БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
МЕХАНИКО-МАТЕМАТИЧЕСКИЙ ФАКУЛЬТЕТ

Кафедра численных методов и программирования

ПРОГРАММИРОВАНИЕ В C++ BUILDER

Учебное пособие
по курсу «МЕТОДЫ ПРОГРАММИРОВАНИЯ»
для студентов специальностей
G31 03 01 «Математика», G31 03 03 «Механика»

Минск
2007

УДК 004.43(075.8)
ББК 32.973.26-018.1я73
Б69

А в т о р ы :

В. С. Романчик, А. Е. Люлькин

Р е ц е н з е н т ы :

старший преподаватель *Н. А. Аленский*,
кандидат физико-математических наук, доцент, зав. кафедрой информационных технологий БГУК П. В. Гляков
кандидат физико-математических наук, доцент *С. В. Суздаль*

Рекомендовано Ученым советом
механико-математического факультета БГУ
2006 года, протокол № ___

Программирование в C++ BUILDER: учебное пособие для студ. механико-матем. фак. / В. С. Романчик, А.Е.Люлькин. Мн.: БГУ, 2007. –126 с.
ISBN 985-485-498-1.

В пособии рассматриваются вопросы, относящиеся к использованию технологии объектно-ориентированного программирования в системе C++ Builder. Описание методологии построения классов и использования компонентов сопровождается многочисленными примерами. Предназначено для студентов 2-го курса механико-математического факультета, изучающих курс «Методы программирования».

УДК 004.43(075.8)
ББК 32.973.26-018.1я73

ISBN 985-485-498-1

ВВЕДЕНИЕ

Основные характеристики C++Builder. C++Builder включает язык C++, компилятор, интегрированную среду разработки приложений IDE (Integrated Development Environment), отладчик и различные инструменты. C++Builder содержит комплект общих элементов управления, доступ к Windows API, библиотеку визуальных компонентов VCL (Visual Component Library), компоненты и инструменты для работы с базами данных.

C++Builder добавляет к процессу программирования на языке C++ возможность быстрой визуальной разработки интерфейса приложений. Кроме библиотек OWL (Object Windows Library) и MFC (Microsoft Foundation Classes), он использует библиотеку VCL и позволяет включить в форму диалоги с пользователем, оставляя разработчику для реализации только функциональную часть, воплощающую алгоритм решения задачи.

C++Builder имеет общую с Delphi библиотеку классов, часть из которых осталась написанной на языке Object Pascal. Благодаря этому, а также включению в C++Builder компиляторов C++ и Object Pascal, в приложениях можно использовать компоненты и код, написанные на Object Pascal, а также формы и модули Delphi.

Компоненты C++Builder. Создание пользовательского интерфейса приложения заключается в добавлении в окно формы объектов, называемых компонентами. C++Builder позволяет разработчику создавать собственные компоненты и настраивать Палитру компонентов.

Компоненты разделяются на видимые (визуальные) и невидимые (невизуальные). Визуальные компоненты появляются как во время выполнения, так и во время проектирования. Невизуальные компоненты появляются во время проектирования как пиктограммы на форме. Они не видны во время выполнения, но обладают функциональностью. Для добавления компонента в форму можно выбрать мышью нужный компонент в Палитре компонентов и щелкнуть левой клавишей мыши в нужном месте проектируемой формы. Компонент появится на форме, и далее его можно перемещать и изменять. Каждый компонент C++ Builder имеет три характеристики: свойства, события и методы. Инспектор объ-

ектов автоматически показывает свойства и события, которые могут быть использованы с компонентом. Свойства являются атрибутами компонента, определяющими его внешний вид и поведение. Инспектор объектов отображает опубликованные (published) свойства компонентов на странице свойств (properties) и используется для установки published-свойств во время проектирования. Для изменения свойств компонента во время выполнения приложения нужно добавить соответствующий код. Помимо published-свойств, компоненты могут иметь открытые (public) свойства, которые доступны только во время выполнения приложения.

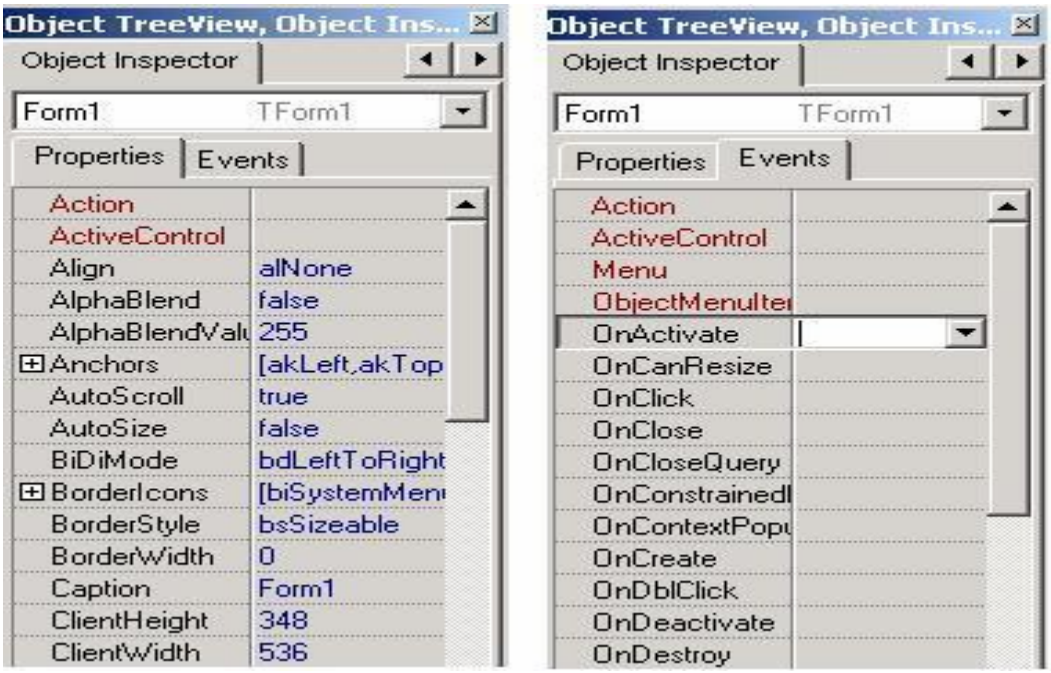

Рис. 1. Окно Инспектора объектов

События. Страница событий (Events) Инспектора объектов показывает список событий, распознаваемых компонентом и возникающих при изменении состояния компонента. Каждый экземпляр компонента имеет свой собственный набор функций - обработчиков событий. Создавая обработчик события, вы поручаете программе выполнить указанную функцию, если это событие произойдет. Чтобы добавить обработчик события, нужно выбрать компонент, затем открыть страницу событий Инспектора объектов и дважды щелкнуть левой клавишей мыши рядом с событием. Это заставит C++ Builder сгенерировать текст пустой функции с курсором в том месте, где следует вводить код. Далее нужно ввести код, который должен выполняться при наступлении данного события.

Среда разработки (IDE). C++ Builder представляет собой приложение, главное окно которого содержит меню (сверху), инструментальную панель (слева) и Палитру компонентов (справа). Помимо этого при запуске C++ Builder появляются окно Инспектора объектов и окно Object TreeView (слева), а также форма нового приложения (справа). Под окном формы приложения находится окно Редактора кода.

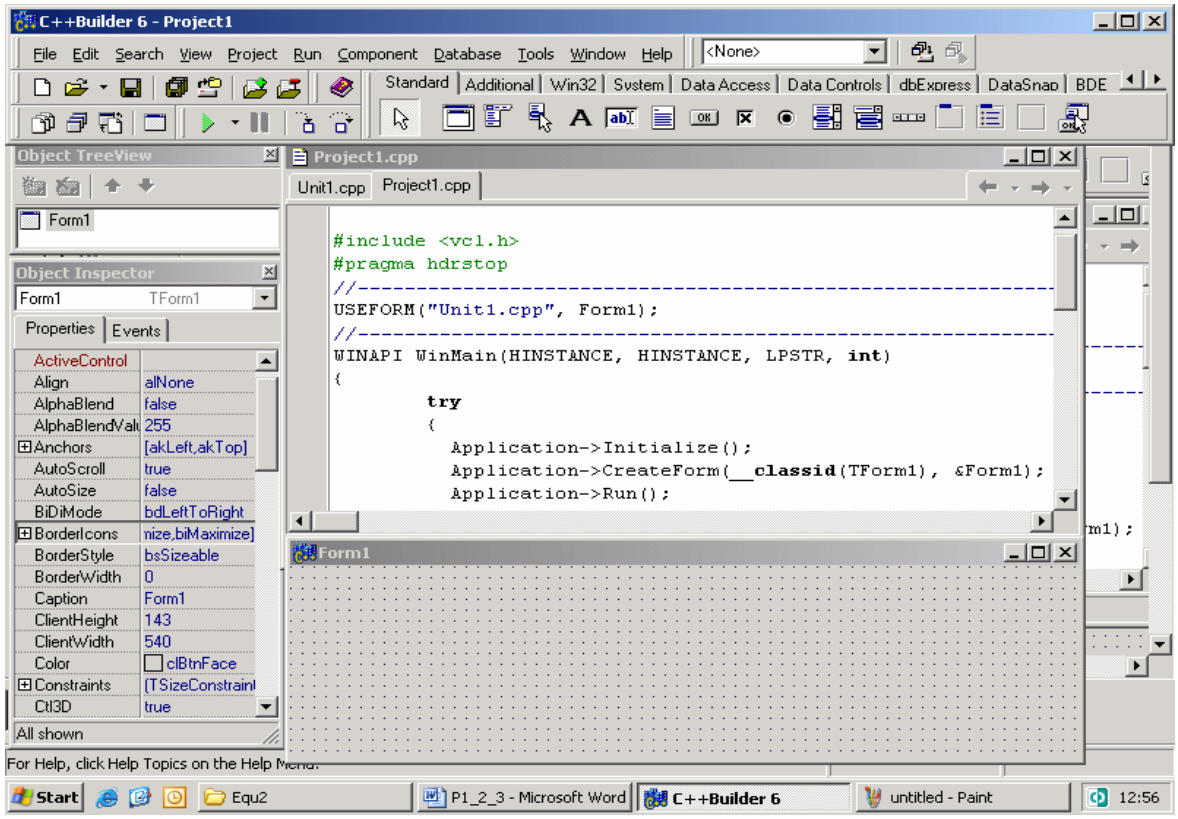

Рис. 2. Главное окно интегрированной среды разработки

Создание приложений в C++Builder. Первым шагом в разработке приложения C++ Builder является создание проекта. Чтобы создать новый проект, нужно выбрать пункт меню File|New| Application.

C++ Builder создает файл Project.bpr, а также головной файл проекта Project.cpp, содержащий функцию WinMain(). Функция WinMain() в Windows-приложениях используется вместо функции main(). При добавление новой формы C++ Builder обновляет файл проекта и создает следующие дополнительные файлы:

- файл формы с расширением .dfm, содержащий информацию о форме;
- файл модуля с расширением .cpp, содержащий код на C++;
- заголовочный файл с расширением .h, содержащий описание класса формы.

Для того чтобы откомпилировать текущий проект, нужно выбрать пункт меню Compile. Для того чтобы откомпилировать проект и создать исполняемый файл, из меню Run нужно выбрать пункт Run. В результате выполнения будет получена следующая форма:

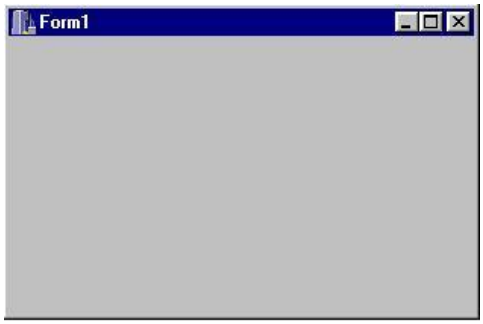

Рис. 3. Результат выполнения приложения

Структура файла проекта. Для каждого приложения C++Builder создается xml-файл проекта Project.bpr и файл ресурсов. Еще один файл - головной файл проекта, содержащий функцию WinMain(), генерируется при выборе пункта меню File|New Application. Первоначально по умолчанию этому файлу присваивается имя Project1.cpp. Если в процессе разработки приложения добавляются формы и модули, C++Builder обновляет файл. Для просмотра файла следует выбрать пункт меню Project|View Source.

В головном файле проекта имеется определенный набор ключевых элементов:

- Директива препроцессора #include <vcl.h> предназначена для включения заголовочного файла, ссылающегося на описания классов библиотеки VCL.
- Директива #pragma hdrstop предназначена для ограничения списка заголовочных файлов, доступных для предварительной компиляции.
- Директива USEFORM показывает модули и формы используемые в проекте.
- Директива USERES компилятора присоединяет файлы ресурсов к выполняемому файлу. При создании проекта автоматически создается файл ресурсов .res для хранения курсоров, пиктограмм и других ресурсов.
- Application->Initialize(). Это утверждение инициализирует приложение.

- Application->CreateForm(). Это утверждение создает форму приложения. Каждая форма в приложении имеет свое утверждение CreateForm.
- Application->Run(). Это утверждение запускает приложение.
- Блок try...catch используется для корректного завершения приложения в случае возникновения ошибки.

Типичный головной файл проекта имеет следующий вид:

```
//Project1.cpp   ----------------------------------------------------
#include <vcl.h>
#pragma hdrstop
USERES("Project1.res");
USEFORM("Unit1.cpp", Form1);
//---------------------------------------------------------------------------
WINAPI WinMain(HINSTANCE, HINSTANCE, LPSTR, int)
{
      try
      {
            Application->Initialize();
            Application->CreateForm(__classid(TForm1), &Form1);
            Application->Run();
      }
      catch (Exception &exception)
      {  Application->ShowException(&exception);  }
      return 0;
}
```

Структура файла Project1.bpr. Файл Project1.bpr представляет XML-проект (C++Builder XML Project), содержащий описание создаваемого приложения. Это текстовый файл, содержащий указания на то, какие файлы должны компилироваться и компоноваться в проект, а также пути к используемым каталогам.

Структура модуля. Модуль содержит реализацию функциональной части объекта на языке C++ и по умолчанию представляет собой файл Unit1.cpp. Каждый такой файл компилируется в объектный файл с расширением .obj. При добавлении к проекту новой формы генерируется новый модуль.

Имя исходного файла модуля и файла формы (*.dfm) должны быть одинаковыми. При создании обработчика событий в тексте модуля генерируется шаблон функции обработчика события, в который вводится код, выполняемый при наступлении обрабатываемого события.

Ниже приводится текст модуля, генерируемый для исходной формы:

```
//Unit1.cpp      ----------------------------------------------------------
#include <vcl.h>
#pragma hdrstop
```

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

Структура заголовочного файла. Заголовочный файл (файл с расширением .h, по умолчанию Unit1.h) генерируется при создании нового модуля и содержит описание класса формы. Такие описания генерируются автоматически и изменяются при внесении в форму новых компонентов или генерации новых обработчиков событий. В заголовочном файле содержится интерфейс, а в самом модуле – реализация методов.

При удалении из формы компонентов их описания удаляются из заголовочного файла. При переименовании компонентов изменяются их описания в заголовочном файле, а также имена и описания обработчиков событий. Однако при этом не изменяются ссылки на эти компоненты и обработчики событий, используемые в других функциях. В связи с этим рекомендуется переименовывать компоненты и обработчики событий сразу же после их создания, пока на них не появились ссылки.

В модуле могут содержаться классы и функции, не описанные в заголовочном файле, однако видимость их в этом случае ограничивается данным модулем.

Ниже приводится заголовочный файл для исходной формы:

```
//Unit1.h-------------------------------------------------------------------------
#ifndef Unit1H
#define Unit1H
//---------------------------------------------------------------------------
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
class TForm1 : public TForm
{
__published:       // IDE-managed Components
private:    // User declarations
public:             // User declarations
      __fastcall TForm1(TComponent* Owner);
};
//---------------------------------------------------------------------------
extern PACKAGE TForm1 *Form1;
#endif
//---------------------------------------------------------------------------
```

Файл формы. Форма является одним из важнейших элементов приложения C++ Builder. Процесс редактирования формы происходит при добавлении к форме компонентов, изменении их свойств, создании обработчиков событий. Когда к проекту добавляется новая форма, создаются три отдельных файла: 1) файл модуля (*.cpp) содержит код методов, связанных с формой; 2) заголовочный файл (*.h) содержит описание класса формы; 3) файл формы (*.dfm) содержит сведения об опубликованных (доступных в Инспекторе объектов) свойствах компонентов, содержащихся в форме.

При добавлении компонента к форме заголовочный файл и файл формы модифицируются. При редактировании свойств компонента в Инспекторе объектов эти изменения сохраняются в файле формы.

Хотя в C++ Builder файл *.dfm сохраняется в двоичном формате, его содержание можно просмотреть с помощью редактора кода. Для этого нужно нажать правую клавишу мыши над формой и из контекстного меню формы выбрать пункт *View as Text*.

Ниже приводится листинг файла некоторой формы:

```
//Unit1.dfm---------------------------------------------------------------------------
object Form1: TForm1
  Left = 197
  Top = 358
  Width = 544
  Height = 181
  Caption = 'Form1'
  Color = clBtnFace
  Font.Charset = DEFAULT_CHARSET
  Font.Color = clWindowText
  Font.Height = -11
  Font.Name = 'MS Sans Serif'
  Font.Style = []
  OldCreateOrder = False
  PixelsPerInch = 96
  TextHeight = 13
End
```

Простейшее приложение. Важнейшей особенностью C++Builder является автоматическая генерация кода программы. Когда к форме добавляете компонент, в тексте файла Unit1.h появляется объявление объекта класса данного компонента. Например, перенос на пустую форму компонента кнопки TButton сгенерирует объявление объекта Button1, а определение события OnClick – объявление метода ButtonlClick , являющегося обработчиком этого события.

Рассмотрим простейшее приложение. Создается форма для сложения двух чисел. Используются компоненты: TEdit – для ввода чисел и отображения результата; TLabel – для вывода строк “+” и “=”; TButton – кнопка, связанная с событием OnClick, для сложения чисел.

Будем использовать следующие свойства и методы для компонентов:

TEdit: Name (имя объекта), ReadOnly (режим "только чтение"), Text (текстовое поле);

TLabel: Caption (текстовое поле);

TButton: Caption (надпись на кнопке), OnClick (событие типа нажатия кнопки).

На рис. 4 приводятся окна Инспектора объектов в процессе работы со свойствами компонентов.

Двойной щелчок по компоненту Button1 приведет к выводу окна редактирования для Unit1.cpp и предложит определить тело метода Button1Click(), которое для рассматриваемой задачи должно иметь вид:

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
Edit3->Text=IntToStr(StrToInt(Edit1->Text)+StrToInt(Edit2->Text));
}
```

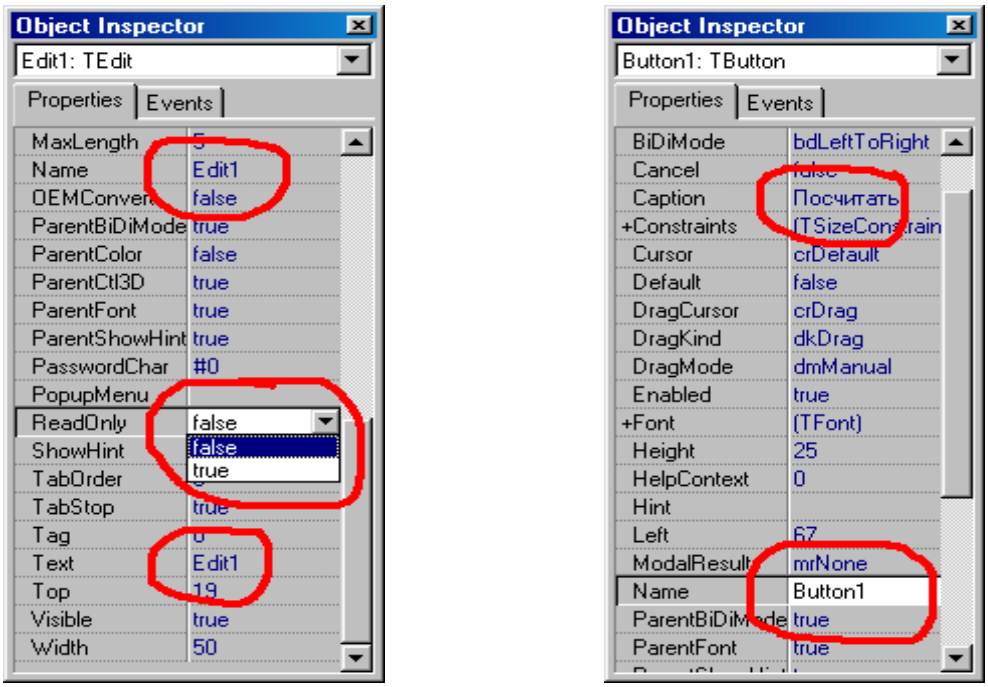

Рис. 4. Работа с компонентами Edit1 и Button1

Для преобразования строки в целое число и обратно используются функции StrToInt() и IntToStr(), соответственно. На рис. 5 показан результат выполнения приложения.

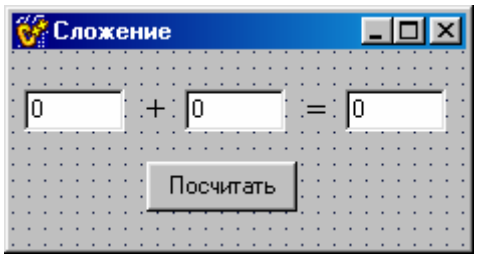

Рис. 5. Форма в процессе выполнения приложения

Рассмотрим основные инструменты визуальной разработки приложений.

Администратор проектов. Предназначен для манипуляций с текущим проектным файлом с расширением .cpp. Чтобы открыть окно администратора, выполните команду View|Project|Manager.

Редактор форм. Форма представляет объект, отображаемый в виде окна с управляющими компонентами. C++Builder создает форму в окне Редактора при добавлении формы к проекту или берет ее из Хранилища объектов.

Инспектор объектов. Инспектор объектов используется при проектировании объектов и методов и имеет две вкладки: Свойства (Properties) и События (Events).

Просмотрщик объектов Object TreeView позволяет просматривать дерево объектов.

Хранилище объектов. Обеспечивает возможность разделения (sharing) или повторного использования (reuse) содержащихся в нем объектов. В качестве объектов хранения могут выступать созданные пользователем формы, проекты, модули данных.

Редактор кода. Предоставляет средство для просмотра и редактирования текста программного модуля (Unit).

Палитра компонентов. C++Builder поставляется вместе с Библиотекой Визуальных Компонентов VCL (Visual Component Library), содержащей множество повторно используемых компонентов. C++Builder позволяет не только пользоваться готовыми компонентами, но и создавать новые компоненты.

Формы и диалоговые окна. Главная форма (Form1) открывается при открытии приложения. Метод Close() закрывает форму и прекращает

выполнение приложения. Свойства формы могут устанавливаться как на этапе проектирования, так и на этапе выполнения приложения. Форма используется в качестве контейнера для размещения других компонентов. В табл. 1 приведены некоторые свойства формы.

С формой связаны события: OnActivate – вызывается при активизации формы после получения фокуса ввода; OnCreate –вызывается при создании формы; OnClose – вызывается при закрытии формы; OnClick – вызывается при щелчке кнопки; OnKeyDown, OnKePress –первое событие активизируется при нажатии любой клавиши, в том числе функциональной, для второго события клавиша не должна быть функциональной; OnMouseDown, OnMouseMove и другие – связаны с мышью. При возникновении каждого события вызывается соответствующй метод-обработчик события. Например:

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

Обработчику события передается указатель на вызвавший его объект. Дополнительными параметрами могут быть, например, текущие координаты указателя мыши и т.п.

Форма создается при создании приложения. Приложение может содержать несколько форм. Существует два вида оконных приложений: SDI и MDI. SDI-приложения могут отображать несколько окон, не привязанных к главному. Каждая форма может отображаться в модальном режиме и требовать закрытия при переходе к другой форме (Form2->ShowModal(), Form2->Close()). Пока модальное окно не закрыто, нельзя перейти к другому окну. В немодальном режиме разрешается доступ к нескольким формам. В качестве примера рассмотрим приложение, состоящее из двух немодальных SDI форм: Form1 и Form3. Создадим кнопку и запишем код обработчика события

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
Form3->Show();
}
```

При этом необходимо включить в заголовочный файл Unit1.h первой формы заголовочный файл второй формы: #include "Unit3.h"

На рис. 6 показан результат выполнения приложения.

Рассмотрим еще одно приложение состоящее из двух форм: Form1 и AboutBox. Для этого в меню выберем File|New|Forms|AboutBox. В свойствах формы AboutBox выберем: Visible=true и FormStyle=fsStayOnTop.

Таблица 1

ActiveControl	Присваивается значение обекта, на который устанавливается фокус ввода при загрузке формы
Cursor	Определяет форму указателя мыши
BorderStyle	Вид рамки объекта. Принимает значения: bsNone (нет рамки); bsSingle (простая рамка); bsSizeable (рамка, позволяющая изменять размеры объекта мышью); bsDialog (рамка в стиле диалоговых окон); bsToolWindow (как bsSingle, но с небольшим заголовком); bsSizeToolWin (как bsSizeable, но с небольшим заголовком)
Caption	Заголовок. Для одних объектов применяется, чтобы задать заголовок в окне или надпись на кнопке, для других – описывает их содержимое (например, у полей надписи)
Constraints	Содержит четыре подсвойства, определяющие минимальный и максимальный допустимый размер объекта
Default	Определяет, будет ли происходить для данного объекта событие OnClick, когда пользователь нажмет клавишу Enter (для этого свойство Default должно иметь значение true)
DragKind	Определяет, можно ли объект произвольно перетаскивать по окну (dkDrag) или же его можно перемещать как стыкуемый объект (dkDock), который сам определяет свою форму при стыковке с другими объектами
Enabled	Определяет доступность объекта. Когда свойство Enabled имеет значение false, объект становится недоступным для пользователя
Font	Определяет шрифт, которым будут делаться все надписи внутри объекта. Содержит множество подсвойств
Height	Высота объекта
Hint	Текст подсказки, которая всплывает при наведении указателя мыши на объект. Эта подсказка будет показываться, если свойство ShowHint установлено в true
Name	Имя объекта
PopupMenu	Контекстное меню, связанное с объектом и вызываемое при щелчке правой кнопки мыши над этим объектом. Выбирается в раскрывающемся списке доступных меню
Scaled	Если имеет значение true, то учитывается свойство PixelsPerlnch
ShowHint	Определяет, надо ли показывать всплывающую подсказку, хранящуюся в свойстве Hint
Top	Верхняя координата объекта на компоненте-родителе
Visible	Определяет, будет ли видим объект во время работы программы (по умолчанию – false)
Width	Ширина объекта

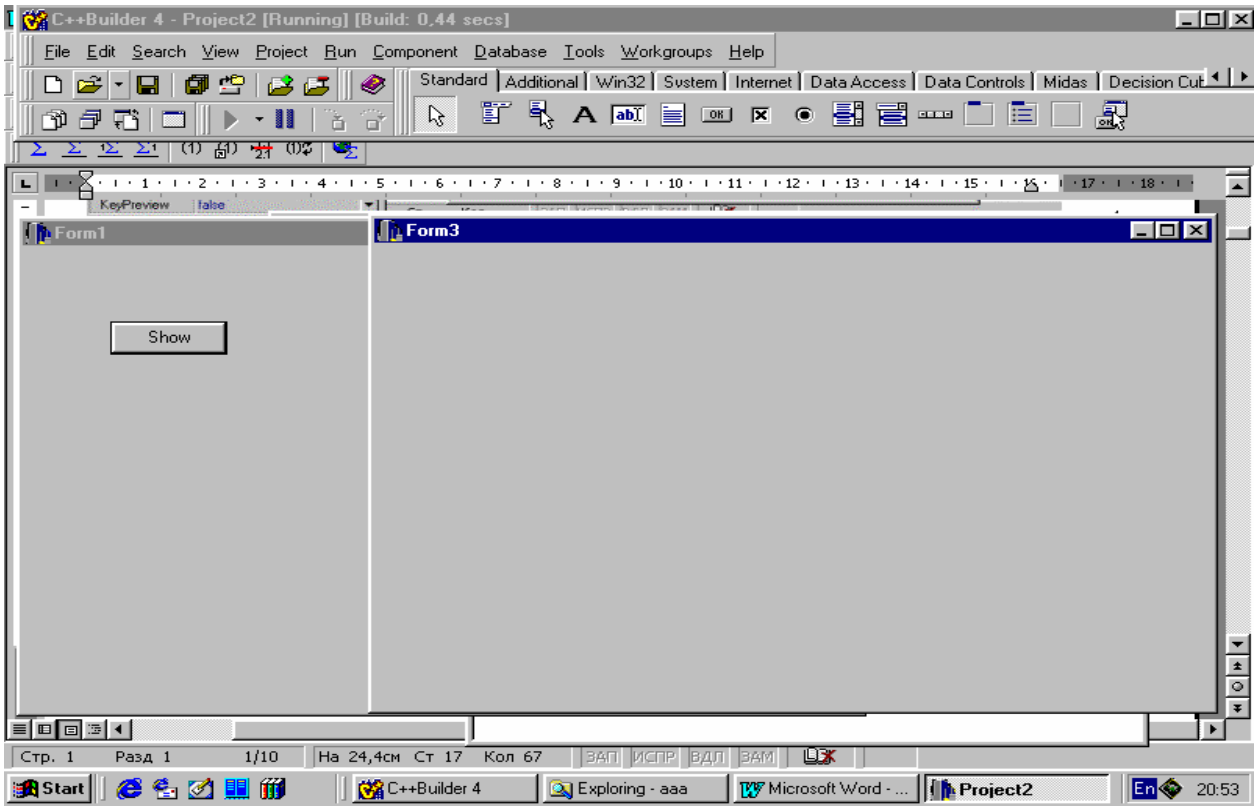

Рис. 6. Результат вывода формы в немодальном режиме

В результате получим следующий головной файл проекта:

```
USERES("Project2.res");
USEFORM("Unit1.cpp", Form1);
USEFORM("Unit2.cpp", AboutBox);
WINAPI WinMain(HINSTANCE, HINSTANCE, LPSTR, int){
        try
        {
                Application->Initialize();
                Application->CreateForm(__classid(TForm1), &Form1);
                Application->CreateForm(__classid(TAboutBox), &AboutBox);
                Application->Run();
        }
        catch (Exception &exception)
        {
                Application->ShowException(&exception);
        }
        return 0;
}
```

Закроем форму AboutBox по нажатию кнопки OK. Для этого запишем следующий обработчик события для кнопки:

```
void __fastcall TAboutBox::OKButtonClick(TObject *Sender)
{
```

Close();
}
На рис. 7 показан результат выполнения приложения.

