**Самарский государственный аэрокосмический университет
имени академика С.П. Королева**

Работа в C++Builder

Самара 2005

Учебное издание

РАБОТА В C++BUILDER

Составитель: *Стенгач Михаил Сергеевич*

Редактор Н.С. Куприянова
Лицензия ЛР 020301 от 30.12.96 г.
Подписано в печать __.__.2004 г. Формат 60x84 1/16
Бумага офсетная. Печать офсетная.
Усл. печ.л. 1,62 Усл.кр.-отт. 1,74 Уч.-изд.л. 1,75
Тираж 100 экз. Арт. С-3(Д5)/200_. Зак.

Самарский государственный аэрокосмический университет
имени академика С.П. Королева.
443086 Самара, Московское шоссе, 34.

ИПО СГАУ 443001 Самара, ул. Молодогвардейская, 151.

Министерство образования и науки Российской Федерации
Самарский государственный аэрокосмический университет
имени академика С.П. Королева

Работа в C++Builder

Методические указания

Самара 2005

Составитель: Стенгач М.С.

УДК 681.3.06

Работа в C++Builder: /Самар. гос. аэрокосм. ун-т;
Сост. Стенгач М.С. Самара, 2005, 28 с.

В методических указаниях приводятся начальные сведения о работе в популярной интерактивной среде программирования C++Builder.

Предназначены для выполнения курсовых и лабораторных работ по курсу «Информатика» для студентов всех специальностей.

Составлены на кафедре "Компьютерные системы".

Печатаются по решению редакционно-издательского совета Самарского государственного аэрокосмического университета им. академика С.П. Королева.

Рецензент Павлов О.В.

Начало работы

Интерактивная среда программирования C++Builder позволяет с минимальными затратами времени создавать различные приложения для операционной системы Windows. Поскольку в основе C++Builder лежит концепция быстрого создания приложений (RAD — Rapid Application Development), особое внимание в методических указаниях уделено объектно-ориентированной и визуальной технологиям программирования.

После запуска C++Builder на экране появляются окна, представленные на рис.1 (версия C++Builder 5.0). Это Главное окно, Окно формы, Инспектор объектов и Окно кода.

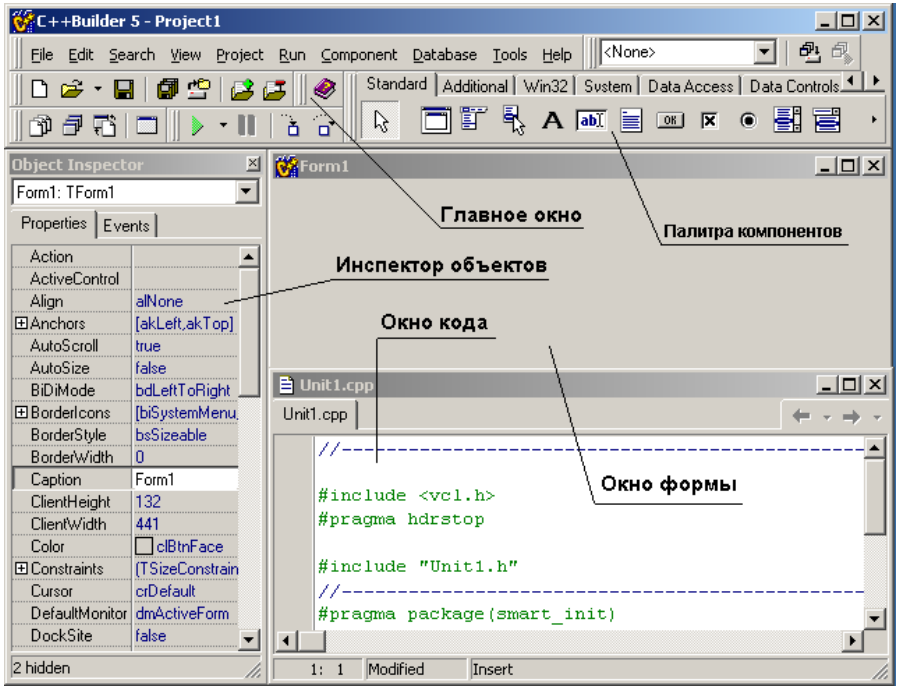


Рис. 1. Окна C++Builder

1.1. Главное окно

Главное окно C++Builder осуществляет основные функции управления проектом создаваемой программы. Здесь располагается главное меню C++Builder, набор пиктографических командных кнопок и палитра компонентов. Все элементы главного окна располагаются на специальных панелях.

.1.1. Пиктографические кнопки

Пиктографические кнопки Главного окна открывают быстрый доступ к наиболее важным опциям главного меню. По функциональному признаку они разделены на 3 группы (приводится набор кнопок для версии C++Builder 5.0):

1. Группа Standard

 - открывает существующий файл.

 - сохраняет файл на диске

 - сохраняет все файлы проекта.

 - открывает созданный ранее проект программы.

 - добавляет новый файл в проект.

 - удаляет файл из проекта

2. Группа View

 - выбирает файл с исходным текстом из списка файлов, связанных с текущим проектом.

 - выбирает форму из списка форм, связанных с текущим проектом.

 - переход из окна формы в окно кода программы (F12).

 - создает новую форму и добавляет ее к объекту.

3. Группа Debug

 - компилирует и выполняет программу (F9).

 - пауза в работе отлаживаемой программы.

 - пошаговая трассировка программы с отслеживанием работы вызываемых подпрограмм (F7).

 - пошаговая трассировка программы, но без отслеживания работы вызываемых подпрограмм (F8).

1.1.2. Палитра компонентов

Палитра компонентов C++Builder занимает правую часть главного окна (см. рис.1) и имеет набор страниц, содержащих различные компоненты. Компоненты – это стандартные заготовки наиболее популярных элементов Windows-приложений.

Рассмотрим кратко содержание некоторых страниц.

Standard. Большинство компонентов на этой странице являются аналогами экранных элементов Windows - меню, кнопки, полосы прокрутки.

Additional. Эта страница содержит более развитые компоненты. Например, компонент SpeedButton является специальной кнопкой для инструментальных панелей. Данная страница также содержит компоненты, главное назначение которых – отображение графической информации. Компонент Image загружает и отображает растровые изображения, а компонент Shape – стандартные фигуры: окружность, квадрат и т.д.

Win32. Эта страница содержит компоненты, позволяющие созданным с помощью C++Builder программам использовать такие нововведения в пользовательском интерфейсе 32-разрядной Windows, как просмотр древовидных структур, просмотр списков, панель состояния, присутствующая в интерфейсе программы Windows Explorer (Проводник), расширенный текстовый редактор и другое.

