Министерство образования Республики Беларусь
Учреждение образования
«Белорусский государственный университет
информатики и радиоэлектроники»

Кафедра электронных вычислительных средств

Д. С. Лихачёв, М. З. Лившиц

ИНТЕГРИРОВАННАЯ СРЕДА РАЗРАБОТКИ BORLAND BUILDER C++

Методическое пособие
по курсу
«Системное программирование»
для студентов специальности 1-40 02 02
«Электронные вычислительные средства»
дневной формы обучения

Минск БГУИР 2009

УДК 004.45 (075.8)
ББК 32.973.26 – 018.2 я73
Л 65

Р е ц е н з е н т
доцент кафедры ЭВМ БГУИР, канд. техн. наук А. А. Петровский

Лихачев, Д. С.

Л 65 Интегрированная среда разработки Borland Builder C++ : метод. пособие по курсу «Системное программирование» для студ. спец. 1-40 02 02 «Электронные вычислительные средства» днев. формы обуч. / Д. С. Лихачёв, М. З. Лившиц. – Минск : БГУИР, 2009. – 40 с. : ил.

ISBN 978-985-488-368-7

Данное методическое пособие содержит описание инструментальной системы программирования C++Builder 2006 применительно к задаче разработки Windows-приложений с графическим пользовательским интерфейсом и специфики использования программного кода на ассемблере совместно с языком C/C++. Описывается порядок разработки проекта, а также ход выполнения двух лабораторных работ по данной тематике.

УДК 004.45 (075.8)
ББК 32.973.26 – 018.2 я73

ISBN 978-985-488-368-7

1 Назначение и возможности инструментальной среды C++ Builder 2006

Система C++Builder 2006 предназначена для разработки Windows-приложений на языке C/C++ и содержит все необходимые для этого инструменты, включая средства визуальной разработки на базе технологии «drag-and-drop», утилиты автоматического завершения синтаксических конструкций, редактор web-страниц, работающий по принципу WYSIWYG, панель для просмотра структуры исходного кода C++, а также многочисленные встроенные средства контроля версий и поиска дефектов. Используемая в C++ Builder 2006 библиотека визуальных компонентов Visual Component Library (VCL) позволяет разработчикам на C++ использовать в процессе разработки программные компоненты от сторонних производителей. Являясь компонентом пакета Borland Developer Studio, система C++Builder 2006 предполагает полноценную поддержку механизмов организации коллективной работы в средах C/C++, C#, Delphi Win32 и Delphi .NET. Кроме того, имеется поддержка драйверов локальных баз данных Borland InterBase, Paradox, dBASE, dbGO for ADO и MySQL.

Для эффективной работы с системой C++ Builder 2006 персональные компьютеры должны удовлетворять следующим системным требованиям:

- компьютер на базе процессора Intel Pentium III 1.4 ГГц или Pentium IV 1.4 ГГц (Intel Pentium Pentium IV 2 ГГц или выше);
- Microsoft Windows Server 2003 (с пакетом исправлений SP1), Microsoft Windows XP Professional (с пакетом SP2), Windows 2000 Professional (с пакетом SP4), Windows 2000 Server (с пакетом SP4);
- 512 Мбайт оперативной памяти (рекомендуется 1 Гбайт);
- 1 Гбайт свободного дискового пространства для установки продукта;
- 750 Мбайт дискового пространства для установки необходимого сопутствующего программного обеспечения (включая Microsoft .NET Framework и SDK);
- устройство для чтения дисков CD-ROM или DVD.

2 Основные принципы работы с C++ Builder 2006

После запуска C++ Builder 2006 на экране появляется главное окно с пятью основными областями (рисунок 1):

- главное окно C++ Builder 2006 с основным меню и панелями инструментов;
- окно для визуальной разработки графического интерфейса, т.е. так называемое окно формы (по умолчанию имеет заголовок **Form1)**;
- окно редактора свойств объектов – **Object Inspector**;
- окно просмотра списка объектов – **ObjectTreeView**;
- окно редактора кода (по умолчанию имеет заголовок **Unit1.cpp)**.

Первоначально окно редактора кода почти полностью закрыто окном стартовой формы.

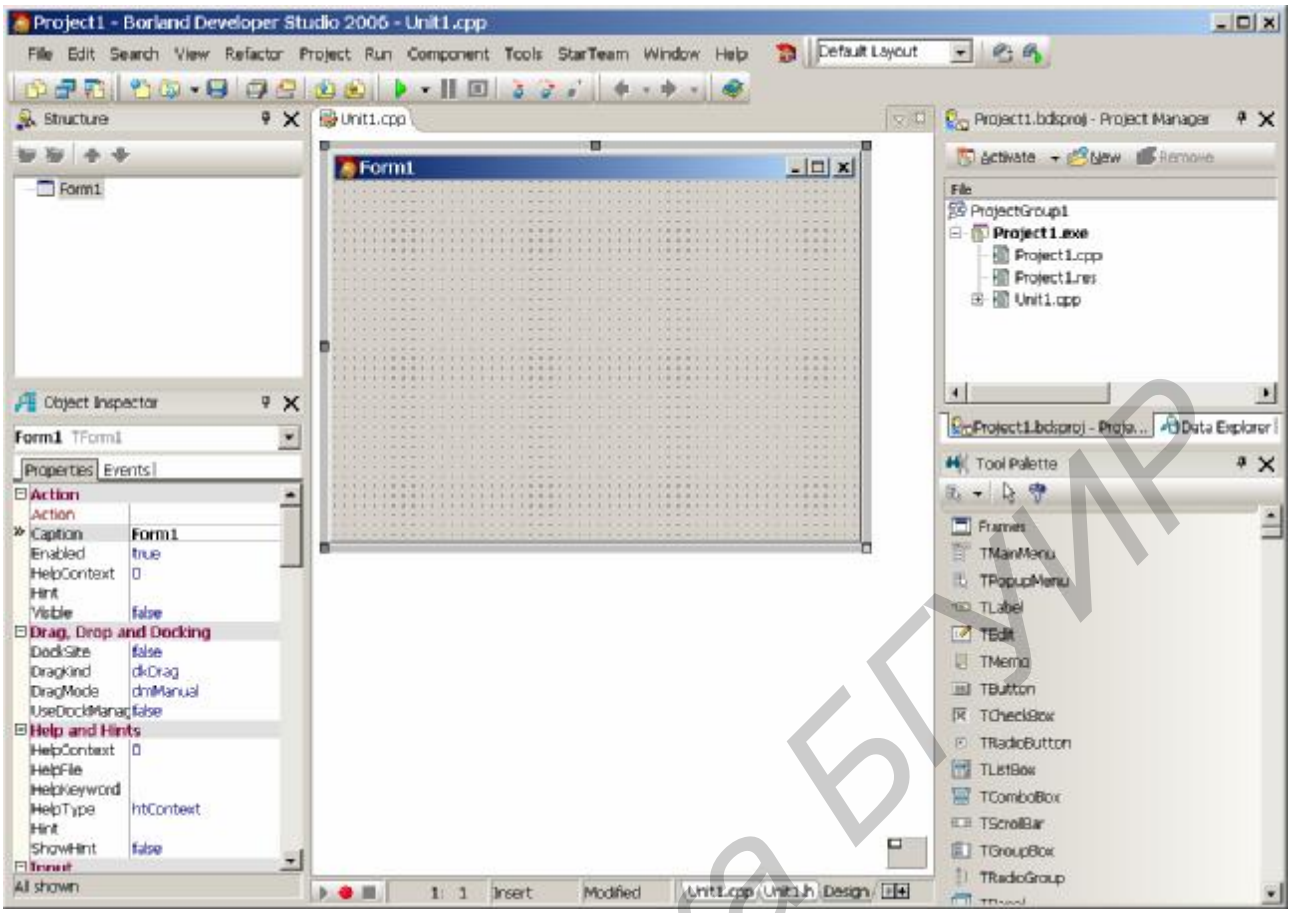

Рисунок 1 – Вид экрана после запуска C++ Builder 2006

В главном окне находится меню команд, панель инструментов и палитра компонентов (рисунок 2).

Рисунок 2 – Панель инструментов

Окно стартовой формы (**Form1**) представляет собой заготовку главного окна разрабатываемой программы (приложения).

Окно **Object Inspector** (рисунок 3) – окно редактора свойств объектов предназначено для редактирования значений свойств объектов. В терминологии визуального проектирования *объекты* – это диалоговые окна и элементы управления (поля ввода и вывода, командные кнопки, переключатели и др.).

Свойства объекта – это характеристики, определяющие вид, положение и поведение объекта. Например, свойства *Width* и *Height* задают размер (ширину и высоту) формы, свойства *Top* и *Left* – положение формы на экране, свойство *Caption* – текст заголовка. В верхней части окна указан объект (имя объекта), значения свойств которого отражены в окне **Object Inspector**.

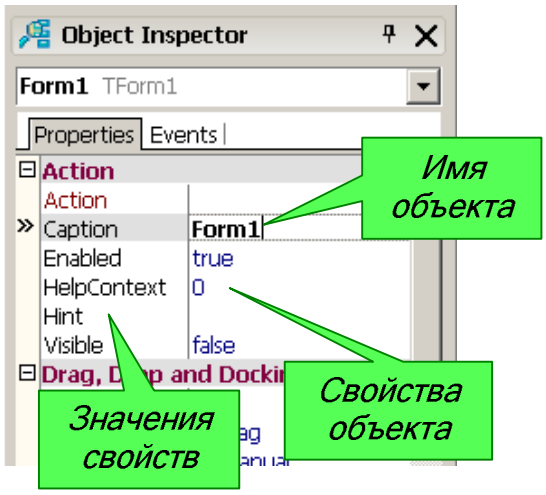

Рисунок 3 – Вкладка Properties окна **Object Inspector**

В окне редактора кода (рисунок 4), которое можно увидеть, отодвинув в сторону окно формы, набирается текст программы. В начале работы над новым проектом окно редактора кода содержит сформированный C++ Builder шаблон программы.

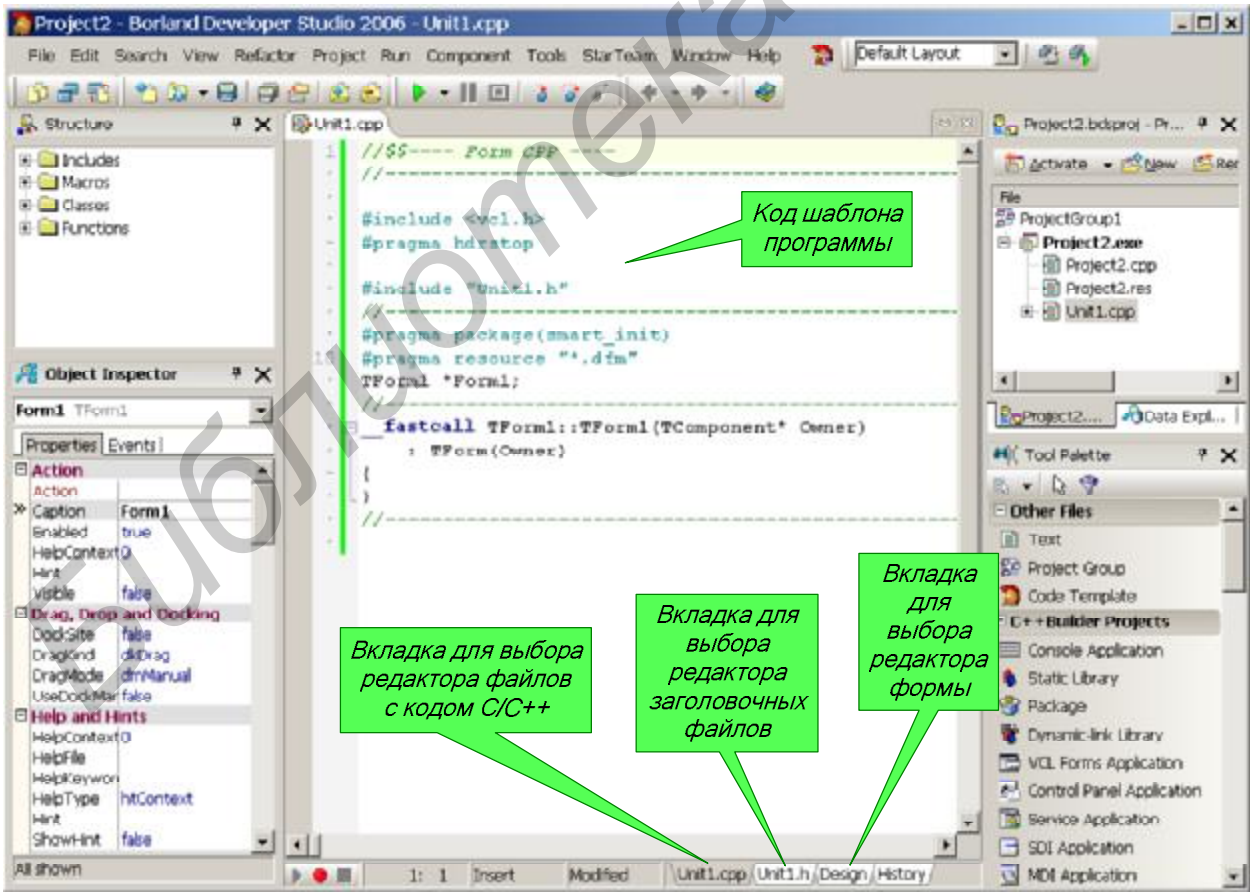

Рисунок 4 – Окно редактора кода

3 Последовательность разработки приложения со стандартным графическим интерфейсом

3.1 Пример создания пользовательского приложения

Для демонстрации технологии визуального проектирования приложений в среде C++ Builder 2006 ниже рассматривается пример разработки программы, которая производит вычисление выражения с двумя переменными в зависимости от выбранного условия: либо сумму двух чисел **a** и **b**, либо их произведение. Входные значения и результат отображаются на форме. Пользовательский интерфейс программы показан на рисунке 5.