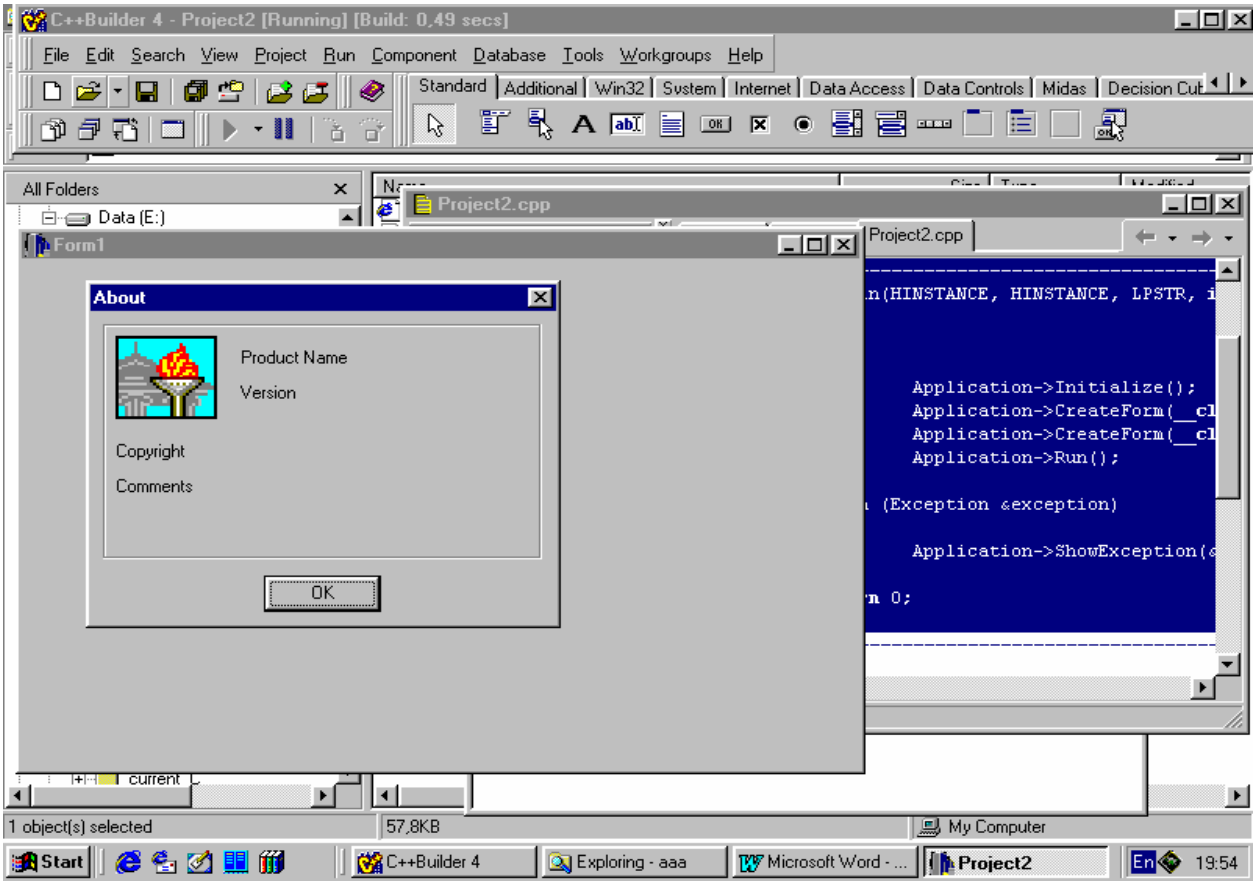

Рис. 7. Приложение с двумя формами

Для создания MDI приложения надо поменять свойство формы FormStyle с fsNormal на fsMDIForm. После этого новая подчиненная форма создается вместе с изменением свойства FormStyle на fsMDIChild. При попытке закрытия подчиненная форма сворачивается. Чтобы закрыть подчиненную форму, можно в обработчике события FormClose набрать: Action=caFree;.

Диалоговые окна. Кроме оконных форм для вывода и ввода строк можно использовать диалоговые окна. Первое окно, используемое для вывода вызывается функцией ShowMessage():

```
void __fastcall TForm1::FormClick(TObject *Sender)
{
ShowMessage("Вывод в окно ShowMessage");
}
```

Это окно открывается модально.

Второе окно, используемое как для вывода, так и для ввода, вызывается функцией InputBox():

```
void __fastcall TForm1::FormKeyDown(TObject *Sender,
WORD &Key,  TShiftState Shift)
{
 String Name=InputBox("InputBox","What is your name","");
}
```

Результаты выполнения этих функций приведены на рис. 8.

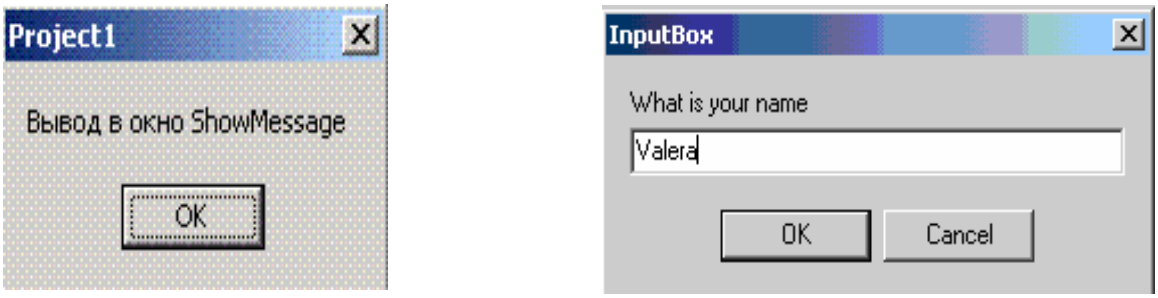

Рис. 8. Примеры диалоговых окон

Окна для ввода и вывода можно также вызывать с помощью методов InputQuery("str1","str2", inputstr) и MessageDlg(str, mtInformation, TMsgDlgButtons() << mbOK, 0). Рассмотрим пример:

```
 void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button,
TShiftState Shift, int X, int Y)
{ String Name;
InputQuery("InputQuery","Prompt",Name);
 MessageDlg("MessageDlg:"+Name, mtInformation, TMsgDlgButtons() << mbOK,
0);
}
```

Результаты выполнения функции приведены на рис. 9.

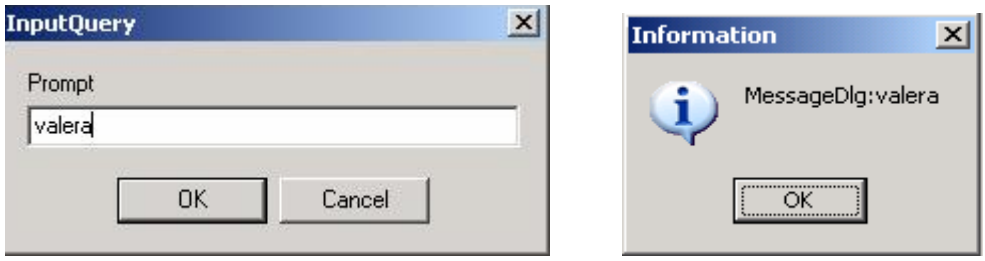

Рис. 9. Результаты вывода диалоговых окон

Вопросы

1. Из каких файлов состоит приложение C++Builder и как организованы эти файлы?

2. Какие коды автоматически генерируются в головном файле приложения при создании этого приложения?

3. В чем отличие действия команд Run, Make и Build при работе с проектом?

4. Каково назначение окон Инспектора объектов?

5. Для чего используется окно Редактора кода ?

6. Что содержит файл представления формы (.dfm)?

7. Что такое событие и как создать обработчик события для компоненты?

8. Какие существуют отличия модальной формы, SDI и MDI форм?

9. Что находится в разделах __published, private и public класса в заголовочном файле формы Unit.h?

10. Что произойдет в результате выполнения кода:

```
{Form1->Button1Click(Form1);}
```

11. Что произойдет в результате выполнения кода:

```
Form1->WindowState = wsMaximized;
Form1->WindowState = wsMinimized;
Form1->WindowState = wsNormal;
```

Упражнения

1. Создать приложение, состоящее из модальной и SDI форм. Создать кнопки для закрытия этих форм. Изменить свойства форм.

2. Создать приложение, состоящее из формы AboutBox и двух MDI форм. Создать кнопки для закрытия этих форм. Изменить свойства форм.

3. Создать форму и написать несколько обработчиков событий, связанных с созданием и изменением свойств формы.

4. Из окна редактирования (Edit) ввести символьную строку и преобразовать в целое и вещественное числа. При преобразовании в число предусмотреть обработку исключительной ситуации

5. Разработать калькулятор, реализующий арифметические операции и операции со стандартными функциями.

6. Разработать калькулятор для перевода чисел в двоичную, восьмеричную и шестнадцатеричную системы счисления и реализовать основные операции.

7. Разработать калькулятор для работы с комплексными числами и реализовать основные операции.

1. C++BUILDER И ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ

1.1. Классы, компоненты и объекты

C++Builder использует понятие *компонентов* – специальных классов, содержащих, кроме обычных данных-членов класса и методов, также свойства и события. Свойства и события позволяют манипулировать видом и функциональным поведением компонентов как на стадии проектирования приложения, так и во время его выполнения.

Свойства (properties) компонентов представляют собой расширение понятия данных-членов и используют ключевое слово **__property** для объявления. При помощи *событий (events)* компонент сообщает пользователю о том, что на нее оказано некоторое воздействие. *Обработчики событий (event handlers)* представляют собой методы, реализующие реакцию программы на возникновение событий. Типичные события – нажатие кнопки или клавиши на клавиатуре. Компоненты имеют ряд особенностей:

• Все компоненты являются прямыми или косвенными потомками класса TComponent. При этом иерархия наследования следующая: Tobject->Tpersistent-> Tcomponent->Tcontrol->… .

• Компоненты используются непосредственно, они не могут служить базовыми классами для построения новых подклассов.

• Компоненты размещаются только в динамической памяти с помощью оператора **new**.

• Компоненты можно добавлять к Палитре компонентов и манипулировать с ними посредством Редактора форм.

1.2. Разработка классов

C++Builder дает возможность объявить базовый класс, который инкапсулирует имена свойств, данных, методов и событий. Каждое объявление внутри класса определяет привилегию доступа к именам класса в зависимости от того, в каком разделе имя появляется. Каждый раздел начинается с одного из ключевых слов: **private, protected** и **public**, определяющих возможности доступа к элементам соответствующего раздела.

Рассмотрим пример объявления класса. Отметим объявление свойства Count в защищенном разделе, а метода SetCount, реализующего запись в данное Fcount, – в закрытом разделе.

```
class TPoint {
 private:
int FCount; // Закрытый член данных
void __fastcall SetCount(int Value);
protected:
__property int Count = // Защищенное свойство
{ read= FCount, write=SetCount };
double x; // Защищенный член данных
double y; // Защищенный член данных
 public:
TPoint(double xVal, double yVal); // Конструктор
double getX();
double getY();
};
```

Объявления класса и определения методов обычно хранятся в разных файлах (с расширениями .h и .cpp, соответственно). Следующий пример показывает, если методы определяются вне класса, то их имена следует уточнять с помощью имени класса.

```
TPoint::TPoint(double xVal, double yVal)
{ // Тело конструктора
}
void __fastcall TPoint::SetCount( int Value ){
if ( Value != FCount )
{
FCount = Value; // Запись нового значения
Update(); // Вызов метода Update
}
}
double TPoint::getX(){
// Тело метода getX(), объявленного в классе TPoint
}
```

1.3. Объявление производных классов

C++Builder дает возможность объявить производный класс, который наследует свойства, данные, методы и события всех своих предшественников в иерархии классов, а также может объявлять новые характеристики и перегружать некоторые из наследуемых функций. Объявление производного класса можно выполнить следующим образом:

```
class derivedClass : [<спецификатор доступа>] parentClass {
private:
<закрытые данные-члены>
<закрытые методы>
protected:
<защищенные данные-члены>
```

```
 <защищенные методы>
public:
<открытые свойства>
<открытые данные-члены>
<открытые конструкторы>
<открытый деструктор>
<открытые методы>
__published:
<опубликованные свойства>
<опубликованные данные-члены>
<объявления дружественных функций>
};
```

Отметим появление нового раздела с ключевым словом **__published –** дополнение, которое C++Builder вводит в стандарт ANSI C++ для объявления опубликованных элементов компонентных классов. Этот раздел отличается от раздела **public** только тем, что компилятор генерирует информацию RTTI (информация времени выполнения) о свойствах, данных-членах и методах объекта и C++Builder организует передачу этой информации Инспектору объектов.

Когда класс порождается от базового, все имена базового класса в производном классе автоматически становятся закрытыми по умолчанию (если спецификатор доступа при наследовании не указывается). Но это можно изменить, указав следующие спецификаторы доступа при наследовании базового класса:

• **protected.** Наследуемые (т.е. защищенные и открытые) имена базового класса становятся защищенными в экземплярах производного класса.

• **public.** Открытые имена базового класса и его предшественников будут открытыми в экземплярах производного класса, а все защищенные останутся защищенными.

Рассмотрим применение методики расширения и ограничения характеристик на примере создания разновидностей кнопки при наследовании базового компонента TButtonControl из Библиотеки Визуальных Компонентов. Базовый класс TButtonControl способен с помощью родительского метода Draw отображать кнопку в виде двух вложенных прямоугольников: внешней рамки и внутренней закрашенной области. Чтобы создать простую кнопку без рамки, нужно построить производный класс SimpleButton, использовав в качестве родительского TButtonControl, и перегрузить метод Draw:

```
class SimpleButton: public TButtonControl {
public:
SimpleButton(int x, int y) ;
```

```
void Draw() ;
~SimpleButton() { }
};
SimpleButton::SimpleButton(int x, int y) : TButtonControl(x, y)
 { }
void SimpleButton::Draw()
{outline->Draw();}
```

Единственная задача конструктора объекта для SimpleButton – вызвать конструктор базового класса с двумя параметрами. Именно переопределение метода SimpleButton:: Draw () предотвращает вывод обводящей рамки кнопки (как происходит в родительском классе). Чтобы изменить код метода, надо изучить его по исходному тексту базового компонента TButtonControl.

Создадим кнопку с пояснительным названием. Для этого нужно построить производный класс TextButton из базового TButtonControl и перегрузить метод Draw с расширением его функциональности:

```
class Text {//Вспомогательный класс
 public:
Text(int x, int y, char* string) { }
 void Draw() { }
};
class TextButton: public TButtonControl {
Text* title;
public:
TextButton(int x, int y, char* title);
void Draw();
~TextButton() { }
};
TextButton::TextButton(int x, int y, char* caption):
TButtonControl(x, y) {
title = new Text(x, y, caption);
}
void TextButton::Draw () {
TButtonControl::Draw() ;
title->Draw() ;
}
```

1.4. Идентификация типов времени выполнения RTTI

Идентификация типов при выполнении программы RTTI (Run-Time Type Identification) позволяет вам написать переносимую программу, которая способна определять фактический тип объекта в момент выполнения даже в том случае, если программе доступен только указатель на этот объект. Это дает возможность, например, преобразовывать тип ука-

зателя на базовый класс в указатель на производный тип фактического объекта данного класса. Таким образом, преобразование типов может происходить не только статически – на фазе компиляции, но и динамически – в процессе выполнения. Динамическое преобразование указателя в заданный тип осуществляется с помощью оператора dynamic_cast.

Механизм RTTI также позволяет проверять, имеет ли объект некоторый определенный тип, или принадлежат ли два объекта одному и тому же типу. Оператор typeid определяет фактический тип аргумента и возвращает указатель на объект класса typeinfo, который этот тип описывает.

Передавая RTTI Инспектору объектов во время выполнения, C++Builder информирует его о типах свойств и членов данного класса.

1.5. Пакеты

Пакеты – это особый тип динамических библиотек *DLL* для Windows. Как и обычные *DLL,* пакетные файлы *BPL (Borland Package Library)* содержат код, разделяемый многими приложениями. C++Builder размещает наиболее часто используемые компоненты в пакете под названием *VCL50.BPL.* При создании исполняемого кода приложения в нем остаются только уникальные инструкции и данные, а разделяемый код подгружается из указанных пакетов во время исполнения.

Объявление нового компонентного класса в интерфейсном модуле должно включать предопределенный макрос PACKAGE сразу же за ключевым словом class:

class PACKAGE MyComponent : ...

Этот же макрос должен присутствовать в кодовом модуле там, где объявлена функция регистрации компонента:

void __fastcall PACKAGE Register(){…}

Образующийся при подстановке макроса PACKAGE код обеспечивает возможность импортирования и экспортирования объявленного компонентного класса в результирующий файл с расширением *BPL.* Если при создании нового компонента вы пользуетесь мастером (по команде Component | New Component), C++Builder автоматически вводит PACKAGE в нужное место.

1.6. Объявления компонентных классов

Опережающие объявления классов Библиотеки Визуальных Компонентов VCL, входящей в состав C++Builder, используют модификатор **_declspec**:

_declspec(<спецификатор>)

Это ключевое слово может появляться в любом месте перечня объявлений, причем спецификатор принимает одно из следующих значений:

delphiclass используется для опережающего объявления прямых или косвенных производных от VCL-класса TObject;

delphireturn используется для опережающего объявления прямых или косвенных производных от VCL-классов Currency, AnsiString, Variant, TDateTime и Set. Он определяет правила совместимости VCL при обращении с параметрами и возвращаемыми значениями функций-членов.

pascalimplementation указывает, что компонентный класс реализован на Объектном Паскале.

1.7. Объявления свойств

C++BuiIder использует модификатор **_property** для объявления свойств компонентных классов. Синтаксис описания свойства имеет вид:
_property <тип свойства> <имя свойства> = {<список атрибутов>} ;

Список атрибутов содержит перечисление следующих атрибутов свойства:

write = < член данных или метод записи > – определяет способ присваивания значения члену данных;

read = **<** член данных или метод чтения > – определяет способ получения значения члена данных;

default = < булева константа > – разрешает или запрещает сохранение значения свойства по умолчанию в файле формы *.dfm;

stored = < булева константа или функция > – определяет способ сохранения значения свойства в файле формы с расширением *.dfm.

C++BuiIder использует модификатор **__published** для спецификации тех свойств компонентов, которые будут отображаться Инспектором объектов на стадии проектирования приложения. Правила видимости, определяемые этим ключевым словом, не отличаются от правил видимости членов данных, методов и свойств, объявленных как **public.**

1.8. Объявления обработчиков событий

C++Builder использует модификатор **_closure** для объявления функций обработчиков событий:

<тип> (**_closure** * <name>) (<список параметров>)

Это ключевое слово определяет указатель функции с именем name. В отличие от 4-байтового указателя обычной функции, 8-байтовый указатель **_closure** передает еще и скрытый указатель **this** на экземпляр класса.

Введение 8-байтовых указателей делает возможным вызывать некоторую функцию определенного класса, а также обращаться к функции в определенном экземпляре этого класса.

1.9. Право владения

Любой компонент может находиться во *владении (ownership)* других компонентов. Свойство компонента Owner (Владелец) содержит ссылку на компонент, который им владеет. Владелец отвечает за освобождение тех компонентов, которыми владеет, когда сам разрушается. Так, в процессе конструирования формы она автоматически становится владельцем всех компонентов, размещенных на ней, даже если часть их размещена на другом компоненте, например, таком как TPanel. Владение применимо не только к видимым, но и к невидимым (TTimer, TDataSource) компонентам.

Когда компонент не переносится на форму, а создается динамически в процессе выполнения программы, конструктору компонента передается ее владелец в качестве параметра. В следующем примере неявный владелец (например, форма) передается конструктору компонента TButton как параметр. Конструктор TButton выполнит присваивание значения переданного параметра свойству Owner кнопки MyButton:

MyButton = **new** TButton(this);

Когда форма, уничтожается, автоматически уничтожается и кнопка MyButton.

Можно создать компонент, у которого нет владельца, передавая значение параметра 0 конструктору компонента. Его уничтожение выполняется с помощью оператора **delete.**

Свойство Components класса Tcomponent содержит перечень компонентов, которыми владеет данный компонент. Ниже приводится фрагмент кода обработчика события *OnClick* с циклом отображения имен классов всех компонентов, которыми владеет некоторая форма.

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
for (int i=0; i<ComponentCount; i++){
ShowMessage(Components[i]->ClassName()) ;
ShowMessage(Components[i]->ClassParent()->ClassName()) ;
 }
```

```
}
```

Метод ClassParent() возвращает родительские классы для находящихся на форме классов.

1.10. Родительское право

Понятие *родительского права (parentship)* отличается от права владения и применимо только к видимым компонентам. В качестве родительских компонент могут выступать форма, панель, groupbox, на которых располагаются объекты – потомки. Родительские компоненты обращаются к соответствующим внутренним функциям, чтобы вызвать отображение компонентов-потомков. Родитель также отвечает за освобождение своих потомков, когда сам родитель уничтожается. Свойство компонента Parent (Родитель) содержит ссылку на компонент, который является его родителем. Многие свойства видимых компонентов (например, Left, Width, Top, Height) относятся к родительским элементам управления. Другие свойства (например, ParentColor и ParentFont) позволяют потомкам использовать свойства родителей.

Компоненту надо присвоить родителя, ответственного за отображение. Это присваивание выполняется автоматически на стадии проектирования при перетаскивании компонента из Палитры компонентов на форму. При создании компонента во время выполнения программы необходимо явно записать это присваивание, иначе компонент не будет отображен:

```
void __fastcall TForm1::FormCreate(TObject *Sender)
{
MyEdit = new TEdit(this); // Передать this как владельца
MyEdit->Parent = this; // Передать this как родителя
}
```

В качестве примера приведем объявление компонента с единственным свойством IsTrue, имеющим значение по умолчанию true, а также конструктор, который устанавливает это значение при инициализации компонентного объекта. Заметим, что если свойство имеет значение по умолчанию false, то не нужно явно устанавливать его в конструкторе, поскольку все объекты (включая компоненты) всегда инициализируют свои члены данных значением 0, т.е. false.

```
class TMyComponent : public TComponent
{ private:
Boolean FIsTrue;
public:
__fastcall TMyComponent(TComponent* Owner);
__published:
```

```
__property Boolean IsTrue =
{ read=FIsTrue, write=FIsTrue, default=true };
};
__fastcall TMyComponent:: TMyComponent (TComponent* Owner) : TComponent
(Owner) { FIsTrue = true; }
```

Вопросы

1. Каким образом, и в каких файлах можно создать собственный класс, динамический объект и вызвать методы класса?

2. Что представляют собой свойства компонентов? Приведите примеры нескольких свойств формы.

3. Что представляют собой события компонентов? Приведите примеры.

4. Каким образом можно изменить значения свойств компонентов?

5. Объявить класс MyClass, содержащий некоторую строку, и метод swap(), заменяющий строку другой строкой. Выполнить тестирование.

6. Объявить класс и определить размерность объектов данного класса.

7. Чем отличаются оператор-функции, объявленные как friend, от оператор-функций членов класса C++ и можно ли их использовать в C++Builder?

8. На каком этапе происходит выделение памяти под объекты компонентных классов?

9. Что представляет собой общедоступное свойство класса и его опубликованное свойство?

10. Что поизойдет в результате выполнения следующего кода?

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
TForm * newForm= new TForm(this);
TButton* button=new TButton(Application);
button->Parent=newForm;
button->Caption="New Button";
button->Left=10;
button->Top=15;
button->Show();
newForm->Caption="newForm";
newForm->ShowModal();
button->Click();
delete newForm;
}
```

12. Как распознать нажатые функциональные клавиши?

13. Куда поступают события клавиатуры? Какое свойство формы влияет на то, куда поступают события клавиатуры?

14. В следующем коде какой из объектов отвечает за удаление кнопки?

```
TButton* B=new TButton(this);
B->Parent=Panel1;
```

2. КОМПОНЕНТЫ БИБЛИОТЕКИ VCL

2.1. Стандартные компоненты

Компоненты вкладки Standard Палитры компонентов осуществляют включение в программу стандартных управляющих элементов Windows.

Компонент TLabel отображает статический текст, являющийся значением свойства Caption. Например:

Label1->Caption="Это Метка";

Компонент TEdit отображает область редактируемого ввода строки. Содержимое области редактирования определяется значением свойства Text. Например:

```
double r1=Form1->Edit1->Text.ToDouble(), y=0,
r2=Form1->Edit2->Text.ToDouble();
y=r1+r2;
ShowMessage(FloatToStr(y));
```

Компонент TButton создает кнопку с надписью. Нажатие на кнопку инициирует некоторое событие. Кнопка, выбранная со значением true свойства Default, запускает обработчик события OnClick для кнопки всякий раз, когда нажимается клавиша Enter в окне диалога. Кнопка прерывания, выбранная со значением true свойства Cancel, запускает обработчик события OnClick для кнопки всякий раз, когда нажимается клавиша Escape.

Ниже приводится пример использования рассматриваемых компонентов Label1-Label4, Edit1-Edit6 и Button1 при решении квадратного уравнения.

```
//Uni1.cpp
#include <vcl.h>
#pragma hdrstop
#include <math.h>
#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)
{
try{
double a=StrToFloat(Edit1->Text);
double b=StrToFloat(Edit2->Text);
double c=StrToFloat(Edit3->Text);
if(a==0){ShowMessage("Уравнение не квадратное");return;}
 double d = b * b - 4 * a * c;
  String str="";
    if( d < 0)
    {
      str += ( - b ) / ( 2 * a);
      str += " + i * ";
      str += sqrt( - d ) / ( 2 * a);
      Edit4->Text = str;
      str = "";
      str += ( - b ) / ( 2 * a);
      str += " - i * ";
      str += sqrt( - d ) / ( 2 * a);
      Edit5->Text = str;
    }
    else
    {
      Edit4->Text = ( - b + sqrt( d ) ) / ( 2 * a);
      Edit5->Text = ( - b - sqrt( d ) ) / ( 2 * a);
    }
    Edit6->Text=d;
    }
    catch(...)
    {ShowMessage("Input Error");}
  }
```

На рис. 10 показана форма приложения в процессе выполнения.

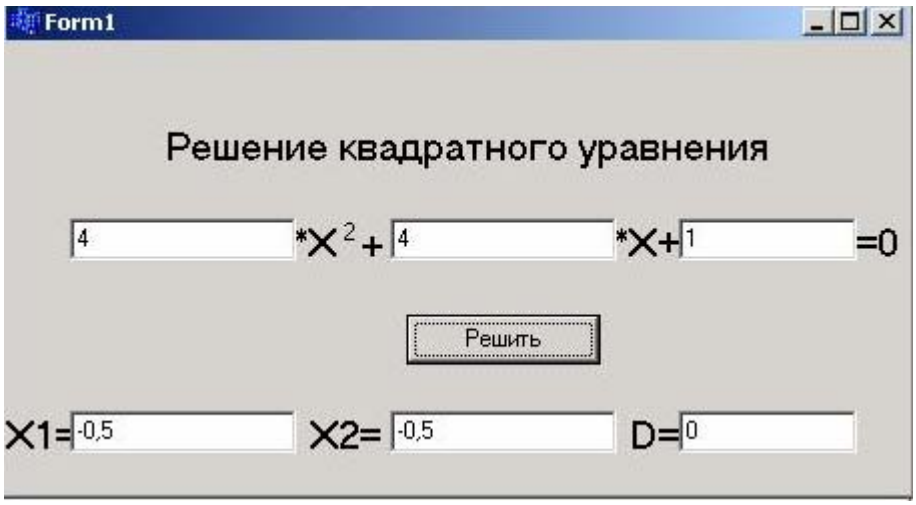

Рис. 10. Форма в процессе выполнения

Можно использовать кнопку с картинкой вместо кнопки типа TButton. Для этого можно применить компонент TBitBtn со вкладки Additional Палитры компонентов.

Компонент TMemo отображает прямоугольную область ввода множества строк. Содержимое области редактирования определяет массив строк, являющийся значением свойства Lines. Например:

```
void __fastcall TForm1::Edit1Change(TObject *Sender)
{ Memo1->Lines->Add(Edit1->Text);
}
```

Компонент TCheckBox создает квадратный флажок с двумя состояниями. Состояние check флажка соответствует выбору некоторого варианта (отмечается перечеркиванием квадрата), а состояние uncheck соответствует снятию выбора. При этом свойство компонента Checked меняется с true на false и возникает событие OnClick. Описательный текст флажка хранится в свойстве Caption.

Компонент TRadioButton создает круглую кнопку (радиокнопку) с двумя состояниями и описательным текстом. Радиокнопки представляют набор взаимоисключающих вариантов выбора: только одна кнопка может быть выбрана в данный момент времени (отмечается внутренним черным кружком), а с ранее выбранной кнопки выбор автоматически снимается. При нажатии радиокнопки свойство компонента Checked меняется и возникает событие OnClick.

Обычно радиокнопки размещаются внутри предварительно установленного на форме группового контейнера. Если выбрана одна кнопка, выбор всех прочих кнопок в той же группе автоматически снимается. Например, две радиокнопки на форме могут быть выбраны одновременно только в том случае, когда они размещены в разных

контейнерах. Если группировка радиокнопок явно не задана, то по умолчанию все они группируются в одном из оконных контейнеров (TForm, TGroupBox или TPanel).

Компонент TListBox отображает прямоугольную область списка текстовых вариантов для выбора, добавления или вычеркивания. Если все элементы списка не умещаются в отведенную область, то список можно просматривать с помощью линейки прокрутки. Элементы списка содержатся в свойстве Items, а номер элемента, который будет выбран во время выполнения программы, – в свойстве ItemIndex. Окно текстового редактора элементов списка открывается кнопкой в графе значений свойства Items. Можно динамически добавлять, вычеркивать, вставлять и перемещать элементы списка с помощью методов Add, Append, Delete и Insert объекта Items. Например:

ListBoxl->Items->Add("Последний элемент списка");

Значение true свойства Sorted устанавливает сортировку элементов списка по алфавиту.

Рассмотрим пример приложения, позволяющего вводить текст в редактируемое поле и добавлять этот текст к списку при нажатии на кнопку. Выберем пункт меню File|New Application для создания проекта и сохраним его главную форму под именем samp1.cpp, а сам проект под именем samp.bpr. Поместим на форму компоненты TButton, TEdit, TRadioButton и TListBox со страницы Standard Палитры компонентов. После этого выберем на форме компонент Edit1 и удалим текущее значение свойства Text. Затем установим свойство Caption для RadioButton1 равным "Добавить", для RadioButton2 – "Удалить", для кнопки Button1 – "Выполнить", а для кнопки Button2 – "Выход".

Чтобы добавить обработчик события OnClick для кнопки «Выполнить», нужно выбрать эту кнопку на форме, открыть страницу событий в Инспекторе объектов и дважды щелкнуть мышью на колонке справа от события OnClick. В соответствующей строке ввода появится имя функции. C++ Builder сгенерирует прототип обработчика события и покажет его в редакторе кода. После этого следует ввести следующий код в тело функции:

```
    void __fastcall TForm1::Button1Click(TObject *Sender)
{
     if(RadioButton1->Checked)
     {
      if (!(Edit1->Text == ""))
    {
       ListBox1->Items->Add(Edit1->Text);
      Edit1->Text = "" ;
```

```
      }
      }
      else
      {
      if(RadioButton2->Checked){
      if (!(ListBox1->ItemIndex == -1))
       ListBox1->Items->Delete(ListBox1->ItemIndex);
      }
      }
}
```

В обработчик события OnClick для кнопки "Выход" добавим вызов функции Close().

