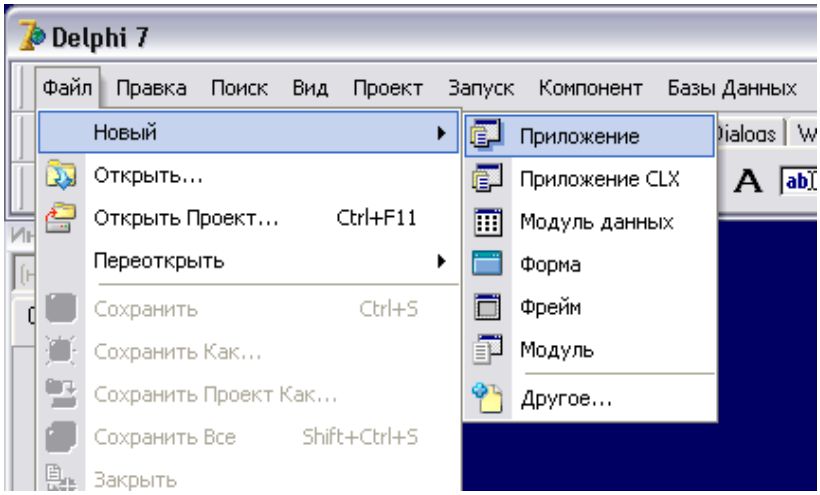
Лабораторная работа №1.
Знакомство со средой Delphi.

Время: 180 мин.
Что нужно освоить:
1) каким образом создать и запустить простейшее приложение в среде Delphi;
2) каким образом добавлять на форму различные компоненты из библиотеки компонентов и как производить простейшие настройки их свойств;
3) как формируются обработчики событий на компонентах;
4) как организовать ввод/вывод и преобразование данных с использованием компонентов Edit, Label, Button;
5) как динамически менять свойства компонентов во время выполнения программы..

1. О ПОДКЛЮЧЕНИИ МОДУЛЕЙ

Создайте папку (назовите её своей фамилией), в которую будете сохранять разрабатываемые приложения. Для каждого приложения в дальнейшем в главной папке следует создавать отдельный каталог. Создайте новое приложение в среде Delphi:



Перейдите в окно кода программы, нажав клавишу F12 или кликнув по окну кода мышкой. Вы увидите текст модуля:

```
unit Unit1;

interface

uses
Windows, Messages, SysUtils, Variants, Classes, Graphics,
Controls, Forms, Dialogs;

type
TForm1 = class(TForm)
```

```
private
{ Private declarations }
public
{ Public declarations }
end;

var
Form1: TForm1;

implementation

{$R *.dfm}

end.
```

Данное приложение уже работоспособно, его можно запустить на выполнение нажав F9 или экранную клавишу пуск. Его (окно приложения) можно подвигать, изменить размеры, минимизировать, максимизировать, закрыть.

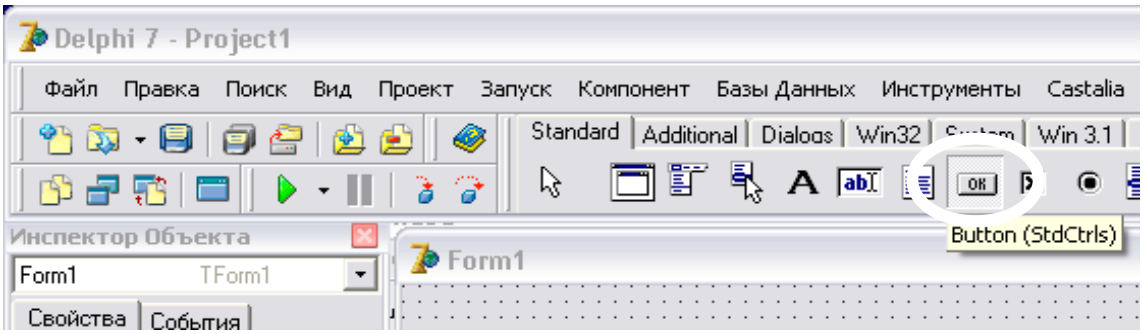
Обратим внимание на раздел подключения модулей. Ключевое слово Uses определяет начало списка модулей, которые используются текущим модулем, программой или библиотекой. Следует заметить, что при создании нового приложения Delphi самостоятельно размещает несколько модулей в разделе подключения модулей uses. Для такого простейшего приложения, которое вы только что запустили это количество избыточно. Попробуйте из раздела подключения модулей uses удалить все модули за исключением Forms и снова запустите программу.

