Лекция № 11. Визуальное программирование на C++

Визуальное программирование в MS Visual C++ рассмотрим на базе диалогового окна.

11.1. Визуальное программирование в MS Visual C++

Для создания шаблона *диалогового окна* выполним команду **File – New – Project**. В появившемся диалоговом окне (рис. 11.1), в качестве типа проекта (Project types) выбираем MFC, а в качестве шаблона (Templates) – MFC Application. Щелкнув по кнопке OK, попадаем в первое окно мастера создания шаблона приложения (рис. 11.2).

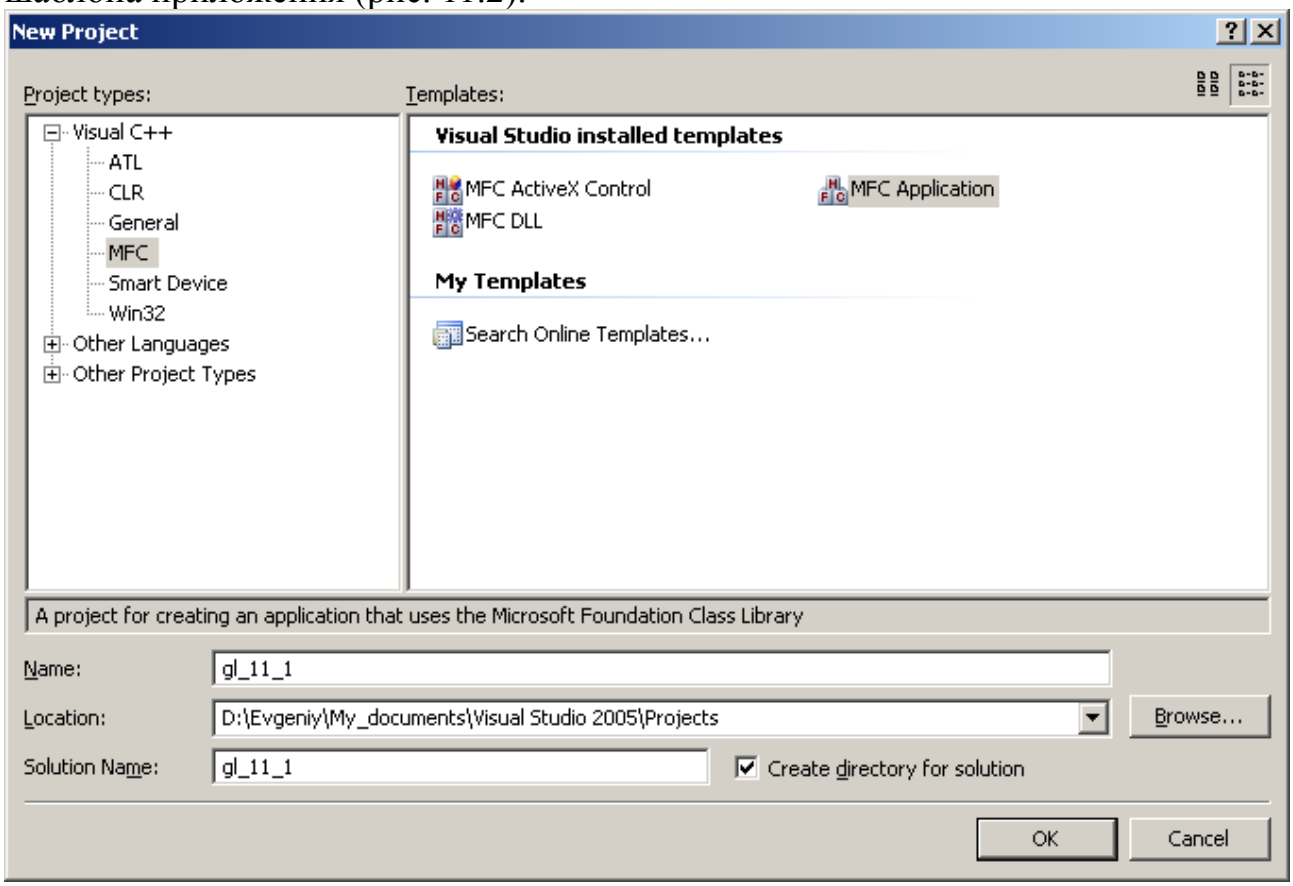

Рис. 11.1. Окно выбора типа создаваемого проекта

На следующем шаге (рис. 11.3) необходимо выбрать тип создаваемого приложения:

- **Single document** (Однооконное приложение);
- **Multiple documents** (Многооконное приложение);
- **Dialog based** (приложение – «диалоговое окно»).

Выбираем приложение типа диалоговое окно (**Dialog based**). На следующем шаге (рис. 11.4) выбираем в группе **Main frame styles** какие элементы должны быть в диалоговом окне (оставляем значения предлагаемые мастером создания шаблона приложения) и определяем заголовок нашего

диалогового окна (**Dialog title**). Следующее окно оставляем без изменений. На последнем этапе (рис. 11.5) пользователь увидит классы, сгенерированные мастером создания шаблона приложения. При выделении класса в верхнем окне в соответствующих текстовых полях, расположенных ниже, появляется имя этого класса, имя базового класса, имя файла заголовка, в котором содержится заголовок данного класса, и имя файла реализации, в котором содержится описание методов данного класса.

Если соответствующая информация расположена на белом фоне, то она может быть изменена, если она расположена на сером фоне, то она не подлежит редактированию. После щелчка по кнопке **Finish** будет создан шаблон приложения типа «Диалоговое окно» (рис. 11.6).

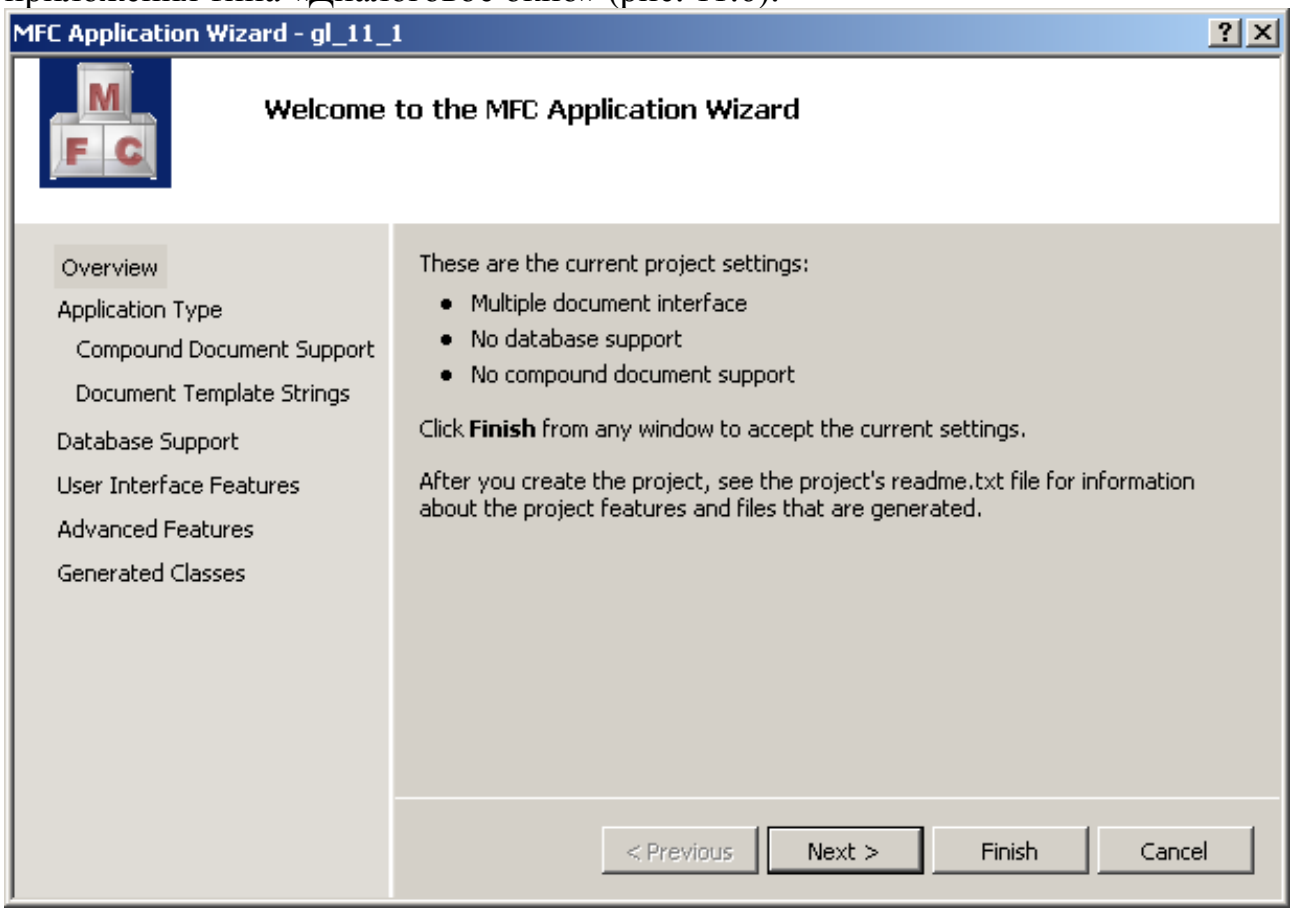

Рис. 11.2. Первое окно мастера создания шаблона приложения

Можно попытаться запустить это приложение, нажав **Ctrl+F5**. При запуске появиться диалоговое окно с именем **First program**, окно будет перемещаться по экрану, и закрываться с помощью кнопки закрытия окна. Кнопки **Cancel** и **OK** также будут закрывать окно. Больше это приложение ничего делать не умеет.

При создании диалогового окна, необходимо указать язык разрабатываемого приложения (рис. 11.3), однако, русского языка среди предлагаемых компанией Microsoft нет. На сайте http://www.flint-inc.ru/Russian/Programs.html можно скачать небольшую программу, которая после инсталляции добавляет русский язык в список языков разрабатываемого

приложения. После его установки, в окне выбора типа приложения (рис. 3) в параметре **Resource Language** появится русский язык. После этого без проблем можно создавать элементы разрабатываемого приложения на русском языке.