Для компиляции приложения в меню Run выберем пункт Run. На рис. 11 показано окно приложения в процессе выполнения.

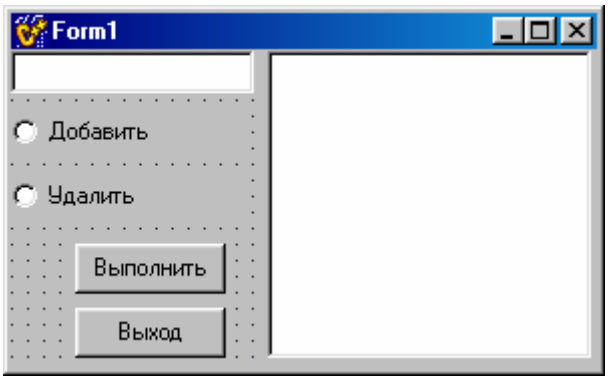

Рис. 11. Окно приложения в процессе выполнения

Компонент TComboBox создает комбинацию области редактирования и выпадающего списка текстовых вариантов для выбора. Значение свойства Text заносится непосредственно в область редактирования. Элементы списка, которые может выбирать пользователь, содержатся в свойстве Items, номер элемента, который будет выбран во время выполнения программы, – в свойстве ItemIndex, а сам выбранный текст – в свойстве SelText. Свойства SelStart и SelLength позволяют установить выборку части текста или обнаружить, какая часть текста выбрана.

Можно динамически добавлять, вычеркивать, вставлять и перемещать элементы списка с помощью методов Add, Append, Delete и Insert объекта Items. Например:

ComboBoxl->Items->Insert(0, "Первый элемент списка");

Компонент TScrollBar создает линейку прокрутки с бегунком для просмотра содержимого окна. Поведение прокручиваемого объекта определяется обработчиком события OnScroll. Значение свойства LargeChange определяет насколько должен продвинуться бегунок, когда

пользователь щелкает мышью на самой линейке (по обеим сторонам от бегунка). Значение свойства SmallChange определяет насколько должен продвинуться бегунок, когда пользователь щелкает мышью по кнопкам со стрелками (на концах линейки) или нажимает клавиши позиционирования.

Рассмотрим пример взаимодействия компонентов TScrollBar и TComboBox, которое происходит следующим образом: изменение положения бегунка компонента TScrollBar вызывает изменение соответствующего ему элемента списка компонента TComboBox и наоборот. Ниже приводится листинг программы, содержащий обработчики событий компонентов:

```
//Unit1.cpp
#include <vcl.h>
#pragma hdrstop
#include "desyat.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
//---------------------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner) : TForm(Owner)
{ }
//---------------------------------------------------------------------------
void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
 ScrollBar1->Position=ComboBox1->ItemIndex+1;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::ScrollBar1Change(TObject *Sender)
{
ComboBox1->ItemIndex=ScrollBar1->Position-1;
ComboBox1->Items->Add(ScrollBar1->Position-1);
}
//---------------------------------------------------------------------------
```

На рис. 12 показано окно программы в процессе выполнения.

Компонент TGroupBox создает контейнер в виде прямоугольной рамки, визуально объединяющий на форме логически связанную группу некоторых интерфейсных элементов. Этот компонент представляет собой инкапсуляцию одноименного объекта Windows.

Компонент TRadioGroup создает контейнер в виде прямоугольной рамки, визуально объединяющий на форме группу логически взаимоисключающих радиокнопок.

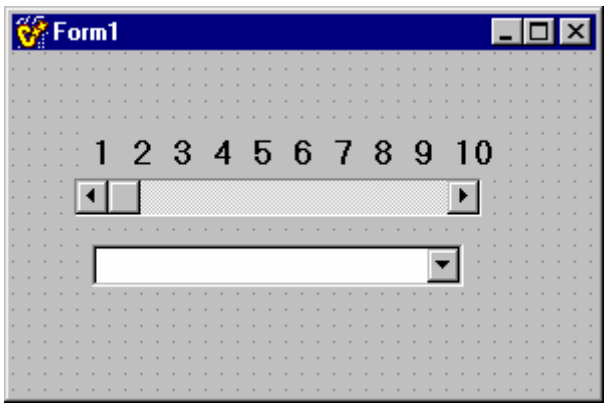

Рис. 12. Окно программы

Радиокнопки образуют группу при помещении их в один и тот же контейнер. Только одна кнопка из данной группы может быть выбрана. Добавление кнопок к компоненту TRadioGroup выполняется редактированием свойства Items. Присвоение названия радиокнопки очередной строке свойства Items приводит к появлению этой кнопки в группирующей рамке. Значение свойства ItemIndex определяет, какая радиокнопка выбрана в настоящий момент. Можно группировать радиокнопки в несколько столбцов, устанавливая соответствующее значение свойства Columns.

Компонент TPanel создает пустую панель (контейнер), которая может содержать другие компоненты. Можно использовать компонент TPanel для создания на форме панелей инструментов или строк состояния (однако для этого лучше использовать специальные компоненты TToolBar, TPageScroller, TStatusBar).

Компонент TMainMenu создает панель команд главного меню и соответствующие им выпадающие меню для формы.

В качестве примера приведем создание простейшего текстового редактора с возможностью открытия и сохранения текстовых файлов. Будем использовать компоненты TMainMenu, TOpenDialog, TSaveDialog, TMemo. На рис. 13 показан процесс проектирования меню с помощью редактора меню.

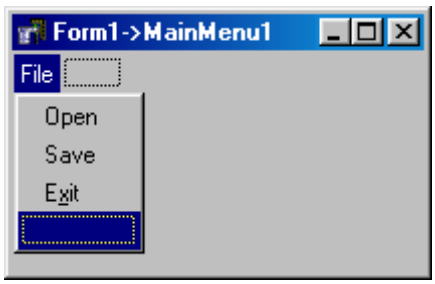

Рис. 13. Редактирование меню

Далее приводятся обработчики событий для соответствующих пунктов меню:

```
void __fastcall TForm1::Open1Click(TObject *Sender)
{
// Программная установка свойств компонента OpenDialog1
  OpenDialog1->Options.Clear();
  OpenDialog1->Options << ofFileMustExist;
  OpenDialog1->Filter = "Text files (*.txt)|*.txt|All files (*.*)|*.*";
  OpenDialog1->FilterIndex = 2;
  if (OpenDialog1->Execute()) Memo1->Lines->
  LoadFromFile (OpenDialog1->FileName);
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Save1Click(TObject *Sender)
{
// Программная установка свойств компонента SaveDialog1

  SaveDialog1->Options.Clear();
  SaveDialog1->Options << ofFileMustExist;
  SaveDialog1->Filter = "Text files (*.txt)|*.txt|All files (*.*)|*.*";
  SaveDialog1->FilterIndex = 2;
  if (SaveDialog1->Execute()) Memo1->
  Lines->SaveToFile(SaveDialog1->FileName);
}
void __fastcall TForm1::Exit1Click(TObject *Sender)
{
 exit(0);
}
```

На рис. 14 показано окно приложения после запуска приложения и открытия в полученном текстовом редакторе файла модуля формы.

Компонент TPopupMenu создает всплывающее меню для формы или для другого компонента. Если необходимо, чтобы специальное меню появлялось при нажатии правой кнопки мыши на элемент, которому приписан данный компонент (свойство PopupMenu содержит имя компонента всплывающего меню), установите значение true свойства AutoPopup. Если всплывающее меню относится к форме, то свойство формы PopupMenu следует установить в имя всплывающего меню, например, PopupMenu1.

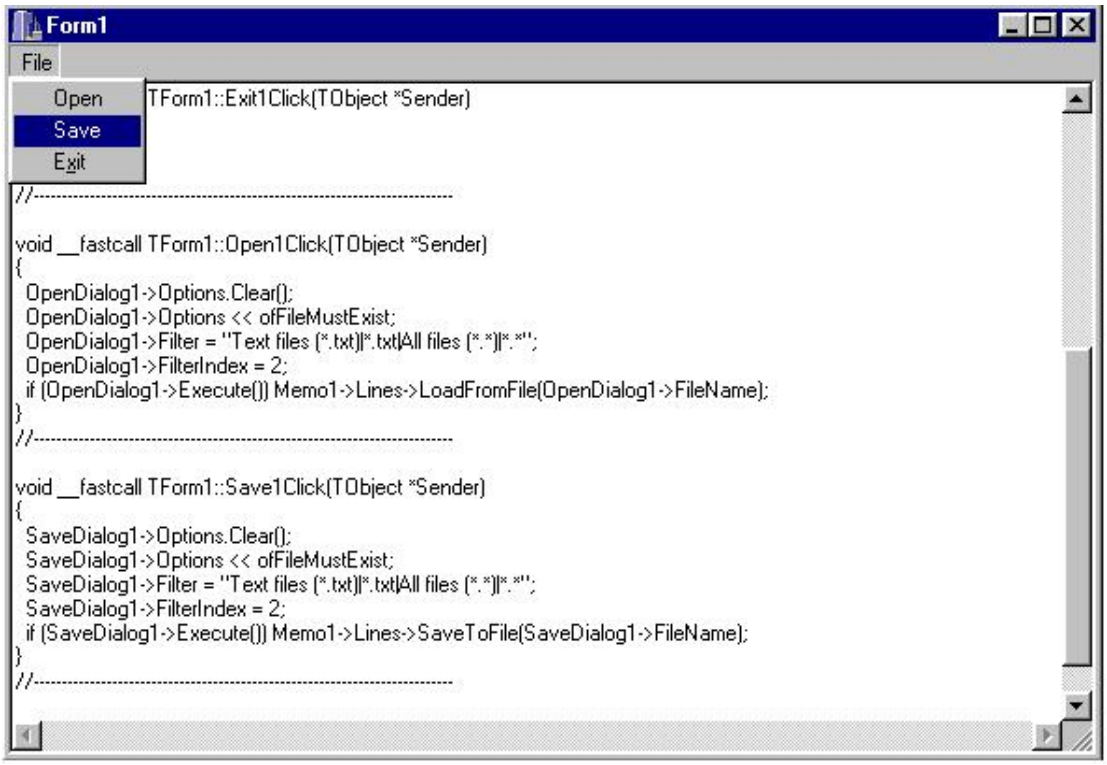

Рис. 14. Окно текстового редактора

2.2. Компоненты Win32

Компоненты вкладки Win32 Палитры компонентов осуществляют включение в программу следующих интерфейсных элементов.

Компонент TTabControl отображает набор частично перекрывающих друг друга картотечных вкладок. Названия вкладок вводятся в список свойства Tabs кнопкой в графе значений этого свойства. Если все поля не умещаются на форме в один ряд, то можно установить значение true свойства MultiLine или прокручивать вкладки с помощью кнопок со стрелками. Принципиальное отличие данного компонента от похожего компонента TPageControl состоит в том, что он не имеет множества страниц (панелей). Компонент представляет собой одну страницу с управляющим элементом типа кнопки со многими положениями. Необходимо написать соответствующие обработчики событий OnChanging и OnChange, чтобы определить, что именно должно происходить на панели при переключениях закладок пользователем.

Компонент TPageControl отображает набор полей, имеющих вид частично перекрывающих друг друга картотечных вкладок (страниц), для организации многостраничного диалога. Многостраничные панели

позволяют экономить пространство окна приложения, размещая на одном и том же месте страницы разного содержания. Таким образом, компонент TPageControl является контейнером для размещения нескольких страниц, которые можно открывать в процессе работы приложения спомощью щелчка на соответствующей вкладке.

Чтобы создать новую страницу диалога с соответствующей вкладкой, выберите опцию New Page из контекстного меню данного компонента. Можно активизировать конкретную страницу одним из следующих способов: с помощью мыши, выбрав ее из выпадающего списка свойства ActivePage, а также перелистывая вкладки с помощью опций Next Page и Previous Page контекстного меню. Свойство PageIndex содержит номер активной страницы. Работу с вкладками реализует встроенный компонент управления TTabSheet.

Компонент TListView отображает поле с иерархическим (древовидным) списком элементов в различных видах. Свойство ViewStyle определяет вид отображения элементов списка: по столбцам с заголовками, вертикально, горизонтально, с малыми или с большими пиктограммами. Свойство Items позволяет добавлять, вычеркивать и модифицировать подписи, а также подбирать пиктограммы для элементов списка. Редактор списка вызывается кнопкой в графе значений этого свойства. Для выбора источника пиктограмм из выпадающего списка свойства LargeImages (SmallImages) задайте значения vsIcon (vsSmallIcon) для свойства ViewStyle. В режиме AutoArrange свойства IconOptions пиктограммы выравниваются в соответствии с выбранным значением свойства Arrangement, а свойство WrapText указывает необходимость переноса текста подписи, когда она не умещается на пиктограмме по ширине.

Компонент TImageList создает контейнер для коллекции из графических изображений одинакового размера K*k, каждое из которых можно выбирать по его индексу в интервале значении от 0 до n-1. Графические коллекции используются для эффективного обслуживания больших наборов битовых образов или пиктограмм, которые хранятся как единый битовый образ шириной k*n. Окно редактора коллекции изображений открывается двойным щелчком мышью по компоненту или опцией ImageList Editor из его контекстного меню.

Компонент THeaderControl создает контейнер для набора заголовков столбцов, ширину которых можно менять в процессе выполнения программы. Заголовки, перечисленные в свойстве Sections, можно размещать над информационными полями, например, над списками

компонентов TListBox. Окно редактора заголовочных секций открывается кнопкой в графе значений этого свойства.

Компонент TRichEdit отображает область редактируемого ввода множества строк информации в формате RTF, который включает различные вариации атрибутов шрифта и форматирования абзацев. Данный формат принимают многие профессиональные текстовые процессоры, например, Microsoft Word. Компонент TRichEdit является прямым производным классом от класса TCustomRichEdit, полностью наследуя его свойства, методы и события.

Компонент TStatusBar создает строку панели состояния (обычно выравниваемую по нижней границе формы) для отображения статусной информации, выдаваемой при работе программы.

Каждая панель представлена в списке свойства Panels. Панели нумеруются слева направо, начиная с индекса 0. Окно редактора панелей открывается кнопкой в графе значений этого свойства. Свойство SimplePanel используется для переключения вида отображения строки состояния (одно- или многопанельная строка состояния).

Компонент TTrackBar создает шкалу с метками и регулятором текущего положения (вариант линейки прокрутки).

Компонент TProgressBar создает индикатор, который отслеживает процесс выполнения некоторой процедуры в программе. По мере выполнения процедуры, прямоугольный индикатор, например, постепенно окрашивается слева направо заданным цветом.

Компонент TUpDown создает спаренные кнопки со стрелками вверх и вниз. Нажатие этих кнопок вызывает, соответственно, увеличение или уменьшение численного значения свойства Position. Данный компонент обычно используется совместно с компонентом TEdit для ввода целых чисел.

Компонент THotKey используется для установки клавиш быстрого вызова (shortcut) во время выполнения программы. Пользователь может ввести комбинацию "горячих" клавиш, обычно состоящую из модификатора (Ctrl, Alt или Shift) и любого символа, включая функциональные клавиши F1,…,F12. Введенную комбинацию, записанную в свойстве HotKey, можно присвоить свойству Shortcut другого компонента. Чтобы выбрать горячие клавиши на стадии проектирования, используйте свойства HotKey и Modifiers, а чтобы отменить их – свойство InvalidKeys. Чтобы изменить комбинацию во время выполнения программы, удерживайте нажатой клавишу модификатора и одновременно введите новый символ.

2.3. Дополнительные компоненты

Компоненты вкладки Additional Палитры компонентов осуществляют включение в программу следующих элементов управления.

Компонент TBitBtn создает кнопку с изображением битового образа.

КомпонентTSpeedButton создает графическую кнопку, обычно располагаемую на панели (TPanel), для быстрого вызова определенных команд меню или установки режимов. Различным состояниям быстрой кнопки (например, "нажата", "отпущена", "запрещена") могут соответствовать разные графические образы. Окно редактора файлов изображений открывается кнопкой в графе значений свойства Glyph.

Компонент TStringGrid создает регулярную сетку для отображения символьных последовательностей по строкам и столбцам.

Во время выполнения программы символьные последовательности и связанные с ними объекты некоторого столбца сетки адресуются свойством Cols. Свойство Rows позволяет подобным образом оперировать со строками сетки. Все символьные последовательности сетки содержатся в свойстве Cells, которое адресует нужную ячейку сетки.

Компонент TDrawGrid создает регулярную сетку для отображения структурированных графических данных по строкам и столбцам. Свойства RowCount и ColCount задают число ячеек сетки по вертикали и по горизонтали. Значение свойства Options позволяет изменить вид сетки (например, с разделительными линиями между столбцами) и ее поведение (например, с переходом от столбца к столбцу по клавише Tab). Ширина разделительных линий сетки задается свойством GridLineWidth, а линейки прокрутки добавляются свойством ScrollBars. Свойства FixedCols и FixedRows позволяют запретить прокрутку столбцов и строк, а свойство FixedColor присваивает определенный цвет всем столбцам и строкам.

Во время работы программы вы можете получить в свое распоряжение область для рисования некоторой ячейки с помощью метода CellRect. Метод MouseToCell возвращает координаты номера столбца и строки ячейки, на которую установлен курсор мыши. Выбранная ячейка сетки становится значением свойства Selection.

Компонент TImage создает на форме контейнер графического изображения (битового образа, пиктограммы или метафайла). Окно редактора файлов изображений открывается кнопкой в графе значений свойства Picture. Чтобы контейнер изменил свои размеры так, чтобы вместить изображение целиком, установите значение true свойства AutoSize. Чтобы исходное изображение меньшего размера растянулось

на весь контейнер, задайте значение true свойства Stretch. Используйте методы LoadFromFile и SaveToFile объектного свойства Picture для динамической загрузки и сохранения файлов изображений с помощью инструкций типа:

Image->Picture->LoadFromFile("<имя файла>");

Image->Picture->SaveToFile("<имя файла>");

Компонент TShape рисует простые геометрические фигуры: окружность и эллипс, квадрат и прямоугольник (можно с закругленными углами).

Компонент TBevel создает линии, боксы (контейнеры) или рамки, которые выглядят объемными. Рисуемый компонентом объект определяется свойством Shape, а значение свойства Style меняет вид объекта, делая его выпуклым или вдавленным. Чтобы сохранить относительное положение объекта неизменным, установите значение true свойства Align.

Компонент TScrollBox создает в окне бокс переменного размера, который автоматически снабжается линейками прокрутки.

2.4. Диалоговые компоненты

Компонент TOpenDialog используется для выбора и открытия файлов. Компонент TSaveDialog предназначен для выбора и сохранения файлов. Компонент TOpenPictureDialog используется для выбора и открытия графических файлов. Компонент TFontDialog высвечивает диалоговое окно для выбора шрифтов. Компонент TColorDialog отображает диалоговое окно для выбора цвета. Компоненты становятся активными после вызовы метода Execute(). Например:

```
if(OpenDialog1->Execute()){
  filename = OpenDialog1->FileName;
};
```

Вопросы

1. Назовите разновидности меню и укажите их отличия.
2. Опишите технологию создания главного меню.
3. Как скопировать нужные разделы из главного меню во всплывающее меню?
4. Как создать всплывающее меню и закрепить его за компонентом?
5. Как ограничить типы вводимых символов в текстовые компоненты?
6. Чем отличается обработка событий OnKeyPress и OnKeyDown?
7. Поясните действие следующего фрагмента программы:

```
for (int i=0; i<ListBox1→Items→Count; i++)
ListBox1→Items→Strings[i]=”строка№”+IntToStr(i);
```

8. Что означает следующий код?

```
Button1->OnClick=0;
Button1->OnClick=Button2Click;
```

Упражнения

1. Ввести и транспонировать квадратную матрицу.
2. Уплотнить заданную матрицу, удаляя из нее строки и столбцы, заполненные нулями.
3. Разработать текстовый редактор.
4. Создать записную книжку.

3. СТРОКИ И ПОТОКИ ВВОДА/ВЫВОДА В C++BUILDER

3.1. Строки

В C++ Builder можно использовать следующие текстовые строки: символьные массивы типа char* языка C с нуль-символом (‘\0’) в конце; класс string из стандартной библиотеки C++; класс String (другое название AnsiString) из библиотеки VCL.

Символьные массивы. В языке C строки представляют собой массив данных символьного типа, заканчивающийся символом ‘\0’. Для работы со строками в стандартную библиотеку языка C включен ряд функций, которые определены в заголовочном файле <string.h>. Например:

strcpy() – копирование одной строки в другую;
strcat() – добавление одной строки в конец другой;
strcmp() – сравнение двух строк в лексикографическом порядке;
strstr() – поиск в строке заданной подстроки;
strupr() – преобразование символов строки к верхнему регистру;
strlen() – определение длины строки без учета нуль-символа.

Использование символьных массивов часто оказывается самым удобным при разработке приложений.

Класс string языка C++. Класс string (название начинается со строчной буквы) представляет собой динамический массив и дает преимущества при выполнении операций соединения и обрезания строк, вычеркивания части строки, поиска комбинаций символов. Данный класс определен в заголовочном файле <cstring.h>.

Класс string содержит несколько конструкторов, из которых приведем объявления следующих:

1) string();
2) string(const string &s);
3) string(const char *cp);
4) string(const char *cp, size_t n);
5) string(const string &s, size_type start, size_type length);
6) string(size_type n, char c);

Первый конструктор создает пустую строку. Второй конструктор является конструктором копирования. Третий конструктор создает строку, инициализированную С-строкой (нуль-символ в объект класса string не копируется). Четвертый конструктор создает строку, инициализированную не более чем *n* первыми символами С-строки. Пятый конструктор создает строку из другой строки, начиная с индекса start и содержащую не более чем length символов. Шестой конструктор создает строку, содержащую *n* символов *c*.

Пример.

```
#include <iostream.h>
#include <string.h>
using namespace std;
void main(){
// Создание пустой строки и ее вывод
string str1;
cout<<”str1: “<<str1<<endl;
// Строка инициализируется С-строкой
string str2(“My string”);
cout<<”str2: “<<str2<<endl;
// Используется конструктор string(const char *cp, size_t n)
string str3(“Tekct”,4);
cout<<”str3: “<<str3<<endl; // Выводится “Tekc”
// Используется конструктор копирования
string str4(str3);
cout<<”str4: “<<str4<<endl; // Выводится “Tekc”
// Используется конструктор string(const string &s, size_type start,
// size_type length)
string str5(str3, 2, 2);
cout<<”str5: “<<str5<<endl; // Выводится “kc”
// Используется конструктор string(size_type n, char c)
string str7(5, ‘A’);
cout<<”str7: “<<str7<<endl; // Выводится “AAAAA”
}
```

Для объектов класса string перегружены следующие операции (операторы):

= – присваивание. Данная операция (как и другие рассматриваемые ниже операции) перегружена с помощью нескольких функций-операторов, что позволяет выполнять присваивание строке класса string не только строку этого же класса, но и С-строку или символ.

+ – конкатенация строк.

== – проверка строк на равенство.

!= – проверка строк на неравенство.

< – меньше. Здесь и далее сравнение строк выполняется в лексикографическом порядке.

<= – меньше или равно.

> – больше.

>= – больше или равно.

[] – индексация. Обеспечивает доступ к элементам строки по индексу. Контроль выхода за пределы строки не выполняется. Отметим, что имеется метод at(int i), который также позволяет обеспечить доступ к символам строки, но он выбрасывает исключение out_of_range, если индекс i выходит за пределы строки. Индексация символов строки ведется с нуля.

+= – добавление к строке с изменением исходной строки.

<< – вывод строки.

>> – ввод строки.

Для сравнения строк класс string содержит также несколько перегруженных версий метода compare(), выполняющего сравнение строк в лексикографическом порядке:

1) int compare(const string &str) const;
2) int compare(size_t pos, size_t n, const string &str) const;
3) int compare(size_t pos1, size_t n1, const string &str, size_t pos2,size_t n2) const;
4) int compare(const const char *str) const;
5) int compare(size_t pos, size_t n, const char *str, size_t length) const;

Здесь size_t является беззнаковым целочисленным типом, определенным в заголовочном файле <stdio.h>:

typedef unsigned size_t;

Метод возвращает отрицательное значение, если вызывающая строка предшествует строке, с которой выполняется сравнение; 0 – если строки совпадают; положительное значение – если вызывающая строка следует за другой строкой, участвующей в сравнении.

Версия 2) метода сравнивает строку str с *n* символами вызывающей строки, начиная с позиции pos. Если значение параметра *n* выходит за пределы вызывающей строки, то строка рассматривается от позиции pos

до ее конца. Если позиция pos выходит за пределы вызывающей строки, то выбрасывается исключение out_of_range.

Версия 3) позволяет выполнить сравнение подстроки вызывающей строки (характеризуется параметрами pos1 – индекс первого элемента строки и n1 – длина подстроки) с подстрокой строки str (характеризуется аналогичными параметрами pos2 и n2). Если значения pos1 или pos2 выходят за пределы соответствующих строк, то выбрасывается исключение out_of_range.

Версии 4) и 5) предназначены для сравнения вызывающей строки или ее подстроки (характеризуется параметрами pos и *n*) с С-строкой или ее начальной подстрокой, содержащей length символов.

Пример.

```
#include <string.h>
#include <iostream.h>
#include <fstream.h>
using namespace std;
void main(){
string str1(“ABCDEFGH”), str2, str3, str4, str5;
// Присваивание. Прототип функции-оператора
// string& operator =(const string& str)
str2=str1;
cout<<”str2:  “<<str2<<endl;
// Присваивание. Прототип функции-оператора
// string& operator =(const char *str)
str3=”a1b2c3d4e5”;
cout<<”str3:  “<<str3<<endl;
// Присваивание. Прототип функции-оператора
// string& operator =(char c)
str4=’R’;
cout<<”str4:  “<<str4<<endl;
// Доступ к символам строки
cout<<”str1[3]: “<<str1[3]<<endl; // Будет выведено str1[3]: D
cout<<”str1.at(2): “<<str1.at(2)<<endl;
// Будет выведено str1.at(2): C
// Ввод из файла. Файл data.txt содержит последовательность
// символов 12345
ifstream inf(“data.txt”);
inf>>str5;
if(!inf){
  cout<<”ERROR”<<endl;
  return;
  }
cout<<”str5:  “<<str5<<endl;
// Добавление к строке
str1+=str5;
```

```
cout<<"str1+=str5:  "<<str1<<endl;
// Будет выведено str1+=str5:  ABCDEFGH12345
 }
```

Методы класса string. Класс string содержит множество методов, которые можно разделить на группы в соответствии с их функциональным назначением: получение характеристик строк; присваивание, добавление и обмен строк; вставка, удаление и замена строк; поиск строк, подстрок и символов в строках. Ниже приводится краткое описание методов. При этом используются следующие обозначения:

size_type – тип, определенный в пространстве имен std и эквивалентный unsigned int;

npos – статический член класса string, содержащий максимальное значение для типа size_type.

size_type length() const – возвращает длину (количество символов) строки.

size_type size() const – то же назначение, что и у метода length().

size_type max_size() const – возвращает максимальный возможный размер строки.

size_type capacity() const – возвращает текущую емкость строки, т. е. количество символов, которые можно поместить в строку без перераспределения памяти. Перераспределение памяти под строки происходит автоматически по мере необходимости. Нужно помнить, что в результате динамического перераспределения памяти становятся недействительными все ссылки, указатели и итераторы (специальный контролируемый тип указателя), ссылающиеся на символы строки. Емкость строки может быть изменена и с помощью метода

reserve(size_type n=0).

bool empty() const – возвращает true, если строка является пустой, в противном случае возвращает false.

string& assign(const string &s) – присваивает строку s вызывающей строке (то же самое, что и операция присваивания).

string& assign(const string &s, size_type pos, size_type n) – присваивает вызывающей строке подстроку строки s, начиная с позиции pos и содержащую не более n символов. Если pos выводит за пределы строки, то выбрасывается исключение out_of_range. Если значение n выводит за пределы строки, то присваивается подстрока, начиная с позиции pos и до конца строки s.

string& assign(const char *s, size_type n) – присваивает вызывающей строке подстроку С-строки s длиной n символов.

string& append(const string &s) – к вызывающей строке добавляется строка s (действие эквивалентно операции +=). Если длина результата превосходит максимальную допустимую длину строки, то выбрасывается исключение length_error.

string& append(const string &s, size_type pos, size_type n) – к вызывающей строке добавляется подстрока строки s, начиная с позиции pos и содержащая не более n символов. Если значения pos или n выводят за пределы строки s, то происходит то же, что и при вызове метода assign().

string& append(const char *s, size_type n) – к вызывающей строке добавляется подстрока С-строки s, содержащая n символов.

swap(string &s) – вызывающая строка и строка s обмениваются содержимым.

string& insert(size_type pos, const string &s) – в вызывающую строку, начиная с позиции pos, вставляется строка s. Если значение pos выводит за пределы вызывающей строки, то выбрасывается исключение out_of_range. Если длина строки после вставки превосходит максимальную допустимую длину строки, то выбрасывается исключение length_error.

string& insert(size_type pos1, const string &s, size_type pos2, size_type n) – вставляет в вызывающую строку, начиная с позиции pos1, часть строки s, начиная с позиции pos2, и содержащую не более n символов. Если значения pos1 или pos2 выводят за пределы соответствующих строк, то выбрасывается исключение out_of_range. Если значение n выводит за пределы строки s, то вставляется часть строки s, начиная с позиции pos2 и до конца строки. Если длина строки-результата превосходит максимальную допустимую длину строки, то выбрасывается исключение length_error.

string& insert(size_type pos, const chat *s, size_type n) – вставляет в вызывающую строку, начиная с позиции pos, n первых символов С-строки s. Могут выбрасываться исключения, которые рассматривались при описании других вариантов метода insert().

string& erase(size_type pos=0, size_type n=npos) – удаляет из вызывающей строки n символов, начиная с позиции pos. Если n не указано, то удаляется остаток строки, начиная с позиции pos.

string& replace(size_type pos, size_type n, const string &s) – заменяет часть вызывающей строки, начиная с позиции pos, длиной n символов, строкой s. Если значение pos выводит за пределы вызывающей строки, то выбрасывается исключение out_of_range. Если длина строки-результата превосходит максимальный допустимый размер строки, то выбрасывается исключение length_error.

string& replace(size_type pos1, size_type n1, const string &s, size_type pos2, size_type n2) – заменяет часть вызывающей строки, начиная с позиции pos1 длиной n1 символов, подстрокой строки s, начинающейся с позиции pos2 и длиной n2 символов. Может выбрасывать исключения того же типа, что и предыдущий вариант метода.

string& replace(size_type pos, size_type n1, const char *s, size_type n2) – заменяет часть вызывающей строки, начиная с позиции pos длиной n1 символов, первыми n2 символами С-строки s. Может выбрасывать те же исключения.

clear() – выполняет очистку строки, т. е. строка имеет нулевую длину.

size_type find(const string &s, size_type pos=0) const – находит самое левое вхождение строки s в вызывающую строку, начиная с позиции pos вызывающей строки. Метод возвращает позицию начала строки s в вызывающей строке или npos, если вхождение не найдено.

size_type find(char c, size_type pos=0) const – находит самое левое вхождение символа с в вызывающую строку, начиная с позиции pos.

size_type find(const char *s, size_type pos=0) const – находит самое левое вхождение С-строки s в вызывающую строку, начиная с позиции pos вызывающей строки.

size_type find(const char * s, size_type pos, size_type n) const – находит самое левое вхождение подстроки длиной n символов С-строки s в вызывающую строку, начиная с позиции pos вызывающей строки.

size_type rfind(const string &s, size_type pos=0) const – находит самое правое вхождение строки s в вызывающую строку. Вызывающая строка сканируется справа налево, начиная с конца и заканчивая позицией pos.

size_type rfind(char c, size_type pos=0) const – находит самое правое вхождение символа с в подстроку вызывающей строки, начинающуюся с позиции pos.

size_type rfind(const char *s, size_type pos=0) const – находит самое правое вхождение С-строки s в подстроку вызывающей строки, начинающуюся с позиции pos.

size_type rfind(const char *s, size_type pos, size_type n) const – находит самое правое вхождение подстроки длиной n символов С-строки s в подстроку вызывающей строки, начинающуюся с позиции pos вызывающей строки.

string substr(size_type pos=0, size_type n=npos) const – возвращает подстроку вызывающей строки, начинающуюся с позиции pos и содержащую не более n символов. Метод может выбрасывать исключения рассмотренных выше типов.

const char* c_str()const – возвращает указатель на константную С-строку, эквивалентную вызывающей строке. Указатель, который ссылается на полученную константную строку, может стать некорректным после выполнения любой неконстантной операции с исходной строкой.

size_type copy(char *s, size_type n, size_type pos=0) const – копирует не более n символов в символьный массив s, начиная с позиции pos вызывающей строки. Нуль-символ в массив s не заносится. Метод возвращает количество скопированных символов. Могут выбрасываться исключения перечисленных выше типов.

const char* data() const – аналогичен методу c_str(), но нуль-символ в конец возвращаемого символьного массива не помещается.

