Типичными операторами в традиционных языках программирования являются операторы ввода-вывода, присваивания, перехода, цикла, процедуры и др. Грамматическая конструкция каждого из них определяется синтаксисом конкретного языка программирования.

Оператор ввода — оператор в программе, предписывающий передачу данных из устройства ввода или внешней памяти в оперативную память и делающий эти данные доступными программе.

Оператор выбора, переключатель — оператор в программе, определяющий выбор одной из нескольких ветвей алгоритма. Например, с помощью оператора выбора программируется меню (выбор пользователем последовательности действий в соответствии с пунктом меню).

Оператор вывода — оператор в программе, предписывающий передачу данных из оперативной памяти во внешнюю память или на устройство вывода.

Оператор присваивания — оператор, передающий значение арифметического, логического или другого выражения одной или нескольким переменным либо имени функций.

Оператор цикла — оператор, упрощающий программирование цикла. Оператор цикла формально состоит из заголовка и тела цикла. Заголовок цикла указывает на последовательность повторяемых операторов, образующих тело цикла, и либо определяет множество значений параметра цикла и предписывает многократное выполнение тела цикла при этих значениях параметра, либо определяет условие повторного выполнения тела цикла.

Операция — действие, выполняемое над данными.

Операционная система (ОС) — комплекс программ, обеспечивающий выполнение других программ, распределение ресурсов, планирование, ввод-вывод данных, управление данными, взаимодействие с оператором.

Описание — раздел программы, идентифицирующий структуры данных, которыми должна манипулировать программа, и описывающий их типы.

Описание алгоритмического языка — документ, специфицирующий возможности алгоритмического языка. Обычно описание содержит алфавит допустимых символов и ключевых слов, синтаксические правила построения из алфавита допустимых конструкций языка и семантику, объясняющую смысл и назначение конструкций языка.

Опция — элемент меню; один из предлагаемых вариантов выбора.

Останов — прекращение автоматической выборки и выполнения команд центральным процессором. Останов может быть вызван причинами, связанными с выходом из строя одного из устройств ЭВМ или операционной системы, — аварийный останов, а может быть программируемым — предусмотренным выполняемой программой.

Открытие файла — операция начала работы программы с файлом. «Открыть файл» — означает связать программу с файлом: установить соответствие между физическим файлом на внешнем запоминающем устройстве, буферной памятью и набором данных программы, заполнить соответствующие поля в управляющих таблицах операционной системы и т. д.

Открытый файл — файл, для которого выполнена операция открытия файла, в результате чего его записи стали доступными для чтения и обработки.

Отладка программы (Debugging) — этап разработки программы, состоящий в выявлении и устранении программных ошибок, факт существования которых уже установлен.

Отладчик (Debugger) — программа, позволяющая исследовать внутреннее устройство рабочей программы. Отладчик обеспечивает пошаговое выполнение программы, просмотр текущих значений переменных, вычисление значения любого выражения в программе и другие функции.

П

Пакет прикладных программ — комплект программ, предназначенных для решения задач из определенной проблемной области. Обычно применение пакета прикладных программ предполагает наличие специальной документации: лицензионного свидетельства, паспорта, инструкции пользователя и т. п.

Пакетная обработка, пакетный режим — обработка данных или выполнение заранее подготовленных заданий без участия пользователя.

Палитра — 1. Множество цветов, которые можно отобразить на экране дисплея. Монитор и видеоадаптер компьютера определяют разрешающую способность экрана и максимально возможное количество отображаемых на нем цветов. Однако имеющееся программное обеспечение может не использовать всех графических возможностей, предоставляемых аппаратурой, и ограничить число цветов, доступных для формирования изображения. Цвет каждого пикселя в машинной программе задается двоичным кодом. Поэтому количество цветов в палитре определяется числом разрядов, отведенным для представления цвета одного пикселя. Например, если для представления цвета пикселя отводится 4 разряда, то пиксель может иметь 16 цветов (коды от 0000 да 1111). Следовательно, 4-разрядный пиксель допускает шестнадцатичетную палитру. 2. Таблица соответствия между кодами цветов в программах рисования или набор цветов на экране дисплея. 3. Набор цветов в программах рисования или набор цветов и инструментов рисования в программах раскраски.

Панель инструментов, инструментальная панель — панель экрана с размещенным на ней графическим меню.

Панель экрана, экранная панель, панель — выделенная часть экрана дисплея или окна, предназначенная для размещения меню, управляющих кнопок, справочной и другой информации.

Параметр цикла, переменная цикла, управляющая переменная цикла — переменная, используемая в операторе цикла для управления повторением тела цикла.

Пассивный диалог — режим взаимодействия пользователя и программной системы, инициатива ведения которого принадлежит программной системе. При этом программная система ведет за собой пользователя, запрашивая в точках ветвления вычислительного процесса дополнительную информацию, необходимую для принятия заложенных в алгоритм решений. В пассивном диалоге программная система обеспечивает пользователя информационными сообщениями и подсказками, облегчающими использование диалоговой системы. Запросы к пользователю строятся обычно либо в виде меню, либо в виде шаблонов.

Паскаль — язык программирования высокого уровня. Первая версия языка Pascal, разработанная швейцарским ученым Н. Виртом, была опубликована в 1968 г. и предназначалась для обучения программированию как систематической дисциплине.

Переменная (Variable) — именованный элемент данных в программе. Переменные различаются по имени и могут принимать разные значения. Значение переменной может быть получено и изменено программой. Тип данных, к которому могут принадлежать значения переменной, устанавливается описанием переменной.

Переносимость программы, мобильность программы — свойство программы, позволяющее переносить ее без изменений или с минимальными модификациями с одного компьютера на другой, в особенности когда эти ЭВМ имеют различную архитектуру.

Переполнение — ситуация, при которой результат операции превышает по абсолютному значению наибольшую представимую в вычислительной системе величину. Обычно переполнение приводит к аварийному завершению задачи.

Периферийные устройства, внешние устройства — часть технического обеспечения, конструктивно отделенная от основного блока компьютера.

Периферийные устройства — комплекс устройств для внешней обработки данных, обеспечивающий их подготовку, ввод, хранение, управление, защиту, вывод и передачу на расстояние по каналам связи.

Пиксель — элемент графической информации. Минимальный независимый участок изображения, для которого можно задать цвет, яркость и другие характеристики. Пикселем называются точки, на которые делится экран дисплея при графическом режиме работы, или точки, из которых формируется изображение на струйных или лазерных принтерах.

Платформа (Platform) — в зависимости от контекста — либо комбинация аппаратуры и операционной системы, либо аппаратура, включая тип процессора.

Подменю — меню, вызываемое выбором пункта меню вышележащего уровня. Обычно пункты меню, в результате выбора которых раскрывается подменю, помечаются многоточием или стилизованной стрелкой.

Подпрограмма — выделенная часть программы, реализующая определенный алгоритм и допускающая обращение из разных мест остальной части программы. Применение подпрограммы сокращает текст программы, если на разных этапах решения задачи требуется выполнить один и тот же алгоритм, так как при этом исключается необходимость многократного повторения в программе одних и тех же групп команд или операторов.

Поле данных, поле (Field) — часть записи или заполняемой формы, имеющая функционально самостоятельное значение и обрабатываемая как отдельный элемент данных.

Полное имя файла — составное имя, включающее последовательно записываемые через разделитель «\» имя диска, последовательность имен вложенных каталогов, через которые проходит путь к файлу (начиная с высшего в их иерархии), и имя файла с расширением. Таким образом, полное имя файла состоит из маршрута поиска файла и имени файла с расширением.

Полнотекстовая база данных — база данных, в которой хранятся записи полнотекстовых документов или их частей.

Полоса прокрутки — элемент управления, служащий для управления прокруткой изображения. Обычно это расположенная на границе окна полоса с бегунком, перемещая который с помощью мыши можно переместить изображение или текст в окне по вертикали или по горизонтали.

Последовательность команд (Command sequence) — в алгоритмических языках — совокупность операторов или зарезервированных слов.

Последовательный доступ — 1. Способ доступа к данным, при котором записи файла считываются в том же порядке, в котором они были записаны при его создании. Такая логическая последовательность записей вовсе не означает, что и физически записи расположены последовательно, например, для файла, размещенного на магнитном диске, хотя логическая и физическая последовательности часто совпадают. 2. Способ доступа, при котором данные считываются в оперативную память в порядке их физического размещения на носителе данных внешнего запоминающего устройства. При этом для обращения к нужной записи необходимо последовательно «просмотреть» все участки носителя начиная либо с самого первого, либо со следующего за тем, к которому было предыдущее обращение. Такой способ доступа осуществляется, например, к данным, хранящимся на магнитных лентах.

Прерывание — временное прекращение выполнения команд программы с сохранением информации о ее текущем состоянии и передаче управления специальной программе — обработчику прерываний.

Прерывание ввода-вывода — внешнее прерывание, связанное с работой устройства ввода-вывода.

Прикладная программа, приложение (Application) — программа, реализующая обработку данных в определенной области применения.

Прикладное программное обеспечение — программное обеспечение, состоящее из отдельных прикладных программ и пакетов прикладных программ, предназначенных для решения различных задач пользователей и созданных на их основе автоматизированных систем.

Проблемно-ориентированная база данных — база данных, содержащая тематически связанные документы и/или данные, предназначенные для решения прикладных задач определенного вида.

Программа (Program) — последовательность машинных команд, предназначенная для достижения конкретного результата.

Программное обеспечение (Software) — совокупность входящих в состав вычислительной системы программных средств: программ, данных и докумен-

тов. Программное обеспечение предназначено для эффективной работы ЭВМ и предоставляет пользователю определенные виды обслуживания.

Программное прерывание — прерывание, вызванное либо обращением программы к аппаратным средствам (например, ввод исходных данных, вывод результатов, очистка экрана и т. п.), либо некорректным представлением или использованием команд или данных.

Программный модуль — программа или часть программы, оформленная в виде, допускающем ее независимую трансляцию.

Программный продукт, программное изделие — программа (пакет программ), предназначенная для продажи или передачи в эксплуатацию другим лицам и удовлетворяющая ряду требований, важнейшие из которых следующие: 1) сама программа и предлагаемая к ней инструкция должны содержать достаточное количество данных для своего полноценного использования; 2) программа должна сопровождаться производителем, т. е. замеченные ошибки должны устраняться бесплатно для покупателей; 3) программа должна поставляться в удобном для установки и использования виде, как правило, на гибких или лазерных дисках с инструкцией и в защитной упаковке; 4) программа должна быть изготовлена с помощью законно приобретенных программных средств и запатентована.

Программирование — процесс подготовки задач для их решения с помощью компьютера; процесс составления программ.

Прокрутка изображения — вертикальное или горизонтальное перемещение изображения в окне экрана. Управление прокруткой изображения осуществляется клавишами управления курсором или с помощью мыши путем перемещения бегунков на полосах прокрутки.

Процессор — устройство компьютера, выполняющее команды. Обязательными компонентами процессора являются арифметико-логическое устройство и устройство управления. Процессор характеризуется архитектурой, набором выполняемых команд, скоростью их выполнения и длиной машинного слова. Архитектура определяет типы обрабатываемых данных, регистры, стеки и систему адресации. Набор команд может быть фиксированным или допускать возможность микропрограммирования.

Прямой доступ — способ доступа к данным, при котором все элементы данных или записи в файле являются равнодоступными, т. е. время доступа к элементу или записи не зависит от расположения данных в памяти и для доступа к любому из них не требуется «просмотр» других элементов или записей.