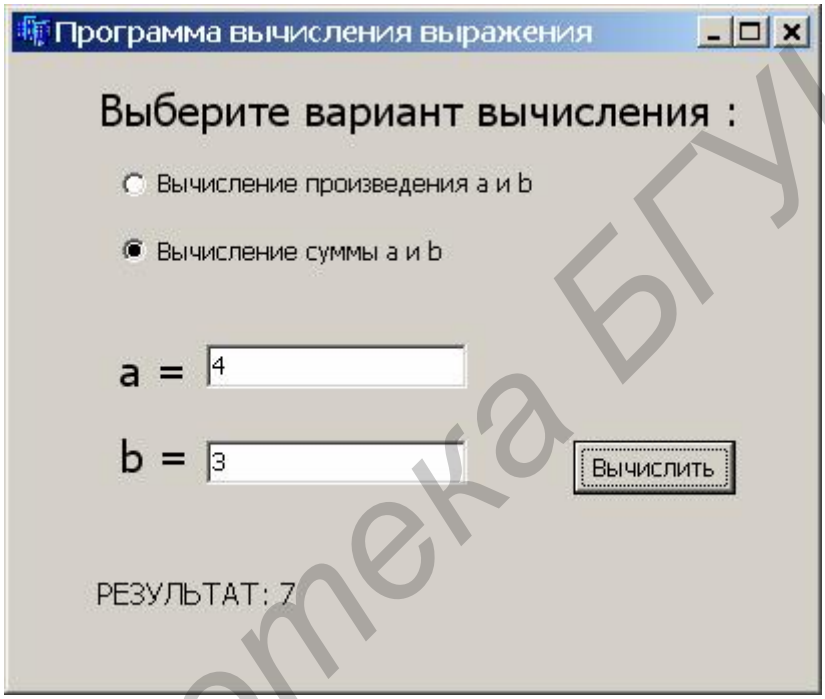

Рисунок 5 – Пользовательский интерфейс программы

Чтобы начать разработку нового приложения (так обычно называют прикладные программы) с графическим интерфейсом, надо запустить C++ Builder 2006, в меню **File** выбрать команду **New | VCL Forms Application – C++ Builder**. Ниже описана последовательность дальнейших действий.

3.2 Создание формы

Работа над новым проектом начинается с создания *стартовой формы* – главного окна программы. Стартовая форма создается путем изменения значений свойств формы **Form1** (настройки формы) и добавления к форме необходимых компонентов (полей ввода, полей вывода текстовой информации, командных кнопок и т.д.).

Основные свойства формы, которые определяют ее вид и поведение во время работы программы, приведены в таблице 1.

Таблица 1 – Свойства формы (объекта типа TForm)

Наименование свойства	**Значение свойства**
Name	*Имя формы*. В программе имя формы используется для управления формой и доступа к компонентам формы
Caption	*Текст заголовка*
Width	*Ширина формы*
Height	*Высота формы*
Top	*Расстояние от верхней границы формы до верхней границы экрана*
Left	*Расстояние от левой границы формы до левой границы экрана*
BorderStyle	*Вид границы*. Форма может быть с изменяемой границей (bsSizeable), фиксированной (bsSingle) или без границы (bsNone). Если у окна *изменяемая граница*, то во время работы программы пользователь может при помощи мыши изменить размер окна. Изменить размер окна с *фиксированной границей* нельзя. Если граница отсутствует, то на экран во время работы программы будет выведено окно без заголовка. Положение и размер такого окна во время работы программы изменить нельзя
BorderIcons	*Кнопки управления окном*. Значение свойства определяет, какие кнопки управления окном будут доступны пользователю во время работы программы. Значение свойства задается путем присвоения значений уточняющим свойствам biSystemMenu, biMinimaze, biMaximaze и biHelp. Свойство biSystemMenu определяет доступность кнопки **Свернуть** и кнопки системного меню, biMinimaze – кнопки **Свернуть**, biMaximaze – кнопки **Развернуть**, biHelp – кнопки вывода справочной информации
Icon	*Значок в заголовке диалогового окна, обозначающий кнопку вывода системного меню*
Color	*Цвет фона*. Цвет можно задать, указав название цвета или привязку к текущей цветовой схеме операционной системы. Во втором случае цвет определяется текущей цветовой схемой, выбранным компонентом привязки и меняется при изменении цветовой схемы операционной системы
Font	*Шрифт*. Шрифт, используемый «по умолчанию» компонентами, находящимися на поверхности формы. Изменение свойства **Font** формы приводит к автоматическому изменению свойства **Font** компонента, располагающегося на поверхности формы. То есть компоненты наследуют свойство **Font** от формы (имеется возможность запретить наследование)

Для изменения значений свойств объектов, в том числе и формы, используется вкладка *Properties* (Свойства) диалогового окна **Object Inspector**.

В левой колонке этой вкладки перечислены свойства выбранного объекта, а в правой указаны значения свойств.

При создании формы в первую очередь следует изменить значение свойства *Caption* (Заголовок). Чтобы это сделать, нужно в окне **Object Inspector** щелкнуть левой кнопкой мыши в строке *Caption* (в результате будет выделено значение свойства и появится курсор) и ввести текст. Аналогичным образом можно установить значения свойств *Height* и *Width*, которые определяют высоту и ширину формы. Размер формы, а также размер других компонентов задают в пикселах, т.е. точках экрана.

Форма – это обычное окно. Поэтому размер формы можно изменить точно так же, как размер любого окна Windows, т.е. путем перетаскивания границы. По окончании перемещения границы значения свойств *Height* и *Width* автоматически изменятся. Они будут соответствовать установленному размеру формы. Положение диалогового окна на экране после запуска программы соответствует положению формы во время разработки, которое определяется значением свойств *Top* (отступ от верхней границы экрана) и *Left* (отступ от левой границы экрана). Значения этих свойств также можно задать путем перемещения формы при помощи мыши. При выборе некоторых свойств, например, *Borderstyle*, справа от текущего значения свойства появляется значок раскрывающегося списка. Очевидно, что значение таких свойств можно задать путем выбора из списка (рисунок 6).

Некоторые свойства являются вложенными, т.е. их значение определяется совокупностью значений других (уточняющих) свойств. Например, свойство *BorderIcons* определяет, какие кнопки управления окном будут доступны во время работы программы. Значения этого свойства определяется совокупностью значений свойств *biSystemMenu*, и *biHelp*, каждое из которых в свою очередь определяет наличие соответствующей командной кнопки в заголовке окна во время работы программы. Перед именами сложных свойств стоит значок «+», в результате щелчка на котором раскрывается список уточняющих свойств (рисунок 7), значения которых можно задать обычным образом (ввести в поле или выбрать в списке допустимых значений).

В результате выбора некоторых свойств (щелчка кнопкой мыши на свойстве) рядом со значением свойства появляется командная кнопка с тремя точками.

Это значит, что задать значение свойства можно в дополнительном диалоговом окне, которое появится в результате щелчка на этой кнопке. Например, значение сложного свойства *Font* можно задать в окне **Object Inspector** путем ввода значений уточняющих свойств, а можно воспользоваться стандартным диалоговым окном *Шрифт*, которое появится в результате щелчка на кнопке с тремя точками (см. рисунок 7). Вид формы после редактирования свойств показан на рисунке 8.

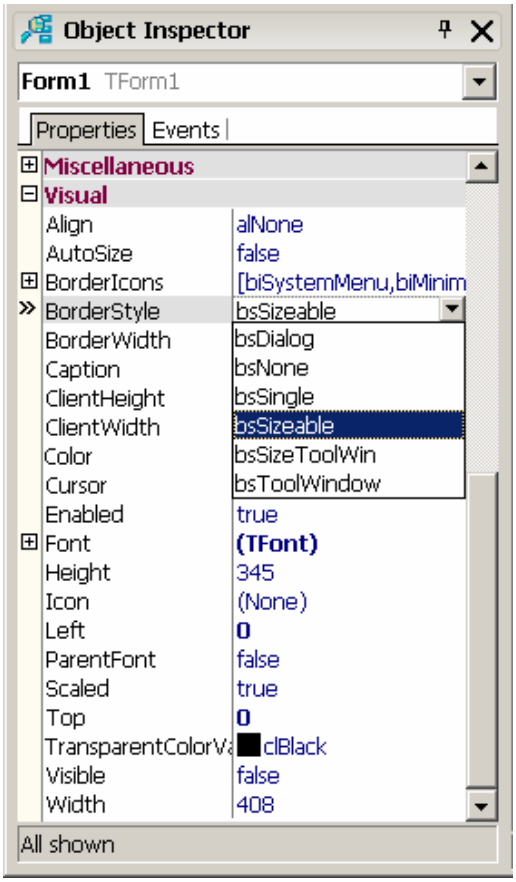

Рисунок 6 – Свойства окна Object Inspector BorderStyle

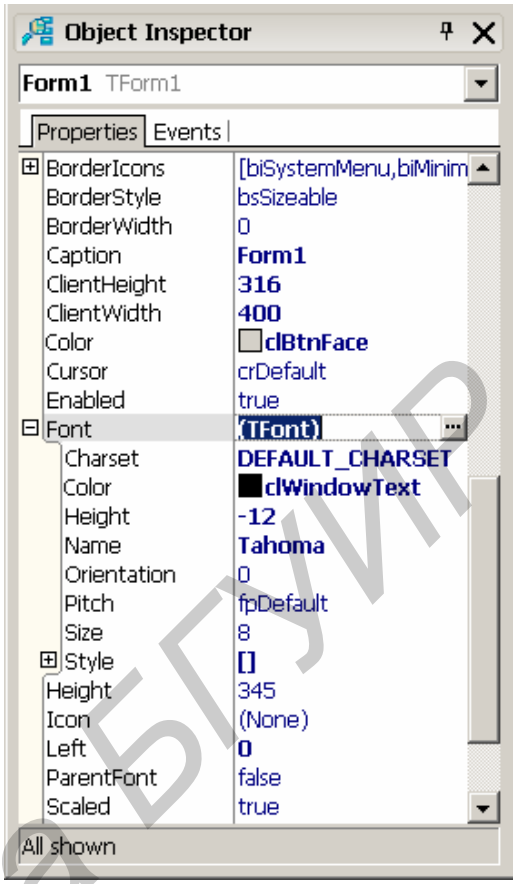

Рисунок 7 – Изменение значения уточняющего свойства

Рисунок 8 – Вид формы после редактирования свойств

3.3 Добавление элементов управления на форму

По условию разрабатываемая программа должна получить от пользователя исходные данные – целочисленные значения *a* и *b* и вариант вычисляемого выражения (сумма или произведение).

Значения переменных *a* и *b* удобно вводить с клавиатуры в редактируемые текстовые поля (компонент *TEdit*).

Для того чтобы в форму разрабатываемого приложения добавить редактируемое поле, надо в окне палитры компонентов **(ToolPalette)** найти и щелкнуть на значке компонента *TEdit* (рисунок 9), установить курсор в ту точку формы, в которой должен быть левый верхний угол компонента, и еще раз щелкнуть кнопкой мыши.

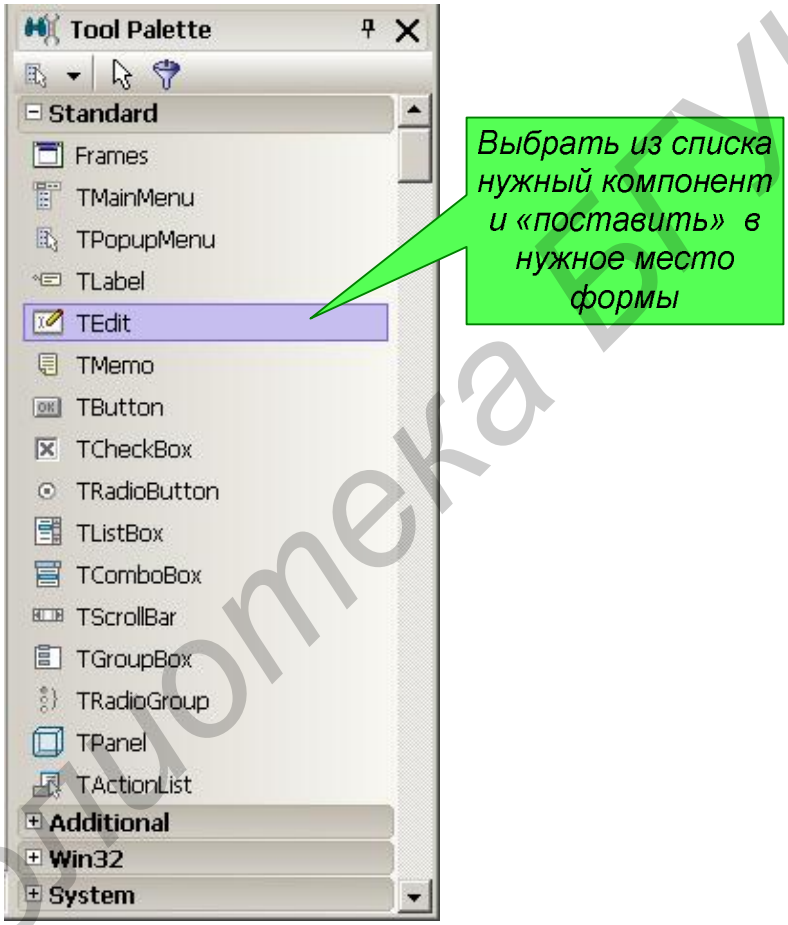

Рисунок 9 – Выбор компонента TEdit

В результате на форме появляется редактируемое поле *Edit1* (рисунок 10). Местоположение добавленных на форму элементов управления (компонентов), а также их размеры можно менять в течение всего процесса проектирования графического интерфейса.

