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

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

Лабораторная работа №2

«Динамическое создание объектов в интегрированной среде разработке *Borland C++ Builder* на примере приложения «Калькулятор»»

по дисциплине

«**Персональные компьютеры в инженерной практике**»

для студентов дневной и заочной формы обучения

по направлению 6.050901 — Радиотехника.

Севастополь
2013 г.

УДК.519.72

Методические указания «Лабораторная работа №2 Динамическое создание объектов в интегрированной среде разработке *Borland C++ Builder* на примере приложения «Калькулятор»» по дисциплине «Персональные компьютеры в инженерной практике» для студентов дневной и заочной форм обучения по направлению 6.050901 — Радиотехника / Сост. А.В. Лукьянчиков — Севастополь: Изд-во СевНТУ, 2013. — 14 с.

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

Допущено учебно-методическим центром СевНТУ в качестве методических указаний.

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

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

2. ЛАБОРАТОРНАЯ РАБОТА № 2
ДИНАМИЧЕСКОЕ СОЗДАНИЕ ОБЪЕКТОВ В ИНТЕГРИРОВАННОЙ СРЕДЕ РАЗРАБОТКЕ *BORLAND C++ BUILDER* НА ПРИМЕРЕ ПРИЛОЖЕНИЯ «КАЛЬКУЛЯТОР»

2.1. Цель работы: Разработать приложение калькулятор, использующее динамическое создание объектов.

2.2. Теоретические сведения

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

Компоненты разделяются на видимые (визуальные) и невидимые (невизуальные). Визуальные компоненты появляются во время выполнения точно так же, как и во время проектирования. Примерами являются кнопки и редактируемые поля. Невизуальные компоненты появляются во время проектирования как пиктограммы на форме. Они никогда не видны во время выполнения, но обладают определенной функциональностью (например, обеспечивают доступ к данным, вызывают стандартные диалоги *Windows* и др.)

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

Если выбрать компонент из палитры и добавить его к форме, инспектор объектов автоматически покажет свойства и события, которые могут быть использованы с этим компонентом. В верхней части инспектора объектов имеется выпадающий список, позволяющий выбирать нужный объект из имеющихся на форме. Таким образом, среда *C++ Builder* позволяет выполнять объектно-ориентированное программирование.

Ключевым понятием объектно-ориентированного программирования (ООП) является объект, который представляет собой особый структурированный тип переменных. Подобно обыкновенной структуре переменная типа объект под одним именем объединяет как данные различных типов (называемые, как и у записей, полями объекта), так функции обработки этих данных (называемые функциями-членами объекта).

В *C++* для описания объектов введен специальный тип - c*lass*. Синтаксис объявления класса:

```
class <имя класса> : <классы - родители>
{
public:
// Данные, доступные для внешнего использования
__publshed:
// Данные, которые видны в Инспекторе Объектов
protected:
// Данные, доступные только для потомков данного класса
private:
// Данные, используемые только внутри данного класса
}
```

Работа с классами осуществляется, например, следующим образом:

```
class TMyCl
{
public:
int m;
int readme(void);
void writeme(int i);
};
TMyCl ObjX;
ObjX.m=33; int y= ObjX.readme(); ObjX.writeme(z);
```

Как видно, обращение к полям и методам объекта подобно обращению к полям записи. Однако функции-члены (методы) являются подпрограммами. В описании объекта указывается лишь его заголовок, а его тело должно быть описано ниже:

int ObjX:: *readme*(*void*) { *return m*; }

void ObjX::*writeme*(*int i*) { *m=i*; }

В теле метода при обращении к полям и методам самого вызывающего их объекта имя объекта опускается, а при обращении к полям и методам других объектов . указывается явно. В каждом экземпляре объекта содержатся все его поля, но функции-члены класса хранятся в памяти в одной-единственной копии.

Создание объектов. Для инициализации значений полей объекта явно объявляют функцию-член, предназначенную для этого. Поскольку такая функция конструирует значение данного типа, она называется конструктором. Конструктор распознается потому, что имеет то же имя, что и сам класс.