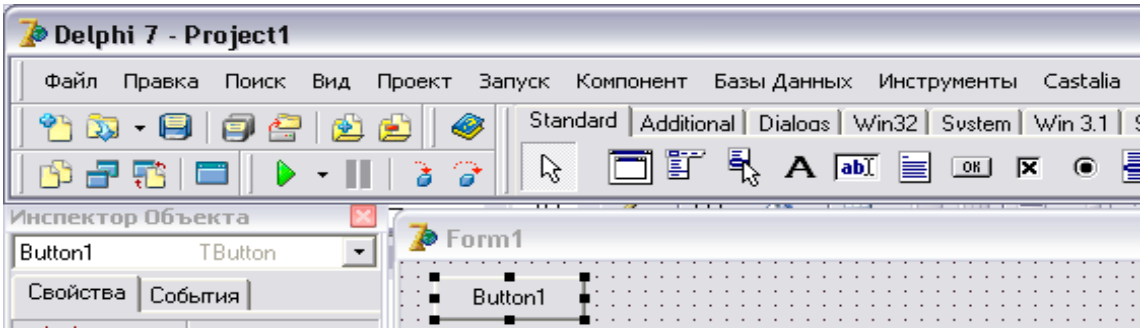
Подключаемые модули Delphi имеют расширение dcu (например, Forms.dcu) и представляют собой откомпилированный модуль проекта в промежуточном формате. Когда компилируется программа, все модули компилируются в файлы формата dcu, а потом собираются в один исполняемый exe файл. Если модуль не изменялся с последней компиляции, то Delphi пропустит его, а во время сборки будет использовать существующий файл формата dcu, чтобы сократить время компиляции. Загляните в папку где вы сохранили свой проект и вы обнаружите файл, совпадающий по имени с модулем вашего проекта Unit1, но с расширением dcu.

Стандартные подключаемые модули Delphi располагаются в каталоге C:\Program Files\Borland\Delphi7\Lib. При добавлении на форму визуального компонента из библиотеки компонентов Delphi самостоятельно добавляет необходимые модули в раздел uses.

Добавить компонент на форму в среде визуального программирования просто: кликните на соответствующую иконку компонента – она станет активной, затем кликните в том месте формы, где хотите компонент разместить, он там и окажется.

Начнем с того, что добавим на форму кнопку Button1, иконка этого компонента располагается на вкладке Standart.