Псевдокод — система обозначений и правил, предназначенная для единообразной записи алгоритмов. Псевдокод занимает промежуточное место между естественным и формальным языками.

Пункт меню, элемент меню — один из вариантов, предлагаемых пользователю для выбора. Каждый пункт меню обозначает какое-то действие или набор действий системы, поэтому, выбирая тот или иной пункт меню, пользователь тем самым определяет последующие действия системы.

Путь к файлу — данные, указывающие операционной системе место в памяти, где следует искать файл. Обычно путь к файлу указывает логическое имя

накопителя и последовательность имен вложенных каталогов, в последнем из которых содержится нужный файл.

Р

Рабочий лист — формализованная анкета, предназначенная для обработки и записи структурированных данных. Рабочий лист содержит состав полей данных, соответствующих виду обрабатываемых документов или данных, а также набор сведений об их содержании и правилах заполнения.

Размерность массива — количество индексов, необходимое для однозначной идентификации любого элемента массива.

Размер шрифта, кегль шрифта — высота символов шрифта, измеряемая в пунктах (1 пт = 1/72 дюйма, или примерно 0,36 мм). Наиболее ходовым размером шрифта считается 10 пт. Более мелкие шрифты (8 пт и менее) трудно читаются.

Раскрывающееся меню (Pull-down menu) — список команд меню, раскрывающийся из строки меню и остающийся доступным пользователю по мере необходимости.

Раскрывающийся список — элемент управления, предназначенный для выбора одного или нескольких значений из списка значений. Аналогичен элементу управления «список», но занимает меньше места на экране.

Распространение программы или базы данных — согласно законодательству РФ предоставление доступа к воспроизведенной в любой материальной форме программе или базе данных, в том числе сетевыми и иными способами, а также путем продажи, проката, сдачи внаем, предоставления взаймы, включая импорт для любой из этих целей.

Расширение файла — последовательность символов, предназначенных для идентификации типа файла. Обычно расширение состоит не более чем из трех символов, отделяемых от имени файла точкой.

Редактирование — внесение изменений в текст или преобразование программ или данных к виду, требуемому для их дальнейшего использования.

Резидентная программа — программа, постоянно присутствующая в оперативной памяти. После считывания в оперативную память резидентная программа не выполняется до конца, а остается в состоянии готовности к продолжению выполнения до своей выгрузки, перезагрузки или выключения компьютера. Она активизируется и выполняет заданные действия либо при обращении к ней, например, нажатием определенной клавиши или сочетания клавиш, либо при достижении компьютером определенного состояния (например, по сигналу таймера).

Рекурсивная подпрограмма — подпрограмма, которая вызывает сама себя.

С

Семантика — система правил истолкования отдельных языковых конструкций. Семантика определяет смысловое значение предложений алгоритмического языка.

Сервер (Server) — в широком смысле — объект, предоставляющий сервис другим объектам по их запросам.

Сервер — в информационных сетях — компьютер или программная система, предоставляющая удаленный доступ к своим службам или ресурсам с целью обмена информацией.

Сетка — совокупность вертикальных и горизонтальных координатных линий, на фоне которых размещаются какие-либо графические объекты.

Синтаксис языка программирования — совокупность правил написания чисел, переменных, выражений, операторов, процедур и других элементов и предложений (синтаксических конструкций) данного языка программирования.

Синтаксическая ошибка — ошибка в программе, связанная с нарушением синтаксиса языка программирования.

Система программирования — система для разработки программ на конкретном языке программирования. Система программирования предоставляет пользователю специальные средства разработки программ: транслятор, специальный редактор текстов программ, библиотеки стандартных подпрограмм, отладчик и др.

Система управления базами данных — комплекс программных и лингвистических средств общего или специального назначения, реализующий поддержку создания баз данных, централизованного управления и организации доступа к ним различных пользователей в условиях принятой технологии обработки данных.

Системные программы — программы общего пользования, выполняемые вместе с прикладными программами и служащие для управления ресурсами компьютера: центральным процессором, памятью, вводом-выводом.

Слово — не содержащая пробелов последовательность символов некоторого алфавита, имеющая определенное смысловое значение.

Сообщение (Message) — 1. Информация о ходе или состоянии вычислительного процесса, выдаваемая пользователю компонентами вычислительной системы. 2. Порция данных, оформленная для передачи данных принятым в данной сети ЭВМ или системе компьютерной связи образом.

Сопровождение программного изделия — меры, направленные на поддержание в работоспособном состоянии программы, находящейся в эксплуатации: устранение выявленных в ходе эксплуатации ошибок и внесение изменений в программу с целью улучшения ее функциональных возможностей в соответствии с изменением предъявляемых к ней требований.

Сортировка данных, сортировка, упорядочение — распределение элементов данных по группам в соответствии с определенными правилами.

Сохранение — запись группы данных из оперативной памяти в файл, находящийся во внешней памяти.

Специальные клавиши — группа клавиш, предназначенных для управления работой компьютера или для вызова стандартных действий.

Список — элемент управления, предназначенный для выбора одного или нескольких значений из их предопределенного набора.

Справка, справочная система — программное средство, предназначенное для предоставления пользователю информации о некотором приложении и о порядке работы с ним в интерактивном режиме, т. е. непосредственно во время работы с приложением.

Среда программирования — интегрированная система разработки программ. В среде программирования все программные средства имеют единый пользовательский интерфейс, общую базу данных и не требуют специального вызова, так что программист может выполнять свою работу, не выходя в операционную систему.

Ссылка — имя, указатель или адрес в программе, указывающий на объект программы, подпрограмму, другую программу или устройство.

Строка, строка символов — 1. Последовательность слов, букв, цифр или других знаков, написанных в одну линию. Например, битовая строка — последовательность двоичных цифр (нулей и единиц). 2. Тип данных, значениями которого являются последовательности знаков. Обычно реализуется в виде одномерного массива переменной длины, минимальный размер которого равен единице, а максимальный размер (длина строки), может изменяться. 3. Горизонтальная линия на экране дисплея или на бумаге, заполненная последовательностью знаков или предназначенная для такого заполнения. Например, командная строка, строка подсказки или строка текста при работе в текстовом редакторе либо напечатанная на принтере.

Строка меню — элемент управления в графическом интерфейсе пользователя, реализующий функции горизонтального меню.

Строка подсказки — в интерактивных системах — строка на экране дисплея, содержащая доступные команды и разъясняющая их смысл. Например, при наведении указателя мыши на одну из кнопок графического меню в строке подсказки появляется текст, поясняющий, какие действия вызовет «нажатие» этой кнопки.

Структура данных (Data structure) — организационная схема записи или массива, в соответствии с которой упорядочены данные с тем, чтобы их можно было интерпретировать и выполнять над ними определенные операции.

Структура базы данных — принцип или порядок организации записей в базе данных и связей между ними.

Структурное программирование — метод разработки программ, требующий разбиения программы на небольшие независимые части (модули). Структурное программирование обеспечивает возможность проведения строгого доказательства правильности программ.

Сценарий — 1. План работы программы во взаимодействии с пользователем. Например, создание компьютерной игры начинается с разработки сценария, описывающего, что и в какой последовательности должна выполнять ЭВМ в зависимости от игровой ситуации и реакции играющего. 2. Данные, описывающие последовательность связанных событий.

Т

Таймер — программируемое устройство отсчета времени, часы. Таймер обеспечивает измерение астрономического времени и позволяет предварительно установить временной интервал, по истечении которого может быть выдан соответствующий сигнал. Этот сигнал можно использовать в программах для прерывания или начала некоторых запрограммированных действий.

Текстовое поле — элемент управления, предназначенный для ввода, отображения и редактирования небольших текстовых значений.

Текстовый редактор — программа для ввода и изменения текстовых данных: документов, книг, программ и т. д. Редактор обеспечивает модификацию строк текста, контекстный поиск и замену частей текста, автоматическую нумерацию страниц, обработку и нумерацию сносок, выравнивание абзацев, проверку правописания слов, построение оглавлений, распечатку текста на принтере.

Текстовый режим — режим работы адаптера дисплея, при котором на экране выводятся изображения только текстовых и псевдографических символов.

Текстовый файл — 1. Файл, содержащий текст. 2. Файл, состоящий исключительно из кода ASCII.

Текущий дисковод — дисковод, с которым в данный момент непосредственно работает операционная система. В IBM PC это дисковод, с которым в настоящий момент работает пользователь.

Текущий каталог — каталог, файлы которого в данный момент непосредственно доступны программам и пользователю.

Тело процедуры — исполняемая часть процедуры, представляющая последовательность операторов, реализующую алгоритм процедуры.

Тело цикла — последовательность повторяемых в цикле команд или операторов.

Тест (Test) — совокупность входных данных для программы, а также точное описание всех результатов, которые должна выработать программа на этих данных.

Тестирование (Testing) — этап разработки программы, в процессе которого проверяется работоспособность программы, не содержащей явных ошибок.

Тип данных (Data type) — характеристика набора данных, которая определяет диапазон возможных значений данных из набора, допустимые операции, которые можно выполнять над этими значениями, способ хранения этих значений в памяти.

Тип файла — характеристика файла, отражающая его назначение и область применения, например командный, текстовый, Pascal-программа и т. п.

Точка возврата — место в вызывающей программе, в которое осуществляется возврат из подпрограммы после ее завершения.

Транслятор — программа, преобразующая текст, написанный на алгоритмическом языке, в программу, состоящую из машинных команд.

Трансляция программы — перевод программы с одного языка программирования на другой. Обычно трансляция является преобразованием программы, написанной на машинно-независимом языке, в эквивалентную программу на машинном языке конкретной ЭВМ.

Трассировка программы — выполнение программы или ее участка, сопровождающееся выводом на экран, принтер или другой регистрацией в хронологической последовательности информации о событиях, связанных с выполнением программы. Трассировка применяется при отладке или тестировании программы, когда программа пользователя или ее отлаживаемый участок выполняется под управлением специальной программы-трассировщика.

У

Указатель мыши — значок на экране дисплея, передвигающийся при движении мыши по плоскости.

Управляющая клавиша — клавиша, которая, будучи нажатой в сочетании с другой клавишей, придает последней альтернативный смысл.

Условие выхода из цикла, условие завершения цикла — условие, в зависимости от выполнения которого происходит или повторение выполнения тела цикла, или выход из цикла.

Ф

Файл (File) — совокупность связанных записей, хранящихся во внешней памяти компьютера и рассматриваемых как единое целое. Каждый файл однозначно идентифицируется указанием имени файла, его расширения и пути доступа к нему.

Файл исходных данных — совокупность записей, расположенных в порядке их получения в пункте записи.

Файл последовательного доступа, последовательный файл — файл, записи которого можно читать только последовательно, друг за другом, в порядке их расположения.

Файл прямого доступа — файл, записи которого можно читать в произвольном порядке. В таком файле каждая запись снабжена своим номером — ключом, по которому и осуществляется поиск нужной записи.

Файл расчетных данных — совокупность записей, элементы которой получены путем обработки исходных или инвертированных файлов.

Флажок — элемент управления, предназначенный для выбора одного из двух возможных значений.

Форма — окно, которое служит основой проектируемого приложения Delphi. На форме размещаются компоненты — элементы создаваемого приложения.

Формат (Format) — структура информационного объекта. Формат определяет способ расположения и представления данных в таблицах, базах данных, принтерах, блоках данных и других информационных объектах.

Форматирование текста — процесс придания тексту определенного вида, связанный с определением левой и правой границ текста, абзацного отступа и т. д.

