Тема: Среда **Embarcadero RAD Studio – C++ Builder**

- Основные понятия ООП
- Визуальное программирование. Основные понятия
- Основы работы в **C++ Builder**
- Компонент форма **- TForm**
-

Основные понятия ООП

Объектно-ориентирование программирование (ООП) - это методика разработки программ, в основе которой лежит понятие объекта, как некоторой структуры, описывающей объект реального мира, его поведение.

Функциональную возможность и структуру объектов задают классы – типы данных, определенные пользователями.

Класс – это сложная структура, включающая, помимо описания данных, описание процедур и функций, которые могут быть выполнены над представителем класса – объектом.

Основные понятия ООП

Данные класса называются полями, процедуры и функции – методами – выполняемые объектом действия. Через методы происходит общение с данными объекта.

Совокупность данных и методов их чтения и записи называется свойством. Свойства объекта характеризуется полем, сохраняющим значение свойства, и двумя методами, обеспечивающими доступ к полю свойства.

В программе **C++** для установки значения свойства достаточно записать обычную команду присваивания значения свойству. Для обращения к свойству указывают имя объекта и через **->** значение свойства.

Например: Человек->Имя = “Иван”

Основные понятия ООП

Для того, чтобы метод был выполнен, необходимо указать имя объекта и имя метода, отделив одно имя от другого точкой.

Например: Человек->Есть(пища)

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

События наступают, прежде всего вследствие действий пользователя – перемещения курсора мыши, нажатия кнопок мыши или клавиш клавиатуры.

Реакцией на событие должно быть какое-либо действие. В **C++** реакция на событие реализуется как процедура его обработки, называемая обработчиком события.

Основные понятия ООП

Основные механизмы (постулаты, принципы) ООП

1. Инкапсуляция – механизм, связывающий вместе код и данные, которыми он манипулирует, и одновременно защищающий их от произвольного доступа со стороны другого кода, внешнего по отношению к рассматриваемому. Доступ к данным жестко контролируется интерфейсом.
2. Наследование – механизм, с помощью которого один объект (производного класса) приобретает свойство другого объекта (родительского, базового класса).
3. Полиморфизм – механизм, позволяющий использовать один и тот же интерфейс для общего класс действий.

Основные понятия ООП

Декомпозиция заключается в том, то сложные проекты разбиваются на более простые составные части, что позволяет упростить разработку ПО.

Декомпозиция бывает *алгоритмическая* и *объектно-ориентированная*.

Пример алгоритмической декомпозиции -выделение в рамках программы на ЯВУ, например, **C**, функций и процедур – структурное программирование.

Процедуры и функции – это фрагменты кодов (блоки программы), предназначенные для выполнения повторяющихся операций.

Объектно-ориентированная декомпозиция – разбиение программы на отдельные независимые модули.

Визуальное программирование. Основные понятия

Среда визуальной разработки — среда разработки программного обеспечения, в которой наиболее распространенные блоки программного кода представлены в виде графических объектов.

Среда визуального программирования предоставляет возможность быстрого создания программ путем визуального проектирования макета в графическом виде.

Технология работы в среде **C++ Builder** базируется на идеях объектно-ориентированного и визуального программирования.

Визуальное программирование. Основные понятия

C++ Builder – интегрированная среда разработки **(IDE - Integrated Development Enviroment) -** средство быстрой разработки приложений**,** позволяющее создавать приложения на языке **C++,** используя при этом среду разработки и библиотеку компонентов **Delphi.**

Структурной единицей является визуальный объект**,** называемый компонентом**.**

Автоматизация программирования достигается благодаря возможности переноса компонента на форму **(**в программу**)** с палитры компонентов и изменения его свойств**,** не внося вручную изменений в программный код**.**

Введение в визуальное программирование

Все пользовательские программы в среде **C++ Builder** называются приложениями и оформляются в виде специальных структур, называемых проектом. Проект – это совокупность файлов, из которых состоят приложения.

Работа над новым проектом (приложением **C++ Builder**) начинается с создания стартовой формы - окна, которое появляется при запуске приложения.

На этапе разработки программы диалоговые окна называются формами.

Форма – компонент, который обладает свойствами окна **Windows** и предназначен для размещения других компонентов.

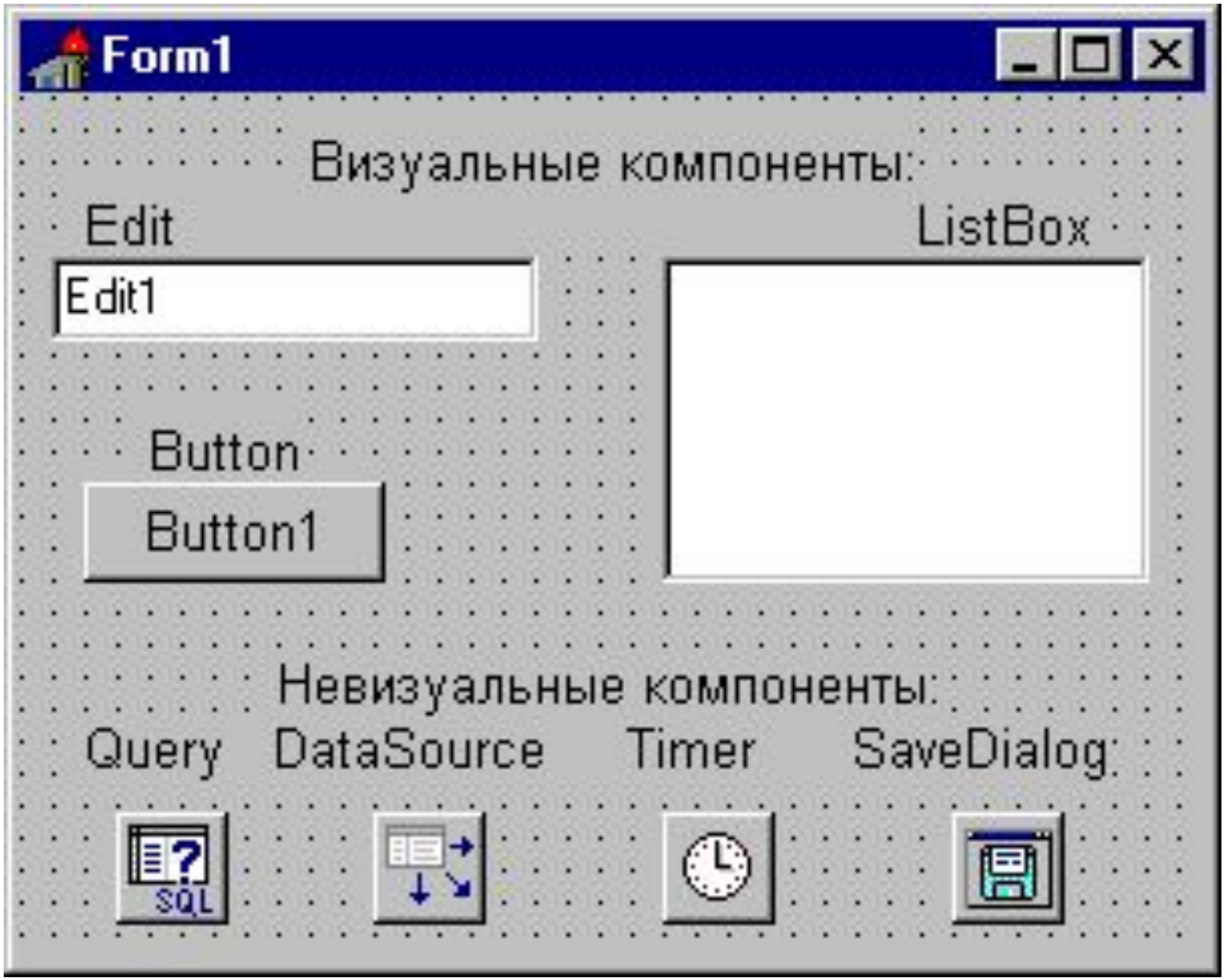
Визуальное программирование. Основные понятия

Компоненты на форме могут быть видимыми и не видимыми.

Видимые предназначены для организации диалога с пользователем. Это кнопки, списки, текстовые поля, изображения и т.д.

Невидимые служат для доступа к системным ресурсам компьютера (таймер и т.д.).

Визуальное программирование. Основные понятия

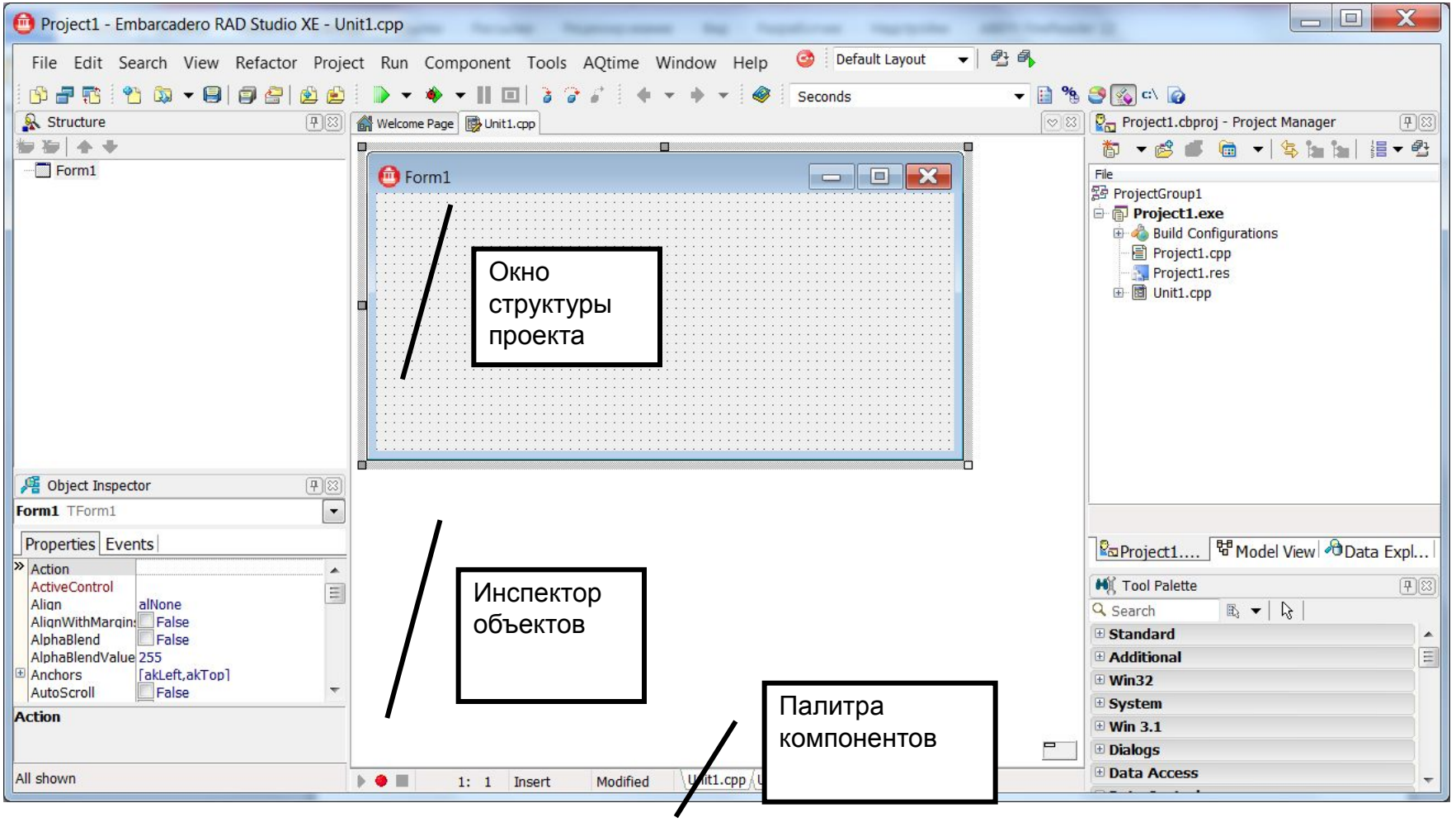


Основы работы в C++ Builder

1. Инструменты среды

- **Главное меню**
- **Панель инструментов**
- **Палитра компонентов (Component Palette / Tool Palette)**
- **Инспектор объектов (Object Inspector)**
- **Окно формы**
- **Редактор кода программы (Code Editor)**

Главное окно среды **Embarcadero RAD Studio XE2**



2. Главное меню и Панель инструментов


открывает доступ к репозиторию объекта


открывает последние 10 проектов, над которыми работали в систем


сохраняет активный файл


сохраняет все файлы проекта


открывает созданный ранее проект с указанного места на носителе


добавляет файлы в проект


удаляет файлы из проекта


вызов справочной системы


выбирает модуль из списка модулей проекта


выбирает форму из списка форм проекта

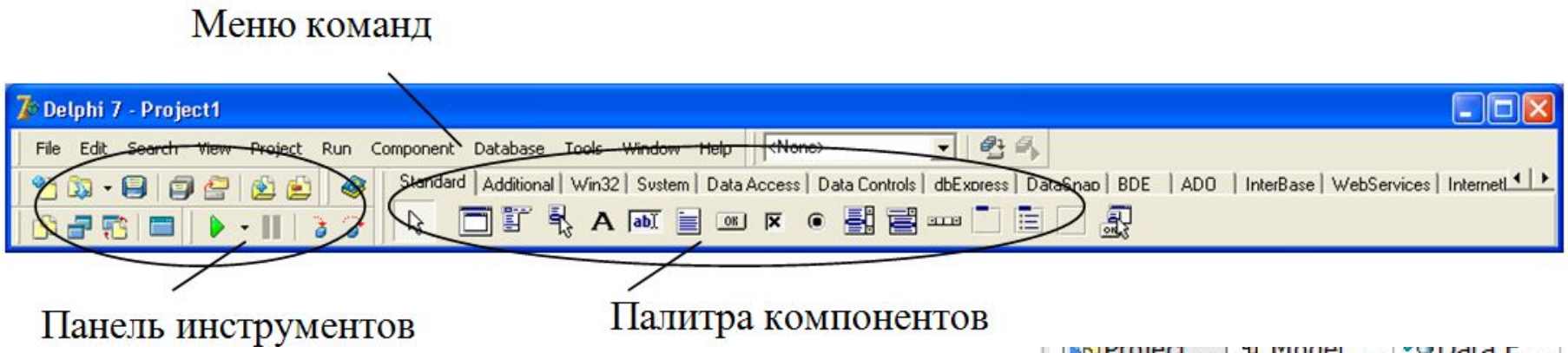

быстрый переход от окна формы к окну кодов и, наоборот


добавляет в проект новую форму

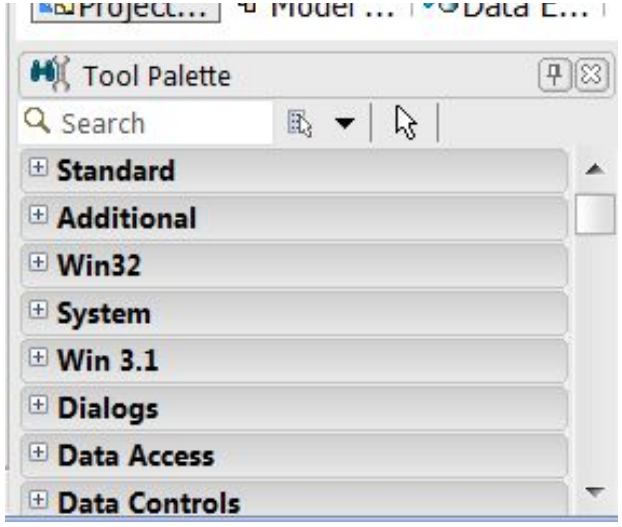

компиляция и выполнение проекта


пауза в выполнении программы

3. Палитра компонентов



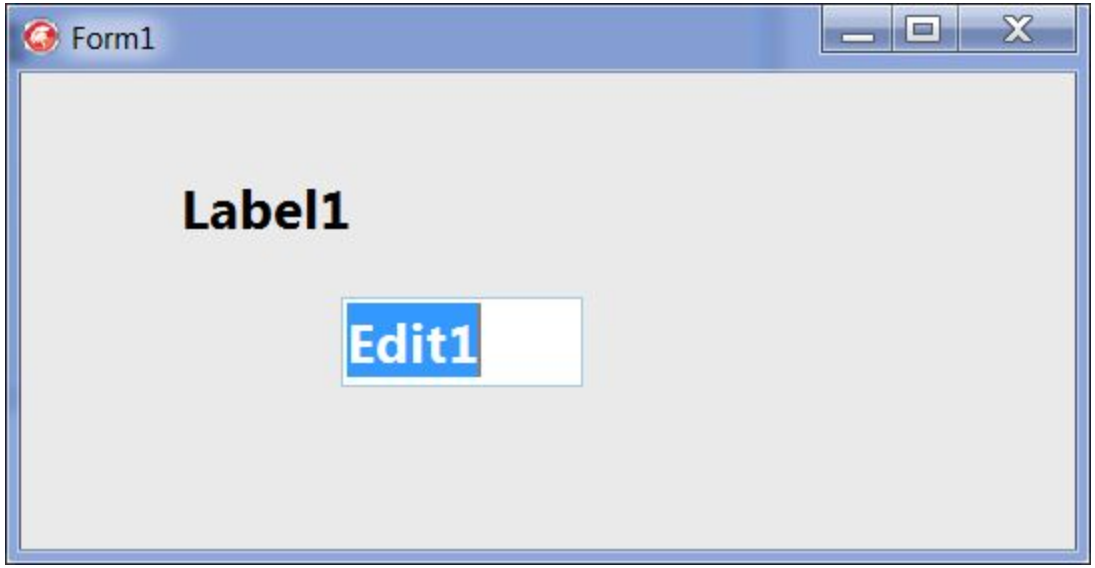
- **Для помещения одного компонента на форму следует его выбрать щелчком мыши на палитре и затем щелкнуть на форме**
- **Для размещения нескольких одноименных компонентов следует выбрать инструмент с клавишей Shift. Для прекращения выбора текущего инструмента следует щелкнуть на палитре по инструменту**



4. Окно формы

Все внутреннее пространство формы является рабочей областью, которая имеет сетку выравнивания для удобного размещения компонентов на форме.

Для выполнения групповых операций несколько компонентов можно объединять с помощью перетягивания мыши или щелчком на объектах с клавишей Shift



Библиотека классов системы

История:

1) **Вначале первые программы, написанные для выполнения под управлением Windows использовали API (Application Programming Interface - интерфейс прикладного программирования) для вызова различных системных функций ОС.**

2) **Первая библиотека классов OWL1.0 (Object Windows Library) от Borland**

3) **Библиотека классов MFC (Microsoft Foundation Class) от Microsoft**

ЗАМЕЧАНИЕ: MFC имела большее распространение, но была ближе к API-интерфейсу, а не ООП

Библиотека классов системы

Сегодня в C++ Builder библиотекой классов является библиотека визуальных компонентов VCL – Visual Component Library) – библиотека визуальных компонентов. В ее основе лежит концепция свойств, методов и событий.

В библиотеке VCL `#include <vcl.h>` **П, как наследование.**

Класс TObject является базовым классов для всех классов библиотеки VCL – содержит общие для всех объектов свойства и методы, позволяющие создавать и удалять объект, обрабатывать сообщения и др.

4. Инспектор объектов

С помощью инспектора объектов можно задавать начальные значения свойств объектов и их реакцию на стандартные события.

Окно инспектора объектов содержит список компонентов текущей формы, а также две закладки: свойства **(Propeties)** и события **(Events)**.