Например:

class date {

//..

date (*int*, *int*, *int*);

};

Конструктор вызывается в момент создания класса, и предназначен для инициализации данных. Наличие конструктора в классе не обязательно, но при его отсутствии необходимо использовать другие функции для задания начальных значений. Для уничтожения динамически размещенных объектов класса и освобождения выделенной памяти используется деструктор. Имя деструктора представляет собой имя класса и стоящую перед ним тильду (~), например для класса *TMyCl* деструктор иметь имя ~ *TMyCl* ().

Наследование и полиморфизм. Свойство наследования заключается в том, что любой класс может быть порожден от другого класса. Класс, от которого порождаются другие классы, называют .классом-родителем., а класс, порожденный от другого класса называется классом-потомком. Порожденный класс наследует поля, методы и свойства своего родителя и обогащает их новыми полями, методами и свойствами. Таким образом, принцип наследования позволяет эффективно использовать уже наработанный задел программ и на его основе создать классы для решения все более сложных задач.

Полиморфизм это свойство классов, которое позволяет использовать одинаковое название метода для решения сходных, но несколько отличающихся в разных родственных классах задач. Обеспечивается это тем, что в классе-потомке одноименный метод переписывается по новому алгоритму, т.е. перекрывается. В результате в объекте-родителе и объекте-потомке будут действовать два одноименных метода с разными алгоритмами.

Метод потомка может вызывать методы предков. Для этого используют запись:

<базовый класс> : : <метод базового класса>

В случае, если у потомка неизмененный родительский метод должен вызывать измененный метод потомка, вызываемый метод должен быть описан каквиртуальный: в родительском классе с помощью слова *virtual*.

2.2.1. Классы, компоненты и объекты.

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

Свойства (*properties*) компонентов представляют собой расширение понятия данных-членов и используют ключевое слово *__property* для объявления. При помощи событий (*events*) компонент сообщает пользователю о том, что на него оказано некоторое воздействие.

Обработчики событий (*event handlers*) представляют собой методы, реализующие реакцию программы на возникновение событий. Типичные события – нажатие кнопки или клавиши на клавиатуре. Компоненты имеют ряд особенностей:

• Все компоненты являются прямыми или косвенными потомками класса *TComponent*. При этом иерархия наследования следующая: *TObject->TPersistent->TComponent->TControl->*….

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

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

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

2.2.2. Назначение библиотеки визуальных компонентов *VCL* среды разработки *Borland C++ Builder* 6.

Разработка программ на *Borland C++ Builder* 6 построена на основе выбора необходимых визуальных компонентов *VCL* и расположения их на поле форм (окон будущей программы). Компоненты, таким образом, служат кирпичиками, из которых строятся программы. Кстати, *Builder* в переводе на русский язык означает "строитель". Все компоненты *VCL* располагаются на палитре, расположенной ниже и правее главного меню. Палитра компонентов состоит из вкладок. Вкладки позволяют разделить большое число компонентов на группы, близкие по назначению. Щелкая левой кнопкой мыши по вкладкам, можно выбрать необходимую группу компонентов, которая будет отображаться на экране. Например, при щелчке по вкладке *Standard* на экране появятся компоненты, изображенные на рис. 2.1.

Рисунок 2.1 — Окно с компонентами

Среди них легко найти компонент *Button* (Кнопка) с надписью OK и компонент *Label* (Этикетка) с надписью A. Иногда часть компонентов не умещается на экране. Для их просмотра существуют кнопки прокрутки в виде треугольных стрелок слева и справа от самих компонентов. Щелкая по ним кнопкой мыши, можно сдвигать компоненты группы влево или вправо. Все компоненты видны на экране в виде иконок. При наведении на них курсора мыши эти иконки приподнимаются, как кнопки, а под курсором появляется строка с надписью, отображающей название данного компонента.