System. Поскольку не каждая потребность, связанная с обработкой файлов, может быть удовлетворена с помощью стандартных диалоговых окон, страница System предоставляет возможность комбинировать отдельные элементы, такие как списки дисков, каталогов и файлов. Страница System также содержит компоненты, обрабатывающие обмен высокого уровня между программами посредством OLE (Object Linking and Embedding). Компонент Timer может генерировать события через определенные, заранее установленные промежутки времени, а MediaPlayer позволяет программам воспроизводить звук, музыку и видео.

Data Access и **Data Controls.** C++Builder 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 C++Builder использование сервиса баз данных, предоставляемого BDE, например многопользовательского считывания, записи, индексации и выдачи запросов для таблиц dBASE и Paradox.

С использованием этих компонентов создание программы просмотра и редактирования базы данных почти не требует навыков в программировании.

Internet. Эта страница предоставляет компоненты для разработки приложений, позволяющих создавать HTML-файлы непосредственно из файлов баз данных и других типов, взаимодействующих с другими приложениями для Internet. Версия C++Builder 4 дает возможность создавать приложения для Web-сервера в виде DLL-файлов: (Dynamic Link Library — Динамически компонуемая библиотека). С помощью компонентов страницы Internet довольно просто создавать обработчики событий для обращения к определенному URL (Uniform Resource Locator — Унифицированный локатор ресурса), представлению документов в HTML-формате и пересылки их клиент-программе.

QReport. Эта страница предоставляет компоненты баз данных. Здесь содержатся особые версии надписей, полей, примечаний и других элементов управления.

ActiveX. Эта страница содержит компоненты ActiveX, разработанные независимыми производителями программного обеспечения: сетка, диаграмма, средство проверки правописания.

Midas и **Decision Cube.** Здесь собраны компоненты для доступа к удаленным серверам и осуществления SQL – запросов.

1.2. Окно формы

Форма представляет собой фундамент программы. Приложение может иметь несколько форм, каждая из которых выполняет свое особое предназначение. Изначально окно формы пустое. Точнее, оно содержит стандартные для Windows интерфейсные элементы – кнопки максимизации, минимизации и закрытия окна, полосу заголовка и очерчивающую рамку.

Для размещения компонента на форме требуется щелкнуть на нужной закладке палитры компонентов, а затем на кнопке с пиктограммой соответствующего компонента и после этого щелкнуть в окне формы.

1.3. Инспектор объектов

Размещаемые на форме компоненты характеризуются некоторым набором параметров. Для изменения этих параметров предназначено окно Инспектора объектов. Это окно содержит две страницы – `Properties` (Свойства) и `Events` (События). Страница `Properties` служит для установки нужных свойств компонента, страница `Events` позволяет определить реакцию компонента на то или иное событие.

Каждая страница Инспектора состоит из двух колонок. Левая колонка содержит название свойства или события, а правая – конкретное значение свойства или имя подпрограммы, обрабатывающей соответствующее событие. Если слева от названия свойства стоит значок "+", это означает, что данное свойство определяется совокупностью значений. Щелчок по значку "+" приведет к раскрытию списка составляющих данного свойства.

В верхней части окна Инспектора объектов располагается раскрывающийся список всех помещенных на форму компонентов.

1.4. Окно кода

Окно кода предназначено для создания и редактирования текста программы. Первоначально окно кода содержит минимальный исходный текст, обеспечивающий нормальное функционирование пустой формы в качестве полноценного Windows-окна.

Файлу, содержащему исходный текст приложения, присваивается имя `Unit1.cpp`. (см. рис.1, Окно кода). Одновременно создается заголовочный файл с именем `Unit1.h`, который можно увидеть, если щелкнуть правой клавишей мыши по закладке `Unit1.cpp` и выбрать в появившемся контекстном меню `Open Source/Header File`.

В Окне кода в файле Unit1.cpp сразу после создания нового проекта будут следующие строки:

```
//---------------------------------------------------------

#include <vcl.h>
#pragma hdrstop

#include "Unit1.h"
//---------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
//---------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------
```

В самом начале программы располагаются директивы препроцессора: #include – подключение файлов и #pragma – установка параметров компилятора. Назначение остальных операторов будет рассмотрено ниже.

В процессе создания приложения C++Builder автоматически добавляет необходимые строки.

1.5. Размещение компонентов на форме

Размещать компоненты на форме очень просто. Требуется только щелкнуть на нужной вкладке палитры компонентов, затем на кнопке с пиктограммой соответствующего компонента и после этого щелкнуть в окне формы. Можно также щелкнуть на компоненте, а затем нарисовать прямоугольник с помощью мыши на форме – компонент появится внутри этого прямоугольника. Если размеры компонента поддаются изменению, то при появлении на форме он заполнит собой прямоугольник. Если заранее не известно, на какой странице расположен конкретный компонент, выберите пункт Component List (Компоненты) из меню View (Вид), и на экране появится список компонентов в алфавитном порядке.

Если щелкнуть на каком-либо компоненте в палитре компонентов, то его кнопка окажется нажатой. При выборе другого компонента кнопка предыдущего компонента вернется в исходное состояние, так как одновременно не может быть выбрано несколько компонентов. Для того чтобы все кнопки оказались в исходном состоянии и было восстановлено нормальное использование мыши, следует щелкнуть на кнопке со стрелкой выбора, которая появляется с левой стороны каждой страницы палитры (см. рис. 1).

Пример размещения компонента на форме:

1. Выберите страницу Standard палитры компонентов.
2. Поместите указатель мыши на компонент с изображением буквы **A**. После того как мышь побудет в недвижимом состоянии секунду или две, появится поле помощи, идентифицирующее этот компонент как Label.
3. Щелкните на кнопке, затем щелкните на вашей форме. Появится надпись Label1.
4. Обратите внимание, что поле надписи на форме имеет небольшие квадратные маркеры (угловые и боковые), позволяющие изменять размер. Используйте их, чтобы с помощью мыши изменить размер поля надписи (но не размер самой надписи).
5. Для изменения стандартной надписи Label1 в строке Caption окна Инспектора объектов введите надпись Привет!
6. Для изменения цвета и шрифта надписи щелкните мышью по свойству Font окна Инспектора объектов и с помощью кнопки в правой части строки раскройте диалоговое окно настройки шрифта. В списке Размер этого окна выберите высоту шрифта 24 пункта, а с помощью списка Цвет выберите нужный цвет.
7. Теперь щелкните на надписи Привет! и перетащите ее в новое место.