Клавиша **F11** выводит (скрывает) окно инспектора объектов.

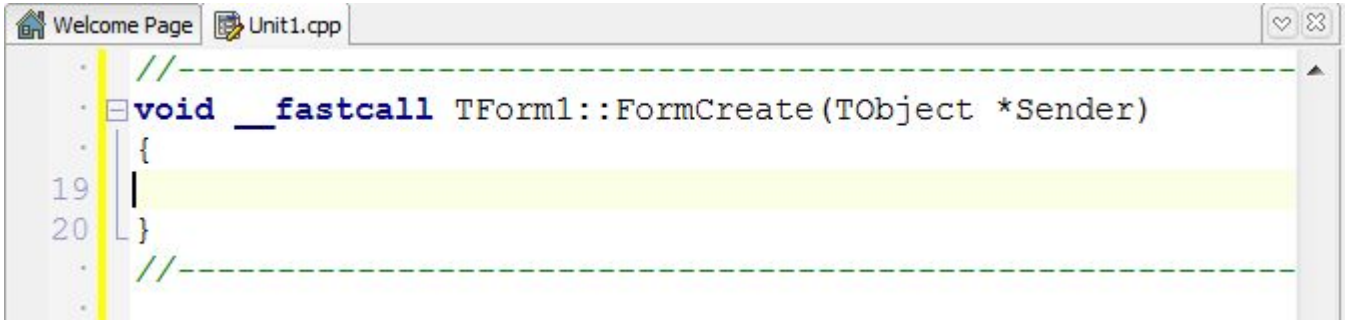
4. Инспектор объектов

Свойства являются атрибутами компонента, определяющими его внешний вид и поведение.

Каждый компонент имеет свой собственный набор обработчиков событий.

```
Label1->Caption = "First";
```

В **C++ Builder** следует писать функции, называемые обработчиками событий, и связывать события с этими функциями. Создавая, обработчик того или и ого события, вы поручаете программе выполнить написанную функцию, если это событие произойдет.



4. Инспектор объектов

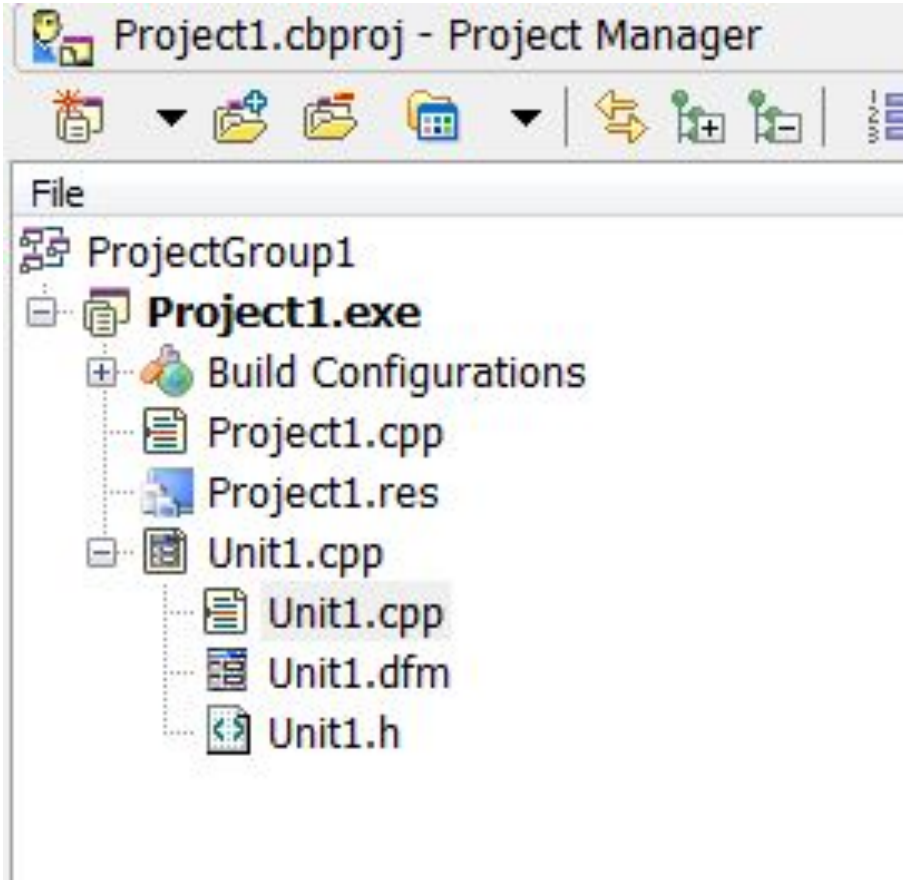
Метод является функцией, которая связана с компонентом, и которая объявляется как часть объекта. Создавая обработчики событий, можно вызывать методы, используя следующую нотацию: **->**, например:

```
Label1->Show();
```

ЗАМЕЧАНИЕ: 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.

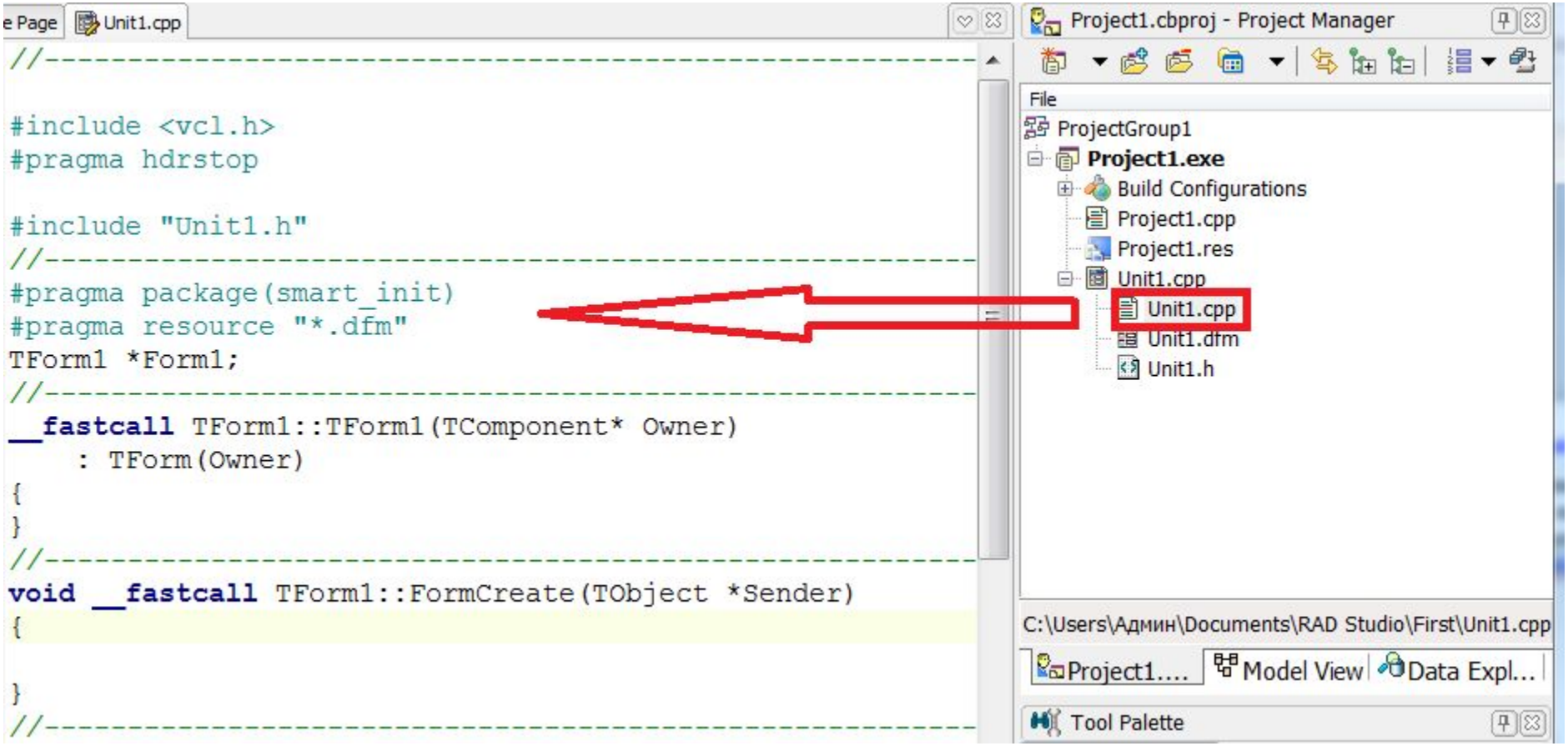
5. Менеджер проектов

Менеджер проектов показывает списки файлов и модулей приложения и позволяет осуществлять навигацию между ними. Можно вызвать менеджер проектов , выбрав пункт меню **View/Project Manager.**



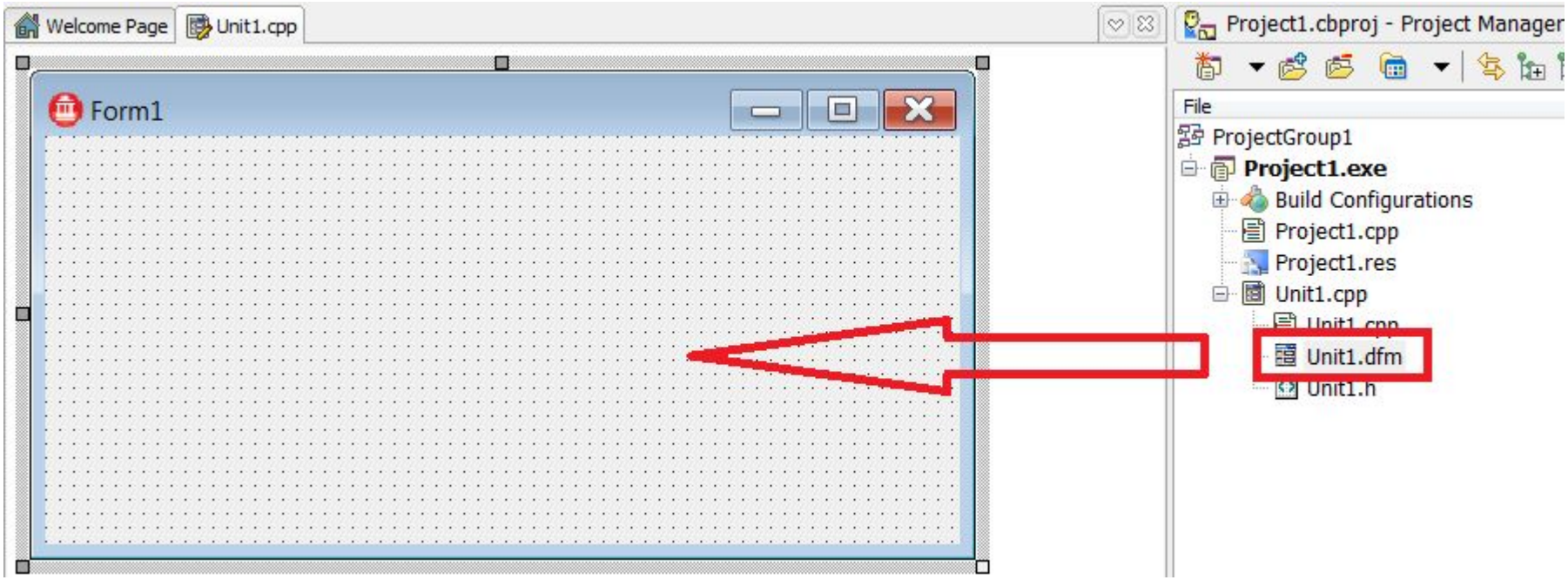
5. Менеджер проектов

Файл модуля с расширением **.CPP,** содержащий код на **C++.**



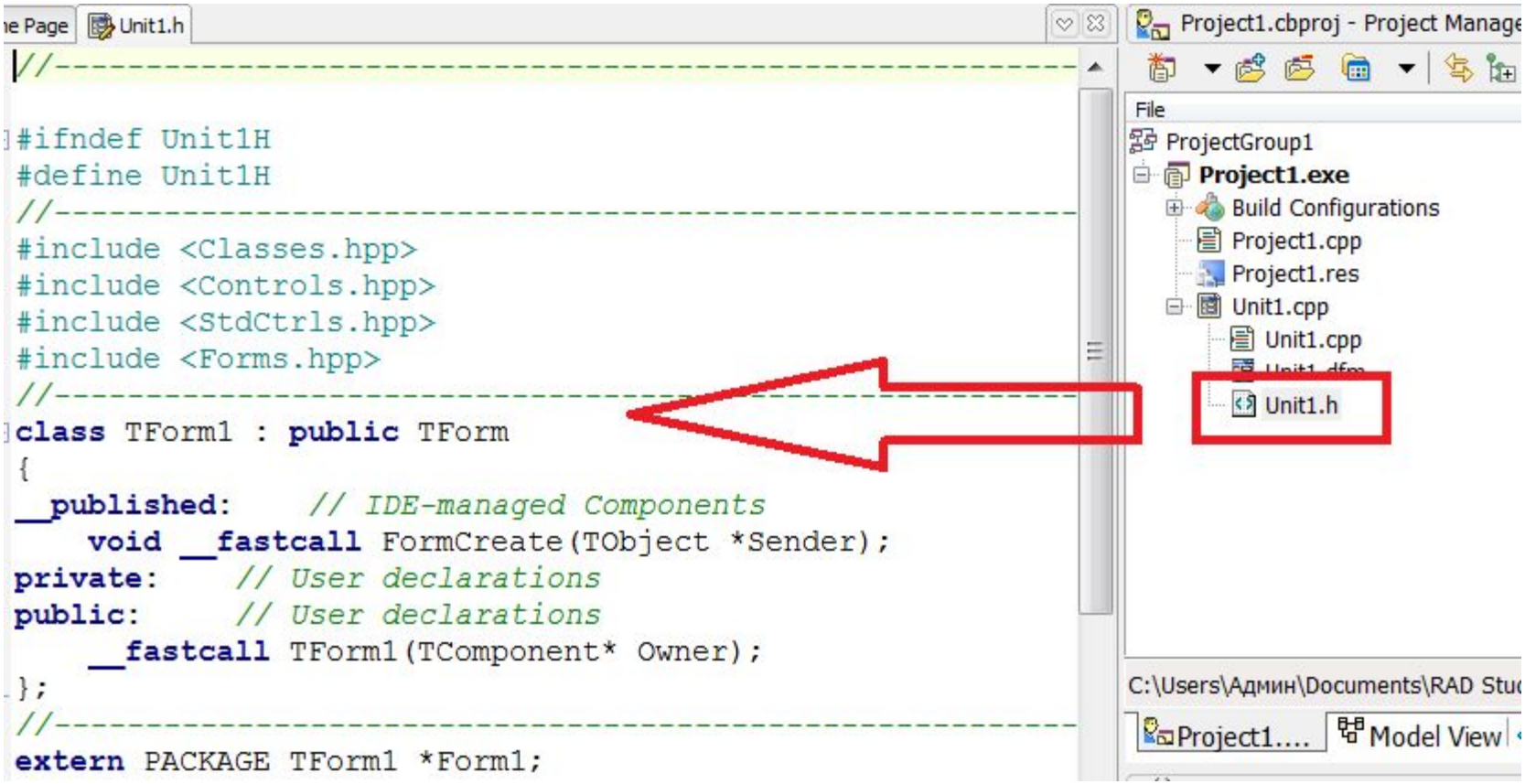
5. Менеджер проектов

Файл формы с расширением **.DFM,** содержащий информацию о ресурсах окон для конструирования формы

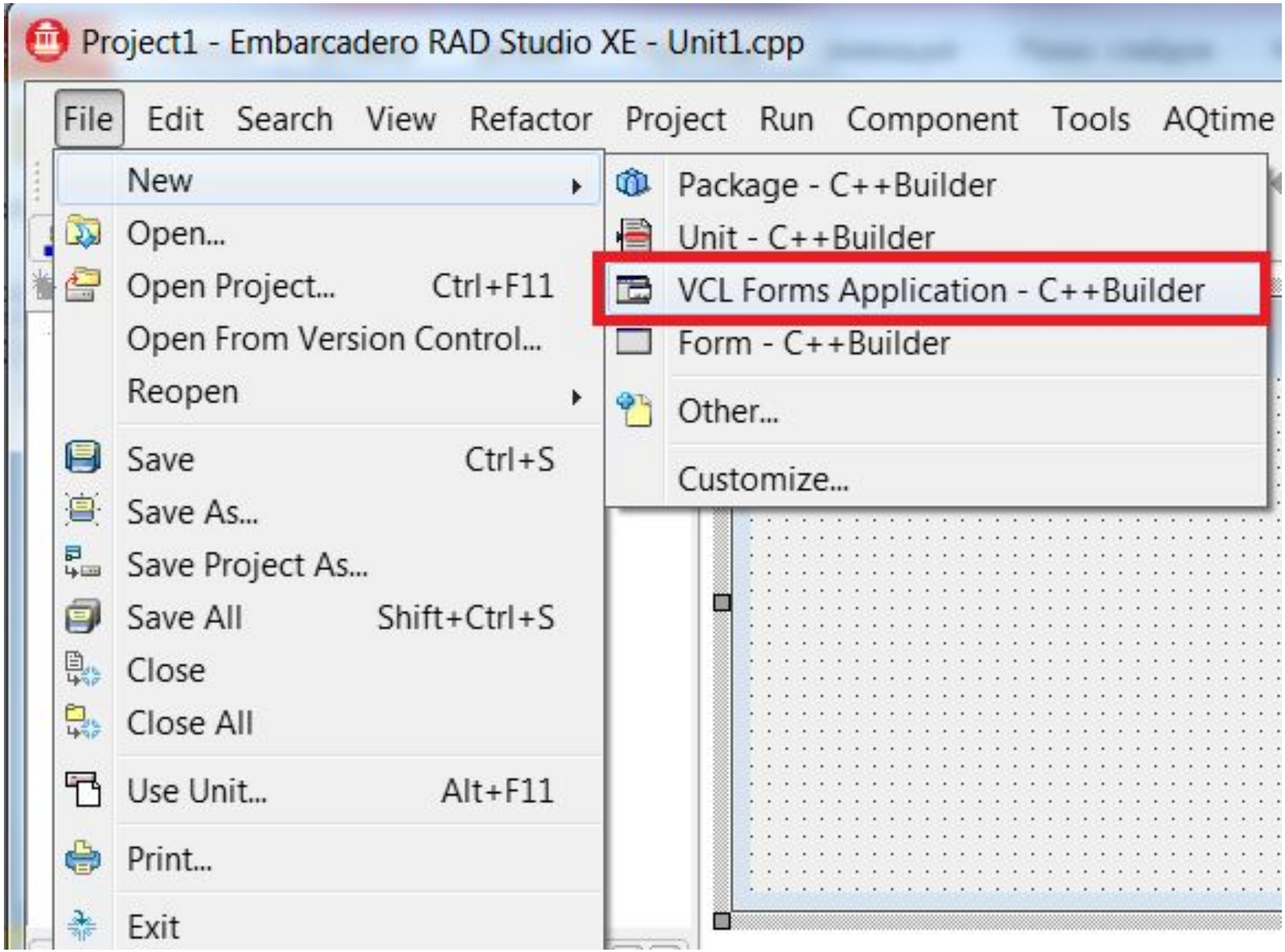


5. Менеджер проектов

Заголовочный файл с расширением **.H,** содержащий описание класса формы**.**



7. Создание проекта



8. Сохранение проекта (первое)

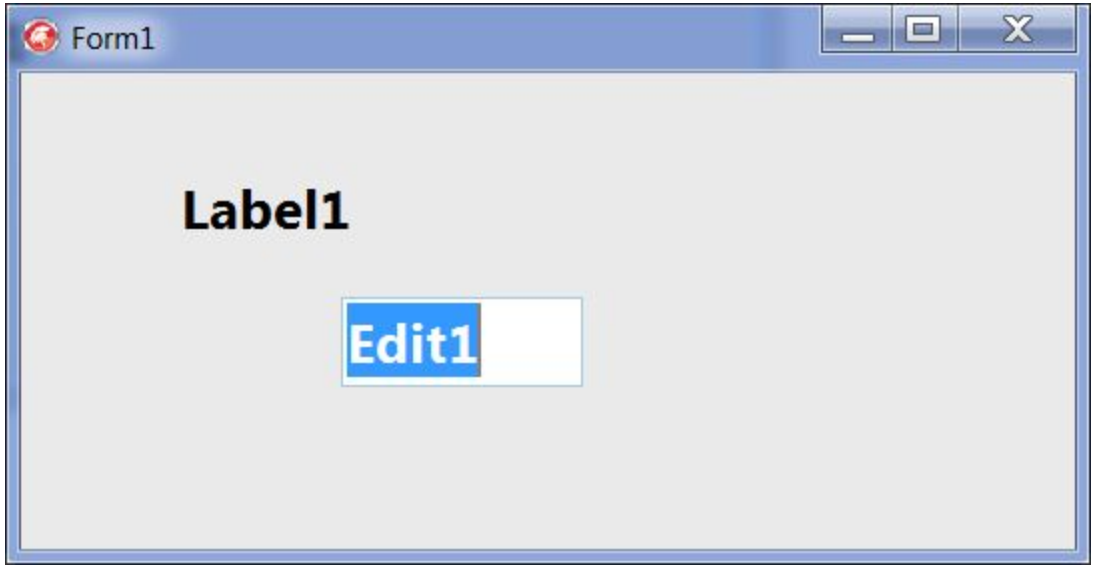
1. Выполнить команду **File – Save Project As**
2. В окне выбрать папку для сохранения (каждый проект сохранять в отдельной папке).
3. Ввести имя модуля формы (например, согласиться с именем **Unit1.cpp**).
4. Ввести имя главного модуля (например, согласиться с предложенным **Project1.cbproj**).