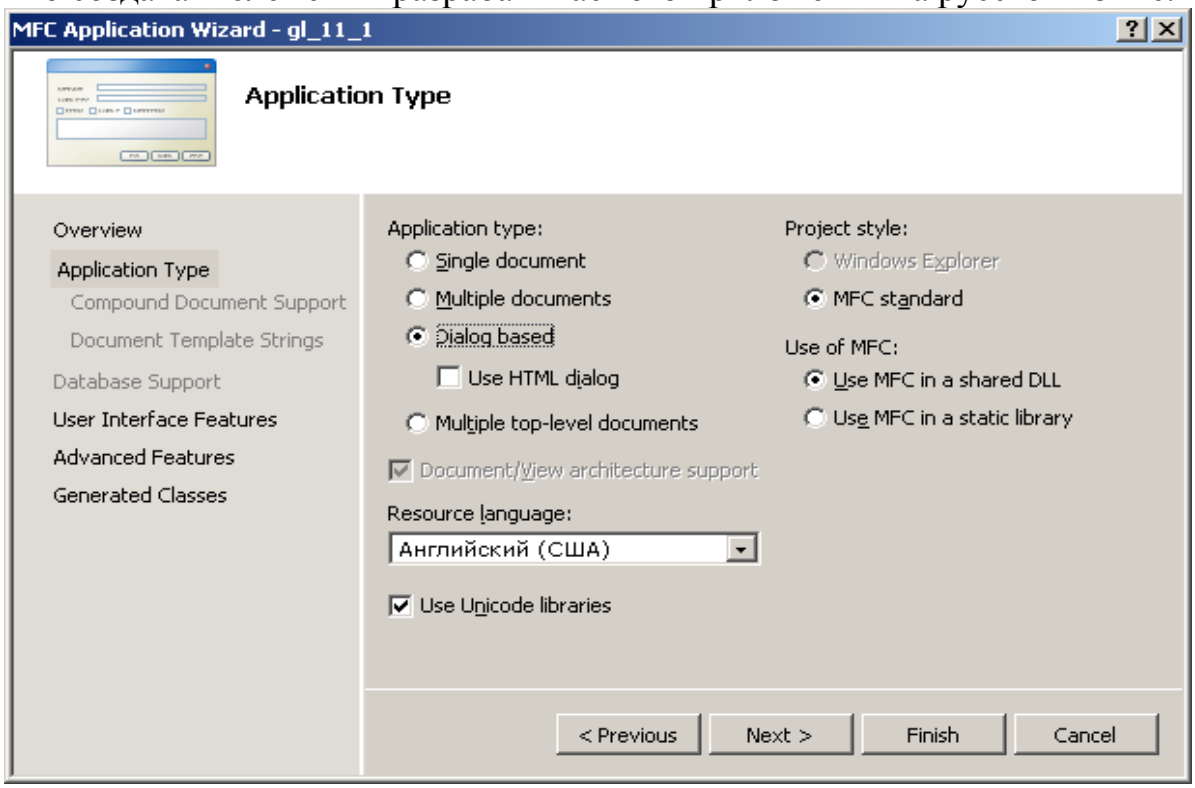

Рис. 11.3. Выбор типа приложения

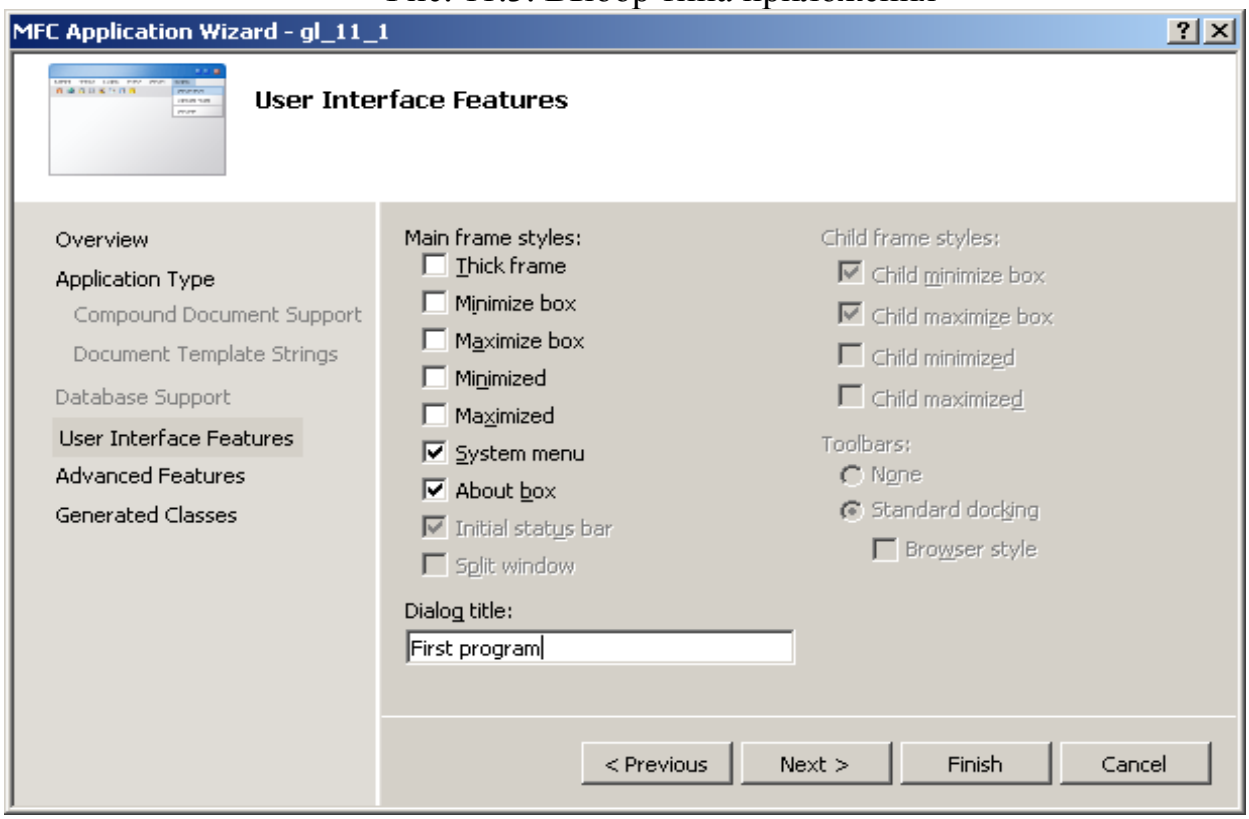

Рис. 11.4. Выбор оформления создаваемого диалогового окна

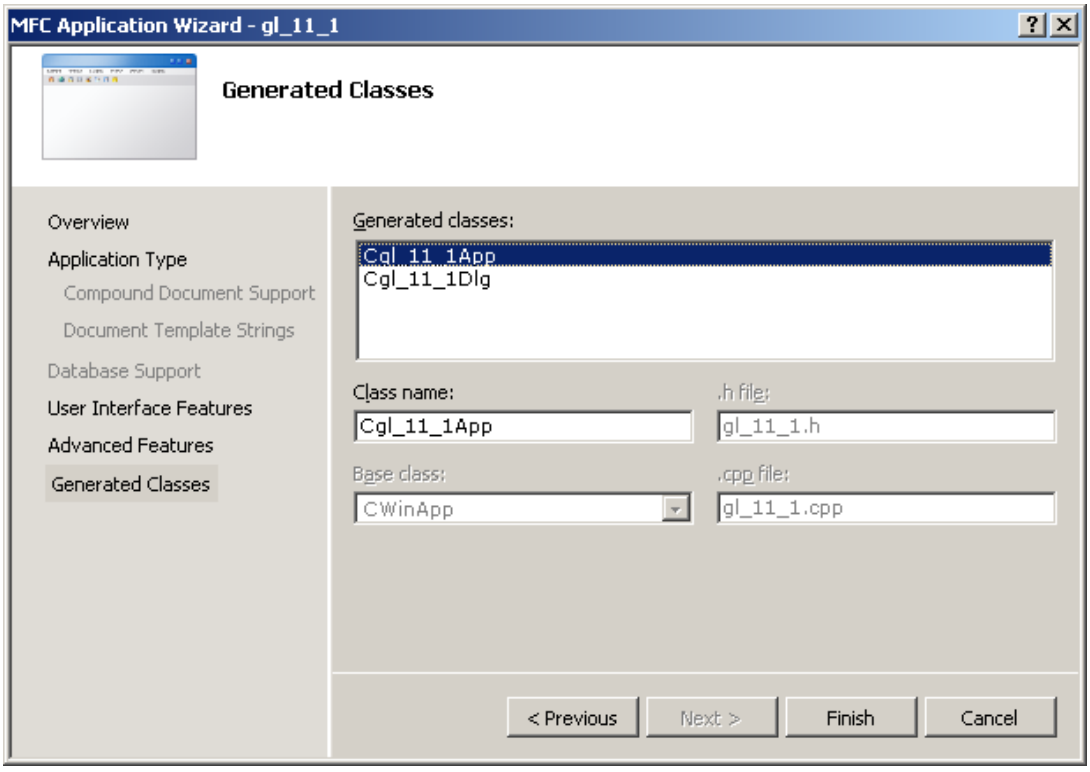

Рис. 11.5. Генерируемые классы

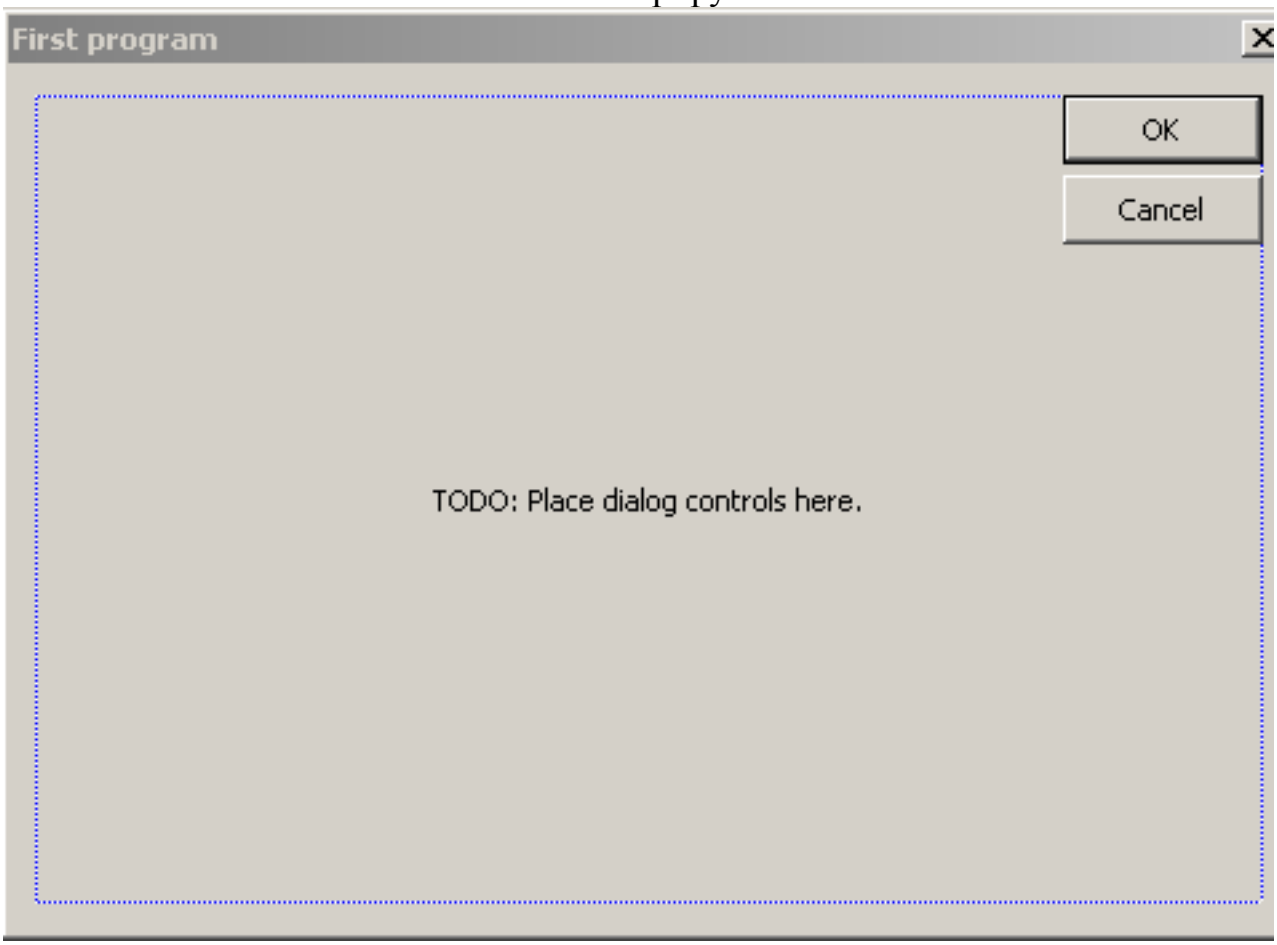

Рис. 11.6. Шаблон диалогового окна

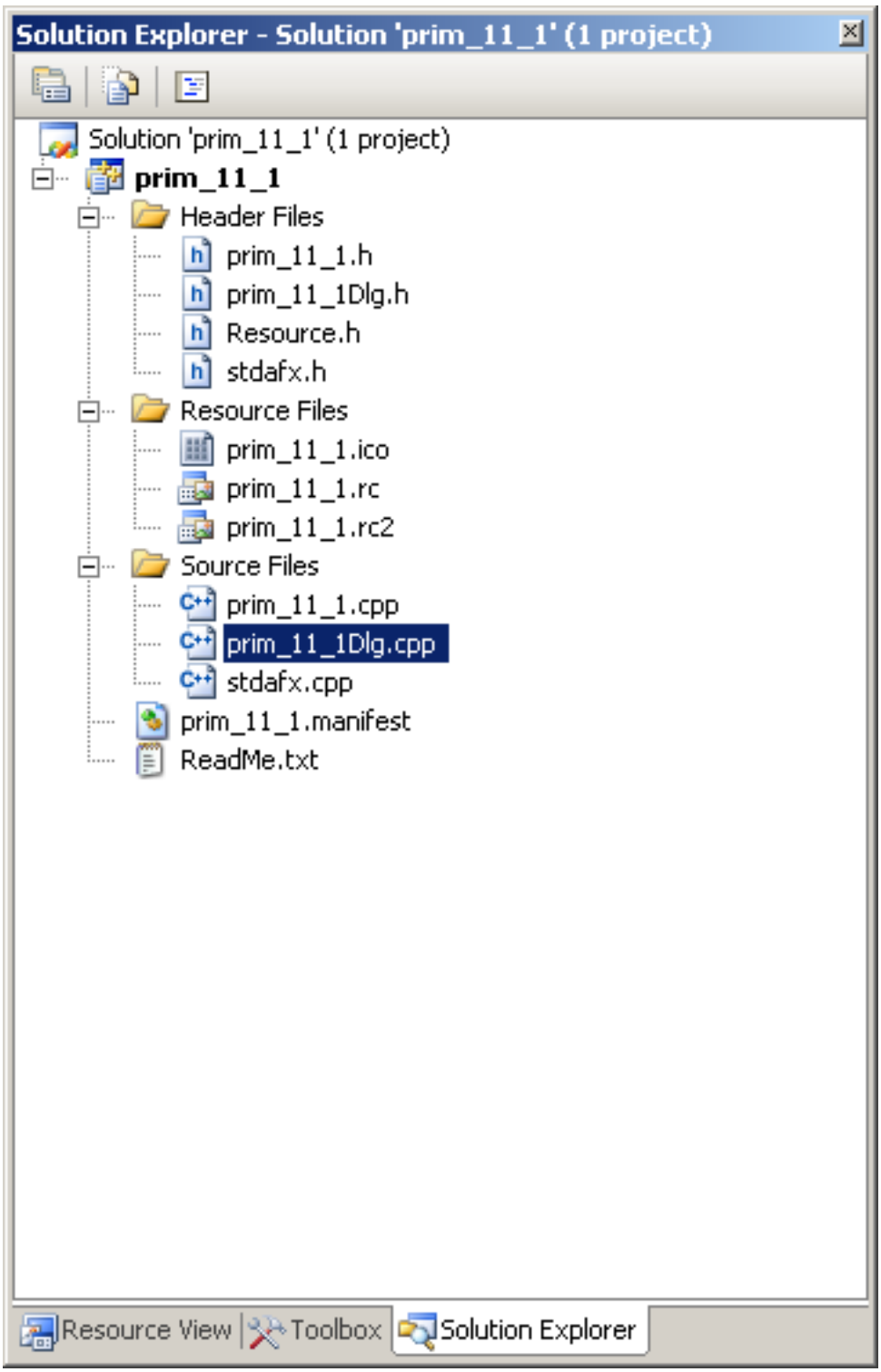

Рис. 11.7. Вкладка **Solution Explorer** окна **Solution Explorer**

При разработке приложений программист будет писать текст программы, и разрабатывать интерфейс окна приложения (помещать компоненты на диалоговое окно). Поэтому у него должна быть возможность переключаться между разрабатываемым диалоговым окно и соответствующим ему кодом программы. Для этого в окне среды MS Visual C++ (рис. 11.7) есть окно **Solution Explorer**. Вкладка **Solution Explorer** которого позволяет передвигаться между любыми файлами, входящими в проект, а вкладка **Resource View** (рис.

11.8) позволяет перейти к разрабатываемому диалоговому окну, для этого в папке **Dialog** вкладки **Resource View** необходимо выбрать имя разрабатываемого окна (имя окна состоит из символов «ID_» и имени проекта).

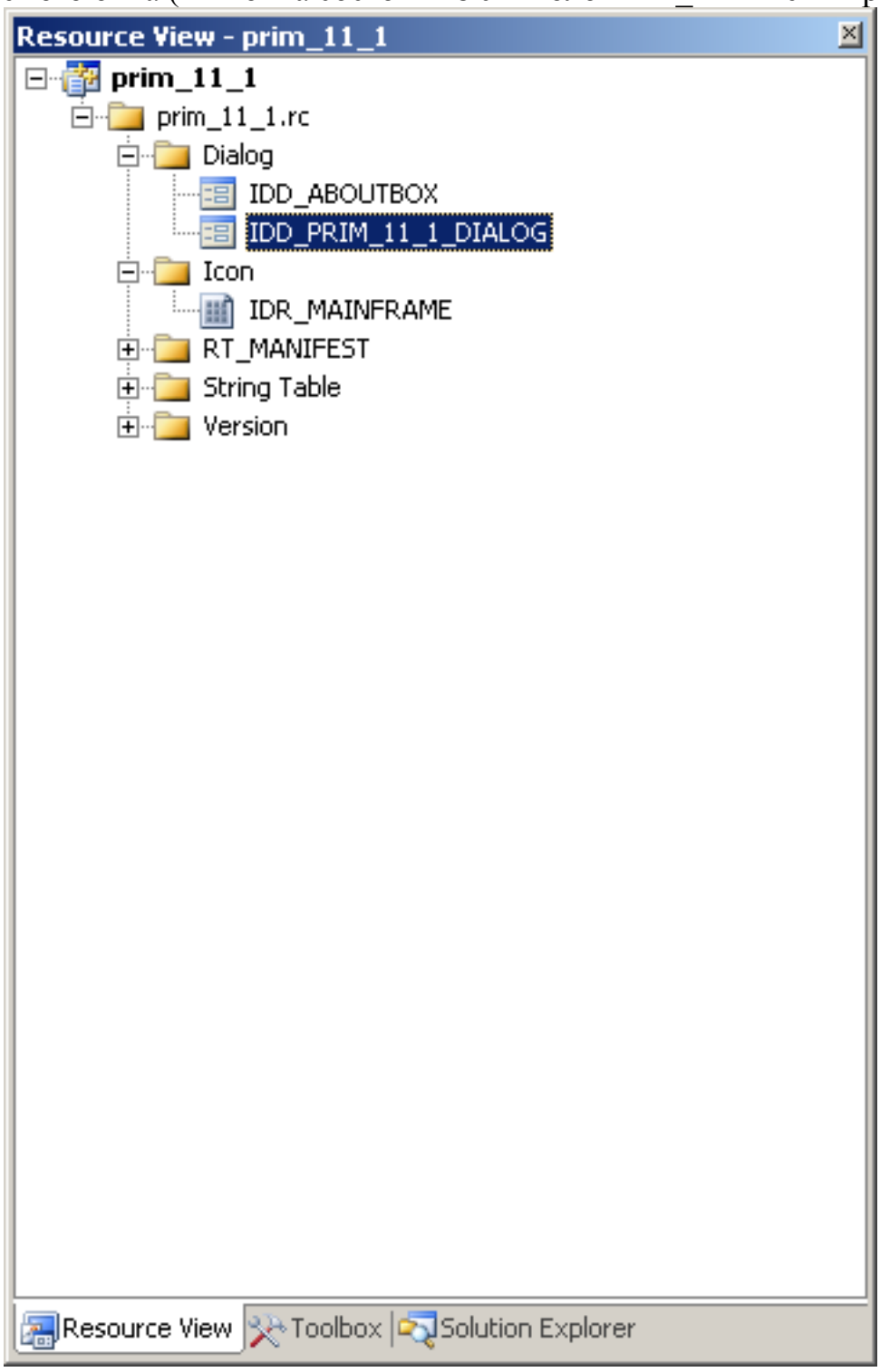

Рис. 11.8. Вкладка **Resource View** окна **Solution Explorer**

ПРИМЕР 11.1. Написать программу пересчета из градусов в радианы. Пользователь будет вводить значение в градусах, а программа будем возвращать значение в радианах. Окно приложения будет иметь вид подобный, представленному на рис. 11.9. Пользователь вводит значение в градусах,

щелкает по кнопке Пересчитать и видит окно, с пересчитанным значением в радианах.

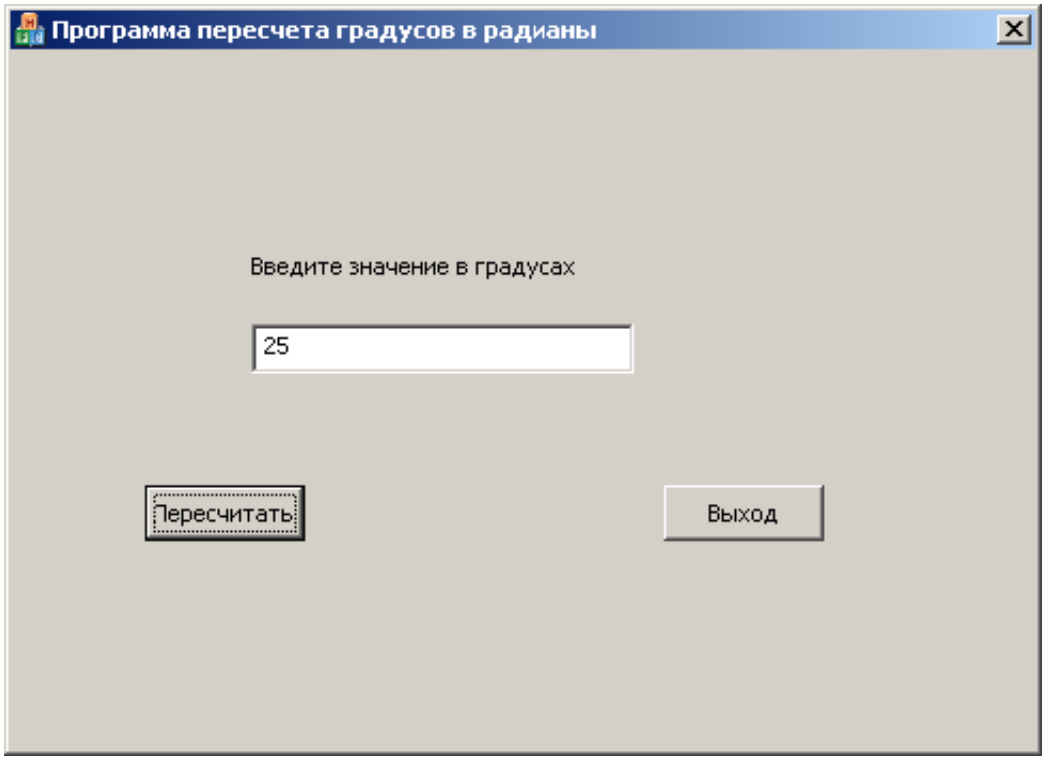

Рис. 11.9. Окно разрабатываемого приложения

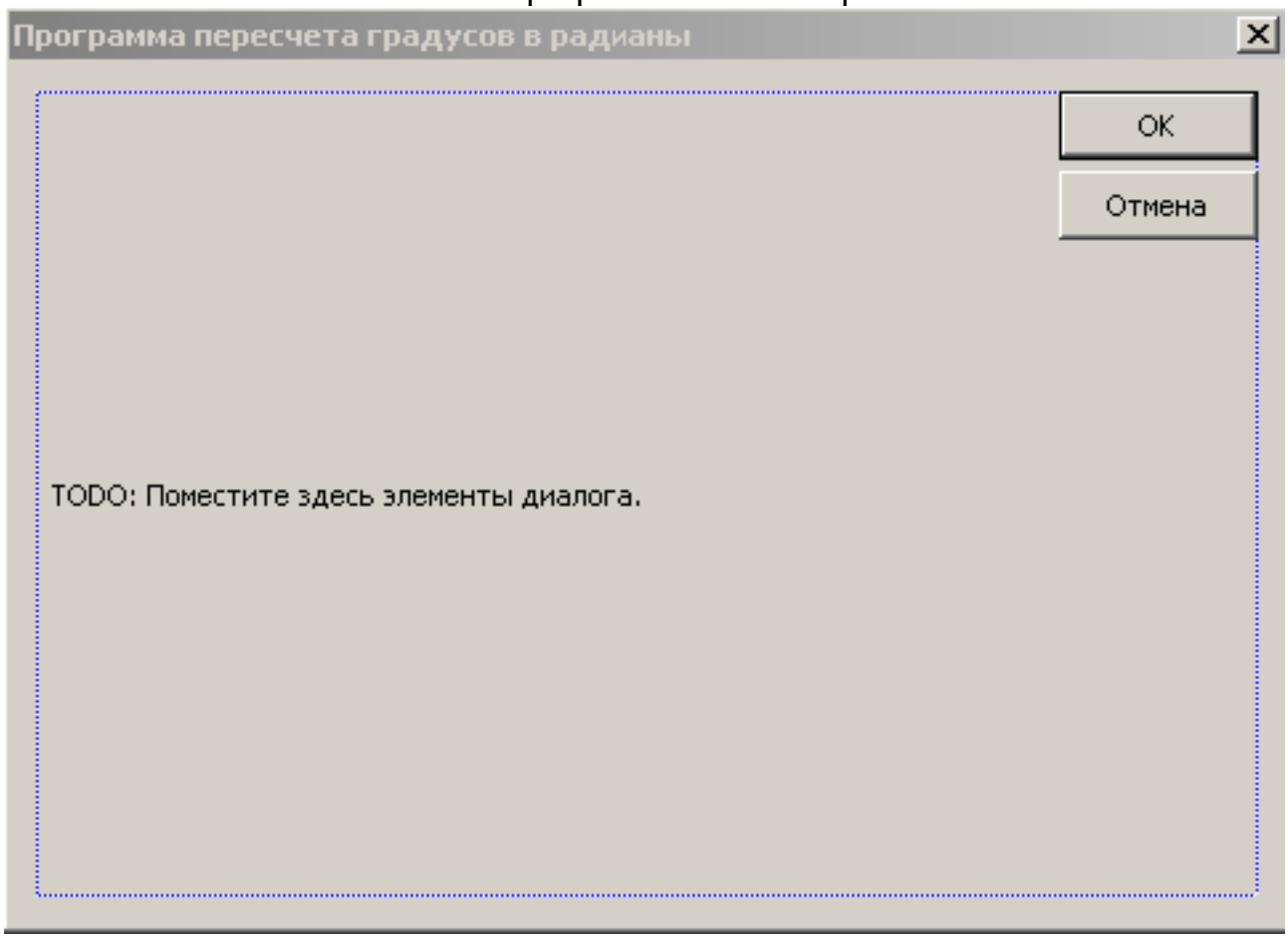

Рис. 11.10. Шаблон диалогового окна

Создадим шаблон приложения диалоговое окно с именем «**Программа пересчета из градусов в радианы**» (рис. 11.10). При создании не забудем указать русский язык в качестве языка разрабатываемого приложения.