Видно, что при перемещении и изменении размера компоненты выравниваются по точкам координатной сетки формы. Такая возможность помогает поддерживать порядок в формах. Для отмены координатной сетки или изменения плотности точек, выберите пункт Environment Options из меню Tools. Первая страница параметров предназначена для настройки пользователем параметров среды. На этой странице имеется группа Form designer (Конструктор форм), флажки опций Display grid (Отображение сетки) и Snap to grid (Привязка к сетке) которой определяют, видна ли координатная сетка и активна ли она. Можно также изменить значения параметров Grid size X (Шаг по оси X) и Grid size Y (Шаг по оси Y), что приведет к изменению шага координатной сетки.

Практика показывает, что для лучшего управления размещением и размерами компонентов значение шага координатной сетки должно быть равным 4 вместо 8, отключено изображение координатной сетки и включен параметр Snap to grid.

Не каждый компонент виден на форме во время запуска программы. Например, размещение на форме компонента MainMenu приводит к появлению в разрабатываемом приложении меню, но соответствующая пиктограмма во время запуска программы не отображается. Компоненты, представляющие диалоговые окна общего назначения, вообще никак не визуализируются во время работы программы. Размеры невидимого компонента в процессе разработки не изменяются - он всегда отображается в виде пиктограммы.

2. Объектно-ориентированное программирование

Двумя фундаментальными концепциями всех объектно-ориентированных языков программирования (C++, Delphi) являются *объекты* и *классы*.

Объект – фрагмент программы, предназначенный для решения отдельной задачи. Объекты обладают бóльшими возможностями, чем подпрограммы, поскольку в них могут содержаться как переменные, так и функции.

Таким образом, объектно-ориентированное программирование – это такая методология реализации, при которой программа организуется как совокупность сотрудничающих объектов, каждый из которых является экземпляром какого-либо класса.

Объектно-ориентированный язык C++, который является основным инструментом C++Builder, разработан на основе популярного языка программирования C. По этой причине все основные элементы этих двух языков практически совпадают. Это относится к операторам языка, типам данных, выражениям, операциям, функциям. Поскольку языку C посвящено большое количество изданных ранее на кафедре пособий и методических указаний, мы в этой главе подробно разберем те существенные дополнения, которые были внесены разработчиками в C++Builder.

2.1. Классы

Классы предназначены для создания объектов.

Структура описания класса:

```
class <имя класса>
{ <описание переменных и функций> };
<имя класса> <имя объекта>;
```

Например, определим класс с именем `Tmycl` и объект этого класса `myobj`:

```
class Tmycl
{
   int a;
   float b;
};
Tmycl myobj;
```

Видно, что описание класса напоминает описание типа `struct`. Обращение к переменным и функциям класса будет аналогично обращению к переменным типа `struct`:

```
main()
{
   myobj.a=1;
   myobj.b=3.2;
}
```

Следует помнить, что все рабочие данные хранятся в объекте. Класс не содержит никаких данных – он лишь описывает общую структуру объекта.

По используемому в C++Builder соглашению все имена классов начинаются с буквы T.

2.1.1. Составляющие класса

Поля. Полями называются инкапсулированные в классе данные. Поля могут быть любого типа, в том числе и классами, например:

```
class Tmycl {
      int a;
      class TSomeClass {int b;};
      };
```

Методы. Принадлежащие классу функции называются методами. Определение метода:

```
<имя класса>::< метод >
```

Доступ к методам класса, как и к его полям, осуществляется с помощью составных имен:

```
<имя объекта>.< метод>;
```

Пример:

```
class Tmycl
      {void a();}; // описание метода a() в классе Tmycl
/* определение метода a() */
void Tmycl::a(){printf("Method A\n");}
void main()
{
Tmycl myobj;
myobj.a();   // вызов метода a()
}
```

В C++ имеются два специальных метода – конструктор и деструктор. Конструктор распределяет объект в динамической памяти, деструктор удаляет его. Имена конструктора и деструктора совпадает с именем класса, но имя деструктора снабжено префиксом тильда ~.

2.1.2. Объявления класса

Класс может содержать разделы: `private`, `public`, `protected` и `__published`. Внутри разделов сначала определяются поля, а затем – методы.

Разделы определяют области видимости элементов. Элементы раздела `private` доступны только внутри методов данного класса. Это так называемый закрытый доступ. Элементы раздела `public` доступны любым функциям программы. Элементы раздела `protected` доступны только методам самого класса, а также методам потомков этого класса. Элементы раздела `__published` эквивалентны элементам `public` за исключением того, что они доступны также в режиме проектирования в Инспекторе объектов.

По умолчанию элементы класса имеют доступ `private`.

2.1.3. Основные понятия класса

В основе классов лежат три фундаментальных принципа, которые называются *инкапсуляция, наследование* и *полиморфизм*.

Инкапсуляция. Класс представляет собой единство полей (переменных) и методов (функций). Объединение этих сущностей в единое целое

и называется инкапсуляция. Инкапсуляция позволяет сделать класс самодостаточным для решения конкретной задачи. Например, класс TForm содержит в себе (инкапсулирует в себе) все необходимое для создания Windows-окна. Класс TMemo представляет собой полноценный текстовый редактор, а класс TTimer обеспечивает работу программы с таймером.

Наследование. Любой класс может быть порожден от другого класса. Для этого при его объявлении через двоеточие указывается имя класса-родителя:

```
class  TParentClass{
public: int a();
        void b();
};
class TChildClass : public TParentClass {};
```

Порожденный класс автоматически наследует поля и методы своего родителя и может дополнять их новыми.

Все классы в C++Builder порождены от единственного родителя – класса TObject. Этот класс не имеет полей, но включает в себя методы самого общего назначения – от создания до уничтожения объектов.

Все новые классы будут дочерними классу TObject.

Принцип наследования привел к созданию в C++Builder дерева классов, где каждый потомок дополняет возможности своего родителя новыми.

Полиморфизм – это свойство классов решать сходные по смыслу задачи разными способами. Можно изменить (*перекрыть*) метод родителя в потомке. Для этого необходимо объявить в потомке одноименный метод и реализовать в нём нужные свойства. В результате в родителе и потомке будут действовать два одноименных метода, но с разными свойствами.

2.1.4. Класс TForm и реакция на события

Вернемся к нашей первой программе в C++Builder (см. пункт 1.5). В Окне кода щелкните правой клавишей мыши по закладке Unit1.cpp и выберите в появившемся контекстном меню Open Source/Header File. В Окне кода появится закладка Unit1.h, где мы увидим описание класса TForm1:

```
class TForm1 : public TForm
{
…
};
```

В состав C++Builder входит несколько сотен стандартных классов, которые описывают стандартные объекты. При разработке приложений вновь создаваемые объекты получают имя класса-родителя с добавленным числовым индексом. Таким образом, имя TForm1 означает имя класса, который *порожден от* (создан по образцу) стандартного класса TForm.