ЗАМЕЧАНИЕ: при последующих сохранениях либо нажимать на кнопку Сохранить или Сохранить все, либо команда **File – Save All**

9. Компиляция и запуск проекта на выполнение

1. Выполнить команду **Run – Run** (или **F9**)
2. Закрыть окно приложения после работы

ЗАМЕЧАНИЕ: если компиляция произошла успешно, то в папке с проектом в папке **Debug\Win32** будет сформирован исполнимый файл, например, **Project1.exe**



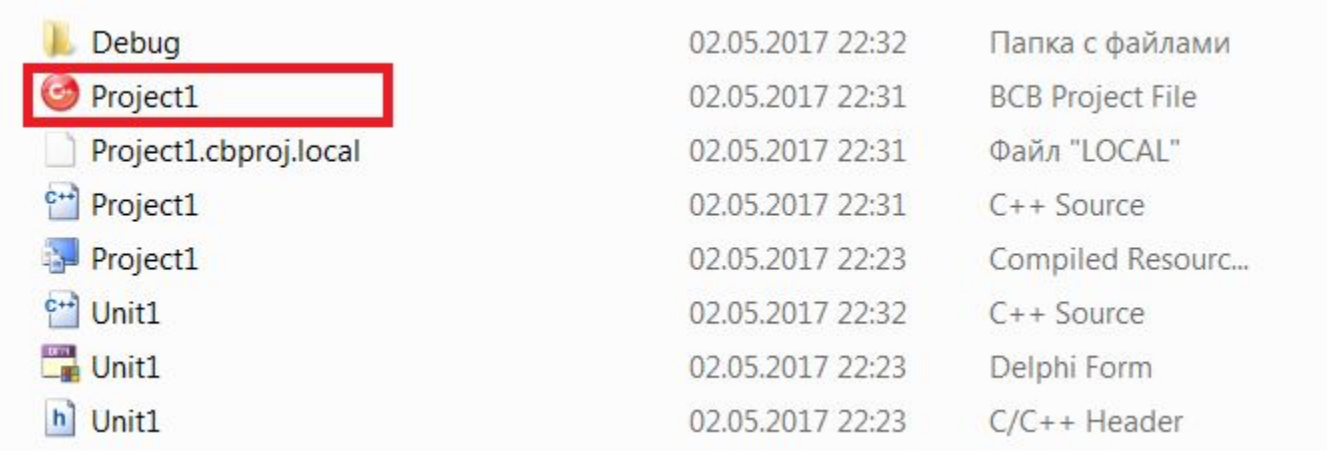
10. Закрытие проекта в среде

1. Выполнить команду **File – Close All.**
2. Закрыть окно среды визуального программирования.

11. Открытие готового проекта в среде для редактирования

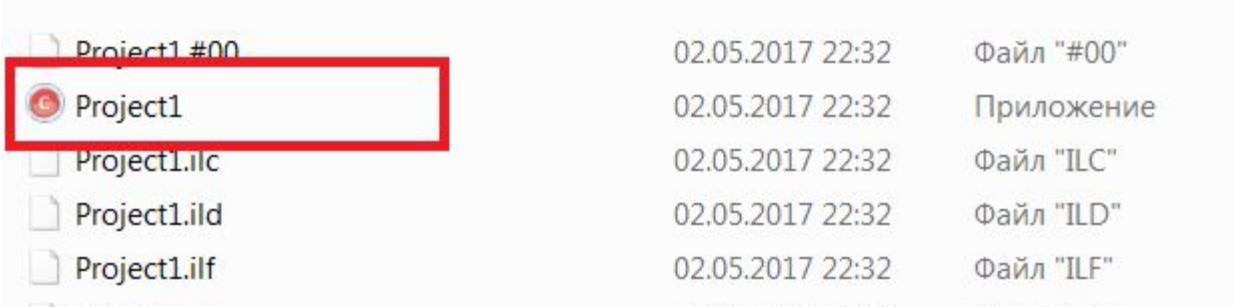
1. Выполнить команду **File – Open Project**
2. Выбрать папку с проектом и открыть файл проекта, например, **Project1.cbproj** в среде **Embarcadero RAD Studio.**

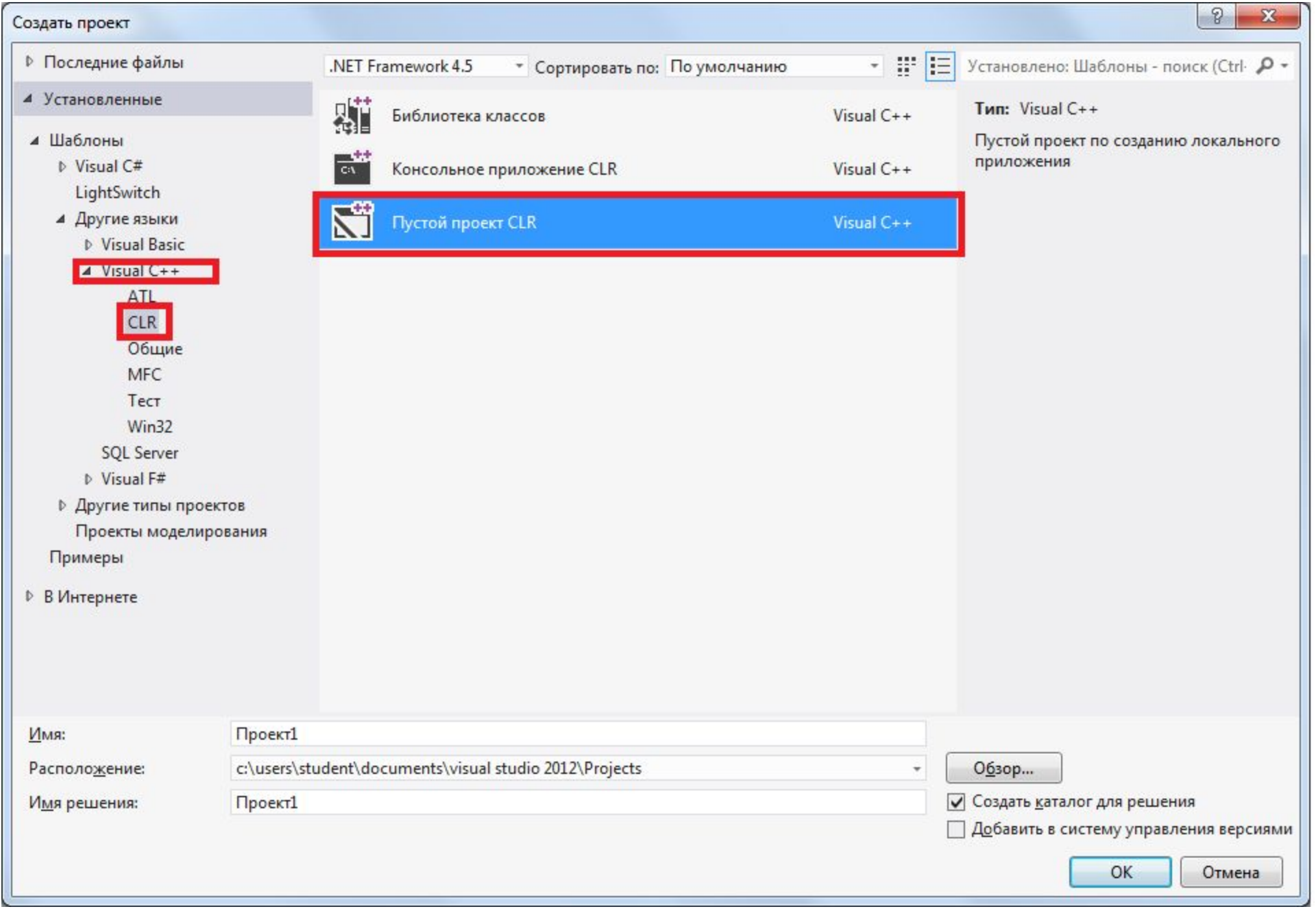
   запустить из проводника файл **Project1. cbproj** в среде **Embarcadero RAD Studio**

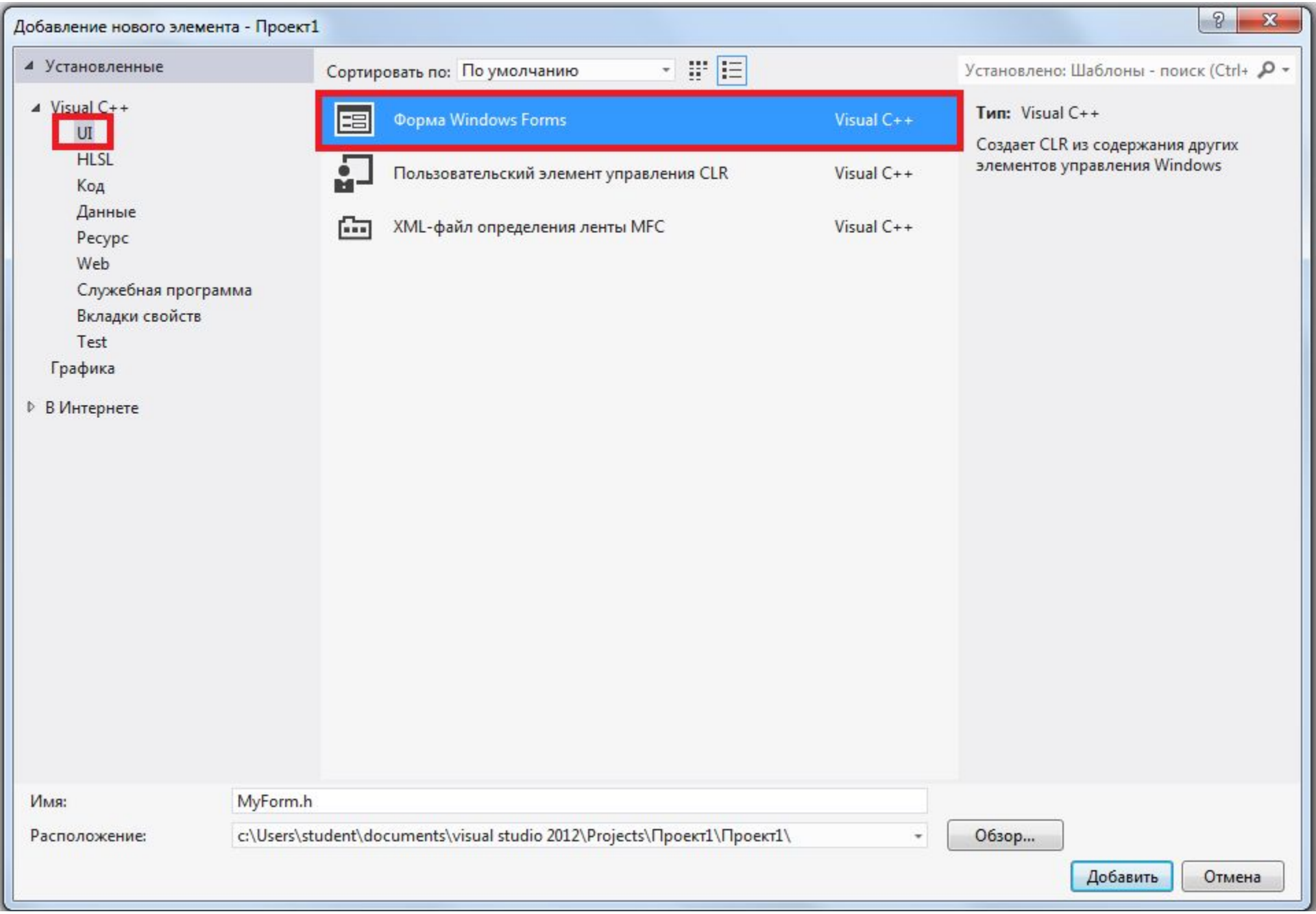


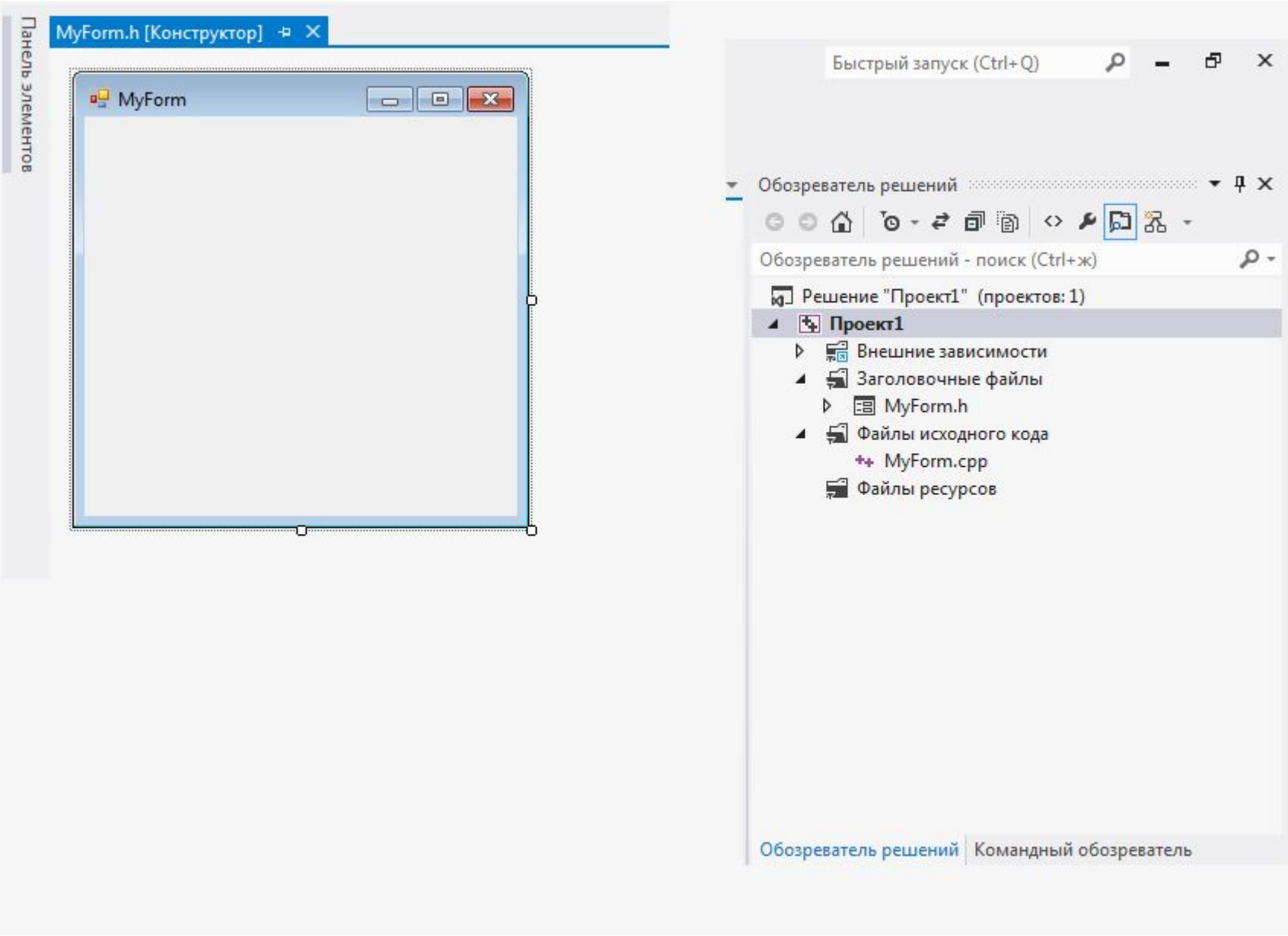
12. Запуск готового приложения на любом компьютере

1. Выполнить приложение из среды программирования для формирования исполнимого файла (команда **Run – Run** (или **F9**)
2. Закрыть проект и среду программирования.
3. Скопировать в нужное место папку **Win32** (для проектов, созданных в среде **Embarcadero RAD Studio**).
4. Запустить файл *.exe из **Debug\Win32**




Создать проект
Последние файлы
Установленные
Шаблоны
Visual C#
LightSwitch
Другие языки
Visual Basic
Visual C++
ATL
CLR
Общие
MFC
Тест
Win32
SQL Server
Visual F#
Другие типы проектов
Проекты моделирования
Примеры
В Интернете
.NET Framework 4.5
Сортировать по:
По умолчанию
Установлено: Шаблоны - поиск (Ctrl·
Библиотека классов
Visual C++
Консольное приложение CLR
Visual C++
Пустой проект CLR
Visual C++
Тип: Visual C++
Пустой проект по созданию локального приложения
Имя:
Проект1
Расположение:
c:\users\student\documents\visual studio 2012\Projects
Обзор...
Имя решения:
Проект1
Создать каталог для решения
Добавить в систему управления версиями
OK
Отмена


Добавление нового элемента - Проект1
Установленные
Visual C++
UI
HLSL
Код
Данные
Ресурс
Web
Служебная программа
Вкладки свойств
Test
Графика
В Интернете
Сортировать по: По умолчанию
Установлено: Шаблоны - поиск (Ctrl+
Форма Windows Forms Visual C++
Пользовательский элемент управления CLR Visual C++
XML-файл определения ленты MFC Visual C++
Тип: Visual C++
Создает CLR из содержания других элементов управления Windows
Имя: MyForm.h
Расположение: c:\Users\student\documents\visual studio 2012\Projects\Проект1\Проект1\
Обзор...
Добавить
Отмена


Панель элементов
MyForm.h [Конструктор]
MyForm
Быстрый запуск (Ctrl+Q)
Обозреватель решений
Обозреватель решений - поиск (Ctrl+ж)
Решение "Проект1" (проектов: 1)
Проект1
Внешние зависимости
Заголовочные файлы
MyForm.h
Файлы исходного кода
MyForm.cpp
Файлы ресурсов
Обозреватель решений
Командный обозреватель

Компонент форма - **TForm**

Как и любой другой визуальный компонент, форма имеет свойства, методы и события, общие для всех визуальных компонентов.