Кроме рассмотренных выше методов класс string включает также методы для поиска самого левого вхождения любого символа заданной строки в другую строку, методы для поиска самого правого вхождения любого символа заданной строки в другую строку, методы для поиска самого левого символа заданной строки, который не входит в другую строку, методы для поиска самого правого символа заданной строки, который не входит в другую строку. Функциональное назначение данных методов понятно из их названий, а работают они аналогично уже рассмотренным методам поиска. В связи с этим их подробное описание здесь не приводится.

Пример.

```
#include <string.h>
#include <iostream.h>
using namespace std;
void main(){
string str1(“ABCDEFGH”),str2(“”),str3;
char s[]=”Programming”;
cout<<”length of str1: “<<str1.length()<<endl;
// Выводится length of str1: 8
cout<<”size of str1: “<<str1.size()<<endl;
// Выводится size of str1: 8
if(str1.empty())
  cout<<”The string str1 is empty”<<endl;
else
  cout<<”The string str1 is not empty”<<endl;
// Выводится  The string str1 is not empty
if(str2.empty())
  cout<<”The string str2 is empty”<<endl;
else
  cout<<”The string str2 is not empty”<<endl;
// Выводится  The string str2 is empty
str3.assign(str1);
```

```
cout<<”str3:  “<<str3<<endl;
// Выводится str3: ABCDEFGH
str3.assign(str1,2,4);
cout<<”str3: “<<str3<<endl;
// Выводится str3: CDEF
str3.assign(s,7);
cout<<”str3: “<<str3<<endl;
// Выводится  str3: Program
str3.append(“ming”,4);
cout<<”str3: “<<str3<<endl;
// Выводится str3: Programming
str3.append(str1);
cout<<”str3: “<<str3<<endl;
// Выводится str3: ProgrammingABCDEFGH
str1.append(str3,3,5);
cout<<”str1: “<<str1<<endl;
// Выводится str1: ABCDEFGHgramm
str1.swap(str3);
cout<<”str1: “<<str1<<endl;
// Выводится str1: ProgrammingABCDEFGH
cout<<”str3: “<<str3<<endl;
// Выводится str3: ABCDEFGHgramm
str1.erase(11);
cout<<”str1: “<<str1<<endl;
// Выводится str1: Programming
str3.erase(8,5);
cout<<”str3: “<<str3<<endl;
// Выводится str3: ABCDEFGH
str3.insert(2,str1);
cout<<”str3: “<<str3<<endl;
// Выводится str3: ABProgrammingCDEFGH
str1.insert(10,str3,1,2);
cout<<”str1: “<<str1<<endl;
// Выводится str1: ProgramminBPg
str1.replace(10,3,str3,5,1);
cout<<”str1: “<<str1<<endl;
// Выводится str1: Programming
str3.replace(2,13,”CD”,2);
cout<<”str3: “<<str3<<endl;
// Выводится str3: ABCDEFGH
str1.insert(2,str3);
str1.insert(13,str3);
cout<<”str1: “<<str1<<endl;
// Выводится str1: PrABCDEFGHogrABCDEFGHamming
string::size_type p;
p=str1.find(str3);
cout<<”p= “<<p<<endl;
```

```
// Выводится p= 2
p=str1.find(str3,3);
cout<<"p= "<<p<<endl;
// Выводится p= 13
p=str1.find('A');
cout<<"p= "<<p<<endl;
// Выводится p= 2
p=str1.find("CDE");
cout<<"p= "<<p<<endl;
// Выводится p= 4
p=str1.rfind(str3);
cout<<"p= "<<p<<endl;
// Выводится p= 13
p=str1.rfind('A');
cout<<"p= "<<p<<endl;
// Выводится p= 13
p=str1.rfind("AB");
cout<<"p= "<<p<<endl;
// Выводится p= 13
str2=str3.substr(2,4);
cout<<"str2: "<<str2<<endl;
// Выводится str2: CDEF
cout<<"C-string for str3: "<<str3.c_str()<<endl;
// Выводится C-string for str3: ABCDEFGH
}
```

Следующий пример также иллюстрирует применение класса string в C++Builder. Строка отображается в компонентах типа строки редактирования Edit1 и Edit2. Процедура запускается кнопкой Button1.

```
#include <cstring.h>
#include <string.h>
// Обработчик события, возникающего при нажатии
// кнопки "Show"
void __fastcall TForm1::ButtonlClick(TObject *Sender)
{
// Создание экземпляра класса string
string str;
str = "C++ ";
str += "is the best..";
Edit1->Text = str.c_str();// Преобразование к С-строке
// Объявление символьного массива языка С
char buff[30];
strcpy(buff, "Builder C++ ");
strcat(buff, "is the best too.");
int len = strlen(buff);
Edit2->Text = buff;
if (len >= 30)
```

```
ShowMessage("Переполнение буфера!!!");
}
```

Метод c_str () возвращает указатель на символьный массив, содержащий копию строки из объекта str. Попытка присваивания значения объекта str свойству Text объекта Edit2 выдала бы ошибку компиляции. Отметим, что при создании экземпляра str не пришлось указывать предельный размер строки. Класс string динамически отводит дополнительную память при необходимости.

В следующем примере для выбора файла открывается диалоговое окно класса TOpenDialog при нажатии кнопки Button2. Имя файла отображаетсяся в объекте Memo1 главной формы Form1. Имя файла затем используется для открытия выбранного файла, ввода строки и ее вывода в объект Memo1.

```
#include <fstream.h>
#include <cstring.h>
void __fastcall TForm1::Button2Click(TObject *Sender){
string str, str1; // Строки будут содержать путь и имя файла
char buf[80];
if (OpenDialog1->Execute())
{
 Caption = OpenDialog1->FileName; // полный путь и имя файла
 str = OpenDialog1->FileName.c_str();
}
str1=str;
str.insert(0,"Имя файла: ") ;
Memo1->Lines->Add(str.c_str());
// Вывести в Memo1 путь и имя файла
ifstream in(str1.c_str());
in>>buf;
Memo1->Lines->Add(buf); // Вывести в Memo1 введенную строку
}
```

Класс String из библиотеки VCL. При работе с компонентами VCL часто встречается тип AnsiString для описания строк произвольной длины. Из-за того, что название AnsiString не кажется привлекательным, пользователям C++Builder рекомендуется его синоним – класс String (название начинается с заглавной буквы) Действительно, в файле sysdefs.h, который автоматически включается в заголовки разрабатываемых приложений, можно найти определение тождественности типов

typedef AnsiString String;

Конструкторы класса String позволяют создавать строчные объекты из переменных типа char*, int или даже double. Например:

String str = "Текстовая строка";

Класс String имеет ряд переопределенных операторов присваивания (=), конкатенации (+), сравнения, характерных для строчных классов вообще. При сравнении разнотипных объектов неявное преобразование типа выполняется автоматически:

```
char* buff = "Текстовая строка";
String str = "Текстовая строка";
if (str == buff) {…} // если строки одинаковые
```

Класс String инкапсулирует набор методов, которые позволяют реализовать операции над строками. Наиболее часто используются следующие методы:

c_str() – возвращает указатель char* на С-строку, содержащую копию исходной строки.

Length() – возвращает длину строки (число символов в строке).

IsEmpty() – возвращает значение true, если длина строки равна нулю.

Insert() – вставляет текст в указанную позицию строки.

Delete() – удаляет заданное число символов из строки.

Pos() – возвращает индекс начала заданной комбинации символов в строке.

LastDelimiter() – возвращает индекс последнего вхождения в строку заданного символа-ограничителя.

LowerCase() – возвращает измененную исходную строку, символы которой заменяются строчными буквами (приводятся к нижнему регистру).

UpperCase() – возвращает измененную исходную строку, символы которой приводятся к верхнему регистру.

Trim() – возвращает измененную исходную строку, в которой удалены управляющие символы, а также пробелы в начале и конце строки.

Format() – форматирование строки похожее на работу функции sprintf().

ToInt() – преобразование строки в целое число.

ToDouble() – преобразование строки в число с плавающей точкой.

SubString() – создает строку, равную некоторой подстроке исходной строки.

Существуют два способа преобразования цифровых строк, вводимых в окна редактирования, в числа:

```
String str;
str = Editl->Text;
int vail = atoi(str.c_str()); // с помощью функции atoi()
int val2 = str.ToInt(); // с помощью метода ToInt()
```

Отметим особенность класса String: символы строки индексируются начиная с 1 (как в Паскале), а не с 0 (как в С++).

Пример. Создается приложение, позволяющее выполнять сложение двух чисел.

Будем использовать компоненты TEdit, TLabel и TButton.

Используемые свойства и методы:

TEdit: Name (имя объекта), ReadOnly (режим "только чтение"), Text (строка типа String), GetTextLen (длина строки), GetTextBuf (преобразование свойства Text в С-строку).

Используемые свойства и события компонента TButton: Caption (надпись на кнопке), OnClick (событие, действия проводимые при нажатии на кнопку).

Используемое свойство компонента TLabel: Caption (надпись).

Ниже приводится обработчик события нажатия на кнопку:

```
//Unit1.cpp: ----------------------------------------------------
void __fastcall TForm1::Button1Click(TObject *Sender){
 int Size = Edit1->GetTextLen() + 1;
 char *Buffer = new char[Size];
 Edit1->GetTextBuf(Buffer,Size);
 for(int i = 0; i < Size-1; i++)
 if(!isdigit(Buffer[i])&&Buffer[i]!='.') {
  Application->MessageBox("Вводите только цифры и точку",
 "Ошибка!", MB_OK);
  Edit1 -> Text = “”;
  return; }
 double n = atof(Buffer);
 delete[] Buffer;
// Выбрасывается исключение, если в Edit2 введено не число
 try
 {  double m;
 m= Edit2->Text.ToDouble();
 Edit3->Text = n+m; }
catch (...){
  Application->MessageBox("Ошибка при вводе второго числа",
 "Ошибка!", MB_OK);
  Edit2 -> Text = “”;
  return; }
}
```

На рис. 15 показана форма приложения в процессе выполнения.

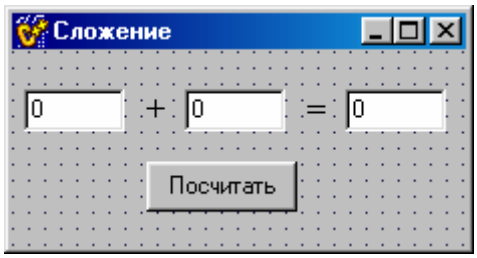

Рис. 15. Результат запуска приложения

Работа со строками в текстовых компонентах. Компонент TLabel позволяет создать надпись:

Label1->Font->Color=clRed;

Label1->Caption="Это красная метка";

Компонент TEdit представляет собой окно для ввода/вывода строки:

Edit2->Text=Edit1->Text;

Компонент TMemo используется для ввода и вывода нескольких строк:

Memo1->Lines->Strings[i]; // доступ к i-й строке

Свойство Lines содержит ряд методов: Add(), Append() – добавить новую строку; Delete(n) – удалить строку с номером n; SaveToFile("имя файла") – сохранить в файле, LoadFromFile("имя файла") – загрузить из файла. Например:

Memo1->Lines->Add(Edit1->Text);

Списки – являются производными от двух основных классов: TStringList – строчный список, TList – список объектов общего назначения. Для отображения списков используются компоненты TListBox, TComboBox и другие.

Компонент TListBox – список строк с линейной прокруткой.

Пример.

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
TStringList *MyList=new TStringList;
ListBoxl->Clear(); // очистить список
ListBox1->Items->Add("Первый элемент");
// Свойство Items содержит метод Add() и другие методы
MyList->Add(ListBox1->Items[ListBox1->ItemIndex+1].GetText());
MyList->Add("Последняя строка");
ListBox1->Items->AddStrings(MyList);
}
```

Контейнерные списочные классы TStringList и TList решают проблему динамического распределения памяти. Хотя списочные классы TStringList и TList не являются прямыми потомками одного предка, однако они имеют много общих методов: Add() – добавить элемент в конец

списка; Clear() – очистить список; Delete() – удалить элемент с указанным номером; Exchange() – поменять два элемента местами; Insert() – вставить элемент в указанную позицию; Move() – переместить элемент из одной позиции в другую; Sort() – отсортировать список. Важным дополнением к перечисленным методам является свойство Count, содержащее текущее число элементов в списке.

Класс TStringList является потомком класса TStrings, наследующим свойства и методы родителя. Однако TStrings это абстрактный базовый класс, который не способен производить объекты.

Ниже приводится пример инициализации списка типа TStringList в конструкторе главной формы:

```
void _fastcall TForml::TForml(TComponent* Owner) : TForm(Owner)
{ TStringList * MyList;
MyList = new TStringList; // создать объект
MyList->Add("Раз");
MyList->Add("Два");
MyList->Add("Три");
MyList->Duplicates = dupIgnore; // игнорировать дубликаты
MyList->Sorted = true; // сортировка списка
ListBox1->Items->AddStrings(MyList); // отображение списка
}
```

Метод Add() добавляет в конец списка строку вместе с указателем на сопровождающий объект. Метод AddStrings() добавляет в конец списка группу строк, взятых из другого списочного объекта. Метод Find() выдает номер строки в отсортированном списке и возвращает значение false, если такой строки нет в списке. Метод IndexOf() возвращает номер строки, под которым она впервые встретилась в списке (как отсортированном, так и нет). Метод LoadFromFile() загружает строки списка из текстового файла. Метод SaveToFile() сохраняет список в текстовом файле

Для хранение объектов используется другой списочный класс TList. Ниже приводится пример использования класса TList для хранения различных объектов (модуль Unit2.cpp). Конструктор главной формы создает список MyList, а затем заносит в него названия файлов, содержащих значки выполняемых программ, и сами значки.

```
TList* MyList; // объявление в файле Unit2.h
// Инициализация списка конструктором главной формы
__fastcall TForm2::TForm2(TComponent* Owner) : TForm(Owner)
{
MyList = new TStringList; // создать объект MyList
// Загрузить файл имен и отобразить их в компоненте ListBox2
ListBox2->Items->LoadFromFile("Names.txt") ;
// Перебрать имена программ и сформировать список MyList
```

```
for (int i=0; i<ListBox2->Items->Count; i++)
{
 String str = ListBox2->Items->Strings[i];
int pos=str.LastDelimiter(“.”); // найти символ ‘.’
str.Delete(pos,4); // стереть расширение файла .exe
str.Insert(“.ico”,pos); // заменить расширение файла на .ico
TIcon* icon = new TIcon; // создать новый объект icon
icon->LoadFromFile(str); // загрузить объект icon
MyList->AddObject(str, icon); // добавить str, icon в список
}
}
```

Элементы списка нумеруются, начиная с 0, и адресуются посредством индексируемого свойства Strings[i]. Например, заголовку формы можно присвоить название, взятое из второй строки списка следующим образом:

```
Form1->Caption = MyList->Strings[2];
```

В ряде случаев удобно интерпретировать строчный список как единую строку текста. Свойство Text содержит все строки списка:

```
Form1->Caption = MyList->Text;
```

Далее приводится реакция на событие выбора строки из списка ListBox2:

```
void__fastcall TForm2::ListBox2Click(TObject *Sender)
int i = ListBox2->ItemIndex; // номер строки
//Сдвинуть изображение и вывести пиктограмму
Label2->Top=8+i*16; // сдвинуть изображение стрелки
Image2->Top = i*16; // сдвинуть изображение значка
Image2->Picture->Icon =
dynamic_cast<TIcon*> (MyList->Objects [i] ) ;
// Свойство CommaText содержит все строки списка,
// разделенные запятыми:
Form1->Caption = MyList->CommaText;
}
```

Свойство CommaText преобразует список к формату SDF (System Data Format), обеспечивая возможность импорта данных из внешней программы непосредственно в контейнерный объект VCL. При этом любая строка, содержащая пробелы или запятые, заключается в двойные кавычки.

Листинг кодового модуля Unit2.cpp, приведенный выше, иллюстрирует применение метода AddObject() на примере списка, в котором имена выполняемых программ файла Names.txt ассоциируются с их пиктограммами (объектами класса TIcon). Конструктор главной формы создает список MyList, объявленный в интерфейсном модуле Unit2.h, а затем заносит в него названия и изображения пиктограмм. Упомянутое ранее ин-

дексируемое свойство Objects содержит типизированный указатель TObject*, поэтому при выборе пиктограммы оператор dynamic_cast преобразует ее тип обратно к TIcon*.

Наиболее характерное свойство списка TList это Capacity, которое содержит максимальную текущую емкость контейнера. Это значение не является пределом, список всегда можно при необходимости расширять и далее. Преимущества этого свойства появляются, когда списки становятся слишком объемными. Дело в том, что алгоритм перераспределения памяти под списочные классы VCL начинает работать медленно при больших размерах контейнера. Если значение свойства Capacity определить заранее, сразу же после создания списка, перед добавлением в него новых элементов, то этот алгоритм вообще не включается до тех пор, пока заданная емкость контейнера не будет исчерпана. Если предполагается хранить большое количество объектов в списке, то его емкость следует определить сразу же после создания:

```
TList* MyList = new TList;
MyList->Capacity = 1000;
MyList->Add(new TMyObject); // добавить объект
```

Рассмотрим, как можно извлекать объекты из списка. Для динамического преобразования указателя типа void* обратно к типу TMyObject* (в нашем примере) можно использовать оператор

```
dynamic_cast<TMyObject*> (MyList->Items [i]) ;
```

или применить статическое преобразование типа в стиле языка C:

```
(TMyObject*)MyList->Items[i] ;
```

Класс TList предназначен для хранения данных любого типа. Адресный указатель void* на объект любого типа передается как параметр таким методам, как Add() и Insert(). Класс TList инкапсулирует ряд специализированных методов: First() – возвращает первый указатель списка (элемент индексируемого свойства Items[0]); Last() – возвращает последний указатель списка (элемент индексируемого свойства Items[Count]); Expand() – увеличивает размер списка, когда его текущая емкость Count достигла значения свойства Capacity

3.2. Файлы и потоки

C++Builder предоставляет разработчикам несколько способов организации ввода-вывода файлов:

- в стиле языка C;
- поточные классы стандартного C++;
- класс TFileStream из библиотеки VCL.

Поточные классы и потоки C++. Классы ifstream и ofstream, используемые для чтения и записи в потоки, являются производными от базового класса ios. Класс fstream способен одновременно читать и писать информацию в поток. Под потоками понимают объекты поточных классов. Для использования классов необходимо включить в заголовок модуля файл fstream.h.

Открытие файла. Потоки открываются с помощью конструктора или с помощью метода open().

Режим открытия файлов задают флаги-члены данных перечисляемого типа класса ios:

```
enum open_mode { app, binary, in, out, trunc, ate };
```

В табл. 2 приведены возможные значения флагов.

Таблица 2

Режим	Назначение
in	Открыть для ввода (по умолчанию для ifstream)
out	Открыть для вывода (по умолчанию для ofstream)
binary	Открыть файл в бинарном режиме
app	Присоединять данные; запись в конец файла
ate	Установить указатель позиционирования на конец файла
trunc	Уничтожить содержимое, если файл существует (выбирается по умолчанию, если флаг out указан, а флаги ate и app – нет)

Например:

```
ifstream file;
file.open("test.dat", ios::in | ios::binary);
ofstream file;
file.open("test.txt", ios:out | ios::app);
```

Операторы включения и извлечения. Оператор поточного включения (<<) записывает данные в поток. Например:

```
ofstream file("temp.txt");
char buff[] = "Текстовый массив ";
int v = 100;
file << buff << endl << v << endl;
file.close();
```

В результате образуются две строки в текстовом файле temp. txt:

Текстовый массив
100

Оператор поточного извлечения (>>) производит обратные действия. Из-за ограниченности оператора извлечения для ввода могут быть использованы другие методы, например getline().

Класс ifstream: чтение файлов. Класс ifstream предназначен для ввода из файлового потока. В табл. 3 перечислены основные методы класса.

Таблица 3

Метод	Описание
open	Открывает файл для чтения
get	Читает один или более символов из файла
getline	Читает символьную строку из текстового файла или данные из бинарного файла до определенного ограничителя
read	Считывает заданное число байтов из файла в память
eof	Возвращает ненулевое значение (true), когда указатель потока достигает конца файла
peek	Выдает очередной символ потока, но не выбирает его
seekg	Перемещает указатель позиционирования файла в указанное положение
tellg	Возвращает текущее значение указателя позиционирования
close	Закрывает файл

Класс ofstream: запись файлов. Класс ofstream предназначен для вывода данных в файлы. В табл. 4 перечислены основные методы класса:

Таблица 4

Метод	Описание
open	Открывает файл для записи
put	Записывает одиночный символ в файл
write	Записывает заданное число байтов из памяти в файл
seekp	Перемещает указатель позиционирования в заданное положение
tellp	Возвращает текущее значение указателя файла
close	Закрывает файл

Бинарные файлы. Бинарные файлы представляют собой последовательность байтов, которая вводится или выводится без каких-либо преобразований.

Для записи бинарных файлов или чтения из них используются методы write() и read(). Первым параметром методов является адрес блока записи/чтения, который должен иметь тип символьного указателя char*. Второй параметр указывает размер блока.

Приведем пример приложения для создания и отображения данных записной книжки. Записи файла считываются и отображаются в строках объекта Memo1.

```
#include <fstream.h>
struct Notes // структура данных записной книжки
{ char Name[60]; // Ф.И.О.
char Phone[16]; // телефон
int Age; };   // возраст
```

```
// Обработчик события, возникающего при нажатии кнопки
void __fastcall TForm1::Button1Click(TObject *Sender)
{Notes Notel =
{"Иванов Иван Васильевич", "не установлен", 60};
Notes Note2 =
{"Балаганов Шура ", "095-111-2233 ", 30};
ofstream ofile("Notebook.dat", ios::binary);
ofile.write((char*)&Notel, sizeof(Notes)); // 1-й блок
ofile.write((char*)&Note2, sizeof(Notes)); // 2-й блок
ofile.close (); // закрыть записанный файл
ifstream ifile("Notebook.dat", ios::binary) ;
Notes Note; // структурированная переменная
char str[80]; // статический буфер строки
Memo1->Clear (); // очистить объект Memo2
//Считывать и отображать строки в цикле, пока не достигнут
// конец файла
while (!ifile.read((char*)&Note, sizeof(Notes)).eof())
{ sprintf(str, "Имя %s Тел: %s\t Возраст: %d",
Note.Name, Note.Phone, Note.Age);
Memo1->Lines->Add(str); }
ifile.close (); // закрыть прочитанный файл
}
```

В результате выполнения этого кода образуется бинарный файл Notebook. dat из блоков размером 80 байтов каждый:

Иванов Иван Васильевич Тел: не установлен Возраст: 60

Балаганов Шура Тел: 095-111-2233 Возраст: 30

Класс ввода и вывода fstream: произвольный доступ к файлу. Предположим, что мы хотим считать пятидесятую запись из книжки. Очевидное решение – установить указатель файла pos прямо на нужную запись и считать ее:

```
ifstream ifile ("Notebook.dat", ios : :binary) ;
int pos = 49 * sizeof(Notes);
ifile.seekg(pos) ; // поиск 50-й записи
Notes Note;
ifile.read((char*)&Note, sizeof(Notes));
```

Чтобы заменить содержимое произвольной записи, надо открыть поток вывода в режиме модификации:

```
ofstream ofile("Notebook.dat", ios::binary | ios::ate);
int pos = 2 * sizeof(Notes);
ofile.seekp(pos); // поиск 3-й записи
Notes Note=
{"Иванов Иван Николаевич", "095-222-3322", 64};
ofile.write((char*)&Note, sizeof(Notes)); // замена
```

Если не указать флаг ios::ate (или ios::app), то при открытии бинарного файла Notebook.dat его предыдущее содержимое будет стерто. Можно

открыть файл по одновременному чтению/записи, указав флаги ios::in | ios::out.

TFileStream: поточный класс VCL. Основные принципы потоков ввода-вывода C++ используются и в классе TFileStream. В табл. 5 перечислены основные свойства и методы класса TFileStream.

Конструктор предназначен для открытия файла. Методы Read() и Write() идентичны методам read() и write(). Свойство Position реализует те же действия, что и методы tellg() и seekg(), делая процедуру поиска простой. Класс TFileStream лишен метода, аналогичного getline(). В результате оказывается, что класс TFileStream плохо приспособлен к построчному чтению текстовых файлов. Некоторым оправданием этого недостатка являются методы загрузки LoadFromFile() и сохранения SaveToFile() файла, применимые ко многим компонентам библиотеки VCL (TMemo, TListBox, TComboBox, TTreeView и др.).

Таблица 5

Свойство	Описание
Position	Текущее значение указателя позиционирования файла
Size	Размер данных файла; свойство доступно только для чтения
Метод	Описание
Конструктор TfileStream	Открывает файл в указанном режиме (по созданию, чтению, записи или по чтению/записи одновременно)
CopyFrom	Копирует заданное число байтов из некоторого потока в данный
Read	Считывает заданное число байтов из файла в память
Write	Записывает заданное число байтов из памяти в файл
Seek	Перемещает указатель позиционирования в указанное положение относительно начала, конца или текущего положения файла

Режим открытия файла. По сравнению с флагами аргумента open_mode установки режима Mode в классе TFileStream заметно отличаются, что видно из табл. 6.

Режим передается конструктору объекта файлового потока в качестве второго параметра следом за именем файла. Например, так следует открывать файл для чтения:

```
TFileStream* fs = new TFileStream("Notebook.dat",fmOpenRead);
```

а так – для модификации:

```
TFileStream* fs = new TFileStream("Notebook.dat",
fmOpenReadWrite);
```

Таблица 6

Режим	Описание
fmCreate	Создать новый файл. Если файл с указанным именем существует, то открыть его для записи
fmOpenRead	Открыть файл только для чтения
fmOpenWrite	Открыть файл только для записи с полной заменой текущего содержания
fmOpenReadWrite	Открыть файл для чтения/записи (модификации)
fmShareExclusive	Другие приложения не могут открыть этот файл (исключительное пользование)
fmShareDenyNone	Не делается никаких попыток предотвратить чтение/запись этого файла из других приложений (полное разделение)
fmShareDenyWrite	Другие приложения могут открыть и читать этот файл параллельно с вашим заданием (разделение только по чтению)
fmShareDenyRead	Другие приложения могут открыть и записывать в этот файл параллельно с вашим заданием (разделение только по записи)

Ниже приводится пример использования класса TFileStream для работы с файлом, содержащим записную книжку (выше данная задача решалась с использованием потоков C++):

```
TForm4 *Form4;
struct Notes // структура данных записной книжки
{ char Name[60]; // Ф.И.О.
char Phone[16]; // телефон
int Age;}; // возраст
// Обработчик события, возникающего при нажатии кнопки
 void __fastcall TForm4 ::Button4Click(TObject *Sender)
{ Notes Notel, Note3;
// Конструктор TFileStream открывает файл для чтения/записи
TFileStream* fs = new TFileStream("Notebook.dat ",
 fmOpenReadWrite);
fs->Position = 2*sizeof(Notes);
fs->Read(&Note3, sizeof(Notes)); // найти и считать Note3
fs->Position = 0;
fs->Read(&Notel, sizeof(Notes)); // найти и считать Notel
fs->Position = 0;
fs->Write(&Note3, sizeof(Notes));
// Note3 переносится на место Notel
fs->Position = 2*sizeof(Notes);
fs->Write(&Notel, sizeof(Notes));
// Note1 переносится на место Note3
char str[80]; // статический буфер строки
Memo4->Clear(); // очистить объект Memo4
// Считывать и отображать все записи в объекте Мето4
```

```
fs->Position = 0; // вернуться к началу файла
for (unsigned i=0; i<fs->Size/sizeof(Notes); i++)
{ fs->Read(&Notel, sizeof(Notes));
sprintf(str, "%s\tТел: %s\tBo3pacT: %d", Notel.Name, Note1.Phone,
 Note1.Age);
Memo4->Lines->Add(str);
}
delete fs; // вызов деструктора
}
```

Свойство Size поточного объекта fs позволяет вычислить число записей в файле Notebook. dat.

Пример. Создание простейшего текстового редактора с возможностью открытия и сохранения текстовых файлов.

Используются компоненты: TMainMemu. TOpenDialog, TSaveDialog. На рис. 16 и 17 показана форма в процессе проектирования приложения.

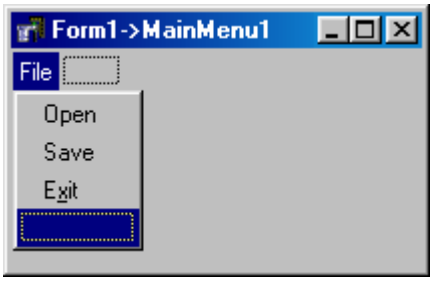

Рис. 16. Создание меню

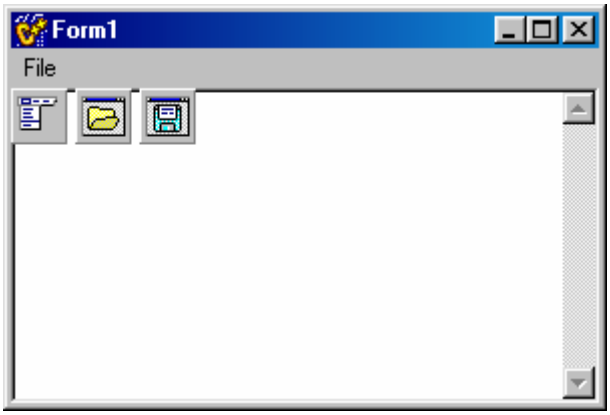

Рис. 17. Форма в процессе проектирования

```
void __fastcall TForm1::Exit1Click(TObject *Sender)
{ exit(0); }
void __fastcall TForm1::Open1Click(TObject *Sender)
{  OpenDialog1->Options.Clear();
 OpenDialog1->Options << ofFileMustExist;
 OpenDialog1->Filter = "Text files (*.txt)|*.txt|All files (*.*)|*.*";
 OpenDialog1->FilterIndex = 2;
 if (OpenDialog1->Execute())
```

```
 Memo1->Lines->LoadFromFile(OpenDialog1->FileName);
}
void __fastcall TForm1::Save1Click(TObject *Sender)
{  SaveDialog1->Options.Clear();
  SaveDialog1->Options << ofFileMustExist;
  SaveDialog1->Filter = "Text files (*.txt)|*.txt|All files (*.*)|*.*";
  SaveDialog1->FilterIndex = 2;
  if (SaveDialog1->Execute())
 Memo1->Lines->SaveToFile(SaveDialog1->FileName);
}
```

Вопросы

1. Какие типы строк можно использовать в C++Builder?

2. Какие функции используются для преобразования чисел в строки типа AnsiString и обратно?

3. Что означает следующий код:

```
AnsiString a = "Hello World";
char *b = a.c_str();
```

4. Какая из строк преобразования является правильной?

