Министерство образования и науки, молодежи и спорта Украины
Севастопольский национальный технический университет

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

Лабораторная работа №1

«Основы работы в интегрированной среде разработки *C++Builder* на примере создания приложения простейшего текстового редактора»

по дисциплине

«**Персональные компьютеры в вычислительной практике**»

для студентов дневной и заочной формы обучения

по направлению 6.050901 — Радиотехника.

Севастополь
2013 г.

УДК.519.72

Методические указания Лабораторная работа №1 «Основы работы в интегрированной среде разработки *C++Builder* на примере создания приложения простейшего текстового редактора» по дисциплине «Персональные компьютеры в вычислительной практике» для студентов дневной и заочной форм обучения по направлению 6.050901 — Радиотехника / Сост. А.В. Лукьянчиков — Севастополь: Изд-во СевНТУ, 2013. — 10 с.

Методические указания рассмотрены и утверждены на заседании кафедры радиотехники и телекоммуникаций (протокол № _ от __ ____ 2013 г.)

Допущено учебно-методическим центром СевНТУ в качестве методических указаний.

Рецензент: канд. техн. наук, доцент Савочкин А.А.

Ответственный за выпуск: заведующий кафедрой радиотехники и телекоммуникаций, доктор техн. наук, профессор Гимпилевич Ю.Б.,

1. ЛАБОРАТОРНАЯ РАБОТА № 1
ОСНОВЫ РАБОТЫ В ИНТЕГРИРОВАННОЙ СРЕДЕ РАЗРАБОТКИ *C++BUILDER* НА ПРИМЕРЕ СОЗДАНИЯ ПРИЛОЖЕНИЯ ПРОСТЕЙШЕГО ТЕКСТОВОГО РЕДАКТОРА

1.1. Цель работы: На примере простейшего приложения «текстовый редактор» понять принцип создания приложений под операционную систему *WINDOWS* в интегрированной среде разработки *C++ Builder.*

1.2. Теоретические сведения

Borland C++ Builder — это средство быстрой разработки, позволяющее создавать приложения на языке *C++*.

C++ Builder представляет собой *SDI*-приложение, главное окно которого (см. рис. 1.1) содержит настраиваемую инструментальную панель и палитру компонентов. Помимо этого, по умолчанию при запуске *C++ Builder* появляются окно инспектора объектов и форма нового приложения. Под окном формы приложения находится окно редактора кода.