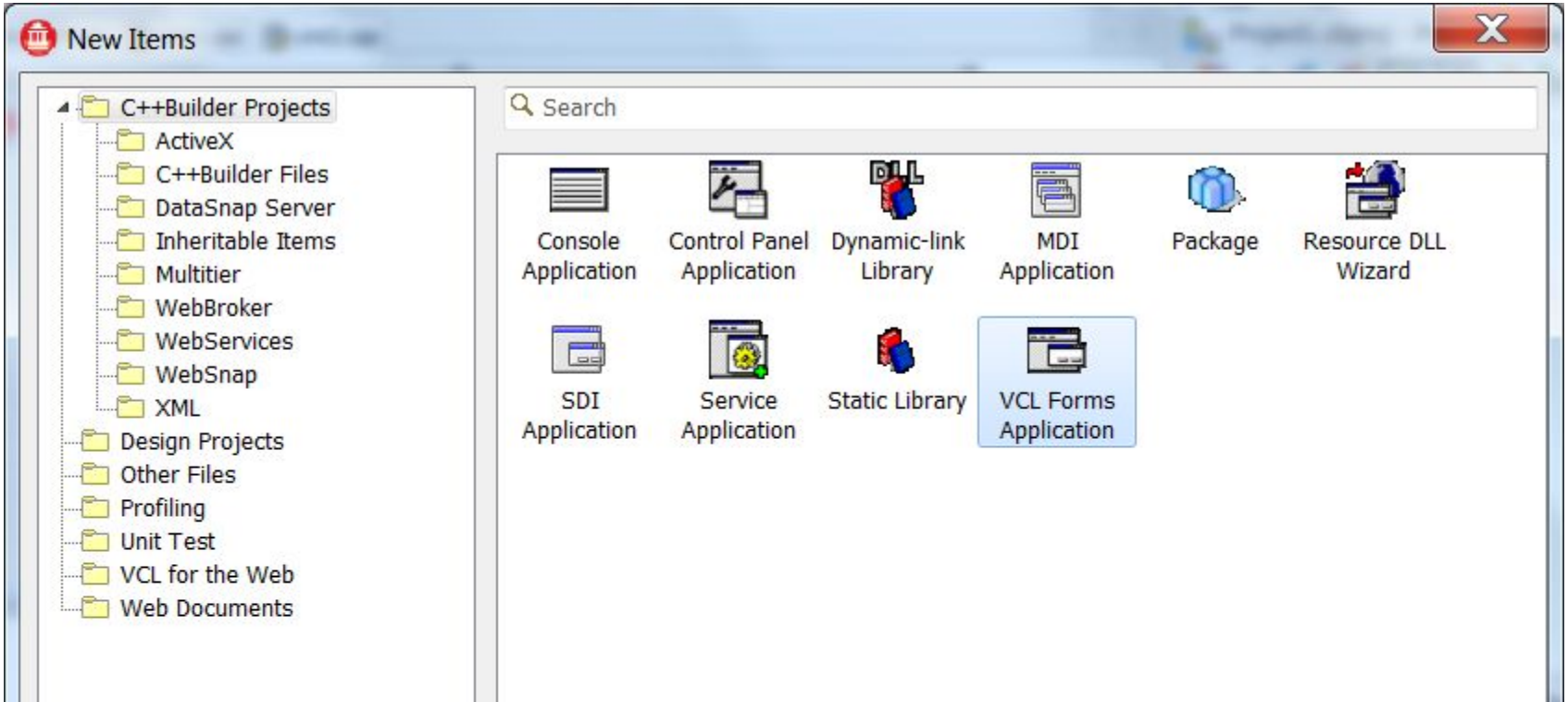
По умолчанию проект первоначально содержит файлы для одной формы и исходного кода одного модуля. Однако большинство проектов содержат несколько форм и модулей.

Компонент форма - TForm

Добавление форм к проекту

1.File → New → Form –C++ Builder (или кнопка **New Items** на панели инструментов Стандартная)

2.File → Save All → задать имя (например, **Unit2**)



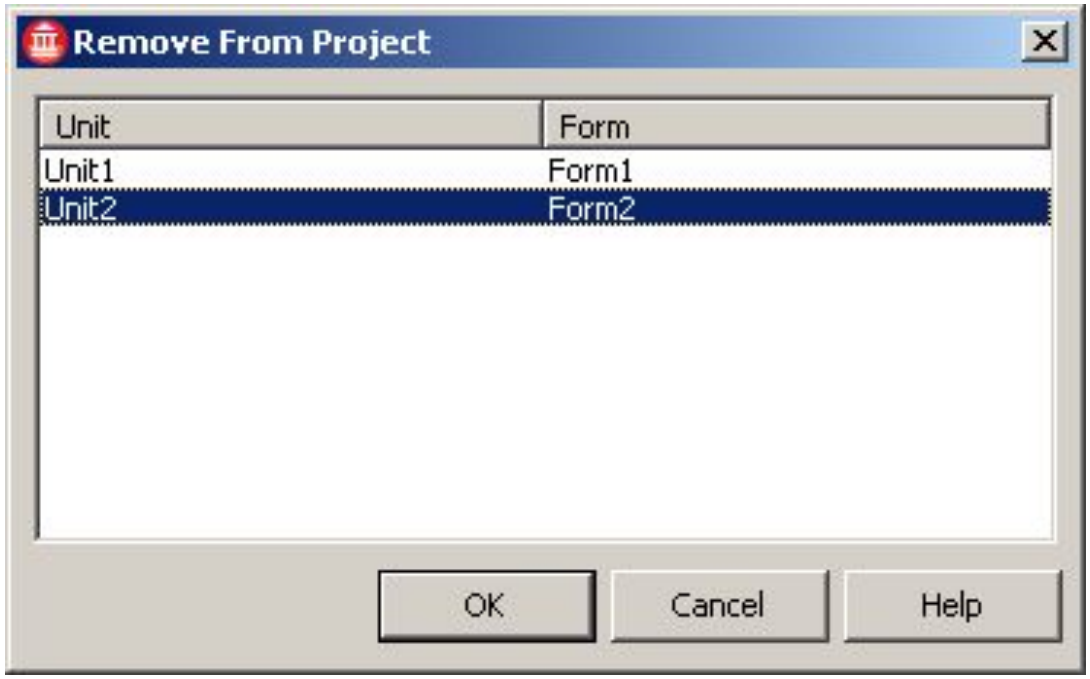
Компонент форма - TForm

Удаление формы из проекта

1.File → Project → Remove from Project

2.Выбрать нужную форму → ОК

3.File → Save All

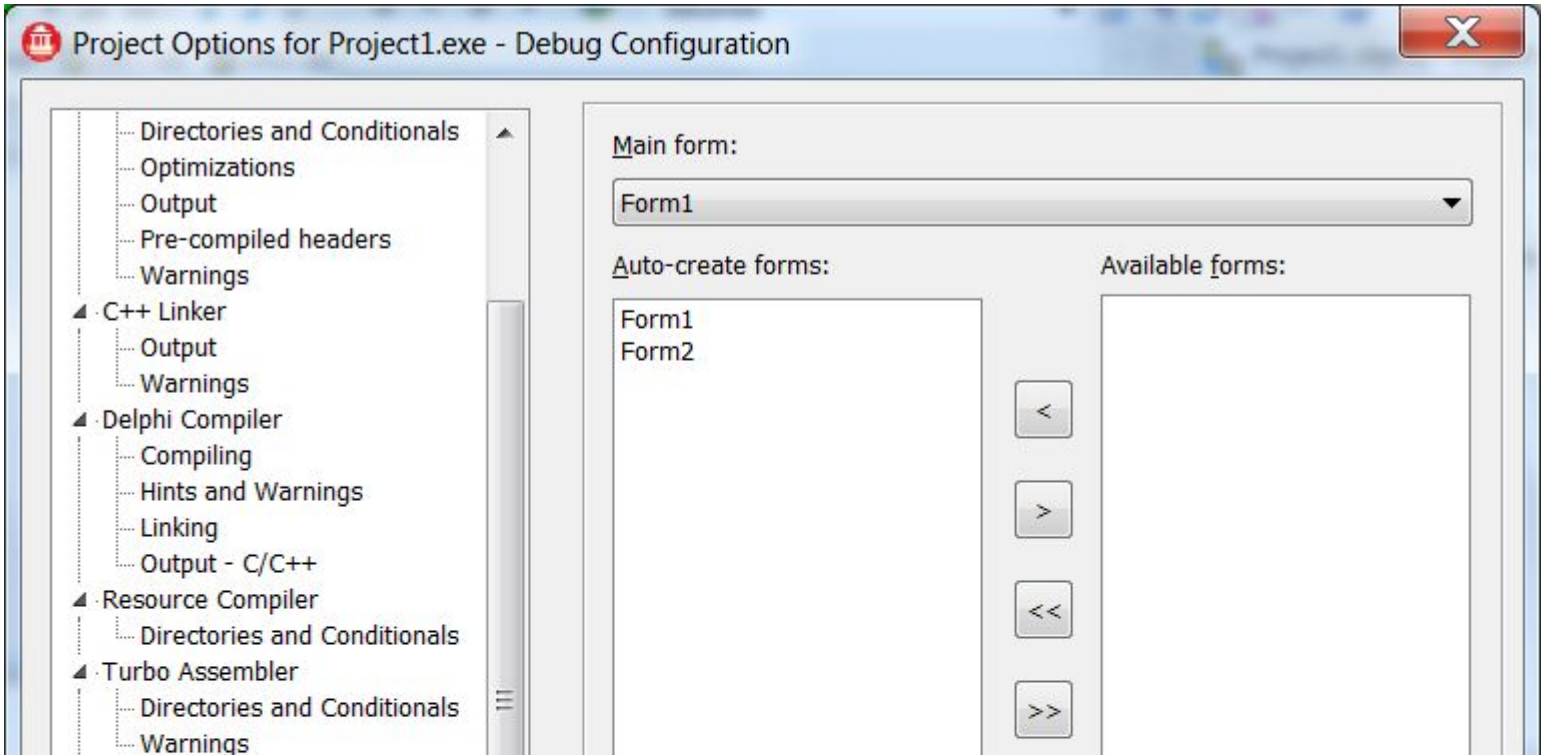


Компонент форма - TForm

Организация работы с множеством форм

- **Выбор главной формы в проекте**

1. **Project → Options → Forms**
2. Выбрать нужную форму → OK



Компонент форма - **TForm**

- **Связь между формами**

Если в проекте много форм**,** то из главной можно организовать вызов на выполнение остальных форм **(**а можно и из каждой формы вызвать любую другую**).**

*Способы вызова форм***:**

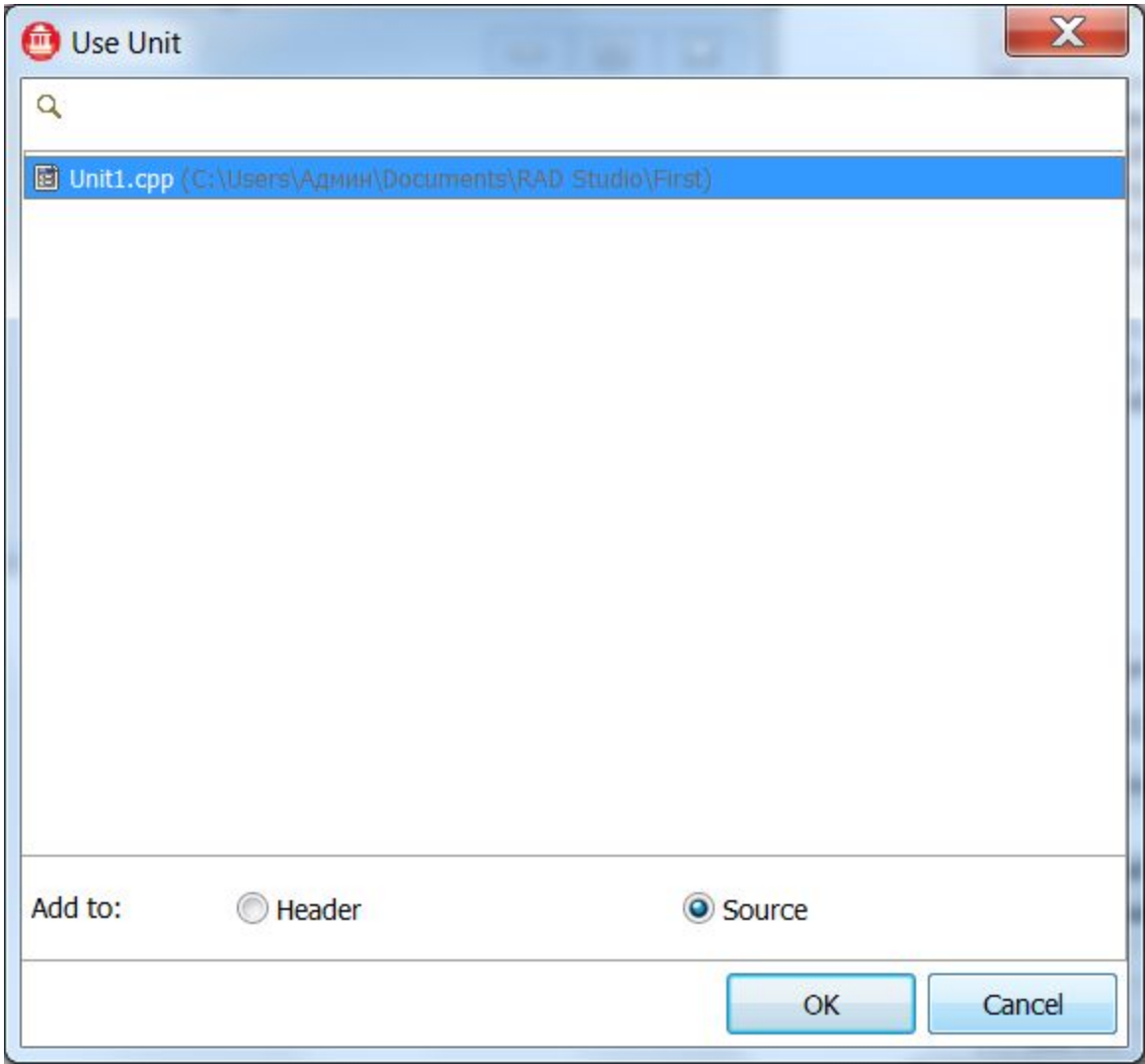
1. С помощью обработчиков кнопок вызова конкретных форм
2. С помощью меню

Компонент форма - TForm

- **Вызов**

1. **File → Use Unit**
2. Выбрать

нужный модуль



Компонент форма - TForm

- **Вызов формы на выполнение**

Бывает двух типов: модальный и немодальный (обычный)

При немодальном (обычном) выполняется метод класса **TForm ,** который называется **Show()**

```
Form1 -> Show();
```

ЗАМЕЧАНИЕ: В обычном режиме можно вызвать сколько угодно форм, одну за другой – большой расход памяти, если они не закрыты (`Form1 -> Close()`).

Компонент форма - TForm

При модальном режиме вызова формы, приложение не сможет дальше нормально работать, если форма не будет закрыта:

```
Form1 -> ShowModal();
```

ЗАМЕЧАНИЕ: на формах, которые вызываются модально, помещают кнопку с именем и в ее обработчик помещают операторы:

```
ModalResult = 'OK';
Formi->Close();
Form1->Show();
```

Компонент форма - TForm

Здесь **Formi** – это модальная форма, **Form1** – главная форма, из которой вызывалась **Formi**

В случае вызова **Formi** из **Form1** обычным образом, то в обработчики ее кнопки **Close.** можно выло бы написать:

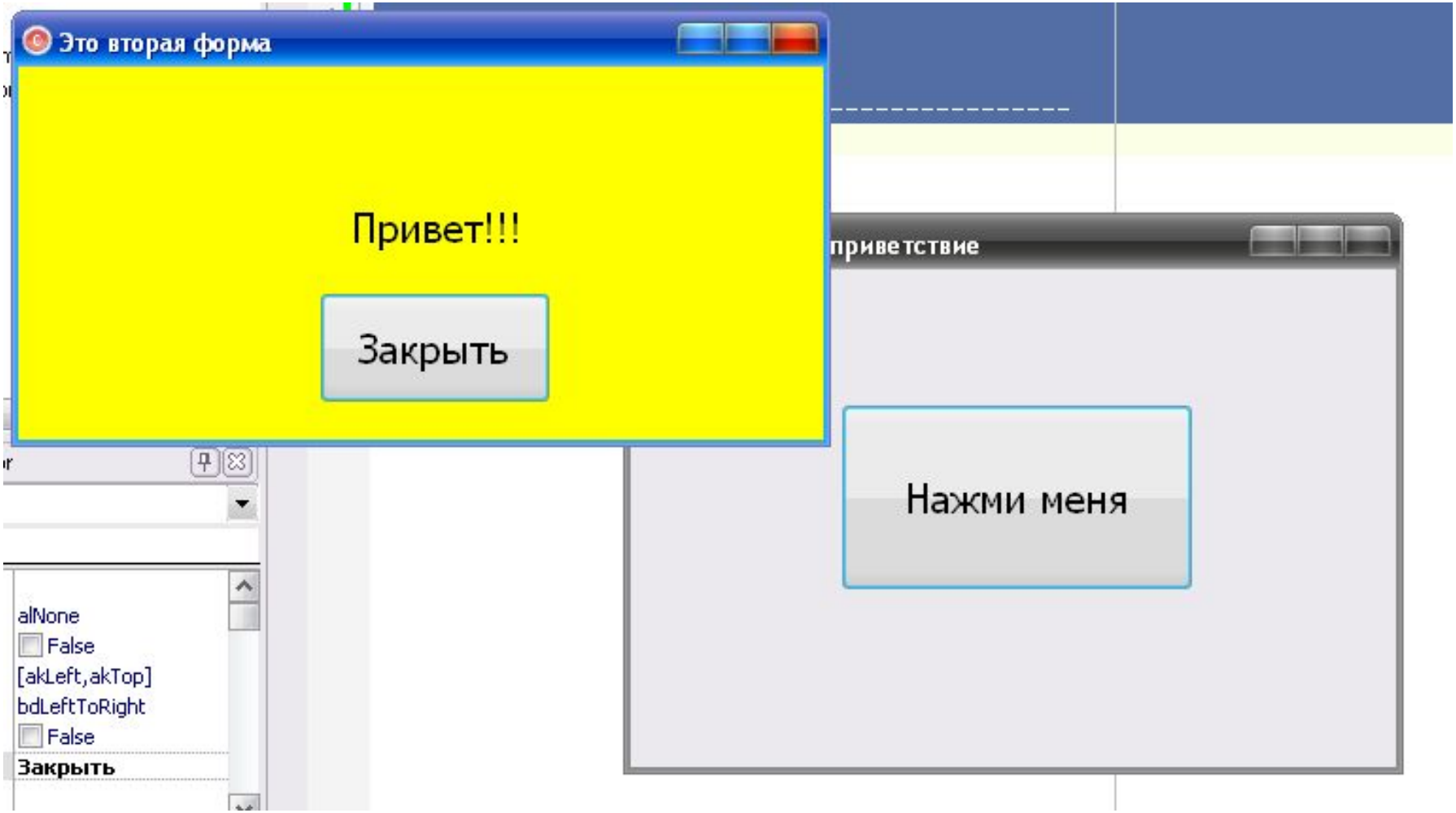
```
Formi->Close();
Form1->Show();
```

Компонент форма - TForm

Пример

Напишем программу, содержащую кнопку Нажми меня, по выбору которой открывается вторая форма с текстом приветствия.