Каждому добавленному компоненту автоматически присваивается имя, которое состоит из названия компонента и его порядкового номера. Например, если к форме добавить два компонента *TEdit*, то их имена будут *Edit1* и *Edit2*.

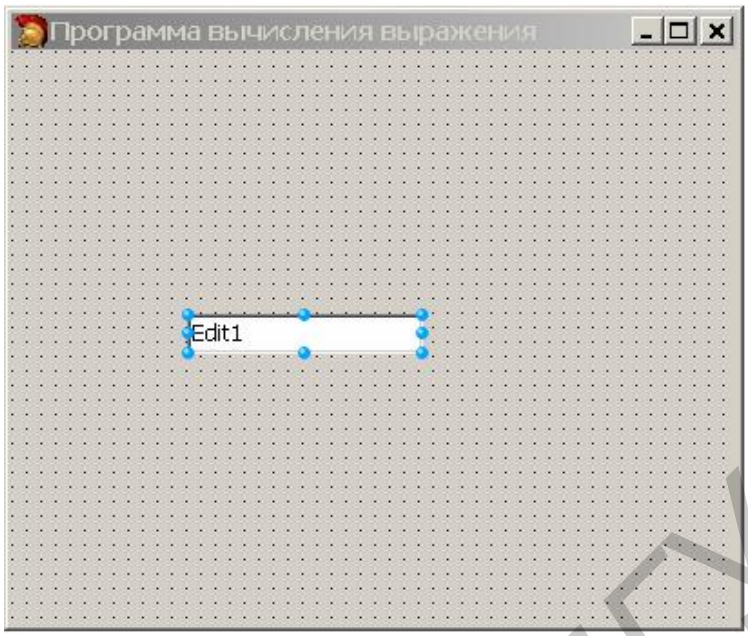

Рисунок 10 – Форма с редактируемым полем Edit1

Программист путем изменения значения свойства *Name* может изменить имя компонента. Основные свойства компонента *TEdit* приведены в таблице 2.

Таблица 2 – Свойства компонента Edit (объект типа TEdit)

Свойство	**Значение**
Name	*Имя компонента*. Используется в программе для доступа к компоненту и его свойствам, в том числе к тексту, который находится в поле редактирования
Text	*Текст, который находится в поле ввода/редактирования*
Left	*Расстояние от левой границы компонента до левой границы формы*
Top	*Расстояние от верхней границы компонента до верхней границы формы*
Height	*Высота поля*
Width	*Ширина поля*
Font	*Шрифт, используемый для отображения вводимого текста*
ParentFont	*Наследование шрифта*

При выборе установленного на форме компонента он выделяется по контуру «маленькими квадратиками». Свойства выбранного компонента отображаются в окне **Object Inspector**. Чтобы увидеть и, если необходимо, изменить свойства другого компонента, этот компонент выбирают, щелкнув левой кноп-

кой мыши на изображении компонента, или выбирают имя компонента в раскрывающемся списке, который находится в верхней части окна **Object Inspector** (рисунок 11). Компонент, свойства которого надо увидеть или изменить, можно выбрать и в окне **Structure** (рисунок 12).

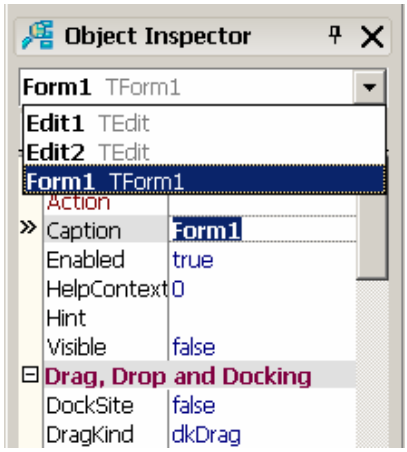

Рисунок 11 – Выбор компонента в окне Object Inspector

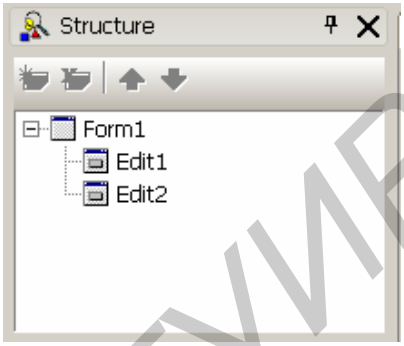

Рисунок 12 – Выбор компонента в окне Structure

Значения некоторых свойств компонента, определяющих, например, размер и положение компонента на поверхности формы, можно изменить при помощи мыши.

Для того чтобы изменить положение компонента, необходимо установить курсор мыши на его изображение, нажать левую кнопку мыши и, удерживая ее нажатой, переместить контур компонента в нужную точку формы, а затем отпустить кнопку мыши. Во время перемещения компонента отображаются текущие значения координат левого верхнего угла компонента – значения свойств *Left* и *Top*, рисунок 13.

Для того чтобы изменить размер компонента, необходимо его выделить, установить указатель мыши на один из маркеров, помечающих границу компонента, нажать левую кнопку мыши и, удерживая ее нажатой, изменить положение границы компонента. Затем отпустить кнопку мыши. Во время изменения размера компонента отображаются его текущие размеры: высота и ширина, т.е. значения свойств *Height* и *Width*, рисунок 14.

На рисунке 15 приведен вид формы после добавления и настройки двух полей редактирования для ввода входных значений переменных **a** и **b**. Значением свойства *Text* обоих компонентов является пустая строка.

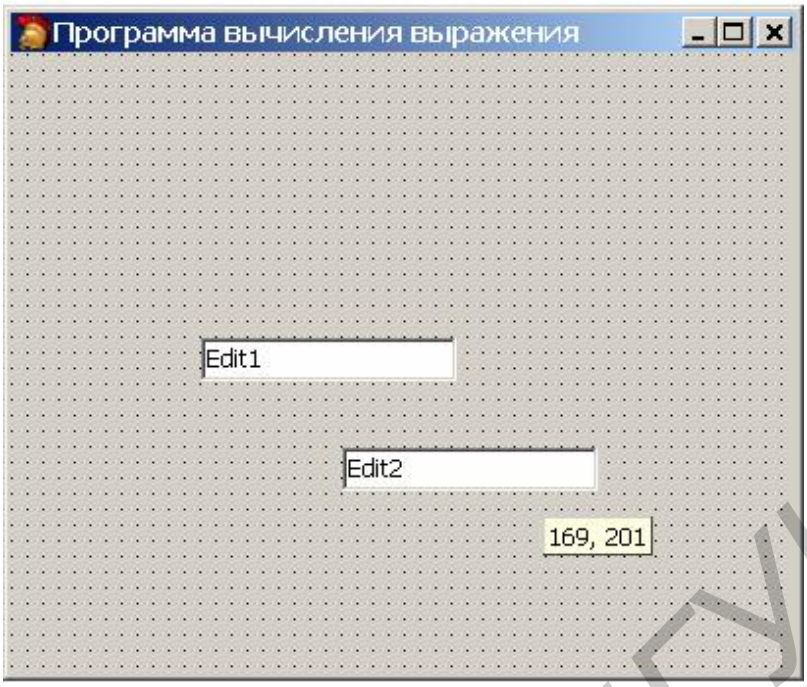

Рисунок 13 – Отображение значений свойств Left и Top при изменении положения компонента

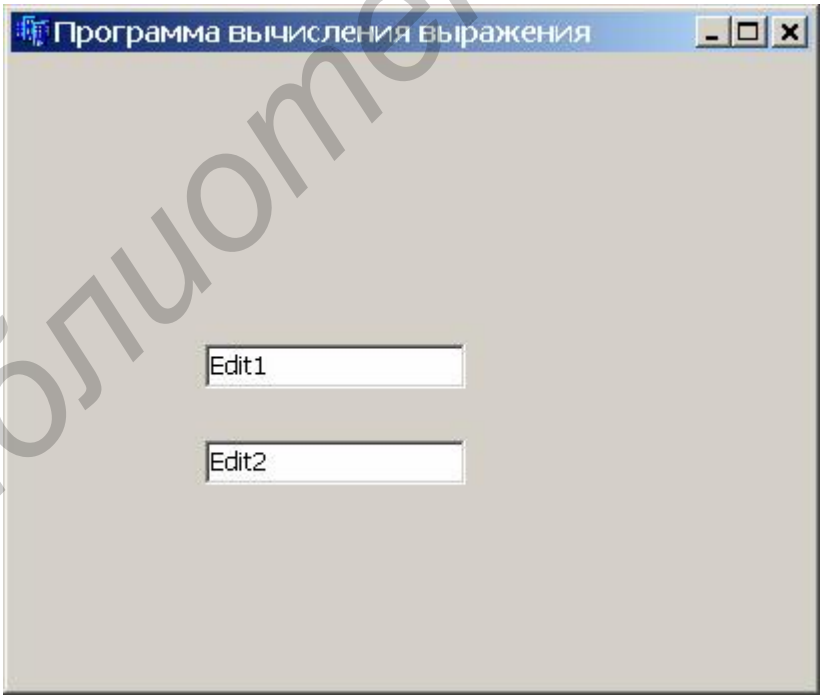

Рисунок 14 – Вид формы с двумя установленными редактируемыми полями Edit1 и Edit2

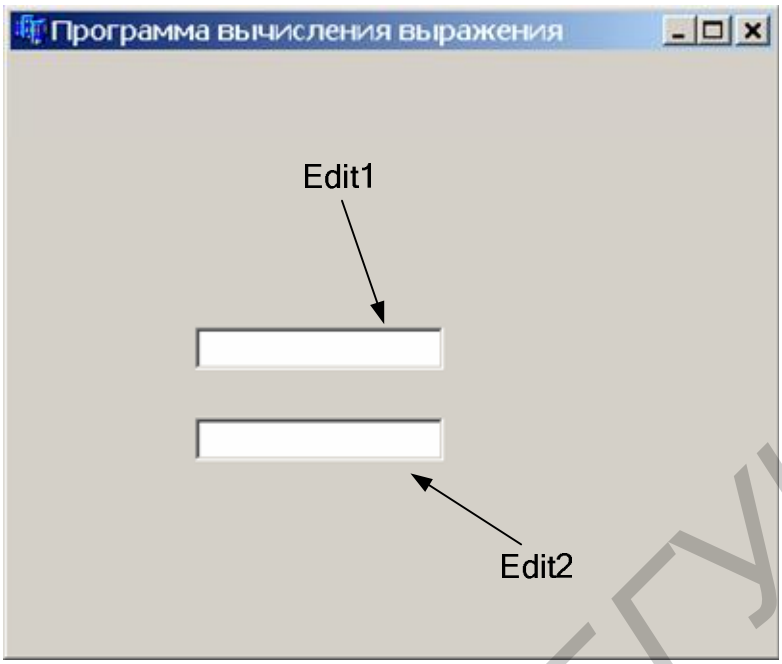

Рисунок 15 – Форма после настройки компонентов TEdit

Для вывода нередактируемого текста на поверхность формы обычно используют специальные поля для вывода текста типа *TLabel*. Значок компонента *TLabel* находится на вкладке **ToolPalette:** 

. Добавляется компонент *TLabel* в форму точно так же, как и редактируемое поле. Основные свойства компонента *TLabel* перечислены в таблице 3.

Если поле *Label* должно содержать несколько строк текста, то перед тем как ввести в поле текст (изменить значение свойства *Caption*), нужно присвоить свойству *AutoSize* значение **false**, а свойству *WordWrap* – **true**. Затем надо установить требуемый размер поля (при помощи мыши или вводом значений свойств *Height* и *Width*) и только после этого ввести значение свойства *Caption*.

Вид формы разрабатываемого приложения после добавления четырёх компонентов типа *TLabel* показан на рисунке 16. В данном случае поле *Label1* предназначено для вывода результата вычисления, поля *Label2* и *Label3* – для вывода информации об имени переменных, поле *Label4* – для вывода информации о назначении программы.

Следующий этап – это добавления компонента *TRadioButton*. Компонент *TRadioButton* создает круглую кнопку с двумя состояниями и описательным текстом, специфицирующим ее назначение. Такие кнопки представляют набор взаимоисключающих вариантов выбора: только одна кнопка может быть выбрана в данный момент времени (отмечается внутренним черным кружком), а с ранее выбранной кнопки выбор автоматически снимается. При нажатии радиокнопки свойство компоненты *Checked* меняется соответственно и возникает событие *OnClick*.