Любой из компонентов можно поместить на форму двумя способами. Первый способ заключается в двойном щелчке левой кнопкой мыши по самому компоненту- При этом компонент появится на форме строго по центру. После чего его можно переместить с помощью мыши в любую часть формы, а также изменить размеры так, как это делается с окнами программ. Второй способ заключается в том, что сначала производится один щелчок левой кнопкой мыши по компоненту, а затем — в нужном месте формы. При этом можно сразу задать расположение компонента на форме и его размеры. Для этого необходимо перед щелчком на форме выбрать место расположения левого верхнего угла компонента и щелкнуть в этом месте, не отпуская кнопки мыши. Далее необходимо продвинуть указатель мыши на место, выбранное для правого нижнего угла компонента, и отпустить кнопку. Во время такой операции ниже указателя мыши будет высвечиваться окно с координатами курсора для справки о его местоположении в сетке координат формы. Удаление компонента с формы производится клавишей <*Delete*> после его выделения щелчком левой кнопки мыши.

Палитру компонентов можно совсем убрать с экрана, как и другие элементы главной панели интерфейса. Для этого необходимо нажать правую кнопку мыши на панели главного меню или быстрых кнопок, и в открывшемся контекстном меню (рис. 2.2) убрать галочку напротив названия *Component Palette* (Палитра компонентов) щелчком левой кнопки мыши.

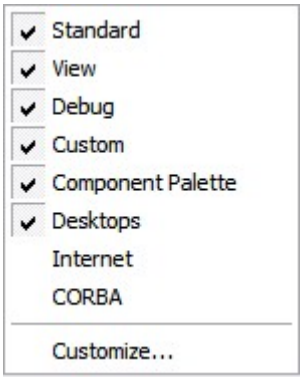

Рисунок 2.2 — Окно контекстного меню

Положение палитры компонентов на главной панели интерфейса *Borland C++ Builder* 6 не фиксировано, ее можно разместить по своему усмотрению. Для этого необходимо подвести курсор к левому краю палитры. Нажав и удерживая левую кнопку мыши, переместить палитру на новое место. Если группа вынесена за пределы главной панели, ей отводится отдельное окно. Для доступа к дополнительным командам и настройки палитры компонентов необходимо нажать правую кнопку мыши на поле палитры. При этом на экране появится окно контекстного меню команд палитры (рис. 2.3).