Компонент форма - **TForm**



Компонент форма - **TForm**

Чтобы первая форма **«**видела**»** вторую вызываемую форму**,** в файле **Unit1.cpp** добавим директиву на подключение заголовочного файла формы **2:**

```
#include <vcl.h>
#pragma hdrstop

#include "Unit1.h"
#include "Unit2.h"
```

Компонент форма - TForm

Напишем следующий обработчик события на кнопку **Нажми меня**:

```
void __fastcall
TForm1::Button1Click(TObject *Sender)
{
   Form2->Show();
}
```

Компонент форма - TForm

Для кнопки **Закрыть** на Форме 2 напишем обработчик события:

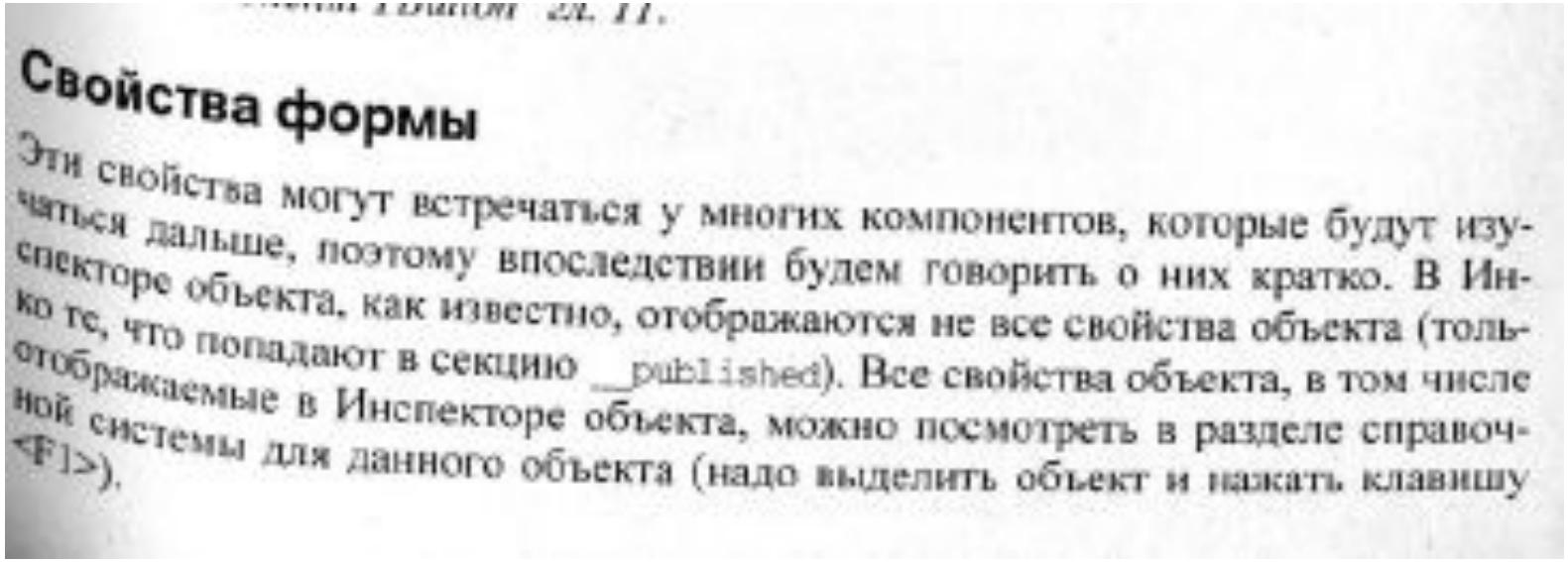
```
void __fastcall
TForm2::Button1Click(TObject *Sender)
{
   Close();
   Form1->Show();
}
```

Компонент форма - TForm

Свойства формы

События формы

Методы формы



Свойства формы

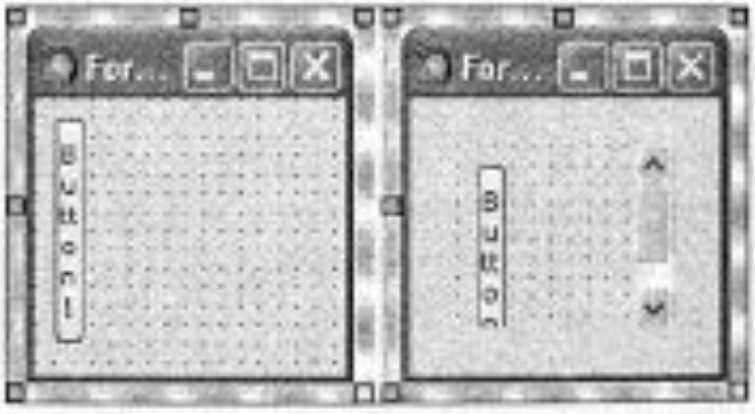
Эти свойства могут встречаться у многих компонентов, которые будут изучаться дальше, поэтому впоследствии будем говорить о них кратко. В Инспекторе объекта, как известно, отображаются не все свойства объекта (только те, что попадают в секцию __published). Все свойства объекта, в том числе отображаемые в Инспекторе объекта, можно посмотреть в разделе справочной системы для данного объекта (надо выделить объект и нажать клавишу <F1>).

Компонент форма - **TForm**

- `Caption` — сюда помещается название формы. По умолчанию первая форма проекта получает имя `Form1`, вторая — `Form2` и т. д.
- `ActiveControl` — это свойство определяет, какой компонент, помещенный на форму, в данный момент является активным (или, как еще говорят, имеет фокус).
- `AutoScroll` — это свойство определяет, будут ли автоматически появляться полосы прокрутки формы, если на ней не будут помещаться компоненты (т. е., чтобы их увидеть, надо будет форму "прокрутить"). Если значение этого свойства равно `true`, то полосы прокрутки будут появляться, а если `false` — нет.
- `AutoSize` — если значение свойства равно `true`, то форма автоматически будет принимать прежние размеры, как бы вы ее не растягивали. При свойстве, равном `false`, форма "позволяет" себя растягивать.
- `BorderStyle` — свойство задает появление и поведение границ формы: можно ли мышью менять размеры формы, когда приложение находится в режиме исполнения

Компонент форма - TForm

- `BorderWidth` — здесь задается в пикселах величина отступа координатной сетки формы от границ окна формы, т. е. фактически размеры формы можно изменить за счет изменения координатной сетки, задав ее отступ от границ окна. По умолчанию величина отступа равна нулю, т. е. форма занимает все пространство окна. На рис. 10.28 показано применение свойства `BorderWidth`: у левой формы на рисунке значение этого параметра равно нулю (значение по умолчанию), а у правой формы значение равно 20 пикселам. Мы видим, что форма в окне сжалась, и на форме перестала помещаться расположенная на ней кнопка. Чтобы обеспечить просмотр непомещающегося компонента, на форме появилась вертикальная полоса прокрутки. Но она появилась не сама по себе, а из-за того, что свойство формы `AutoScroll` было установлено в значение `true`.



Компонент форма - TForm

События формы

- `OnActivate` — возникает, когда форма активизируется;
- `OnClick` — возникает при щелчке мышью на форме;
- `OnClose` — возникает, когда форма закрывается;
- `OnCreate` — возникает, когда форма создается;
- `OnDeactivate` — возникает, когда форма перестает быть активной;
- `OnKeyDown` — возникает, когда пользователь нажимает некоторую клавишу на клавиатуре. Это событие может возникать в ответ на нажатие любых клавиш, включая функциональные (<F1>—<F12>) и комбинации с <Shift>, <Alt> и <Ctrl>;
- `OnKeyPress` — возникает, когда пользователь нажимает некоторую (только одну) клавишу на клавиатуре, кроме функциональных;
- `OnKeyUp` — возникает, когда пользователь отпускает клавишу клавиатуры, которая до этого была нажата. Это событие возникает и тогда, когда до этого были нажаты комбинации клавиш с <Shift>, <Alt> и <Ctrl>, а также функциональные клавиши;

Компонент форма - TForm

События формы

- OnMouseDown — возникает при нажатии любой кнопки мыши, а также при комбинации клавиш <Shift>, <Ctrl> и <Alt> и кнопок мыши. В обработчике этого события X и Y — это координаты в пикселах указателя мыши;
- OnMouseMove — возникает, когда пользователь двигает указатель мыши, пока он находится над объектом (в нашем случае — над формой). В обработчике этого события X и Y — это координаты в пикселах указателя мыши;
- OnMouseUp — возникает, когда пользователь отпускает кнопку мыши, которая была нажата на компоненте. В обработчике этого события X и Y — это координаты в пикселах указателя мыши;
- OnPaint — возникает, когда начинается прорисовка формы;
- OnShow — возникает, когда форма появляется на экране.

Компонент форма - TForm

Методы формы

Форма имеет большое количество методов, которое можно посмотреть в справочной системе среды, нажав клавишу <F1> при активной форме. Рассмотрим только некоторые из методов формы:

- `Close()` — закрывает форму. Если закрывается главная форма, закрывается и само приложение ;
- `Hide()` — свойство `Visible` устанавливается в `false`, и форма становится невидимой;
- `Print()` — форма печатается;
- `Release()` — форма разрушается, и память, занятая ею, освобождается;
- `SetFocus()` — делает форму активной: свойства `Visible` и `Enabled` становятся равными `true`, форма становится видимой и доступной;
- `Show()` — отображает форму: в этом случае свойство `Visible` устанавливается в `true`, и форма перемещается поверх всех форм на экране;
- `ShowModal()` — показать форму в модальном режиме. Когда форма показана в модальном режиме, приложение не может выполняться, пока форма не будет закрыта. Чтобы закрыть такую форму, надо установить ее свойство `ModalResult` в ненулевое значение.

Функции выдачи сообщений и перевода данных из одного типа в другой

Все рассматриваемы строковые данные – данные типа **AnsiString**.

Если необходимо перейти к строкам, которые заданы как **char *a** или **char m[**количество**]** , то к такам строкам легко перейти, используя метод **c_Str()** класса **AnsiString**.

Функции преобразования:

- `StrToCurr()` — преобразует строку `AnsiString` в объект типа `Currency` (валюта). Например:

```
AnsiString S = "250,8";
Currency x = StrToCurr(S);
```

Строка `AnsiString` представляет собой число с плавающей точкой, соответствующее объекту типа `Currency`. Лидирующие и хвостовые пробелы в строке игнорируются.

- `CurrToStr()` — выполняет обратное преобразование. Объект типа `Currency` преобразуется в строку данных типа `AnsiString`.

Функции выдачи сообщений и перевода данных из одного типа в другой

- `FloatToStr()` — преобразует число с плавающей точкой в строку типа `AnsiString`. Например:

```
float x = 12.5;
AnsiString S = FloatToStr(x);
```

- `StrToFloat()` — выполняет обратное преобразование: строковые данные преобразуются в число с плавающей точкой. Например:

```
AnsiString S = "12.5";
float x = StrToFloat(S);
```

Функции выдачи сообщений и перевода данных из одного типа в другой

- `IntToStr()` — преобразует целое число в строку типа `AnsiString`. Например:

```
int x = -12;
AnsiString S = IntToStr(x);
```

- `StrToInt()` — выполняет обратное преобразование: строковые данные преобразует в целое число. Например:

```
AnsiString S = "-12";
int x = StrToInt(S);
```

Функции выдачи сообщений и перевода данных из одного типа в другой

Для перевода изображения числа (строкового значения из текстового поля) в число (числовое значение – переменную числового типа) воспользоваться следующими процедурами:

StrToFloat(имя объекта -> свойство) – для вещественных чисел

StrToInt(имя объекта -> свойство) – для целых чисел

Например, блок ввода исходных данных может выглядеть следующим образом:

{ввод исходных данных из полей редактирования }

```
m = StrToFloat(Edit1->Text);
c = StrToInt(Label1 -> Caption);
c = Edit1->Text.ToInt();
```

Функции выдачи сообщений и перевода данных из одного типа в другой

Для преобразования численного значения переменной в строковое для помещения в текстовое поле вывода, например, в компоненте **Label**, воспользоваться процедурами:

`FloatToStr(имя переменной) – для вещественных чисел`

`FloatToStrF(имя переменной, формат вывода)`

`IntToStr(имя переменной) – для целых чисел`

ЗАМЕЧАНИЕ: можно использовать форматы вывода чисел

Функции выдачи сообщений и перевода данных из одного типа в другой

```
Edit1->Text = “Ответ ” + Chr(#13) +
FloatToStr(m);   – для вещественных
чисел
Label1->Caption = IntToStr(m); – для
целых чисел
```

Функции выдачи сообщений и перевода данных из одного типа в другой

Функции вывода текстовых сообщений на жкран

- `ShowMessage()` — выводит сообщение в специальном окне с кнопкой **OK**, которая организует ожидание. Имя исполняемого приложения выводится в заголовочной части окна. Обращение к этой функции:

```
AnsiString MSg = "Пример текстового сообщения";
ShowMessage(MSg);
```

- `MessageBox()` — эта функция выводит окно с сообщением. Окно содержит несколько кнопок, обеспечивающих выбор ситуации, которую можно обрабатывать. Вместе с окном сообщения функция выводит значение клавиши, которая была нажата в ответ на сообщение. Вид функции:

Функции выдачи сообщений и перевода данных из одного типа в другой

Функции вывода текстовых сообщений на экран

```
int MessageBox(hWnd, lpText, lpCaption, uType);
```

где `hWnd` — дескриптор окна, в которое попадет окно сообщений, `lpText` — текст сообщения, которое попадет в поле окна сообщения, `lpCaption` — текст сообщения, которое попадет в заголовок окна сообщения, `uType` — комбинация битовых флажков, определяющих, какие кнопки и какие значки должны появиться в окне сообщений.

Примечание

Дескриптор окна, в которое попадет окно сообщений, по умолчанию имеет значение `NULL`.

Функции выдачи сообщений и перевода данных из одного типа в другой

Возможные комбинации битов для определения кнопок задаются в виде:

- MB_YESNO — в окне сообщения будут две кнопки: **Да** (Yes) и **Нет** (No);
- MB_OK (значение по умолчанию) — одна кнопка **OK**;
- MB_OKCANCEL — две кнопки: **OK** и **Отмена** (Cancel);
- MB_RETRYCANCEL — две кнопки: **Повторить** (Retry) и **Отмена** (Cancel);
- MB_YESNOCANCEL — три кнопки: **Да** (Yes), **Нет** (No) и **Отмена** (Cancel).

Какой значок появится в поле сообщения, определяется заданием следующих флажков, которые записываются после задания кнопок через операцию ИЛИ (|): MB_ICONINFORMATION, MB_ICONEXCLAMATION, MB_ICONSTOP, MB_ICONQUESTION.

Функция MessageBox() возвращает значение кнопки, нажатой в ответ на сообщение. Значение можно применять, пользуясь табл. 11.1.

В операторы можно вставлять как символические константы, так и их числовые эквиваленты. Приведем пример приложения, использующего функцию MessageBox() (листинг 11.4).

```
int x = StrToInt(S);
```

Функции выдачи сообщений и перевода данных из одного типа в другой

Таблица 11.1. Значения, выдаваемые функцией MessageBox ()

Символическая константа	Значение
IDOK	1
IDCANCEL	2
IDABORT	3
IDRETRY	4
IDIGNORE	5
IDYES	6
IDNO	7

Функции выдачи сообщений и перевода данных из одного типа в другой

Функции вывода текстовых сообщений на жкран

Листинг 11.4

```
void __faStcall TForm1::Button1Click(TObject *Sender)
{
  AnSiString S = "250,8";
  Currency x = StrToCurr(S);
  int i = MessageBox(NULL, CurrToStr(x).c_Str(),
                     "Сведения о валюте", MB_YESNOCANCEL | MB_ICONSTOP);
  if(i == IDYES || i == IDNO)
    return;
  else
    exit(0);
}
```

Функции выдачи сообщений и перевода данных из одного типа в другой

Функция MessageDlg используется для вывода на экран messagebox с неким сообщением и несколькими кнопками (Yes, No, OK и т. д.).

Параметры у этой функции следующие. Первый задает строку, которая будет показываться в нашем сообщении (Как и все строки в C++Builder, заключаем ее в двойные кавычки). Второй параметр определяет иконку на нашем Messagebox и сообщение в его заголовке. Возможные значения для этого параметра - mtConfirmation (Подтверждение), mtInformation (Сообщение), mtWarning (Предупреждение), mtError (Ошибка).

ЗАМЕЧАНИЕ: при записи любого из перечисленных значений Messagebox получит и соответствующий заголовок. Для этого параметра есть еще одно значение - mtCustom. Если использовать его, то никакой иконки в нашем messagebox не будет, а заголовок будет совпадать с именем exe-файла.

Функции выдачи сообщений и перевода данных из одного типа в другой

Третий параметр определяет кнопки, которые будут присутствовать на messagebox. Вот некоторые из возможных значений: mbYes (кнопка Yes), mbNo (кнопка No), mbCancel (кнопка Cancel), mbOK (кнопка OK).

Если кнопок несколько, то они объединяются посредством символов << (например, если мы хотим, чтобы присутствовало две кнопки Yes и No, то пишем << mbYes << mbNo). Так же необходимо написать ключевое слово TMsgDlgButtons().

Пример: TMsgDlgButtons() << mbYes << mbNo.

Функции выдачи сообщений и перевода данных из одного типа в другой

Функции вывода текстовых сообщений на жкран

Последний, четвертый параметр - это идентификатор контекстной справки.

В качестве результата функции MessageDlg возвращает значение, информирующее о том, что за кнопку пользователь нажал в messagebox.

Вот несколько возможных значений: mrYes, mrCancel, mrOK.

Функции выдачи сообщений и перевода данных из одного типа в другой

Функции вывода текстовых сообщений на жкран

Пример использования функции MessageDlg:

```
if (MessageDlg("Закрыть приложение?",
mtConfirmation, TMsgDlgButtons() << mbYes << mbNo,0)
== mrYes)
        {
                Close();
        }
```

В этом фрагменте у пользователя спрашивается, надо ли выходить из программы, и при положительном ответе программа закрывается.

Функции выдачи сообщений и перевода данных из одного типа в другой

Функция InputBox отображает диалоговое окно для *ввода строки* текста.

```
AnsiString InputBox (AnsiString ACaption, AnsiString APrompt, AnsiString ADefault)
```

Окно выводится в центре экрана и содержит поле ввода с надписью, а также кнопки **ОК** и **Cancel.**

Функции выдачи сообщений и перевода данных из одного типа в другой

Параметр ACaption задает заголовок окна.

Параметр APrompt содержит поясняющий текст к полю ввода.

Параметр ADefault определяет строку, возвращаемую функцией при отказе пользователя от ввода информации (нажатие кнопки **Cancel** или клавиши <Esc>).