```
AnsiString a = "Hello World";
int i = a.ToInt();
```

или

```
int ni = StrToInt(a);
```

5. Что означает следующий код?

```
AnsiString a = "12.5";
float d;
d=a.ToDouble();
```

6. Что означает следующий код?

```
char Arr[240]= "Hello World";
AnsiString Str;
Str=(AnsiString)Arr;
```

7. Какие средства можно использовать в C++Builder для работы с файлами?

8. Охарактеризуйте возможности и основные методы класса TFileStream.

9. Чем отличаются текстовый и двоичный режимы работы с файлами?

10. Каково назначение компонента MaskEdit? Вместо какого компонента он используется?

11. Как работают методы Add и Insert в многострочных текстовых компонентах?

12. С помощью каких методов можно выполнить сохранение строк списка в текстовом файле на диске и последующее чтение списка строк из файла?

13. Назовите отличительные особенности компонента RichEdit.

14. Как извлекать строки из компонента Memo и организовать поток ввода из строк для этих строк?

15. Как выполнить фильтрацию файлов при использовании компонентов OpenDialog и SaveDialog?

16. Чем комбинированный список отличается от простого списка?

17. Что означает следующий код?

```
#include <clipbrd.hpp>
void __fastcall TForm1::Button1Click(TObject *Sender)
{
AnsiString s = "Hello, World !";
Clipboard()->AsText = s;
if (Clipboard()->HasFormat(CF_TEXT))
    Edit1->Text = Clipboard()->AsText;
    else
    application->MessageBox("The clipboard does not contain text.", NULL, MB_OK);
}
```

18. Что означает следующий код?

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
for(int i = 0; i < ListBox1->Items->Count; i++)
if(ListBox1->Selected[i])
ShowMessage(ListBox1->Items->Strings[i]);
}
```

Упражнения

1. В заданном тексте выполнить замену заданного слова (если слово встречается несколько раз, то замену выполнить каждый раз) другим словом, также заданным.

2. Удалить в тексте строки, содержащие заданное слово.

3. Текст представляет собой сведения об успеваемости студентов. Строка содержит фамилию студента и результаты сдачи экзаменов. Упорядочить строки в соответствии с убыванием среднего балла.

5. Файл содержит последовательность чисел. Переставить числа в обратном порядке.

6. Файл содержит последовательность чисел. Упорядочить последовательность чисел в файле, считывая в память не более двух чисел.

4. ПОДДЕРЖКА ГРАФИКИ И ГРАФИЧЕСКИЕ КОМПОНЕНТЫ

Система Windows предоставляет средства рисования GDI (Graphics Device Interface) для построения графических изображений на графическом контексте независимо от типа устройства вывода. При прямом вызове функций GDI им необходимо передавать *дескриптор контекста устройства (HDC – device context handle)*, который задает инструменты рисования. Графический контекст представляет модель графического устройства. После завершения работы с изображениями, необходимо восстановить контекст устройства в исходное состояние и освободить его. C++Builder берет на себя работу GDI, связанную с поиском дескрипторов изображений и ресурсов памяти. Разрешается прямое обращение приложений к функциям Windows GDI. Рассмотрим пример:

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{ HDC hdc =Canvas->Handle;
LineTo(hdc,100,100);
MessageBox(Form1->Handle,"Вызов Windows API","",MB_OK);
}
```

C++Builder предоставляет простой интерфейс посредством свойства Canvas графических компонентов. Это свойство инициализирует и освобождает контекст устройства. Свойство Canvas представляет собой класс, инкапсулирующий свойства и методы для работы с графикой.

В среде C++Builder три способа для работы с графикой:

• **Канва** предоставляет битовую карту поверхности для рисования на форме, графическом компоненте, или на другом битовом образе. Кроме формы, для рисования используются объекты классов TImage и TPaintBox. Canvas является свойством этих классов. Рассмотрим пример рисования на форме:

```
void __fastcall TForm1::FormPaint(TObject *Sender) {
Canvas->Pen->Color = clBlue; // выбрать цвет контура
 Canvas->Brush->Color = clYellow; // выбрать цвет заливки
 Canvas->Ellipse(10, 20, 50, 50); // нарисовать эллипс
}
```

Метод FormPaint здесь вызывается событием OnPaint формы при ее рисовании. Отметим, что при рисовании непосредственно на форме возникает ряд проблем, связанных с ее перерисовкой. В следующем примере рассматриваются различные способы задания цвета формы и цвета линий для рисования на компоненте TImage.

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

```
 if (ColorDialog1->Execute())
   { Form1->Color = ColorDialog1->Color;
}
}
void __fastcall TForm1::Button2Click(TObject *Sender)
{ Image1->Canvas->Pen->Width=2;
randomize();
Image1->Canvas->Pen->Color =(Graphics::TColor)random(256) ;
Image1->Canvas->LineTo(100,200);
}
```

При нажатии на первую кнопку выбирается цвет формы, при нажатии на вторую кнопку рисуются линии случайно выбранным цветом.

• **Графика (**TGraphic**)** представляет абстрактный базовый класс для работы с графикой. C++Builder определяет графические классы TBitmap, TClipBoard, TImage, TIcon и TMetafile, производные от базового класса TGraphic. Объекты этих классов используют методы рисования на канве и имеют собственные методы. Широко используются методы класса TGraphic LoadFromClipboardFormat(), LoadFromFile(), LoadFromStream(), SaveToClipboardFormat(), SaveToFile(), SaveToStream().

• **Рисунок (**TPicture**)** представляет собой контейнер для графических объектов и может содержать любые графические объекты. Свойствами этого класса являются: Bitmap, Graphic, Icon, Metafile, PictureAdapter. Контейнерный класс TPicture может содержать битовый образ, пиктограмму, метафайл или некоторый другой графический тип, а приложение будет обращаться ко всем объектам контейнера посредством объекта класса TPicture. Если необходимо указать доступ к конкретному графическому объекту, задайте его в свойстве Graphic данного рисунка. Методы этого класса: LoadFromClipboardFormat(), LoadFromFile(), LoadFromStream(), SaveToClipboardFormat(), SaveToFile(), SaveToStream().

Использование канвы. Класс Canvas инкапсулирует графические функции Windows на различных уровнях, начиная с функций высокого уровня для рисования линий, фигур и текста. Далее идут свойства и методы среднего уровня для манипуляций с канвой. В табл. 7 приводятся характеристики основных методов и свойств канвы.

Таблица 7

Уровень	Действие	Методы	Свойства
Высокий	Определяет текущую позицию пера	MoveTo	PenPos
	Рисует прямую до заданной точки	LineTo	PenPos
	Рисует прямоугольник заданного размера	Rectangle	
	Рисует эллипс заданного размера	Ellipse	
	Выводит текстовую строку	TextOut	
	Задает высоту текстовой строки	TextHeight	
	Задает ширину текстовой строки	TextWidth	
	Вывод текста внутри прямоугольника	TextRect	
	Заливка указанного прямоугольника цветом и текстурой текущей кисти	FillRect	
	Заливка области канвы (произвольной формы) заданным цветом	FloodFill	
Средний	Используется для установки цвета, стиля, ширины и режима пера		Pen
	Используется для установки цвета и текстуры при заливке графических фигур и фона канвы.		Brush
	Используется для установки шрифта заданного цвета, размера и стиля		Font
	Используется для чтение и записи цвета заданного пикселя канвы		Pixels
	Копирует прямоугольную область канвы в режиме CopyMode	CopyRect	CopyMode
	Копирует прямоугольную область канвы с заменой цвета	BrushCopy	
	Рисует битовый образ, пиктограмму, метафайл в заданном месте канвы	Draw	
	Рисует битовый образ, пиктограмму или метафайл так, чтобы целиком заполнить заданный прямоугольник	StretchDraw	
	Используется как параметр при вызове функций Windows GDI		Handle

Объекты класса Canvas являются свойствами формы и всех графических объектов, например TForm, TImage, TBitmap. Рассмотрим пример:

```
void __fastcall TForm1::FormActivate(TObject *Sender){
WindowState = wsMaximized;
Timer1->Interval = 50;
randomize();
}
//---------------------------------------------------------
void __fastcall TForm1::Timer1Timer(TObject *Sender)
{ int x, y;
```

```
x = random(Screen->Width - 10);
y = random(Screen->Height - 10);
  Canvas->Pen->Color = clYellow;
Canvas->Brush->Color = (Graphics::TColor) random(256);
switch (random(5))
{ case 0: Canvas->Pen->Style = psSolid; break; // стиль линии
  case 1: Canvas->Pen->Style = psDash; break;
  case 2: Canvas->Pen->Style = psDot; break;
  case 3: Canvas->Pen->Style = psDashDot; break;
  case 4: Canvas->Pen->Style = psDashDotDot; break;
}
Canvas->Rectangle(x, y, x + random(400), y + random(400)); }
```

Здесь в методе FormActivate() формы задается интервал времени, через который происходит событие OnTimer компонента TTimer, которое осуществляет перерисовку прямоугольника на форме.

В качестве примера рассмотрим рисование графика функции y=f(x). Пусть x принимает значения из промежутка $[x_1, x_2]$, а y изменяется в промежутке $[y_1, y_2]$. График функции будем рисовать в прямоугольнике на экране с координатами верхнего левого угла (X_1, Y_1) и нижнего правого угла – (X_2, Y_2). Таким образом, необходимо выполнить масштабирование: преобразование координат точек функции (x,f(x)) в соответствующие координаты точек (пикселей) заданного прямоугольника на экране. В процессе преобразования будем учитывать, что ось Oy на экране имеет направление сверху вниз. Поэтому изменим направления оси Oy в заданном прямоугольнике на экране на привычное для нас направление снизу вверх. Для преобразования координат точки функции (x, y) в координаты соответствующего пикселя (X, Y) на экране можно использовать следующие формулы:

$$X=X_1+[(X_2-X_1)/(x_2-x_1)](x-x_1);$$

$$Y=Y_2-[(Y_2-Y_1)/(y_2-y_1)](y-y_1).$$

Ниже приводится фрагмент программы для рисования графика функции y=cos x, где x изменяется в промежутке $[0, 2\pi]$. График будем рисовать на всей поверхности компонента Image1, т. е. $x \in$[0,Image1->Width], а $y \in$[0,Image1->Height]. Получим следующий код:

```
#define pi 3.14159
double X,Y; // координаты точек функции
int PX,PY; // координаты пикселей на экране
Y=cos(0);
PY=Image1->Height-Image1->Height/2*(Y+1);
Image1->Canvas->MoveTo(0,PY); // перемещаем перо в начальную
                              // точку рисования графика
for(X=0.05;X<=2*pi;X+=0.05){
```

```
Y=cos(X);
PX=Image1->Width/(2*pi)*X;
PY=Image1->Height-Image1->Height/2*(Y+1);
Image1->Canvas->LineTo(PX,PY);
}
```

Графические файлы. Приложения C++Builder поддерживают загрузку и сохранение рисунков и графиков в файлах изображений. Для загрузки изображения из файла можно использовать метод LoadFromFile(). Чтобы сохранить изображение в файле, используется метод SaveToFile(). Единственным параметром этих методов является имя файла. Рисунок распознает стандартное расширение файлов битовых образов .bmp и создает свою графику как объект класса TBitmap, а затем вызывает метод LoadFromFile загрузки изображения из файла с указанным именем.

Ниже приводится пример приложения, иллюстрирующего работу с графическим компонентом TImage. На рис. 18 приведена форма приложения в процессе проектирования.

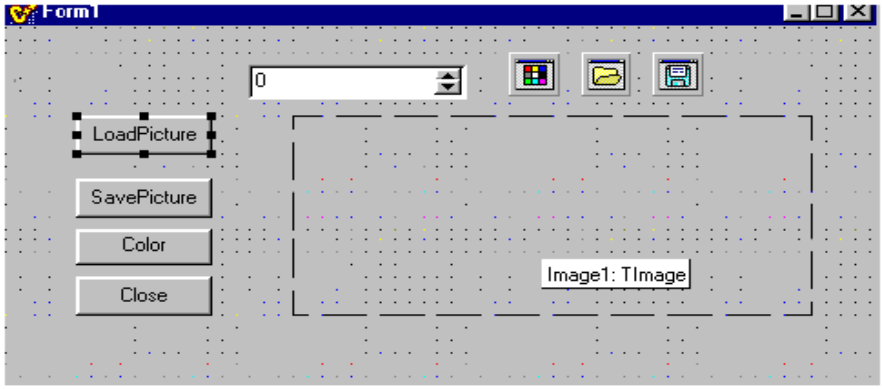

Рис. 18. Форма в процессе проектирования

```
// Обработчик событий для перемещения мыши по
// компоненту Image1
void __fastcall TForm1::Image1MouseMove(TObject *Sender,
 TShiftState Shift, int X, int Y)
{  if( (Shift.Contains(ssLeft)) && (drawFlag) )
  Image1->Canvas->LineTo(X,Y);
 else
 Image1->Canvas->MoveTo(X,Y);
}
//---------------------------------------------------------------------------
// Обработчик события нажатия на клавишу мыши
void __fastcall TForm1::Image1MouseDown(TObject *Sender,
     TMouseButton Button, TShiftState Shift, int X, int Y)
{drawFlag=true;
 }
```

```
//---------------------------------------------------------------------------
// Обработчик события отпускания клавиши мыши
void __fastcall TForm1::Image1MouseUp(TObject *Sender,
 TMouseButton Button, TShiftState Shift, int X, int Y)
{ drawFlag=false;
 }
//---------------------------------------------------------------------------
// Обработчик события для кнопки Close
void __fastcall TForm1::Button1Click(TObject *Sender)
{ Close();
}
//---------------------------------------------------------------------------
// Обработчик события для кнопки Color
void __fastcall TForm1::Button2Click(TObject *Sender)
{ if( ColorDialog1->Execute())
  Image1->Canvas->Pen->Color = ColorDialog1->Color;
 }
//---------------------------------------------------------------------------
// Обработчик события для кнопки LoadPicture
void __fastcall TForm1::Button3Click(TObject *Sender)
{  if (OpenDialog1->Execute())
  Image1->Picture->LoadFromFile( OpenDialog1->FileName );
  }
//---------------------------------------------------------------------------
// Обработчик события для кнопки SavePicture
void __fastcall TForm1::Button4Click(TObject *Sender)
{  if (SaveDialog1->Execute())
  Image1->Picture->SaveToFile( SaveDialog1->FileName );
  }
//---------------------------------------------------------------------------
// Обработчик изменений в компоненте CSpinEdit1
void __fastcall TForm1::CSpinEdit1Change(TObject *Sender)
{ Image1->Canvas->Pen->Width= CSpinEdit1->Value;
 }
```

Обслуживание палитр. Компонентам, содержащим графические изображения, может потребоваться взаимодействие с Windows и экранным драйвером, чтобы обеспечить надлежащее отображение данных компонентов. Реализация палитр призвана обеспечить, чтобы самое верхнее активное окно использовало полную цветовую палитру, в то время как фоновые окна максимально использовали оставшиеся цвета палитр.

C++Builder не содержит самостоятельных средств для создания и обслуживания иных палитр, кроме палитры битовых образов. Однако, можно получить дескриптор некоторой палитры с помощью метода GetPalette(), а затем воспользоваться методом Windows API CreatePalette().

Компоненты C++Builder автоматически поддерживают механизм реализации палитр. Таким образом, если компонент имеет палитру, вы можете воспользоваться двумя методами GetPalette() и PaletteChanged(), наследованными от базового компонентного класса TControl, чтобы управлять тем, как Windows обращается с этой палитрой.

Когда ваш компонент ассоциируется с некоторой палитрой посредством перегрузки метода GetPalette(), C++Builder автоматически берет на себя реакцию на сообщения Windows от палитр с помощью метода PaletteChanged().

Внеэкранные битовые образы. Методика программирования графических приложений заключается в создании внеэкранного битового образа, заполнении его конкретным изображением и копировании изображения из битового образа в экранное окно. Благодаря этому уменьшается заметное глазу мерцание экрана монитора, вызванное повторным рисованием непосредственно в экранном окне. C++Builder позволяет создавать объекты класса TBitmap для представления изображений файлов и других ресурсов, которые также способны работать как внеэкранные изображения.

C++Builder предусматривает четыре способа копирования изображений, которые приведены в табл. 8.

Таблица 8

Требуемый результат	Метод
Полное копирование графики	Draw
Копирование с масштабированием	StretchDraw
Копирование прямоугольного участка канвы	CopyRect
Копирование с растровыми операциями	BrushCopy

Далее приводится пример копирования изображения в обработчике события для кнопки:

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{  Graphics::TBitmap *pBitmap = new Graphics::TBitmap();
  try  {
    pBitmap->LoadFromFile("C:\\factory.bmp ");
    pBitmap->Transparent = true;
    pBitmap->TransparentColor = pBitmap->Canvas->Pixels[50,50];
    Form1->Canvas->Draw(0,0,pBitmap);
    pBitmap->TransparentMode = tmAuto;
    Form1->Canvas->Draw(50,50,pBitmap);
  }
  catch (...)
 {ShowMessage("Нельзя загрузить или отобразить изображение");
}
  delete pBitmap;
```

}

Анимация. Для воспроизведения анимации, состоящей в последовательном отображении кадров из стандартных avi-файлов Windows используется компонент Animate из страницы Win32 Палитры компонентов. Последовательность действий может быть следующей:

```
Animate1->FileName=”p.avi”;
Animate1->play(1, Animate1->FrameCount,1);
Animate1->stop();
```

Для обеспечения процесса мультимедиа и воспроизведения звука и анимации при отображении war-, rmi-, mid-файлов используется компонент MediaPlayer(страница System).

При анимации широко используется невидимый компонент Timer. При этом в одном из методов, например FormCreate() или Button1Click(), устанавливается частота прерывания: Timer1->Interval=time;

Компонент Timer генерирует событие OnTimer, в обработчике которого осуществляется перерисовка.

Вопросы

1. Как установить режимы рисования на канве? Какие режимы есть у пера?
2. Как очистить поверхность графического компонента Image?
3. Как установить цвет и ширину линии, которой рисуется график функции?
4. Назовите основные методы для рисования класса Canvas.
5. Назовите основные события графических компонентов Image и PaintBox.
6. Как вывести текст в графические компоненты?
7. Как вывести на графическом экране:

а) точку в позицию, указанную курсором мыши;

б) свою фотографию, загруженную из графического файла;

в) установить толщину линии, ее цвет и провести линию указанной длины и под заданным углом.

8. Как установить цвет рисунка при использовании графики и нарисовать указанным цветом две касающихся внутренним образом окружности?
9. Опишите методику анимации изображения.
10. Какой режим пера Pen используется для удаления рисунка при повторном его рисовании?
11. Какие свойства имеет компонент Timer?
12. Назовите основные события мыши.

13. Как в программе распознать координаты курсора мыши?

14. Как распознать нажатую кнопку мыши?

15. Когда наступают события мыши OnStartDrag, OnDragOver, OnDragDrop, OnEndDrag? Что можно распознать при обработке этих событий?

16. С помощью какой клавиши клавиатуры можно перемещать фокус с элемента на элемент?

17. Какие способы вывода графической информации Вы знаете?

18. Какие проблемы могут возникнуть при рисовании непосредственно на форме? Как их преодолеть? Почему такие проблемы не возникают при использовании компонента Image?

19. С помощью какого метода класса Canvas можно вывести текст?

Упражнения

1. Нарисовать график функции y=sin x.

2. Нарисовать график функции $y=-2x^2+3x$.

3. Вывести аналоговые часы (со стрелками). Предусмотреть кнопку для запуска и остановки часов. Использовать функцию Time() и класс TDataTime для получения текущего времени.

4. “Летающий” шарик. По достижении границы компонента он отражается от границы по правилам отражения. Предусмотреть кнопку остановки и запуска шарика.

5. Точка равномерно движется по окружности. Предусмотреть возможность увеличения скорости движения.

6. Построить простейший графический редактор с возможностью выбора цвета рисования.

7. Равносторонний треугольник вращается вокруг своего центра. Предусмотреть возможность увеличения скорости вращения.

8. Изобразить прямоугольник (квадрат), вращающийся вокруг своего центра. Предусмотреть возможность увеличения скорости вращения.

9. Вращаются два отрезка, каждый вокруг своей концевой точки. Предусмотреть возможность изменения скорости вращения каждого отрезка отдельно.

10. “Ипподром”. Играющий выбирает одну из трех лошадей, состязающихся на бегах, и выигрывает, если его лошадь приходит первой. Скорость передвижения лошадей на разных этапах выбирается программой с помощью датчика случайных чисел.

11. Игра в слова. Программа выбирает слово и рисует на экране столько прочерков, сколько букв в слове. Отгадать, какое слово загадано

программой. За один ход играющий указывает одну букву. Если буква не угадана, то играющий теряет одно очко. В начальный момент у играющего 15 очков.

12. Требуется ввести курсор в область экрана (небольшой круг), расположение которого неизвестно играющему. Если курсор приближается к области, то его цвет становится ярче, если удаляется, то тускнеет.

5. РАБОТА С БАЗАМИ ДАННЫХ

База данных – это поименованная совокупность взаимосвязанных данных, находящихся под управлением системы управления базами данных (СУБД). Различают иерархические, сетевые и реляционные СУБД. В C++ Builder используется несколько механизмов и инструментов для работы с базами данных: BDE (Borland Database Engine), ADO (ActiveX Data Object) позволяет коннектится к базам данных, используя объекты ActiveX, позволяет напрямую коннектится к базам данных поддерживающих InterBase.

5.1. Основные концепции реляционных баз данных

Реляционная база данных представляет собой одну или совокупность взаимосвязанных таблиц, состоящих из записей. С позиции реляционной алгебры БД это совокупность отношений над n множествами M1, M2, ..., Mn. Например, это множества студентов, преподавателей, экзаменов. Отношение представляет собой таблицу. При этом наборы вида <m1, m2,...,mn>, где mk – элемент из Mк, называются кортежами (строками таблицы или записями). Запись состоит из атрибутов, являющихся столбцами таблицы (полями записи). Каждый столбец имеет уникальное в таблице имя, которое записывается в верхней части таблицы и является именем поля. Атрибут, который может быть использован для однозначной идентификации конкретной записи, называется первичным ключом. Для таблицы Student (табл. 9) первичным ключом может быть номер студенческого билета. Для ускорения доступа к данным по первичному ключу в СУБД имеется механизм, называемый индексированием. Индекс представляет собой древовидный список, указывающий на местоположение записи для каждого первичного ключа. Возможно индексирование отношения с использованием атрибутов, отличных от первичного ключа (вторичного ключа), например фамилии студента.

Таблица 9 (Student)

Stud_Id	StudName	Kurs	Address	Predmet	Prepod
111111	Котов П.А.	2	Витебск, Ленина 1	Prog	Романчик
111112	Павлов А.И.	2	Минск, Репина 2	Matan	Примачук
111113	Коваль Н.К.	1	Гомель,Пушкина 3	Difur	Громак

Для поддержания ссылочной целостности данных в нескольких взаимосвязанных таблицах используется механизм внешних ключей. При этом некоему атрибуту одного отношения назначается ссылка на первичный ключ другого отношения; тем самым закрепляются связи подчиненности между этими отношениями. При этом отношение, на первичный ключ которого ссылается внешний ключ другого отношения, называется master-отношением, а отношение, от которого исходит ссылка, называется detail-отношением, или подчиненным отношением. После назначения такой ссылки СУБД имеет возможность автоматически отслеживать вопросы “ненарушения“ связей между отношениями. Если делается попытка вставить в подчиненную таблицу запись, для внешнего ключа которой не существует соответствия в главной таблице, СУБД сгенерирует ошибку. Если попытаться удалить из главной таблицы запись, на первичный ключ которой имеется ссылка из подчиненной таблицы, или изменить первичный ключ, СУБД также сгенерирует ошибку. Существуют два подхода к удалению и изменению записей из главной таблицы:

1) Запретить удаление всех записей, а также изменение первичных ключей главной таблицы, на которые имеются ссылки подчиненной таблицы.

2) Распространить всякие изменения в первичном ключе главной таблицы на подчиненную таблицу.

5.2. Проектирование баз данных

Проектирование базы данных (БД) можно представить в виде следующей последовательности шагов:

1. Разработка модели БД, которая включает в себя: идентификацию функциональной деятельности предметной области; идентификацию объектов, которые осуществляют эту деятельность; идентификацию взаимосвязей между объектами.

2. Определение атрибутов, которые уникальным образом идентифицируют каждый объект (первичный ключ). Первичный ключ гарантирует, что в таблице не будет содержаться двух одинаковых строк.

3. Устанавливание связи между объектами и проведение операции исключения избыточности данных – нормализация таблиц.

Таким образом, для проектируемой базы данных сначала определяются таблицы, поля, индексы и связи между таблицами

Существует несколько типов связей между таблицами: “один-к-одному”, “один-ко-многим”, “многие-ко-многим”. Связь “один-к-одному” представляет собой простейший вид связи данных, когда первичный ключ таблицы является внешним ключом, ссылающимся на первичный ключ другой таблицы. Связь “один-ко-многим” отражает реальную взаимосвязь в предметной области. Эта связь описывает механизм классификаторов или кодов. Например, 01.01 – математика. Связь “многие-ко-многим” в явном виде в реляционных базах данных не поддерживается.

Процесс нормализации. После разработки структуры таблиц проектируемую базу данных следует проанализировать, используя правила нормализации. Нормализация заключается в приведении таблиц к нормальным формам. Этот процесс включает: устранение повторяющихся групп (приведение к первой нормальной форме), удаление частично зависимых атрибутов (приведение ко второй нормальной форме), удаление транзитивно зависимых атрибутов (приведение к третьей нормальной форме).

На практике используется не более трех первых нормальных форм – следует учитывать время, необходимое для “соединения” таблиц при отображении на экране. После применения правил нормализации логические группы данных располагаются не более чем в одной таблице. На пересечении каждой строки и столбца таблицы всегда находится единственное значение.

Приведение к первой нормальной форме. Если поле в данной записи содержит более одного значения, такие группы данных называются повторяющимися. Первая нормальная форма не допускает наличия таких многозначных полей. Рассмотрим пример БД, содержащей таблицу Student (табл. 10; номер студенческого билета является первичным ключом). Для приведения таблицы к первой нормальной форме мы должны поле Address разбить на два поля, содержащие город и улицу, если в дальнейшем будет выполняться обработка данных по регионам.

Приведение ко второй нормальной форме. Следующий шаг в процессе нормализации состоит в удалении всех неключевых атрибутов, которые зависят только от части первичного ключа. Такие атрибуты называются частично зависимыми. Для приведения таблицы ко второй нормальной форме удалим из нее атрибуты Predmet и Prepod и создадим две

таблицы (табл. 11 и 12), которые будут содержать только эти атрибуты, и они же будут составлять их первичный ключ.

Таблица 10 (Student)

Stud_Id	StudName	Kurs	City	Street
111111	Котов П.А.	2	Витебск	Ленина 1
111112	Павлов А.И.	2	Минск	Репина 2
111113	Коваль Н.К.	1	Гомель	Пушкина 3

Таблица 11 (Subject)

Subj_Id	Subj_Name	Hour
10	Prog	120
20	Matan	180
30	Difur	70

Таблица 12 (Prepod)

Prepod_Id	Prepod_Name
11	Романчик
20	Примачук
32	Громак

Кроме этого будем использовать также таблицу Exams (табл. 13).

Таблица 13 (Exams)

Exam _Id	Stud_Id	Subj_Id	Exammark	ExamDate
23	111111	10	8	6/01/06
34	111112	20	6	10/01/06
42	311113	30	3	20/01/06

Приведение к третьей нормальной форме. Третий этап процесса приведения таблиц к нормальной форме состоит в удалении всех неключевых атрибутов, которые зависят от других неключевых атрибутов или вычисляются по ним. Каждый неключевой атрибут должен быть логически связан с атрибутом, являющимся первичным ключом. Например, из таблицы можно удалить атрибут Stipend. Получим табл. 14.

Таблица 14 (Stipend)

Stud_Name	Summa
Котов П.А.	120
Павлов А.И.	130
Коваль Н.К.	100

Добавим таблицу, связывающую преподавателей и предметы (табл. 15).

Таблица 15 (Sub_Prepod)

Prepod_Id	Subj_Id
11	10
32	20
43	30

Разбиение информации на мелкие единицы способствует повышению надежности базы данных, но снижает ее производительность.

На последнем шаге разработки БД необходимо спланировать вопросы безопасности и конфиденциальности информации и определить права на использование и на модификацию данных.

Теперь можно приступать к созданию приложений, работающих с базами данных. Рассмотрим сначала средства для разработки таблиц.

5.3. Утилита Database Desktop

Database Desktop – это старая утилита, которая поставляется для создания таблиц и интерактивной работы с таблицами различных форматов. При этом могут использоваться таблицы для локальных баз данных типа Paradox, dBase, MSAccess а также распределенных и SQL-серверных баз данных InterBase, Oracle, Informix, Sybase.

После запуска Database Desktop выберите команду меню File|New|Table для создания новой таблицы. Появится диалоговое окно выбора типа таблицы, например Paradox, DB2, dBase, MSAccess, MSSQL. После выбора типа таблицы появляется диалоговое окно, в котором можно определить поля таблицы и их тип. Таблицы создаются в online – режиме в рабочем пространстве DataBase Desktop, связанном с каталогом, который необходимо предварительно установить.

Для таблиц Paradox можно определить поля, находящиеся в начале записи и составляющие первичный ключ. Достаточно дважды щелкнуть мышкой по этому полю. С таблицей можно связать некоторые свойства.