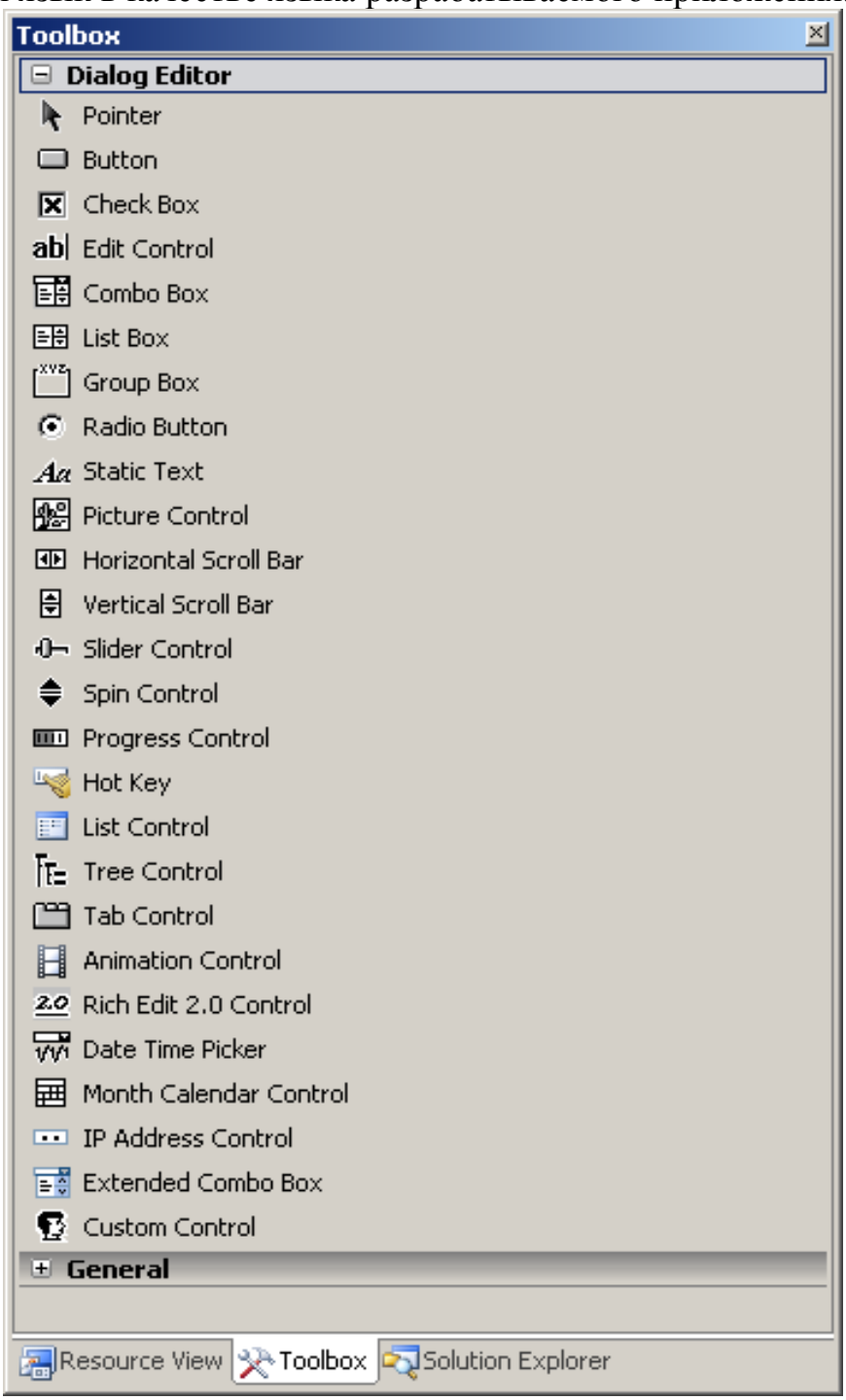

Рис. 11.11. Окно **ToolBox**

Выделим в диалоговом окне метку «TODO: ...», и нажав клавишу **Delete** удалим ее. Теперь надо добавить в окно разрабатываемого приложения метку и поле редактирования. Эти компоненты называются `Static Text` и `Edit Box` соответственно. Компоненты находятся в окне `ToolBox` (рис. 11). Окно `ToolBox` появляется при переходе к разрабатываемому диалоговому окну,

если оно не появилось, его явно можно включить с помощью команды **View** – **ToolBox** (комбинация клавиш **Ctrl+Alt+X**) или с помощью соответствующей кнопки на панели инструментов.

Перетащим компоненты `Static Text` и `Edit Control` из окна **ToolBox**. Расположим метку (`Static Text`), поле редактирования (`Edit Control`) и кнопки (`Button`) таким образом, чтобы диалоговое окно разрабатываемого приложения приняло вид, подобный, представленному на рис. 11.12.

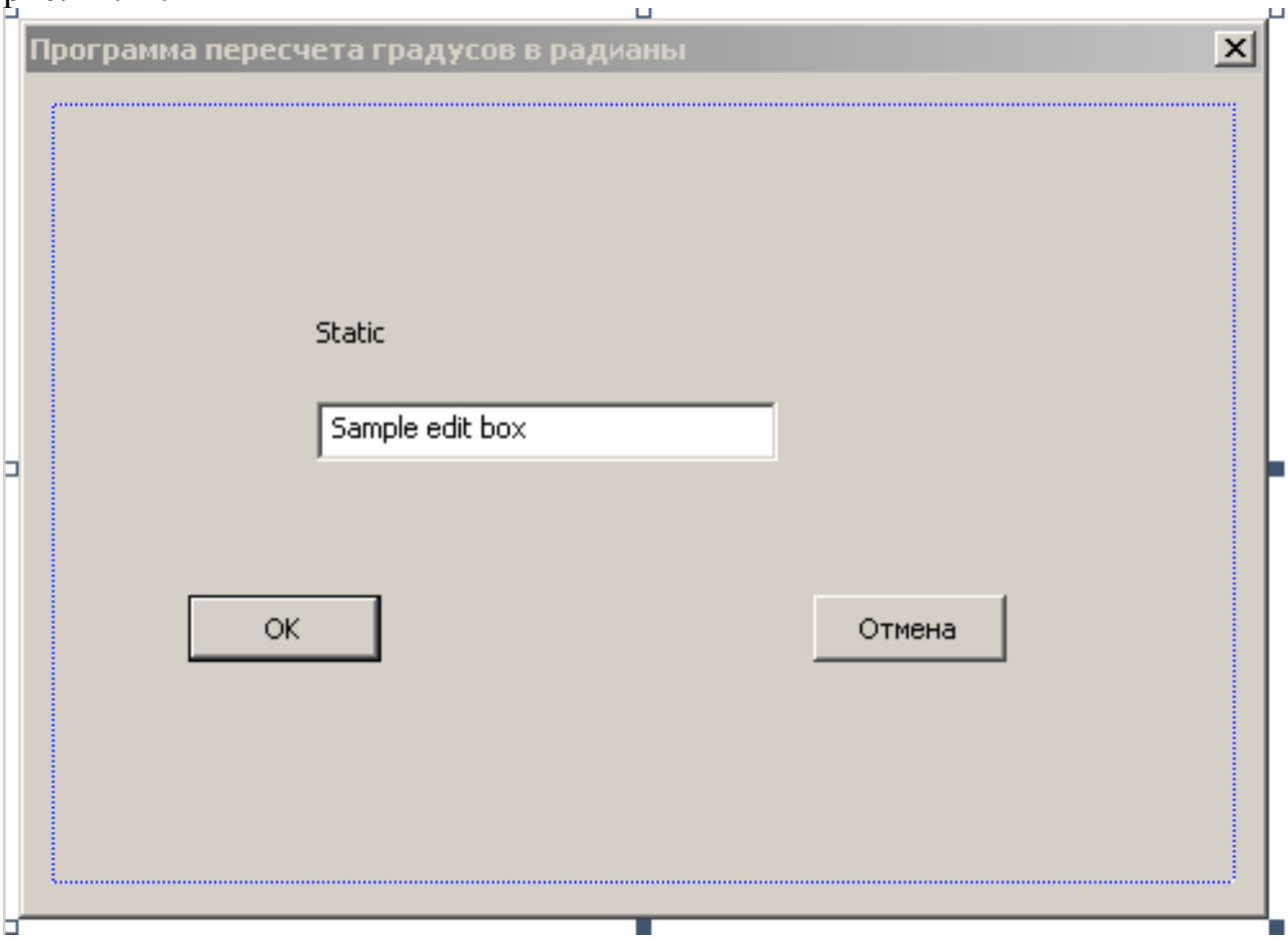

Рис. 11.12. Окно программы с компонентами `Static Text` и `Edit Control`

Проектирование внешнего вида формы завершено, теперь необходимо изменить текст метки, названия кнопок, и добиться того, чтобы разрабатываемое приложение заработало.

Для изменения текста метки, выделим ее, и в контекстном меню выберем команду **Properties**, и изменим значение свойства `Caption` на «`Введите значение в градусах`» (рис. 11.13). Свойство `Caption` определяет текст, отображаемый на метке.

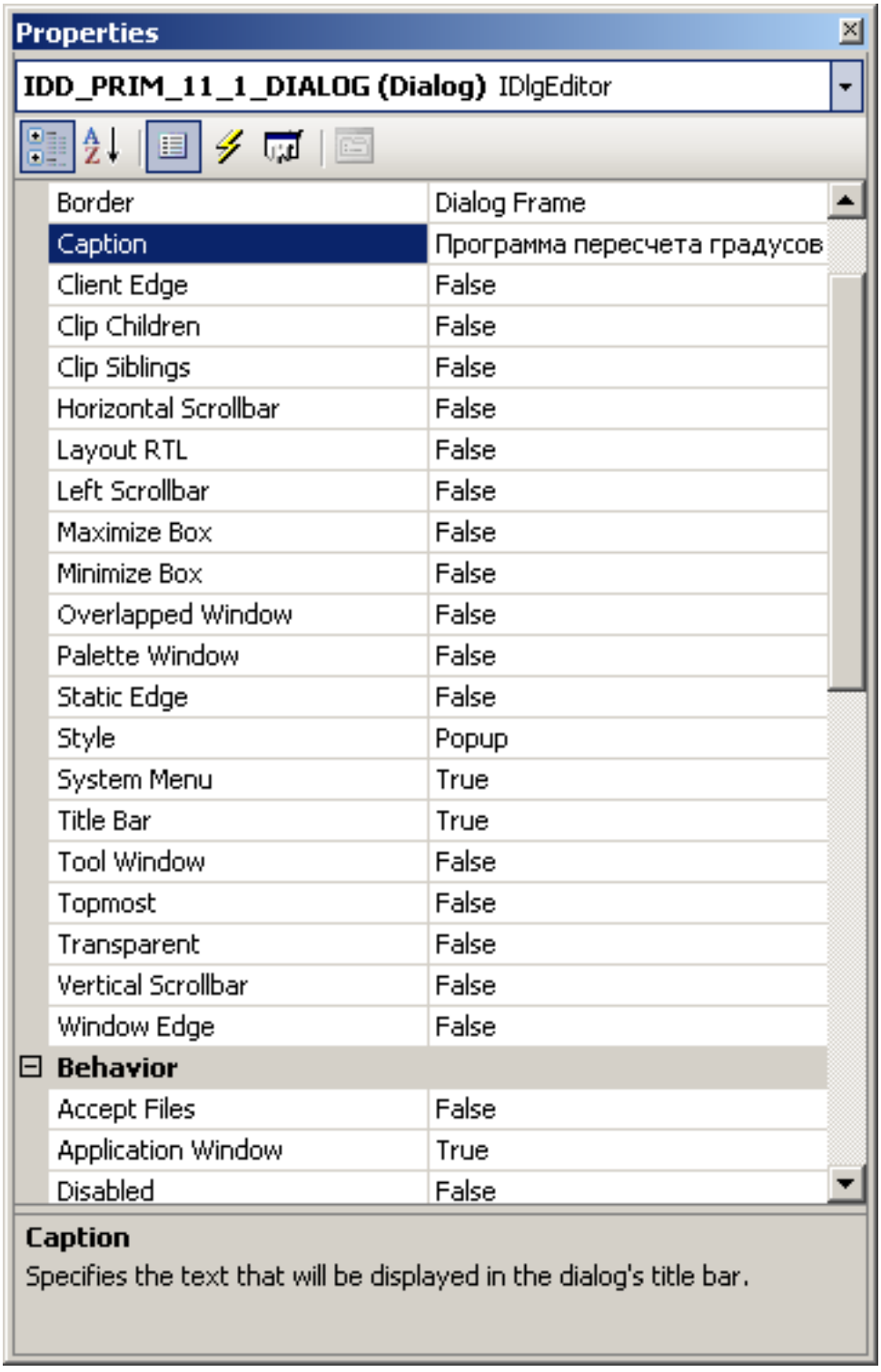

Рис. 11.13. Свойства метки

Изменим свойство `Caption` у кнопок **OK** и **Отмена**, для этого щелкнув по кнопке правой кнопкой мыши, выберем команду `Properties` и изменим значения свойства `Caption`. После всего этого диалоговое окно примет вид (рис. 11.14).

Теперь необходимо написать функцию (метод класса «Диалоговое окно»), которая будет определять действия программы при щелчке пользователя по кнопке **Пересчитать**. Функция, при щелчке по этой кнопке должна считать

текст из поля редактирования, преобразовывать его число, умножить на и вывести полученный результат.

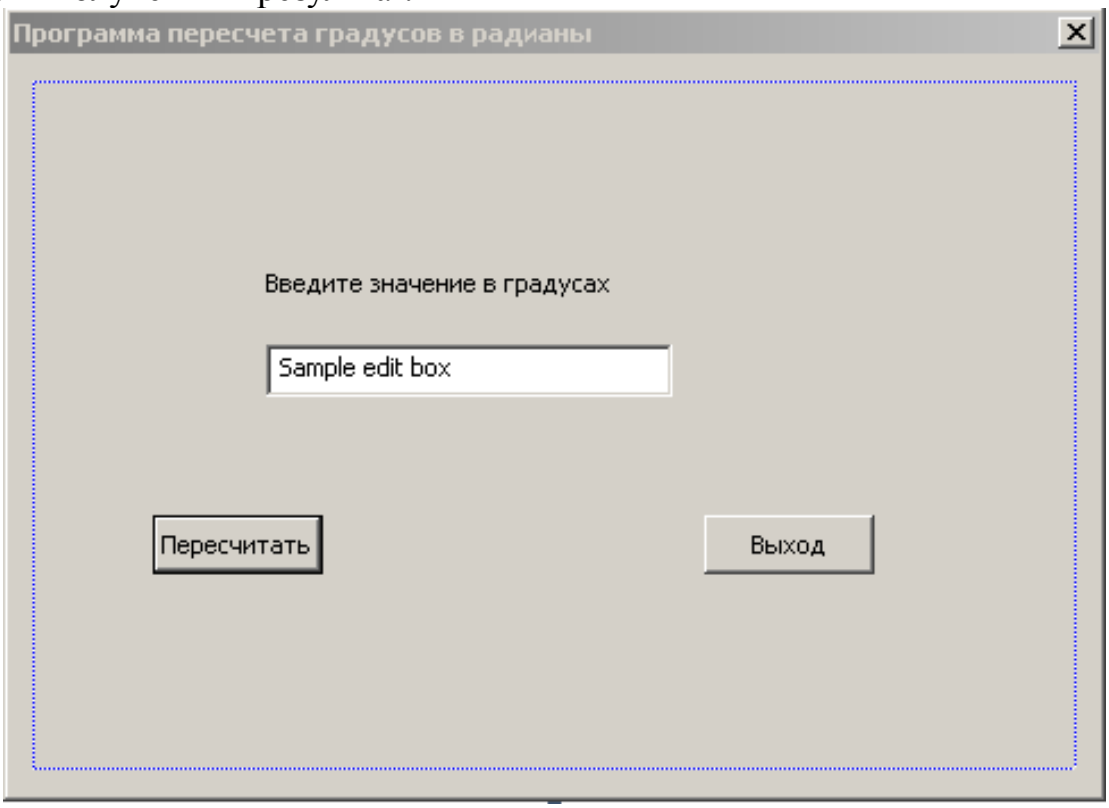

Рис. 11.14. Окончательный вид диалогового окна разрабатываемого приложения

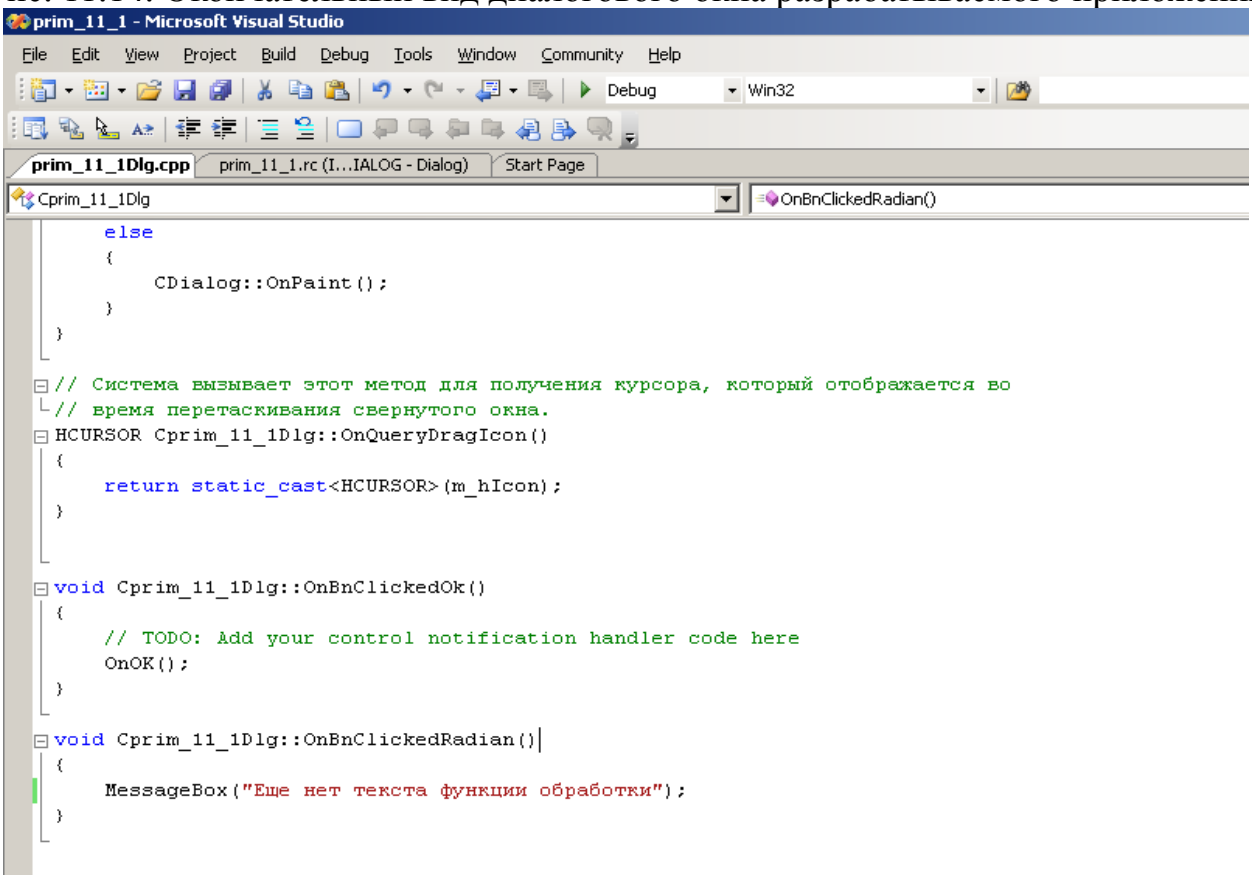

Рис. 11.15. Окно редактора с введенным текстом функции

Для переопределения действий, выполняемых по щелчку по кнопке **Пересчитать** необходимо:

1. В свойствах кнопки (контекстное меню, команда **Properties**) установить новое значение параметра `ID`, пусть это будет значение **ID_radian**. Стандартное значение параметра `ID` определяет стандартный метод (закрытие диалогового окна), который будет вызываться при щелчке мышью. Переопределив стандартное значение параметра `ID`, мы тем самым говорим, что будем определять новый метод, обрабатывающий щелчок мышкой по кнопке.
2. После этого щелкаем дважды по кнопке **Пересчитать** и попадаем в окно редактора (рис. 11.15), где в качестве тела функции `OnBnClickedRadian` вводим следующую строку кода: `MessageBox("Еще нет текста функции обработки");`

Запускаем приложение. При щелчке по кнопке **Пересчитать**, появляется следующее диалоговое окно (рис. 11.16).

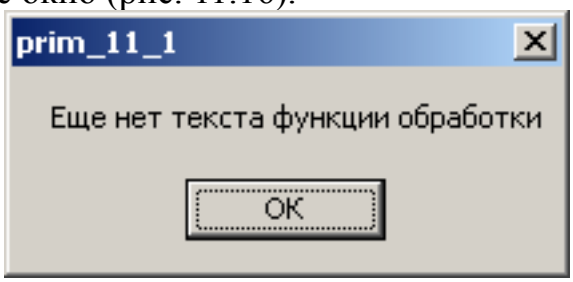

Рис. 11.16. Окно с сообщением

Таким образом, мы добились того, что кнопка стала работать по другому, осталось в функции `OnBnClickedRadian` написать нужный текст. Для этого понадобиться переменная, которая будет отвечать за кнопку, т.е. нужна будет переменная класса `Edit Control` (`Поле редактирование`). Вообще говоря, нужны будут переменные, отвечающие за все компоненты, размещенные на форме, которые будут участвовать в программе.