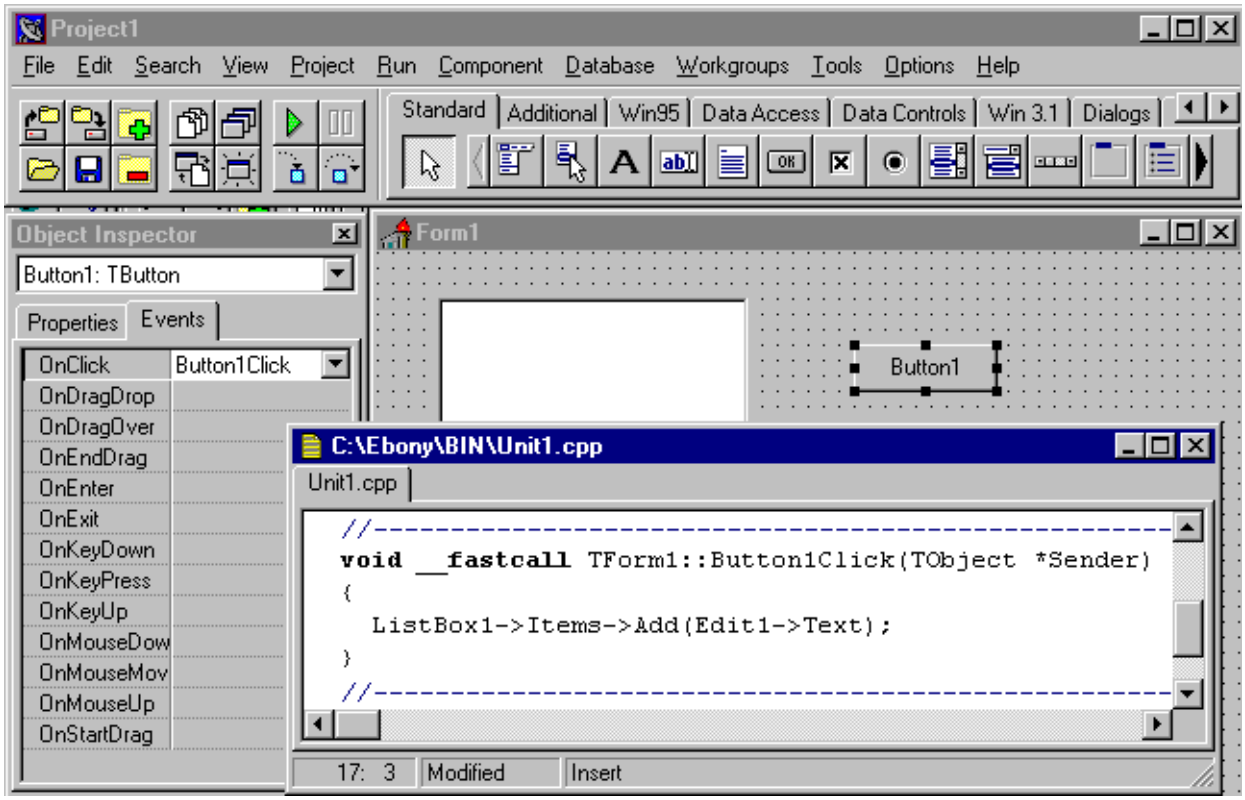

Рис.1.1 — Среда разработки *C++ Builder*

Формы являются основой приложений *C++ Builder*. Создание пользовательского интерфейса приложения заключается в добавлении в окно формы элементов объектов *C++ Builder*, называемых компонентами. Компоненты *C++ Builder* располагаются на палитре компонентов, выполненной в виде многостраничного блокнота. Важная особенность *C++ Builder* состоит в том, что он позволяет создавать собственные компоненты и настраивать палитру компонентов, а также создавать различные версии палитры компонентов для разных проектов.

Компоненты *C++ Builder*. Компоненты разделяются на видимые (визуальные) и невидимые (невизуальные). Визуальные компоненты появляются во время выполнения точно так же, как и во время проектирования. Примерами являются кнопки и редактируемые поля. Невизуальные компоненты появляются во время проектирования как пиктограммы на форме. Они никогда не видны во время выполнения, но обладают определенной функциональностью.

Каждый компонент *C++ Builder* имеет три разновидности характеристик: свойства, события и методы.

Если выбрать компонент из палитры и добавить его к форме, инспектор (см. рис. 1.3) объектов автоматически покажет свойства и события, которые могут быть использованы с этим компонентом. В верхней части инспектора объектов имеется выпадающий список, позволяющий выбирать нужный объект из имеющихся на форме.

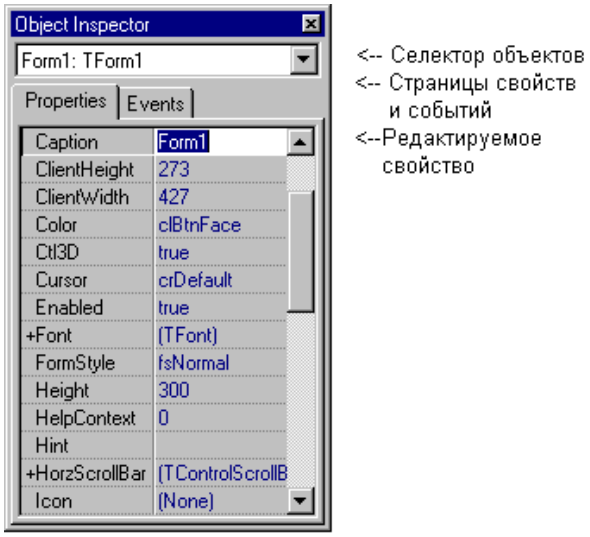

Рис.1.2 — Инспектор объектов

Свойства компонентов. Свойства являются атрибутами компонента, определяющими его внешний вид и поведение. Многие свойства компонента в колонке свойств имеют значение, устанавливаемое по умолчанию (например, высота кнопок). Свойства компонента отображаются на странице свойств (*Properties*). Инспектор объектов используется для установки свойств во время проектирования. Список свойств располагается на странице свойств инспектора объектов. Можно определить свойства во время проектирования или написать код для видоизменения свойств компонента во время выполнения приложения.