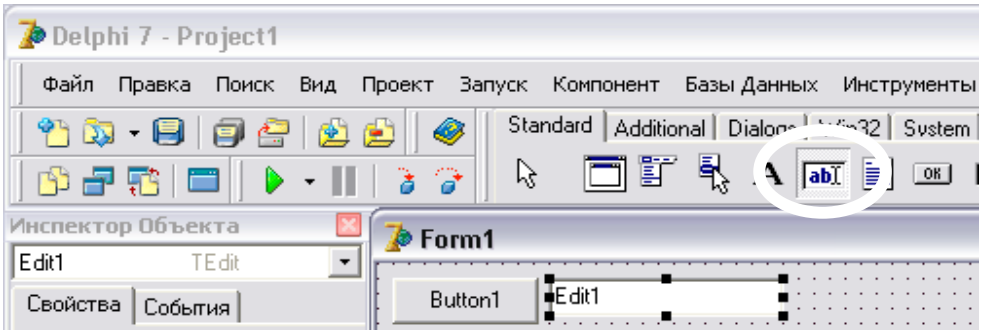




Запустите программу на выполнение – она работает, кнопка нажимается, но пока не выполняет никаких действий, так как мы и не задали никаких обработчиков событий. Однако все эти новые возможности стали доступны благодаря тому, что Delphi самостоятельно подключил еще несколько стандартных модулей – обратите внимание вновь на раздел uses. Так Delphi поступает при размещении на форме визуальных компонентов. Но часто возникают ситуации когда программисту приходится самостоятельно прописывать подключаемый модуль. К таким, например, относятся модули Math (математические функции – см. приложение I) или SysUtils (делает доступными множество подпрограмм манипулирования данными, таких как IntToStr).

Убедимся в этом при проведении простого эксперимента.

Закройте программу, если еще этого не сделали. Добавьте на форму ещё один компонент – однострочное редактируемое текстовое поле – Edit1.



Кликните два раза на кнопке, на что Delphi отзовется созданием процедуры обработчика события – нажатие на кнопке:

```
procedure TForm1.Button1Click(Sender: TObject);
begin

end;
```

Заполним самостоятельно кодом этот обработчик:

```
procedure TForm1.Button1Click(Sender: TObject);
```

```
begin
  Edit1.Text:=5;
end;
```

Предполагая, что по нажатию клавиши Button1 в текстовое поле будет выведена цифра 5. Запустим программу на выполнение.

Что-то не так – не работает. Пишет про несовместимые типы String и Integer и про невозможность откомпилировать модуль. Действительно, текстовое поле оно потому текстовое, что предназначено для отображения строк. Внесем изменения: сделаем обрамление из одинарных кавычек ‘5’:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Edit1.Text:='5';
end;
```

и запустим программу. Работает. Однако, если в программе задумана какая-то обработка данных, то хранить их, видимо, придется все-таки в более подходящем – нетекстовом виде. Значит в момент вывода в текстовое поле следует полученный результат преобразовать из текстового в числовой формат – для целых чисел подойдет функция IntToStr. Как узнать о ее формате и посмотреть примеры ее использования. Наберите в окне кода модуля Unit1 такой текст: IntToStr, оставьте курсор где-то в пределах IntToStr и нажмите Ctrl+F1 (*таким способом можно узнать много полезной информации и про другие функции*). Откроется справка:

```
Delphi syntax:
function IntToStr(Value: Integer): string; overload;
function IntToStr(Value: Int64): string; overload;
```

из которой станет ясно, что аргументом этой функции является переменная целого типа, а результатом строковая переменная. Кроме того, ключевое слово overload укажет, что эта функция относится к переподгружаемым подпрограммам. Наличие механизма переподгружаемых подпрограмм позволяет реализовывать подпрограммы, выполняющие одинаковые действия на основе параметров разных типов, что упрощает их вызов, так как отпадает необходимость приведения параметров к конкретным типам данных. Кроме того, переподгружаемые подпрограммы могут создаваться не только с параметрами разных типов, но и вообще с разным количеством параметров. Для того, чтобы компилятор мог выбрать правильную подпрограмму из нескольких перегруженных, они должны отличаться так называемой сигнатурой – последовательностью типов данных в списке параметров. В данном случае в качестве параметра функции IntToStr может использоваться данное типа Integer или Int64 (о числовых типах данных см. Приложение II).

Для приведения программы к корректному виду внесем соответствующие изменения:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Edit1.Text:=IntToStr(5);
end;
```

Запустим программу на выполнение … и опять не получим желаемого результата – оказывается идентификатор `IntToStr` не задекларирован. В чем же проблема? Вот тут то мы и возвращаемся к тому с чего начинали – Delphi не всегда самостоятельно подключает необходимые стандартные модули. В данном случае не было добавления на форму визуального компонента и связанного с этим автоматического добавления необходимого модуля/модулей в список подключаемых модулей в раздел uses, поэтому чтобы добиться положительного результата следует самостоятельно добавить в раздел uses модуль SysUtils.