На закладке Unit1.cpp мы можем найти следующую строку:

```
TForm1 *Form1;
```

которая означает, что создан *объект* класса TForm1 с именем Form1.

Стандартный класс TForm описывает пустое Windows-окно, в то время как TForm1 описывает окно с добавленным компонентом *метка*.

В файле Unit1.cpp имеется также описание метода TForm1, принадлежащего классу TForm1:

```
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
}
```

– это конструктор класса TForm1. По существующему в C++Builder соглашению все методы классов формы объявляются со спецификацией **__fastcall**.

Проведем модернизацию нашей программы. Поместим на форму еще один компонент – кнопку – и заставим ее откликаться на событие, связанное с нажатием левой клавиши мыши. Компонент "кнопка" изображается пиктограммой на странице Standard палитры компонентов как кнопка с надписью OK. Щелкните по этой кнопке, а затем щелкните на форме. Появится кнопка с надписью Button1.

При щелчке по кнопке мышью в работающей программе возникает событие OnClick (по щелчку). Пока это событие никак не обрабатывается программой, и поэтому "нажатие" кнопки не приведет ни к каким последствиям. Чтобы заставить программу реагировать на нажатие кнопки, необходимо написать на языке C++ фрагмент программы, который называется *обработчиком события*. Этот фрагмент программы оформляется в виде функции.

Чтобы C++Builder создал заготовку для функции обработчика события OnClick, необходимо выделить вновь вставленный компонент (кнопку), перейти на страницу Events (События) в окне Инспектора объектов и дважды щелкнуть мышью в окне левее надписи OnClick. В ответ C++Builder активизирует Окно кода со следующий фрагментом программы:

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
}
```

Попробуем разобраться, что он содержит. Текст **void __fastcall** извещает компилятор о начале определения функции. За ним следует имя функции TForm1::Button1Click. Это имя составное – оно состоит из имени класса TForm1 и собственно имени функции Button1Click.

За именем функции TForm1::Button1Click в круглых скобках следует описание вызова TObject *Sender (параметр с именем Sender принадлежит классу TObject). Параметр Sender вставлен C++Builder "на всякий случай": с его помощью функция Button1Click может, при желании, определить, какой именно компонент создал событие OnClick.

Само тело функции TForm1::Button1Click (т.е. строки между символами {…}) пока еще не содержит описания каких-либо действий, т.к. C++Builder лишь создал заготовку для функции. Наполнить тело нужными операторами – задача программиста.

Каждый раз при нажатии кнопки Button1 управление будет передаваться в тело функции, а значит, между { и } мы можем написать фрагмент программы, который будет выполняться в ответ на это событие.

Чтобы убедиться в этом напишите в пустой строке между { и } следующее предложение:

```
Label1->Caption="Hello world!";;
```

и сделайте прогон программы (клавиша F9). Теперь при нажатии кнопки Button1 надпись метки Label1 заменится на Hello world!

Заметьте, что доступ к элементу класса осуществляется с помощью стрелки ->, т.к. при передаче данных в функцию (в нашем случае – в функцию TForm1::Button1Click) мы имеем дело с указателями на переменные.

2.2. Структура программы в C++Builder

Известно, что программирование на языке С основано на понятии функции и любая программа должна содержать, как минимум, одну главную функцию. При разработке нашего первого приложения в C++Builder мы имели дело с файлами Unit1.cpp и Unit1.h. Первоначально они содержат описание класса, объекта и метода-конструктора, однако описание главной функции в них отсутствует.

Приложение C++Builder содержит, кроме исходных и заголовочных файлов, ещё и так называемый файла проекта. Файл проекта создается автоматически и имеет по умолчанию имя Project1.cpp. Чтобы его увидеть необходимо в главном меню выбрать Project/View Source. В окне кода программы появится закладка Project1.cpp, где можно увидеть главную функцию Windows-приложения, имеющую имя WinMain:

```
WINAPI WinMain(HINSTANCE, HINSTANCE, LPSTR, int)
{
  try
  {
     Application->Initialize();
     Application->CreateForm(__classid(TForm1), &Form1);
     Application->Run();
  }
  catch (Exception &exception)
  {
     Application->ShowException(&exception);
  }
  return 0;
}
```

В теле главной функции происходит обращение к методам объекта Application. В этом объекте собраны данные и подпрограммы, необходимые для функционирования Windows-приложения. Метод CreateForm создает и показывает на экране окно главной формы, а метод Run реализует бесконечный цикл получения и обработки поступающих от Windows сообщений о действиях пользователя.

3. Компоненты C++Builder

В этой главе мы рассмотрим наиболее популярные компоненты C++Builder.

3.1. Компоненты страницы Standard

MainMenu – главное меню. После размещения компонента MainMenu на форме необходимо задать его пункты. Для этого следует дважды щелкнуть по компоненту левой кнопкой мыши. На экране покажется окно конструктора меню. Создание пунктов не вызывает проблем. В Окне инспектора объектов в строке Caption вводится текст пункта. После нажатия клавиши Enter можно переходить к следующему пункту. Для создания у пункта раскрывающегося меню необходимо щелкнуть мышью по строке ниже пункта и ввести первый пункт подменю. Если в свойстве Caption задать символ "-", то в списке подменю появится разделительная черта.

PopupMenu – локальное меню, появляющееся после нажатия правой кнопки мыши. Может быть создано для любого компонента, для этого в свойстве PopupMenu этого компонента необходимо поместить имя компонента-меню.

Label – метка для отображения текста. В разделе 1.5. мы уже подробно рассмотрели размещение этого компонента на форме.

Edit – ввод и отображение строки. Компонент класса TEdit представляет собой однострочный редактор. Главным свойством компонента является свойство Text, содержащее строку, которая вводится или выводится в компоненте. По умолчанию строка содержит слово Edit и порядковый номер компонента (например, Edit1).

Строка свойства Text имеет тип AnsiString. Базой для создания класса AnsiString послужил тип String из языка Паскаль, который был расширен в соответствии с возможностями языка С++. Особенностью AnsiString является то, что два экземпляра этого класса могут физически занимать один и тот же участок памяти.

Так как строка имеет тип AnsiString, для ввода с помощью компонента чисел необходимо воспользоваться следующими стандартными функциями:

StrToFloat(s) – преобразует строку s в вещественное число,

StrToInt(s) – преобразует строку s в целое число.

Например:

```
float a;
a=StrToFloat(Edit1->Text);
```

Чтобы отобразить в компоненте число, необходимо проделать обратную операцию:

```
Edit1->Text=FloatToStr(a);
```

Заметим, что для отображения в компоненте чисел преобразование можно и не производить – C++Builder сделает это автоматически:

```
Edit1->Text=a;
```

Memo – ввод и отображение текста. Компонент предназначен для ввода и редактирования длинного текста. Текст хранится в свойстве `Lines` и представляет собой пронумерованный набор строк, имеющих тип `AnsiString`. Нумерация строк начинается с нуля.

При работе с компонентом Memo наиболее часто употребляются следующие операции:

1. `Memo1->Lines->Strings[i];` – доступ к i-ой строке, например:

`Memo1->Lines->Strings[0]="2";` – в первой строке вывели цифру 2,

`Memo1->Lines->Strings[0]=2;` – в первой строке вывели цифру 2 (число преобразовалось в текст автоматически),

`Memo1->Lines->Strings[0]="два";` – в первой строке вывели слово «два».

2. `Memo1->Lines->Count;` – общее количество строк;

3. `Memo1->Lines->Add(s);` – добавить новую строку `s` в список;

4. `Memo1->Clear();` - очистить содержимое компонента.

Компонент имеет также свойство `Text`, которое, в отличие от `Lines`, содержит текст в виде длинной строки.

Button – кнопка. В разделе 2.1.4. уже был рассмотрен пример размещения кнопки на форме, а также создания отклика на событие, связанное с щелчком на кнопке левой клавишей мыши. Чтобы событие `OnClick` возникало также и при нажатии клавиши Enter, необходимо свойство `Default` задать `True`.

В отличие от большинства других компонентов кнопка является компонентом самой Windows и поэтому не может изменять свой цвет произвольным образом – она его меняет вместе с изменением палитры самой Windows.

 **CheckBox** – независимый переключатель или флаг выбора. В составе диалогового окна их может быть несколько и состояние любого из них не зависит от состояния остальных. Логическое свойство `Checked` определяет наличие флага выбора:

`if (CheckBox1->Checked) …` – если флаг установлен,

`if (!CheckBox1->Checked) …` – если флаг не установлен.

Можно использовать свойство `State`, которое может принимать 3 значения:

`cbChecked` - флаг установлен,

`cbUnchecked:` - флаг не установлен,

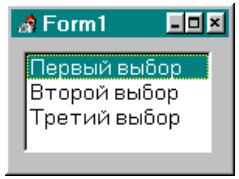
`cbGrayed` - в окошке компонента устанавливается флаг выбора, но само окошко окрашивается серым цветом.

Пример:

```
switch (CheckBox1->State)
{
case cbChecked:   …
case cbUnchecked: …
case cbGrayed:    …
}
```

 **RadioButton** – зависимые переключатели. На форме имеет смысл размещать минимум два компонента `RadioButton`. Если в одном из компонентов свойство `Checked` принимает значение `True`, то у других компонентов оно автоматически принимает значение `False`.

ListBox – список выбора.



В списке предусмотрена возможность программной прорисовки элементов, поэтому список может содержать не только строки, но и произвольные изображения. Для ввода пунктов меню необходимо выбрать свойство `Items`. После того, как меню сформировано, можно создать реакцию на событие OnClick:

```
void __fastcall TForm1::ListBox1Click(TObject *Sender)
```

Количество строк в списке определяется с помощью свойства `Count`, имеющего целый тип, например:

```
M=ListBox1->Items->Count;
```

В данном примере `M` будет равно 3.

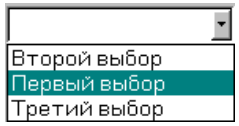
Номер выбранного пункта определяется свойством `ItemIndex`:

```
I=ListBox1->ItemIndex;
```

Номер первой строки равен 0, второй 1 и т.д. В случае выбора второго пункта списка возможна такая конструкция:

```
if (ListBox1->ItemIndex==1) … ;
```

ComboBox – раскрывающийся список выбора. Пункты меню также задаются в свойстве `Items`.



В свойстве `Style` можно задать следующие модификации компонента:

`CsDropDown` (по умолчанию) – раскрывается после нажатия кнопки;

`CsSimple` – список всегда раскрыт.

ScrollBar – управление значением величины. 

Свойства компонента:

`Max` - максимальное значение диапазона изменения числовой величины;

`Min` - минимальное значение диапазона изменения числовой величины;

`Position` - текущее значение числовой величины.

GroupBox и **Panel** – панели, предназначены для размещения в них групп элементов.

RadioGroup – контейнер зависимых переключателей. Чтобы создать в контейнере переключатели, следует использовать свойство `Items`. Свойство `ItemIndex` по умолчанию имеет значение `–1` (не выбран ни один переключатель). Если значение `ItemIndex` равно `0`, то выбран первый зависимый переключатель, если равно `1`, то второй и т. д. В случае выбора второго зависимого переключателя возможна такая конструкция:

```
if (RadioGroup1->ItemIndex==1) …
```

3.2. Компоненты страницы Additional

BitBtn – кнопка с изображением – разновидность кнопки `Button`. Её отличительная особенность – свойство `Glyph`, с помощью которого определяется изображение, рисуемое на поверхности кнопки. Свойство `Kind` определяет одну из 11 стандартных разновидностей кнопки. Например:

- свойство `Kind` равно `bkOk`;

- свойство `Kind` равно `bkClose`.

Нажатие любой из этих 11 кнопок (кроме `bkCustom` и `bkHelp`) возвращает в программу результат `mrИмякнопки`, например: `bkOk - mrOk`, `bkCancel - mrCancel`. У кнопок `bkOk` и `bkYes` свойство `Default` автоматически устанавливается `True`. Кнопка `bkClose` закрывает главное окно программы. Кнопка `bkHelp` автоматически вызывает раздел справочной службы, связанный с `HelpContext` формы, на которую она помещена.

 **SpeedButton** – кнопка для инструментальных панелей.

Особенности этих кнопок:

1. могут фиксироваться в утопленном состоянии;
2. не могут закрывать модальное окно;
3. не могут быть умалчиваемыми (отсутствует свойство `Default`).

Для фиксации кнопки в утопленном состоянии свойство `GroupIndex` должно быть не равным 0. Если у нескольких кнопок свойство `GroupIndex` имеет одно и то же значение, то эти кнопки принадлежат одной группе.

Логическое свойство `AllowAllUp` определяет поведение кнопки:

– если `AllowAllUp==false` - утопленная кнопка отпускается только при нажатии другой кнопки, входящей в ту же группу,

– если `AllowAllUp==true` - кнопку можно освободить повторным щелчком.

MaskEdit – специальный редактор ввода текста по шаблону, задаваемому свойством `EditMask`.

StringGrid – таблица строк, предназначена для создания таблиц. Таблица делится на две части – фиксированную (служит для показа заголовков столбцов и рядов) и рабочую.

Главное свойство компонента: `Cells` – двухмерный массив ячеек, где первый номер соответствует столбцу, а второй – ряду. Например:

```
StringGrid1->Cells[1][1]="Левая верхняя ячейка";
```

Свойства компонента:

`ColCount` – общее количество столбцов в таблице;

`RowCount` – общее количество строк в таблице;

`FixedCols` –количество фиксированных столбцов;

`FixedRows` –количество фиксированных строк;

`Options->goEditing` – если задано `true`, то можно редактировать текст в таблице;

DrawGrid – произвольная таблица, является родителем таблицы строк `StringGrid`.

Image – отображение картинок. Служит для размещения на форме изображений. Свойство `Picture` позволяет отыскать файл с нужным изображением и поместить это изображение на форме.

Shape – прорисовка стандартных фигур. В свойстве `Shape` имеется следующий набор стандартных фигур:

`stSquare` – прямоугольник;

`stCircle` – круг;

`stEllipse` – эллипс;

`stRoundRect` – скругленный прямоугольник;

`stRoundSquare` – скругленный квадрат.

Свойства `Brush` и `Pen` позволяют менять цветовое оформление компонента.

Bevel – декоративная рамка. Свойство `Style` определяет стиль рамки – вдавленная или выпуклая, а свойство `Shape` определяет её внешний вид.

ScrollBox – панель с прокруткой. Служит контейнером для размещения других компонентов. Использование компонента не отличается сложностью: положите его на форму и разместите затем на нем другие компоненты. Если очередной компонент выйдет за пределы рабочей области, по сторонам контейнера возникнут полосы прокрутки.

3.3. Компоненты страницы Win32

TabControl – набор закладок. Представляет собой контейнер с закладками.

Основные свойства компонента:

`Tabs` – определяет название и количество закладок;

`TabIndex` – определяет индекс выбранной закладки. Индексация начинается с 0.

При выборе новой закладки возникает событие `OnChange`.

PageControl – набор страниц с закладками (рис. 2). Изначально содержит одну страницу. Чтобы добавить новую страницу необходимо щелкнуть по компоненте правой кнопкой мыши и выбрать `New Page`.

Рис. 2. Демонстрация компонента `PageControl`

Каждая страница компонента `PageControl` будет представлять отдельный компонент со своими свойствами. Свойство `ActivePage` компонента `PageControl` содержит ссылку на активную страницу.

Например, нужно, чтобы при выборе первой закладки вторая исчезала:

```
if (PageControl1->ActivePage==TabSheet1) TabSheet2->TabVisible=false;
```

ImageList – контейнер для хранения множества рисунков одинакового размера. Двойным щелчком мыши по компоненту раскрывается редактор `ImageList Editor`. С помощью кнопки `Add` можно поместить в контейнер новое изображение. Пример использования контейнера см. ниже у компонента `ListView`.

TreeView – дерево иерархии. Служит для показа ветвящихся иерархических структур, таких как, например, файловая структура диска. Для наполнения списка необходимо раскрыть свойство `Items` – на экране появится окно редактора компонента. Щелкните по кнопке `New Item` и введите связанный с узлом текст в поле `Text`. Для ввода подузла сначала нужно щелкнуть в окошке `Items` по узлу, который должен стать родительским, а затем – на кнопке `New SubItem`.

ListView – список просмотра. Предназначен для показа и выбора нескольких элементов. Допускается не более одного уровня вложенности.

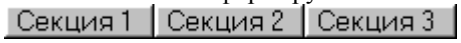
Пример создания списка просмотра:

1. На пустой форме расположите компоненты `ListView` и `ImageList`.
2. Для наполнения контейнера `ImageList1` дважды щелкните по нему мышью.
3. В появившемся окне нажмите кнопку `Add` и выберите файл `Program Files\Borland\CBuilder5\Examples\Icons\database`.
4. Повторите п.3 для выбора нескольких пиктограмм.
5. Поместите в свойствах `SmallImages` и `LargeImages` компонента `ListView1` значение `ImageList1`
6. Щелкните по кнопке в свойстве `Items` компонента `ListView1` – на экране появится окно редактора компонента.
7. Щелкните по кнопке `New Item` и введите в полях `Caption` и `Image Index` значение 0 (первая пиктограмма из `ImageList1`).
8. Повторите п.7 и задайте у последующих элементов списка значений `Caption` и `ImageIndex` равными 1, 2, 3 и т.д.
9. Выберите значение свойства `ViewStyle` компонента `ListView1` равным `vsList`.

Результат работы программы представлен на рис. 3.

Рис. 3. Демонстрация компонента `ListView`

HeaderControl – заголовок с регулируемыми размерами ширины колонок. Колонки формируются в свойстве `Sections`.



– пример заголовка из трех колонок.

При перемещении границ колонок возникает событие `OnResize`, возвращающее следующие значения:

`Sections->Items[i]->Left` – левая граница i-ой колонки,

`Sections->Items[i]->Width` – ширина i-ой колонки,

`HeaderControl1->Height` – высота заголовка.

TRichEdit – редактор RTF-текста. Текст формата RTF хранит информацию, управляющую свойствами каждого абзаца и сменой шрифта по ходу текста. Для хранения атрибутов шрифта используются объекты класса `TTextAttributes`. Эти атрибуты распространяются на весь текст через свойство редактора `DefAttributes` или на выделенную часть текста – через свойство `SelAttributes`. Атрибуты абзаца хранятся в объектах класса `TParaAttributes`.

Например, необходимо, чтобы текст в редакторе вводился наклонный и синего цвета. Данную задачу реализует следующая функция:

```
void __fastcall TForm1::RichEdit1Change(TObject *Sender)
{
 RichEdit1->SelAttributes->Color = clBlue;
 RichEdit1->SelAttributes->Style =
 RichEdit1->SelAttributes->Style << fsItalic;
}
```

StatusBar – панель состояния. Компонент может иметь несколько секций, а также кнопку изменения размеров окна, в которое он помещен.

– пример компонента `StatusBar` из двух панелей и кнопки изменения размеров окна.

Панели задаются в свойстве `Panels`. Каждая панель относится к классу `TStatusPanel`.

ProgressBar – компонент предназначен для отображения хода выполнения длительного по времени процесса (рис. 4).

Рис. 4. Демонстрация компонента `ProgressBar`

Главные свойства компонента:

`Position` - текущее значение числовой величины;

`Max` - максимальное значение диапазона изменения свойства `Position`;

`Min` - минимальное значение диапазона изменения свойства `Position`.

Следующий оператор реализует работу компонента `TProgressBar`:

```
for (i=1;i<100;i++) ProgressBar1->Position=i;
```

UpDown – спаренная кнопка  . Компонент предназначен для регулирования числовой величины. Имеет, как и компонент `ProgressBar`, свойства `Position`, `Max` и `Min`, а также свойство `Increment` - шаг регулируемой величины.

HotKey – компонент служит для ввода и отображения горячих клавиш. При вводе распознает нажатие клавиш Shift, Ctrl и Alt и преобразует их в текст `"Shift+"`, `"Ctrl+"` и `"Alt+"`.

3.4. Компоненты страницы System

Timer – таймер. Служит для отсчета интервалов реального времени. Его свойство `Interval` определяет интервал времени в миллисекундах, который должен пройти от включения таймера до наступления события `OnTimer`. Таймер включается при установке значения `true` в свойстве `Enabled`.

PaintBox – окно для рисования произвольных рисунков. Графические объекты находятся в свойстве `Canvas`, графические инструменты – в свойствах `Brush`, `Pen` и `Font`. Например:

```
PaintBox1->Canvas->Pen->Width=10;//толщина линии 10 пиксел
PaintBox1->Canvas->Brush->Color=clBlue; // цвет заливки синий
PaintBox1->Canvas->Rectangle(10,10,100,100); // рисуется
                                             // прямоугольник
```

MediaPlayer – медиаплейер, представляет собой набор кнопок, предназначенных для управления различными мультимедийными устройствами.

Рис. 5. Кнопки компонента `MediaPlayer`

Если компьютер оснащен звуковой картой, разместите этот компонент на пустой форме, в свойстве `FileName` поместите название любого файла с расширением WAV или MID, установите в свойстве `AutoOpen` компонента значение `TRUE` и запустите программу – после щелчка мышью по кнопке  прозвучит выбранный музыкальный фрагмент.

OleContainer – контейнер для размещения OLE-объектов. Такие объекты (таблицы, картинки и пр.) на форме выглядят как обычно или заменяются пиктограммами. Активизация OLE-объекта с помощью двойного щелчка мышью приводит к активизации связанной с объектом программы, которая называется OLE-сервером и которая после загрузки показывает на экране свое окно и предоставляет пользователю средства редактирования объекта.

Типичными OLE-серверами являются Paint, Word, Excel и др.

Пример создание связанного OLE-объекта:

1. Расположите на пустой форме компонент `OleContainer`. Установите в его свойстве `Align` значение `AllClient`.

2. Сохраните проект в каком-нибудь каталоге и создайте в этом же каталоге графический файл с расширением bmp, например pic.bmp.

3. Напишите такой обработчик события `OnCreate` для формы `Form1`:

```
void __fastcall TForm1::FormCreate(TObject *Sender)
{
OleContainer1->CreateObjectFromFile("pic.bmp",false);
}
```

Запустите программу и дважды щелкните по изображению мышью. Появится окно графического редактора Paint, возможности которого можно использовать для редактирования изображения. Для завершения редактирования нажмите клавишу Esc.

3.5. Компоненты страницы Win3.1

FileListBox – панель с именами файлов.

DirectoryListBox – панель с именами каталогов. Совместно с компонентами `FileListBox`, `DriveComboBox` и `FilterComboBox` может использоваться для создания диалоговых окон доступа к файлам.

DriveComboBox – список выбора с именами доступных дисков.

FilterComboBox – список выбора с расширениями файлов.

3.6. Компоненты страницы Samples

CGauge – отображение некоторой изменяющейся числовой величины. Данный компонент отличается от `ProgressBar` разнообразием форм (рис. 6).

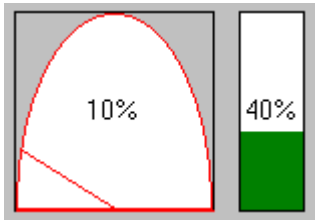


Рис. 6. Примеры вариантов компонента `CGauge`

Главные свойства компонента:

`Kind` - определяет форму индикатора;

`BackColor` - цвет не закрашенной части индикатора;

`ForeColor` - цвет закрашенной части индикатора;

`Progress` - текущее значение числовой величины.

Следующий оператор реализует работу компонента `CGauge`:

```
for (i=1;i<100;i++) CGauge1->Progress=i;
```

CColorGrid – выбор и отображение цвета из 16-цветной палитры. Выбирать можно два цвета: основной и фоновый.

CSpinButton – спаренная кнопка, не связанная с числовой величиной.

CSpinEdit – редактор для ввода целого числа 

Главное свойство компонента `Value` – текущее значение числовой величины. С помощью свойства `Value` также устанавливается и начальное значение числовой величины.

CDirectoryOutLine – отображение древовидной структуры каталогов. В отличие от `DirectoryListBox` компонент отображает полную структуру каталогов, а не маршрут доступа к одному из них.

CCalendar – отображает календарь на выбранный месяц и год. Свойства `Day`, `Month` и `Year` могут содержать любую дату от 1 до 9999 года.

4. Форма

Форма является основным строительным блоком в C++Builder. Любая программа имеет как минимум одну связанную с ней форму, которая называется главной - эта форма появляется в момент старта программы.

4.1. Разновидности форм

Разновидности формы определяются значениями свойств `TFormStyle: fsNormal,fsMDIChild,fsMDIForm,fsStayOnTop.`

`fsNormal` – обычная форма, в том числе форма общего управления программой (главная форма).

`fsMDIChild` и `fsMDIForm` – формы приложений в стиле MDI (Multi Document Interface). Этот стиль предполагает создание главного (рамочного) окна, внутри которого появляются дочерние окна. Дочерние окна не могут выходить за границы рамочного окна и не имеют собственного главного меню. Для создания рамочного окна используется стиль `fsMDIForm`, а для создания дочернего MDI-окна – стиль `fsMDIChild`.

`fsStayOnTop` – окно-поплавок (popup), которое располагается над всеми другими окнами программы.

Современные многооконные приложения чаще всего строятся в стиле SDI (Single Document Interface), который не накладывает ограничения на размер и положение вспомогательных форм. Для создания формы в этом случае используется стиль `fsNormal`.

C++Builder имеет множество стандартных форм-заготовок, предназначенных для решения конкретных задач (опция `File=>New…`). Помимо универсальной пустой формы Form (страница `New`) имеются, например, следующие специализированные формы:

Название	Страница	Назначение
About Box	Forms	Окно "О программе"
Password Dialog	Dialogs	Диалоговое окно с редактором `Edit`, кнопками `OK` и `Cancel` для ввода пароля
Quick Report Wizard	Business	Мастер создания отчетов для баз данных

4.2. Создание приложений с несколькими формами

Выбранная форма автоматически подключается к проекту. Самая первая подключенная к проекту форма (`Form1`) становится главным окном. Можно назначить главным окном любую форму, раскрыв в опции `Project=>Options…` список `Main form` и выбрав нужную форму.

Каждое следующее окно становится видно только после обращения к его методам `Show()` или `ShowModal()`. Для этого нужно вставить ссылку на заголовочный файл этой формы в исходном файле главного окна.
Например, главное окно имеет имя `Form1` и файл с исходным текстом с именем `Unit1.cpp`. Следующее созданное окно имеет имя `Form2` и имя исходного текста `Unit2.cpp`. В файле `Unit1.cpp` помещаем:

```
#include "Unit2.h"
```

после чего вызываем окно на экран:

`Form2->Show();` или `Form2->ShowModal();`

При вызове метода `Show()` второе окно появляется на экране и работает одновременно с первым, поэтому управление сразу передается оператору, стоящему за обращением к этому методу.

Обращение к методу `ShowModal()` создает модальное окно, которое полностью берет на себя дальнейшее управление программой, поэтому оператор за обращением к `ShowModal()` в вызывающей части программы получит управление только после закрытия модального окна.

5. Примеры создания приложений

5.1. Консольное приложение

Для создания консольного приложения в C++Builder необходимо выполнить `File=>New` и в появившемся диалоговом окне на странице `New` выбрать `Console Wizard`. В появившемся окне `Console Wizard` (рис. 7) выбрать для `Source type` значение `C++` и выбрать пункт `Console Application`.

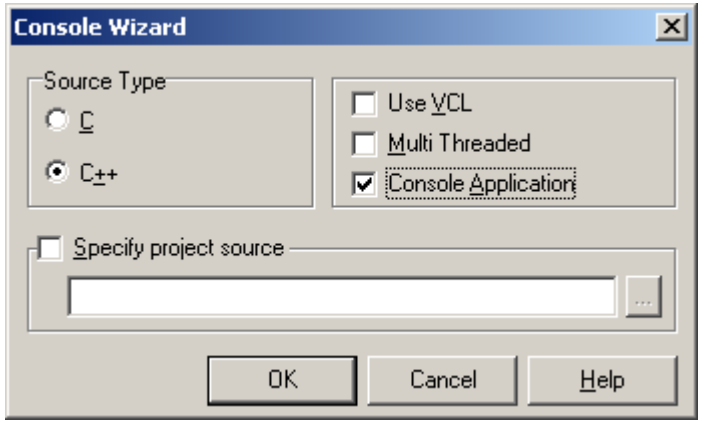


Рис. 7. Окно `Console Wizard`

Появится окно редактора кода. Первоначально окно кода содержит минимальный исходный текст:

```
#pragma hdrstop
#pragma argsused
int main(int argc, char* argv[])
{
        return 0;
}
```

Добавим в программу следующие операторы (выделены жирным шрифтом):

```
#include <stdio.h>
#include <conio.h>
#pragma hdrstop
#pragma argsused
int main(int argc, char* argv[])
{
        printf("Hello world!\n");
        printf("Press any key");
        getch();
        return 0;
}
```

Появится экран с результатом работы программы:

```
Hello world!
Press any key
```

5.2. Windows-приложение

Создадим приложение, вычисляющее значение гипотенузы Z (значения катетов X и Y вводятся). Результат работы приложения представлен на рис. 8.

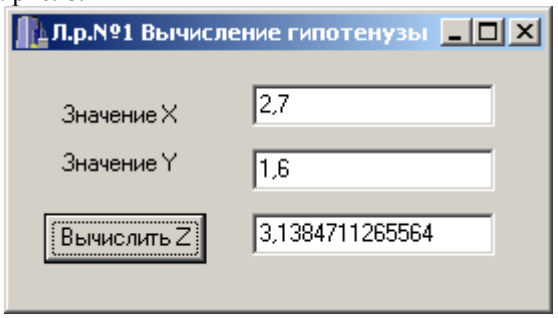


Рис. 8. Окно Windows-приложения

Порядок создания приложения:

1. Выберите страницу `Standard` на Палитре компонентов.
2. Поместите указатель мыши на кнопку A (кнопка компонента `Label`).
3. Щелкните по этой кнопке, а затем щелкните на форме. На форме появится надпись `Label1`.
4. В строке `Caption` окна Инспектора объектов измените стандартную надпись `Label1` на надпись `Значение X`.
5. Аналогично пунктам 2…4 создайте надпись `Значение Y`.
6. Поместите указатель мыши на кнопку abI на Палитре компонентов (кнопка компонента `Edit`).
7. Щелкните по кнопке, затем щелкните на вашей форме. На форме появится поле ввода данных `Edit1`.
8. Замените в строке `Text` окна Инспектора объектов текст `Edit1` на `0` (ноль).
9. Аналогично пунктам 6…8 создайте редакторы данных `Edit2` и `Edit3`.
10. Щелкните по форме в любом месте вне размещенных на ней компонентов.
11. Измените название формы `Form1` на `Л.р.№1 Вычисление гипотенузы` в свойстве `Caption` окна Инспектора объектов.
12. Поместите указатель мыши на кнопку с надписью `OK` на Палитре компонентов (кнопка компонента `Button`).
13. Щелкните по этой кнопке, а затем щелкните на форме. На форме появится кнопка с надписью `Button1`.
14. Поменяйте название кнопки на `Вычислить Z` в свойстве `Caption` окна Инспектора объектов.
15. Перейдите в окне Инспектора объектов на страницу `Events`.
16. Дважды щелкните мышью по пустому полю пункта `OnClick`. В ответ C++ Builder активизирует окно кода программы с функцией

```
void __fastcall TForm1::Button1Click(TObject *Sender)
```

17. Напишите в пустой процедуре фрагмент программы ввода исходных данных, вычисления и вывода на экран результатов расчета.

Текст программы (добавленные операторы выделены жирным шрифтом):

```
//---------------------------------------------------------------
#include <math.h>
#include <vcl.h>
#pragma hdrstop
#include "Unit1.h"
//---------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
//---------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------
void __fastcall TForm1::Button1Click(TObject *Sender)
{
float x,y,z;
x=StrToFloat(Edit1->Text);
y=StrToFloat(Edit2->Text);
z=sqrt(x*x+y*y);
Edit3->Text=FloatToStr(z);
}
//---------------------------------------------------------------
```

Содержание

Учебное издание

РАБОТА В C++BUILDER

Составитель: *Стенгач Михаил Сергеевич*

Редактор Н.С. Куприянова

Лицензия ЛР 020301 от 30.12.96 г.

Подписано в печать __.__.2004 г. Формат 60x84 1/16

Бумага офсетная. Печать офсетная.

Усл. печ.л. 1,62 Усл.кр.-отт. 1,74 Уч.-изд.л. 1,75

Тираж 100 экз. Арт. С-3(Д5)/200_. Зак.

Самарский государственный аэрокосмический университет
имени академика С.П. Королева.
443086 Самара, Московское шоссе, 34.

ИПО СГАУ 443001 Самара, ул. Молодогвардейская, 151.