Ц

Центральный процессор (Central Processing Unit) — основной рабочий компонент компьютера, который выполняет арифметические и логические операции, заданные программой, управляет вычислительным процессом и координирует работу всех устройств компьютера.

Цикл — оператор языка программирования, позволяющий многократно повторять одну и ту же последовательность команд (тело цикла).

Цикл со счетчиком, арифметический цикл — при выполнении такого цикла определенная часть программы периодически повторяется, а число повторений (проходов) регистрируется счетчиком. По достижении нужного количества проходов выполнение цикла прекращается. Текущее значение счетчика часто используется в теле цикла. Циклы со счетчиком могут быть как циклами с предусловием, так и циклами с постусловием.

Цикл с постусловием, цикл «до» — цикл, в котором проверка условия выхода из цикла осуществляется в конце выполнения тела цикла.

Цикл с предусловием, цикл «пока» — цикл, в котором проверка условия выхода из цикла осуществляется в начале выполнения тела цикла.

Ш

Шаблон — в информатике — формализованный кадр изображения, выводимый на экран дисплея и содержащий тексты запросов к пользователю и специальные поля, предназначенные для занесения туда ответов пользователя (текстов или чисел).

Шаблон имени файла — последовательность знаков, позволяющих обозначить имена сразу нескольких файлов. Шаблон имени файла образуется из символов, присутствующих в имени файла, и знаков, указывающих, что на их месте в имени файла символы либо отсутствуют, либо могут стоять любые допустимые символы.

Шрифт — конкретный способ изображения символов из некоторого набора. Обычно в шрифт входят буквы одного или нескольких алфавитов, цифры и специальные знаки.

Щ

Щелчок — действие, заключающееся в быстром нажатии и отпускании кнопки неподвижной мыши. Щелчок применяется при задании команды с помощью элемента управления, быстрой активизации окон и в других случаях. При этом указатель мыши должен указывать на выбранный элемент или программу.

Э

Электронные таблицы (Spreadsheet) — компьютерная программа, поддерживающая представление данных в виде таблиц, состоящих из строк и граф, на пересечении которых располагаются клетки (ячейки таблицы). Значение в числовой клетке таблицы либо указывается в явном виде, либо рассчитывается по ассоциированной с клеткой формуле. Электронные таблицы являются инструментом анализа финансовой информации.

Элемент данных — неделимый информационный элемент, являющийся минимальной структурной единицей информации. Вид элемента данных определяется характером содержащихся в нем сведений и особенностью его организации или записи.

Элемент управления — средство графического интерфейса пользователя, предназначенное для ввода-вывода информации и управления работой программы.

В настоящее время используются множество различных элементов управления, например кнопки, флажки, переключатели, поля, списки, полосы прокрутки.

Я

Язык ассемблера (Assembler) — алгоритмический язык низкого уровня, основной состав исполнительных операторов которого базируется на машинных командах.

Язык программирования (Programming language) — искусственный (формальный) язык, предназначенный для записи алгоритмов. Язык программирования задается своим описанием и реализуется в виде специальной программы: компилятора или интерпретатора.

Язык программирования высокого уровня — язык программирования, в который введены элементы, допускающие описание задачи в наглядном, легко воспринимаемом виде, упрощающие и автоматизирующие процесс программирования. Управляющие конструкции и структуры данных такого языка отражают естественные для человека понятия, а не архитектуру вычислительной системы.

Язык описания данных — язык для описания физической и/или логической структуры данных.

ПРИЛОЖЕНИЕ

Справочные сведения о среде программирования Delphi 6

Для проектирования, запуска и тестирования приложений Delphi используется интегрированная среда разработки (ИСП) (Integrated Development Environment – IDE), в которой объединены редактор кодов, отладчик, инструментальные панели, редактор изображений, инструментарий баз данных, – все, с чем приходится работать программисту при создании приложения. В результате интеграции разработчик получает гармоничный набор инструментов, дополняющих друг друга. В ИСП имеется возможность расширять меню, включая в него вызов необходимых вам дополнительных программ, в том числе и собственных.

Своим внешним видом среда программирования Delphi отличается от многих других сред программирования, например MDI-приложений Turbo Pascal 7.0 и Word for Windows.

Примечание. MDI (Multiple Document Interface) определяет особый способ управления несколькими дочерними окнами внутри одного большого окна.

Среда Delphi является SDI-приложением (Single Document Interface) и состоит из нескольких отдельно расположенных окон. Это было сделано из-за того, что SDI близок к той модели приложений, что используется в Windows 95/98.

В главе 2 было рассмотрено назначение основных элементов ИСП Delphi 6. Здесь же рассмотрим назначение меню ИСП Delphi, а также некоторых элементов, не описанных ранее.

Система меню

Полоса главного меню ИСП Delphi отображается в верхней части окна (рис.П.1).



Рис. П.1. Меню ИСП Delphi 6

Назначение пунктов меню можно уточнить в справочной системе Delphi. Для получения справки Delphi о пункте меню выберите интересующий вас пункт меню, например команду Run в меню Run, и нажмите клавишу F1. После этого откроется окно Delphi Help, в котором будет выведена справочная информация.

Выбор команды меню выполняется любым из стандартных способов: нажатием F10, Alt+«горячая клавиша» или щелчком мыши по нужному пункту меню. Ниже представлено назначение команд меню.

Назначение команд меню UCP Delphi 6

Меню File – разделы меню позволяют создать новый проект, новую форму, открыть ранее созданные проект или форму, сохранить проекты или формы в файлах с заданными именами.

New – создать новый объект из элементов депоzitария.

Open – открыть существующий модуль или проект.

Open Project – открыть существующий проект.

Reopen – открыть проект или модуль, с которыми работали раньше.

Save – сохранить текущий модуль.

Save As – сохранить текущий модуль под новым именем.

Save Project As – сохранить текущий проект под новым именем.

Save All – сохранить текущий проект и все его файлы.

Close – закрыть текущий модуль.

Close All – закрыть все файлы текущего проекта или группы.

Use Unit – сформировать оператор uses.

Print – напечатать текст модуля или форму.

Exit – завершить работу UCP Delphi.

Меню Edit – разделы меню позволяют выполнять обычные для приложений Windows операции обмена с буфером Clipboard, а также дают возможность выравнивать группы размещенных на форме компонентов по размерам и местоположению.

Undo – отмена операции редактирования.

Redo – отказаться от отмены. Действует немедленно после команды Отмена.

Cut – вырезать.

Copy – копировать.

Paste – вставить.

Delete – удалить.

Select All – выделить все.

Align to Grid – выровнять выделенные компоненты по сетке. Чтобы выбрать больше чем один компонент, нажмите Shift при щелчке на каждом или, нажав левую кнопку мыши, выделите его выравниваемые компоненты.

Bring to Front – переместить избранный компонент на передний план (перед всеми другими компонентами на форме).

Send to Back – переместить избранный компонент на задний план (позади всех других компонентов на форме).

Align – открыть диалоговое окно Выравнивание.

Size – открыть диалоговое окно Размер, чтобы задать одинаковые размеры нескольких компонентов.

Scale – открыть диалоговое окно Масштаб, чтобы пропорционально изменить размеры всех компонентов на текущей форме.

Tab Order – открыть диалоговое окно Edit Tab Order, чтобы изменить порядок перехода между компонентами на форме или внутри избранного компонента, если этот компонент содержит другие компоненты.

Creation Order – открыть диалоговое окно Creation Order, чтобы определить порядок, в котором ваше приложение создает невизуальные компоненты, когда вы загружаете форму во время разработки или выполнения.

Flip Children – зеркально повернуть компоненты в текущей форме справа налево.

Lock Controls – зафиксировать все компоненты на активной форме в их текущем положении.

Add to Interface – добавить новую процедуру, функцию или свойства для ActiveX-компонента. Эти элементы будут добавлены в интерфейс ActiveX-компонента, а также обеспечен их доступ к другим приложениям.

Меню Search – разделы меню позволяют осуществлять поиск фрагментов текста, ошибок, объектов, модулей, переменных и символов в редакторе кодов.

Find – найти.

Find in Files – найти в файлах.

Replace – заменить.

Search Again – искать снова.

Incremental Search – быстрый поиск вводимых символов.

Go to Line Number – перейти на строку с заданным номером.

Find Error – найти ошибку выполнения.

Browse Symbol – просмотр информации об идентификаторе.

Меню View – разделы меню позволяют вывести на экран или скрыть различные элементы среды проектирования Delphi и открыть окна, связанные с интегрированным отладчиком.

Project Manager – активизирует окно Менеджера проекта (Project Manager).

Object Inspector – активизирует окно Инспектор объектов (Object Inspector).

Alignment Palette – палитра выравнивания.

Browser – окно наследования.

Code Explorer – активизирует окно Исследователь кода, помогающего анализировать текст модуля.

Component List – список компонентов.

Window List – список открытых окон.

Toggle Form/Unit – переключение между формой и кодом ее модуля.

Units – список модулей проекта.

Forms – список форм проекта.

Type Library – библиотека типов.

New Edit Window – новое окно редактора кода.

Toolbars – делает видимыми быстрые кнопки, если до этого они были невидимы.

Меню Project – позволяет добавлять и убирать из проекта формы, задавать опции проекта, компилировать проект без его выполнения и делать много других полезных операций.

Add To Project – добавить в проект.

Remove from Project – удалить из проекта.

Add To Repository – добавить текущую форму в депозитарий.

View Source – занести в окно Редактор кода исходный файл проекта.

Add New Project – эта команда создает новый проект выбором из депозитария и добавляет его в текущую группу проектов.

Compile – компилировать все файлы проекта, которые были изменены с момента последнего выполнения программы.

Build All – компилировать все компоненты, модули, формы вне зависимости от того, изменялось ли что-нибудь с момента последнего выполнения вашей программы.

Syntax Check – проверить синтаксис приложения без генерации исполняемого файла.

Information – дать информацию о размерах приложения: числе строк кода, размере выполняемого модуля, размере стека и т. п.

Web Deploy – развернуть спроектированную активную форму на сервере Web.

Options – установить опции проекта (опции компиляции, редактирования связей) и задать каталоги в диалоговом окне опций проекта Project Options.

Меню Run – предоставляет возможность выполнять проект в нормальном или отладочном режимах, продвигаясь по шагам, останавливаясь в указанных точках кода, просматривая значения переменных и т.д.

Run – выполнить приложение; если до этого не была осуществлена компиляция программы в ее текущем состоянии, то перед запуском эта компиляция выполняется.

Parameters – позволяет задать параметры командной строки, необходимые при запуске приложения.

Register ActiveX Server – зарегистрировать в Windows активный сервер ActiveX.

Un-Register ActiveX Server – снять с регистрации в Windows активный сервер ActiveX; тем самым экземпляр вашего активного элемента удаляется из системы.

Inspect – открыть окно Инспектор отладки.

Step Over – выполнить приложение по шагам без захода в функции.

Trace Into – выполнить приложение по шагам с заходом в функции.

Trace to Next Source Line – выполнить приложение до следующей выполняемой команды.

Run to Cursor – выполнить приложение вплоть до точки в исходном тексте, в которой находится курсор.

Show Execution Point – показать точку выполнения.

Program Pause – пауза в выполнении приложения.

Program Reset – завершить выполнение приложения и выгрузить его из памяти.

Add Watch – добавить переменную в окно наблюдения.

Add Breakpoint – добавить точку прерывания.

Evaluate/Modify – изменить значение переменной в процессе выполнения приложения.