При определении свойств компонента во время проектирования нужно выбрать компонент на форме, открыть страницу свойств в инспекторе объектов, выбрать определяемое свойство и изменить его с помощью редактора свойств (это может быть простое поле для ввода текста или числа, выпадающий список, раскрывающийся список, диалоговая панель и т.д.).

События. Страница событий (*Events*) инспектора объектов показывает список событий, распознаваемых компонентом. Каждый компонент имеет свой собственный набор обработчиков событий. В *C++ Builder* следует писать функции, называемые обработчиками событий, и связывать события с этими функциями. Создавая обработчик того или иного события, вы поручаете программе выполнить написанную функцию, если это событие произойдет.

Для того, чтобы добавить обработчик событий, нужно выбрать на форме с помощью мыши компонент, которому необходим обработчик событий, затем открыть страницу событий инспектора объектов и дважды щелкнуть левой клавишей мыши на колонке значений рядом с событием, чтобы заставить *C++ Builder* сгенерировать прототип обработчика событий и показать его в редакторе кода. При этом автоматически генерируется текст пустой функции, и редактор открывается в том месте, где следует вводить код. Курсор позиционируется внутри операторных скобок {...}. Далее нужно ввести код, который должен выполняться при наступлении события. Обработчик событий может иметь параметры, которые указываются после имени функции в круглых скобках.

Методы. Метод является функцией, которая связана с компонентом, и которая объявляется как часть объекта. Создавая обработчики событий, можно вызывать методы, используя следующую нотацию: →, например:

*Edit*1 → *Show*();

Необходимо понимать, что при создании формы связанные с ней модуль и заголовочный файл с расширением *.*h* генерируются обязательно, тогда как при создании нового модуля он не обязан быть связан с формой (например, если в нем содержатся процедуры расчетов). Имена формы и модуля можно изменить, причем желательно сделать это сразу после создания, пока на них не появилось много ссылок в других формах и модулях.

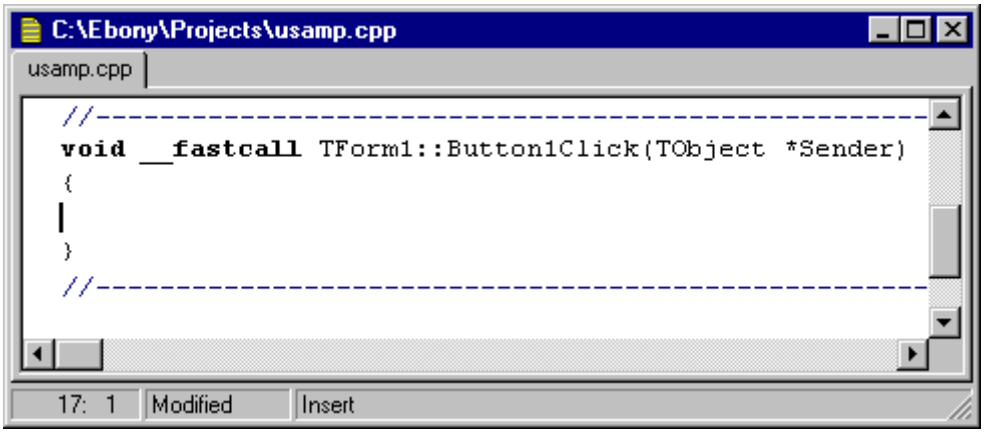

Рис.1.3 — Прототип обработчика событий

Менеджер проектов. Файлы, образующие приложение — формы и модули — собраны в проект. Менеджер проектов показывает списки файлов и модулей приложения и позволяет осуществлять навигацию между ними. Можно вызвать менеджер проектов, выбрав пункт меню *View*/*Project Manager*. По умолчанию вновь созданный проект получает имя *Project*1.*cpp*.