Validity Checks (проверка правильности) – относится к полю записи и определяет минимальное и максимальное значение поля, а также значение по умолчанию.

Table Lookup (таблица для “подсматривания”) – позволяет вводить значение в таблицу, используя уже существующее значение в другой таблице.

В табл. 16 приведены типы полей записей для БД Paradox.

Таблица 16

Alpha	Строка длиной от 1 до 255 байт
Number	Числовое поле длиной 8 байт для чисел от 10^{-308} до 10^{308} с 15 значащими цифрами
Money	Форматированное числовое поле для отображения денежного знака
Short	Числовое поле длиной 2 байта для целых чисел от -32768 до 32767
Long Integer	Числовое поле длиной 4 байта для целых чисел от -2147483648 до 2147483648
BCD	Числовое поле данных в формате BCD (Binary Coded Decimal). Может иметь от 0 до 32 цифр после десятичной точки
Date	Поле даты длиной 4 байта
Time	Время в миллисекундах от полуночи
Timestamp	Обобщенное поле даты длиной 8 байт. Содержит дату и время
Memo	Поле для хранения символов, суммарное число которых превышает 255
Graphic	Поле для графической информации
OLE	Содержит OLE-данные: образы, звук, видео
Logical	Содержит значения T(true) или F(false)
Autoincrement)	Содержит значение типа *integer,* которое автоматически увеличивается на 1, начиная с 1
Binary	Содержит любую двоичную информацию
Bytes	Строка длиной от 1 до 255 байт

Secondary Indexes (вторичные индексы) – обеспечивают доступ к данным в порядке, отличном от задаваемого первичным ключом. Вторичные ключи могут использоваться в подчиненной таблице для указания ссылки на главную таблицу.

Referential Integrity (ссылочная целостность) – позволяет задать связи между таблицами и поддерживать эти связи на уровне ядра.

Password Security (парольная защита) – позволяет защитить таблицу паролем.

Table Language – позволяет задать языковый драйвер.

Поля таблиц формата dBase описаны в табл. 17.

В таблицах dBase не существует первичных ключей. Однако, это обстоятельство можно преодолеть путем определения уникальных (Unique) и поддерживаемых (Maintained) индексов (Indexes).

Отметим, что использование утилиты Database Desktop является одним из способов создания таблиц. Например, таблицы можно создавать средствами конкретной СУБД.

Таблица 17

Character	Строка
Float (numeric)	Числовое поле размером от 1 до 20 байт
Number (BCD)	Числовое поле размером от 1 до 20 байт, содержащее данные в формате BCD
Date	Поле даты длиной 8 байт
Logical	Поле может содержать только значения “истина” – T, t, Y, y или ложь – F, f, N, n
Memo	Поле для хранения последовательности символов, длина которой превышает 255 байт
OLE	Поле, содержащее OLE-данные
Binary	Поле, содержащее двоичную информацию

5.4. Структурированный Язык Запросов SQL

Язык Запросов SQL – основной язык для работы с реляционнми базами данных. Состав языка SQL следующий:

Язык манипулирования данными состоит из команд: SELECT (выбрать), INSERT (вставить), UPDATE (обновить), DELETE (удалить).

Язык определения данных используется для создания и изменения структуры БД и ее составных частей – таблиц, индексов, представлений (виртуальных таблиц). Основными его командами являются: CREATE DATABASE (создать базу данных), CREATE TABLE (создать таблицу), CREATE INDEX (создать индекс), CREATE PROCEDURE (создать сохраненную процедуру), ALTER DATABASE (модифицировать базу данных), ALTER TABLE (модифицировать таблицу), ALTER INDEX (модифицировать индекс), ALTER PROCEDURE (модифицировать сохраненную процедуру), DROP DATABASE (удалить базу данных), DROP TABLE (удалить таблицу), DROP INDEX (удалить индекс), DROP PROCEDURE (удалить сохраненную процедуру).

Язык управления данными (управления доступом) состоит из двух основных команд: GRANT (дать права), REVOKE (забрать права).

5.5. Команды языка манипулирования данными

Наиболее важной командой языка манипулирования данными является команда SELECT. Формат команды SELECT в языке SQL:

SELECT поля FROM таблицы WHERE условие;

Базовыми операциями являются: **выборка, проекция, соединение, объединение**.

Операция выборки позволяет получить все либо часть строк таблицы.

SELECT * FROM Student; – Получить все строки таблицы Student

SELECT * FROM Student WHERE Kurs=2 – Получить подмножество строк таблицы, удовлетворяющих условию Kurs=2. Точка с запятой является стандартным признаком конца команды, который вставляется автоматически.

Операция проекции позволяет выделить подмножество столбцов таблицы.

SELECT StudName FROM Student WHERE Kurs=2; – Получить имена студентов второго курса.

Операция соединения позволяет соединять строки из более чем одной таблицы:

SELECT StudName, Exammark FROM Students, Exams WHERE Students.Stud_Id =Exams.Stud_Id – Получить список студентов и экзаменационных оценок.

Операция объединения позволяет объединять результаты отдельных запросов. Предложение UNION объединяет вывод двух или более SQL-запросов.

SELECT name FROM employee WHERE country = "Беларусь" UNION SELECT contact_name, FROM customer WHERE country = "Беларусь"; – Получить список работников и заказчиков, проживающих в Беларуси.

Простейшие конструкции команды SELECT. Список выбираемых элементов может содержать: имена полей, символ ‘*’, вычисления, литералы, функции, агрегирующие конструкции.

Вычисления:

SELECT StudName, Summa, Summa * 1.15 FROM Stipend – Получить список студентов и их стипендию, в том числе увеличенную на 15%.

Литералы. Для наглядности результата в запросах можно использовать литералы – строковые константы, заключенные в одинарные или двойные кавычки. Например:

SELECT StudName, "получает", Summa, " в месяц" FROM Stipend

Два или более столбца, имеющих строковый тип, можно соединять друг с другом, а также с литералами с помощью операции конкатенации (||). Например:

SELECT "сотрудник " || first_name || " " || last_name FROM Prepods

Работа с датами. В языке SQL имеются возможности конвертирования дат в строки и работы с датами. Внутренне дата содержит значения

даты и времени. Внешне дата может быть представлена строками различных форматов, например:

- “October 27, 2005”
- “10/27/2005”

Дата может неявно конвертироваться в строку (из строки), если имеет один из допустимых форматов. Например:

SELECT StudName, ExamDate FROM Student, Exams WHERE ExamDate > '6/01/06' – получить список студентов, сдававших экзамен после 6/06/06.

Значения дат можно сравнивать, а также вычитать одну дату из другой.

Агрегатные функции. К агрегатным функциям относятся функции вычисления суммы (SUM), максимального (MAX) и минимального (MIN) значений столбцов, среднего арифметического (AVG), количества строк, удовлетворяющих заданному условию. Например: SELECT count(*),sum(budget),avg (budget), min(budget), max(budget) FROM department WHERE head_dept = 100
– вычислить количество отделов, являющихся подразделениями отдела 100, их суммарный, средний, минимальный и максимальный бюджеты.

Условия отбора. Директива WHERE содержит условия отбора (предикат). Запрос возвращает только строки, для которых предикат имеет значение true. Типы предикатов, используемых в предложении WHERE:

Сравнение: = (равно); <> (не равно); != (не равно); > (больше); < (меньше); >= (больше или равно); <= (меньше или равно); BETWEEN, IN , LIKE, CONTAINING, IS NULL, EXIST, ANY, ALL.

Предикат BETWEEN задает диапазон значений, для которого истинно значение выражения. Например:

SELECT StudName, Stipend FROM Student WHERE Stipend BETWEEN 120 AND 200 – получить список студентов стипендия которых больше 120 и меньше 200.

Тот же запрос с использованием операторов сравнения будет выглядеть следующим образом:

SELECT StudName, Stipend FROM Student WHERE Stipend>=120000 AND Stipend<=200000

Предикат IN (NOT IN) проверяет, входит ли заданное значение, предшествующее ключевому слову “IN”, в указанный в скобках список. Например:

SELECT name FROM employee WHERE job_code IN ("VP", "Admin", "Finan") – получить список сотрудников, занимающих должности “вице-президент”, “администратор”, “финансовый директор”.

Предикат LIKE проверяет, соответствует ли данное символьное значение строке с указанной маской. В качестве маски используются все разрешенные символы (с учетом верхнего и нижнего регистров), а также специальные символы: % – замещает любое количество символов, _ – замещает только один символ. Например:

SELECT StudName FROM Student WHERE StudName LIKE "Ф%" – получить список студентов, фамилии которых начинаются с буквы ‘Ф’.

Предикат CONTAINING аналогичен предикату LIKE, однако он не чувствителен к регистру букв.

Предикат IS NULL принимает значение true только тогда, когда выражение слева от “IS NULL” имеет значение null (пусто, не определено).

Логические операторы. К логическим операторам относятся NOT, AND, OR. В одном предикате логические операторы выполняются в указанном порядке.

Преобразование типов. В SQL имеется возможность преобразовать значение к другому типу для выполнения операций сравнения. Для этого используется функция CAST.

Изменение порядка выводимых строк. Порядок выводимых строк может быть изменен с помощью предложения ORDER BY в конце SQL-запроса. Это предложение имеет вид: ORDER BY [ASC | DESC]

Способом по умолчанию является упорядочивание “по возрастанию” (ASC). Если указано “DESC”, упорядочивание производится “по убыванию”. Например:

SELECT StudName, Stipend FROM Student ORDER BY StudName – получить список в алфавитном порядке.

Операция соединения. Используется в языке SQL для вывода связанной информации, хранящейся в нескольких таблицах. Операции подразделяются на внутренние и внешние. Связывание производится, как правило, по первичному ключу одной таблицы и внешнему ключу другой таблицы. Предложение WHERE может содержать множественные условия соединений.

Внутренние соединения. Внутреннее соединение возвращает только те строки, для которых условие соединения принимает значение true. Рассмотрим пример запроса:

SELECT StudName, ExamMark FROM Student, Exams WHERE Kurs=2 AND ExamMark=5 – получить список студентов 2-го курса, сдававших экзамен на 5.

В запросе можно использовать способ непосредственного указания таблиц или указания таблиц с помощью алиасов (псевдонимов).

Внешние соединения. Внутреннее соединение возвращает только строки, для которых условие соединения принимает значение true. Внешнее соединение возвращает все строки из одной таблицы и те строки из другой таблицы, для которых условие соединения принимает значение true. Существуют два вида внешнего соединения: в левом соединении (LEFT JOIN) запрос возвращает все строки из_ таблицы, стоящей *слева* от LEFT JOIN и только те из правой таблицы, которые удовлетворяют условию соединения. Для правого соединения – все наоборот. Например:

SELECT name, department FROM employee e LEFT JOIN department d ON e.dept_no = d.dept_no – получить список сотрудников и название их отделов, включая сотрудников, еще не назначенных ни в какой отдел.

5.6. Выполнение инструкций SQL

Для выполнения инструкции SQL создается содержащая ее строка и передается свойству SQL компонента TQuery. Компонента TQuery, должна быть помещена на форму, ее свойство DatabaseName настроено на нужный алиас (если базы данных не существует, можно создать ее в SQL Explorer).

Для создания статического запроса можно ввести SQL-предложение в свойство SQL компонента TQuery.

Способ создания динамического запроса состоит в создании строки и добавлении ее в свойство SQL при выполнении приложения. Для выполнения запроса, изменяющего структуру данных, вставляющего или обновляющего данные на сервере, нужно вызвать метод ExecSQL() компонента TQuery. Например:

```
Query1->Close();
Query1->SQL->Clear();
Query1->SQL->Add("Delete   * from Country where Name = 'Blr' ");
Query1->ExecSQL();
```

Приведем упрощенный синтаксис SQL-предложения для создания таблицы на SQL-сервере:

CREATE TABLE table (<col_def> [, <col_def> | <tconstraint> ...]);

где table – имя создаваемой таблицы, <col_def> – описание поля, <tconstraint> – описание ограничений и/или ключей (квадратные скобки [] означают необязательность, вертикальная черта | означает “или”).

Приведем несколько примеров создания таблиц с помощью SQL.

Пример. Простая таблица с конструкцией PRIMARY KEY на уровне поля:

CREATE TABLE REGION (REGION REGION_NAME NOT NULL PRIMARY KEY, POPULATION INTEGER NOT NULL);

Предполагается, что в базе данных определен домен REGION_NAME, например, следующим образом:

CREATE DOMAIN REGION_NAME AS VARCHAR(40) CHARACTER SET WIN1251 COLLATE PXW_CYRL;

Для выполнения запроса на получение данных с помощью оператора SELECT, нужно вызвать метод Open() компонента TQuery.

```
Queryl->Close ();
Queryl->SQL->Clear ();//Очистить свойство SQL от  запроса
Queryl->SQL->Add(str) ; //Присвоить  текст свойству SQL
Query1->Open(); // выполнить команду SQL
```

5.7. Разработка приложений баз данных

Механизм BDE. Механизм BDE (Borland Database Engine), обеспечивающий работу визуальных компонентов баз данных, действует как интерфейс между приложением и базой данных. BDE обращается к драйверам для баз данных указанного типа, возвращая запрошенные данные. Используя BDE, можно получить доступ к локальным стандартным базам данных, к источникам данных ODBC и к SQL-серверам баз данных. Чтобы получить доступ к содержимому базы данных, приложению необходимо знать только ее алиас.

Использование визуальных компонентов. C++Builder предоставляет разработчикам компоненты для работы с базами данных из VCL:

1. Компоненты управления данными на вкладке Data Control (такие как TDBEdit, сетка TDBGrid или DBNavigator) для отображения и редактирования записей на форме.

2. Компоненты доступа к данным на вкладке Data Access. Компонент источника TDataSource служит как интерфейс межкомпонентной связи между таблицей TTable или запросом Tquery и компонентой управления.

C++Builder поддерживает трехступенчатую модель разработки приложения баз данных. В этой модели компонент управления связан с компонентом источника TDataSource, а тот, в свою очередь, получает фактические данные из таблицы или запроса посредством механизма BDE.

Рассмотрим работу компонента доступа.

Источники данных. Невидимый компонент TDataSource действует как интерфейс между некоторым объектом набора данных (таблица, запрос) и визуальным компонентом управления. С одной стороны, все наборы данных должны быть ассоциированы с некоторым источником. С другой стороны, каждый компонент управления должен быть ассоцииро-

ван с источником, чтобы получать данные для отображения и редактирования.

Свойство DataSet компонента TDataSource определяет имя конкретного набора данных (таблицы или запроса), который питает данный источник. С помощью этого свойства можно переключаться с одного набора данных на другой во время выполнения программы. Следующий код реализует попеременное подключение объекта источника DataSource1 к таблице заказчиков "Заказчики" или к таблице "Заказы":

```
if (DataSourcel->DataSet == "Заказчики")
DataSourcel->DataSet = "Заказы";
```

Чтобы синхронизировать работу компонентов управления на двух формах, достаточно установить свойство DataSet на один и тот же набор данных:

```
void__fastcall TForm2::FormCreate (TObject *Sender) {
 DataSource1->DataSet = Form1->Table1;}
```

С компонентом TDataSource связаны три события. Событие OnDataChange возникает при перемещении курсора на новую запись. Событие OnStateChange возникает при изменении свойства State наборов данных. Например, следующий обработчик события будет отслеживать изменения состояния таблицы MyTable.

```
void__fastcall TForm1::StateChange(TObject *Sender)
{String s;;
switch (MyTable->State) {
case dsInactive: s="Таблица неактивна"; break;
case dsBrowse: s = "Идет просмотр"; break;
case dsEdit: s = "Идет редактирование"; break;
case dsInsert: s = "Идет вставка записи"; break;
}
// Вывод текстовой строки s
Form1->Caption=s;
}
```

Событие OnUpdateData возникает перед фактическим обновлением текущей записи.

3. Компоненты DDE.

Таблицы. Компонент TTable устанавливает прямую связь с таблицей базы данных посредством BDE, причем все записи или столбцы этой таблицы становятся доступными для приложения.

Свойство Active компонента разрешает или запрещает режим просмотра "живых данных" таблицы на этапе проектирования. Значение true или метод Open() открывают просмотр таблицы. Значение false или метод Close() закрывают просмотр.

Свойство DatabaseName содержит псевдоним базы данных или путь к ее каталогу. Использование псевдонима всегда предпочтительнее: вы можете переназначить физический носитель данных. Перекомпиляция приложения не требуется – просто измените путь на вкладке Aliases в утилите конфигурации BDE.

Свойство TableName позволяет выбрать фактическое имя таблицы из выпадающего списка.

Свойство Exclusive разрешает или запрещает другому приложению обращаться к таблице, пока вы ее используете сами.

Свойство IndexFiles открывает диалог выбора индексного файла для таблицы.

Свойство IndexName задает правило сортировки данных, отличное от упорядочивания по первичному ключу (primary key order).

Свойство Filter позволяет устанавливать критерий фильтрации, в соответствии с которым адресуется некоторое подмножество записей таблицы.

Свойства MasterFields и MasterSource участвуют в образовании связи двух таблиц (ведущей и ведомой) по принципу master-detail.

С компонентом TTable связаны следующие методы:

GotoCurrent() – синхронизирует перемещения курсора по нескольким табличным компонентам, ассоциированным с одной и той же фактической таблицей.

First(), Next(), Prior(), Last() и MoveBy() – используются для навигации по данным таблицы.

SetKey(), FindKey(), FindNearest(), GotoKey() и GotoNearest() – используются для поиска по специфическим значениям ключей.

Append(), Insert(), AppendRecord() и InsertRecord() добавляют новую запись к таблице.

Delete() – вычеркивает текущую запись.

Edit() – разрешает приложению модифицировать записи, а Post() вызывает фактическое изменение содержимого базы данных.

Запросы. Компонент TQuery обеспечивает доступ к нескольким таблицам одновременно. Вид возвращаемого набора данных зависит от SQL запроса, который может быть либо статическим, когда параметры запроса задаются на стадии проектирования, или динамическим, когда параметры определяются во время выполнения программы.

Свойство Active разрешает или запрещает режим просмотра "живых данных", возвращаемых запросом на этапе проектирования. Свойство DatabaseName содержит псевдоним базы данных или полный путь к ее каталогу, необходимые для разрешения запроса. Свойство SQL исполь-

зуется для ввода команды SQL посредством строчного редактора списка, который открывается двойным щелчком мышью.

Аналогично таблице, компонент запроса TQuery также инкапсулирует следующие методы:

First(), Next(), Prior(), Last() и MoveBy() – используются для навигации по результатам динамического запроса;

Append(), Insert(), AppendRecord() и InsertRecord() – добавляют новую запись к таблице;

Delete() – удаляет текущую запись;

Edit() – разрешает приложению модифицировать записи;

Post() – вызывает фактическое изменение содержимого базы данных.

Следующая процедура иллюстрирует процесс создания простой формы для базы данных StudentBase:

1.Поместите на форму Form1 пять наборов TTable, TDataSource, TDBGrid, TDBNavigator.

2. Установите свойства объектов таблиц Table1 – Table5

DatabaseName =StudentBase; Table1Name =students,

Table2Name=predmets, Table3Name =prepods, Table4Name=exams,

TableName5=raspisanie.

3. Установите свойства объектов источников DataSourcel ->

DataSet =Table1, …, DataSource5 -> DataSet =Table5

4. Установите свойства компонентов отображения данных

DBGrid1 -> DataSource = DataSource1, …, DBGrid5 ->DataSource = DataSource5

5. Установите свойство Active = true для таблиц, чтобы сразу же отобразить данные в сетках на форме.

6.Создайте новую форму Form2 и поместите на нее компоненты Query1, Table1, DataSourse1, DBGrid1. Задайте свойства этих компонентов.

7. В свойстве SQL компонента Query1 задайте SQL запрос:

```
SELECT StudentsName, ExamMark, PrepodName, PredmetName
FROM students, exams, predmets,prepods,raspisanie
WHERE students.Id_stud=exams.Id_stud
AND raspisanie.Id_predmet=exams.Id_predmet
AND raspisanie.Id_prepod=prepods.Id_prepod
AND exams.Id_predmet=predmets.Id_predmet
AND predmets.PredmetName LIKE 'Програм'
AND exams.ExamMark >'5';
```

8. Выполните компиляцию и запустите приложение. На рис. 19 показан результат выполнения приложения.

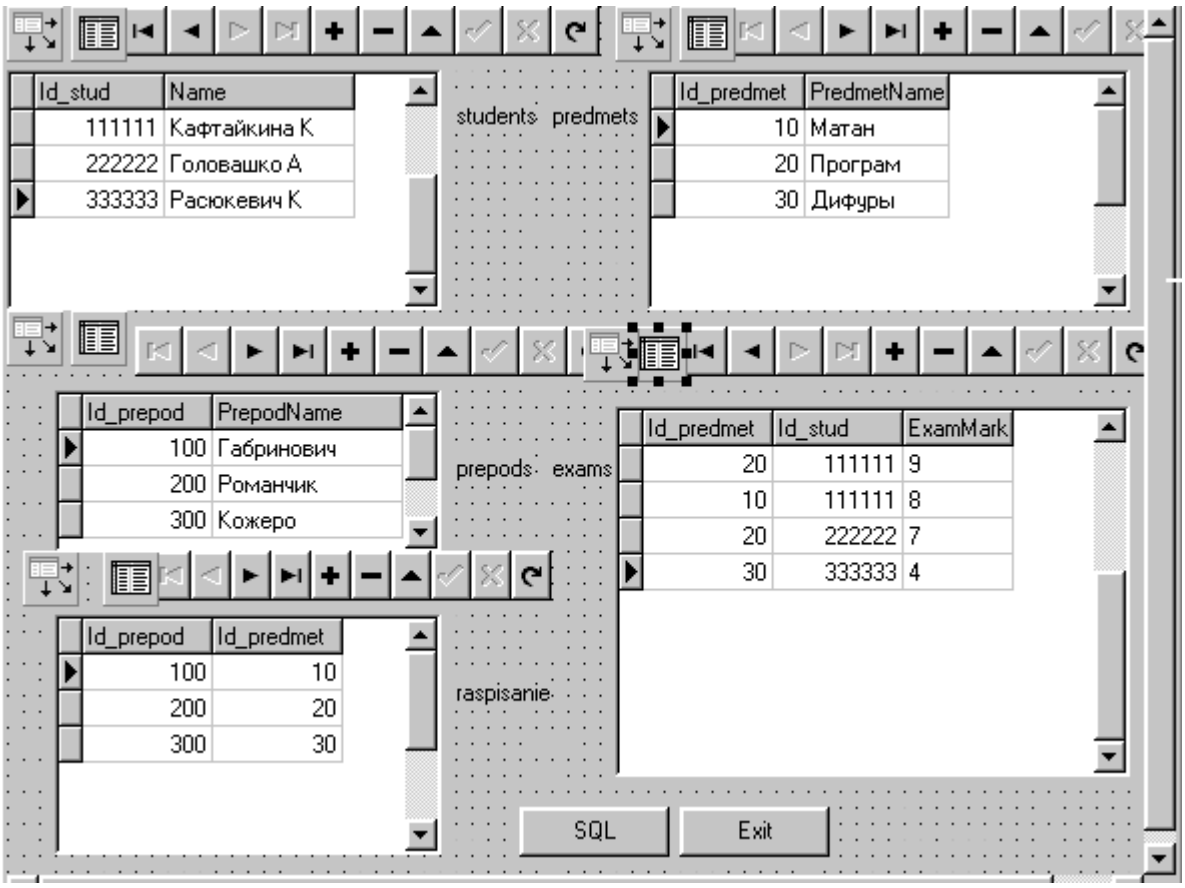

Рис. 19. Результат выполнения приложения

Чтобы связать ведущую таблицу students с ведомой таблицей exams необходимо выполнить следующее:

1. Активизируйте ведомую таблицу Table5 и установите свойство MasterSource = DataSource1

2. Дважды щелкните мышью в графе значений свойства MasterFields и в открывшемся окне редактора связи полей выберите Id_stud (связующее поле таблиц) из выпадающего списка Available Indexes; задайте Id_stud в списках Detail Fields и Master Fields; нажмите кнопку Add, чтобы добавить в список Joined Fields соединение Id_Stud-> Id_Stud; нажмите кнопку OK, подтверждая сделанный выбор.

Следующая процедура иллюстрирует процесс создания формы со статическим запросом к таблице EMPLOYEE для получения всей информации о служащих, зарплата которых превышает заданную величину:

1. Поместите компонент TQuery на форму.

2. Установите псевдоним адресуемой базы данных сервера в свойстве DatabaseName. В примере используется псевдоним BCDEMOS локаль-

ной демонстрационный базы данных, содержащей, в частности, таблицу служащих некоторого предприятия.

3. Откройте строчный редактор списка, введите команду SQL

SELECT * FROM EMPLOYEE WHERE Salary>40000 и нажмите кнопку ОК.

4. Поместите на форму компонент TDataSource и установите его свойство DataSet = Query1.

5. Поместите на форму компонент управления TDBGrid и установите его свойство DataSource = DataSource1.

6. Установите свойство Active = true для запроса Queryl с тем, чтобы сразу же отобразить живые данные в компоненте управления.

Далее приводится пример приложения для выполнения динамических SQL-запросов к таблице служащих.

Свойство SQL компонента TQuery имеет тип TStrings и содержит список текстовых строк наподобие массива. Листинг показывает обработчик события Button1Click, реализующий ввод запроса пользователем при нажатии кнопки на форме. Введенная команда SQL записывается в строчный массив (того же типа TStrings) свойства Memo1->Lines компонента TMemo. Результаты запроса можно, как и в предыдущем примере, отобразить с помощью компонента TDBGrid.

```
void__fastcall TForm1::Button1Click(TObject *Sender) {
// Проверить, введена ли какая-то строка в Memo1
if (strcmp(Memo1->Lines->Strings[0].c_str(), "") = = 0) (
MessageBox(0, "Не введен SQL-запрос", "Ошибка", MB_OK) ;
return;
}
else {
// Исключить предыдущий запрос, если он имел место
Queryl->Close ();
// Очистить свойство SQL от предыдущего запроса
Queryl->SQL->Clear ();
// Присвоить введенный в Memo1 текст свойству SQL
Queryl->SQL->Add(Memo1->Lines->Strings[0].c_str()) ;
try{
Query1->Open(); // выполнить команду SQL
}
catch(EDBEngineError* dbError){} .// обработка ошибок BDE
{
for (int i=0; i<dbError->ErrorCount; i++)
MessageBox (0, dbError[i].Message.c_str(), "SQL Error", MB_OK) ;
   }
  }
 }
```

Для динамического формирования текста командной строки SQL во время выполнения программы удобно использовать стандартную функцию языка C sprintf(). Эта функция замещает параметры форматирования (%s, %d, %n и т.д.) передаваемыми значениями. Например, в результате подстановки значений параметров форматирования

```
tblName = "Student";
fldName = "Kurs";
fldValue =2;
char sqls[80];
sprintf(sqls, "SELECT * FROM %s WHERE %s = %d", tblName, fldName, fldValue);
```

символьный массив sqls будет содержать следующую команду:

```
SELECT * FROM Student WHERE Kurs =2.
```

Далее созданный запрос выполняется

```
Query1->Close() ;
Query1->SQL->Clear() ;
Query1->SOL->Add(sqls);
try{
Query1 -> Open(); // выполнить команду SELECT
}
catch(EDBEngineError* dbError) // обработка ошибок BDE
{ }
}
```

Метод Open() предназначен для передачи серверу команды Select языка SQL для исполнения. Существуют другие команды SQL, например, команда UPDATE, которая обновляет содержимое некоторой записи, но не возвращает какой бы то ни было результат. Для исполнения сервером таких запросов следует использовать метод ExecSQL() вместо метода Open().

5.8. Соединения с базой данных и транзакции

Компонент TDatabase позволяет создавать локальный псевдоним базы данных, не требуя его наличия в конфигурационном файле BDE. Свойство AliasName содержит псевдоним существующей базы данных, определенный утилитой конфигурации BDE. Указание этого свойства является альтернативой значения свойства DriverName.

Свойство DatabaseName позволяет создать локальный псевдоним базы данных в дополнение к значениям AliasName или DriverName.

Свойство DriverName содержит имя драйвера BDE при создании локального псевдонима по значению DatabaseName. Указание этого свойства является альтернативой значения свойства AliasName.

Свойство Params содержит строчный массив параметров одиночного соединения (полный путь, имя сервера и т.д.).

Компонента TDatabase может использоваться для управления транзакциями. Примером транзакции является перевод денежных средств с банковских счетов. Такая транзакция состоит в добавлении суммы к новому счету и вычитании этой суммы из исходящего счета. Если выполнение любой из операций терпит неудачу, вся операция считается незавершенной. SQL-серверы дают возможность "прокручивать назад" команды при возникновении ошибки, не производя изменений в базе данных. Как правило, транзакция содержит несколько команд, поэтому начало транзакции надо отметить методом StartTransaction(). Как только транзакция началась, все ее исполняемые команды находятся во временном состоянии до тех пор, пока один из методов Commit() или Rollback() не отметит конец транзакции. Вызов Commit() фактически модифицирует данные, а вызов Rollback() отменяет всякие изменения.

Ниже приводится листинг, который реализует транзакцию по изменению адреса фирмы на примере связанных таблиц CUSTOMER и ORDERS. Старый адрес, введенный пользователем в область редактирования EditOld, заменяется на новый, введенный в область редактирования EditNew. В этом примере компонентный объект Database1 использовался для одиночного соединения с базой данных, поддерживающего выполнение одиночной транзакции. Этот объект необходимо каким-то образом связать с псевдонимом базы данных: или установкой соответствующих свойств компонента, или определив параметры соединения (такие как тип драйвера, имя сервера, имя пользователя, пароль) во время выполнения программы. Воспользуемя первым способом соединения на стадии проектирования формы приложения, установив нужные значения свойств компонента.

```
void__fastcall TForm1::Button1Click(TObject *Sender) {
char sqls[250]; // массив для хранения команды SQL
try{
Database1->StartTransaction ();
Query1->SQL->Clear () ;
// Изменить EditOld на EditNew в таблице CUSTOMER
sprintf(sqls, "UPDATE CUSTOMER SET Addrl = \"%s\" WHERE
(Addrl = \"%s\")", EditNew->Text.c_str(), EditOld->Text.c_str());
Query1->SQL->Add(sqls) ;
Query1->ExecSQL ();
Query1->SQL->Clear () ;
// Изменить EditOld на EditNew в таблице ORDERS
sprintf(sqls, "UPDATE ORDERS SET ShipToAddrl = \"%s\" WHERE
(ShipToAddrl = \"%s\")", EditNew->Text.c_str(),
```

```
EditOld->Text.c_str()) ;
Query1->SQL->Add(sqls) ;
Query1->ExecSQL();
// Внести все изменения, сделанные до этого момента
Database1->Commit();
Table1->Refresh();
Table2->Refresh();
}
catch(EDBEngineError* dbError) // обработка ошибок BDE
{
for (int i=0; i<dbError->ErrorCount; i++)
MessageBox (0, dbError[i].Message.c_str(), "SQL Error", MB_OK) ;
Database1->Rollback() ;
return;
} catch (Exception* exception) // обработка других исключений
{}
 }
```

5.9. Управление данными

Остановимся на особенностях использования компонента навигатора TDBNavigator. Нажимая на кнопки компонента First, Prior, Next и Last, можно перемещаться от записи к записи, а с помощью кнопок Insert, Delete, Edit, Post, Cancel и Refresh – производить редактирование.