Функции выдачи сообщений и перевода данных из одного типа в другой

Пример использования функции InputBox:

```
void __fastcall
TForm1::Button3Click(TObject *Sender)
{
   AnsiString soname ;
   soname = InputBox("Пользователь",
   "Введите фамилию", "Иванов");
}
```

Приведенная функция отображает окно запроса на ввод фамилии пользователя. По умолчанию предлагается Иванов.

Функции выдачи сообщений и перевода данных из одного типа в другой

Функция ShowMessage. У неё только один параметр - текст сообщения.

Пример:

```
ShowMessage(“Сообщение”);
```

Эта функция используется для вывода простых сообщений, которые не требуют ответа от пользователя (типа Yes, No и т. д.).

Использование диалогов для выбора файлов

Компоненты ***TOpenDialog*** и ***TSaveDialog*** находятся на странице *Dialogs* палитры компонентов.

ЗАМЕЧАНИЕ: Все компоненты этой страницы являются не визуальными, т. е. не видны в момент работы программы. Поэтому их можно разместить в любом удобном месте формы. Оба рассматриваемых компонента имеют идентичные свойства и отличаются только внешним видом.

Использование диалогов для выбора файлов

Диалоговое окно компонентов открывается после вызова метода ***Execute***, который возвращает *true*, если пользователь выбрал файл и нажал «Ок», и *false*, если пользователь отказался от выбора.

Например:

```
if (OpenDialog1->Excecute)
{ // код, выполняющийся после выборе
// файла пользователем
}
```

Использование диалогов для выбора файлов

Свойства компонента ***OpenDialog***

1. В случае успешного завершения диалога имя выбранного файла поиска содержится в свойстве *FileName*.
2. Для фильтрации файлов, отображаемых в окне просмотра, используется свойство *Filter*, а для задания расширения файла, в случае, если оно не задано пользователем, – свойство *DefaultExt*.
3. Если необходимо изменить заголовок диалогового окна, используется свойство *Title*.

Использование диалогов для выбора файлов

Пример использования свойства **Filter:**

в диалоговом окне **Filter Editor** производится настройка типов файлов.

Окно разбито на два поля. В левом поле **Filter Name** вводятся строки пояснения. В правом поле **Filter** — расширения файлов. Для указания нескольких типов файлов необходимо записать их через точку с запятой.

Если задать несколько строк с пояснениями и типами файлов, то при работе приложения их можно будет выбирать с помощью выпадающего списка.

Использование диалогов для выбора файлов

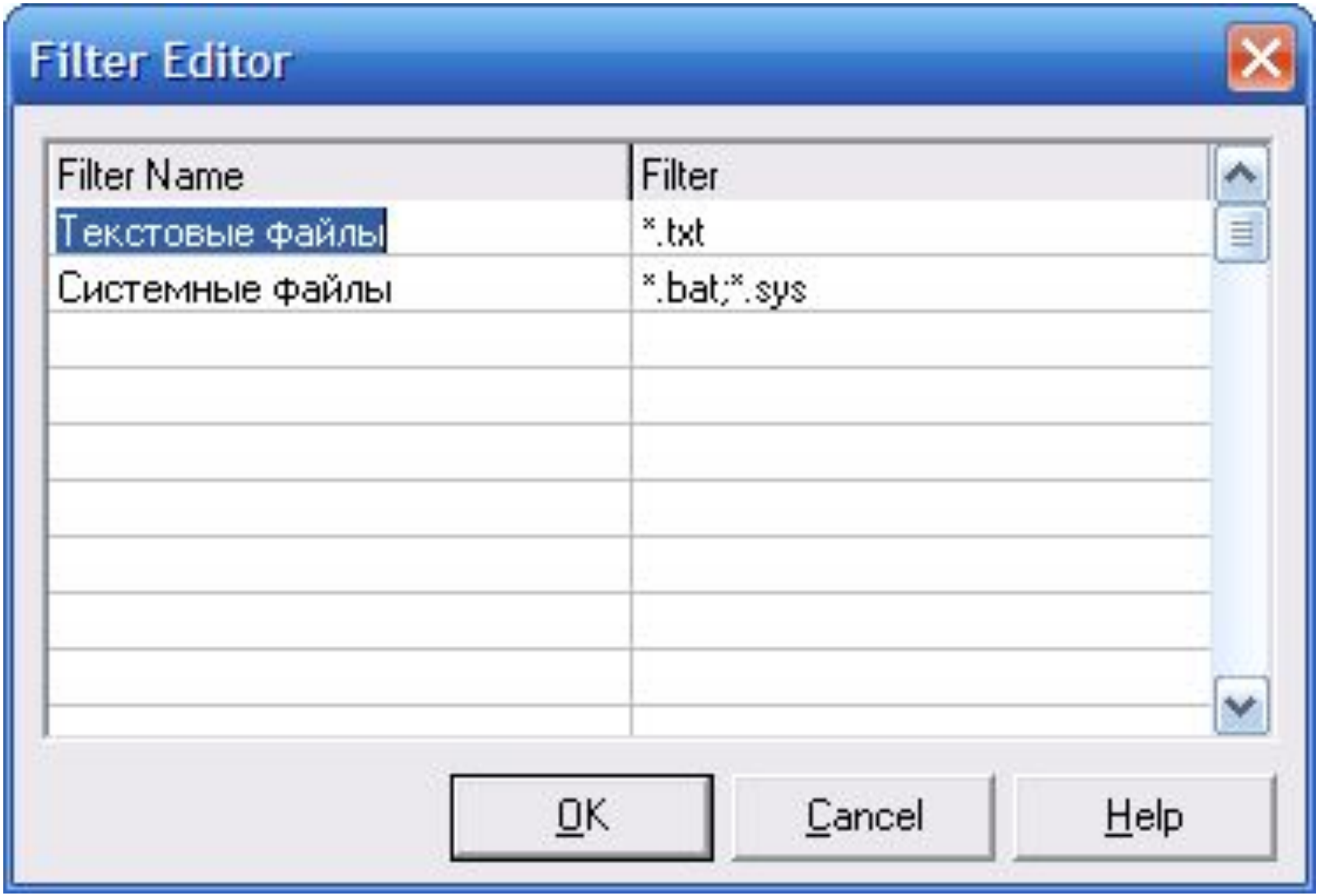


Рис. Окно Filter Editor

Использование диалогов для выбора файлов

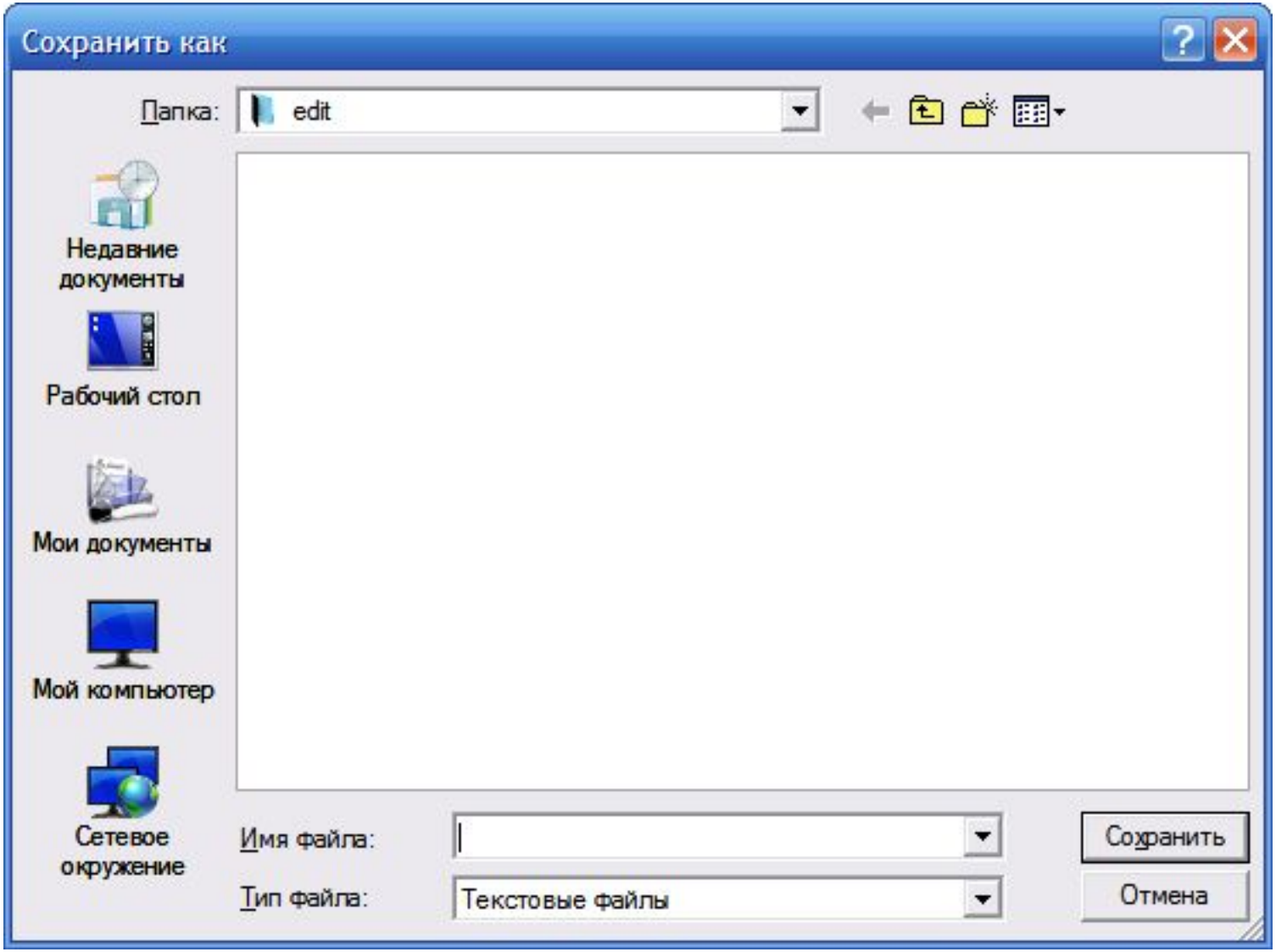
Аналогично выполняется настройка компонента **SaveDialog1**.

ЗАМЕЧАНИЕ: Для корректности работы приложения оба эти диалога должны иметь одинаковую настройку свойства **Filter**

Использование диалогов для выбора файлов

ЗАМЕЧАНИЕ: работу этих компонентов можно проверить еще до выполнения программы. Дело в том, что после задания свойства **Filter** этих компонентов двойной щелчок по компонентам приводит к открытию диалога на основе заданных свойств. Например, после двойного щелчка по компоненту **SaveDialog1** откроется следующее окно:

Использование диалогов для выбора файлов



Работа с файлами с использованием функций компонент

Многие визуальные компоненты (**Memo**, ListBox, ComboBox, RichEdit) и некоторые классы (TStringList) имеют доступ к функциям:

```
LoadFromFile("Имя файла");
SaveToFile("Имя файла");
```

ЗАМЕЧАНИЕ: Эти функции особенно удобны для работы с текстовыми файлами.

Работа с файлами с использованием функций компонент

```
//Визуальный компонент TMemo:
Memo1->Lines->Clear();
Memo1->Lines->LoadFromFile("a.txt");
......
Memo1->Lines->SaveToFile("a.txt");
......
```

При использовании функции SaveToFile если файл не существует, то он будет создан.

Работа с файлами с использованием функций компонент

Имеется счетчик строк:

```
Memo1->Lines->Count;
```

Содержимое строк можно получит по номеру строки n:

```
AnsiString vasS;
vasS = Memo1->Lines->Strings[n];
```

Работа с файлами с использованием функций компонент

Возможно присвоить некоторой строке текст для дальнейшей манипуляции содержимым текста как единой строкой и далее перенести исправленный текст опять в компоненты:

```
AnsiString vasS;
vasS=Memo1->Lines->GetText();
```

... манипуляции с текстом как единой строкой

```
Memo1->Lines->SetText(vasS.c_str());
```

Работа с файлами с использованием функций компонент

Многие визуальные компоненты и некоторые классы имеют доступ к функциям:

```
LoadFromFile("Имя файла");
SaveToFile("Имя файла");
```

ЗАМЕЧАНИЕ: Эти функции особенно удобны для работы с текстовыми фыйлами.

Работа с датами

В C++Builder существует специальный класс TDateTime, который содержит методы работы с датами. Этот класс в качестве члена данных включает в себя переменную типа double, которая в своей целой части содержит дату, а в дробной время

Точка отсчета даты – 30.12.1899г. 12 часов 00 минут.

Значение даты в это время принято считать равным нулю.

Ненулевое значение. вычисляется так: к точке отсчета прибавляется целая часть даты (т.е. количество дней).

Работа с датами

Функции и методы работы с датой:

`DateToStr()` — преобразует дату в строку типа **`AnsiString`**;

`DateTimeToStr()` — преобразует дату и время в строку типа **`AnsiString`**;

`TimeToStr()` — преобразует время в строку типа **`AnsiString`**.

Работа с датами

Чтобы работать с датой/временем, следует объявить переменную типа TDateTime.
Например:

```
TDateTime dl,d;
```

А затем использовать методы этого класса.

`CurrentDate()` - возвращает текущую дату как значение типа TDateTime.

Значение времени возвращается равным нулю.

`CurrentDateTime ()` — возвращает текущую дату и время как значение типа TDateTime.

Работа с датами

`CurrentTime ()` — возвращает текущее время как значение типа TDateTime с нулевым значением даты.

`DayOfWeek()` - возвращает день недели между 1 и 7. Воскресенье считается первым днем недели, суббота — последним.

`DecodeDate()` — разделяет значение даты типа TDateTime на год, месяц и число и помещает эти значения в соответствующие параметры: year, month, day.

Работа с датами

`DecodeTime()` — выделяет значения часов, минут, секунд и миллисекунд из времени даты и помещает их в соответствующие параметры hour, min, sec, msec.

`FormatString()` — форматирует TDateTime-объект, используя определенные форматы.

Построение графиков

Обычно результаты расчетов представляются в виде графиков и диаграмм.

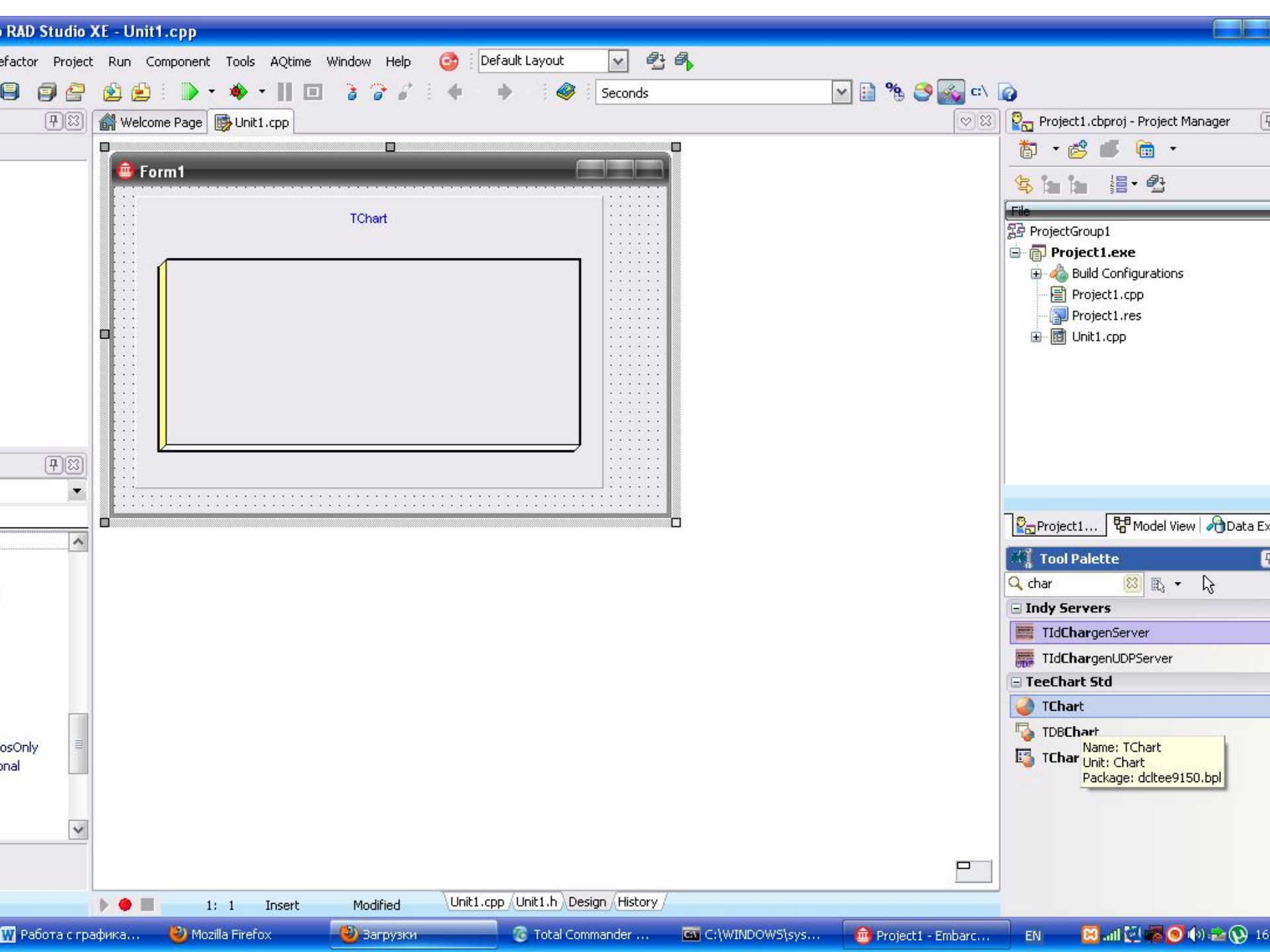
Способы построения графиков и диаграмм:

1. Использование компонента **TChart**
2. Использование свойства **Canvas** формы

Построение графиков

1. Использование компонента **`Tchart`**

Система *C++Builder* имеет мощный пакет стандартных программ вывода на экран и редактирования графической информации, который реализуется с помощью визуально отображаемого на форме компонента *TChart* (вкладка **TeeCartStd**)


RAD Studio XE - Unit1.cpp
efactor Project Run Component Tools AQtime Window Help
Default Layout
Seconds
Welcome Page
Unit1.cpp
Form1
TChart
Project1.cbproj - Project Manager
File
ProjectGroup1
Project1.exe
Build Configurations
Project1.cpp
Project1.res
Unit1.cpp
Project1...
Model View
Data Ex
Tool Palette
char
Indy Servers
TIdChargenServer
TIdChargenUDPServer
TeeChart Std
TChart
TDBChart
TChar
Name: TChart
Unit: Chart
Package: dcltee9150.bpl
osOnly
onal
1: 1
Insert
Modified
Unit1.cpp
Unit1.h
Design
History
Работа с графика...
Mozilla Firefox
Загрузки
Total Commander ...
C:\WINDOWS\sys...
Project1 - Embarc...
EN

Построение графиков

Компонент *TChart* осуществляет всю работу по отображению графиков, переданных в объект $Series_k$: строит и размечает оси, рисует координатную сетку, подписывает название осей и самого графика, отображает переданную таблицу в виде всевозможных графиков или диаграмм.

При необходимости, с помощью встроенного редактора *EditingChart* компоненту *TChart* передаются данные о толщине, стиле и цвете линий, параметрах шрифта подписей, шагах разметки координатной сетки и другие настройки.