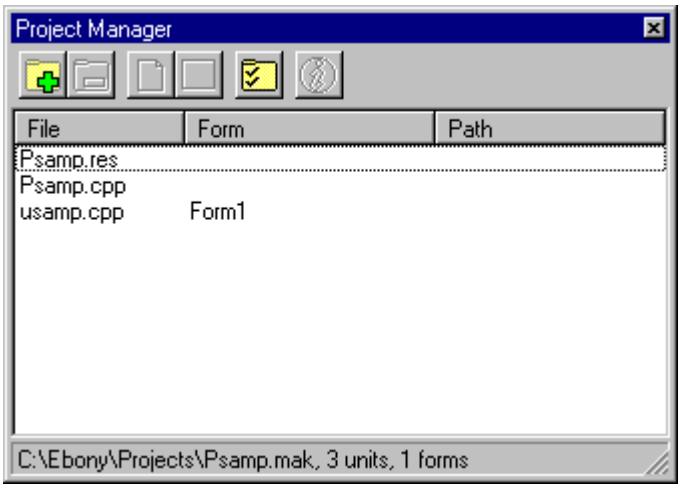

Рис.1.4 — Менеджер проектов

По умолчанию проект первоначально содержит файлы для одной формы и исходного кода одного модуля. Однако большинство проектов содержат несколько форм и модулей. Чтобы добавить модуль или форму к проекту, нужно щелкнуть правой кнопкой мыши и выбрать пункт *New Form* из контекстного меню. Можно также добавлять существующие формы и модули к проекту, используя кнопку *Add* контекстного меню менеджера проектов и выбирая модуль или форму, которую нужно добавить. Формы и модули можно удалить в любой момент в течение разработки проекта. Однако, из-за того, что форма связана всегда с модулем, нельзя удалить одно без удаления другого, за исключением случая, когда модуль не имеет связи с формой. Удалить модуль из проекта можно, используя кнопку *Remove* менеджера проектов.

Если выбрать кнопку *Options* в менеджере проектов, откроется диалоговая панель опций проекта, в которой можно выбрать главную форму приложения, определить, какие формы будут создаваться динамически, каковы параметры компиляции модулей (в том числе созданных в *Delphi* 2.0, так как *C++ Builder* может включать их в проекты) и компоновки.

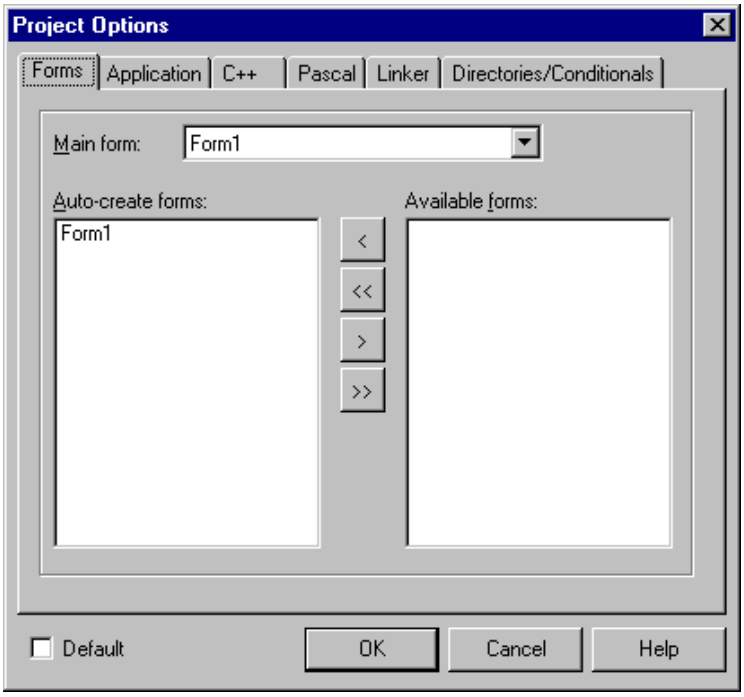

Рис.1.5 — Установка опций проекта

Важным элементом среды разработки *C++ Builder* является контекстное меню, появляющееся при нажатии на правую клавишу мыши и предлагающее быстрый доступ к наиболее часто используемым командам.

C++ Builder обладает встроенной системой контекстно-зависимой помощи, доступной для любого элемента интерфейса и являющейся обширным источником справочной информации о *C++ Builder*.

Создание приложений в C++ *Builder*. Первым шагом в разработке приложения *C++ Builder* является создание проекта. Файлы проекта содержат сгенерированный автоматически исходный текст, который становится частью приложения, когда оно скомпилировано и подготовлено к выполнению. Чтобы создать новый проект, нужно выбрать пункт меню *File*/*New Application*.

C++ Builder создает файл проекта с именем по умолчанию *Project*1.*cpp*, а также *make*-файл с именем по умолчанию *Project*1.*mak*. При внесении изменений в проект, таких, как добавление новой формы, *C++ Builder* обновляет файл проекта.