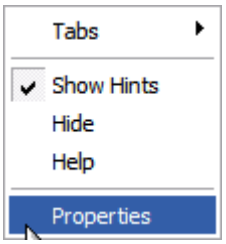

Рисунок 2.3 — Контекстное меню команд палитры

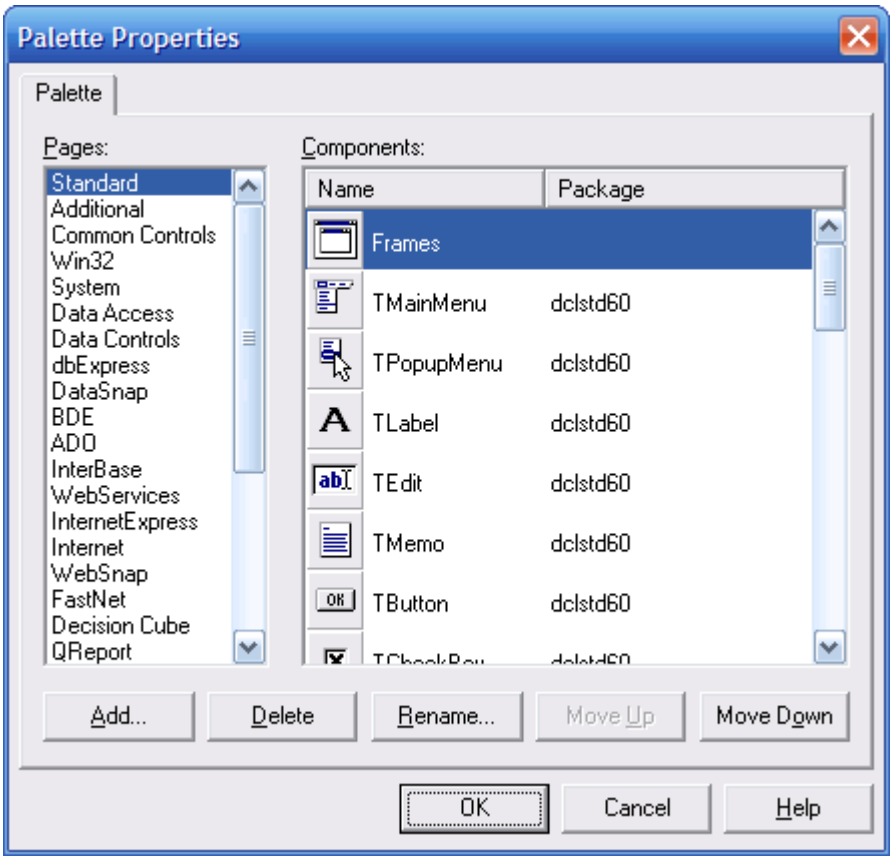

Рисунок 2.4 — Окно настроек палитры компонентов

В этом окне содержатся команды быстрого перехода к любой вкладке *Tabs*, показа подсказки *Show Hints*, скрытия палитры *Hide*, вызова справки *Help* и изменения настроек *Properties*. После щелчка левой кнопкой мыши по команде настроек *Properties* откроется окно настроек палитры компонентов (рис. 7.4).
В этом окне можно настроить каждую из вкладок палитры и всю палитру в целом. С помощью команды *Rename*можно переименовать любую из вкладок. Команда *Add* позволяет добавить новую вкладку, а команда *Delete*удалить любую вкладку. Для того чтобы, например, вкладка *System* следовала сразу за вкладкой *Standard*, необходимо выделить строку *System*, щелкнув по ней левой кнопкой мыши, и с помощью кнопки *Move Up*(Двигать вверх) переместить строку вверх до строки *Standard*. После чего необходимо нажать кнопку ОК. Теперь вкладка *System* на палитре компонентов будет располагаться сразу же за вкладкой *Standard*. Аналогично можно перемещать сами компоненты внутри вкладки с помощью кнопок *Move Up* и *Move Down* (Двигать вниз), или скрыть компонент с помощью кнопки *Hide*. Перемещаемый компонент предварительно необходимо выделить щелчком левой кнопки мыши.

2.2.3. Разработка классов.

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

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

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

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

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

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

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

MyButton = new TButton(*this*);

Когда форма уничтожается, автоматически уничтожается и кнопка *MyButton*. Можно создать компонент, у которого нет владельца, передавая значение параметра 0 конструктору компонента. Его уничтожение выполняется с помощью оператора *delete*.

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

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

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

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

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

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

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

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

(*Owner*) { *FIsTrue* = *true*; }

2.2.4. Создание визуальных компонентов во время выполнения программы.

При создании различных программ (а особенно простых игр сложности типа «Сапёра») регулярно возникает желание создавать визуальные компоненты (кнопки, компоненты типа *TImage* и т. п.) на этапе выполнения программы. В частности, вы можете захотеть создать сто кнопок и присвоить им всем очень похожие обработчики события *OnClick*. Не нужно это делать вручную.

Рассмотрим пример создания и удаления кнопки во время выполнения программы. Кнопка — это объект типа *TButton*. Для её создания необходимо выделить память под новый объект такого типа и вызвать его конструктор, а затем присвоить нужные значения его свойствам. Делается это, например, так:

```
TButton *bt = new TButton(Form1);
bt->Parent = Form1;
bt->Caption = "Caption";
bt->Left = 100;
bt->Top = 80;
/* и так далее */
```

Аналогично можно создать массив кнопок любого размера.

Обработка событий для динамически созданных компонентов. Чтобы присвоить такой кнопке обработчик, необходимо создать в классе, описывающем форму, метод, удовлетворяющий всем требованиям на обработчик (т. е. принимающий нужные параметры).