Таблица 3 – Свойства компонента Label

Наименование свойства	Значение свойства
Name	*Имя компонента*. Используется в программе для доступа к свойствам компонента
Caption	*Отображаемый текст*
Font	*Шрифт, используемый для отображения текста*
ParentFont	*Признак наследования шрифта родительского компонента*
AutoSize	*Признак того, что размер поля определяется его содержимым*
Left	*Расстояние от левой границы поля вывода до левой границы формы*
Top	*Расстояние от верхней границы поля вывода до верхней границы формы*
Height	*Высота поля вывода*
Width	*Ширина поля вывода*
WordWrap	*Перенос по словам*. Признак того, что слова, которые не помещаются в текущей строке, автоматически переносятся на следующую строку (значение свойства AutoSize должно быть false)

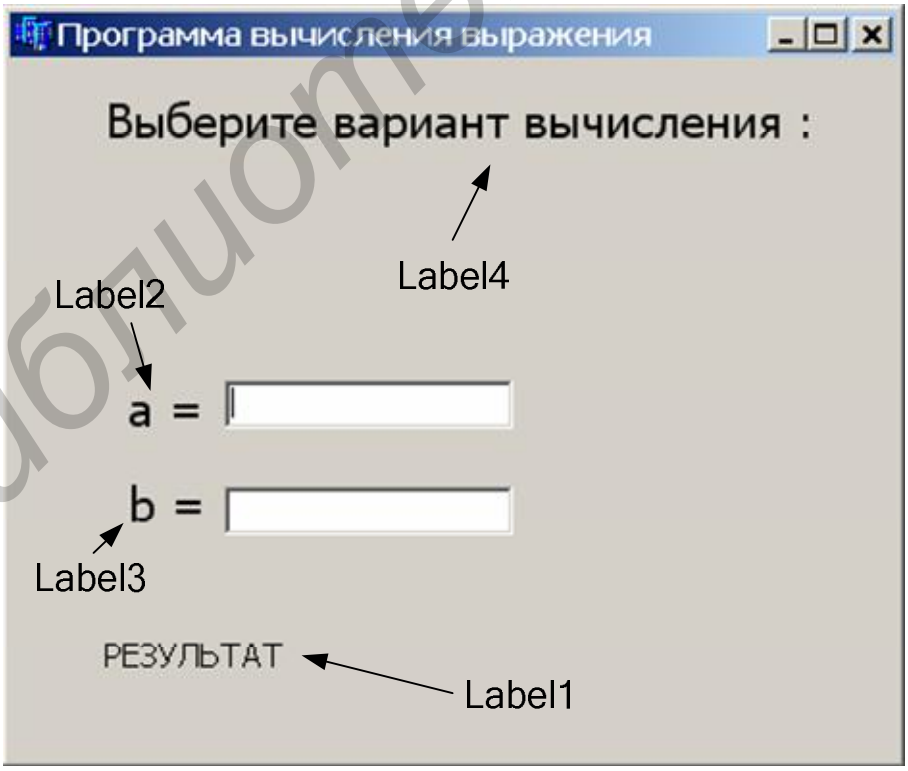

Рисунок 16 – Вид формы с установленными компонентами типа TLabel

Обычно радиокнопки размещаются внутри предварительно установленного на форме группового контейнера. Если выбрана одна кнопка, выбор всех прочих кнопок той же группе автоматически снимается. Например, две радиокнопки на форме могут быть выбраны одновременно только в том случае, когда они размещены в разных контейнерах. Если группировка радио-кнопок явно не задана, то по умолчанию все они группируются в одном из оконных контейнеров: **TForm**, **TGroupBox** или **TPanel**.

Компонент *RadioButton* добавляется в форму точно так же, как и другие компоненты. Значок компонента *Button* находится на вкладке **ToolPalette** 

.

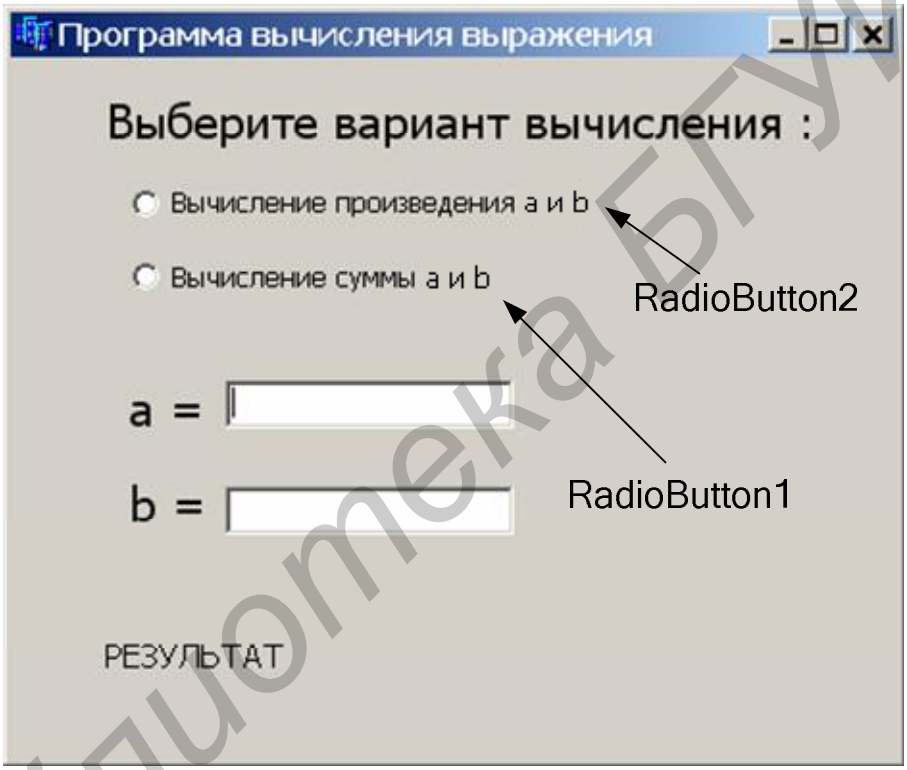

Рисунок 17 – Вид формы с установленными компонентами типа *TRadioButton*

Завершающий шаг – добавление на форму кнопки «ВЫЧИСЛИТЬ», по нажатии на которую запускается процесс вычисления необходимого выражения. Компонент типа *TButton* добавляется в форму точно так же, как и другие компоненты. Значок компонента *Button* находится в разделе **Standard** окна **ToolPalette:** 

. Основные свойства компонента *Button* приведены в таблице 4.

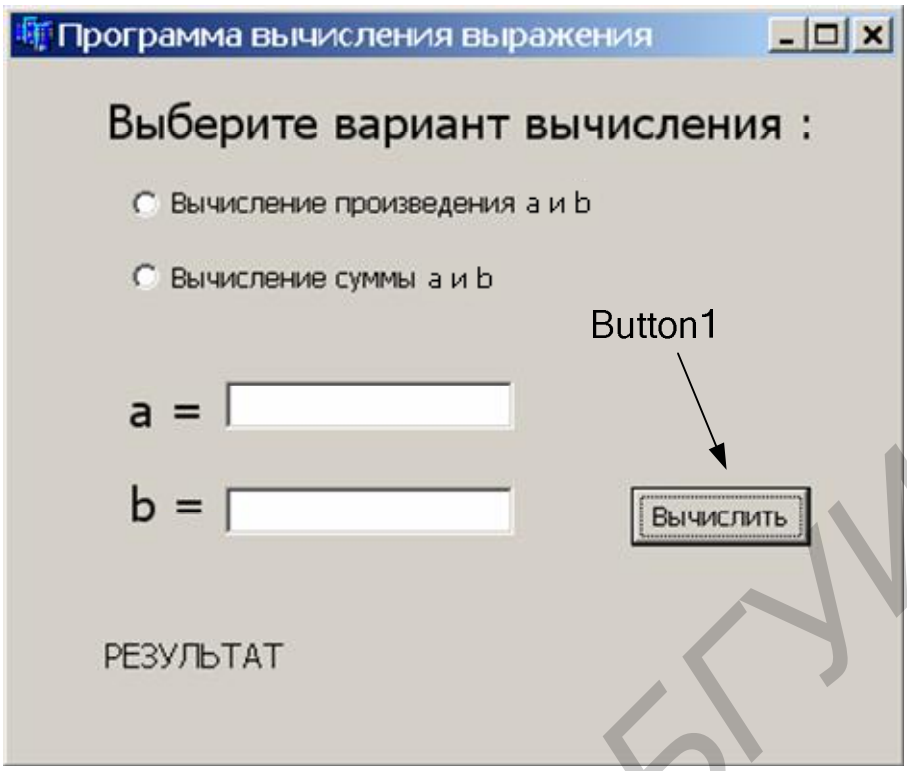

Рисунок 18 – Вид формы с установленным компонентом *TButton*

Таблица 4 – Свойства компонента Button (командная кнопка)

Наименование свойства	**Значение свойства**
Name	*Имя компонента*. Используется в программе для доступа к компоненту и его свойствам
Caption	*Текст на кнопке*. В рассматриваемом примере вводится текст «ВЫЧИСЛИТЬ»
Enabled	*Признак доступности кнопки*. Кнопка доступна, если значение свойства равно **true**, и недоступна, если значение свойства равно **false**
Left	*Расстояние от левой границы кнопки до левой границы формы*
Top	*Расстояние от верхней границы кнопки до верхней границы формы*
Height	*Высота кнопки*
Width	*Ширина кнопки*

Для того чтобы написать обработчик нажатия кнопки, необходимо, находясь на вкладке *Design* в конструкторе графического интерфейса, дважды нажать на объект *Button1* на форме. После этого откроется редактор кода в месте, предназначенном для ввода программного кода обработчика:

```
//---------------------------------------------------------
// Процедура обработки события
//---------------------------------------------------------
void __fastcall TForm1::Button1Click(TObject *Sender)
{
// Здесь необходимо написать код обработчика
}
```

Ниже приведён программный код для выполнения необходимых по условию задачи действий.