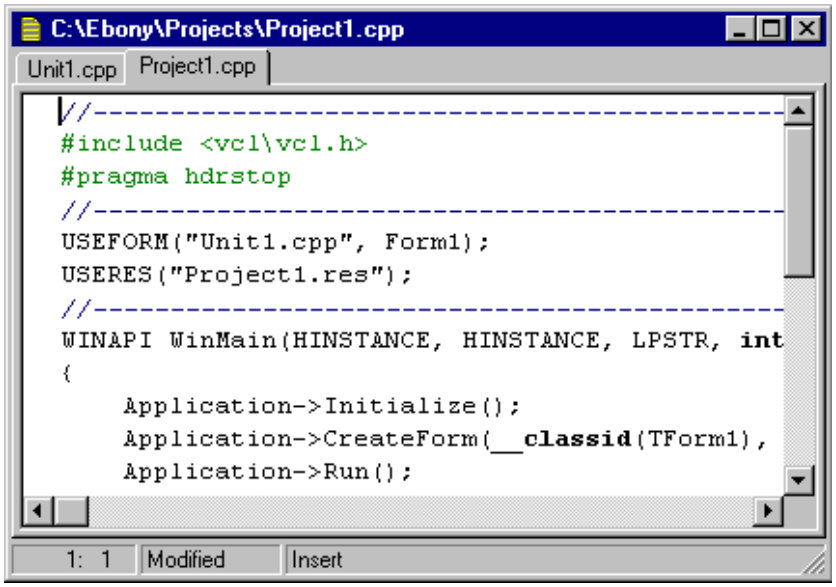

Рис.1.6 — Файл проекта

1.3. Методические рекомендации к выполнению работы

Новый проект создается автоматически при запуске *C++ Builder*. Также можно выбрать команду *File->New Application* или открыть "Хранилище новых объектов" командой *File->New*.

Рассмотрим создание приложение для решение квадратного уравнения.

Перетащите на форму с помощью левой кнопки мыши компоненты *TLabel*  , *TEdit*  и *TButton*  и разместите, как показано на рисунке 1.7 (чтобы разместить сразу несколько компонентов, нажмите на его кнопку при нажатой клавише *Shift*). Чтобы модифицировать размеры компонентов, нужно ввести соответствующие значения в свойства *Top*, *Left*, *Width*, *Height* инспектора или мышью навести на угол компонента, и перетащить его.

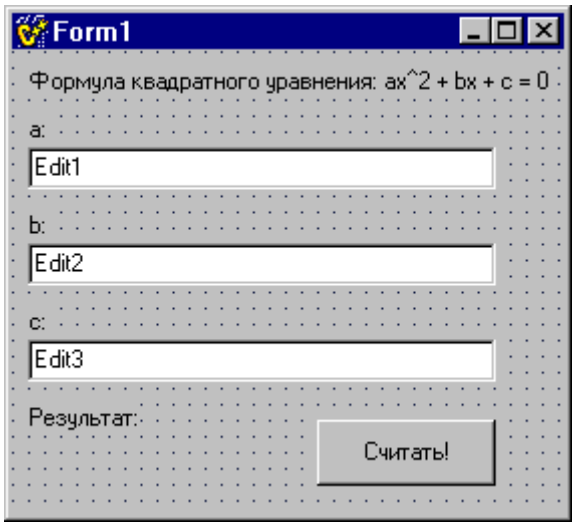

Рис. 1.7 — Вид главного окна приложения

Для изменения имени выбранного компонента используйте свойство *Name*.

Чтобы модифицировать надписи этих компонентов, нужно изменить свойство *Caption* в Инспекторе объектов. Для определения обработчика события *OnClick*(), которое возникает при нажатии на объект кнопкой мыши, можно пойти двумя путями: ввести имя или выбрать его из списка доступных в правой вкладке Инспектора, или просто два раза щелкнуть по кнопке. После этого в Редакторе откроется место для ввода кода, оно изображено на рисунке 1.8.