Меню Component – содержит выпадающее меню, которое позволяет работать с компонентами: создавать новые компоненты, изменять палитру компонентов и т. п.

New Component – создать новый компонент; активизируется Эксперт компонентов (Component Expert), который помогает создавать новые компоненты Delphi.

Install Component – установить новый компонент Delphi в новый или уже существующий пакет.

Import ActiveX Library – импортировать элемент ActiveX, который уже зарегистрирован в системе, в новый или уже существующий пакет Delphi.

Create Component Template – создать шаблон компонента; этот раздел становится активным, когда вы выделили на форме более одного компонента; он позволяет вам скомбинировать их в один компонент, который может рассматриваться на форме как единое целое.

Install Packages – показать список всех имеющихся пакетов и указать, какие пакеты должны компилироваться в ваше приложение; можно также видеть текущий список компонентов каждого пакета, редактировать состав пакетов.

Configure Palette – конфигурировать палитру компонентов, видимую на экране.

Меню Database – в Delphi 6 – 3 позволяет использовать инструментарий для работы с базами данных. В Delphi 1 инструментарий для работы с базами данных включен в меню Tools (инструментарий).

Explore – открыть SQL Explorer для создания, просмотра и редактирования данных.

SQL Monitor – контроль SQL-запросов.

Form Wizard – мастер форм базы данных, позволяет легко генерировать форму, представляющую данные из любой базы данных.

Меню Tools – включает ряд разделов, позволяющих выполнять различные вспомогательные программы, например: вызывать Редактор изображений (Image Editor), работать с программами, конфигурирующими базы данных и сети, и т.д. Кроме того, в это меню вы можете сами включить любые разделы, вызывающие те или иные приложения, и, таким образом, расширить возможности главного меню Delphi, приспособив его для своих задач.

Environment Options – вызов диалогового окна настройки окружения Environmental Options, позволяющего изменять установки редактора, экрана, палитры и опций просмотра.

Repository – вызов диалогового окна просмотра депозитария Object Repository, позволяющего просмотреть помещенные в депозитарий объекты, добавлять объекты в депозитарий и удалять их оттуда.

Configure Tool – настройка раздела меню Tools, добавление в него новых разделов по вашему желанию.

Меню Window – содержит список открытых окон ICP Delphi и предоставляет возможность перехода из одного окна в другое.

Меню Help – содержит разделы, помогающие работать со справочной системой Delphi.

Contents – содержание справки.

Keyword Search – предметный указатель справки.

What's New – справка о новых возможностях Delphi.

Getting Started – начальное обучение.

Using Object Pascal – справка по Object Pascal.

Developing Applications – справка по разработке приложений.

Object and Component Reference – справка по объектам и компонентам библиотеки.

Borland Home Page – просмотр Web-страницы Borland.

Delphi Home Page – просмотр страницы Web Delphi.

Borland Programs and Services – просмотр страниц Web Programs и Services.

About... – справка о версии Delphi.

Примечание. Справочник Delphi является контекстно-зависимым: при нажатии клавиши F1 вы получите подсказку, соответствующую текущей ситуации. Например, находясь в Инспекторе объектов, выберите какое-нибудь свойство и нажмите F1 – вы получите справку о назначении данного свойства. Если в любой момент работы в среде Delphi возникает неясность или затруднение, нажмите F1 и прочтите справку.

Помимо главного меню в Delphi имеется система контекстных всплывающих меню, которые появляются, если пользователь поместил курсор мыши на тот или иной компонент и щелкнул правой кнопкой мыши. Большинство разделов этих контекстных меню дублирует основные разделы главного меню.

Палитра компонентов

Палитра компонентов – это витрина библиотеки визуальных компонентов (Visual Component Library – VCL). Она использует постраничную группировку объектов и позволяет сгруппировать компоненты в соответствии с их смыслом и назначением. Поскольку число предопределенных компонентов возрастает от версии к версии, то наиболее полной является библиотека Delphi 6 (см. рис. 4.3). Как видно на этом рисунке, вверху Палитры компонентов находится набор закладок – Standard, Additional, Win32, System и т.д. Если вы щелкнете мышью по одной из закладок, то можете перейти на следующую страницу Палитры компонентов. Набор и порядок компонентов на каждой странице являются конфигурируемыми, т.е. вы можете добавить к имеющимся компонентам новые, изменить их количество и порядок.

Основная палитра компонентов Delphi имеет двенадцать страниц. Кратко рассмотрим их содержание.

Standard (стандартная страница). Большинство компонентов на этой странице являются аналогами экранных элементов операционной системы Windows: меню, кнопки, полосы прокрутки, панели и т.п. (рис. П.2).



Рис. П.2. Палитра компонентов Standard

Имена компонентов, соответствующих той или иной пиктограмме на палитре, вы можете узнать из всплывающей подсказки, появляющейся, если вы задержите над этой пиктограммой курсор мыши.

Примечание. Имена на ярлычках выглядят, например, так: MainMenu, Button и т.д. Однако в Delphi все имена классов в действительности начинаются с символа «Т», например TMainMenu, TButton. Под такими именами во встроенной в Delphi справочной системе вы сможете найти описания соответствующих компонентов.

Назначение компонентов можно уточнить, используя систему контекстной помощи Delphi. Для этого следует выделить нужную страницу в палитре компонентов и нажать клавишу F1. После этого откроется окно Delphi Help, содержащее справочную информацию. Для просмотра информации в окне используйте линейку прокрутки. Если вы хотите посмотреть справку о конкретном компоненте, выберите нужную ссылку. Если вы выберете в палитре компонент и нажмете клавишу F1, вам будет показана справка по типу данного компонента.

Additional (дополнительная страница). Страница содержит более развитые компоненты (рис. П.3). Например, компонент Outline удобен для отображения информации с иерархической структурой, а компонент MediaPlayer позволит вашим программам воспроизводить звук, музыку и видео. Данная страница также содержит компоненты, главное назначение которых – отображение графической информации. Компонент Image загружает и отображает растровые изображения, а компонент Shape украсит ваши формы окружностями, квадратами и т. п.



Рис. П.3. Палитра компонентов Additional

System (системная страница). Страница предоставляет возможность комбинировать отдельные элементы, такие, как списки дисков, каталогов и файлов, для создания нестандартных диалоговых окон, связанных с обработкой файлов (рис. П.4). Страница System также содержит компоненты, обрабатывающие обмен высокого уровня между программами посредством OLE (Object Linking and Embedding). А компонент Timer может генерировать события через определенные, заранее установленные промежутки времени.



Рис. П.4. Палитра компонентов System

Win32 (Windows 32). Страница содержит компоненты, позволяющие созданным с помощью Delphi программам использовать такие нововведения в пользовательском интерфейсе 32-разрядной Windows, как просмотр древовидных структур, просмотр списков, панель состояния и др. (рис. П.5).



Рис. П.5. Палитра компонентов Win32

Dialogs (диалоговая страница). Страница содержит компоненты – стандартные диалоговые окна для операций над файлами, поиска и замены текста, выбора шрифтов, цветов и т.д. (рис. П.6).



Рис. П.6. Палитра компонентов Dialogs

Data Access и Data Controls (сервис баз данных). Delphi использует механизм баз данных компании Borland (Borland Database Engine – BDE) для организации доступа к файлам баз данных различных форматов. Компоненты этих двух страниц облегчают программам Delphi использование сервиса баз данных, предоставляемого BDE, например многопользовательского считывания, записи, индексации и выдачи запросов для таблиц dBASE и Paradox (рис. П.7).

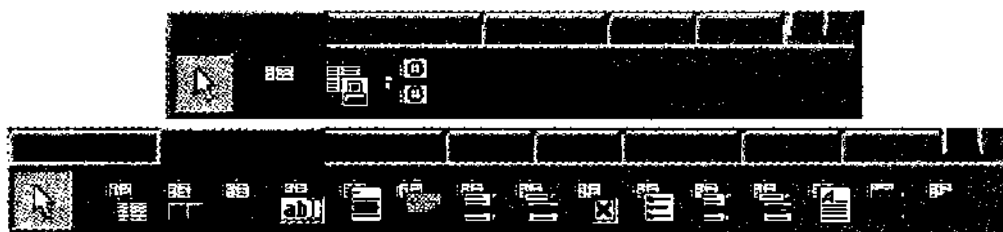


Рис. П.7. Палитры компонентов Data Access и Data Controls

Data Access содержит компоненты для работы с базами данных Microsoft Access. Data Controls содержит специализированные элементы системы управления базы данных, доступные вашим приложениям. С использованием этих

компонентов создание программы просмотра и редактирования базы данных почти не требует программирования.

dbExpress – это новый универсальный механизм доступа к данным, появившийся в Delphi 6 (рис. П.8).

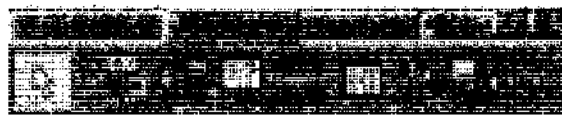


Рис. П.8. Палитра компонентов dbExpress

Для его применения в приложениях Delphi 6 содержит компоненты доступа к данным и драйверы баз данных, реализующие интерфейсы этого универсального механизма доступа к данным с помощью клиентских API соответствующих серверных СУБД. В комплект поставки Delphi 6 входят драйверы dbExpress для InterBase, Oracle, DB2, MySQL. Драйверы dbExpress представляют собой одну динамически загружаемую библиотеку, и обычно только она и требуется при поставке клиентских приложений, использующих dbExpress. Спецификация dbExpress разработана Borland, но является открытой, поэтому при необходимости можно создавать драйверы dbExpress для своих приложений.

Win 3.1. Страница содержит компоненты Delphi 1.0, возможности которых перекрываются аналогичными компонентами Windows 95 (рис. П.9).



Рис. П.9. Палитра компонентов Win3.1

Internet. Страница предоставляет компоненты для разработки приложений, позволяющих создавать HTML-файлы непосредственно из файлов баз данных и других типов, взаимодействующих с другими приложениями для Интернета (рис. П.10).



Рис. П.10. Палитра компонентов Internet

Компоненты этой страницы значительно облегчают создание обработчиков событий для обращения к определенному URL (Uniform Resource Locator – унифицированный локатор ресурса), представления документов в HTML-формате и пересылки их клиент-программе.

Samples (Примеры). Страница содержит компоненты-примеры, которые вы можете добавлять в свои приложения (рис. П.11).



Рис. П.11. Палитра компонентов Samples

Исходный текст к этим типовым компонентам включен в каталог `.\SOURCE\SAMPLES` при инсталляции по умолчанию.

ActiveX. Страница содержит компоненты ActiveX, разработанные независимыми производителями программного обеспечения: сетку, диаграмму, средство проверки правописания (рис. П.12).



Рис. П.12. Палитра компонентов ActiveX

Qreport. Страница предоставляет компоненты для визуального проектирования отчетов баз данных. Здесь содержатся особые версии надписей, полей, примечаний и других элементов управления (рис. П.13).



Рис. П.13. Палитра компонентов QReport

Servers. Страница предоставляет компоненты-наследники `TOleServer` для доступа ко всем серверным объектам Microsoft Office (рис. П.14).



Рис. П.14. Палитра компонентов Servers

Эти компоненты автоматически запускают сервер, когда вызывается один из методов.