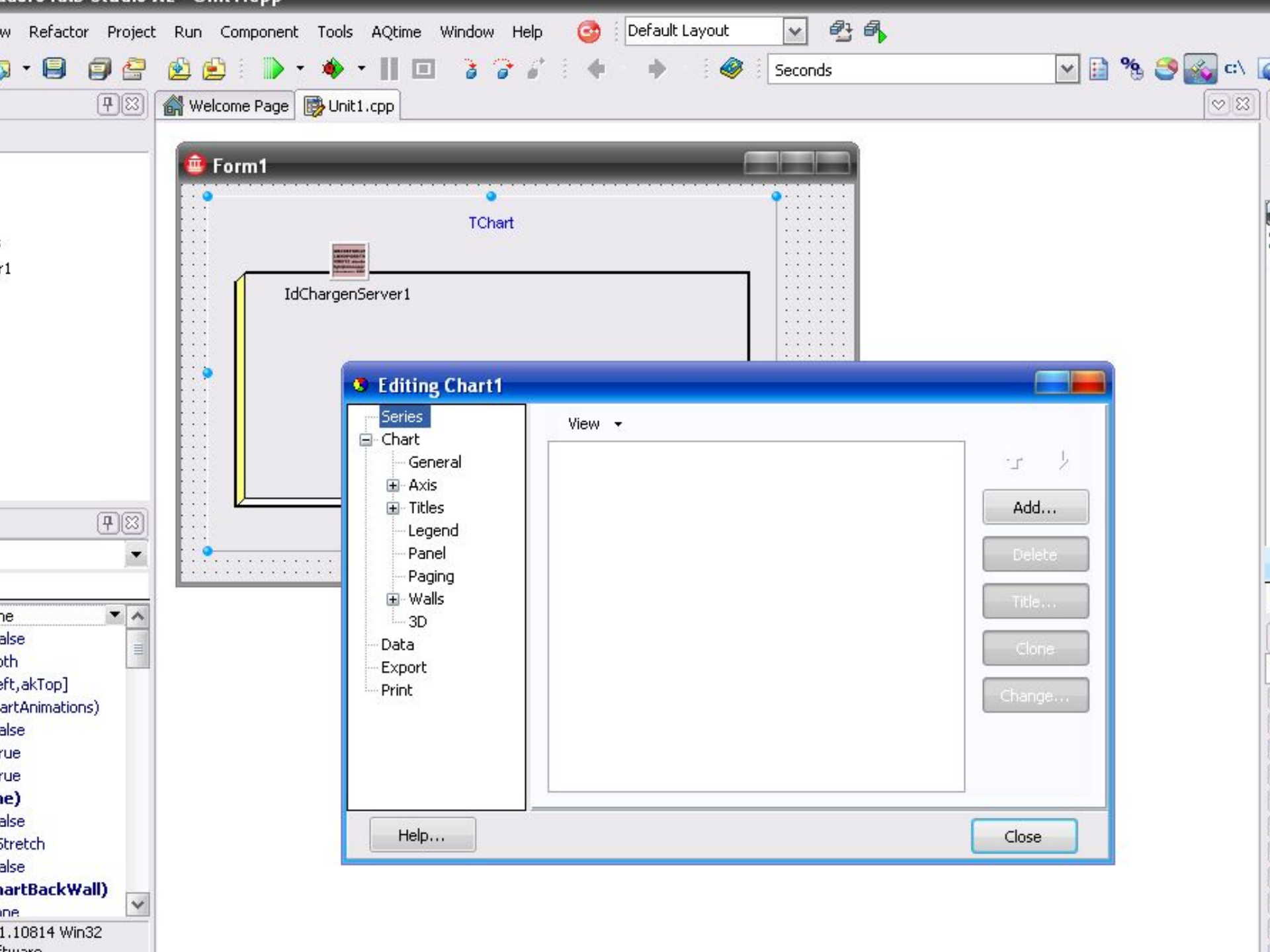
Построение графиков

Вызов редактора *EditingChart:*

-двойной щелчок на объекте Chart1

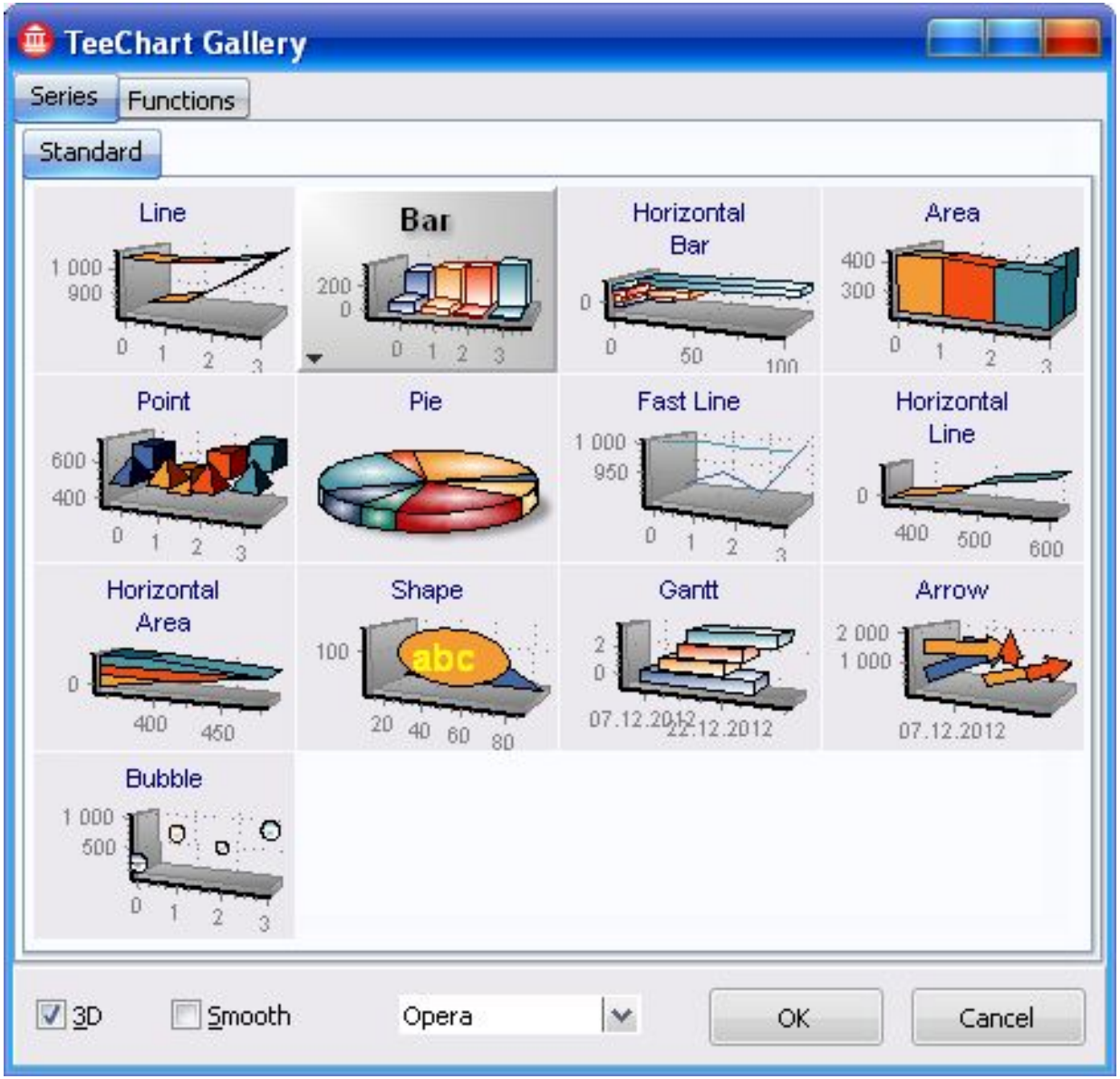
-контекстное меню на объекте Chart1 и выбрать Edit Chart…

ЗАМЕЧАНИЕ: график можно настраивать и изменять параметры и в ходе выполнения программы, то есть его настройка не ограничивается одним окошком.


Refactor
Project
Run
Component
Tools
AQtime
Window
Help
Default Layout
Seconds
Welcome Page
Unit1.cpp
Form1
TChart
IdChargenServer1
Editing Chart1
Series
Chart
General
Axis
Titles
Legend
Panel
Paging
Walls
3D
Data
Export
Print
View
Add...
Delete
Title...
Clone
Change...
Help...
Close

Построение графиков

- Для создания нового объекта *Series1* следует щелкнуть по кнопке *Add* на странице *Series*.



Построение графиков

- Для изменения надписи над графиком используется страница *Titles.*
- Для разметки осей необходимо выбрать страницу *Axis* и установить параметры настройки осей.

Построение графиков

В процессе работы программы изменение параметров возможно через обращение к соответствующим свойствам компонента *TChart*.

Например, свойство *Chart1->BottomAxis* содержит значение максимального предела нижней оси графика и при его изменении во время работы программы автоматически изменяется изображение графика.
ЗАМЕЧАНИЕ: Когда свойство *Chart1->BottomAxis->Automatic* имеет значения *false*, автоматическая установка параметров осей не работает.

Построение графиков

Для добавления координат точек из таблицы в двумерный массив объекта *Seriesk* используется функция

`int Series1->AddXY(AXValue;AYValue; AXLabel;AColor)` ,

где *AXValue, AYValue* – координаты точки по осям *X* и *Y*;

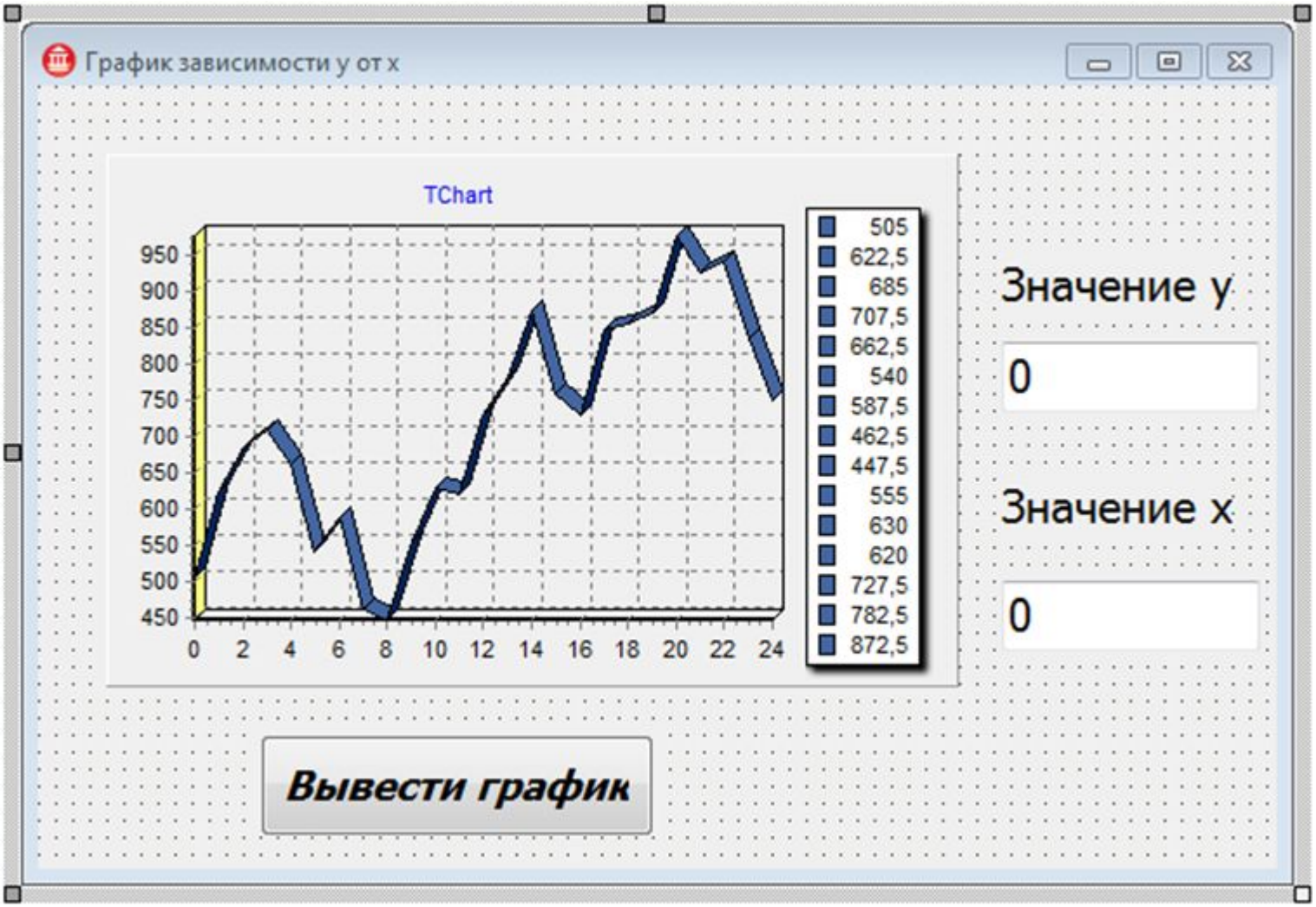
AXLabel – текстовая надпись добавленной точки;

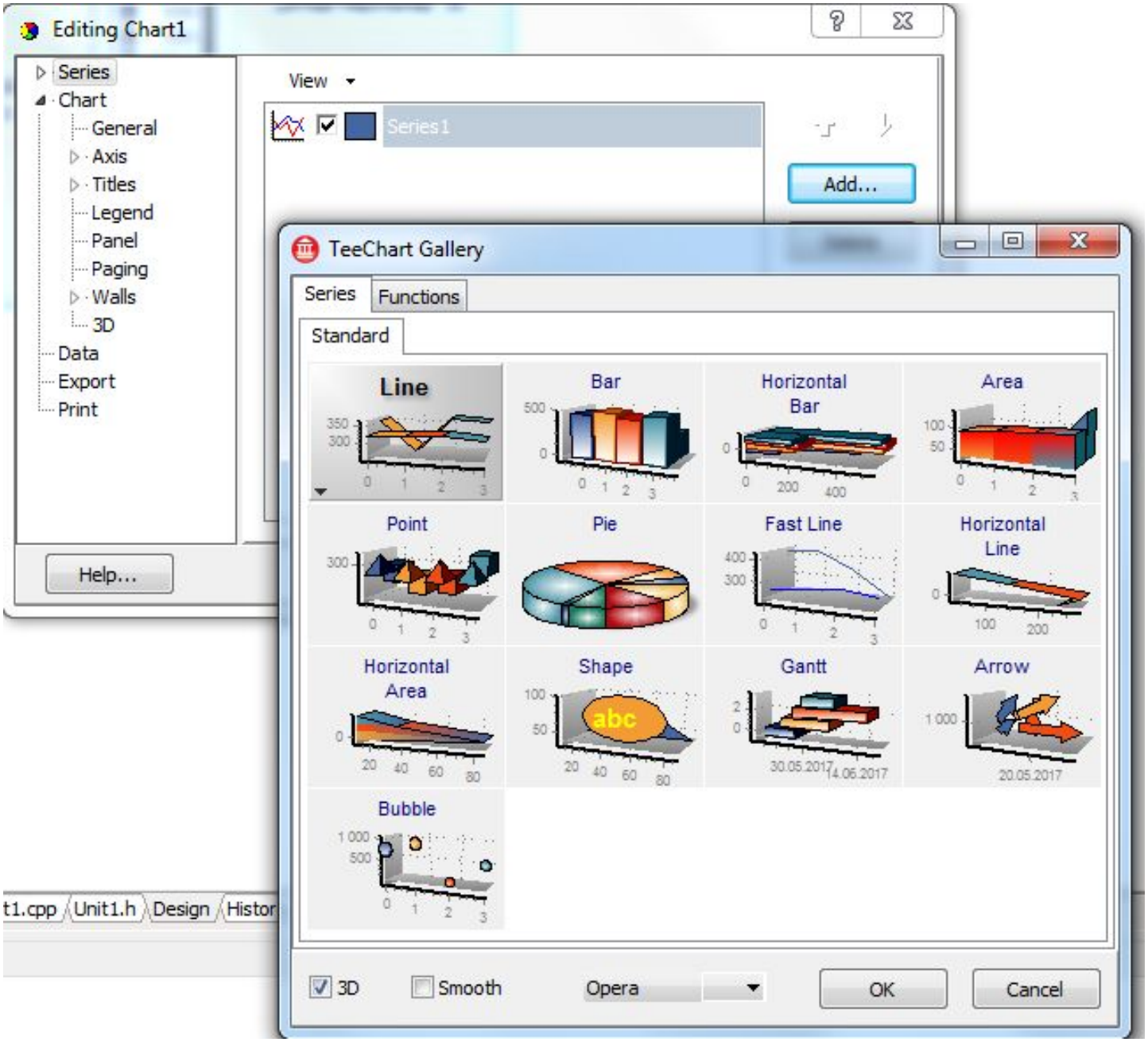
AColor задает цвет линий (если равен *clTeeColor*, то принимается цвет, определенный при проектировании формы).