3. Для добавления переменной выделяем элемент `Edit Control`, вызываем контекстное меню, выбираем команду **Add Variable**, после чего появляется окно, подобное изображенному на рис. 11.17.

Вводим имя переменной (`m_Edit`), отвечающей за работу с компонентом `Edit Control` в поле `Variable Name`. В свойствах переменной `m_Edit`, необходимо установить тип возвращаемого компонентом результата, необходимо указать, что компонент будет возвращать значение (**Category – Value**), и конкретный тип возвращаемого значения, в нашем случае это вещественное число (**Variable type - float**). Окно **Add Member Variable** должно стать таким, как представлено на рис. 11.18.

4. После этого осталось изменить код функции `OnBnClickedRadian` на следующий

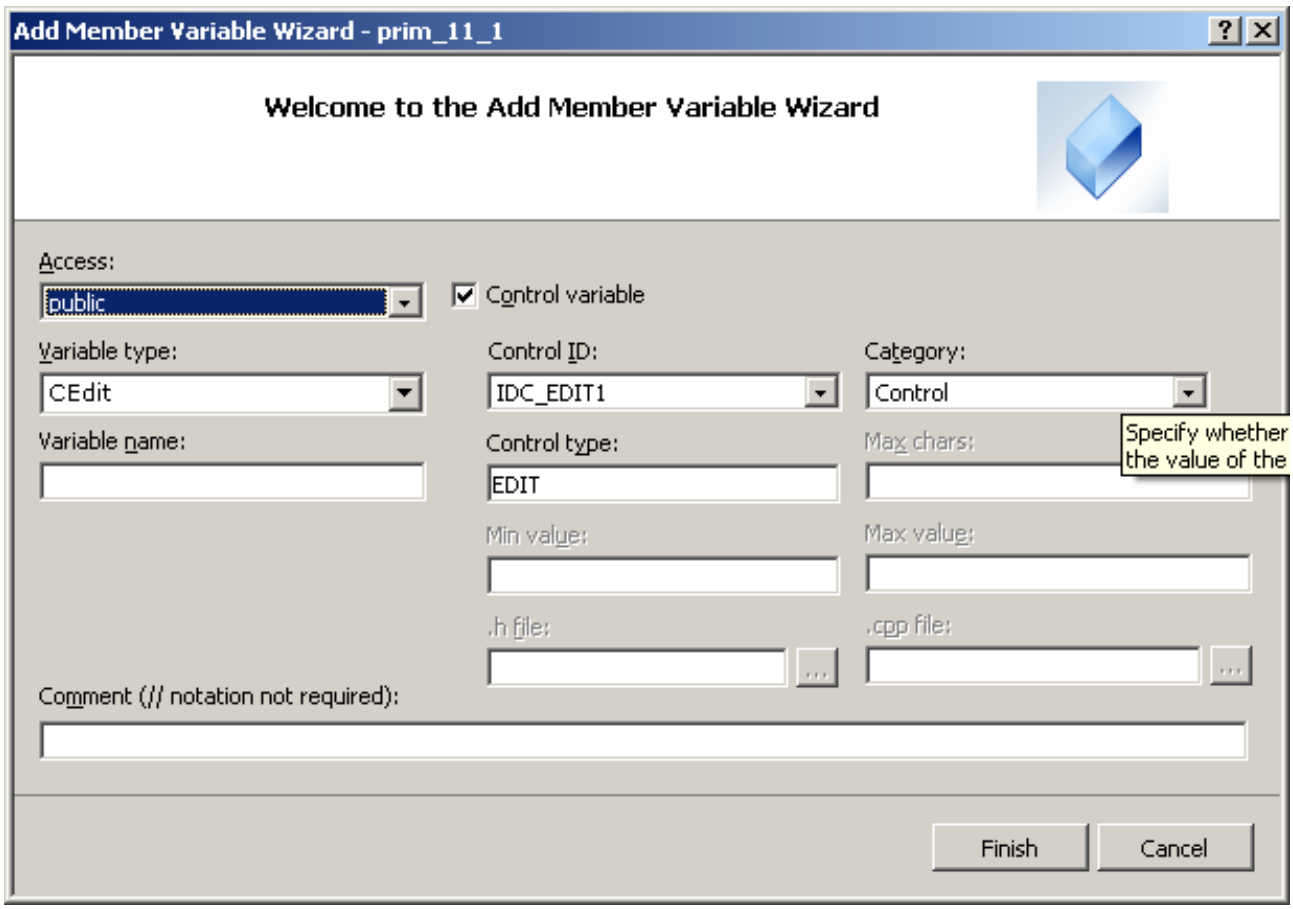

Рис. 11.17. Окно **Add Member Variable**

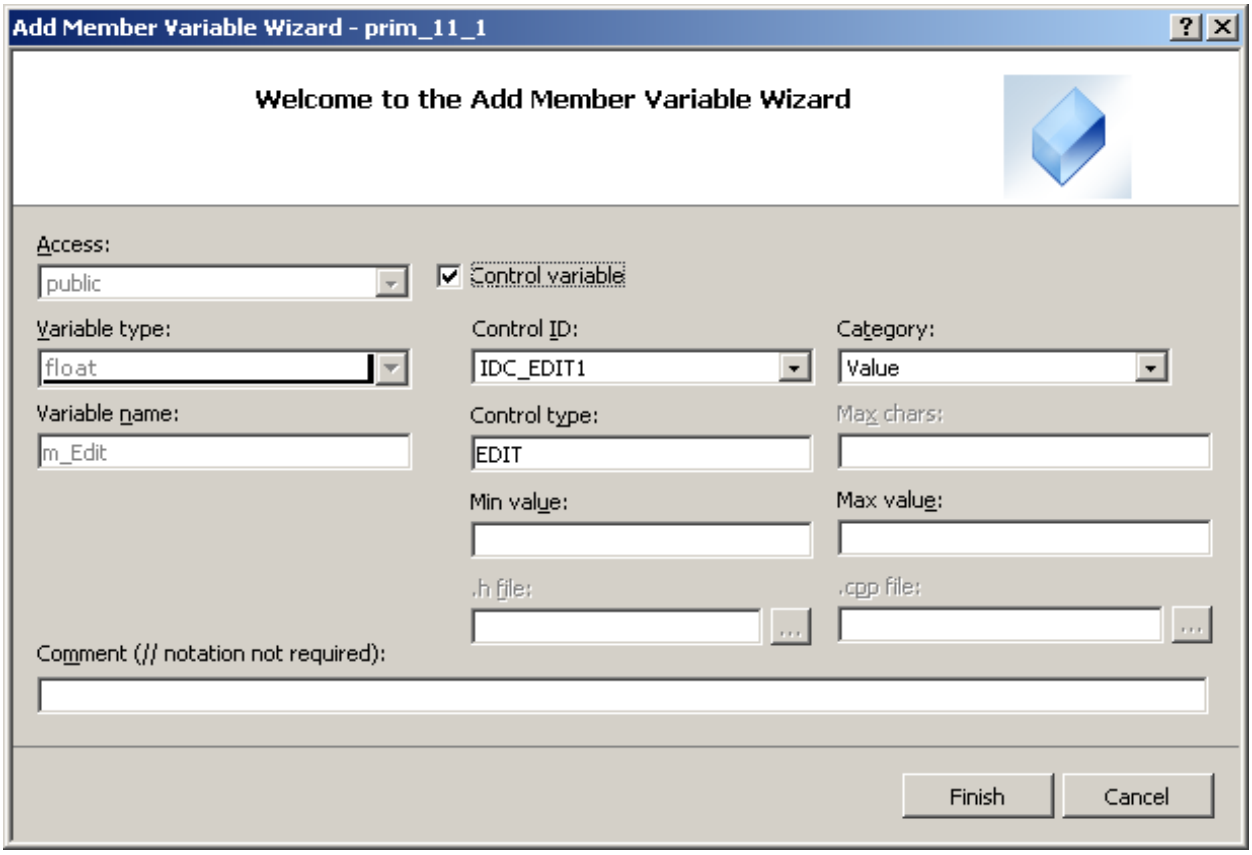

Рис. 11.18. Окно **Add Member Variable** после ввода значений

```
void CVisual_gr_radDlg::Onradian()
{
float a, rad;
char s[80];
//Оператор UpdateData позволяет считать из компонента
Edit Control
//значение введенное пользователем, если пропустить
//этот оператор, то программа не будет видеть
значение, //которое в компонент Edit Control ввел
пользователь.
UpdateData(TRUE);
//Считывание в переменную a содержимого поля ввода,
//a имеет тип float, и в окне Add Member Variable мы
//определяли, что m_Edit возвращает значение типа float.
a=m_Edit;
rad=a*3.14159/180;
sprintf(s,"a=%g , значение в радианах %g",a,rad);
//Функция MessageBox в простейшем случае формирует
//диалоговое окно со строкой, хранящейся в переменной s.
MessageBox(s);
}
```

Теперь приложение работает, после ввода значения угла в градусах и щелчка по кнопке **Пересчитать**, на экране возникнет диалоговое окно с результатом (рис. 11.19).

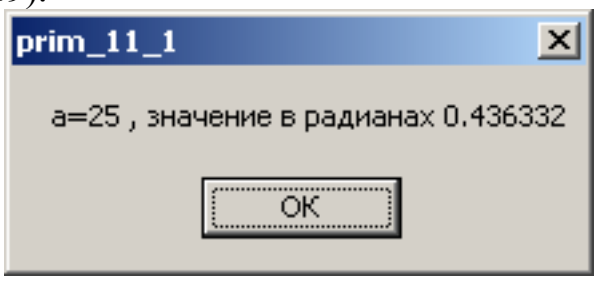

Рис. 11.19. Окно с результатами работы программы

Таким образом, можно выделить основные этапы создания простейшего визуального приложения:

1. Создаем шаблон приложения типа «Диалоговое окно».
2. Размещаем в этом окне необходимые компоненты, при необходимости меняем их свойства.
3. Изменяем значения `ID` тех кнопок, которые будут управлять работой приложения. Можно вместо изменения `ID`
4. Определяем имена и возвращаемые значения компонентов `Edit Control ()`.
5. Программируем текст функций, отвечающих за обработку событий при щелчке мышью по кнопкам (такие функции в программировании носят название *обработчики событий*, в данном случае событием является

щелчок мыши по кнопке).

ПРИМЕР 11.2. Написать программу решения квадратного или биквадратного уравнения. Расположим на диалоговом окне основные элементы (рис. 11.20).

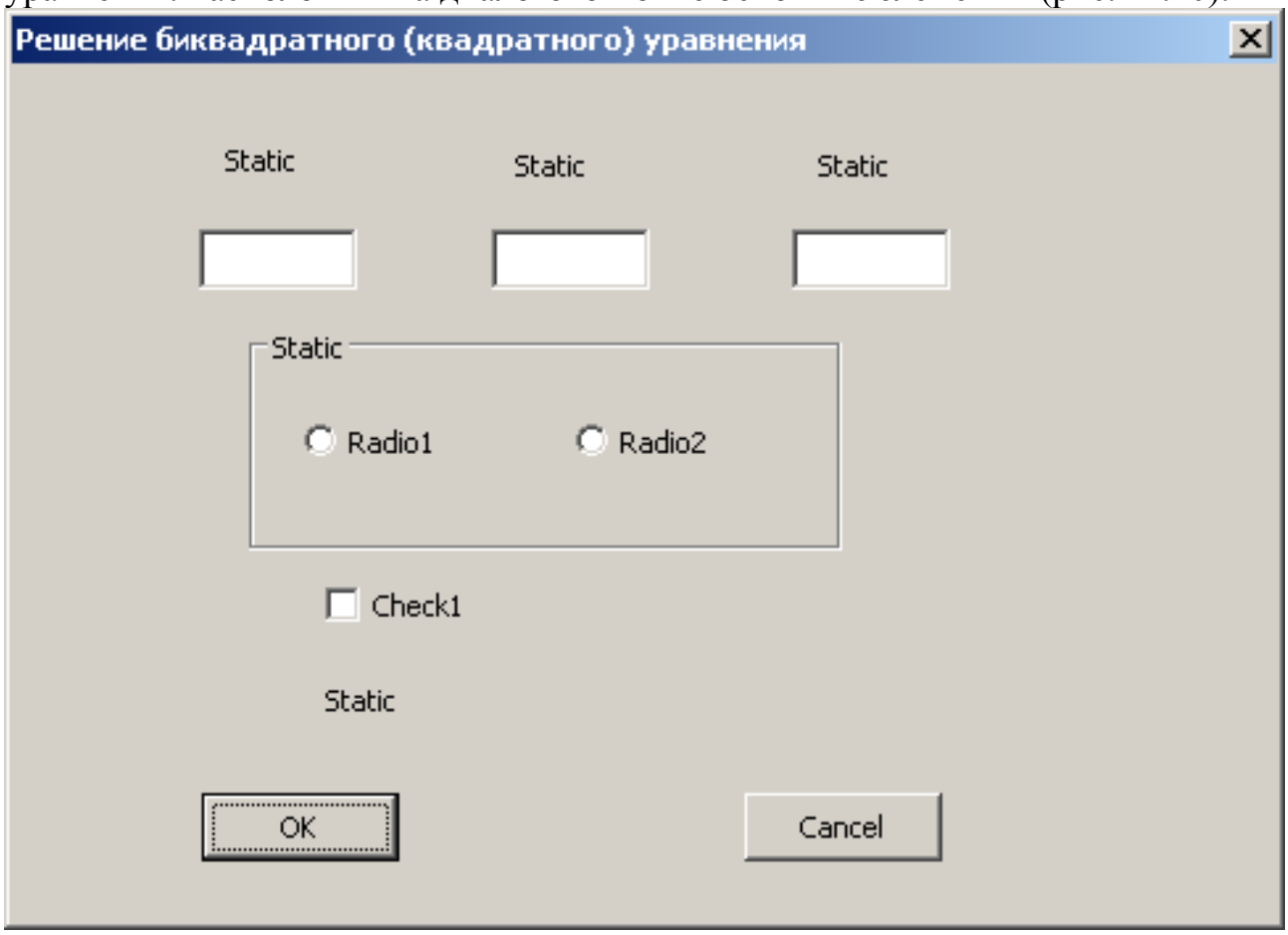

Рис. 11.20. Шаблон программы с основными элементами

Первых три метки (класс `Static Text`) служат для вывода заголовков полей ввода, три поля редактирования (класс `Edit Control`), группа из двух радиокнопок (класс `Radio Button`) для выбора типа уравнения (квадратное или биквадратное), флажок (класс `Check Box`), определяющий, как будет происходить вывод (на четвертую метку или в файл), метка для вывода результатов (класс `Static Text`), кнопки для решения уравнений и завершения программы.

Изменим свойства `Caption` у кнопок, флажков и меток, после этого диалоговое окно станет таким, как изображено на рис. 11.21.

Далее надо определить переменные для полей редактирования, радиокнопки, флажка. Выделяем первое поле ввода, контекстном меню выбираем команду **Add Variable…** и определяем (рис. 11.22):

- `Variable name` (имя переменной) `m_editA`;
- `Category` (категория) – `Value` (значение);
- `Variable type` (тип переменной) – `float`.

Аналогичным образом определяем переменные для остальных полей редактирования (рис. 11.23, 11.24). Переменные `m_editA`, `m_editB`, `m_editC`

будут отвечать за ввод переменных `a`, `b` и `c` из соответствующих полей редактирования.