Свойство DataSource соединяет кнопки управления панели навигатора с компонентами доступа к наборам данных через компонент источника. Изменяя значение этого свойства во время выполнения программы, можно использовать один и тот же компонент для навигации по разным таблицам. Например, можно разместить на форме два компонента редактируемого ввода DBEdit1 и DBEdit2, связанные с таблицами CustomersTable и OrdersTable через источники данных CustomersSource и OrdersSource, соответственно. Когда пользователь выбирает название фирмы (поле Company в DBEdit1), навигатор тоже должен соединяться с источником CustomersSource, а когда активизируется номер заказа (поле OrderNo в DBEdit2), навигатор должен переключаться на источник OrdersSource. Чтобы реализовать подобную схему работы навигатора, необходимо написать обработчик события OnEnter для одного из объектов компонента редактирования, а затем присвоить этот обработчик другому объекту.

Свойство VisibleButtons позволяет убрать ненужные кнопки, например, кнопки редактирования на форме, предназначенной для просмотра данных. Во время выполнения программы можно динамически прятать или вновь показывать кнопки навигатора – в ответ на определенные дей-

ствия пользователя или на изменения состояния приложения. Предположим, вы предусмотрели единый навигатор для редактирования таблицы CustomersTable и для просмотра таблицы OrdersTable. Когда навигатор подключается ко второй таблице, желательно спрятать кнопки редактирования Insert, Delete, Edit, Post, Cancel и Refresh, а при подключении к первой таблице – снова показать их.

СвойствоShowHint разрешает или запрещает высвечивать подсказку с названием кнопки навигатора, когда на нее наведен курсор. Значение false (устанавливается по умолчанию) запрещает подсказки для всех кнопок.

Свойство Hints содержит массив текстовых подсказок для каждой кнопки навигатора

Итак, проектирование формы приложения СУБД в среде C++Builder в простейшем случае требует выполнения следующих действий:

1. Перенесите на форму компонент TTable или TQuery со страницы Data Access и установите его свойства.

2. Перенесите на форму компонент DataSource и в свойстве DataSet укажите ссылку на объект набора данных (например, Table1 или Query1).

3. Перенесите на форму нужные компоненты отображения и редактирования данных со страницы DataControls и в их свойстве DataSource задайте источник данных (например, DataSource1). Определите отображаемое поле набора данных в свойстве DataField.

4. Если на предыдущем шаге вы выбрали компонент TDBGrid, то используйте его совместно с компонентом навигатора TDBNavigator.

Вопросы

1. Что представляет собой псевдоним БД и как он создается?

2. Как создать таблицу с помощью программы Database Desktop?

Какие другие средства для создания таблиц можно использовать?

3. Какие компоненты используются для связи таблиц БД с компонентами визуализации и управления данными DBGrid, DBEdit?

4. Как связать компоненты Table и Query с нужной таблицей БД ?

5. Как связать компонент DBNavigator с нужной таблицей БД ?

6.Что такое первичные и вторичные индексы для таблицы и как их создать?

7. Что такое SQL и из каких частей он состоит?

8. Что означают следующие SQL-запросы:

а) SELECT name, projectname FROM emploee, project
WHERE empno=team_leader;

б) SELECT name, salary FROM emploee, prohibit
WHERE salary>2900;

9. Как использовать SQL-запрос в C++Builder ?

Упражнения

В следующих упражнениях создать указанные таблицы и записать SQL запрос для выборки данных из обеих таблиц.

1. Построить головную таблицу «Телефонный справочник» с полями: Номер телефона, Адрес, а также вспомогательную таблицу «Население города» с полями: Номер телефона Фамилия И.О., Год рождения, Пол, Адрес.

2. Головная таблица содержит данные о подразделениях предприятия: Отдел – номер, название(например, Цех1, Бухгалтерия), тип отдела (например, Управление, Производство и т. д.). Вспомогательная таблица содержит сведения о сотрудниках: ID, Фамилия И.О., Год рождения, Пол, Номер_Отдела. Связать таблицы по полю Номер_Отдела.

3. БД содержит следующие связанные таблицы: «Сведения о покупателях»:ID, Фамилия И.О., адрес, Номер банковского счёта, номер заказа и «Сведения о заказах»: Наименование товара, ID, Количество, Стоимость, Дата заказа.

4. БД содержит следующие связанные таблицы: «Студенты»: Номер зачётной книжки, Фамилия, Имя, Отчество, Специализация, Курс, Группа; «Учебный план»: Специализация, Курс, Предмет1, Предмет2,…; «Журнал успеваемости»: Фамилия, Имя, Отчество, Предмет 1, Предмет 2, …

5. БД «Абитуриенты» содержит следующие связанные таблицы: «Сведения об абитуриентах»: Номер личной карточки, Фамилия, Имя, Отчество, Факультет, Специальность,; «Сведения о сдаваемых предметах»: Специальность, Предмет; «Сведения об оценках»: Фамилия, Имя, Отчество, Номер личной карточки, Предмет, Оценка.

6. БД «Автомобили» содержит следующие таблицы: «Общие сведения об автомобиле»:ID, Марка, Тип, Производитель; «Характеристика автомобиля»: Марка, Мощность двигателя, Тип двигателя, Стоимость.

7. БД «Лабораторные занятия» содержит следующие таблицы: «Преподаватели»: ID, Фамилия, Имя, Отчество, Предмет, Курс, Группа; «Студенты»:ID, Фамилия, Имя, Отчество, Курс, Группа; «Пропуски занятий»: Фамилия, Имя, Отчество, Курс, Группа, Предмет, Пропущено.

8. БД «Конференция» содержит следующие таблицы: «Расписание заседаний»:Номер, Название секции, Дата, Время; «Сведения об участниках»: Фамилия И.О., Название секции, Название доклада; «Оргработа»: Фамилия, Имя, Отчество, Дата приезда, Дата отъезда, Потребность в гостинице, Оргвзнос.

9. БД «Библиотека» содержит следующие таблицы: «Общая характеристика единицы хранения»: Инвентарный номер, Тип издания (журнал, книга, рукопись), Название; «Характеристика издания»: Тип издания, Авторы, Название, Издательство, Год издания, Номер, Количество страниц.

10. БД «Магазин» содержит следующие таблицы: «Товары»: Артикул, Наименование товара, Количество, Дата поставки, Цена; «Поставщики»: Название организации, Наименование товара, Количество, Дата поставки, Адрес организации;
«Покупатели»: Название организации, Наименование товара, Количество, Дата покупки, Адрес покупателя.

11. БД «Поликлиника» содержит следующие таблицы: «Врачи»: Участок, Фамилия И.О., Специальность; «Пациенты»:ID, Фамилия И. О., Диагноз, Возраст, Участок.

12. БД «Спортивная база» содержит следующие таблицы: «Тренеры»:ID, Фамилия, Имя, Отчество, Вид спорта, Секция; «Спортсмены»: ID, Фамилия, Имя, Отчество, Вид спорта, Секция, Рейтинг, Возраст.

6. СЕТЕВЫЕ ПРОГРАММЫ И СОКЕТЫ

Понятие «сокет» (socket) означает "гнездо", "разъем" по аналогии с гнездами на аппаратуре. В соответствии с этой аналогией, можно связать два "гнезда" соединением и передавать между ними данные. Каждое гнездо принадлежит определённому хосту (host – хозяин, держатель). Каждый хост имеет уникальный IP (Internet Packet) адрес, представляющий группу из четырех чисел, разделенных точками.

Переданная по IP адресу на хост информация поступает на один из портов хоста. Порт определяется числом от 0 до 65535. После того, как сокет установлен, он имеет вполне определённый адрес, записывающийся так [host]:[port]. Например, 127.0.0.1:8888 означает, что сокет занимает порт 8888 на хосте 127.0.0.1(на данном компьютере). Чтобы не использовать труднозапоминаемый IP адрес, для доступа к хостам используется система имен DNS (DNS – Domain Name Service), поддерживаемая специальным сервером. Цель этой системы – сопоставлять IP адресам сим-

вольные имена. Например, адресу "127.0.0.1" в большинстве компьютеров соответствует имя "localhost", что означает сам компьютер, на котором выполняется программа.

Сокеты – это абстракция, представляющая узлы соединения двух приложений - клиента и сервера. К этим узлам подключаются клиенты и серверы через соединение, которое можно представить как гипотетический кабель. Соединения с приложением-сервером. всегда устанавливают приложения-клиенты. В обязанность приложения-сервера входит прослушивание клиентов и ожидание соединения. После этого клиенты посылают серверу или получают от сервера сообщения в виде последовательности символов, а в конце работы закрывают соединение.

Для организации сетевых соединений (сокетов) в C++ Builder используются классы TClientSocket и TServerSocket из группы компонентов Internet. Ниже описывается работа с компонентом TClientSocket по установке соединения.

Определение свойств Host и Port. Поместим компонент TClientSocket на форму. Чтобы установить соединение, нужно присвоить свойствам Host и Port компонента TClientSocket значения, соответствующие адресу сервера. Host – это символьное имя компьютера-сервера, с которым надо соединиться (например: localhost, nitro.borland.com или mmf410-2), либо его IP-адрес (например: 127.0.0.1, 192.168.0.88). Port – номер порта (от 1 до 65535) на данном хосте для установления соединения. Обычно номера портов выбираются, начиная с 1001 (номера меньше 1000 могут быть заняты системными службами, например, по умолчанию POP – 110, Http – 80).

Открытие сокета. После назначения свойствам Host и Port соответствующих значений, можно приступить к открытию сокета. Для этого нужно присвоить свойству Active значения true. Здесь полезно вставить обработчик исключительной ситуации на случай, если соединиться не удается. Открытие сокета можно выполнить, также с помощью метода ClientSocket->Open();

Авторизация. На этом этапе вы посылаете серверу свой логин (имя пользователя) и пароль. Этот пункт можно пропустить, если сервер не требует ввода логинов или паролей.

Посылка/прием данных – это, собственно и есть то, для чего открывалось сокетное соединение;

Закрытие сокета – после выполнения операций необходимо закрыть сокет, присвоив свойству Active значение false или вызовом метода ClientSocket->Close();.

Свойства и методы компонента TClientSocket. На рис. 20 и 21 показаны свойства и события компонента TClientSocket в инспекторе объектов. В табл. 18 приведены их краткие описания.

Рис. 20. Свойства компонента TClientSocket

Рис. 21. События компонента TclientSocket

Пример программы-клиента на основе сокета. Поместим на форму компонент TClientSocket, две кнопки Button1 и Button2, два окна типа TEdit и два компонента Memo1 и Memo2. При нажатии на кнопку GetFromServeSendToClient вызывается обработчик события OnClick – Button1Click(). Перед этим в Edit1 нужно ввести хост-имя, а в Edit2 – порт удаленного компьютера. Когда TClientSocket должен прочитать информацию из сокетного соединения, возникает событие OnRead и вызывается функция ClientSocketRead().

Таблица 18

Свойства	События
Active – *True* – сокет открыт, а *False*–закрыт. **Host** – строка (типа String), указывающая на имя компьютера, к которому следует подключиться. **Address** – строка (типа String), указывающая на IP-адрес компьютера, к которому следует подключиться. Если вы укажете в **Host** символьное имя компьютера, то IP адрес будет запрошен у DNS. **Port** – номер порта (от 1 до 65535), к которому следует подключиться. **Service** – строка, определяющая службу (ftp, http, pop и т.д.), к порту которой произойдет подключение. **ClientType** – тип соединения ctNon-Blocking или ctBlocking	**OnConnect** – возникает при установлении соединения. В обработчике события можно начинать авторизацию или прием/передачу данных. **OnConnecting** – возникает при установлении соединения, но соединение еще не установлено. Обычно такие промежуточные события используются для обновления статуса. **OnDisconnect** – возникает при закрытии сокета из вашей программы либо из-за сбоя. **OnError** – возникает при ошибке в работе сокета. Операторы открытия сокета следует заключить в блок try..catch. **OnLookup** – возникает при попытке получения от DNS IP-адреса хоста. **OnRead** – возникает, когда удален-ный компьютер послал данные. **OnWrite** – возникает, когда вам разрешена запись данных в сокет.

На рис. 22 показана форма приложения в процессе проектирования.

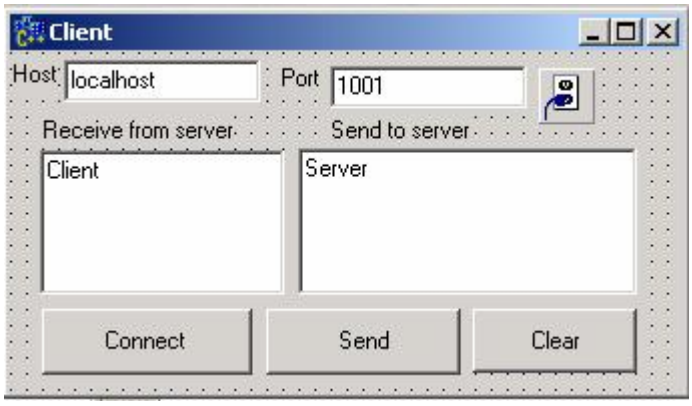

Рис. 22. Форма приложения-клиента

Ниже приводятся коды приложения-клиента.

```
TForm1 *Form1;
//---------------------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
}
//открытие клиента и установка соединения
void __fastcall TForm1::Button1Click(TObject *Sender)
```

```
{
 TClientSocket *cs = Form1->ClientSocket;  // формируем указатель  cs->Close();
// закрытие соединения(если слушали порт ранее)
 try
 {
  cs->Port=Form1->PortEdit->Text.ToInt();//получение номера порта
  cs->Host = HostEdit->Text;  // получения адреса хоста
  cs->Active = true;        // установка соединения
 }
 catch(...)
 {
  Memo1->Text = "Some problem with connection";// если введены
                                                //неверные значения
 }
}
//очистка окна
void __fastcall TForm1::ClearClick(TObject *Sender)
{
 Memo1->Clear();
}
// считывание клиентом сообщения
void __fastcall TForm1::ClientSocketRead(TObject *Sender, TCustomWinSocket
*Socket)
{
 Memo1->Text = Socket->ReceiveText(); // запись сообщения в поле клиента
}
void __fastcall TForm1::ClientSocketWrite(TObject *Sender,
     TCustomWinSocket *Socket)
{
Socket->SendText(Memo2->Text); //
}
//вызов события посылки сообщения
 void __fastcall TForm1::Button2Click(TObject *Sender)
{
  Form1->ClientSocketWrite(Form1,Form1->ClientSocket->Socket);
}
```

Ниже описываются методы свойства Socket компонента TClient-Socket. Свойство Socket в свою очередь является объектом класса TCustomWinSocket.

SendText(String) – посылка текстовой строки через сокет.

SendStream() – посылка содержимого указанного потока через сокет. Пересылаемый поток должен быть открыт.

SendBuf(Buf, Count) – посылка буфера через сокет. Буфером может являться любой тип, например, структура или простой тип *int*. Буфер

указывается параметром Buf, вторым параметром необходимо указать размер пересылаемых данных в байтах (Count).

ReceiveText() – получить сообщение.

Программирование серверов на основе сокетов. Следует заметить, что для сосуществования отдельных приложений клиента и сервера не обязательно иметь несколько компьютеров. Достаточно иметь лишь один, на котором можно одновременно запустить и сервер, и клиент. При этом в качестве имени компьютера, к которому надо подключиться, нужно использовать хост-имя *localhost* или IP-адрес – 127.0.0.1.

Сервер, основанный на сокетном протоколе, позволяет обслуживать сразу множество клиентов. Причем, ограничение на их количество вы можете указать сами (или убрать это ограничение, как сделано по умолчанию). Для каждого подключенного клиента сервер открывает отдельный сокет, по которому можно обмениваться данными с клиентом. Другим решением является создание для каждого подключения отдельного процесса (Thread).

Создание сервера включает следующие шаги:

Определение свойств Port и ServerType – чтобы к серверу могли подключаться клиенты, порт, используемый сервером, должен совпадать с портом, используемым клиентом. Свойство ServerType определяет тип подключения.

Открытие сокета – открытие сокета и указанного порта. Осуществляется установкой свойства ServerSocket->Active=true. Автоматически начинается ожидание подсоединения клиентов.

Подключение клиента и обмен данными с ним – клиент устанавливает соединение и начинается обмен данными с ним.

Отключение клиента – клиент отключается и закрывается его сокетное соединение с сервером.

Закрытие сервера и сокета – по команде администратора сервер завершает свою работу, закрывая все открытые сокетные каналы и прекращая ожидание подключений клиентов.

Далее приводится краткое описание компонента TServerSocket. На рис. 23 и 24 приводятся свойства и события компонента, соответственно, которые отображаются в окне Инспектора объектов. Табл. 19 содержит описание указанных свойств и событий.

Свойство Socket компонента TServerSocket. Как же сервер может отсылать данные клиенту или принимать данные? Если вы работаете через события OnClientRead и OnClientWrite, то общаться с клиентом можно через свойство Socket (TCustomWinSocket). Отправка/посылка данных через свойство Socket класса TServerSocket аналогична работе клиента(

методы SendText(), SendBuf(), SendStream(), ReceiveText()). Однако, следует выделить некоторые полезные свойства и методы, характерные для для сервера: ActiveConnections (*Integer*) – количество подключенных клиентов; ActiveThreads (*Integer*) – количество работающих процессов; Connections (*array*) – массив, состоящий из отдельных классов TClientWinSocket для каждого подключенного клиента. Например, такая команда

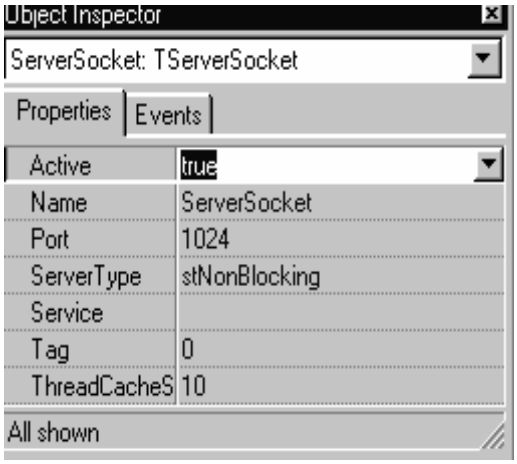

Рис 23. Свойства компонента TServerSocket

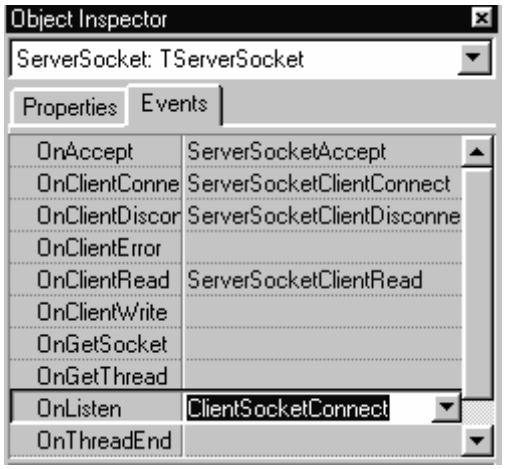

Рис. 24. События компонента TServerSocket

ServerSocket1->Socket->Connections[i]->SendText(“Hello!”);

отсылает i-тому подключенному клиенту сообщение “Hello!”); IdleThreads (*Integer*) – количество свободных процессов (такие процессы кэшируются сервером, см. свойство ThreadCacheSize); LocalAddress, LocalHost, LocalPort – соответственно, локальный IP-адрес, хост-имя, порт; RemoteAddress, RemoteHost, RemotePort – соответственно, удаленный IP-

адрес, хост-имя, порт; методы Lock и UnLock – соответственно, блокировка и разблокировка сокета.

Таблица 19

Свойства	События
Socket – класс TServerWinSocket, через который вы имеете доступ к открытым сокетным каналам. **ServerType** – тип сервера. **ThreadCacheSize** – количество клиентских процессов (Thread), которые будут кэшироваться сервером. Кэширование происходит для того, чтобы не создавать каждый раз отдельный процесс и не уничтожать закрытый сокет, а оставить их для дальнейшего использования. Тип: int. **Active** –Значение *True* указывает на то, что сервер работает и готов к приему клиентов, а *False* – сервер выключен. **Port** – номер порта для установления соединений. Порт у сервера и у клиентов должны быть одинаковыми. Рекомендуются значения от 1025 до 65535. **Service** – строка, определяющая службу (ftp, http, pop и т.д.), порт которой будет использован. Тип: String.	**OnClientConnect** – возникает, когда клиент установил сокетное соединение и ждет ответа сервера. **OnClientDisconnect** – возникает, когда клиент отсоединился . **OnClientError** – возникает, когда операция завершилась неудачно. **OnClientRead** – возникает, когда клиент передал серверу данные. **OnClientWrite** – возникает, когда сервер может отправлять данные клиенту по сокету. **OnGetSocket** – в обработчике этого события можно отредактировать параметр ClientSocket. **OnGetThread** – в обработчике события можно определить уникальный процесс (Thread) для каждого клиента канала, присвоив параметру SocketThread нужную подзадачу TServerClientThread. **OnThreadStart**, **OnThreadEnd** – возникает, когда процесс Thread запускается или останавливается. **OnAccept** – возникает, когда сервер принимает клиента или отказывает ему в соединении. **OnListen** – возникает, когда сервер переходит в режим ожидания подсоединения клиентов.

Пример. Создадим приложение-сервер. На форму нужно поместить кнопки Button1 и Button2, поле Edit1 и компоненты Memo1 и Memo2. При создании формы вызывается обработчик события OnCreate (FormCreate), в котором активизируется сервер. Свойство компонента TServerSocket->Active устанавливается в true, что означает, что сокетное соединение открыто и доступно для коммуникации с клиентами. В компоненте Edit1 указано значение прослушиваемого порта. Когда ServerSocket1 должен записать информацию в ClientSocket1, возникает событие OnClientWrite и вызывается функция ServerSocketClientWrite(). На рис. 25 приведена форма приложения в процессе проектирования.

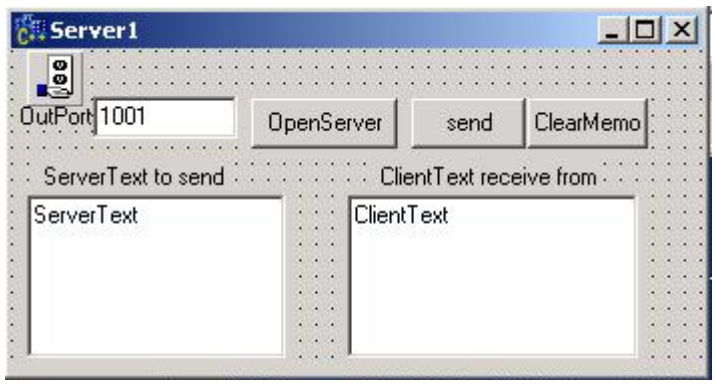

Рис. 25. Форма приложения-сервера

Ниже приводится листинг приложения.

```
void __fastcall TForm1::FormCreate(TObject *Sender)
{
 TServerSocket *ServerSocket = Form1->ServerSocket; // указатель на //сервер сокет
 ServerSocket->Close(); // закрытие соединения(если слушали порт //Ранее)
 ServerSocket->Port = Form1->Edit1->Text.ToInt(); // получение номера //порта
 ServerSocket->Active = true;    //прослушивание данного порта
}
//открытие сервера  по нажатию кнопки
void __fastcall TForm1::Button2Click(TObject *Sender)
{
ServerSocket->Close();                // закрытие соединения
ServerSocket->Port=Form1->Edit1->Text.ToInt(); //получение  порта
ServerSocket->Active = true;         //прослушивание данного порта
}
//---------------------------------------------------------------------------
void __fastcall TForm1::ServerSocketClientWrite(TObject *Sender,
     TCustomWinSocket *Socket)  //запись сообщения
{
 Socket->SendText(Memo1->Text);// отправка сообщения
}
void __fastcall TForm1::Button1Click(TObject *Sender)
{
 Memo1->Clear();
}
//---------------------------------------------------------------------------
void __fastcall TForm1::ServerSocketClientRead(TObject *Sender,
     TCustomWinSocket *Socket)
{
Memo2->Text = Socket->ReceiveText();
}
//отправка сообщения i-тому клиенту
void __fastcall TForm1::Button3Click(TObject *Sender)
{
```

```
int n=ServerSocket->Socket->ActiveConnections;
for(int i=0;i< n ;i++)
ServerSocket->Socket->Connections[i]->SendText(
Memo1->Text+IntToStr(i));
}
```

Ниже рассматриваются некоторые приемы работы с компонентом TServerSocket.

Хранение уникальных данных для каждого клиента. Если сервер будет обслуживать множество клиентов, то потребуется хранить информацию для каждого клиента (имя и др.), причем с привязкой этой информации к сокету данного клиента. В некоторых случаях делать все это вручную (привязка к конкретному сокету, массивы клиентов и т.д.) не очень удобно. Поэтому для каждого сокета существует специальное свойство Data для хранения указанной информации.

Посылка файлов через сокет. Рассмотрим посылку файлов через сокет. Достаточно открыть этот файл как файловый поток (TFileStream) и отправить его через сокет (SendStream). Поясним это на примере:

```
ServerSocket1->Socket->Connections[i]->SendStream(srcfile);
```

Нужно заметить, что метод SendStream() используется не только сервером, но и клиентом.

Передача блоков информации. Посылаемые через сокет данные могут не только объединяться в один блок, но и разъединяться по нескольким блокам. Дело в том, что через сокет передается обычный поток, но в отличие от файлового потока (TFileStream) он передает данные медленнее (сеть, ограниченный трафик и т.д.). Именно поэтому две команды:

```
ServerSocket1->Socket->Connections[0]->SendText(“Hello,”);
ServerSocket1->Socket->Connections[0]->SendText(“world!”);
```

совершенно идентичны одной команде:

```
ServerSocket1->Socket->Connections[0]->SendText(“Hello, world!”);
```

Поэтому, если отправить через сокет файл, скажем, в 100 Кб, то получателю блока придет несколько блоков с размерами, которые зависят от трафика и загруженности линии. Причем, размеры не обязательно будут одинаковыми. Отсюда следует, что для того, чтобы принять файл или любые другие данные большого размера, следует принимать блоки данных, а затем объединять их в одно целое (и сохранять, например, в файле). Другим решением данной задачи является тот же файловый поток TFileStream (либо поток в памяти TMemoryStream). Принимать блоки данных из сокета можно через событие OnRead (OnClientRead), используя универсальный метод ReceiveBuf(). Определить размер полученного блока можно методом ReceiveLength().