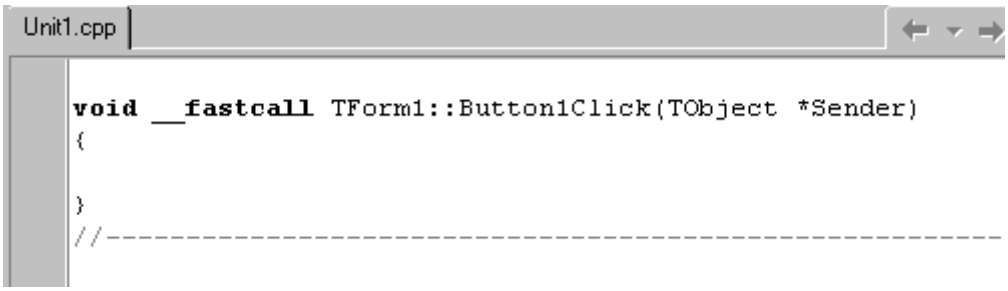

Рис. 1.8 — Заготовка обработчика события нажатия на кнопку

Это, собственно, и есть процедура обработки события. Сюда нужно ввести следующие строчки:

```
#include <math.h>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  double a,b,c;
  a=Edit1->Text.ToDouble();
```

```
  b=Edit2->Text.ToDouble();
  c=Edit3->Text.ToDouble();
  int d=b*b+4*a*c;
  if (d<0){
        Label6->Caption="Нет корней!!!";
        return;
  };
  double x1=(-b+sqrt(d))/(2*a);
  double x2=(-b-sqrt(d))/(2*a);
  Label6-<Caption=String(x1)+"   ;\r\n   "+String(x2);
}
```

Здесь, *Edit*1, *Edit*2 и *Edit*3 — соответствующие области ввода, а *Label*6 — надпись, здесь невидимая, поскольку с нулевым текстом, а в программе выводящая результат. Свойство *Text* объектов класса*TEdit* содержит введенную строку. Программа выводит полученные корни или сообщает, что их нет.

Пример обладает рядом недостатков — программа не контролирует, вводятся числовые или символьные данные и при ошибке просто "вылетает". Но для общего ознакомления подходит.

В этом примере использованы три класса компонентных объектов. Ниже приведены наиболее необходимые и специфичные свойства объектов этих классов. Одинаковые свойства не приводятся по два раза. Про общие свойства и иерархию *VCL* можно найти информацию в [1].

Компонент *TLabel*

Caption — Определяет надпись, выводимую объектом

Align — Выравнивание метки

AutoSize — Автоматическое масштабирование метки по длине введенного текста

Enabled — Разрешена ли метка

Color — Цвет метки

Font — Стандартное свойство типа *TFont*. Содержит вложенные свойства, которые означают именно то, как они называются. Можно выбрать тип шрифта из открывающегося по двойному щелчку диалога.

ParentFont — Определяет, нужно ли использовать шрифт родителя по умолчанию.

Transparent — Прозрачность фона метки.

ShowAccelChar — Можно ли использовать "быструю клавишу", которая определяется символом после символа "&" и переводит фокус ввода на компонент, определенный в свойстве *FocusControl*.

WordWrap — Автоматический перенос слов, если они не умещаются на строке.

Компонент *TEdit*

Anchors — Свойство типа множество, которое определяет, как будет изменяться длина компонента при изменении размеров формы.

BorderStyle — Окантовка области ввода.

Color — Цвет фона.

ReadOnly — Запрещает редактирование введенного текста.

PasswordChar — Отображение одного и того же символа (например, звездочки) вместо вводимого текста.

Text — Содержит введенную строку типа *AnsiString*. Вообще, этот тип является стандартным для *VCL* и будет подробно рассмотрен в дальнейшем.

Компонент *TButton*

OnClick — Событие, возникающее при нажатии на кнопку клавишей мыши.

Cancel — Говорит о том, что при нажатии клавиши *ESC* или закрытии дилогового окна используется обработчик события *OnClick*() данной кнопки.

Default — Кнопка выбирается по умолчанию и обводится рамкой.

Caption — Подпись на кнопке.

ModalResult — В модальных диалоговых окнах значение этого свойства, не равное *mrNone*, при нажатии на кнопку закроет окно и запишет свое значение в свойство *ModalResult* формы.

PopupMenu — Контекстное меню, выходящее при нажатии на компонент правой кнопкой мыши.

Теперь рассмотрим пример создания простейшего текстового редактора.

Сначала необходимо создать новое приложение и конструкторе появится пустая форма. Затем надо перетащите компоненты *TMainMenu* и *TMemo* на форму. Также с вкладки *Dialogs* необходимо добавить компоненты *TOpenDialog* и *TSaveDialog*.