Новые компоненты. Библиотека компонентов Visual Component Library (VCL) в Delphi 6 пополнилась новыми компонентами. Из компонентов, предназначенных для создания пользовательских интерфейсов Windows-приложений, следует особо отметить следующие:

- `TLabeledEdit` – поле для ввода текста с меткой;
- `TValueListEditor` – компонент, предназначенный для отображения списка пар типа «параметр – значение», наподобие того, что представлен в Инспекторе объектов;

- TComboBox – комбинированный список, позволяющий отображать рядом с текстом графические изображения;
- TColorBox – комбинированный список для выбора цвета.

Многие из компонентов приобрели дополнительные свойства (например, указывающие на то, как выглядят границы компонентов, такие, как BevelEdges, BevelInner, BevelOuter, BevelKind, BevelWidth).

Еще одно новшество VCL – возможность использования в качестве свойств субкомпонентов, т.е. компонентов, владельцем которых является не форма, а другой компонент. Это означает, что теперь свойства субкомпонентов должны быть доступны в списке свойств компонентов-владельцев.

Появление в одной из предыдущих версий Delphi компонента TActionList существенно облегчило создание пользовательских интерфейсов приложений. В Delphi 6 для работы с объектами TAction добавлены новые компоненты – TActionManager (для хранения коллекции объектов Taction); TToolActionBar и TMainMenuActionBar (интерфейсные элементы, предназначенные для предоставления пользователю доступа к функциональности, описанной в объектах TAction).

Окно Редактор кода

Одним из наиболее важных элементов среды Delphi является окно Редактор кода. Окно Редактор кода в Delphi 6 показано на рис. П.15.

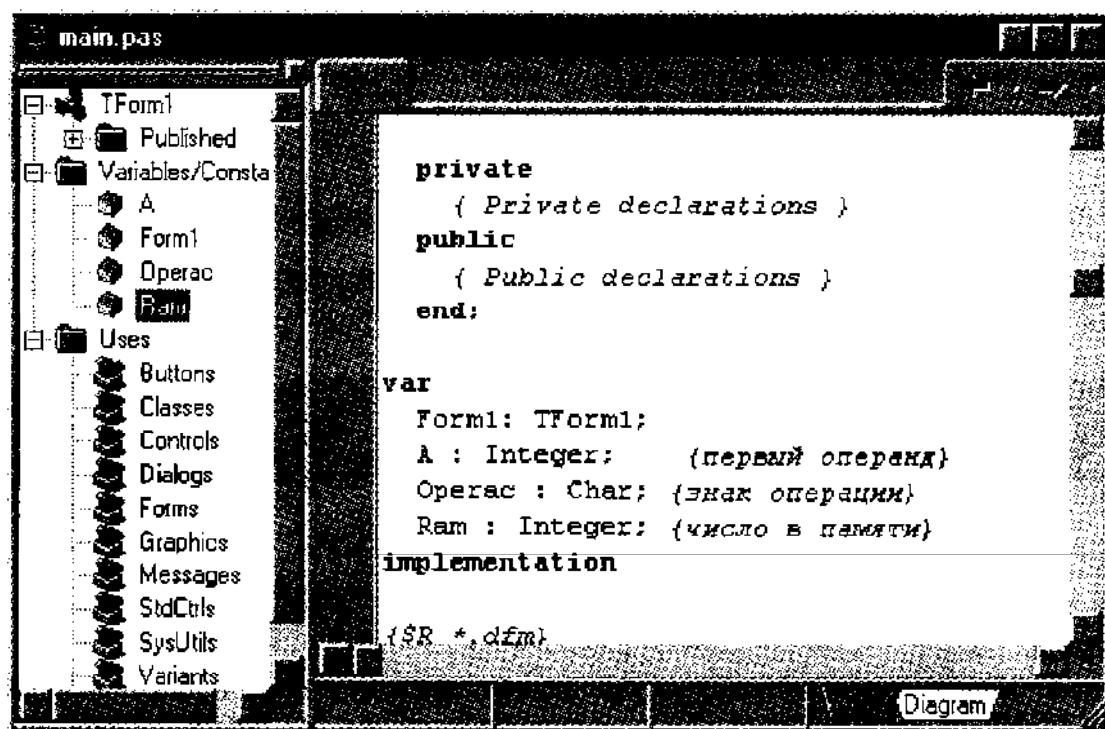


Рис. П.15. Окно Редактор кода с окном Исследователя кода (слева)

В действительности если вы откроете в первый раз это окно в Delphi 6, оно может выглядеть несколько иначе и включать в себя еще одно встроенное окно – окно Исследователя кода (Code Explorer), которое отображает дерево всех типов, классов, реквизитов, методов, глобальных переменных и

глобальных процедур, содержащихся в модуле, открытом в окне Редактор кода. Исследователь кода упрощает исследование кода модуля и автоматизирует создание классов.

Примечание. Во многих случаях вы можете закрыть это дополнительное окно, щелкнув по кнопке в его правом верхнем углу, или задать опции среды проектирования, отменяющие по умолчанию появление окна Code Explorer.

Редактор кода является полноценным программным редактором. Его можно настраивать на различный стиль работы, который вам более привычен. В редакторе применяется выделение цветом синтаксических элементов. Жирным шрифтом выделяются ключевые слова Object Pascal (на рис. П.15 вы видите выделенные слова `private`, `public`, `end`, `var` и др.). Синим курсивом выделяются комментарии (например, на рис. П.15 это комментарии `{Private declarations}` или `{первый операнд}`).

В заголовке окна Редактор кода отображается имя текущего файла, с текстом которого вы работаете. В верхней части окна вы можете видеть также закладки, или ярлычки, указывающие текущую страницу (на рис. П.15 это `main`). Приложения Delphi могут использовать много исходных файлов, и закладки помогают вам переходить от одного из них к другому. Вы можете также открыть дополнительное окно Редактор кода (командой `View ► New Edit Window` или щелчком в окне правой кнопкой мыши и выбором аналогичного раздела из контекстного меню) и одновременно работать с двумя модулями или с двумя разными фрагментами одного модуля.

В нижней части окна Редактор кода вы можете видеть строку состояния. В самой левой ее позиции отображается позиция курсора: номера строки и колонки. Второй элемент строки – индикатор модификации. Когда вы начинаете новый проект, то код, который Delphi автоматически написала для вас, еще не сохранен. Вы должны сделать это сами командой `File ► Save`. Если код изменялся вами или Delphi с того момента, когда вы в последний раз сохраняли его в файле, то в индикаторе модификации (правее индикатора строки и колонки) появляется слово «Modified». Это слово показывает вам, что код, который вы видите, изменен и изменения не сохранены на диске. Третий элемент строки состояния – индикатор режима вставки. Это стандартный индикатор большинства редакторов, который показывает вам, будут ли вводимые символы вставляться в текст (`Insert`) или писаться поверх текста (`Overwrite`). Переключается режим Вставка/Замещение клавишей `Insert`.

В Delphi 6 Редактор кода позволяет выбрать режимы просмотра с помощью ярлычков, размещенных в нижней части редактора кода (список доступных режимов зависит от типа создаваемого приложения). В общем случае, помимо вывода собственно кода приложения, доступны следующие режимы отображения:

Diagram – отображение связей между компонентами доступа к данным (наподобие редактора модулей данных из предыдущей версии Delphi);

HTML Script – отображение кода HTML и JavaScript, сгенерированного компонентами, которые предназначены для создания интерактивных Web-приложений;

HTML Result – отображение HTML-кода, сгенерированного при использовании HTML-шаблона;

Preview – отображение сгенерированного HTML-кода так, как он выглядит в клиентском браузере.

В окно Редактор кода, как и в другие окна Delphi, встроена контекстная справка. Чтобы получить справку о каком-либо слове кода (ключевом слове Object Pascal, написанном имени функции и т. п.), достаточно установить курсор на это слово и нажать клавишу F1.

Окно Инспектор объектов

Инспектор объектов (Object Inspector) обеспечивает простой и удобный интерфейс для изменения свойств объектов Delphi и управления событиями, на которые реагирует объект. Инспектор объектов состоит из двух страниц, каждую из которых можно использовать для определения поведения данного компонента (рис. П.16). Первая страница – это Properties (свойства), вторая – Events (события). Для переключения между страницами свойств и событий используются закладки Properties и Events, расположенные в верхней части Инспектора объектов. Выше страниц Properties и Events имеется выпадающий список всех компонентов, размещенных на форме. В нем вы можете выбрать тот компонент, свойства и события которого вас интересуют. Страница свойств (Properties) Инспектора объектов показывает свойства того объекта, который в данный момент выделен.

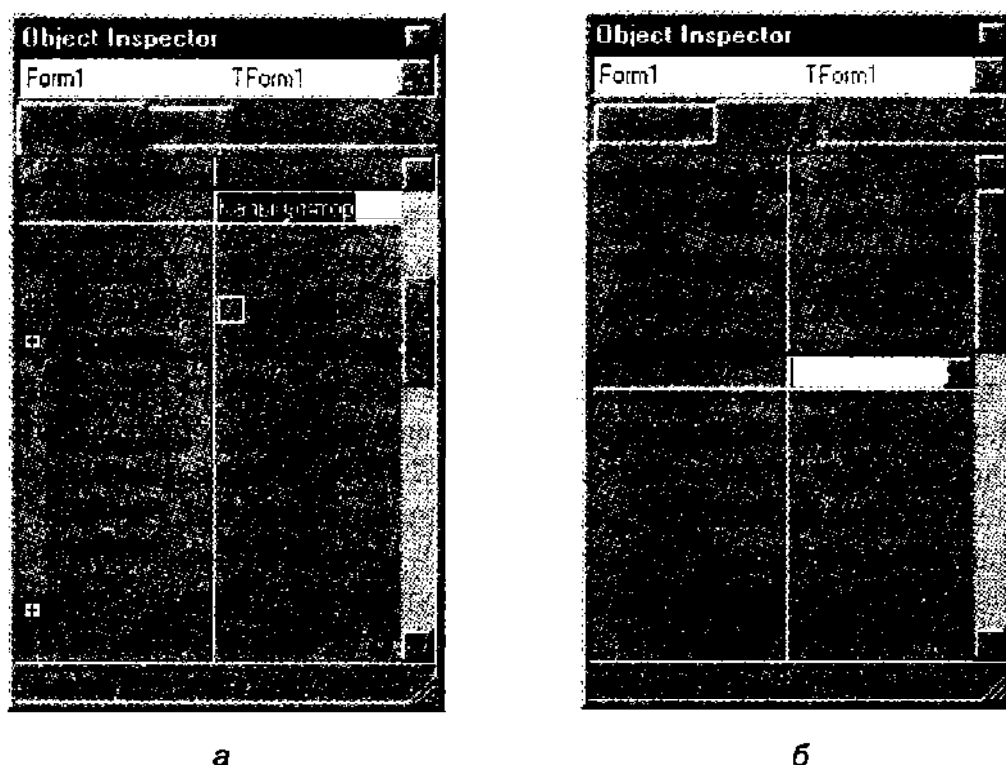


Рис. П.16. Окно Object Inspector:
а – с раскрытой страницей Properties (свойства);
б – с раскрытой страницей Events (события)

Щелкните на окне формы, и на странице свойств Инспектора объектов вы сможете увидеть свойства этой формы (рис. П.16,а), которые можно изменять. Например, измените свойство `Caption` (надпись) формы, написав в ней текст Калькулятор, и вы увидите, что эта надпись появится в полосе заголовка вашей формы.

Если щелкнуть на некоторых свойствах, например на свойстве `Color`, то справа от имени свойства откроется окно выпадающего списка. Нажав в нем кнопку со стрелкой вниз, вы можете увидеть список возможных значений свойства. Список значений можно просмотреть с помощью бегунка. Например, смените значение свойства `Color` с принятого по умолчанию `clBtnFace` (цвет поверхности кнопок) на `clWindow` (цвет окна) – вы увидите, что поверхность формы изменит свой цвет.