Добавили? Пробуем, … работает!

==========

Чтобы в дальнейшем не испытывать затруднений при усложнении программы давайте вернем все заявленные по умолчанию модули в разделе uses на место:

```
Uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics,
Controls, Forms, Dialogs, StdCtrls;
```

==========

Кстати, есть интересная возможность узнать диапазон определенного типа данных. Приведите программу к такому виду:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  min: Integer;
begin
  min:=Low(Integer);
  Edit1.Text:=IntToStr(min);
end;
```

или к такому:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  max: Int64;
begin
```

```
  max:=High(Int64);
  Edit1.Text:=IntToStr(max);
end;
```

Если результат не помещается в Edit1, то несложно в инспекторе объектов изменить значение свойства Width.

==========

Список подключаемых модулей uses может находиться в разделах Interface и Implementation. Если подключаемые модули используются только в разделе Implementation, то их следует объявлять именно там, это поможет избежать излишних перекомпиляций (пример ниже).

Порядок модулей в списке uses важен – те, что объявлены позже в этом списке имеют приоритет над объявленными раннее. Это означает, что в случае, если одна и та же подпрограмма определена в двух или более модулях, то при обращении к ней будет задействована та, которая описана в модуле, стоящем в списке позже, чем другие, содержащие эту же подпрограмму. Если же возникла необходимость обратиться к подпрограмме, находящейся в модуле, стоящем в списке раньше, то можно указать компилятору Delphi использовать желаемый модуль приписыванием имени модуля к имени подпрограммы.

```
unit Unit1;

interface

uses
  Windows, Messages, Variants, Classes, Graphics, Controls,
Forms, Dialogs, StdCtrls;

type
  TForm1 = class(TForm)
    Button1: TButton;
    Edit1: TEdit;
    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation
{$R *.dfm}

uses
  SysUtils;

procedure TForm1.Button1Click(Sender: TObject);
var
  min: Integer;
```

```
begin
  min:=Low(Integer);
  Edit1.Text:=SysUtils.IntToStr(min);
end;

end.
```

==========

В каждом отдельно взятом модуле как правило содержится несколько подпрограмм, но доступными для внешних модулей оказываются только те, которые обозначены в разделе public. В свою очередь раздел private предназначен для объявлений, доступных только текущему (самому владельцу) модулю.

Разберем это обстоятельство на конкретном примере.

Создадим процедуру `itog` в модуле Unit1 и вынесем в нее все действия из обработчика события `TForm1.Button1Click`. Процедуру объявим в разделе private:

```
unit Unit1;

interface

uses
  Windows, Messages, Variants, Classes, Graphics, Controls,
Forms, Dialogs, StdCtrls, SysUtils;

type
  TForm1 = class(TForm)
    Button1: TButton;
    Edit1: TEdit;
    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
    procedure itog;
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation
{$R *.dfm}

procedure TForm1.itog;
var
  min: Integer;
begin
  min:=Low(Integer);
  Edit1.Text:=SysUtils.IntToStr(min);
end;

procedure TForm1.Button1Click(Sender: TObject);
```

```
begin
  itog;
end;

end.
```