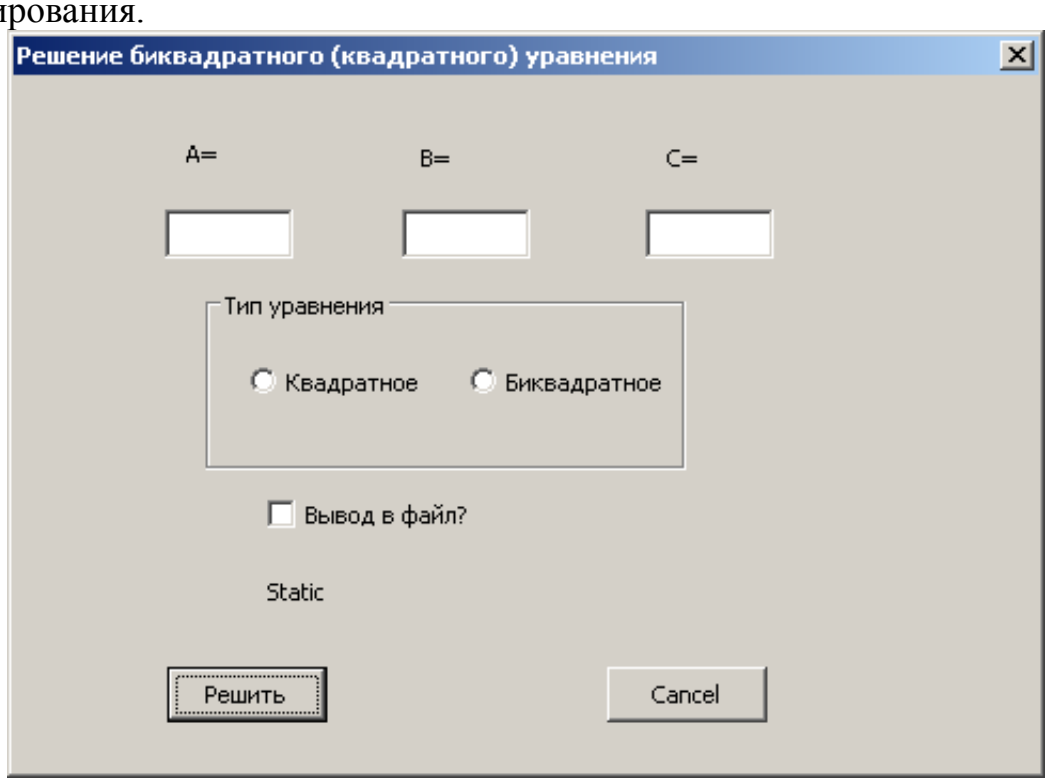

Рис. 11.21. Шаблон программы с измененными основными элементами

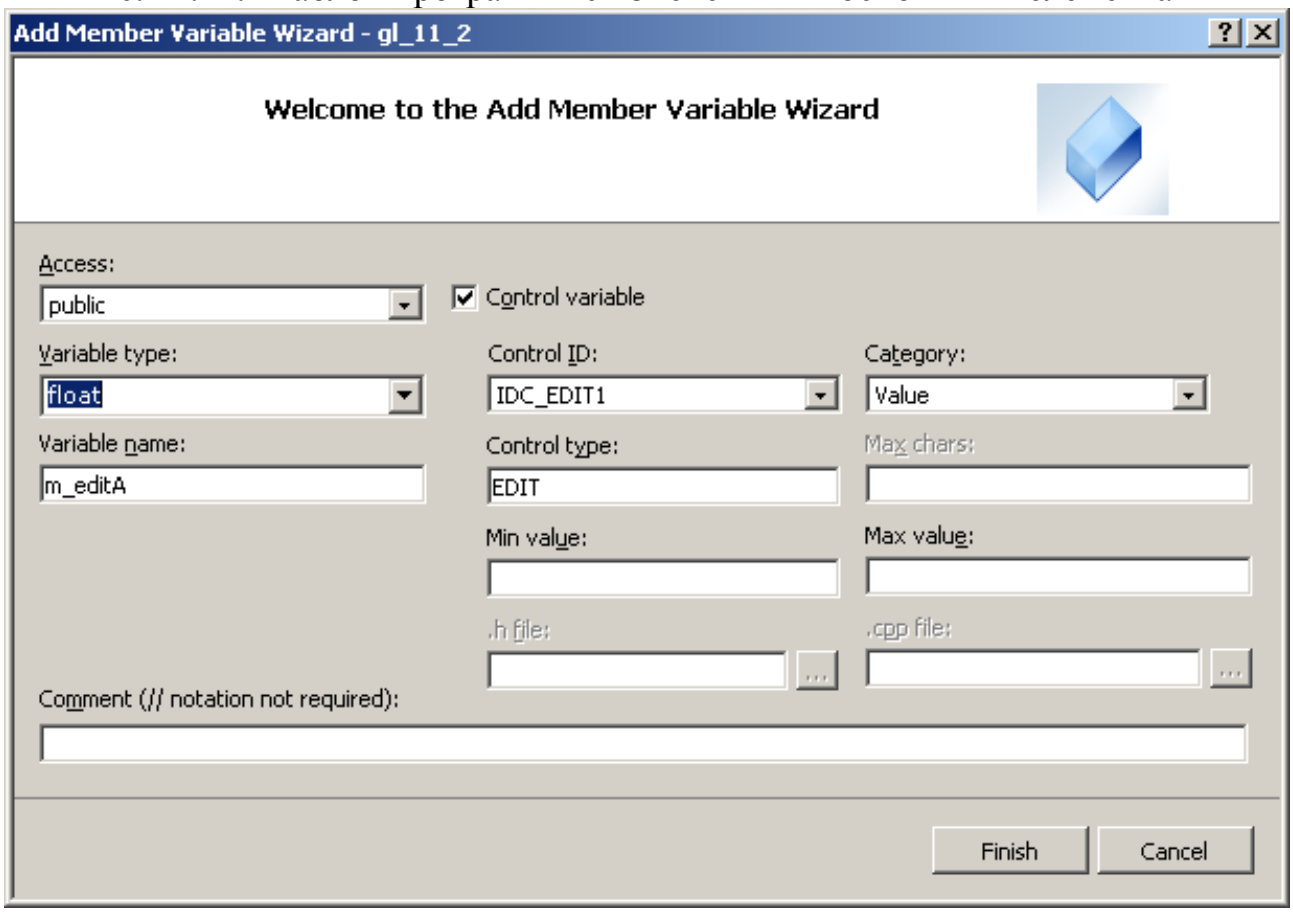

Рис. 11.22. Определение переменной для первого поля редактирования

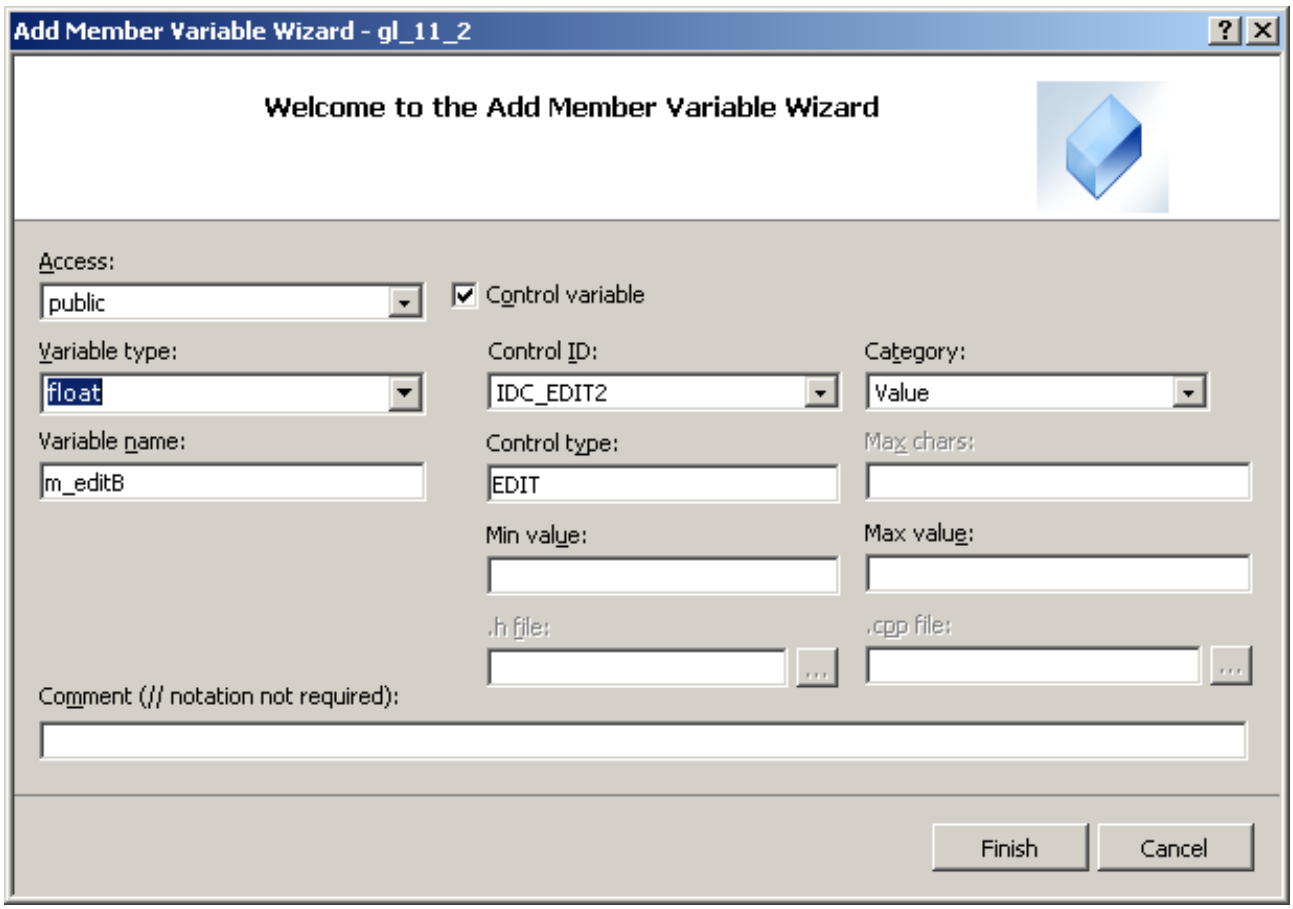

Рис. 11.23. Определение переменной для первого поля редактирования

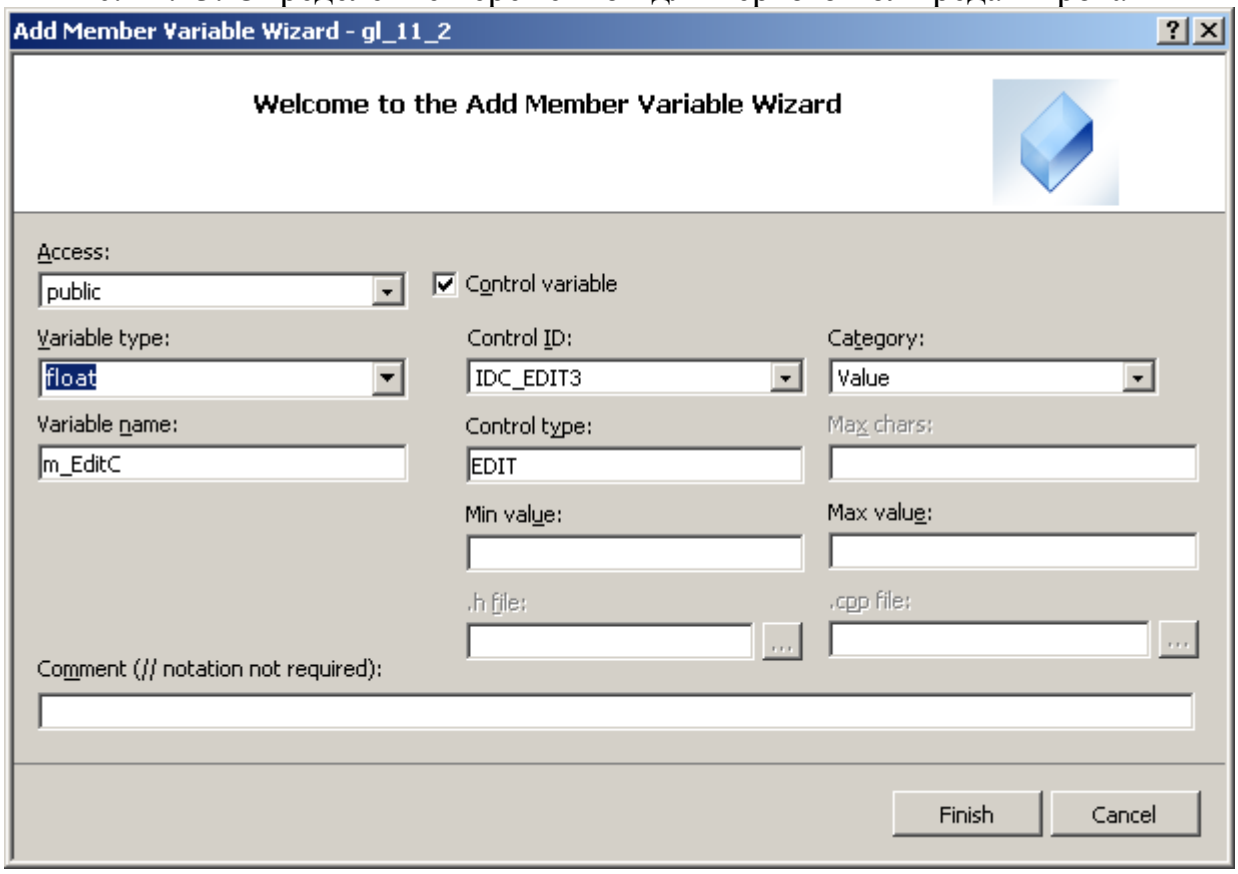

Рис. 11.24. Определение переменной для второго поля редактирования

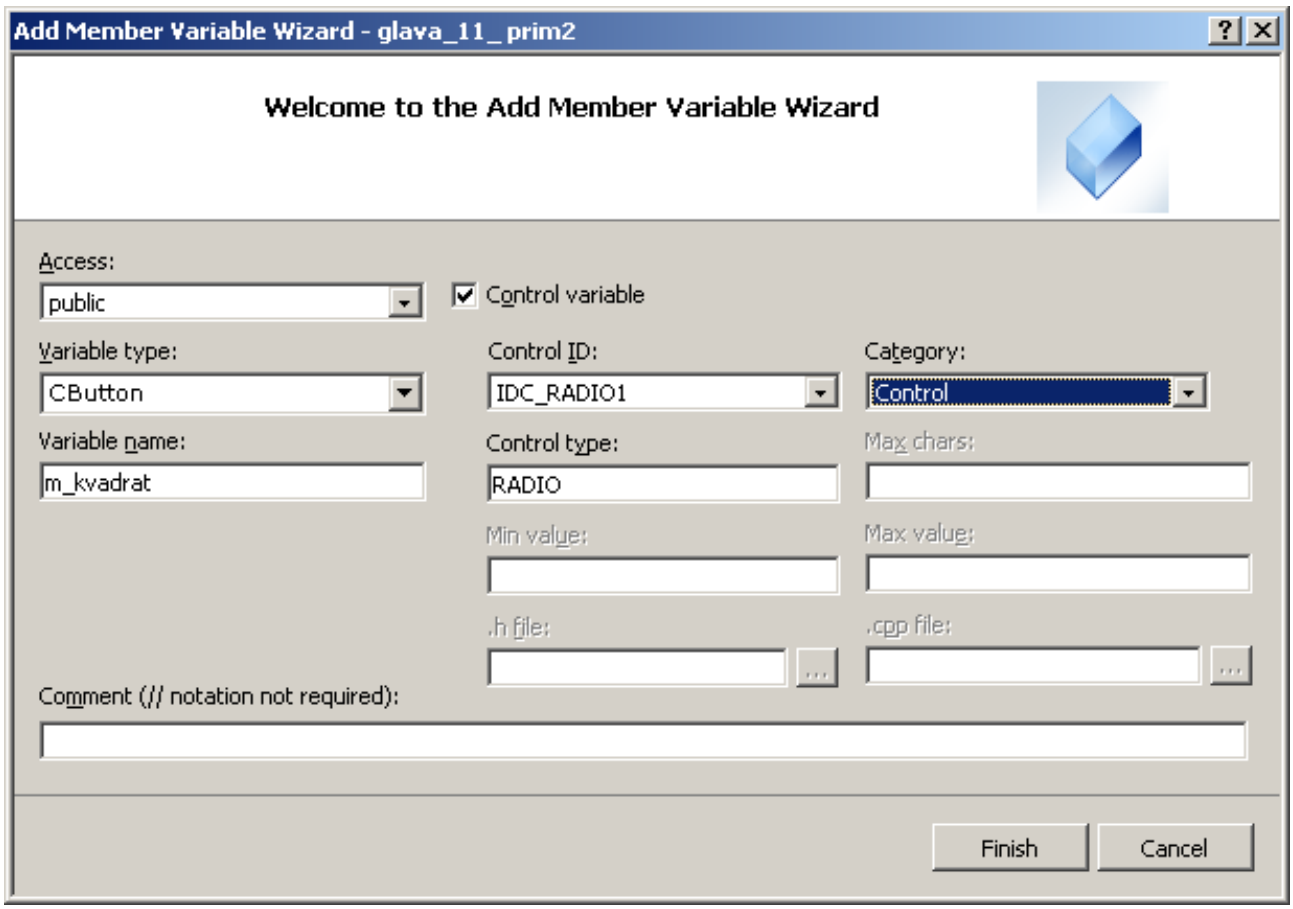

Рис. 11.25. Определение переменной для первой радиокнопки

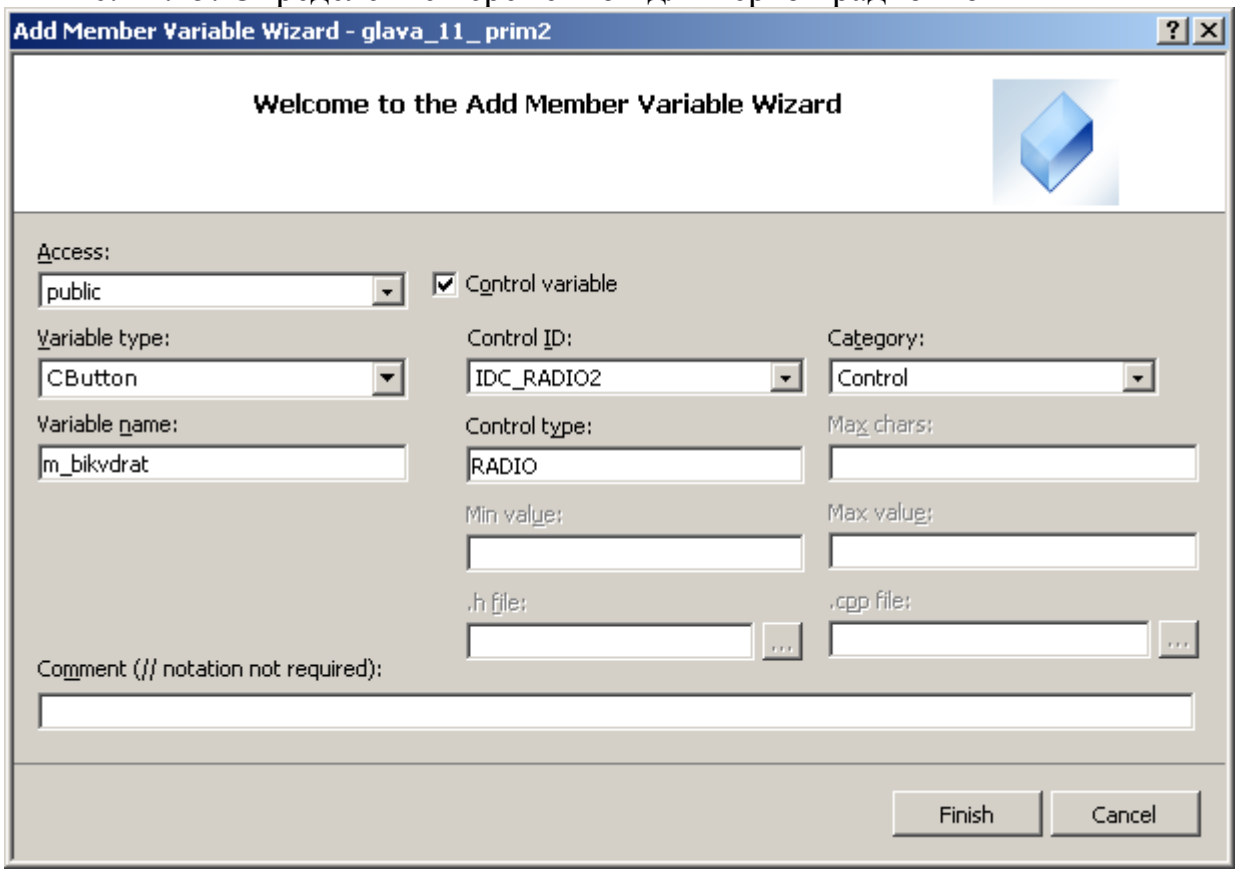

Рис. 11.26. Определение переменной для второй радиокнопки

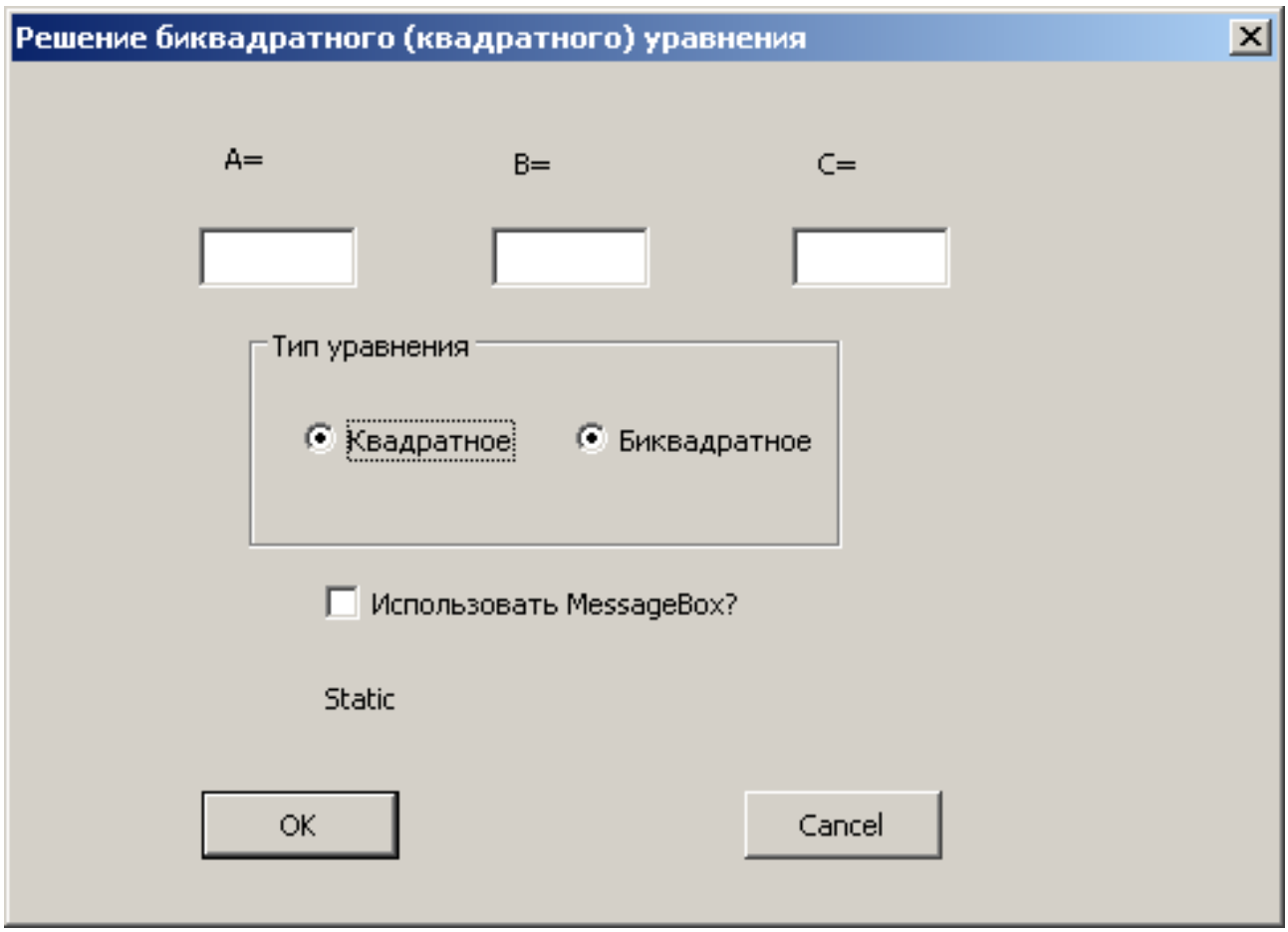

Рис. 11.27. Неправильное функционирование радиокнопок

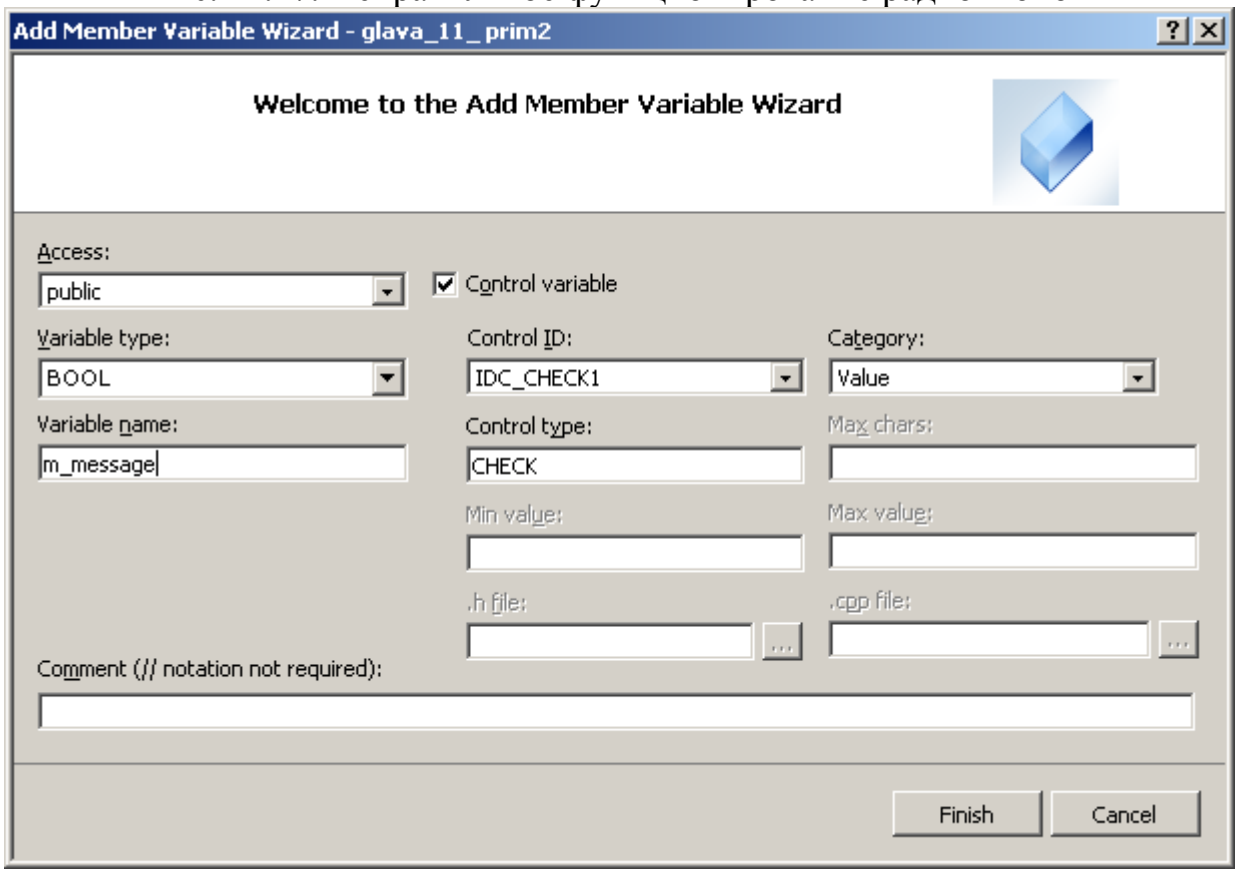

Рис. 11.28. Определение переменной для флажка

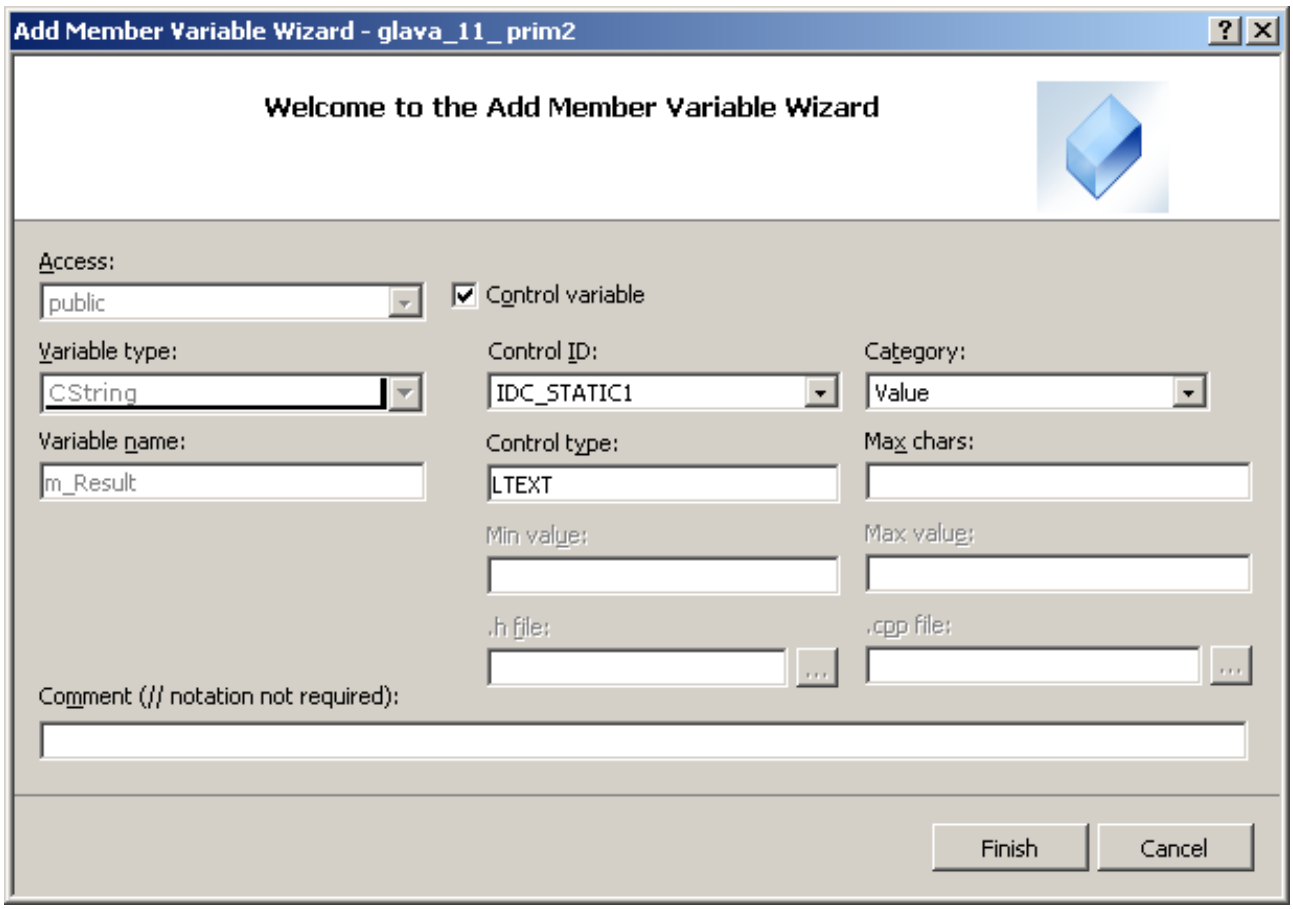

Рис. 11.29. Определение переменной для четвертой метки

Следующим этапом будет определение переменных, отвечающих за радиокнопки. Выделяем элемент `Group Box` и устанавливаем (контекстное меню, **Properties**) значение свойства `Group – True`, затем у обоих радиокнопок также устанавливаем значение свойства `Group – True`. Далее необходимо определить имя и тип переменной для обоих радиокнопок (рис. 11.25, 11.26).

Значением переменной `m_kvadrat` является первая радиокнопка – экземпляр класса `RadioButton`, значением переменной `m_bikvadrat` является вторая радиокнопка – также экземпляр класса `RadioButton`. Установка свойства `Group` в `True` было необходимо для того, чтобы можно было создать переменные `m_kvadrat`, `m_bikvadrat` как экземпляры класса `RadioButton`. Однако, установив это свойства в `True` у обоих кнопок мы тем самым создали не две связанные кнопки, а две группы независимых кнопок (в нашем случае группы из одной кнопки). Переключение между кнопками происходить не будет, если запустить приложение сейчас, то вместо переключения будет происходить выделение последовательно обеих радиокнопок (рис. 11.27).

Только у одной из кнопок может быть установлено значения свойства `Group` в `True`. Поэтому установим у второй радиокнопки свойство `Group` в `False`.

Теперь определяем переменную, которая будет отвечать за флажок, ее тип BOOL (рис. 11.28).

Изменим значения ID четвертой метки (контекстное меню, **Properties**) на IDC_STATIC1. После это определим переменную (контекстное меню, Add Variable), отвечающую за работу с этой меткой (см. рис 11.29). Если не переопределять стандартное значение ID, то невозможно определить переменную, отвечающую за работу с меткой.

Осталось переопределить ID кнопки **Решить**, для этого выделяем ее, в контекстном меню выбираем команду **Properties**, в качестве ID вводим ID_Solve.

Все готово к началу модификации кода программы. Для этого щелкаем по кнопке **Решить** и вводим следующий текст функции OnBnClickedSolve(). Не забудьте подключить все необходимые библиотеки (в этом случае – math.h)

```
void Cglava_11_prim2Dlg::OnBnClickedSolve()
{
char S[80];
FILE *f;
//CRect rect;
double a,b,c,d,x1,x2,y1,y2, y3, y4;
    // Обновление полей перед чтением
UpdateData(TRUE);
    //Считываем значения a, b и c
    //из полей редактирования.
a=m_editA;
b=m_editB;
c=m_editC;
    //Вычисляем дискриминант.
d=b*b-4*a*c;
    //Функция (метод) GetCheck() проверяет выбрана
    //ли радиокнопка, в данном случае, выбрана
    //радиокнопка m_kvadrat, т.е. выбран ли
    //тип уравнения квадратное?
if (m_kvadrat.GetCheck())