Рядом с некоторыми свойствами вы можете видеть знак «плюс» (см., например, свойство `Font` на рис. П.16,а). Это означает, что данное свойство является объектом, который, в свою очередь, имеет ряд свойств. Щелкните на этом знаке или сделайте двойной щелчок на свойстве `Font`. Вы увидите, что откроется таблица таких свойств, как `Color` (цвет), `Height` (высота), `Name` (имя шрифта) и др. Среди них вы увидите свойство `Style` (стиль), около которого тоже имеется «плюс». Щелчок на нем или двойной щелчок на свойстве раскроет дополнительный список подсвойств, в котором вы можете, например, установить в `true` свойство `fsBold` (жирный). Кстати, для смены `True` на `False` и обратно в подобных булевых свойствах не обязательно выбирать значение из выпадающего списка. Достаточно сделать двойной щелчок на значении свойства, и оно изменится. После того как вы просмотрели или изменили подсвойства, можете опять сделать двойной щелчок на головном свойстве или щелчок на знаке «минус» около него, и список подсвойств свернется.

Страница событий (`Events`) составляет вторую часть Инспектора объектов (рис. П.16,б). На ней указаны все события, на которые может реагировать выбранный объект. Страница событий связана с Редактором кода следующим образом: если вы дважды щелкнете мышкой на правой стороне какого-нибудь пункта, то соответствующий данному событию код автоматически запишется в окно Редактор кода, а окно Редактор кода немедленно получит фокус, и вы сразу же получите возможность отредактировать код обработчика данного события. Например, если вам надо выполнить какие-то действия при щелчке левой кнопкой мыши по данному объекту, то вы должны выделить событие `OnClick`. Рядом с именем этого события откроется окно с выпадающим списком. Если вы уже написали в своем приложении какие-то обработчики событий и хотите при событии `OnClick` использовать один из них, то можете выбрать необходимый обработчик из выпадающего списка. Если же вам надо написать новый обработчик, то сделайте двойной щелчок на пустом окне списка. После этого вы попадете в окно Редактор кода, в котором увидите текст

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

Курсор будет расположен в пустой строке между ключевыми словами `begin` и `end`. Увиденный вами код – это заготовка обработчика события, которую автоматически сделала Delphi. Вам остается только написать в промежутке между `begin` и `end` необходимые операторы.

Если вы сделали эти операции, то вернитесь в Инспектор объектов, выделите в нем, например, событие `OnActivate` и нажмите кнопку выпадающего списка. Вы увидите в нем введенный вами ранее обработчик события `FormClick`. Если вам надо использовать тот же самый обработчик и в событии `OnActivate`, просто выберите его из списка. Таким образом, вы можете избежать дублирования в программе одних и тех же фрагментов кода.

Пользуясь Инспектором объектов, вы можете получить контекстную справку о свойствах или событиях. Для этого выделите в окне Инспектор объектов интересующее вас свойство или событие и нажмите клавишу F1, а затем посмотрите в окне Delphi Help справочную информацию.

В Delphi 5 – 6 в Инспектор объектов введена возможность фильтрации свойств и событий и возможность группировать их по категориям. Для того чтобы воспользоваться этими возможностями, щелкните в окне Инспектор объектов правой кнопкой мыши. Во всплывшем меню вы можете выбрать раздел `View`. Вам будет показан ряд категорий свойств и событий, около каждой категории имеется индикатор (рис. П.17).

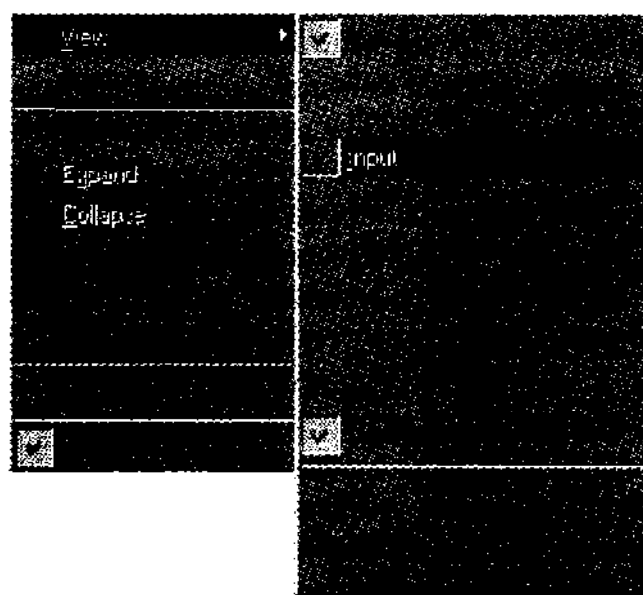


Рис. П.17. Изменение состава видимых свойств и событий

Вы можете включить индикаторы только у некоторых категорий, и тогда в Инспекторе объектов увидите события и свойства только указанных категорий. Выбор раздела `Toggle` переключит видимость разделов: те, которые были видимы, станут невидимыми, и наоборот. Выбор раздела `All` сделает видимыми все свойства и события, а выбор раздела `None` сделает все собы-

тия и свойства невидимыми. Внизу окна Инспектор объектов указывается, сколько свойств или событий в данный момент. В том же меню, всплывающем при щелчке правой кнопкой мыши в окне Редактор кода, вы можете выбрать раздел Arrange и в нем установить одну из двух возможностей: by Name – упорядочить свойства и события в алфавитной последовательности их имен или by Category – упорядочить их по категориям. При упорядочивании по категориям форма представления событий и свойств кардинально меняется. При этом в окне отображаются категории с символами «+», при щелчке на которых раскрывается список элементов, относящихся к данной категории.

Некоторые свойства могут попасть одновременно в несколько категорий, но это не имеет значения: вы можете менять их значения в любой категории, и они синхронно изменятся во всех остальных категориях.

Окно Менеджер проектов

Менеджер проектов Delphi 6 позволяет работать одновременно над несколькими проектами, выбирая активный проект из выпадающего списка (рис. П.18).

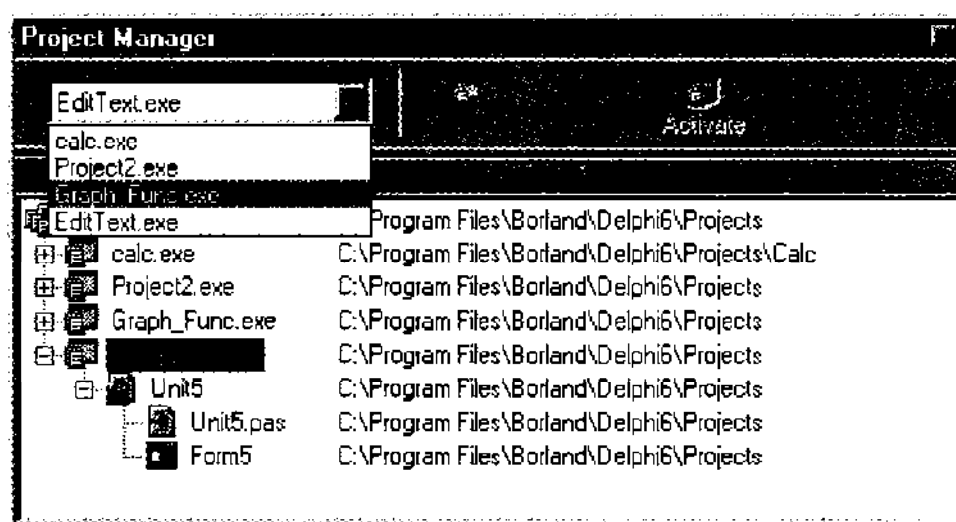


Рис. П.18. Выбор активного проекта в окне Менеджер проектов

Окно Браузер проектов

В Delphi 6 для просмотра объектов используется Браузер проектов (Project Browser). В нем разработчику приложения предоставляется возможность выбрать, какие именно объекты отображать – классы данного приложения или все классы VCL. С помощью двойного щелчка мыши по имени класса можно вывести отдельное окно с детальной информацией о нем (рис. П.19).

Помимо просмотра иерархии классов в нем возможен просмотр глобальных переменных и модулей, а также получение сведений о том, какие еще модули ссылаются на данный модуль.

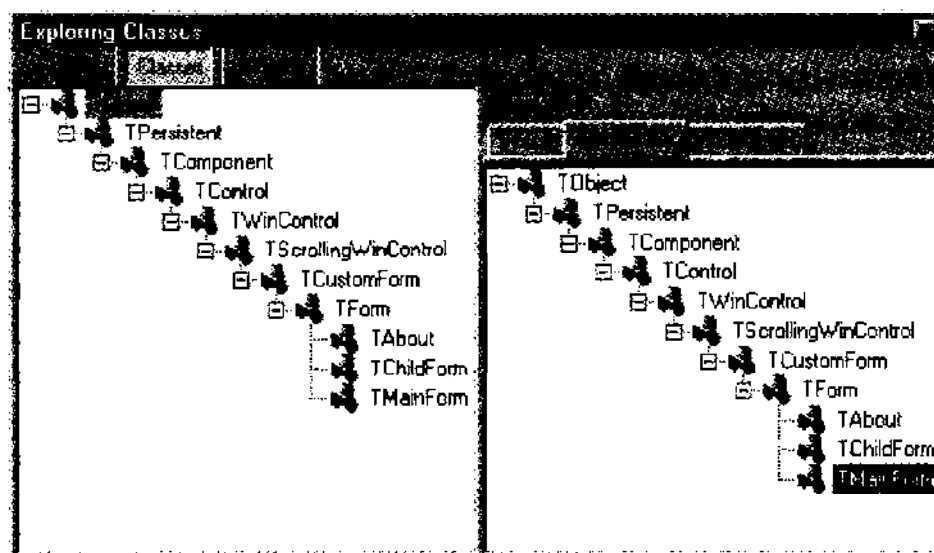


Рис. П.19. Окно Браузер проектов

To-do list: список недоделанных дел

Delphi 6 поддерживает список заданий (to-do list). Задания могут относиться как ко всему проекту в целом, так и к отдельному модулю. Каждое задание может содержать имя исполнителя, приоритет, категорию задания (их может быть сколько угодно), текст задания и отметку, выполнено ли оно. Для добавления записи в список недоделанных дел можно выбрать в контекстном меню редактора кода команду Add To-do Item или нажать комбинацию клавиш Ctrl+Shift+T (рис. П.20).

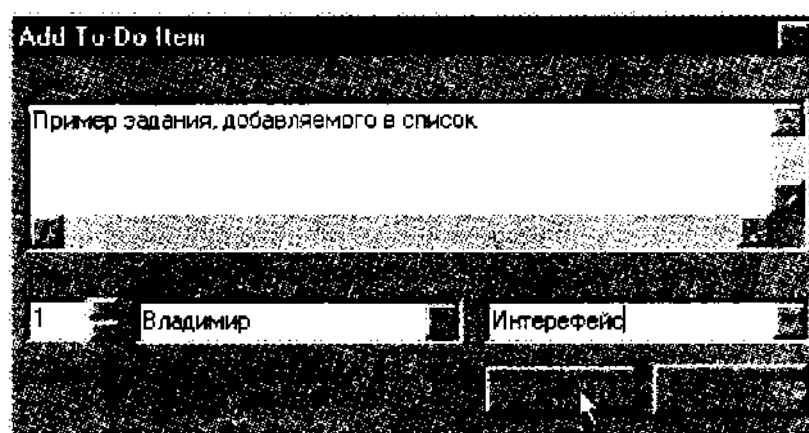


Рис. П.20. Создание нового задания

Список заданий можно просмотреть, выбрав пункт меню View ► To-do list. В этом окне можно добавлять элементы в список, отмечать их как выполненные, редактировать, менять их приоритетный порядок. Список заданий может быть отсортирован по следующим параметрам: имя исполнителя, приоритет, категория задания. Если модуль, к которому относится задание, не загружен в редактор кода, соответствующая строка в списке отображается серым цветом.