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
int a,b,rez=0;
a = StrToInt(Edit1->Text);
b = StrToInt(Edit2->Text);
/* Переключатели RadioButtonl и RadioButton2
   зависимые, поэтому о варианте вычисления можно
   судить по состоянию одного из них          */
if ( RadioButton1->Checked )
{
// выбран переключатель "сумма"
rez = a+b;
}
else
{
if ( RadioButton2->Checked )
{
// выбран переключатель "произведение"
rez = a*b;
}
else
ShowMessage("Необходимо выбрать вариант вычисления");
}
Label1->Caption = "РЕЗУЛЬТАТ: " + IntToStr(rez);
}
```

Окончательный вид формы разрабатываемого приложения приведен на рисунке 19. На рисунке 20 представлен пример выполнения программы.

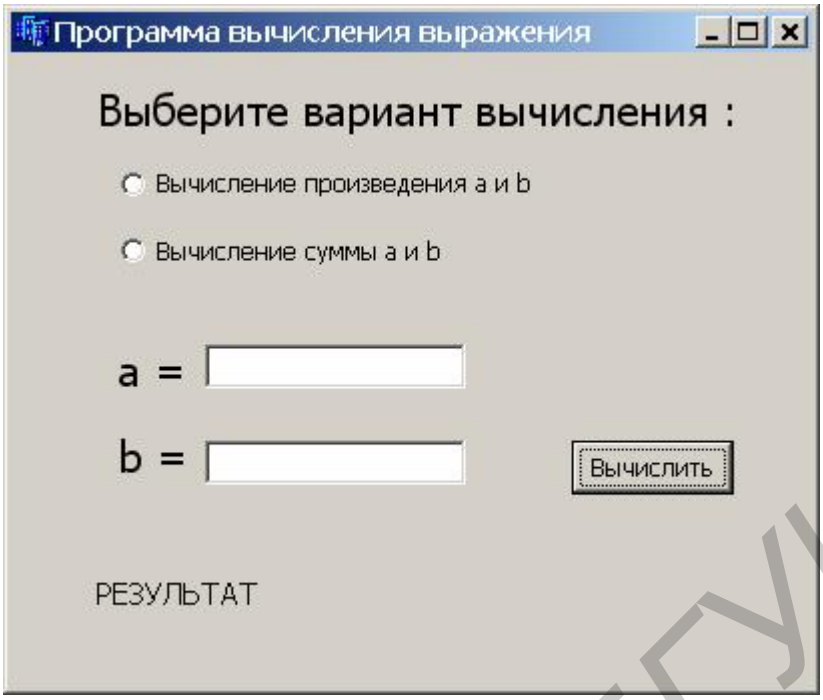

Рисунок 19 – Окончательный вид формы программы

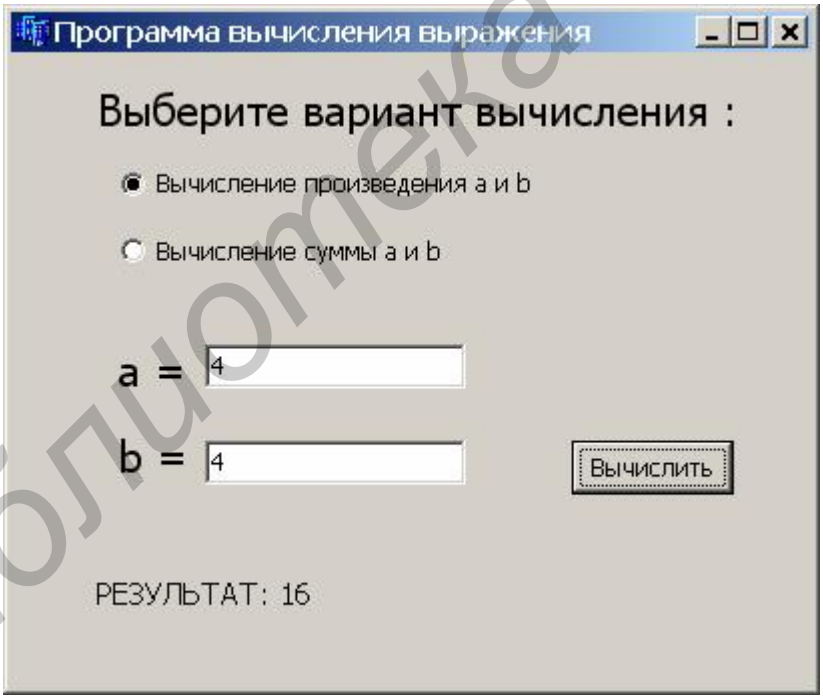

Рисунок 20 – Результат работы программы

3.4 Особенности использования редактора кода

Во время набора текста программы редактор кода автоматически выделяет элементы программы: полужирным шрифтом – ключевые слова языка программирования (**if**, **else**, **int**, **float** и др.), а курсивом выделяются комментарии. Это делает текст программы более выразительным, что облегчает восприятие

структуры программы. В процессе разработки проекта часто возникает необходимость переключения между окном редактора кода и окном формы. Сделать это можно при помощи командной кнопки **Toggle Form/Unit**, которая находится на панели инструментов *View* (рисунок 21), или нажатием клавиши <*F12*>. На панели инструментов *View* находятся командные кнопки **View Unit** и **View Form**, используя которые можно выбрать нужный модуль или форму в случае, если проект состоит из нескольких модулей или форм.

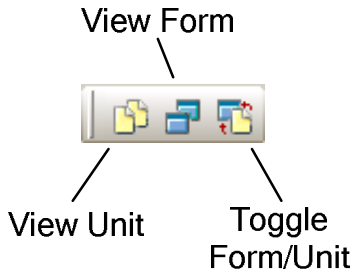

Рисунок 21 – Кнопки для быстрого переключения между формой и редактором кода

3.5 Система подсказок

Редактор кода поддерживает функцию контекстно-зависимой подсказки, которая во время набора текста программы автоматически выводит краткую справочную информацию о свойствах и методах объектов, о параметрах функций.

Например, после того как будет набрано имя объекта (компонента) и символы ->, редактор кода автоматически выведет список свойств и методов объекта (рисунок 22).

Программисту останется только выбрать из списка нужный элемент и нажать клавишу <*Enter*> (быстро перейти к нужному элементу списка или к области, где этот элемент находится, можно, нажав клавишу, соответствующую первому символу этого элемента).

Следует обратить внимание на то, что если список свойств и методов не появляется, то это значит, что в программе обнаружена ошибка , поскольку по умолчанию C++Builder контролирует правильность набираемого программистом текста в «фоновом» режиме.

После набора имени встроенной или объявленной программистом функции редактор кода также выводит подсказку: список параметров. Параметр, который в данный момент вводит программист, в подсказке выделен полужирным шрифтом. Например, если набрать слово **FloatToStr**, которое является именем функции преобразования дробного числа в строку символов, и открывающую скобку, то на экране появится окно, в котором будет указан список параметров функции.

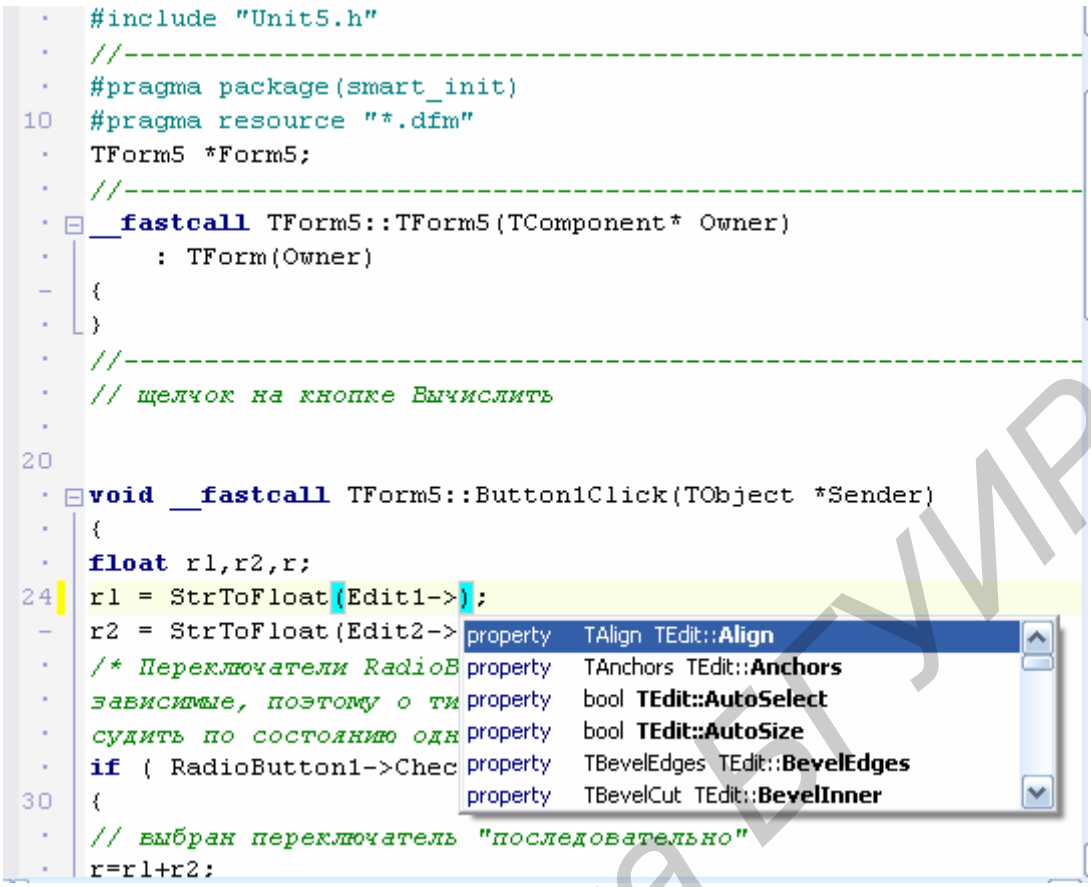

Рисунок 22 – Демонстрация работы системы подсказок в редакторе кода (автоматически выводится список свойств и методов указанного объекта)

3.6 Особенности использования справочной системы

В процессе набора текста программы можно получить справку о конструкции языка, типе данных, классе или функции. Для этого нужно в окне редактора кода набрать слово, о котором необходимо получить справку (например имя функции), и нажать клавишу <*F1*>. Так как с запрашиваемой темой в справочной системе может быть связано несколько разделов, на экране, как правило, появляется окно **Topics Found** (Найденные разделы), в котором можно выбрать нужный раздел. Следует обратить внимание на то, что после имени функции может быть указано имя библиотеки, к которой эта функция относится: **VCL** или **CLX** (библиотека VCL используется при разработке приложений для Windows, а CLX – при разработке кроссплатформенных приложений). Поэтому, выбирая раздел справочной системы, надо обращать внимание на то, к какой библиотеке он относится.

Справочную информацию можно получить также, выбрав из меню *Help* команду **Borland Help**. В этом случае на экране появится стандартное окно справочной системы. В этом окне на вкладке **Index** (Предметный указатель) в поле **Look for** нужно ввести ключевое слово, определяющее тему, по которой нужна справка. В качестве ключевого слова можно ввести, например, первые несколько букв имени функции, свойства или метода. Для того чтобы уточнить

область поиска, можно в поле **Filtered by** дополнительно указать нужный раздел, например *Developer Studio 2006 for Win32*.

3.7 Запуск программы

Пробный запуск программы можно выполнить непосредственно из среды разработки, не завершая работу с C++ Builder. Для этого нужно в меню ***Run*** выбрать команду **Run** или щелкнуть на командной кнопке **Run** на панели инструментов.

3.8 Компиляция программы и поиск ошибок

Процесс преобразования исходной программы в выполняемую состоит из двух этапов: непосредственно компиляции и компоновки. На этапе компиляции выполняется перевод исходной программы в некоторое внутреннее представление. На этапе компоновки выполняется сборка (построение) программы.

После ввода текста функции обработки события и сохранения проекта можно, выбрав в меню *Project* команду **Compile**, выполнить компиляцию. Процесс и результат компиляции отражается в диалоговом окне *Compiling* (рисунок 23). Если в программе нет синтаксических ошибок, то окно будет содержать сообщение: *Done: Compile Unit*, в противном случае будет выведено сообщение *Done: There are errors.*

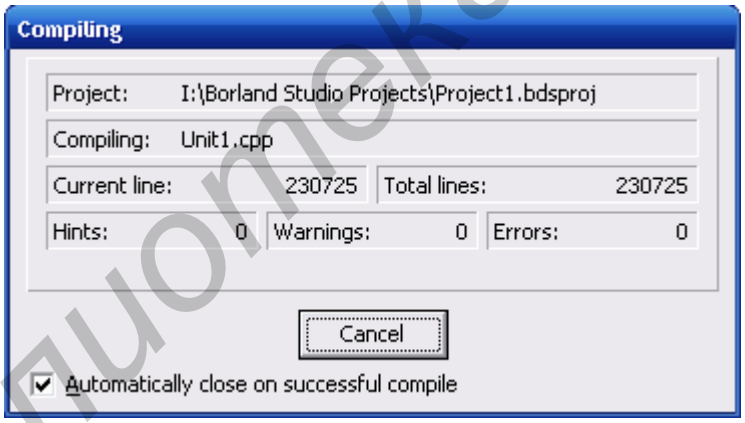

Рисунок 23 – Окно **Compiling**

В случае если компилятор обнаружит в программе ошибки и неточности, диалоговое окно **Compiling** будет содержать информацию о количестве синтаксических (*Errors*) и семантических (*Warnings*) ошибок, а также о числе подсказок (*Hints*). Сами сообщения об ошибках, предупреждения и подсказки находятся в нижней части окна редактора кода.

Компилятор переходит ко второму этапу генерации выполняемой программы только в том случае, если исходный текст не содержит синтаксических ошибок. В большинстве случаев в только что набранной программе есть такие ошибки и программист должен их устранить. Процесс устранения ошибок но-

сит итерационный характер. Обычно сначала устраняются наиболее очевидные ошибки. После очередного внесения изменений в текст программы выполняется повторная компиляция.

Следует обратить внимание на то, что компилятор не всегда может точно локализовать ошибку. Поэтому, анализируя фрагмент программы, который, по мнению компилятора, содержит ошибку, **нужно обратить внимание не только на тот фрагмент кода, на который компилятор установил курсор, но и на тот, который находится выше**.

В таблице 5 перечислены типичные ошибки и соответствующие им сообщения компилятора.

Таблица 5 – Типичные ошибки

Сообщение	Описание ошибки
Undefined symbol (неизвестный символ)	Используется необъявленная переменная, функция или параметры записаны неверно. Например, в программе объявлена переменная Summ, а в инструкциях используется sum
Statement missing (отсутствует точка с запятой)	После инструкции не поставлена точка с запятой
Unterminated string or character constant (незаконченная строковая или символьная константа)	В конце строковой константы, например, текста сообщения, нет двойных кавычек
) expected (ожидается закрывающая скобка)	При записи арифметического выражения, содержащего скобки, нарушен баланс открывающих и закрывающих скобок
if statement missing ((в инструкции **if** нет открывающей скобки)	В инструкции **if** условие не заключено в скобки
Compound statement missing }	Нарушен баланс открывающих и закрывающих фигурных скобок. Вероятно, не поставлена закрывающая фигурная скобка, отмечающая конец функции или группы инструкций, например, после условия или слова **else** в инструкции **if**

Если компилятор обнаружил достаточно много ошибок, то необходимо просмотреть все сообщения и устранить сначала наиболее очевидные ошибки и уже затем выполнять повторную компиляцию.

3.9 Сохранение файлов проекта

Проект – это набор файлов, используя которые компилятор создает исполняемый файл программы (**exe**-файл). В простейшем случае проект составляют: файл описания проекта (**bdsproj** -файл, например Project1.bdsproj), файл главного модуля (**cpp**-файл, например Project1.cpp), файл ресурсов (**res**-файл, например Project1.res), файл описания формы (**dfm**-файл, например Unit1.dfm), заголовочный файл формы (**h**-файл, например Unit1.h) и файл описания функций формы (**cpp**-файл, например Unit1.cpp).

Чтобы сохранить проект, нужно в меню *File* выбрать команду **Save Project As**. Если проект еще ни разу не был сохранен, то C++ Builder сначала предлагает сохранить модуль (содержимое окна редактора кода), и поэтому на экране появляется окно **Save Unit1 As**. В этом окне надо выбрать папку, предназначенную для проектов, создать в ней папку для сохраняемого проекта, открыть ее и ввести имя модуля. В результате щелчка на кнопке **OK** в указанной папке будут созданы три файла: **cpp**, **h** и **dfm**, и на экране появится диалоговое окно **Save Projectl As**, в которое надо ввести имя проекта. Обратите внимание, что имена файла модуля (**cpp**) и файла проекта (**bdsproj**) должны быть разными, т.к. C++Builder в момент сохранения файла проекта создает одноименный **cpp**-файл (файл главного модуля). Кроме того, надо учесть, что имя генерируемого компилятором выполняемого файла совпадает с именем проекта.

4 Назначения и особенности использования некоторых компонентов из комплекта поставки C++ Builder 2006

4.1 Компонент TMemo

Данный компонент отображает прямоугольную область редактируемого ввода множественных строк информации на форме. Начальное содержимое области редактирования определяет массив строк, являющийся значением свойства *Lines*. Окно редактора элементов списка открывается кнопкой в графе значений этого свойства. В таблице 6 перечислены некоторые свойства компонента **TMemo**, а внешний вид представлен на рисунке 24.

4.2 Компонент TStringGrid

Компонент TStringGrid представляет собой таблицу, ячейки которой содержат строки символов. В таблице 7 перечислены некоторые свойства компонента TStringGrid, а внешний вид представлен на рисунке 25.

Таблица 6 – Основные свойства компонента TMemo

Наименование свойства	Значение свойства
Name	*Имя компонента*. Используется в программе для доступа к свойствам компонента
Text	*Текст, находящийся в поле Memo*. Рассматривается как единое целое
Lines	*Текст, находящийся в поле Memo*. Рассматривается как совокупность строк. Доступ к строке осуществляется по номеру
Lines.Count	*Количество строк текста в поле Memo*
Left	*Расстояние от левой границы поля до левой границы формы*
Top	*Расстояние от верхней границы поля до верхней границы формы*
Height	*Высота поля*
Width	*Ширина поля*
Font	*Шрифт, используемый для отображения вводимого текста*
ParentFont	*Признак наследования свойств шрифта родительской формы*

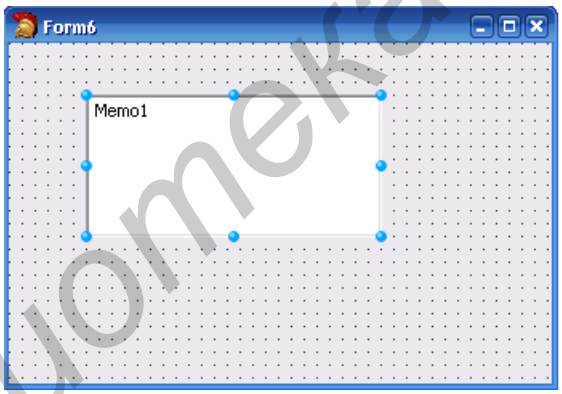

Рисунок 24 – Внешний вид компонента *TMemo*

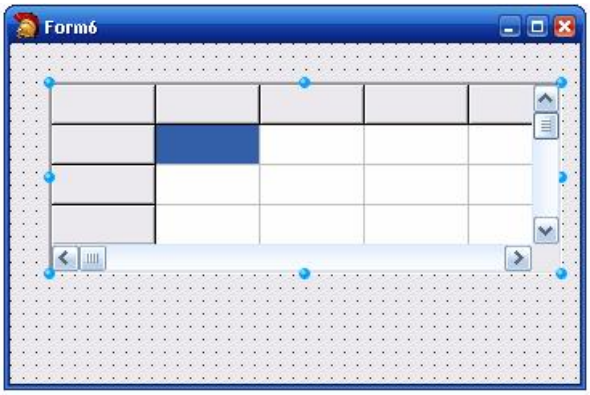

Рисунок 25 – Внешний вид компонента *TStringGrid*

Таблица 7 – Основные свойства компонента TStringGrid

Наименование свойства	Значение свойства
Name	*Имя компонента*. Используется в программе для доступа к свойствам компонента
ColCount	*Количество колонок таблицы*
RowCount	*Количество строк таблицы*
DefaultColWidth	*Ширина колонок таблицы*
Cells	*Соответствующий таблице двумерный массив*. Ячейка таблицы, находящаяся на пересечении столбца номер **col** и строки номер **row,** определяется элементом **cells [col, row]**
FixedCols	*Количество зафиксированных слева колонок таблицы*. Зафиксированные колонки выделяются цветом и при горизонтальной прокрутке таблицы остаются на месте
FixedRows	*Количество зафиксированных сверху строк таблицы*. Зафиксированные строки выделяются цветом и при вертикальной прокрутке таблицы остаются на месте
Options.goEditing	*Признак допустимости редактирования содержимого ячеек таблицы*. True – редактирование разрешено, False – запрещено
Options . goTab	*Разрешает (True) или запрещает (False) использование клавиши <Tab> для перемещения курсора в следующую ячейку таблицы*
Options.GoAlways-ShowEditor	*Признак нахождения компонента в режиме редактирования*. Если значение свойства False, то для того чтобы в ячейке появился курсор, надо начать набирать текст, нажать клавишу <F2> или сделать щелчок мышью
DefaultRowHeight	*Высота строк таблицы*
GridLineWidth	*Ширина линий, ограничивающих ячейки таблицы*
Left	*Расстояние от левой границы поля таблицы до левой границы формы*
Top	*Расстояние от верхней границы поля таблицы до верхней границы формы*
Height	*Высота поля таблицы*
Width	*Ширина поля таблицы*
Font	*Шрифт, используемый для отображения содержимого ячеек таблицы*
ParentFont	*Признак наследования характеристик шрифта формы*

4.3 Компонент TListBox

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