    //Решение квадратного уравнения.
if (d<0)
sprintf(S,"Дискриминант отрицателен, корней нет!!!");
else
{
    x1=(-b+sqrt(d))/2/a;
    x2=(-b-sqrt(d))/(2*a);
    //Из корней формируем строку s.
    sprintf(S,"x1=%g\tx2=%g",x1,x2);
}
```

```
else
{
    if (d<0)
    //Формируем строку вывода s.
    sprintf(S,"Дискриминант отрицателен, корней нет!!!");
    else
    {
    //Если выбрана радиокнопка m_bikvadrat,
    //т.е. выбран ли тип уравнения биквадратное?
    //Решаем биквадратное уравнение.
    x1=(-b+sqrt(d))/2/a;
    x2=(-b-sqrt(d))/(2*a);
    if ((x1<0)&&(x2<0))
    //Формируем строку вывода s.
        sprintf(S,"Корней нет!!!");
    else
        if ((x1>=0)&&(x2>=0))
        {
        y1=sqrt(x1);
        y2=-y1;
        y3=sqrt(x2);
        y4=-y3;
    //Из корней формируем строку s.
    sprintf(S,"Четыре корня:y1=%f\ty2=%f\ty3=%f\t y4=%f",
                                          y1,y2,y3,y4);
        }
        else
        if (x1>=0)
        {
            y1=sqrt(x1);
            y2=-y1;
    //Из корней формируем строку s.
            sprintf(S,"Два корня: y1=%f\ty2=%f",y1,y2);
        }
        else
        {
            y1=sqrt(x2);
            y2=-y1;
    //Из корней формируем строку s.
            sprintf(S,"Два корня: y1=%f\ty2=%f",y1,y2);
        }
    }
}
    //В зависимости от флажка message
```

```
if (m_message)
{
     //выводим результаты в файл или
     f=fopen("Result.txt","w");
     fprintf(f,S);
     fclose(f);
}
else
{
     //на четвертую метку
     m_Result=S;
     UpdateData(FALSE);
}
}
```

При старте программы необходимо будет ввести значения `a`, `b` и `c`, после чего щелкнуть по кнопке **Решить**. Результаты работы программы представлены на рис. 11.30.

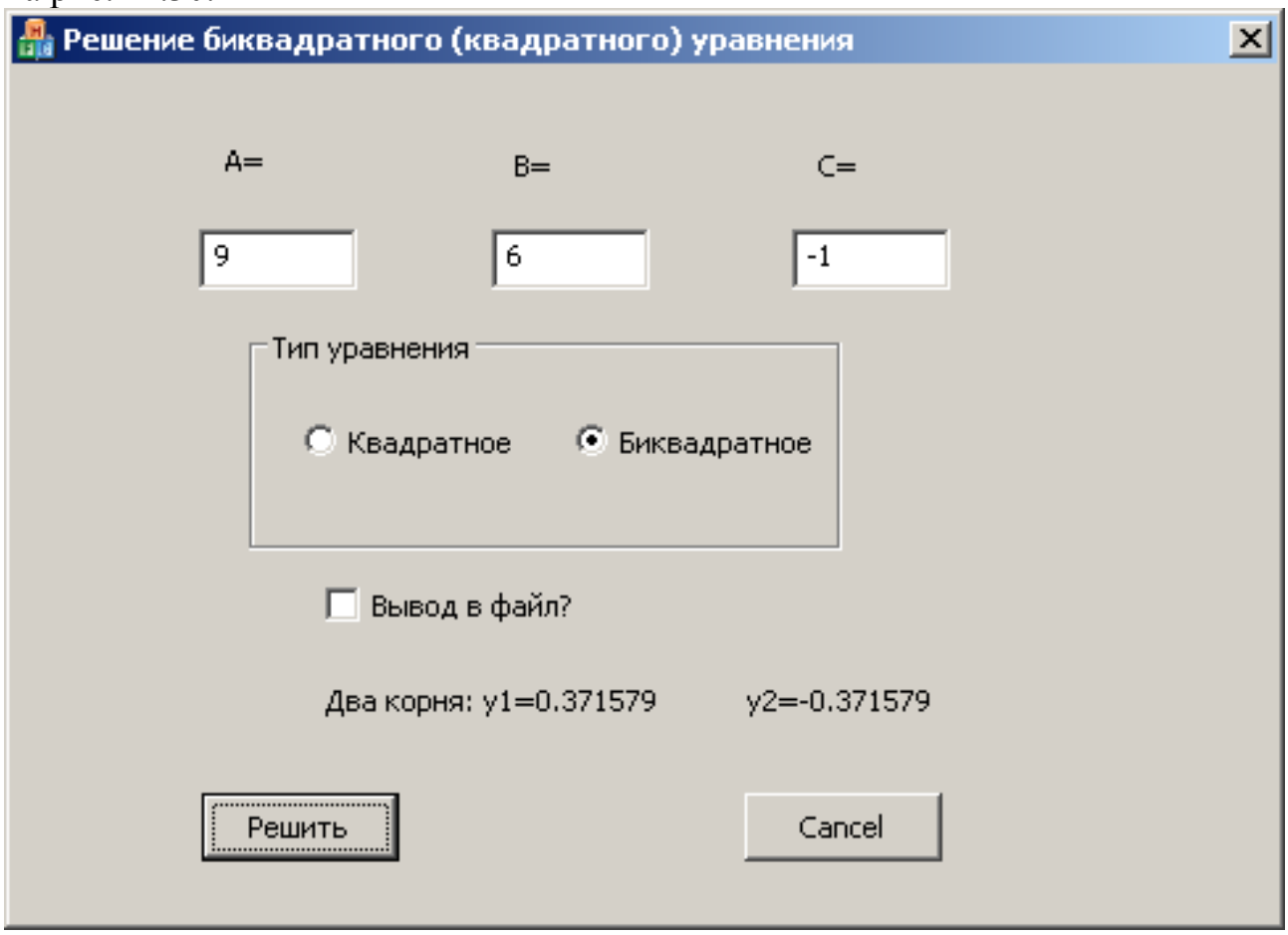

Рис. 11.30. Результаты работы программы

В тексте файла, в который мы дописывали функцию `OnBnClickedSolve()` есть функция `OnInitDialog`, которая запускается при старте программы и инициализирует создаваемое программой диалоговое окно. В ее текст перед оператором `return TRUE;` можно записывать свои

операторы, в которые буду выполняться сразу после создания диалогового окна, очень часто в функцию OnInitDialog записывают операторы, в которых происходит начальное присваивания значений меток и полей редактирования. В нашем случае в функции OnInitDialog можно определить значения переменных m_editA, m_editB, m_editC и m_message.

Можно дописать в функцию OnInitDialog следующие строки перед оператором RETURN TRUE.

```
m_EditA=-2;
m_EditB=3;
m_EditC=-4;
m_message=false;
//Обновление полей для чтения
UpdateData(FALSE);
```

После этого при старте программы, в окнах редактирования появятся начальные значения a, b и c равные −2, 3 и −4 соответственно.

ПРИМЕР 3. Переменная x меняется от xn до xk с шагом dx. Вывести таблицу значений x и y ($y=e^{\sin(x)}$), вычислить сумму и произведение y.

Создадим диалоговое окно и разместим на нем известные нам компоненты: 3 метки (класс Static Text), три поля ввода (класс Edit Control), предназначенные для ввода xn, xk, dx и компонент Список (класс List Box), предназначенный для отображения результатов. Изменяя месторасположение и свойство Caption компонентов, добьемся того, чтобы диалоговое окно стало таким (рис. 11.31).

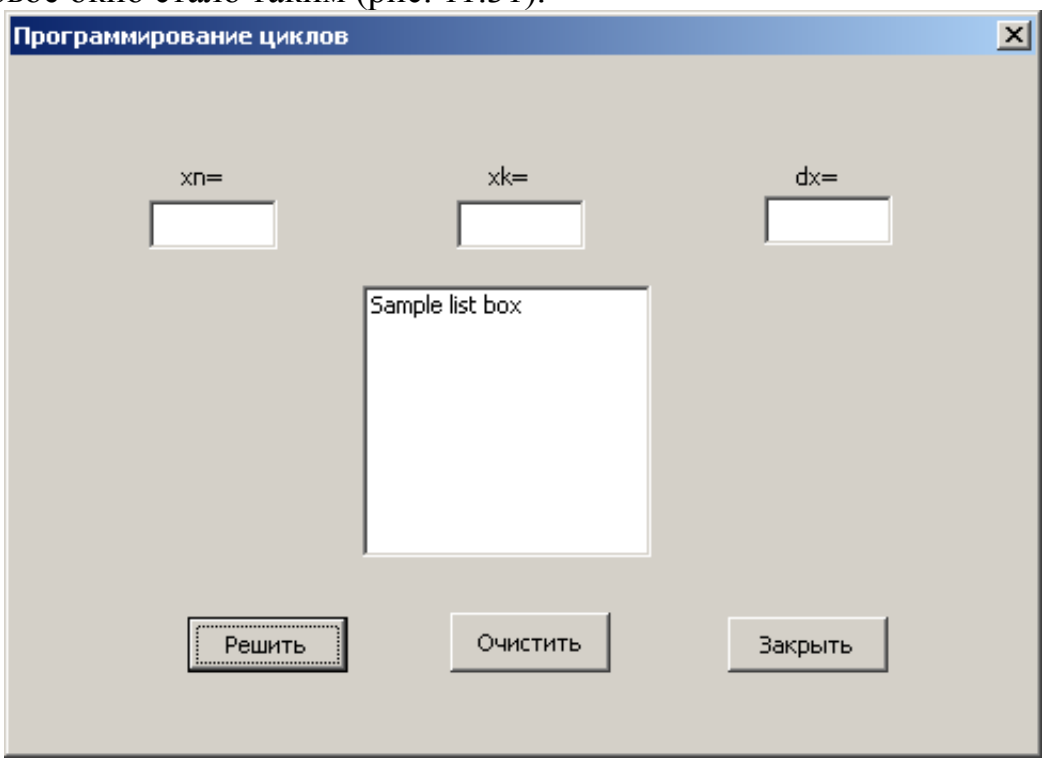

Рис. 11.31. Диалоговое окно для примера 3

Для полей ввода определим переменные m_edit_xn, m_edit_xk и

m_edit_dx (контекстное меню, **Add Variable**), которые будут возвращать значения (Value) типа float.