Любой обработчик события принимает параметром *TObject* Sender*. Этот параметр при вызове обработчика указывает на объект, с которым связано данное событие. Например, определение, какая кнопка была нажата, можно реализовать так:

```
void TForm1::Button1Click(TObject *Sender)
{
TButton *bt = static_cast<TButton *>(Sender); // приведение типа
ShowMessage("Нажата кнопка" + bt->Caption);
}
```

Кроме того, каждый компонент имеет свойство *Tag*, в котором можно хранить произвольное целое число. Его также очень удобно использовать для определения того, какой из компонентов вызвал данное событие.

2.3. Методические рекомендации к выполнению работы

Программа Калькулятор (рис. 2.5) позволяет выполнить простейшие расчеты. Следует обратить внимание, что в качестве индикатора используется компонент *StaticText*.

Рисунок 2.5 — Простейший калькулятор

Форма и окно программы калькулятор в режиме разработки приведены на рис. 2.6, демонстрирует создание компонентов во время работы программы или, как иногда говорят, "в коде". Кнопки калькулятора — это объединенные в массив компоненты `SpeedButton`. Создание и настройку кнопок выполняет конструктор формы, он же определяет (задает) процедуру обработки события `click`. Следует обратить внимание, что объявление массива компонентов `SpeedButton` и функции обработки события `click` находится в объявлении типа формы, в заголовочном файле.

Рисунок 2.6 — Форма и окно программы в режиме дизайна

```
// объявление формы — фрагмент h-файла
class TForml : public TForm
{
__published:
TStaticText *StaticTextl; // индикатор
private:
// ** эти объявления вставлены сюда вручную **
TSpeedButton * btn[16]; // кнопки
// процедура обработки события Click на кнопке
void __fastcall btnClick( TObject *Sender);
public:
__fastcall TForml (TComponent* Owner);
};
// *** модуль формы ***
float ac; // аккумулятор (первый операнд)
int op; // операция
int fd; /* fd == 0 — ждем первую цифру числа, например, после того, как бьша нажата клавиша "+"; fd = 1 — ждем ввода следующей цифры или нажатия клавиш операции */
#define WBTN 35      // ширина кнопки
#define HBTN 20      // высота кнопки
#define DXBTN 6      // шаг кнопок по X
#define DYBTN 6      // шаг кнопок по У
// текст на кнопке
Char btnText[] = "789+456-123=00.c";
#define CM –1 // запятая
#define EQ  –2    //"="
#define PL  –3    // "+"
#define MN  –4 // "–"
#define CL  –5 // "C"
// идентификатор кнопки
int btnTag [] = {7,8,9, PL, 4,5, 6,MN, 1,2, 3,EQ, 0, 0,CM,CL} ;
// конструктор формы
__fastcall TForml::TForml(TComponent* Owner)
```

```
: TForm(Owner)
{
int left, top; // положение кнопки
top = 48;
int к = 0; // индекс массива btn
for ( int i = 0; i < 4; i++ ) // четыре ряда кнопок
{
left = 8;
for ( int j = 0; j < 4; j++) // по 4 в каждом ряду
{
btn [k] = new TSpeedButton (Forml) ;
btn[k] ->Parent = Forml; // "посадить" кнопку на
                         // форму
// настройка кнопки
btn[к]->Left = left;
btn[к]->Top = top;
btn [к] ->Width = WBTN;
btn[k]->Height = HBTN;
//btn[k]->Flat = true;
btn[k]->Font->Color = clNavy;
btn[k]->Caption = btnText[k]; // текст на кнопке
btn[k]->Tag = btnTag[k]; // идентификатор кнопки
// задать процедуру обработки события Click
btn[к]->OnClick = btnClick; // см. объявление в
                            // h-файле формы
k++;
left = left + WBTN+ DXBTN;
}
top = top + HBTN + DYBTN;
}
// объединить две кнопки "0" (btn [12] и btn[13]) в одну
btn[13]->Visible = false;
btn [12] ->Width = 2* WBTN + DXBTN;
op = EQ;
}
// Щелчок кнопке btn[i]
// ( одна процедура для всех кнопок )
void      __fastcall TForml: :btnClick (TObject: *Sender)
{
TSpeedButton *btn;
int id; // идентификатор нажатой кнопки
btn = (TSpeedButton*)Sender;
// свойство Tag кнопки содержит ее идентификатор
id = btn->Tag;
// кнопка "1" .. "9"
if ( id > 0 ) {
if ( fd == 0 ) {
StaticTextl->Caption = btn->Tag;
fd = 1;
}
else
StaticTextl->Caption = StaticTextl->Caption + btn->Tag;
return;
}
```

```
// кнопка "0"
if ( id == 0) {
if ( StaticTextl->Caption != "0" )
StaticTextl->Caption = StaticTextl->Caption + btn->Tag;
return;
}
// кнопка ","
if ( id == CM) {
if ( StaticTextl->Caption.Pos(", " ) == 0 ) {
StaticTextl->Caption = StaticTextl->Caption + ", ";
fd = 1;
}
return;
}
// кнопка "c"
if ( id == CL) {
ac = 0;
id = EQ;
fd = 0;
StaticTextl->Caption = 0; return;
}
// остальные кнопки: "+", "–" и "="
float op2; // операнд (число на индикаторе )
op2 = StrToFloat(StaticTextl->Caption);

/* так как нажата клавиша операции, то это значит,
что надо выполнить предыдущую операцию, код которой находится в переменной op */

switch (op) {
case EQ :
ac = op2;
break;
case PL :
 ac = ac + op2;
 break;
case MN :
ac = ac - op2;
break;
}
StaticTextl->Caption = FloatToStrF(ac,ffGeneral,15,6);
op = id; // запомнить текущую операцию
fd = 0; // ждем новое число
// деструктор формы
voidfastcall TForml::FormDestroy(TObject ^Sender)
{
// уничтожить компоненты, созданные программой
for ( int i = 0; i < 16; i++)
delete btn [i];
}
```