```
ListBoxl->Items->Add("Демонстрация работы компонента
ListBox");
```

Значение *true* свойства *Sorted* устанавливает сортировку элементов списка по алфавиту.

В таблице 8 перечислены некоторые свойства компонента **TListBox**, а внешний вид представлен на рисунке 26.

Таблица 8 – Свойства компонента ListBox

Наименование свойства	**Значение свойства**
Items	Тип TStrings содержит список строк
MultiSelect	*Разрешает или запрещает множественный выбор*
Sorted	*Упорядочивает список по алфавиту*
AutoComplete	*Поиск строки при нажатии на первые символы строк*
ExtendedSelect	Если true, то при множественном выборе его можно осуществлять с удерживанием кнопок Shift и Ctrl
Columns	*Количество столбцов, в которых отображается список*
ItemIndex	Доступно во время выполнения, если MultiSelect=false – это индекс выбранной строки. Если ни одна строка не выбрана – минус 1

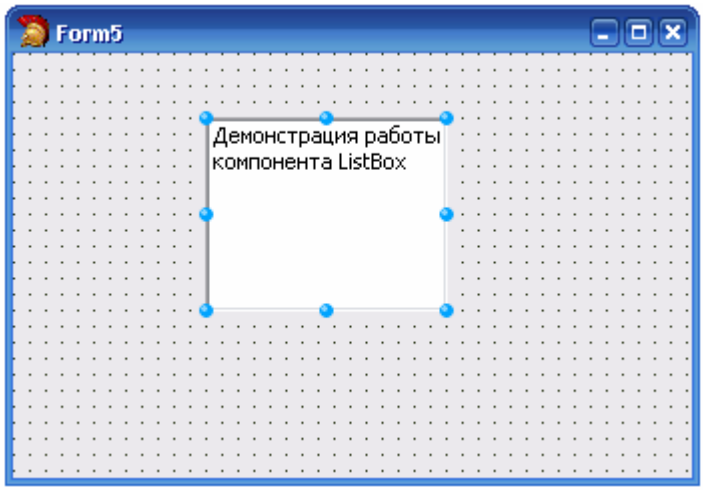

Рисунок 26 – Компонент *TListBox*

4.4 Компонент TGroupBox

Создает контейнер в виде прямоугольной рамки, визуально объединяющий на форме логически связанную группу некоторых интерфейсных элементов. Эта компонента представляет собой инкапсуляцию одноименного объекта Windows.

4.5 Компонент TRadioGroup

Создает контейнер в виде прямоугольной рамки, визуально объединяющий на форме группу логически взаимоисключающих радиокнопок.

Радиокнопки «группируются» при помещении их в один и тот же контейнер. Только одна кнопка из данной группы может быть выбрана. Добавление кнопок к компоненте **TRadioGroup** выполняется редактированием свойства *Items*. Присвоение названия очередной строке свойства *Items* приводит к появлению этой кнопки в группирующей рамке. Значение свойства *ItemIndex* определяет, какая радиокнопка выбрана в настоящий момент. Возможна группировка радиокнопок в несколько столбцов с установлением соответствующего значения свойства *Columns*.

4.6 Компонент TPanel

Создает пустую панель, которая может содержать другие компоненты. Возможно использование **TPanel** для создания на форме панелей инструментов или строк состояния.

5 Обработка событий в C++Builder

В C++Builder каждому событию присвоено имя. Например, щелчок кнопкой мыши – это событие *OnClick*, двойной щелчок мышью – событие *OnDblClick*. В таблице 9 приведены некоторые события, обрабатываемые C++Builder.

Реакцией на событие должно быть какое-либо действие. В C++ Builder реакция на событие реализуется как функция обработки события. Под термином «обработка события» в данном случае понимается реализация набора заранее определённых действий, которые происходят во время работы программы в ответ на возникновение того или иного события.

Таблица 9 – События

Название события	**Условия возникновения события**
OnClick	При щелчке кнопкой мыши
OnDblClick	При двойном щелчке кнопкой мыши
OnMouseDown	При нажатии кнопки мыши
OnMouseUp	При отпускании кнопки мыши
OnMouseMove	При перемещении мыши
OnKeyPress	При нажатии клавиши клавиатуры
OnKeyDown	При нажатии клавиши клавиатуры. События *OnKeyDown* и *OnKeyPress* – это чередующиеся, повторяющиеся события, которые происходят до тех пор, пока не будет отпущена удерживаемая клавиша

Таким образом, для того чтобы программа выполняла некоторую работу в ответ на действия пользователя, программист должен написать функцию обработки соответствующего события.

Пример: По нажатии на клавишу с кодом 32 (пробел) программа закрывает форму. На рисунке 27 представлена форма программы для рассматриваемого примера.

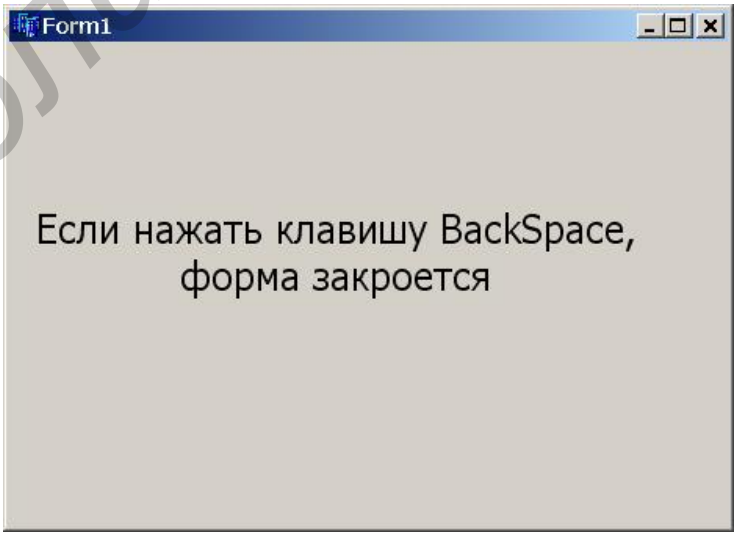

Рисунок 27 – Форма программы для примера обработки события

Чтобы приступить к созданию функции обработки события, сначала необходимо выбрать компонент, для которого создается функция обработки события (в данном случае для формы **Form1**). Выбрать компонент можно в окне **Object Inspector** или щелчком на изображении компонента в форме. После этого в окне **Object Inspector** нужно выбрать вкладку *Events* (События). В левой колонке вкладки *Events* перечислены события, которые может воспринимать выбранный компонент (имя и тип компонента указаны в верхней части окна), рисунок 28.

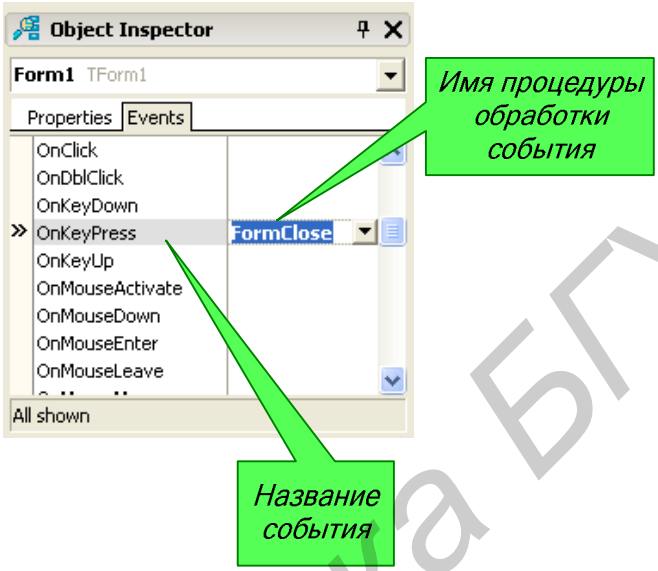

Рисунок 28 – Вкладка Events окна Object Inspector

Если для события определена функция обработки, то в правой колонке рядом с именем события будет выведено имя этой функции. Для того чтобы создать функцию обработки события, нужно сделать двойной щелчок мышью в окне **Object Inspector**, в поле функции обработки соответствующего события. В результате этого откроется окно редактора кода, в которое будет добавлен шаблон функции обработки события, а в окне **Object Inspector** рядом с именем события появится сгенерированное C++Builder имя функции обработки события.

Ниже представлен код обработчика события.