Изменим ID кнопок **Решить** и **Очистить**, пусть это будут значения ID_Solve и ID_Clear соответственно. Кнопка **Решить** будет решать поставленную нами задачу и выводить результат в List Box, а кнопка **Очистить** – очищать List Box.

Для компонента List Box определим переменную (контекстное меню, **Add Variable**) m_List_REsult (рис. 11.32), которые будут возвращать значения (Value) типа float.

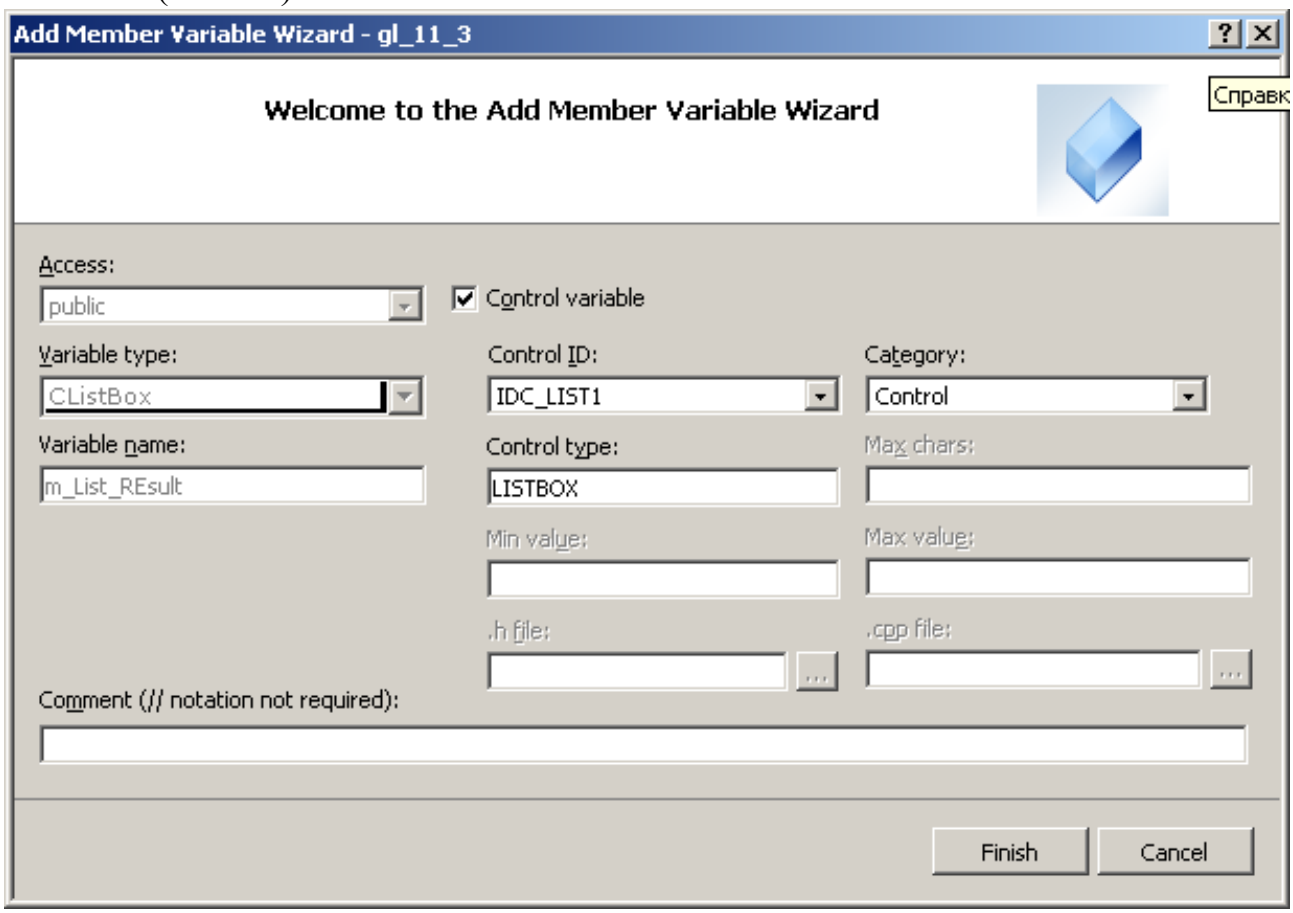

Рис. 11.32. Добавление переменной для компонента Список (List Box)

В свойствах List Box следует установить значение параметра Sort в False (иначе содержимое List Box будет сортироваться по алфавиту), значение параметра Vertical scroll в True.

Для того, чтобы при инициализации диалога выводились выводились начальные значения xn, xk, dx добавим в функцию OnInitDialog (перед оператором return) следующие строки:

```
m_edit_xn=-2;m_edit_xk=2;m_edit_dx=0.2;UpdateData(FALSE);
```

Теперь при запуске программы диалоговое окно примет вид как на рис. 11.33.

Для того, чтобы написать функцию, отвечающую за работу кнопки Решить, щелкаем по ней и вводим следующий текст функции

OnBnClickedSolve ()

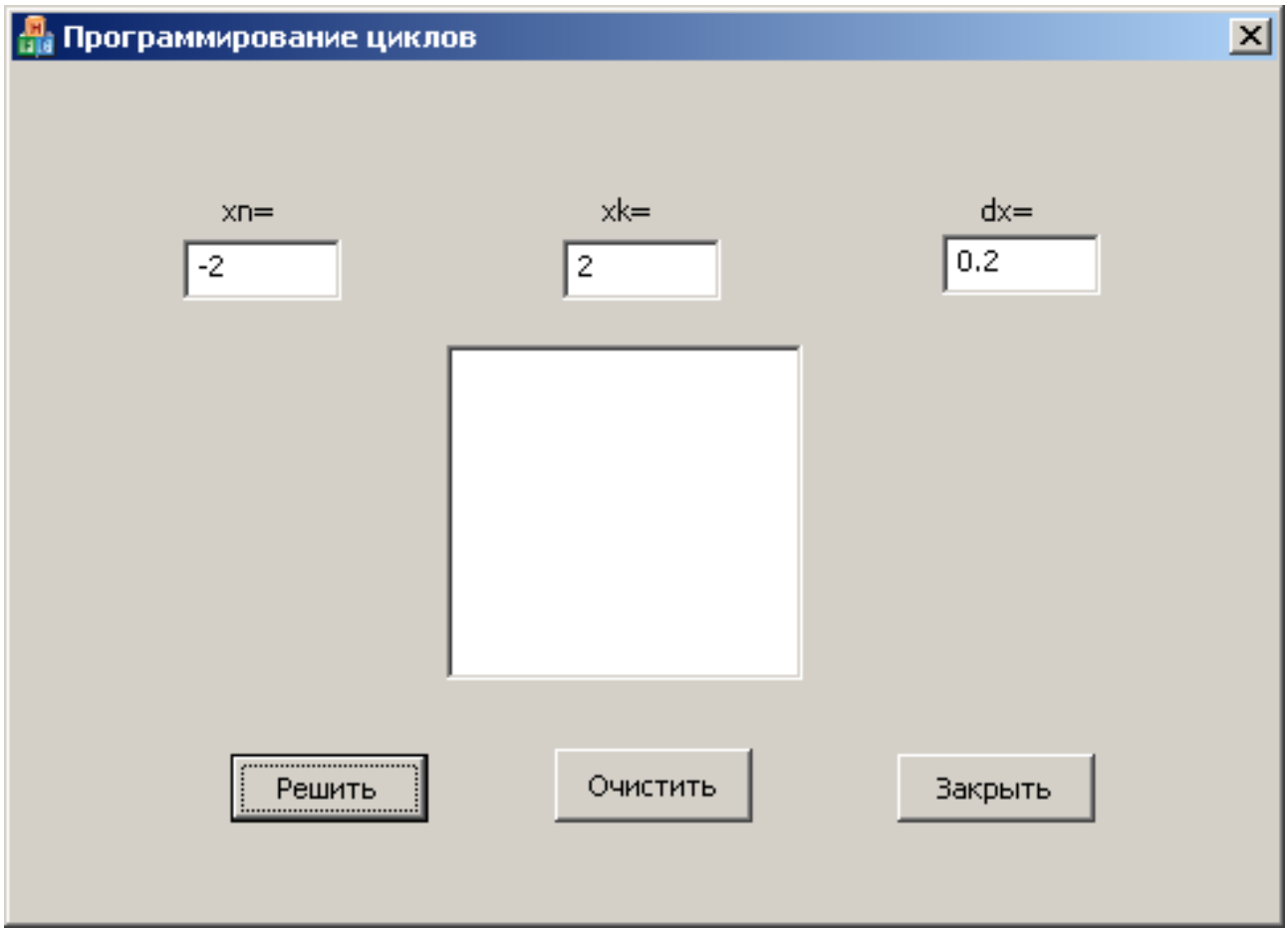

Рис. 11.33. Окно программы после внесения изменений в функцию OnInitDialog

```
//Cgl_11_3Dlg – имя класса «диалоговое окно»,
// совпадающего с именемсозданного проекта.
void Cgl_11_3Dlg::OnBnClickedSolve()
{int i;
float x,xn,xk,dx,y,s,p;
//Строка для хранения результатов
CString S;
UpdateData(TRUE);
xn=m_edit_xn;
xk=m_edit_xk;
dx=m_edit_dx;
for(s=0, p=1, x=xn,i=0; x<=xk; x+=dx,i++)
{
y=exp(sin(x));s+=y;p*=y;
    //Формирование строки вида x=…, y… .
S.Format(_T("x=%g y=%g"),x,y);
    //Добавление сформированной строки в
    //компонент Список (ListBox)
m_List_REsult.AddString(S);
```

```
}
     //Формирование строки вида s=…, p… .
S.Format(_T("s=%g p=%g"),s,p);
     //Добавление сформированной строки S в
     //компонент Список (ListBox) под
     //номером 0. Функция InsertString(k,S)
     //вставляет строку S в список под номером k.
m_List_REsult.InsertString(0,S);
}
```

Для того, чтобы написать функцию, отвечающую за работу кнопки **Очистить,** щелкаем по ней и вводим следующий текст функции `OnClear()`

```
void CTsiklDlg::OnClear()
{
int j,N;
     //Вычисляем количество строк в списке
N=m_List_REsult.GetCount();
     //Последовательно (j меняется от 0 до N-1)
     //удаляем строку с номером 0.
for(j=0;j<=N;j++)
m_List_REsult.DeleteString(0);
}
```

В результате получится работающее приложение, внешний вид которого после щелчка по кнопке **Решить** представлен на рис. 11.34.

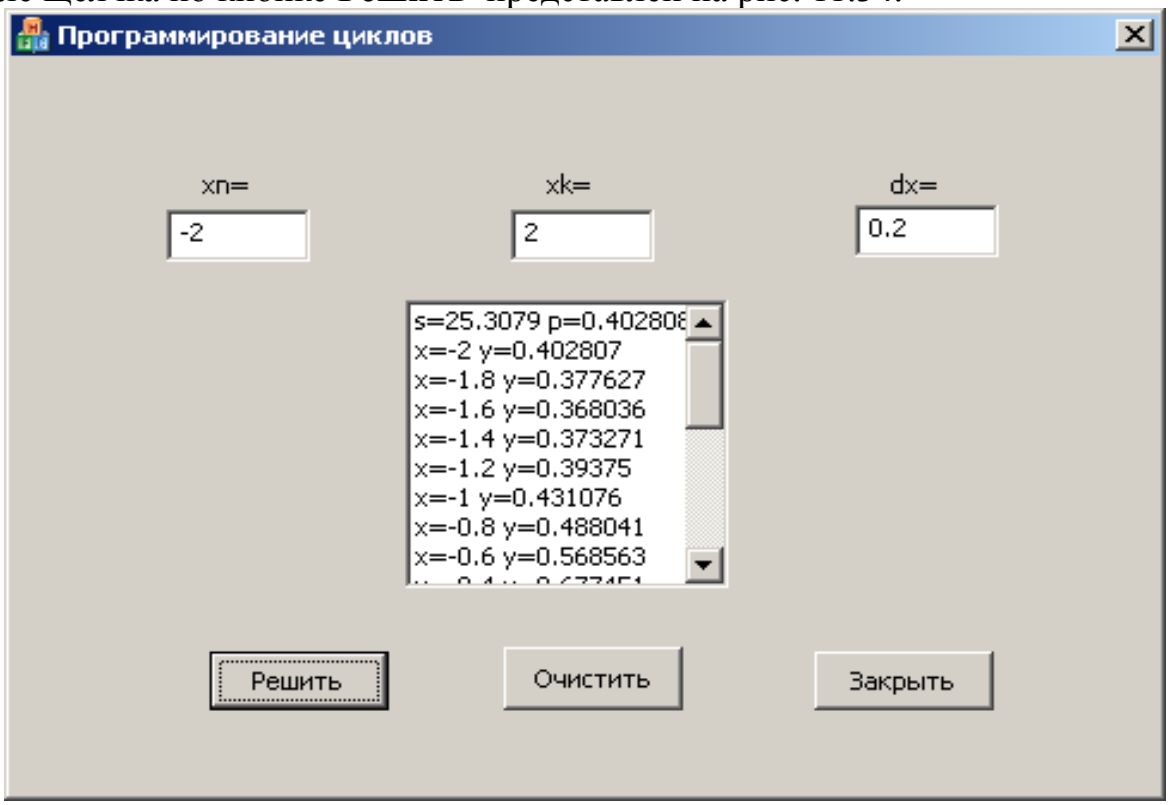

Рис. 11.34. Программа решения задачи

Аналогичным образом можно писать программы обработки массивов в

Microsoft Visual C++. Для вывода можно использовать компонент `List Box`. Для ввода массива можно использовать компонент `Edit Box`, в котором ввести числа, разделенные пробелом, потом считать введенные значения в строку. Преобразовать строку в массив чисел, выполнить необходимые действия с массивом.

Таким образом, при создании программ в среде MS Visual C++ необходимо:

1. Создать шаблон диалогового окна.
2. Разместить на нем необходимые компоненты.
3. Определяем переменные, необходимые для работы с компонентами.
4. Создаем функции-обработчики событий.

11.2. Визуальное программирование в Turbo C++ Explorer

Вернемся к примеру 2. Создание проекта начнем с создания формы, для чего можно выполнить команду **File – New – VCL Forms Application C++ Builder**. После этого Turbo C++ Explorer станет таким, как представлено на рис. 11.35.

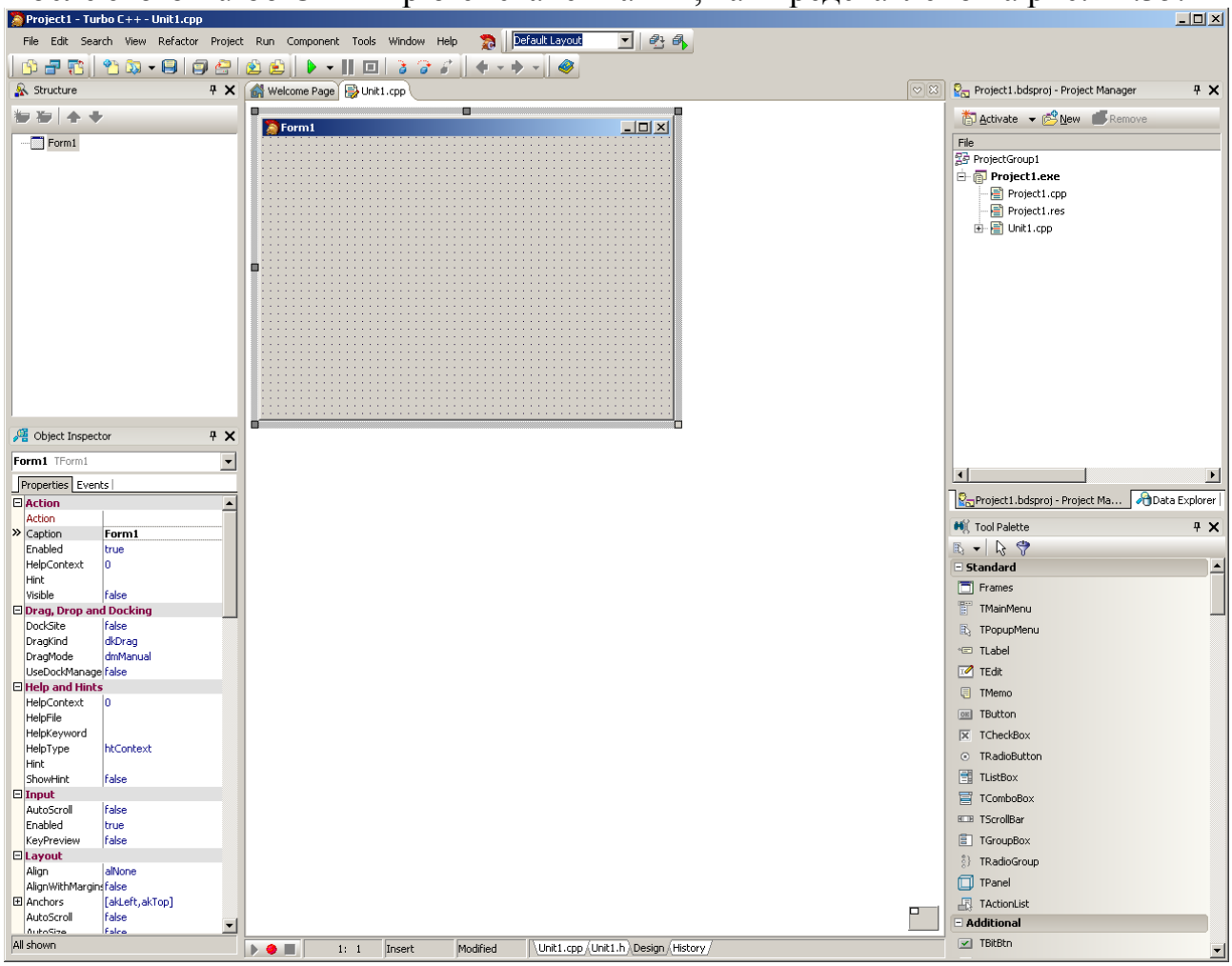

Рис. 11.35. Окно нового проекта в Turbo C++ Explorer

Теперь сразу сохраним проект, для этого выберем команду **File – Save Project As**. Для сохранения проекта лучше выбрать отдельную папку. При сохранении необходимо указать имя файла с расширением **cpp** и имя проекта (с расширением **bdsproj**).

Если выделить форму, то в левой части программы Turbo C++ Explorer появится окно инспектора объектов (**Object Inspector**) (рис. 11.36), в котором перечислены все свойства выделенного объекта (вкладка **Properties**).