Примечание. Список заданий может быть скопирован в буфер обмена в виде текста или HTML-таблицы. Задания к отдельным модулям хранятся в виде комментариев в исходном тексте модуля. Задания для всего проекта хранятся в файле с расширением TODO. В списке задания для всего проекта и для отдельных форм сопровождаются разными пиктограммами.

Перетаскивание и встраивание окон

В ИСП Delphi, начиная с Delphi 4, как и в оконных компонентах Delphi, широко используется технология Drag&Drop – перетаскивание и встраивание окон. Одно из встраиваемых окон – окно Исследователь кода – Code Explorer. По умолчанию оно встроено в окно Редактор кода. Если оно отсутствует на экране, то выполните команду View ► Code Explorer.

Есть также еще много встраиваемых окон, например окно Менеджер проекта (Project Manager), окно наблюдаемых величин (Watch List) и др.

Встраиваемое окно можно отличить от обычного по следующим признакам:

- оно имеет сокращенную полосу системного меню, включающую обычно только кнопку закрытия окна;

- в меню, всплывающем при щелчке в окне правой кнопкой мыши, отмечен переключатель Dockable. Если снять метку с этого переключателя, окно перестанет быть встраиваемым. В дальнейшем вы можете опять пометить этот переключатель, и окно снова станет встраиваемым.

При перетаскивании встраиваемого окна в пределах другого окна размеры его рамки изменяются.

Встраивание окон позволяет вам экономить площадь экрана. Для того чтобы переместить встраиваемое окно, надо потянуть курсором мыши за двойную рамку на одной из его границ (на рис. П.15 она на верхней границе окна). При этом можно вынуть его из окна-контейнера и сделать самостоятельным – так называемым плавающим окном. Для перевода окна в плавающее состояние не обязательно тянуть за двойную рамку – достаточно сделать на ней двойной щелчок. Можно встроить окно Code Explorer в окно Редактор кода иначе, чем это принято по умолчанию, например снизу, чтобы не уменьшать видимую длину строк кода (рис. П.21). Во встроенном состоянии можно курсором мыши передвинуть границы окон, убрав при желании одно из окон, которое в данный момент не нужно.

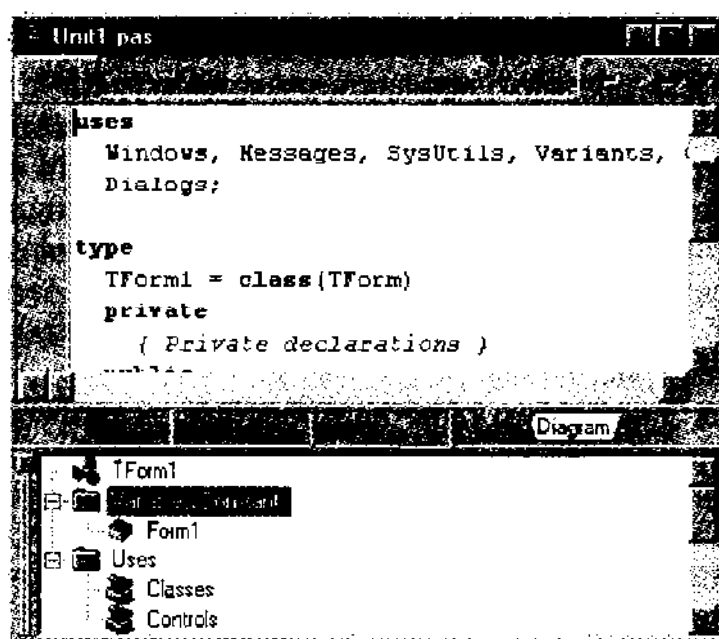


Рис. П.21. Окно Редактор кода с окном Code Explorer, расположенным в нижней части

Очень удобно встраивать окна в Инспектор объектов. Они при этом ложатся на отдельные страницы, совершенно не занимая на экране дополнительного места. А когда требуется, всегда можно посмотреть нужное окно, щелкнув по его закладке. Поскольку не все закладки могут уместиться в заголовке окна, в нем на уровне закладок появляются кнопки со стрелками, направленными влево и вправо. С их помощью можно перейти на страницу, закладка которой не видна.

Технология встраивания окон Drag&Doc реализована в Delphi, начиная с версии 4, в инструментальных панелях. На рис. 2.1 видно, что ИСП содержит 6 панелей, разделенных двойными рамками. Потяните какую-нибудь из панелей (например, палитру компонентов) за эту рамку и увидите, что ее можно перемещать, например, в дополнительный третий ряд панелей, чтобы увеличить доступную длину, или вообще перевести в плавающее состояние.

Среда разработки Delphi 6 поддерживает операции drag-and-drop во время отладки. Например, из окна Редактор кода можно перенести выражение в окно Watch List, после чего это выражение останется в соответствующем списке. Можно перенести выражение в Debug Inspector. Можно также перенести выражение на панель, содержащую дампы памяти в окне CPU, и получить его адрес.

Управление конфигурациями окон

В ИСП Delphi имеется много других окон, облегчающих написание кода и отладку приложения. Каждый пользователь открывает те окна, которые требуются ему для того или иного вида работ, создавая удобную для себя конфигурацию окон. Delphi 6 позволяет сохранять сведения о том, какие окна среды разработки открыты и где именно на экране они расположены. Таких конфигураций может быть создано и сохранено несколько, при этом имеется возможность выделить отдельную конфигурацию, которая будет автоматически загружаться в режиме отладки (так называемый debug desktop) и выгружаться при выходе из него. Для этой цели инструментальная панель содержит кнопки Save current desktop и Set debug desktop, а главное меню – соответствующий подраздел View ► Desktops. Все сохраненные конфигурации среды разработки хранятся в файлах с расширением .dst подкаталога Delphi6\Bin.

Можно сделать так, чтобы при очередном запуске Delphi восстанавливалась конфигурация, которая была на экране в момент завершения предыдущего сеанса работы. Причем может загружаться не только конфигурация окон, но и проект, с которым вы работали в последний раз, так что вы сразу можете продолжать работу над ним. Это делается выполнением команды Tools ► Environment Options (рис. П.22).

В открывшемся диалоговом окне настроек среды разработки на странице Preferences в группе Autosave options надо включить опцию Project desktop (см. рис. П.22). Тогда текущая конфигурация и открытые файлы проекта автоматически сохраняются при завершении сеанса работы с Delphi и автоматически восстанавливаются в начале нового сеанса. При включении этой опции конфигурация окон запоминается также в каждом проекте. Так что если вы впоследствии откроете какой-то проект, с которым работали ранее, то восстановится конфигурация всех окон, которые были открыты в момент окончания предыдущего сеанса работы с данным проектом.

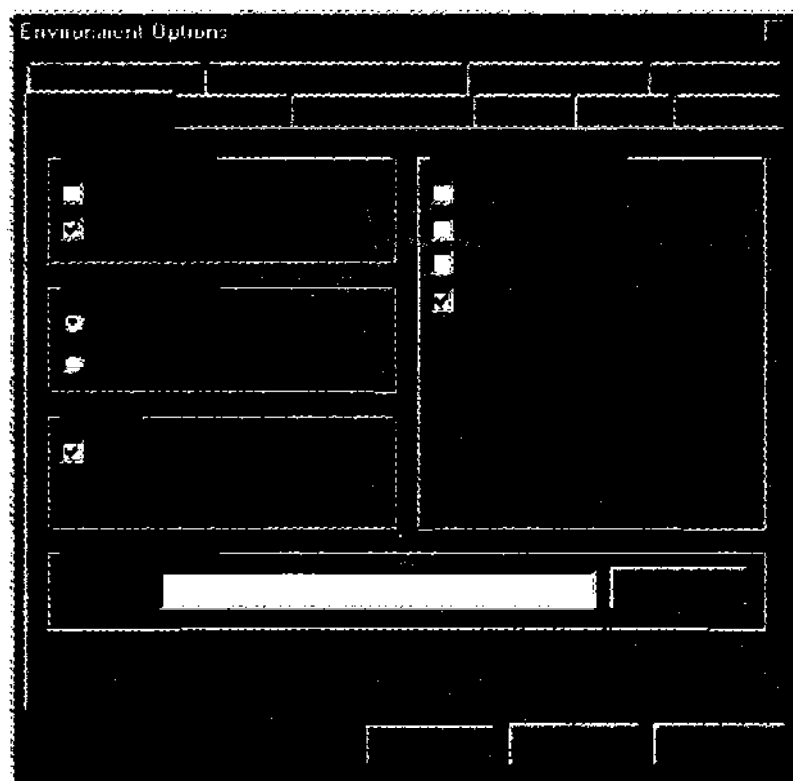


Рис. П.22. Диалоговое окно настроек ИСР

В конфигурациях сохраняются следующие параметры ИСР:

- совокупность открытых окон;
- размеры и расположение окон на экране (но не сохраняется их положение в Z-последовательности, определяющей в случае взаимного перекрытия окон, какие из них находятся поверх других);
- состояние каждого окна (свернутое, развернутое, встроенное в другое окно);
- настройки окна (например, в Инспекторе объектов сохраняется способ отображения информации – по алфавиту или по категориям, а также установки фильтрации, во встраиваемых окнах сохраняется состояние индикатора Docrable).

В Delphi 5 – 6 введены расширенные возможности сохранения конфигураций. Установив на экране некоторую конфигурацию окон, вы можете выполнить команду View ► Save Desktop. Появится диалоговое окно (рис. П.23), в котором вы должны дать имя сохраняемой конфигурации, причем это имя может быть записано по-русски.

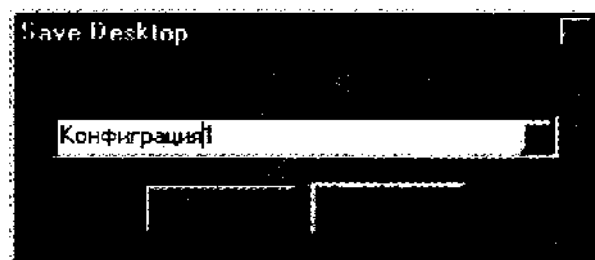


Рис. П.23. Сохранение конфигурации

Эту операцию вы можете повторить несколько раз для разных конфигураций, которые используете в работе. В результате в выпадающем списке панели выбора конфигурации появятся составленные вами конфигурации. В дальнейшем, работая с каким-либо проектом, вы можете в любой момент выбрать в этом списке одну из конфигураций, и она появится на экране.