```
//---------------------------------------------------------------
// программный код функции обработки события
//---------------------------------------------------------------

void __fastcall TForm1::FormClose(TObject *Sender, char &Key)
{
 if (Key==32)

{
```

```
        Form1->Close();
 }
}
//---------------------------------------------------------------------
```

Как видно из приведённого кода, функция обработки события в данном случае называется **FormClose**. С помощью параметра *Key* в эту функцию передаётся код нажатой клавиши.

Работа с другими событиями C++Builder производится аналогичным образом. Однако при этом следует учитывать, что набор и вид параметров будут индивидуальными для каждого события. Информацию о конкретном событии можно оперативно найти в справке системы C++Builder.

6 Особенности использования ассемблерных вставок и дизассемблера в C++ Builder 2006

Применение ассемблера совместно с языками высокого уровня (в данном случае совместно с C/C++) позволяет эффективно решать следующие задачи:

- оптимизация отдельных частей программного кода, от которых в значительной степени зависит производительность программы в целом;
- работать с оборудованием компьютера на аппаратном уровне, учитывая специфические особенности и возможности различных устройств, что особенно важно при использовании нестандартного оборудования;
- создавать и модифицировать процедуры обработки прерываний.

C++ Builder предлагает три способа использования ассемблера совместно с программами, написанными на C/C++:

- компиляции программы в файл с ассемблерным текстом;
- использование так называемых ассемблерных вставок в текст программы с помощью операторов **asm**;
- написание внешних подключаемых модулей, ассемблируемых с помощью Turbo Assembler (TASM).

Наиболее распространенным, практически используемым и удобным для программиста способом совмещения программного кода, написанного на языке высокого уровня, с ассемблером является использование вставок. Ассемблерная вставка оформляется в тексте программы с помощью ключевого слова **asm**, за которым ставится фигурная открывающая скобка **{**. Далее идёт необходимый программный код на ассемблере и закрывающая скобка **}**.

Ниже в качестве примера показан программный код разработанной выше программы с использованием ассемблерных вставок для вычисления суммы либо произведения целых чисел.