Приведем еще один простой пример приложений для создания клиента и сервера в одном приложении, соединения между ними и обмена данными. Ниже приводится соответствующий программный код.

```
//-- Клиент --------------------------------------------------------------
void __fastcall TForm1::ClientSocketRead(TObject *Sender,
  TCustomWinSocket *Socket)
{
 Edit1->Text = Socket->ReceiveText();}
void __fastcall TForm1::Button1Click(TObject *Sender)
{ ClientSocket->Close();
 if((Edit1->Text.Length() > 0)&&(Edit2->Text.Length() > 0)) {
  ClientSocket->Host = Edit2->Text;
  ClientSocket->Port = Edit1->Text.ToInt();
  ClientSocket->Open();
}
}
//-- Сервер--------------------------------------------------------------
void __fastcall TForm1::FormCreate(TObject *Sender){
 ServerSocket->Close(); // закрытие соединения
 ServerSocket->Active = true; } //активизация сокета
//---------------------------------------------------------------------------
void __fastcall TForm1::FormDestroy(TObject *Sender){
 ServerSocket->Close(); }
//---------------------------------------------------------------------------
void __fastcall TForm1::Button1Click(TObject *Sender){
 ServerSocket->Socket->Connections[0]->SendText(Edit1->Text);
 }
```

Потоки. Для создания распределенных приложений, работающих с несколькими клиентами, можно воспользоваться сокетным потоком. Потоки позволяют синхронизировать работу нескольких клиентов. Приведем пример создания многопотокового сервера и клиентской программы. По каждому запросу клиента сервер создаёт поток и определённое время ожидает ответа клиента. Потоки создаваемые сервером, это потомки класса TserverClientThread. Поэтому мы будем объявлять собственный потоковый класс, наследуя его от TServerClientThread. Ниже приводится код программы с комментариями.

```
//Unit1.h- классы для сервера
#ifndef Unit1H
#define Unit1H
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
#include <ScktComp.hpp>
```

```
//---------------------------------------------------------------------------
class TForm1 : public TForm
{
__published:    // IDE-managed Components
        TServerSocket *ServerSocket1;
        TMemo *Memo1;
        TButton *Button1;
        TButton *Button2;
        TMemo *Memo2;
        TLabel *Label1;
        TLabel *Label2;
        void __fastcall ServerSocket1GetThread(TObject *Sender,
         TServerClientWinSocket *ClientSocket, TServerClientThread
*&SocketThread);
        void __fastcall Button1Click(TObject *Sender);
        void __fastcall Button2Click(TObject *Sender);
private:          // User declarations
public:           // User declarations
        __fastcall TForm1(TComponent* Owner);
};
extern PACKAGE TForm1 *Form1;
//---------------------------------------------------------------------------
// Потоки в сервере наследуются от TServerClientThread.
// Мы  будем объявлять потоковый класс  следующим образом:
class PACKAGE TMyServerThread :
public Scktcomp::TServerClientThread {
  public:
/*  перед тем как завершить поток, устанавливаем FreeOnTerminate в false, и
поток останется в кэше. При установке KeepInCache в false,  после завершения вы-
полнения потока, он будет удалён. */
__fastcall TMyServerThread(bool CreateSuspended, TServerClientWinSocket* ASocket)
     : Scktcomp::TServerClientThread(CreateSuspended, ASocket)
         { CreateSuspended = false;
        KeepInCache=true;
         FreeOnTerminate=false; };
/* Чтобы включить поток, переопределяем метод ClientExecute(),
 вызываемый из метода Execute() класса TServerClientThread.*/
        void __fastcall ClientExecute(void);
};
#endif
//Реализация сервера-------------------------------------------------
// ***************       Server-Unit1.cpp        ***************
//  Requires: TServerSocket,   TMemo1, Tmemo2,
//Button1-ServerOpen, Button2-Exit
#include <vcl.h>
#pragma hdrstop
#include "Unit1.h"
```

```
        #pragma package(smart_init)
        #pragma resource "*.dfm"
        TForm1 *Form1;
        //---------------------------------------
        __fastcall TForm1::TForm1(TComponent* Owner) : TForm(Owner)
        {  }
const int WAITTIME = 60000;//Будем ждать клиента 60с
int k=0;
const int BUFSIZE = 32;//Размер буфера чтения/записи
/*Вместо компоненты клиентского сокета  поток сервер-клиент должен использовать
объект ClientSocket класса TServerClientWinSocket, который создаётся, когда ожи-
дающий сокет сервера выполнит клиентское соединение.*/
 //Перегрузка метода ClientExecute() класса TServerClientThread
void __fastcall TMyServerThread::ClientExecute(void) {
   // убеждаемся, что соединение активно
while (!Terminated && ClientSocket->Connected){
  try {//используем TWinSocketStream для чтения/записи  через
       // блокирующий сокет
  TWinSocketStream *pStream=new TWinSocketStream(ClientSocket, WAITTIME);
       try        {
         char buffer[BUFSIZE];
         memset( buffer, 0, sizeof(buffer) );
  if (pStream->WaitForData(WAITTIME)) {
// даём клиенту 60с для начала записи
    if (pStream->Read(buffer, sizeof(buffer)) == 0)
ClientSocket->Close();
//если не удаётся прочитать через 60с, закрываем соединение
             else {   //помещение текста клиента в Memo1 сервера
Form1->Memo1->Lines->Add(String("(Client) ")+String(buffer) );    strcpy(buffer,Form1-
>Memo2->Text.c_str());//из Memo2 в buffer
pStream->Write( buffer, sizeof(buffer));}//Возвращаем буфер клиенту
                 }
           else ClientSocket->Close();//если не передается текст
                }
                __finally      {   delete pStream;   }
    }
/* Для обработки исключений  используется HandleException(). */
              catch (...)       {    HandleException();   }
           }
        }

void __fastcall TForm1::ServerSocket1GetThread(TObject *Sender,
TServerClientWinSocket *ClientSocket, TServerClientThread *&SocketThread)
 { //событие создает поток  при открытии нового клиентского сокета
 //метод возвращает поток через SocketThread параметр
 // Рекомендуется использовать Thread-bloking сервер
   SocketThread = new TMyServerThread(false, ClientSocket);
```

```
 k++;
 Memo1->Clear();
 Memo1->Text="Открыт клиент:" +IntToStr(k);
 }
void __fastcall TForm1::Button1Click(TObject *Sender)
{
ServerSocket1->Close();
ServerSocket1->Open();
}
void __fastcall TForm1::Button2Click(TObject *Sender)
{
Close();
}

//Создание клиента-----------------------------------------------
#ifndef Unit2H
#define Unit2H
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
#include <ScktComp.hpp>
//---------------------------------------------------------------------------
// ******************        ClientMain.h        ******************
class TForm2 : public TForm {
__published: // IDE-managed Components
  TClientSocket *ClientSocket1;
  TButton *Button1;
  TButton *Button2;
  TMemo *Memo1;
      TButton *Button3;
      TLabel *Label1;
      TMemo *Memo2;
      TLabel *Label2;
  void __fastcall Button1Click(TObject *Sender);
  void __fastcall Button2Click(TObject *Sender);
  void __fastcall ClientSocket1Read(TObject *Sender,
       TCustomWinSocket *Socket);
  void __fastcall ClientSocket1Write(TObject *Sender,
       TCustomWinSocket *Socket);
      void __fastcall Button3Click(TObject *Sender);
private: // User declarations
  AnsiString Server;
public:  // User declarations
  __fastcall TForm2(TComponent* Owner);
};
extern PACKAGE TForm2 *Form2;
```

```
//---------------------------------------------------------------------------
#endif

//Реализация клиента-------------------------------------------------
// *****************         Client-Unit2.cpp         *****************
//  Requires:  2 TButtons, 2 TMemo, TClientSocket
//---------------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop
#include "Unit2.h"
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm2 *Form2;
//---------------------------------------------------------------------------
__fastcall TForm2::TForm2(TComponent* Owner)
        : TForm(Owner)
{  }
//---------------------------------------------------------------------------
void __fastcall TForm2::Button1Click(TObject *Sender) {
  if (ClientSocket1->Active)
     ClientSocket1->Active = false;
  if (InputQuery("Computer to connect to", "Address Name:", Server))    {
     if (Server.Length() > 0)       {
        ClientSocket1->Host       =  Server;
        ClientSocket1->Active     =  true;       }
  }
}
void __fastcall TForm2::Button2Click(TObject *Sender) {
  ClientSocket1->Active = false;  }
void __fastcall TForm2::ClientSocket1Read(TObject *Sender, TCustomWinSocket
*Socket)
{    if (ClientSocket1->Active == true)
     if (Socket->Connected == true)
        Memo1->Lines->Add( AnsiString("(Server) ") +Socket->ReceiveText() ); }
void __fastcall TForm2::ClientSocket1Write(TObject *Sender, TCustomWinSocket
*Socket) {
  if (ClientSocket1->Active == true)
     if (Socket->Connected == true)
        Socket->SendText("This text is passed to"+Memo2->Text); }
void __fastcall TForm2::Button3Click(TObject *Sender)
{
Close();
}
```

Вопросы

1. Какие свойства используются при создании приложения-сервера с использованием компонента TserverSocket?

2. Какие свойства используются при создании приложения-клиента с использованием компонента TclientSocket?

3. С помощью какого метода можно получить данные с сервера по инициативе клиента ?

4. С помощью какого метода клиент может переслать текстовые данные на сервер ?

5. Куда в приложении-клиенте поступают данные, пересылаемые от сервера?

6. Какие методы используются для передачи информации от клиента серверу?

7. С помощью какого метода можно получить данные с сервера по инициативе клиента?

8. Как организовать постоянное отслеживание информации на сервере в приложении-клиенте?

9. Какое событие в приложении-сервере может быть использовано для определения того, что от клиента серверу послана информация?

10. Какие способы установления контакта с сервером Вы знаете?

11. Что нужно сделать, чтобы при запуске приложения-клиента автоматически запускалось и приложение-сервер?

Упражнения

1. Организовать пересылку текстовых сообщений между клиентом и сервером в обоих направлениях с постоянным отслеживанием информации.

2. Реализовать передачу координат курсора мыши между клиентом и сервером в обоих направлениях с постоянным отслеживанием координат.

3. Реализовать простейший “чат”.

4. Реализовать игру в “слова”, например по очереди называть названия городов.

5. Реализовать игру «крестики/нолики» в сети.

6. Реализовать игру «морской бой» в сети.

7. СОЗДАНИЕ СОБСТВЕННЫХ КОМПОНЕНТОВ

Процесс разработки собственного компонента (назовем его TMyComp) состоит из следующих этапов:

1. Создание модуля для нового компонента.
2. Наследование производным классом для нового компонента уже существующего базового класса библиотеки VCL.
3. Добавление нужных свойств, событий и методов.
4. Регистрация компонента в среде C++Builder.
5. Отладка.
6. Инсталляция компонента на Палитру компонентов.
7. Сохранение файлов компонента.

Мастер компонентов способен выполнить автоматически создание файлов модуля, наследование компонентного класса, объявление нового конструктора и регистрацию компонента. Разработчик компонента "вручную" или с помощью имеющихся средств добавляет свойства, события, методы и устанавливает компонент на Палитру компонентов. Рассмотрим подробнее действия, которые выполняются на каждом из перечисленных этапов.

7.1. Создание модуля компонента

Программный модуль компонента состоит из двух файлов MyComp.cpp и MyComp.h, которые компилируются в объектный файл MyComp.obj. При разработке компонента либо создают новый модуль, либо модифицируют существующий. Чтобы создать модуль, необходимо выполнить команду File|New и в открывшемся диалоговом окне New Items выбрать значок Unit. Чтобы добавить компонент к существующему модулю, нужно выполнить команду File|Open и в открывшемся диалоговом окне выбрать файл имя_модуля.cpp для модуля, в который будет добавлен компонент. Далее формируется заготовка файла MyComp.h:

```
#ifndef MyCompH
#define MyCompH
#include <SysUtils.hpp>
#include <Classes.hpp>
#include <Controls.hpp>
#include <ExtCtrls.hpp>
//class TMyComponent : public < базовый компонентный класс >
class PACKAGE TMyComponent : public < базовый класс >
{
public:
__fastcall TMyComponent(TComponent *Owner);
```

```
};
#endif
```

Объявление нового класса компонента должно включать макрос PACKAGE. Функционально сходные компоненты группируются в пакеты – динамические библиотечные файлы с расширением .bpl. Как и обычные dll-библиотеки, пакеты содержат код, разделяемый многими приложениями. Пакетная организация ускоряет компиляцию и сборку приложений, позволяет получить более эффективные исполняемые коды за счет того, что к выполняемой программе VCL уже не загружается целиком.

Созданный компонент пока не отличается от своего родителя.

7.2. Наследование компонента

Любой компонент является производным от базового класса TComponent и его наследников (таких как TControl или TGraphicControl) или от существующего компонентного класса.

Простейший способ построить новый компонент – это изменить свойства существующего компонента. Для этого можно использовать любой абстрактный класс VCL, в название которого входит слово Custom. Например, можно создать новый компонент списка со специальными свойствами, которых нет в стандартном классе TListBox. Поскольку нельзя непосредственно модифицировать TListBox, то можно начать с его ближайшего предшественника в иерархии классов. Для этой цели подходит класс TCustomListBox, который реализует все мыслимые свойства производных компонентов списка, однако не выставляет всех их в секции **_published.** Наследуя компонент, нужно объявить в разделе **_published** те свойства, которые требуется включить в создаваемый компонент**.**

Все оконные компоненты являются производными от базового класса TWinControl. Стандартный элемент оконного управления характеризует так называемый "оконный дескриптор" (window handle), который заключен в свойстве Handle. Благодаря дескриптору, Windows идентифицирует данный компонент, в частности, она может принять фокус ввода. Хотя можно создать оригинальный оконный интерфейсный элемент (который не имеет существующих аналогов и никак не связан с ними), используя в качестве базового класса TWinControl, C++Builder предоставляет класс TCustomControl – специализированный оконный элемент управления, который упрощает рисование сложных визуальных изображений. Все компоненты стандартного оконного управления: кнопки, списки, текстовые поля (за исключением компонента TLabel, который никогда не при-

нимает фокус ввода) – являются косвенными производными TWinControl.

В отличие от оконных компонентов, производных от класса TCustomControl, графические компоненты лишены оконного дескриптора и не могут принять фокус ввода. Графические компоненты обеспечивают отображение объектов без использования системных ресурсов.

Новые графические компоненты необходимо наследовать от базового абстрактного класса TGraphicControl (потомка TControl). Класс TGraphicControl предоставляет канву для рисования и обрабатывает сообщения WM_PAINT. Все, что нужно сделать, это переопределить метод рисования Paint в соответствии с заданными требованиями.

Все компоненты имеют общий абстрактный базовый класс TComponent, однако, только невидимые компоненты можно наследовать непосредственно от класса TComponent. Невидимые компоненты используются в качестве интерфейсных элементов с другими компонентами (доступ к базам данных или диалоговым окнам).

7.3. Добавление свойств, событий и методов

Свойства, в отличие от данных-членов, сами не хранят данные, однако методы чтения и записи организуют к ним доступ. Об этом необходимо помнить при создании или изменении компонентных свойств.

Событие – это связь между некоторым воздействием на компонент и кодом обработчика события, который реагирует на это воздействие. Обработчик события пишется прикладным программистом. Используя события, программист может приспособить поведение компонента к своим требованиям без необходимости изменения самих объектов. События, возникающие в результате типичных действий пользователя (например, движений мышью) встроены во все стандартные компоненты VCL, однако вы можете определить новые события. C++Builder реализует события как свойства.

7.4. Регистрация компонента

Регистрация компонента – это процесс, который информирует C++Builder о том, какой компонент добавляется к VCL и на какой вкладке Палитры компонентов он должен появиться. Для регистрации компонента выполняются следующие действия:

1) Добавляется функция Register() в файле MyComp.cpp. При этом функция заключается в пространство имен с именем, совпадающим с

именем модуля компонента (при этом первая буква должна быть заглавной, а все остальные – строчными).

2) В теле функции Register объявляется массив типа TComponentClass, в который вводится имя класса регистрируемого компонента.

3) В теле функции Register вызывается функция RegisterComponents с тремя параметрами: названием вкладки Палитры компонентов, массивом компонентных классов и индексом последнего класса в этом массиве.

Например:

```
namespace Mycomp {
void __fastcall PACKAGE Register()
{
TComponentClass classes[1] = {_classid(TMyComponent)};
RegisterComponents("Samples", classes, 0);}
}
```

Листинг представляет включение в файл MyComp.cpp кода для регистрации компонентf TMyComponent на вкладке **Samples** Палитры компонентов.

Когда компонента зарегистрирована, можно ее испытать и инсталлировать на Палитру компонентов.

7.5. Отладка неинсталлированного компонента

Поведение компонента следует проверить до инсталляции на Палитру компонентов. По существу необходимо смоделировать действия, которые производит C++Builder, когда пользователь перемещает компонент из Палитры компонентов на форму. Процесс требует выполнения следующих шагов:

1) Включить файл модуля MyComp.h компонента в заголовочный файл некоторой формы: #include " MyComp.h "

2) Добавить объектный член данных, представляющий испытываемый компонент, в раздел объявлений public класса формы: TMyComponent *MyComponent1;.

3) Подсоединить обработчик к событию *OnCreate* формы.

4) Вызвать конструктор компонентного объекта из обработчика этого события, передав ему параметр, указывающий на владельца компонента. Обычно это указатель **this** на объект, который содержит компонент (например, форма).

5) Сразу же за вызовом конструктора установить свойство Parent – родителя компонента. Обычно значением этого свойства является указа-

тель **this.** Если компонент не является элементом управления, т.е. не наследовался от TControl, то этот шаг пропускается.

6) Инициировать значения других свойств компонента.

Ниже приводится листинг модуля формы, используемой для отладки компонента:

```
// Заголовочный файл TestForm.h
#ifndef TestFormH
#define TestFormH
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
#include "MyComp.h" // 1
class TForm1 : public TForm {
__published:
private:
public:
TMyComponent * MyComponent1; // 2
__fastcall TForm1 (TComponent* Owner); // 3
};
extern PACKAGE TForm1 *Forml;
#endif

// Файл TestForm.cpp модуля формы
#include <vcl.h>
#pragma hdrstop
#include "TestForm.h"
#pragma package(smart_init)
pragma resource "*.dfm"
TForm1 *Forml;
__fastcall TForm1::TForm1(TComponent* Owner) : TForm(Owner)
{
 MyComponent1 = new TMyComponent(this); // 4
 MyComponent1->Parent = this; // 5
 MyComponentl->Left = 12; // 6
}
```

Для отладки компонента необходимо выполнить компиляцию и запустить тестовое приложение. Если поведение компонента соответствует предъявляемым к ему требованиям, то можно перейти к его установке на Палитру компонентов.

7.6. Инсталляция компонента на Палитру компонентов

Чтобы добавить к VCL компонент, необходимо выполнить следующие шаги:

1) С помощью команды Component|Install Component открывается диалоговое окно инсталляции компонент. Можно выбрать вкладку Into New Package, если компонент включается в новый пакет, или вкладку Into existing package, если используется существующий пакет.

2) Вводится имя модуля компонента MyComp.cpp и путь к нему в поле Unit file name. Далее вводится имя нового пакета (совпадающее с названием вкладки Палитры компонентов, на которую устанавливается новый компонент) в поле Package file name, а его краткое описание – в поле Package description. Щелчок на кнопке ОК закрывает диалоговое окно установки. Добавление компонента в существующий пакет выполняется аналогично.

3) С текущим содержимым пакета можно ознакомиться вь открывшемся окне Менеджера пакетов. Все файлы, составляющие рассматриваемый пакет, будут созданы (или перестроены), и новый компонент установлен на ту вкладку Палитры компонентов, которая была задана в качестве одного из параметров функции регистрации Register. C++Builder автоматически включает в проект системные файлы периода выполнения (с расширениями .bpi, .res), необходимые для сборки пакета.

4) Выбирается команда Install в Менеджере пакетов. В результате выполняется компиляция, перестройка VCL и установка нового компонента на Палитру компонентов.

5) Командой Component|Install Packages или Project|Options открывается список установленных пакетов. По щелчку на кнопке Components можно увидеть список всех компонентов, включенных в выбранный пакет. Если нужно, чтобы текущая конфигурация пакетов принималась по умолчанию в каждом новом создаваемом проекте, то необходимо установить флажок Default.

Чтобы удалить компонент из VCL, необходимо выполнить следующие действия:

1) Выполнить команду Component|Install Component**,** которая открывает диалоговое окно установки компонентов.

2) Найти удаляемый компонентный класс в списке Component classes выбранной группы Библиотеки и нажать кнопку Remove.

3) Нажать кнопку ОК. Библиотека будет перестроена и компонент удален из Палитры компонентов.

Чтобы перестроить Палитру компонентов, необходимо выполнить следующие действия:

1) Открыть диалог установки опций Палитры компонентов с помощью команд Component|Configure Palette или Options|Environment|Palette.

2) Нажать кнопку Add и выбрать имя для новой вкладки. Имя добавленной вкладки появится внизу списка Pages названий вкладок.

3) Перетащить мышью выбранный компонент в списке Components на нужную вкладку списка Pages.

4) Нажать кнопку ОК. Библиотека и Палитра компонентов будут перестроены.

7.7. Сохранение файлов нового компонента

После окончания процесса разработки, компонент будет представлен следующими файлами:

объектный файл результата компиляции MyComp.obj;

файлы модуля компонента (MyComp.h, MyComp.cpp);

файлы пакета с именем имя_пакета и расширениями bpl, bpk, lib, bpi, cpp, res;

файл пиктограммы компонента для Палитры компонентов MyComp.dcr;

файл формы MyComp.dfm, если компонент использует форму;

желательно создать и сохранить контекстно-справочный файл MyComp.hlp.

По умолчанию C++Builder сохраняет компонентный пакетный файл с расширением bpl в каталоге \...\ProgramFiles\Borland\CBuilder6\ Projects\Bpl, а библиотечные файлы с расширениями lib и bpi – в каталоге \...\ProgramFiles\Borland\CBuilder6\Projects\Lib. Используя новый компонент в дальнейшем, нельзя забывать проверять вкладку Directories|Conditionals диалогового окна команды Project|Options, где должен быть указан путь к библиотечным каталогам.

8. ПРИМЕР РАЗРАБОТКИ ПРОСТОГО КОМПОНЕНТА

Перед тем, как приступить к разработке компонент, необходимо знать, что он должен делать и как будет реализовано его оригинальное поведение. Рассмотрим пример компонента, моделирующего бинарный индикатор, который меняет цвет при изменении состояния. Некоторое свойство компонента будет хранить его текущее состояние (true – индикатор включен, и false – в противном случае). Мы уже знаем, что желательно выбрать для наследования наиболее близкий в иерархии VCL базовый компонентный класс. Очевидно, что индикатор представляет собой графический компонент семейства TGraphicControl. Поскольку мы разрабатываем простой компонент, пусть он будет иметь форму круга, а

не более хитроумный образ. Компонент TShape из вкладки Палитры компонентов Additional выглядит ближайшим родственным компонентом. При внимательном рассмотрении Tshape мы видим, что он имеет больше свойств и событий, чем нам требуется. Все что мы хотим изменить при наследовании от TShape – это форму индикатора и цвет кисти при его переключении. Перейдем к фактической разработке компонента:

1. Создание формы тестового приложения

Пока мы не убедились, что разрабатываемый компонент работает, его нельзя включать в VCL. Сначала следует создать тестовое приложение для формы с прототипом нового компонента MyComp:

а) С помощью команды File|New Application создайте пустую форму.

б) Разместите кнопку TButton на форме.

в) С помощью команды File|Save All сохраните форму и проект приложения в файлах под именами MyCompForm.cpp и MyCompProj.bpr.

2. Создание модуля компонента

Мастер компонентов (Component Wizard) упрощает начальные шаги создания компонента:

а) Выполните команду Component|New и в открывшемся диалоговом окне Мастера компонентов заполните поля диалога указанными на рис. 26 значениями. Нажмите кнопку ОК.

б) С помощью команды File|Save As сохраните файл Unitl.cpp под именем MyComp.cpp.

Файл MyComp.cpp будет содержать пустой конструктор объекта и функцию Register для регистрации компонента. Созданный при этом файл MyComp.h будет содержать объявление нового компонентного класса с конструктором, а также несколько заголовочных файлов предкомпиляции.

3. Члены данных, свойства и методы

Прежде всего, в файле MyComp.h опишем булеву переменную состояния индикатора и две переменные перечисляемого типа TColor для хранения цветов, отображающих оба состояния. Члены данных следует поместить в разделе **private** объявлений класса. Там же расположим прототипы методов записи соответствующих свойств, а сами свойства объявим в разделе **_published**.

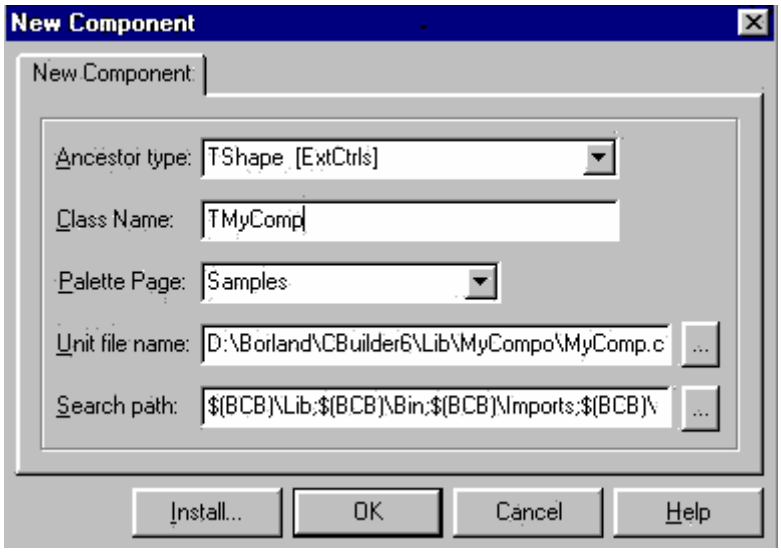

Рис. 26. Окно Мастера компонентов

Ниже приводится заголовочный файл модуля компонента:

```
// MyComp.h ----------------------------------------------------------
#ifndef MyCompH
#define MyCompH
#include <SysUtils.hpp>
#include <Classes.hpp>
#include <Controls.hpp>
#include <ExtCtrls.hpp>
//---------------------------------------------------------------------------
class PACKAGE MyComp : public TShape
{
private:
bool FOnOff;
 TColor FOnColor;
 TColor FOffColor;
void __fastcall SetOnOff(const bool Value) ;
void __fastcall  SetOnColor(const TColor OnColor);
void __fastcall SetOffColor (const TColor OffColor) ;
protected:
public:
    __fastcall MyComp(TComponent* Owner);
__published:
__property bool OnOff= { read= FOnOff,write= SetOnOff} ;
__property TColor OnColor= { read=FOnColor,write=SetOnColor} ;
__property TColor OffColor={ read=FOff Color, write= SetOffColor} ;
};
//---------------------------------------------------------------------------
```

```
#endif
```

Для добавления в файл MyComp.cpp необходимо написать три функции для присваивания значений свойств соответствующим членам данных и наполнить конструктор компонента инструкциями для инициализации переменных. Ниже приводится листинг файла MyComp.cpp после внесения изменений:

```
//MyComp.cpp
#include <vcl.h>
#pragma hdrstop
#include "MyComp.h"
#pragma package(smart_init)
//---------------------------------------------------------------------------
// ValidCtrCheck is used to assure that the components created
// do not have any pure virtual functions.
static inline void ValidCtrCheck(MyComp *)
{       new MyComp(NULL); }
void __fastcall MyComp::SetOnOff(const bool Value) {
FOnOff = Value;
Brush->Color= (FOnOff) ? FOnColor : FOffColor ;
}
void __fastcall MyComp::SetOnColor(const TColor OnColor) {
FOnColor = OnColor;
Brush->Color = (FOnOff) ? FOnColor : FOffColor;
}
void __fastcall MyComp::SetOffColor(const TColor OffColor) {
FOffColor = OffColor;
Brush->Color = (FOnOff) ? FOnColor : FOffColor;
}
//---------------------------------------------------------------------------
__fastcall MyComp::MyComp(TComponent* Owner)
      : TShape(Owner)
{ Width = 15; //ширина по умолчанию
Height = 15; // высота по умолчанию
FOnColor = clLime; // зеленый, когда включен
 FOffColor = clRed; // красный, когда выключен
FOnOff = false; // выключен по умолчанию
Shape = stEllipse; //в форме эллипса по умолчанию
Pen->Color = clBlack; // черный контур по умолчанию
 Pen->Width = 2; // ширина контура по умолчанию
 Brush->Color = FOffColor; // цвет заливки по умолчанию
}
namespace Mycomp
{
      void __fastcall PACKAGE Register()
      {
            TComponentClass classes[1] = {__classid(MyComp)};
```

```
        RegisterComponents("Samples", classes, 0);
    }
}
//---------------------------------------------------------------------------
```

Установленные конструктором значения членов данных по умолчанию появятся в окне Инспектора объектов при создании объекта. Действительно, при помещении компоненты на форму конструктор вызывается автоматически. В результате появляется возможность менять значения свойств компонента не только во время выполнения программы, но и на стадии проектирования приложения.

4. Испытание компонента

С помощью команды File|Save All сохраните все сделанные добавления.

Выбрав вкладку MyCompForm.cpp в окне Редактора кода включите строку #include "MyComp. h" в заголовочный файл формы. Добавьте описание объекта:

```
MyComp* MyComp1;
```

Это можно сделать в разделе **public** в файле MyCompForm.h.

Активизируйте форму Form1 и в окне Инспектора объектов дважды щелкните мышью в графе значений события OnCreate. С помощью Редактора кода введите обработчик этого события в файл MyCompForm.cpp. Следующий код создаст компонент MyComp динамически (определяя ее родителя Parent и помещая в центре родительской формы) во время выполнения тестового приложения:

```
void__fastcall TForm1::FormCreate(TObject *Sender)
{MyComp1 = new MyComp(this);
MyComp1->Parent = this;
// Центрировать компонент по ширине формы
MyCompl->Left = (Width/2)-(MyCompl->Width/2);
// Центрировать компонент по высоте формы
MyCompl->Top = (Height/2)-(MyCompl->Height/2);
}
```

Чтобы кнопка управляла индикатором, дважды щелкните мышью в графе значений события OnClick объекта Button1 в окне Инспектора объектов. С помощью Редактора кода введите следующую инструкцию в тело обработчика события:

```
void __fastcall TForm1::Button1Click(TObject *Sender) (
MyComp1->OnOff = !MyComp1->OnOff;
}
```

Скомпилируйте и запустите тестовое приложение командой Run|Run.

Если компилятор не выдаст ошибок, то в центре формы тестового приложения появится красный индикатор в состоянии "выключен". Нажав кнопку, вы включите индикатор и он окрасится зеленым цветом.

Осталось создать битовый образ пиктограммы, которой новый компонент будет представлена в Палитре компонентов. Из меню редактора изображений, открывающегося по команде Tools|Image Editor**,** выберите File|New|Resource File, а затем – Resource|New|Bitmap. В диалоге свойств битового образа установите размеры пиктограммы 24×24 и число цветов VGA (16 Colors). Переименуйте битовый образ компонента (MyComp) по команде Resourse|Rename и дважды щелкните мышью на выбранном имени в древовидном списке ресурсных файлов, чтобы нарисовать подходящую картинку индикатора (например, зеленый кружок). Командой File|Save As сохраните ресурсный файл MyComp.res в рабочем каталоге и закройте Редактор изображений.

5. Инсталляция компонента

Перед тем, как приступить к инсталляции нового компонента на Палитру компонентов, выполните еще раз команду File|Save All.

С помощью команды Component|Install Component откройте диалоговое окно инсталляции компонентов. Нажмите кнопку Add**,** которая открывает диалоговое окно добавления модуля. Найдите местоположение модуля MyComp.cpp, нажав на кнопку поиска Browse. Нажмите кнопку OK и ждите окончания перестройки VCL и установки нового компонента на Палитру компонентов.

6. Проверка работы

Выполните команду File|Close All**,** а затем File|New Application. Поместите новый компонент MyComp и кнопку TButton на форму. Снова определите обработчик события OnClick кнопки управления индикатором:

```
void __fastcall TForm1::Button1Click(TObject *Sender) {
MyCompl->OnOff = !MyCompl->OnOff;
}
```

Выполните команду Run|Run и вы увидите, что компонент действительно работает (рис. 27).

Рис. 27. Работа компонента

Вопросы и упражнения

1. Опишите схему разработки собственных компонентов.
2. Как добавить свойства и события в проектируемый компонент ?
3. Как выполнить регистрацию компонента в среде C++Builder ?
4. Как выполняется предварительная отладка созданного компонента ?
5. Как поместить разработанный компонент на Палитру компонентов ?
6. Создайте усовершенствованный ползунок (базовый компонент TTrackBar с вкладки Win32 Палитры компонентов), в котором прорезь ползунка можно изобразить в виде тонкой линии. Добавить свойство, которое позволяет выводить ползунок как в привычной форме, так и с прорезью в виде тонкой линии.

Литература

1. *Шамис, В.А.* Borland C++Builder 6. Для профессионалов/*В. А. Шамис.* СПб.:Питер, 2003
2. *Архангельский, А.Я.* Программирование в C++Builder 6/*А. Я. Архангельский.* М.:ЗАО «Издательство БИНОМ», 2003
3. *Глушаков, С.В.* Программирование в среде Borland C++Builder 6/ *С. В. Глушаков, В. Н. Зорянский, С. Н. Хоменко.* Харьков:Фолио, 2003
4. *Сурков, К.А.* Программирование в среде C++Builder/ *К. А. Сурков, Д. А. Сурков, А. Н. Вальвачев.* Мн.:ООО»Попурри», 1998

СОДЕРЖАНИЕ

Введение 3

1. C++BUILDER и объектно-ориентированное программирование 18

2. Компоненты библиотеки VCL 27

3. Строки и потоки ввода/вывода в C++BUILDER 40

4. Поддержка графики и графические компоненты 65

5. Работа с базами данных 74

6. СЕТЕВЫЕ ПРОГРАММЫ И СОКЕТЫ 96

7. Создание собственных компонентов 112

8. Пример разработки простого компонента 118

Учебное издание

Романчик Валерий Станиславович
Люлькин Аркадий Ефимович

ПРОГРАММИРОВАНИЕ В C++ BUILDER

Учебное пособие
по курсу «МЕТОДЫ ПРОГРАММИРОВАНИЯ»
для студентов специальностей
G31 03 01 «Математика», G31 03 03 «Механика»

Редактор
Технический редактор
Корректор

Компьютерная верстка *А.Е. Люлькин*

Подписано в печать __.__.2006. Формат 60×84/16. Бумага офсетная. Печать офсетная.
Усл.печ.л. ____. Уч.-изд.л. . Тираж 100 экз. Зак.

Белорусский государственный университет.
Лицензия на осуществление издательской деятельности №02330/0056804 от 02.03.2004.
220050, Минск, проспект Независимости,4.

Отпечатано с оригинала-макета заказчика.
Издательский центр «Белорусского государственного университета».
Лицензия на осуществление полиграфической деятельности №02330/0056850 от 30.04.2006.
220030, Минск, ул. Красноармейская, 6.