2.4. Индивидуальные задания

Согласно значению последней цифры зачетной книжки *N* из таблицы 2.1 выбрать дополнительную функция которую должен выполнять калькулятор.

Таблица 2.1 — Индивидуальное задание

Вариант	Функция
0	*Sin*(*x*)
1	*Cos*(*x*)
2	*Tan*(*x*)
3	*Ctg*(*x*)
4	1/*x*
5	*Asin*(*x*)
6	*Acos*(*x*)
7	*Atan*(*x*)
8	*Actg*(*x*)
9	x^2

2.5. Содержание отчета

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

Перед защитой отчета студент должен продемонстрировать работу программы.

2.6. Контрольные вопросы

2.6.1. Каким образом и в каких файлах можно создать собственный класс, динамический объект и вызвать методы класса?
2.6.2. Что представляют собой свойства компонентов? Приведите примеры нескольких свойств формы.
2.6.3. Что представляют собой события компонентов? Приведите примеры.
2.6.4. Каким образом можно изменить значения свойств компонентов?
2.6.5. Объявить класс *MyClass*, содержащий некоторую строку, и метод *swap*(), заменяющий строку другой строкой. Выполнить тестирование.
2.6.6. Объявить класс и определить размерность объектов данного класса.
2.6.7. Чем отличаются оператор-функции, объявленные как *friend*, от оператор-функций членов класса *C++* и можно ли их использовать в *C++Builder*?
2.6.8. На каком этапе происходит выделение памяти под объекты компонентных классов?
2.6.9. Что представляет собой общедоступное свойство класса и егоопубликованное свойство?
2.6.10. Что произойдет в результате выполнения следующего кода?

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

2.6.12. Как распознать нажатые функциональные клавиши?
2.6.13. Куда поступают события клавиатуры? Какое свойство формы влияет на то, куда поступают события клавиатуры?
2.6.14. В следующем коде какой из объектов отвечает за удаление кнопки?

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