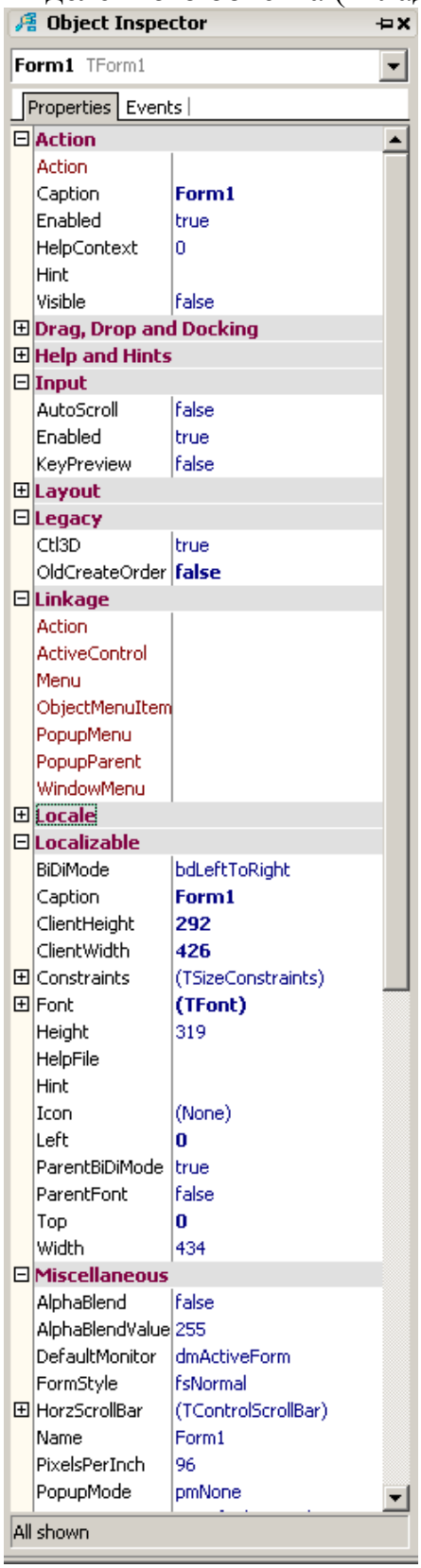

Рис. 11.36. Окно инспектора объектов

В данном случае вы видите свойства формы (окна). Среди свойств любого

объекта следует выделить:

1. `Name` – имя объекта в программе.
2. `Caption` – заголовок, для формы – это текст, который появляется в заголовке окна, для кнопки это свойство будет определять текст, расположенный на кнопке, для метки – текст, выводимых на метку и т.д.

Первоначально свойство `Name` и `Caption` зачастую имею одинаковые значения, но следует понимать, в чем различие между ними. Изменение свойства `Caption` приведет к изменению внешнего вида объекта, а изменение свойства `Name` может привести к необходимости модифицировать весь текст программы. В качестве свойства `Caption` формы запишем текст `Решение биквадратного(квадратного) уравнения`.

После этого нанесем на форму:

- четыре метки (класс `TLabel`), служат три служат для вывода заголовков полей ввода, четвертая – для вывода результатов;
- три поля редактирования (класс `TEdit`) для ввода значения `a`, `b`, `c`;
- группу из двух радиокнопок (класс `TRadioButton`) для выбора типа уравнения (квадратное или биквадратное);
- флажок (класс `TCheckBox`), определяющий, как будет происходить вывод (на четвертую метку или в файл);
- две кнопки (класс `TButton`) для решения уравнений и завершения программы.

После этого форма станет такой, как показано на рис. 11.37. Изменим месторасположение и свойство `Caption` компонентов таким образом, чтобы форма приняла вид, подобный, представленному на рис. 11.38.

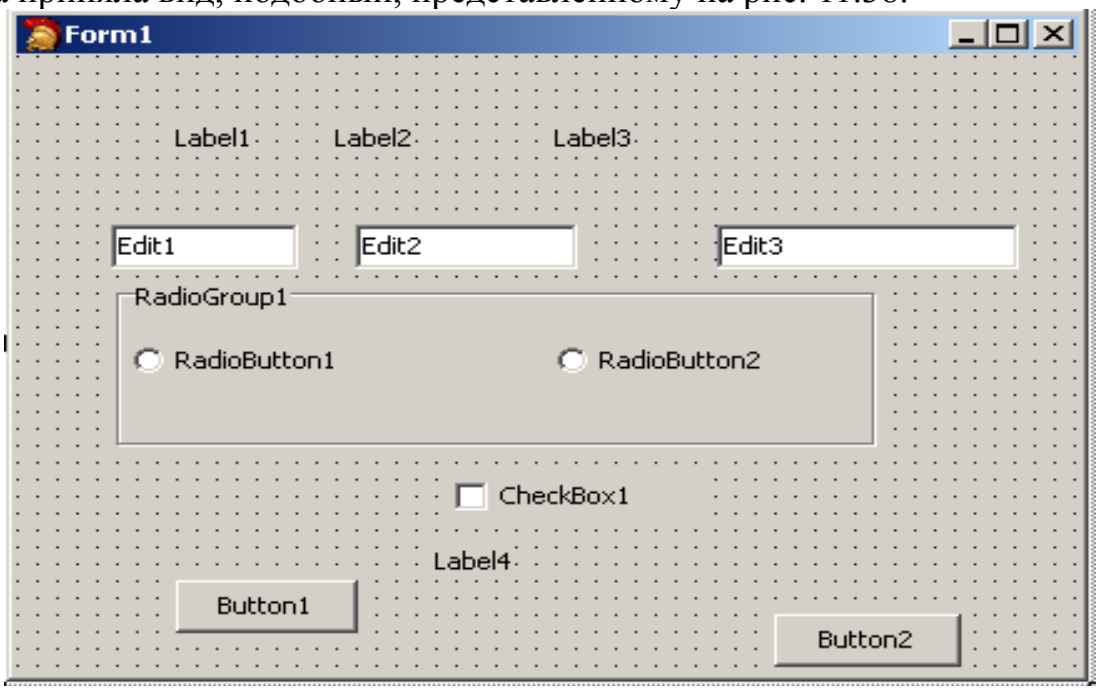

Рис. 11.37. Окно формы с компонентами

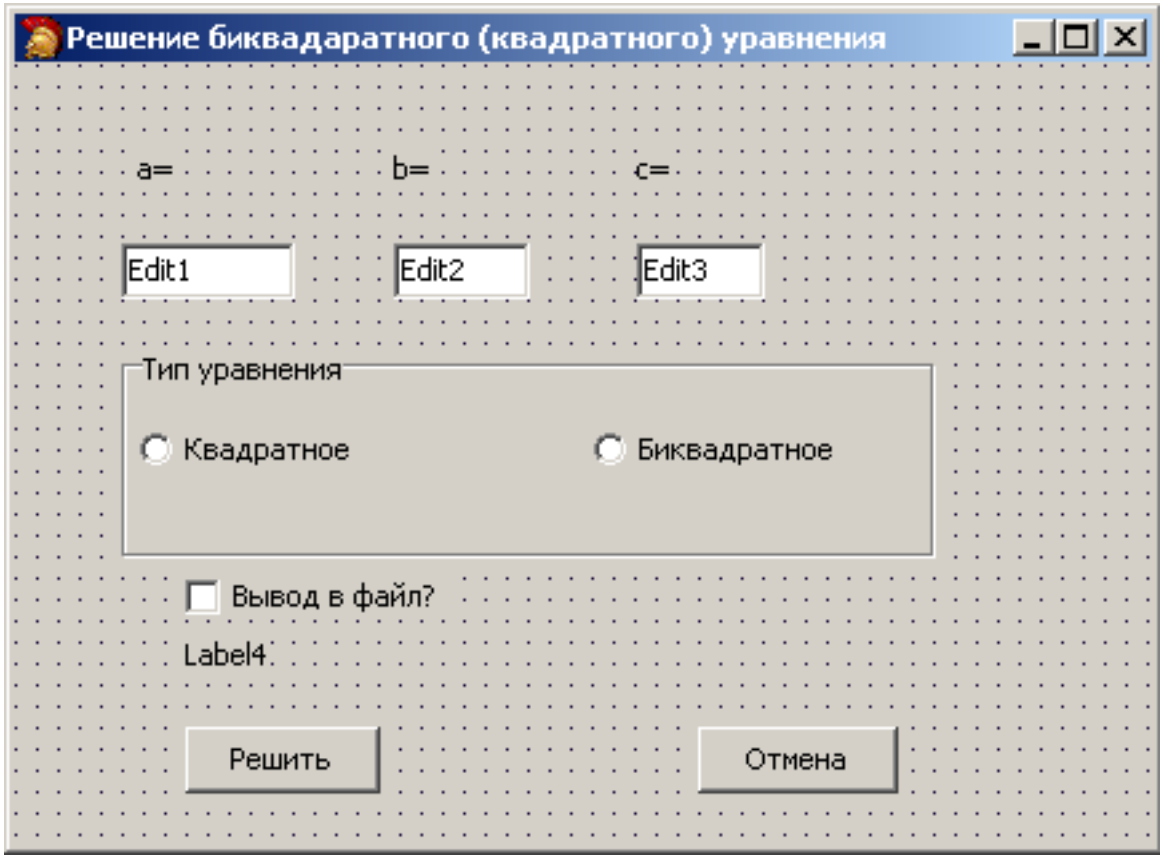

Рис. 11.38. Модифицированное окно формы

Теперь изменим свойство `Name` у полей редактирования (`TEdit`) и четвертой кнопки (`TLabel`). Для полей редактирования значения свойства `Name` установим `m_edit_a`, `m_edit_b` и `m_edit_c` соответственно. Значение `label_result` установим в качестве свойства `Name` четвертой метки. После этого для обращения к полям ввода в программе на C++ необходимо использовать имена `m_edit_a`, `m_edit_b` и `m_edit_c`, а для обращения к четвертой метке – `label_result`.

У большинства компонентов есть свойство `Visible`, отвечающее за видимость его на форме, по умолчанию это свойство принимает значение `true`. Давайте установим это свойство для компонента `label_result` в `false`. Сделаем компонент невидимым до тех пор, пока он не будет использоваться для вывода результатов. Перед выводом на метку результатов установим свойство `Text` для компонента `label_result` в `true`.

У полей редактирования есть свойство `Text`, отвечающее за выводимый в компоненте текст, по умолчанию это свойство совпадает с именем компонента. Установим в качестве значений свойства `Text` строки «`-2`», «`3`» и «`-4`» для компонентов `m_edit_a`, `m_edit_b` и `m_edit_c` соответственно. После этого при старте программы, в окнах редактирования вы увидите начальные значения `a`, `b` и `c` равные `-2`, `3` и `-4`.

У компонента `RadioButton` есть свойство `Checked`, которое принимает значение `true`, если кнопка выбрана и `false` – в противном случае.

Радиокнопка, отвечающая за квадратное уравнение имеет имя (свойство Name) RadioButton1, за биквадратное – RadioButton2. Для определенности у кнопки RadioButton1 свойство Checked установим в true, а у кнопки RadioButton1 – в false.

Флажок (в нашем примере компонент с именем CheckBox1) также имеет свойство Checked, которое принимает значение true, если флажок включен и false – в противном случае.

Осталось написать обработчики событий для кнопок **Решить** и **Отмена**. Для того, чтобы написать обработчик события щелчка по кнопке в Turbo C++ Explorer можно поступить одним из способов:

1. Дважды щелкнуть по кнопке.
2. Выделить кнопку на форме, после чего в инспекторе объектов (рис. 11.36), перейти на вкладку **Events** (события) и дважды щелкнуть правее события **OnClick**.

После чего на экране появится шаблон функции обработчика события и можно начинать вводить код функции на C++.

Дважды щелкнем по кнопке **Отмена** и в появившемся тексте функции введем единственную строку Close().Текст функции обработчика события должен быть таким:

```
void __fastcall TForm1::Button2Click(TObject *Sender)
{
//Функция Close закрывает текущее окно.
Close();
}
```

Дважды щелкнем по кнопке **Решить** и в появившемся тексте функции введем текст функции обработчика события.

В Turbo C++ Explorer обращение к полям (членам и методам) визуальных классов поддерживается только с помощью оператора ->.

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
ofstream f;
//Открываем файл в режиме записи, режим ios::out
//устанавливается по умолчанию.
f.open("result.txt");
float a,b,c,d,x1,x2,y1,y2, y3, y4;
//Если флажок вывода в файл выключен,
if (!(CheckBox1->Checked))
//то метку вывода результатов делаем видимой.
label_result->Visible=true;
else
//Если флажок вывода в файл включен,
//то метку вывода результатов делаем невидимой.
label_result->Visible=false;
```

```
//Считываем значения a, b и c из полей редактирования. //
Функция StrToFloat преобразовывает строку в
//вещественное число. Есть подобная функция StrToInt.
a=StrToFloat(m_edit_a->Text);
b=StrToFloat(m_edit_b->Text);
c=StrToFloat(m_edit_c->Text);
//Вычисляем дискриминант.
d=b*b-4*a*c;
//Проверяем выбрана ли радиокнопка, отвечающая за решение
//квадратного уравнения.
if (RadioButton1->Checked)
//Решение квадратного уравнения.
if (d<0)
//Если флажок вывода в файл включен, то выводим реультаты
//в текстовый  файл,
if (CheckBox1->Checked)
f<<"Дискриминант отрицателен, корней нет!!!";
//иначе результаты выводятся на метку вывода результатов
label_result.
else
label_result->Caption=
             "Дискриминант отрицателен, корней нет!!!";
else
{
    x1=(-b+sqrt(d))/2/a;
    x2=(-b-sqrt(d))/(2*a);
    if (CheckBox1->Checked)
    f<<"x1="<<x1<<"\tx2="<<x2;
    else
//Функция FloatToStr преобразовывает вещественное число в
//строку. Есть подобная функция IntToStr.
    label_result->Caption=
             "x1="+FloatToStr(x1)+"\tx2="+FloatToStr(x2);
}
else
{
    if (d<0)
    if (CheckBox1->Checked)
//Если флажок вывода в файл включен, то выводим реультаты
//в текстовый файл
    f<<"Дискриминант отрицателен, корней нет!!!";
    else
//иначе результаты выводятся на метку вывода
результатов //label_result.
```

```
    label_result->Caption=
            "Дискриминант отрицателен, корней нет!!!";
    else
    {
//Если выбрана радиокнопка, отвечающая за решение
//биквадратного уравнения, то решаем его.
    x1=(-b+sqrt(d))/2/a;
    x2=(-b-sqrt(d))/(2*a);
    if ((x1<0)&&(x2<0))
    if (CheckBox1->Checked)
//Если флажок вывода в файл включен, то выводим реультаты
//в текстовый файл
    f<<"Корней нет!!!";
//иначе результаты выводятся на метку вывода
результатов //label_result.
    else
        label_result->Caption="Корней нет!!!";
    else
        if ((x1>=0)&&(x2>=0))
        {
        y1=sqrt(x1);
        y2=-y1;
        y3=sqrt(x2);
        y4=-y3;
//Если флажок вывода в файл включен, то выводим реультаты
//в текстовый файл
    if (CheckBox1->Checked)
    f<<"Четыре корня
y1="<<y1<<"\ty2="<<y2<<"\ty3="<<y3<<"\ty4="<<y4;
    else
//иначе результаты выводятся на метку вывода
результатов //label_result.
    label_result->Caption="Четыре корня y1=
        " +FloatToStr(y1) + "\ty2=" + FloatToStr(y2) +
        "\ty3="+FloatToStr(y3)+"\ty4="+FloatToStr(y4);
        }
        else
        if (x1>=0)
        {   y1=sqrt(x1); y2=-y1;
//Если флажок вывода в файл включен, то выводим реультаты
//в текстовый файл
    if (CheckBox1->Checked)
    f<<"Два корня, y1="<<y1<<"\ty2="<<y2;
    else
```

```
//иначе результаты выводятся на метку вывода
результатов //label_result.
    label_result->Caption="Два корня y1=
            " +FloatToStr(y1) +"\ty2=" + FloatToStr(y2);
        }
        else
        {y1=sqrt(x2);    y2=-y1;
//Если флажок вывода в файл включен, то выводим реультаты
//в текстовый файл
if (CheckBox1->Checked)
    f<<"Два корня, y1="<<y1<<"\ty2="<<y2;
    else
//иначе результаты выводятся на метку вывода
результатов //label_result.
    label_result->Caption="Два корня y1=
            "+FloatToStr(y1)+"\ty2="+ FloatToStr(y2);
        }   }}}
```

Приложение решения уравнения работает, текст работающего приложения приведен на рис. 11.39.

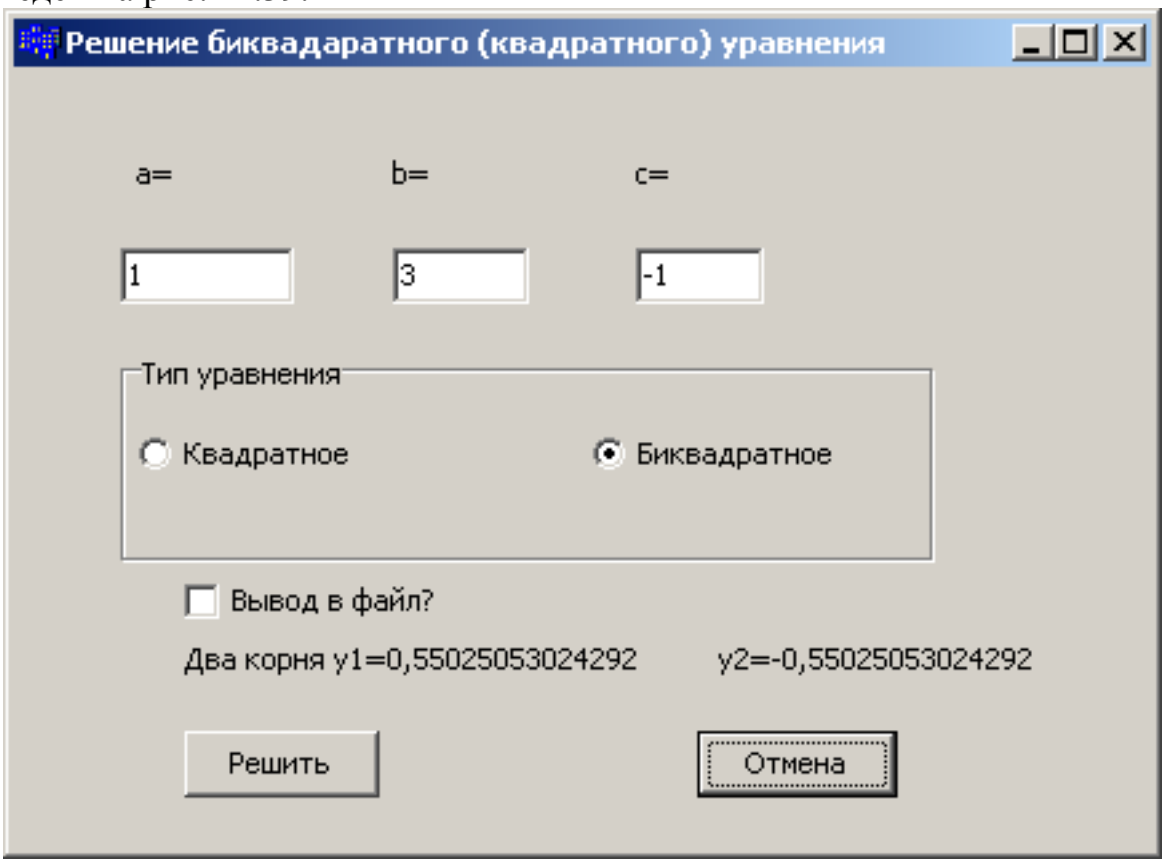

Рис. 11.39. Окно приложения решения уравнения, разработанного в Turbo C++ Explorer

На примере программы решения биквадратного уравнения мы рассмотрели только общие принципы визуального программирования в Turbo C++ Explorer.