Введенные вами конфигурации можно впоследствии реорганизовать, удалив ненужные командой View ► Desktops ► Delete. Конфигурация различных вспомогательных окон, установленная для одной из запомненных конфигураций, используется как конфигурация отладки. Какая именно – устанавливается с помощью команды View Desktops ► Set Debug Desktop. Пользователю предлагается список сохраненных конфигураций, из которого он должен выбрать конфигурацию отладки. В этом случае именно эта конфигурация будет автоматически загружаться, как только вы запустите свое приложение на выполнение в режиме отладки. Например, вы создали и сохранили две конфигурации: одну без всяких вспомогательных окон под именем «Default», вторую с окном наблюдения Watches, встроенным в окно Инспектор объектов, под именем «Watches». Если, создавая конфигурацию отладки приложения, вы указали в качестве конфигурации вариант «Watches», то независимо от того, какую из конфигураций – «Default» или «Watches» – вы указали во время работы с проектом, в момент, когда приложение будет запущено на выполнение, установится конфигурация «Watches». После окончания сеанса отладки конфигурация автоматически вернется к исходной.

Таким образом, интегрированная среда разработки Delphi 6 поддерживает современные технологии эффективной разработки программных продуктов самого разнообразного назначения.

РЕКОМЕНДУЕМАЯ ЛИТЕРАТУРА

1. **Архангельский А.Я.** Приемы программирования в Delphi / А.Я. Архангельский. – М.: Бином, 2006. – 944 с.
2. **Бобровский С.И.** Технологии Delphi. Разработка приложений для бизнеса: учеб. курс / С.И. Бобровский. – СПб.: Питер, 2006. – 720 с.
3. **Бушмелева Н.А.** Задачи по программированию / Н.А. Бушмелева, Т.В. Ашихмина, С.М. Окулов. – М.: Бином. Лаборатория знаний, 2006. – 820 с.
4. **Вирт Н.** Язык программирования Паскаль / Н. Вирт // Алгоритмы и организация решения экономических задач. – М.: Статистика, 1974. – Вып.3. – С. 38–66.
5. **Вирт Н.** Алгоритмы + структуры данных = программы / Н. Вирт: пер. с англ. – М.: Мир, 1985. – 406 с.
6. **Вольский С.В.** Turbo Pascal 7. 0 для студентов и школьников: учеб. пособие / С.В. Вольский, П.А. Дмитриев. – СПб.: Наука и техника, 2006. – 224 с.
7. **Йенсен К.** Руководство для пользователя и описание языка Паскаль / К. Йенсен, Н. Вирт. – М.: Финансы и статистика, 1982. – 150 с.
8. **Мельников С.В.** Delphi и Turbo Pascal на занимательных примерах / С.В. Мельников. – СПб.: BHV-СПб, 2006. – 448 с.
9. **Першиков В.И.** Толковый словарь по информатике / В.И. Першиков, В.М. Савинков. – М.: Финансы и статистика, 1995. – 544 с.
10. **Попов В.Б.** Паскаль и Дельфи: учеб. курс / В.Б. Попов. – СПб.: Питер, 2005. – 576 с.
11. **Пройдаков Э.М.** Англо-русский толковый словарь по вычислительной технике, Интернету и программированию / Э.М. Пройдаков. – М.: Русская редакция, 2004. – 864 с.
12. Виртуальный клуб программистов «Королевство Delphi» (<http://www.delphikingdom.com/>) содержит статьи по программированию на Delphi. Форумы и конференции. Подборка ссылок на аналогичные сайты и обзор личных страниц, посвященных программированию на Delphi.
13. Сайт «Мастера Delphi» (<http://www.delphimaster.ru/>) содержит информацию по программированию в Delphi (советы, статьи, подсказки, вопросы-ответы) и обширный русскоязычный каталог Delphi-ресурсов.
14. Сайт Delphi программирование (<http://www.delphisources.ru/>) содержит исходники программ, примеры и решения, а также советы пользователей.
15. Портал программистов Delphi.int.ru (<http://www.delphi.int.ru/>) содержит большой архив статей, текстов программ.
16. Сайт DelphiExpert.Ru (<http://www.delphiexpert.ru/>) содержит статьи, компоненты, исходники программ, уроки, форум.

ОГЛАВЛЕНИЕ

Предисловие	3
Введение	5
Глава 1. ВВЕДЕНИЕ В ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ	9
Основные понятия объектно-ориентированного программирования	9
Операции и методы	15
События и их обработка	18
Создание и уничтожение объектов	18
Использование объектов при визуальном проектировании интерфейса ..	19
Введение в Object Pascal	20
Области видимости и доступ к объектам, переменным и функциям модуля	25
Приложения Windows	31
<i>Контрольные вопросы</i>	38
Глава 2. ИНТЕГРИРОВАННАЯ СРЕДА РАЗРАБОТКИ DELPHI 6 ..	39
Возможности Delphi	39
Общее описание среды	40
Создание, компиляция и отладка простого приложения	45
Изменение свойств визуальных компонентов	56
<i>Контрольные вопросы и задания</i>	74
Глава 3. ПРИЛОЖЕНИЯ ДЛЯ ОБРАБОТКИ СТРОК, МАССИВОВ И ФАЙЛОВ	76
Обработка строк типа String	76
Создание и обработка линейного массива	79
Линейная сортировка массива	84

Использование компонента StringGrid для представления двухмерных массивов	88
Ввод и обработка элементов массива с использованием StringGrid	98
Обработка файлов	101
<i>Контрольные вопросы и задания</i>	114
Глава 4. ПРИЛОЖЕНИЯ С МУЛЬТИМЕДИА	116
Канва и пиксели	116
Мультипликация движения объекта	138
Воспроизведение звука и видеоклипов	145
<i>Контрольные вопросы и задания</i>	156
Глава 5. СОЗДАНИЕ ПРОСТЫХ ПРИЛОЖЕНИЙ	158
Консольное приложение	158
Многооконный текстовый редактор	161
Взаимодействие приложения с внешними программами	177
Создание приложения с использованием функции ShellExecute	178
Приложение для работы с базами данных	181
Создание справочной системы	189
<i>Задания</i>	195
Глава 6. СОЗДАНИЕ ПРИЛОЖЕНИЯ «ШКОЛЬНАЯ БИБЛИОТЕКА»	196
Создание базы данных Microsoft Access	196
Создание приложения выбора режима работы	199
Создание формы с компонентами доступа к базе данных MS ACCESS	203
Создание формы и обработчиков поиска книг в режиме Читатель ...	205
Создание формы и обработчиков обслуживания базы данных	211
Создание формы графического интерфейса библиотекаря при обработке данных о книгах	219
Создание формы графического интерфейса библиотекаря при обработке данных о читателях	225
Создание формы графического интерфейса библиотекаря при добавлении сведений о книге в таблицу базы данных	228
Создание формы графического интерфейса библиотекаря при добавлении сведений о читателе в таблицу базы данных	232
Создание процедур обработки событий при выполнении инвентаризации книг	235

Создание процедур обработки событий при выполнении списания книг	239
Создание процедур обработки событий при поиске читателя, изъятии книг, исключении читателя из базы данных	242
Создание процедур обработки событий при выдаче книг читателю . . .	251
Создание процедур обработки событий при выводе списка читателей . .	258
<i>Задания</i>	263
Словарь терминов	264
Приложение. Справочные сведения о среде программирования	
Delphi 6	292
Система меню	292
Назначение команд меню UCP Delphi 6	293
Палитра компонентов	297
Окно Редактор кода	302
Окно Инспектор объектов	304
Окно Менеджер проектов	307
Окно Браузер проектов	307
To-do list: список недоделанных дел	308
Перетаскивание и встраивание окон	309
Управление конфигурациями окон	310
Рекомендуемая литература	313

Учебно-методическое издание

Попов Владимир Борисович

DELPHI ДЛЯ ШКОЛЬНИКОВ

Заведующая редакцией *Л.А. Табакова*

Редактор *Л.Д. Григорьева*

Младший редактор *О.О. Салтыкова*

Художественный редактор *Г.Г. Семенова*

Технический редактор *Т.С. Маринина*

Корректор *Т.М. Васильева*

Компьютерная верстка *Н.Е.Епифанова*

Оформление художника *А.Л. Чирикова*

ИБ № 5367

Подписано в печать 01.03.2010. Формат 70х100 1/16

Гарнитура «Таймс». Печать офсетная

Усл.п.л. 26,0. Уч.-изд.л. 25,3. Тираж 2000 экз.

Заказ № 1509. «С» 007.

Издательство «Финансы и статистика»

101000, Москва, ул. Покровка, 7

Телефон (495) 625-35-02. Факс (495) 625-09-57

E-mail: mail@finstat.ru [http:// www.finstat.ru](http://www.finstat.ru)

Издательский Дом «ИНФРА-М»

127282, Москва, ул. Полярная, д. 31в

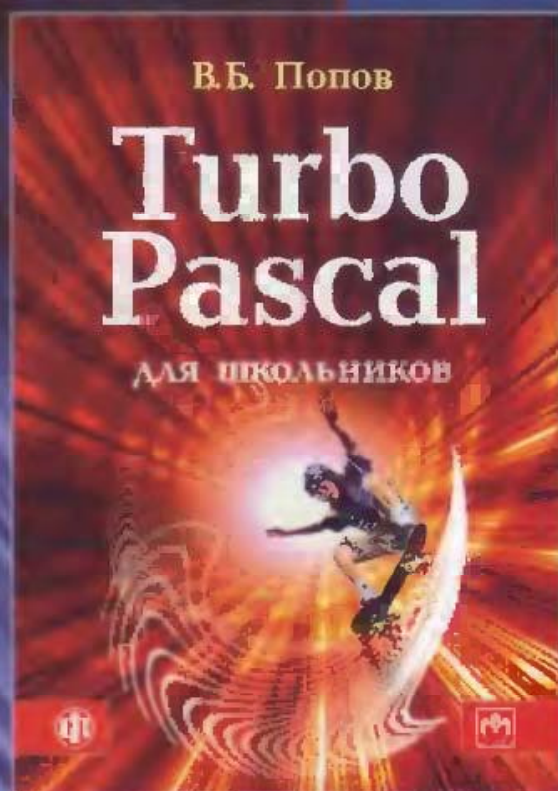
Телефоны: (495) 380-05-40, 380-05-43. Факс (495) 363-92-12

E-mail: books@infra-m.ru [http:// www.infra-m.ru](http://www.infra-m.ru)

Отпечатано в ОАО «Тверской ордена Трудового Красного Знамени
полиграфкомбинат детской литературы им. 50-летия СССР».

170040, г. Тверь, проспект 50 лет Октября, 46.





Издательство
«ФИНАНСЫ И СТАТИСТИКА»

предлагает учебно-методическое пособие

В.Б. Попов
**TURBO PASCAL
ДЛЯ ШКОЛЬНИКОВ**

Пособие представляет собой курс по изучению популярного языка программирования Pascal. В нем последовательно излагаются основные принципы структурного программирования в интегрированной среде программирования Turbo Pascal. Изложение материала ведется на основе примеров рабочих программ. Для проверки усвоения теоретического материала в конце каждой главы имеются контрольные вопросы. Выполнение заданий по разработке приложений поможет сформировать прочные навыки программирования.

Для учащихся общеобразовательных, средних и высших учебных заведений, может быть использовано для самообразования.

**По вопросам приобретения литературы
обращайтесь в Издательство по адресу:**

101000, Москва, ул. Покровка, 7

(метро «Китай-город», выход на ул. Маросейка)

Тел.: (495) 625-35-02, 623-80-42. Факс.: (495) 625-09-57

E-mail: mail@finstat.ru <http://www.flnstat.ru>

При издательстве работает киоск:

понедельник — четверг с 10.00 до 18.45,

пятница — с 10.00 до 17.30

Тел.: (495) 621-86-57

Система «Книга-почтой»

Стоимость пересылки почтовыми бандеролями —
30% от стоимости заказа

ISBN 978-5-279-03470-3