Далее необходимо отредактировать меню. Для этого нужно два раза щелкнуть по компоненту *TMainMenu*. Откроется Дизайнер меню. Введите названия пунктов в свойство *Caption*. Для создания нового пункта необходимо нажать *Enter*. Чтобы создать горячую клавишу, в *Caption* перед ней поставьте символ "&". Чтобы создать подменю, жмите *Ctrl*+"->".

В результате должно получится меню как изображено на рисунке 1.9.

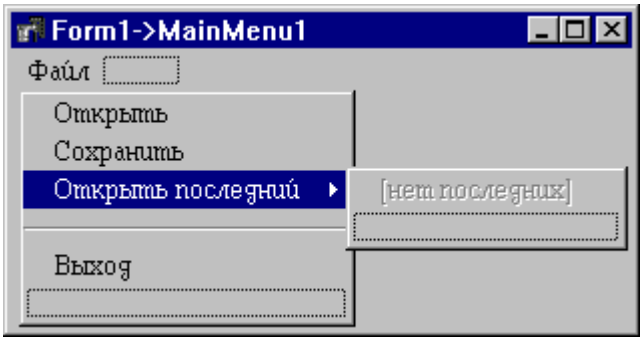

Рис. 1.9 — Внешний вид главного меню формы

Для задания обработчиков событий нужно, два раза щелкнуть по пункту меню.

Лучше задать понятные имена - *Open*, *Save* и *Exit*. Для *TMemo* установите свойство *Align* в *alClient*, а *ScrollBars* в *ssBoth*.

Код должен быть такой:

```
void __fastcall TForm1::ExitClick(TObject *Sender)
{
  Close();
}
//---------------------------------------------------------------

void __fastcall TForm1::OpenClick(TObject *Sender)
{
  if (OpenDialog1->Execute())
        Memo1->Lines->LoadFromFile(OpenDialog1->FileName);
}
//---------------------------------------------------------------

void __fastcall TForm1::SaveClick(TObject *Sender)
{
  if (SaveDialog1->Execute())
        Memo1->Lines->SaveToFile(SaveDialog1->FileName);
}
```

Методы *LoadFromFile*() и *SaveToFile*() загружают из файла и сохраняют в файл текстовое содержимое компонента *Memo*1.

1.4. Индивидуальные задания

Разработать приложение «Текстовый редактор», которое позволяет создавать, открывать, редактировать, и сохранять файлы с расширением *.*txt*, *.*ini* и *.*cmd*. В заголовке приложеня должны отображаться полный путь к файлу, его имя и расширение. В панели состояния должно отображаться время работы программы и время согласно индивидуальному заданию приведенному в таблице 1.1. На панели меню должно быть четыре кнопки: Новый документ (*New*), Открыть (*Open*), Сохранить (*Save*), Сохранить как (*Save as*).

Таблица 3.14 — Индивидуальное задание

N	Задание
1,5	Время работы с программой в формате «чч:мм:сс»
2,6,9	Время работы с документом в формате «чч:мм:сс»
3,7	Время прошедшее с последнего сохранения документа в формате «чч:мм:сс»
4,8,0	Время простоя приложения (когда пользователь не изменяет текст) в формате «чч:мм:сс»

Где *N* — Последняя цифра зачетной книжки студента.

1.5. Содержание отчета

Отчет должен содержать:
1. Титульный лист;
2. Цель работы;
3. Описание и алгоритм работы программы;
4. Текст программы и результаты разработки программы;
5. Результаты работы программы;
6. Выводы.

Перед защитой отчета студент должен продемонстрировать работу программы.

1.6. Контрольные вопросы

1.6.1. Из каких файлов состоит приложение C++Builder и как они организованы?
1.6.2. Какие коды автоматически генерируются в головном файле приложения при создании этого приложения?
1.6.3. В чем отличие действия команд Run, Make и Build при работе с проектом?
1.6.4. Каково назначение окон Инспектора объектов?
1.6.5. Для чего используется окно Редактора кода?
1.6.6. Что содержит файл представления формы (.dfm)?
1.6.7. Что такое событие и как создать обработчик события для компоненты?
1.6.8. Какие существуют отличия модальной формы, SDI- и MDI-форм?
1.6.9. Что находится в разделах __published, private и public класса в заголовочном файле формы Unit.h?
1.6.10. Что произойдет в результате выполнения кода: {Form1->Button1Click(Form1);}
1.6.11. Что произойдет в результате выполнения кода:

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