Например: **`Series1->Add(y,x,clRed);`**

Построение графиков

Пример: построить график зависимости x от y




Editing Chart1
Series
Chart
General
Axis
Titles
Legend
Panel
Paging
Walls
3D
Data
Export
Print
View
Series1
Add...
Help...
TeeChart Gallery
Series
Functions
Standard
Line
Bar
Horizontal Bar
Area
Point
Pie
Fast Line
Horizontal Line
Horizontal Area
Shape
Gantt
Arrow
Bubble
3D
Smooth
Opera
OK
Cancel
t1.cpp
Unit1.h
Design
Histor

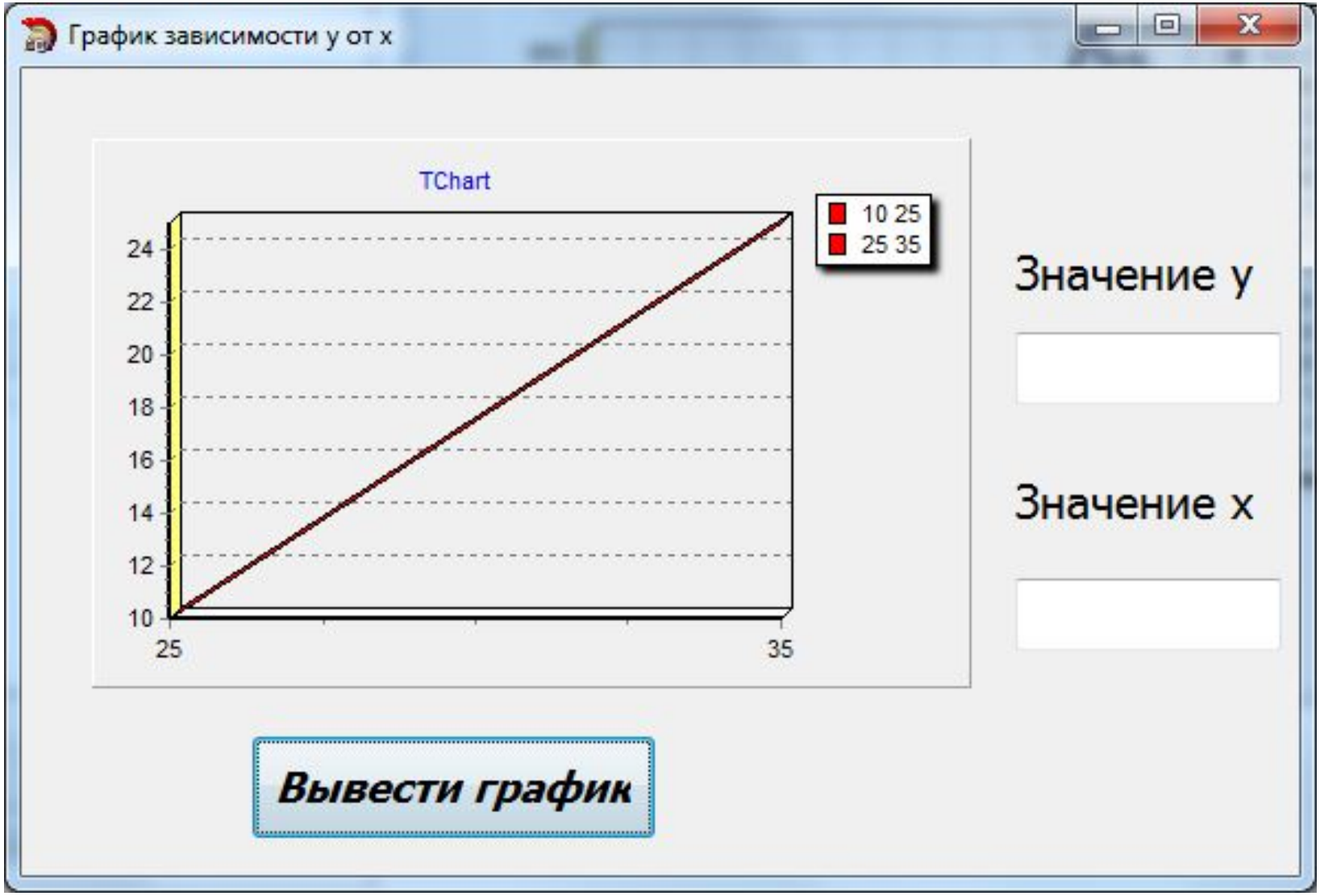


График зависимости y от x
TChart
10 25
25 35
24
22
20
18
16
14
12
10
25
35
Значение y
Значение x
Вывести график

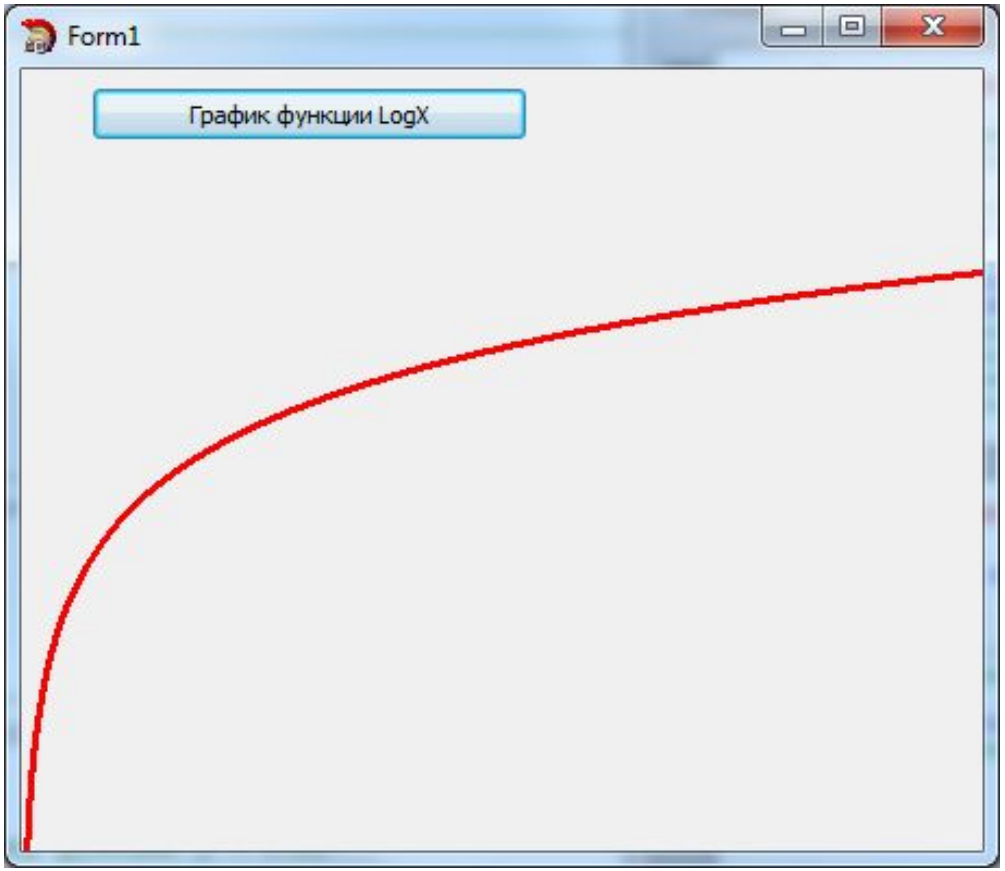
Построение графиков

Программируем кнопку

```
if ((Edit1->Text=="") || (Edit2->Text==""))
{
     ShowMessage("Введите значения x и y");
     return ;
}
// считываем значения с Edit1 в y, Edit2 в x
int x,y;
y = Edit1->Text.ToInt(); // y =
StrToInt(Edit1->Text);
x = Edit2->Text.ToInt();
//выводим текущие значения x и y на графике
Series1->Add(y,x,clRed);
// очищаем поля
Edit1->Text=""; Edit2->Text="";
```

Построение графиков

2. Рисование графиков с помощью свойства объекта TForm – Canvas



Построение графиков

TCanvas (Канва) - это класс, предназначенный для вывода и хранения графических объектов в **C++ Builder**.
Канва входит в состав большинства визуальных компонентов, кроме стандартных оконных контролов **TButton**, **TMemo**, **TPanel** и т.п.).
При помощи методов этого класса можно рисовать как и стандартные примитивы (линии, эллипсы, прямоугольники), так и графические объекты типа **Graphics**::**TBitmap**.

Построение графиков

Доступ к канве любого объекта происходит следующим образом:

```
имя_объекта->Canvas->Свойство/Метод;
```

Канва, в ее графическом представлении, это двумерный массив пикселей. Каждый элемент этого массива хранит в себе информацию о цвете. Доступ к отдельно взятому пикселю можно получить из свойства **Pixels**.

Точка (0,0) - эта верхний левый угол канвы. Значение по **x**-координате возрастает слева направо от точки (0,0) до **width** – свойства TForm, а значение по **y**-координате сверху вниз - до **height**.

Построение графиков

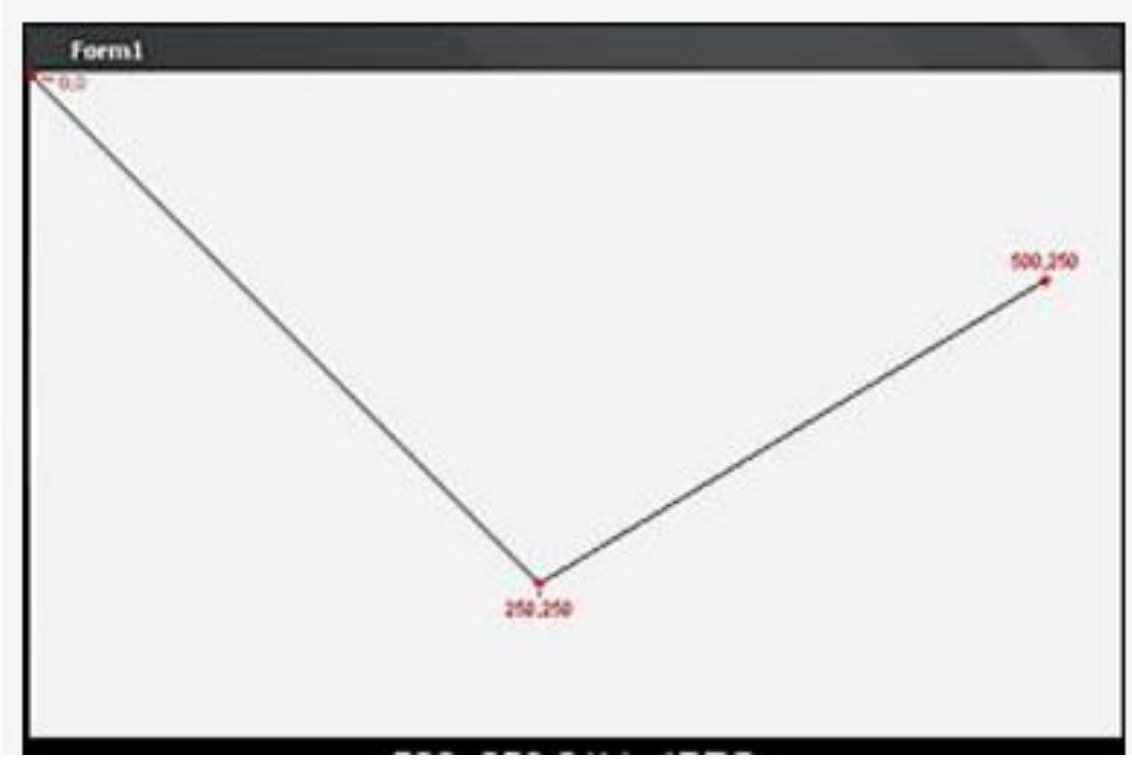
Методы класса **`TCanvas-`**, используемые для создания графика:

- **`MoveTo(x,y)`** – перейти к точке холста с координатами (x,y) (в пикселах);
- **`LineTo(x,y)`** – нарисовать линию из предыдущей точки в точку с координатами (x, y).

Построение графиков

Пример 1:

```
//Проводим первую прямую
Form1->Canvas->LineTo(250,250);
//Теперь проводим другую прямую
Form1->Canvas->LineTo(500,100);
```



Построение графиков

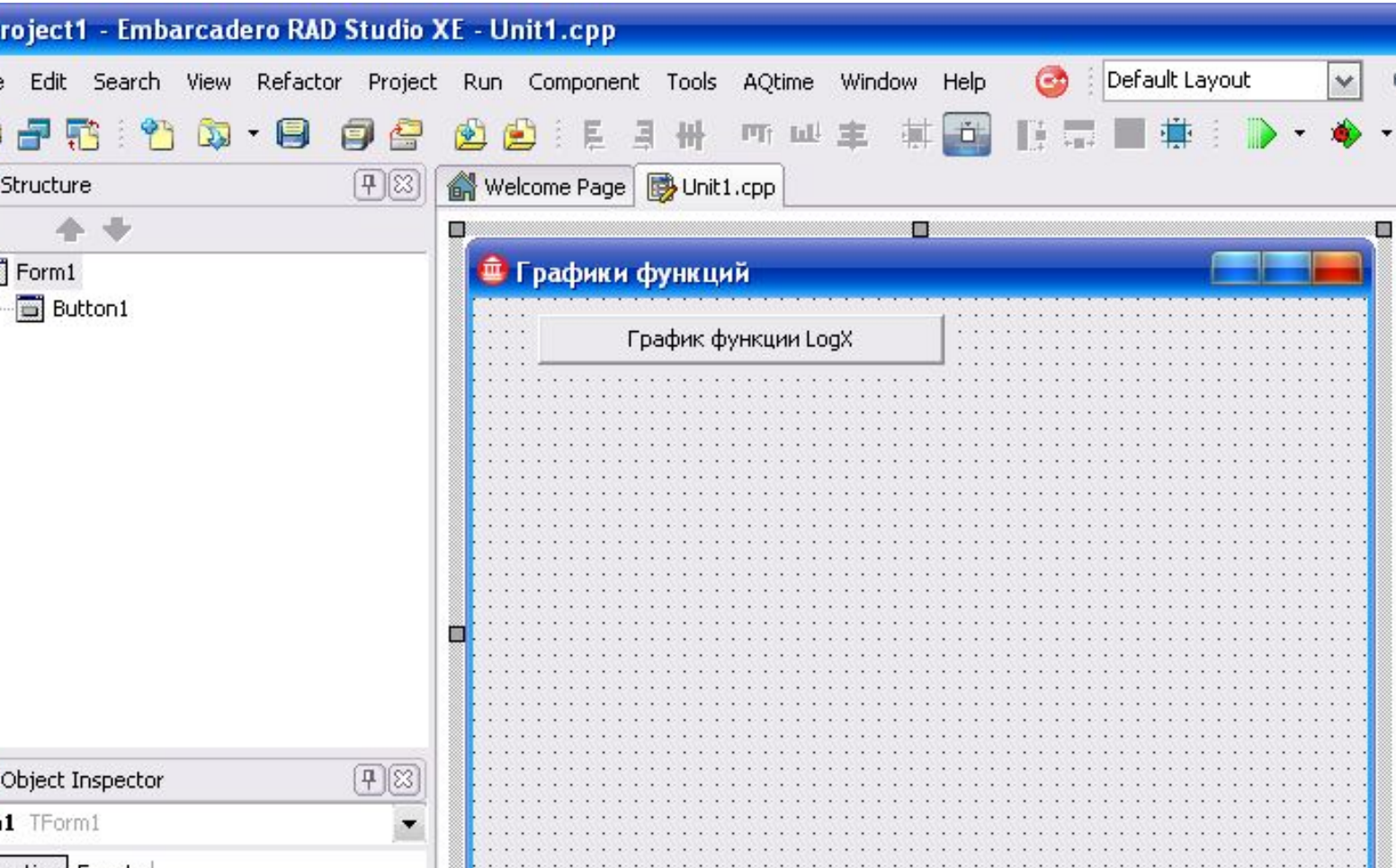
Пример 2:

```
//Перемещаем перо в точку (100,25)
Canvas->MoveTo(100,25);
//Проводим прямую в точку (500,100)
Canvas->LineTo(500,100);
```

Результат:

Построение графиков

Пример: нарисовать на форме график функции y= f(x) , например, y = logx



Построение графиков

Значение координаты x изменяется слева направо тоже через один пиксел. Если двигаться по форме, то текущие координаты точки, в которой мы будем находиться в данный момент, будут определяться величинами (x, `Height - y`). Чтобы нарисовать что-либо на форме, можно воспользоваться событием `OnPaint` (и не только им), которое возникает всякий раз, когда рисунок на форме изменяется. Общий вид обработчика этого события формы следующий:

Построение графиков

```
int x,y;
// начальные координаты          //функции
y = f(x)
y = x = 0;
//начальная точка графика
Form1->Canvas->MoveTo(0,Height);
//установим цвет и толщину графика
Form1->Canvas->Pen->Color = clRed;
Form1->Canvas->Pen->Width = 3;
// Вычисление функции y = f(x), где x
меняется на один пиксел
//от 0 до max. значения, равного ширине
формы (свойство width)
```

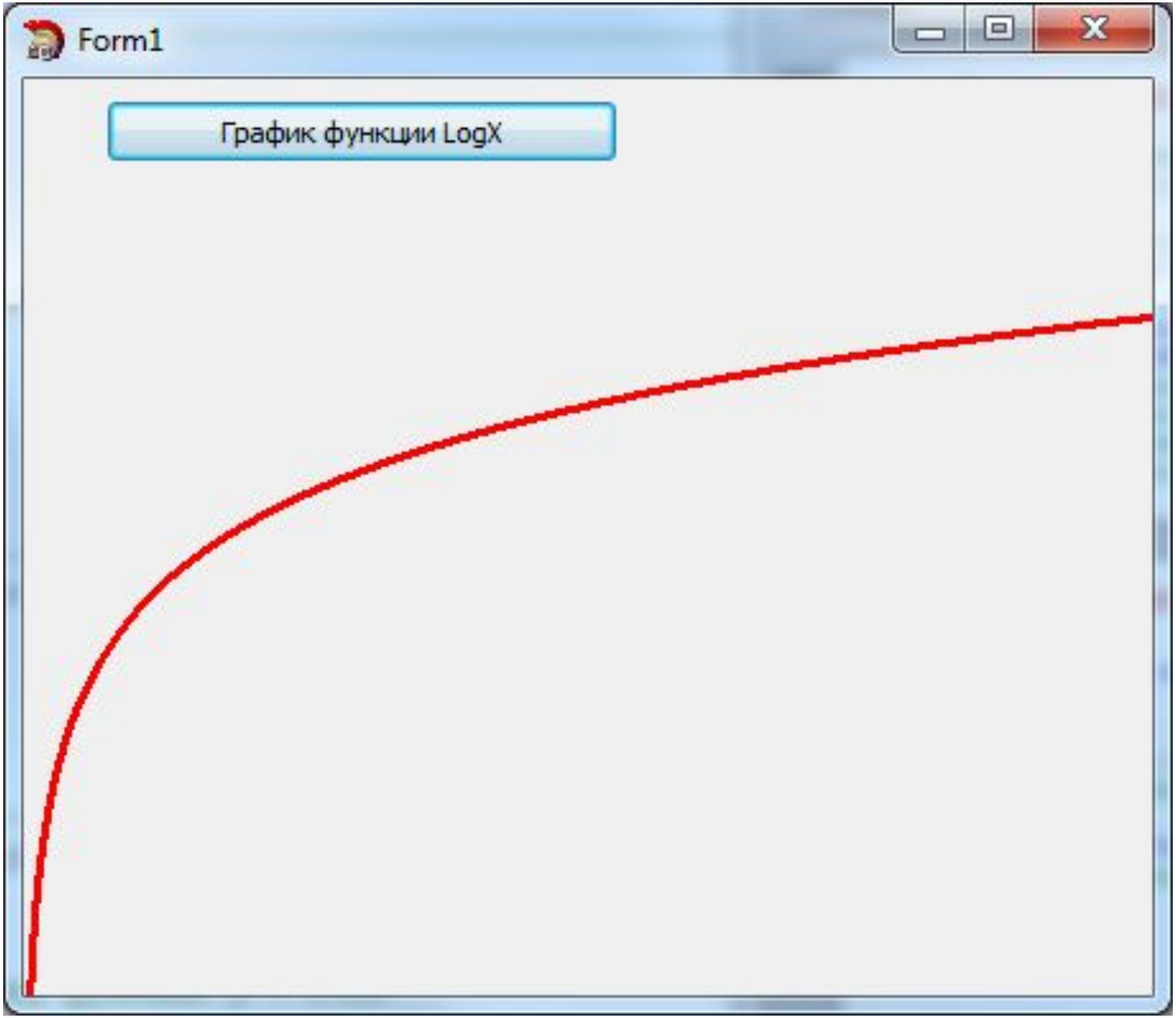
Построение графиков

```
while (x < Form1 -> Width && y <
                Form1->Height)
{
   x++;//следующая точка на оси x
// вызов пользовательской функции
   y = f(x);
   Form1->Canvas->LineTo(x,Height-y);
      //рисуем линию и первую точку
}
```

Построение графиков

Осталось написать конкретную программу для вычисления конкретной функции $y = f(x)$. Для этого к Unit1.cpp подключим h-файл, в котором описаны математические функции, и возьмем одну из них в качестве примера. Пусть это будет функция $y = \log(x)$. Учтем при этом, что мы работаем с пикселами (т. е. с целыми величинами), а логарифмическая функция возвращает дробные числа. Поэтому дробная часть будет отбрасываться с помощью функции `floor(x)`. Кроме того, чтобы график имел наглядность, увеличим его ординату в 50 раз. Теперь можно создать функцию, которая будет вызываться в основной программе и которая станет вычислять график логарифмической функции:

```
int f(int x)
{
    int y = floor(50 * log(x));
    return (y);
}
```


Form1
График функции LogX

Программирование закрытия приложения (окна)

В теле нужной процедуры в коде написать:

```
Application->Terminate(); // закрытие всего приложения
Close();                  //Закрытие  окна
```