```
//---------------------------------------------------------
void __fastcall TForm1::Button1Click(TObject *Sender)
{
int a,b,rez=0;
a = StrToInt(Edit1->Text);
b = StrToInt(Edit2->Text);
/* Переключатели RadioButtonl и RadioButton2
   зависимые, поэтому о варианте вычисления можно
   судить по состоянию одного из них            */
if ( RadioButton1->Checked )
{
// выбран переключатель "сумма"
// вычисление rez = a+b; на ассемблере
asm
{
mov eax,a;    // загрузка в регистр eax значения переменной a
mov edx,b;    // загрузка в регистр edx значения переменной b
add edx,eax;  // edx <-- edx+eax
mov rez,edx;  // rez <-- edx
}
}
else
{
if ( RadioButton2->Checked )
{
// выбран переключатель "произведение"
// вычисление rez = a*b; на ассемблере
asm
{
mov eax,a;    // загрузка в регистр eax значения переменной a
mov edx,b;    // загрузка в регистр edx значения переменной b
imul edx,eax; // edx <-- edx*eax
mov rez,edx;  // rez <-- edx
}
}
else
ShowMessage("Необходимо выбрать вариант вычисления");
}
Label1->Caption = "РЕЗУЛЬТАТ: " + IntToStr(rez);
}
```


Встроенный дизассемблер помогает программисту отлаживать код программы, следить за состоянием регистров процессора в процессе выполнения программы и вместе с другими средствами позволяет локализовать возможные низкоуровневые программные ошибки.

Для просмотра нужного участка ассемблерного кода в окне дизассемблера **CPU** нужно выполнить следующие действия:

- поставить точку остановки на первую команду в интересующем блоке, для чего можно просто щёлкнуть левой кнопкой мыши на сером поле слева от команды, рисунок 29;

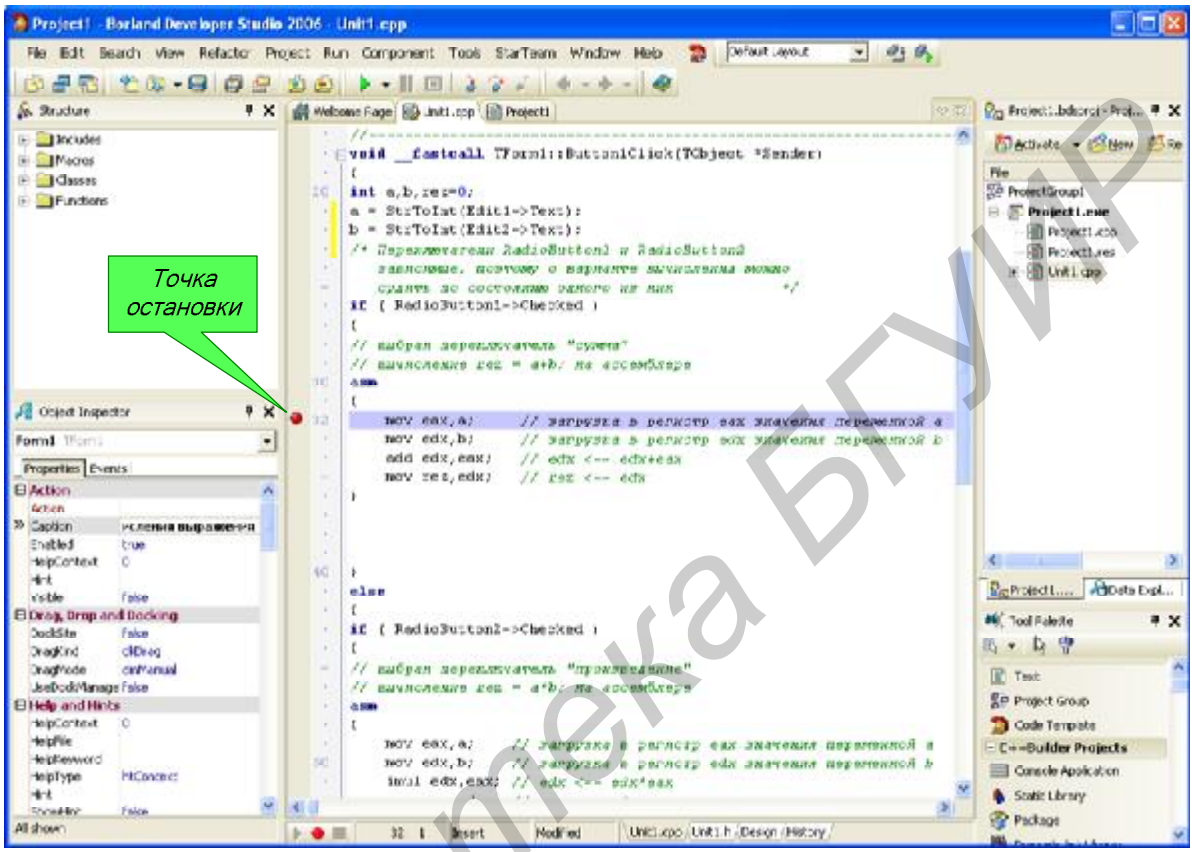

Рисунок 29 – Вид окна редактора кода после установки точки остановки

- выполнить программу до точки остановки с помощью команды **Run** или нажав кнопку **F9**, рисунок 30;
- вызвать окно **CPU** через меню **View / Debug Windows / CPU** или нажатием комбинацию клавиш **ctrl+alt+c**, рисунок 31.

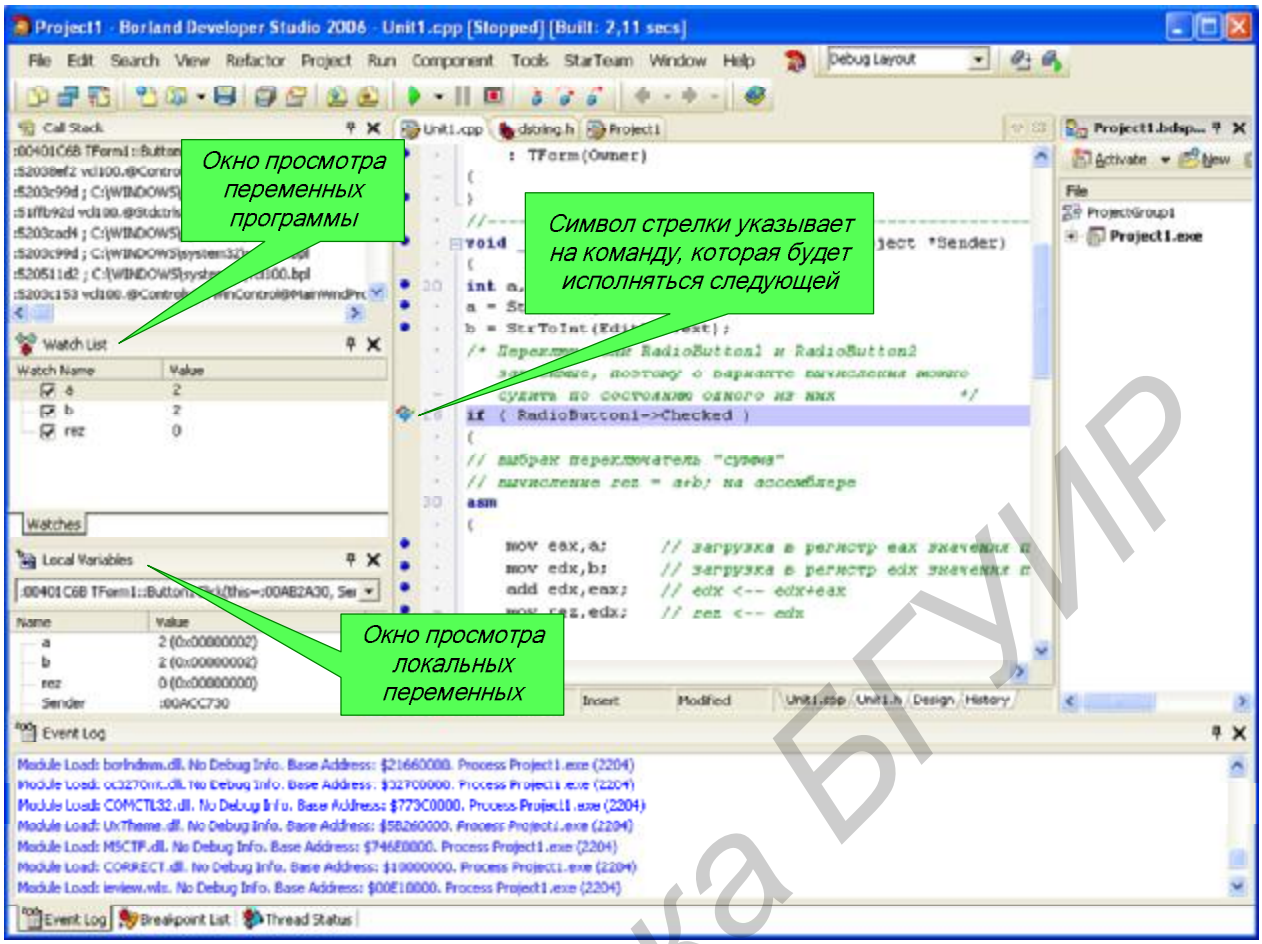

Рисунок 30 – Вид окна редактора кода после выполнения программы до точки остановки

Для выполнения программы в пошаговом режиме необходимо нажимать кнопку **F7** (выполнение следующей одиночной команды с входом в подпрограмму, цикл или блок команд) или кнопку **F8** (выполнение следующей команды, блока команд или подпрограммы).

В процессе пошаговой отладки программы в окне **CPU** можно просматривать регистры процессора, значения флагов и анализировать содержимое памяти по определённым адресам, рисунок 32.

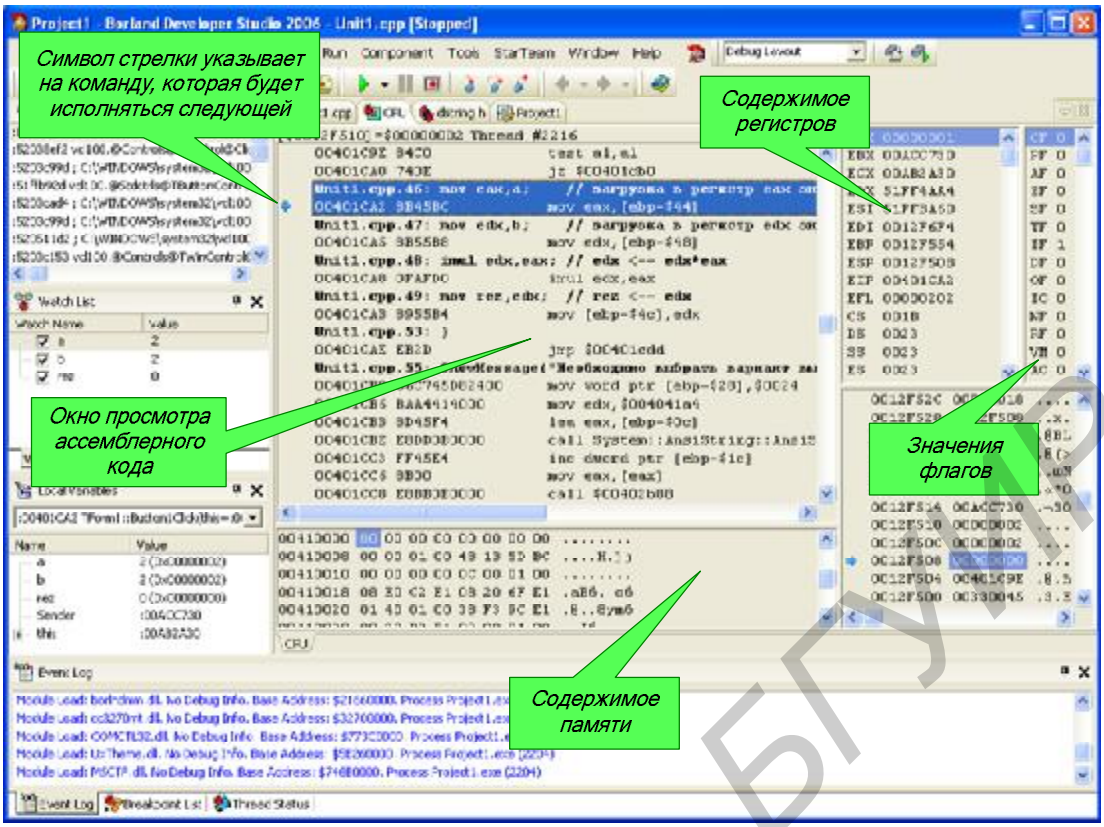

Рисунок 31 – Вид окна **CPU**

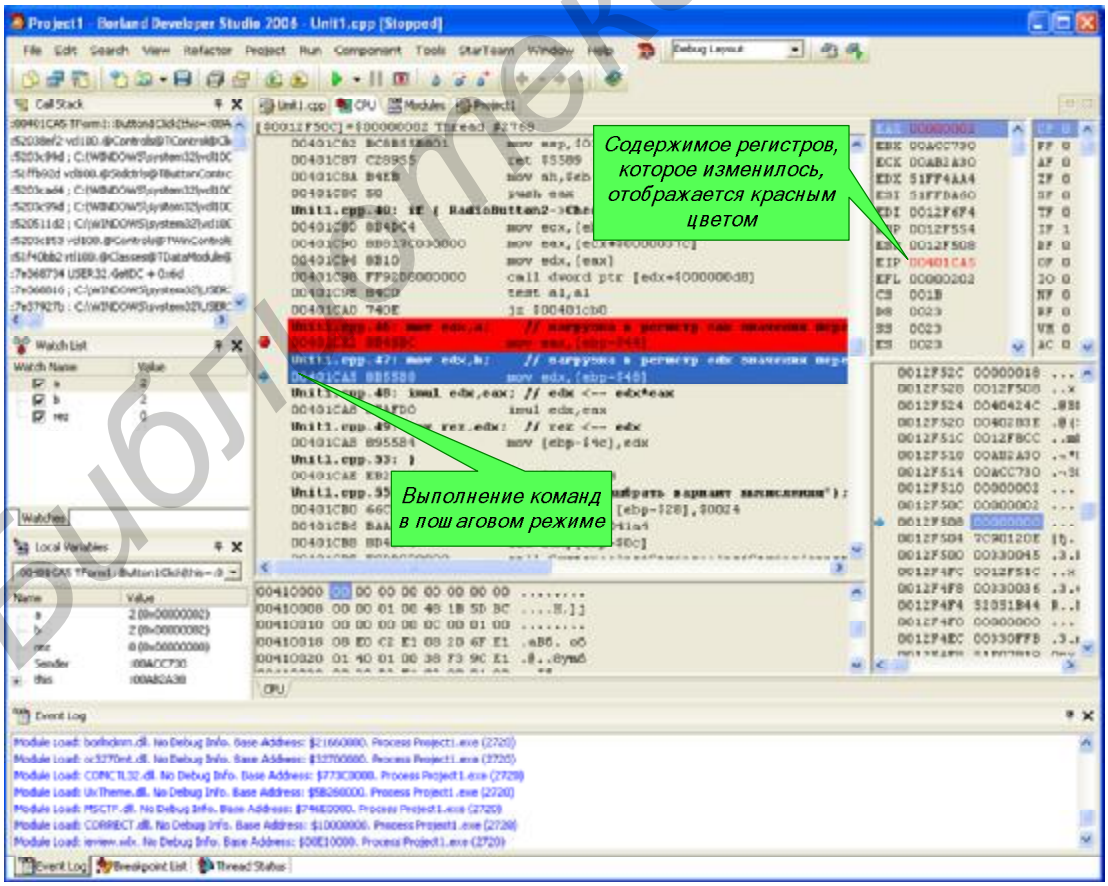

Рисунок 32 – Вид окна **CPU** при выполнении команд в пошаговом режиме

Лабораторная работа №1

Цель работы

Получить практические навыки по использованию встроенного дизассемблера Borland C++Builder.

Порядок выполнения работы

1 Разработать приложение с графическим пользовательским интерфейсом в среде Borland C++ Builder, которое будет выполнять действия над массивами целых чисел типа **int** в соответствии с вариантом:

№ варианта	Выражение
1	`d[i]=a[i]*b[i]-c[i]/2`
2	`d[i]=2/(a[i]*b[i])+c[i]`
3	`d[i]=a[i]/b[i]-2*c[i]`
4	`d[i]=a[i]*b[i]/c[i]-5`
5	`d[i]=a[i]/b[i]-c[i]+10`
6	`d[i]=a[i]*2-b[i]/c[i]`

2 Запустить полученное приложение в *debug*-режиме и выполнить его пошагово. Ознакомиться с возможностями встроенного дизассемблера (для просмотра кода в окне дизассемблера необходимо поставить точку остановки на изучаемую команду, а затем нажать комбинацию клавиш **alt+ctrl+c** и пошагового исполнять программу, нажимая клавишу **F8**).

3 Разобраться с ассемблерным кодом, полученным в результате компиляции приложения.

4 Скомпилировать то же приложение с различными опциями оптимизации компилятора. Для выбора опций оптимизации компилятора необходимо выбрать из меню команду **Project / Options**, в открывшемся окне выбрать раздел **C++Compiler (bcc32)**, а затем – раздел **Optimizations**, рисунок 33.

5 Проанализировать разные варианты полученного ассемблерного кода. Найти отличия.

6 Продемонстрировать результат работы преподавателю.

7 Согласовать с преподавателем содержание отчёта и подготовить его.

8 Защитить работу.

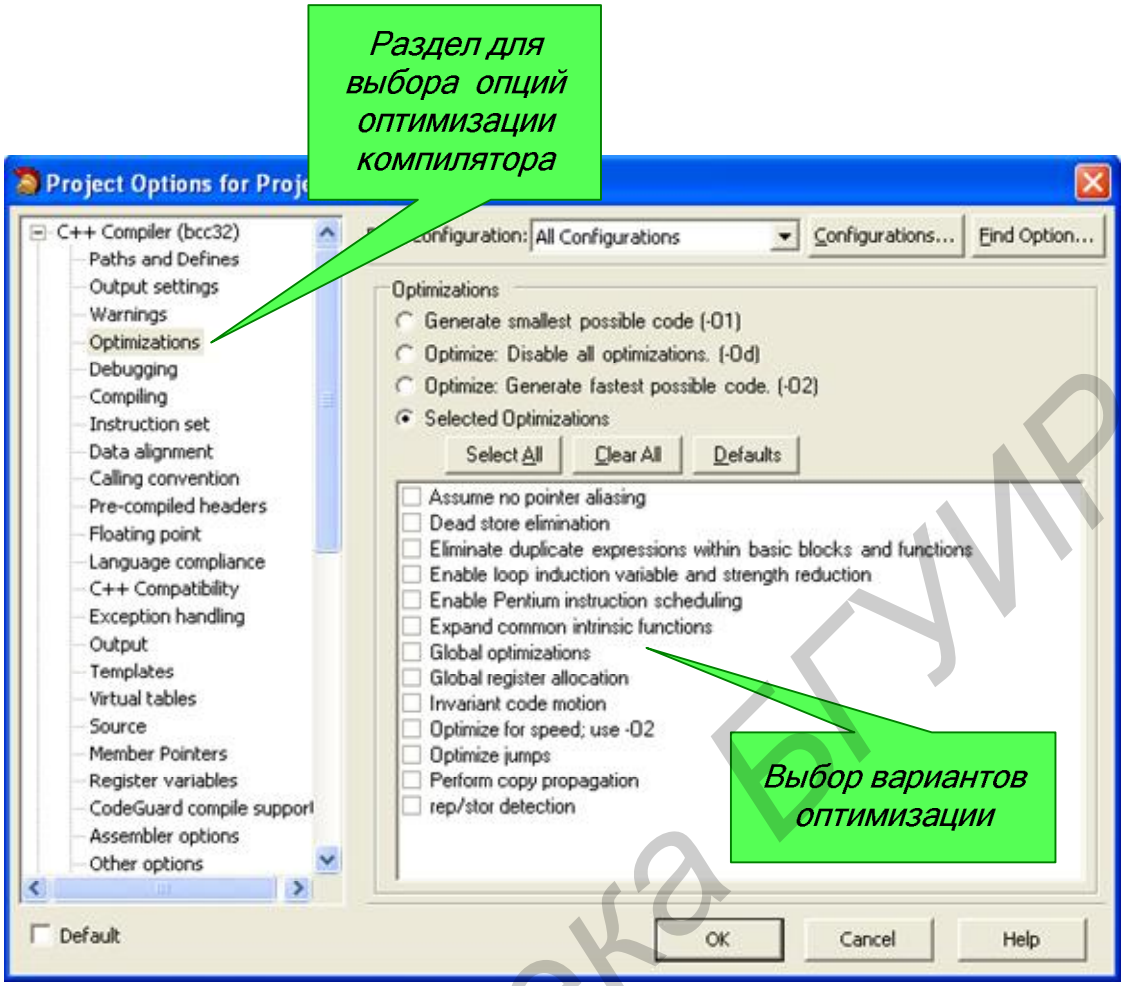

Рисунок 33 – Вид окна для настройки компилятора

Лабораторная работа №2

Цель работы

Получить практические навыки по использованию ассемблерных вставок в среде Borland C++Builder.

Порядок выполнения работы

1 Разработать приложение с графическим пользовательским интерфейсом в среде Borland C++ Builder, которое будет выполнять действия над целыми числами *x*, *y* и *R* в соответствии с выражением*:*

№ варианта	Выражение
1	$R = y \cdot x + 2x - 4y$
2	$R = 2y / x + 5x - 3y$
3	$R = y \cdot x + 2x - 4y$
4	$R = y + y / 2 + x^2$
5	$R = 2xy + x - 3y$
6	$R = y / 4 + x / 5 - xy$

Программный код, который производит вычисления по выражению, оформить в виде *ассемблерной вставки* в среде Borland C++Builder. Значения переменных отобразить на форме с помощью соответствующих компонентов.

2 Продемонстрировать результат работы преподавателю.

3 Согласовать с преподавателем содержание отчёта и подготовить его.

4 Защитить работу.

Литература

1 Щупак, Ю. А. Win32 API. Эффективная разработка приложений / Щупак, Ю. А. – СПб. : Питер, 2006.

2 Дейтел, Х. Как программировать на C++: 3-е изд. / Х. Дейтел, П. Дейтел // пер. с англ. – М. : Издательство БИНОМ, 2003.

3 Лаптев, В. В. C++. Экспресс-курс / В. В. Лаптев – СПб. : БХВ-Петербург, 2004. – 512 с. : ил.

4 Культин, Н. Б. Самоучитель C++ Builder / Н. Б. Культин. – СПб. : БХВ-Петербург, 2004.

5 Юров, В. И. Assembler : учеб. для вузов. 2-е изд. / В. И. Юров – СПб. : Питер, 2003. – 637 с. : ил.

6 Рихтер, Дж. Windows для профессионалов. Создание эффективных Win32-приложений с учетом специфики 64-разрядной версии Windows / Дж. Рихтер // пер.с англ. – 4-е изд. – СПб. : Питер; М. : Издательско-торговый дом «Русская Редакция», 2004.

Содержание

Св. план 2009, поз. 17

Учебное издание

Лихачев Денис Сергеевич
Лившиц Михаил Зенадьевич

Интегрированная среда разработки Borland Builder C++

Методическое пособие

по курсу

«Системное программирование»

для студентов специальности 1-40 02 02
«Электронные вычислительные средства» дневной формы обучения

Редактор Т. П. Андрейченко
Корректор Е. Н. Батурчик

Подписано в печать 17.12.2008.	Формат 60x84 1/16.	Бумага офсетная.
Гарнитура «Таймс»	Печать ризографическая.	Усл. печ. л. 2,44.
Уч.- изд. л. 2,3.	Тираж 100 экз.	Заказ 610.

Издатель и полиграфическое исполнение: Учреждение образования
«Белорусский государственный университет информатики и радиоэлектроники»
ЛИ №02330/0056964 от 01.04.2004. ЛП №02330/0131666 от 30.04.2004
220013, Минск, П. Бровки, 6