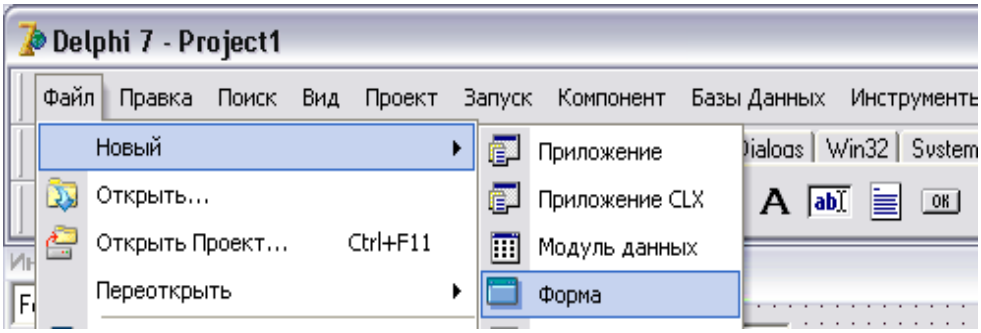
Для чего вообще необходимо что-то оформлять в отдельную процедуру? Для того чтобы не дублировать код в разных местах программы. У нас по замыслу будет еще вторая форма в приложении из которой мы будем выполнять те же действия, что описаны в обработчике `TForm1.Button1Click`.

Испытаем программу – работает как и ранее: ответ пишет в Edit1.

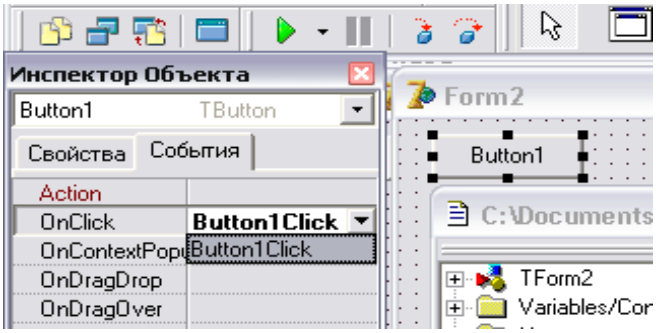
==========

Теперь добавим к проекту еще один модуль Unit2 с формой, и попробуем обращаться к процедуре `itog` модуля Unit1 из модуля Unit2. Результат будет зависеть от того где именно мы объявили процедуру `itog`.

Новый модуль автоматически создается, когда мы создаем новую форму:



На новой форме (Form2) разместите только одну кнопку Button1. Создайте заготовку процедуры обработчика события «нажатие на кнопку» как мы это уже делали ранее (двойным кликом по кнопке). Кстати, создавать заготовку процедуры часто приходится не на самом компоненте кликая мышкой, а выбирая возможное событие из вкладки «События» Инспектора объектов. В данном случае можно дважды кликнуть в Инспекторе объектов по полю, находящемуся справа от пункта OnClick. Туда и запишется обращение к созданной процедуре. Позже, когда компонентов на форме и обработчиков событий будет больше, в этом поле в раскрывающемся списке можно будет выбирать необходимый обработчик событий (уже созданный ранее):



Так как у нас уже несколько модулей и форм, то актуальным становится вопрос как между ними переключаться: переключение между формой и модулем – F12, показать меню выбора формы – Shift+F12, показать меню выбора модуля – Ctrl+F12.

Заготовка процедуры обработчика события пока пуста, но давайте туда запишем код обращения к процедуре `itog` из модуля Unit1.

```
procedure TForm2.Button1Click(Sender: TObject);
begin
  Form1.itog;
end;
```

Попробуем запустить – результат будет отрицательным: сначала Delphi спросит – «включать ли форму 2 в проект?» – соглашаемся, а при повторном запуске окажется, что процедура `itog` незадекларирована. Итак, мы ограничили доступ внешних модулей к процедуре `itog` модуля Unit1. Если же как раз наоборот необходимо эту процедуру использовать и вне модуля Unit1 тогда следует перенести объявление этой процедуры из раздела private в public.

Сделайте этот перенос и запустите программу. Программа заработала, Форма 1 видна и как и раньше работает нажатие на клавишу 1. Однако, а где же форма 2 и как проверить работоспособность нажатия кнопки на ней. Форме 2 никто не указал, что она должна быть видна. При запуске приложения автоматически показывается только главная (как правило первая) форма. Вывести форму 2 на экран можно по разному. Но мы пойдем простым путем и в тексте программы явно укажем наши пожелания. Напомню, что текст программы находится в файле Project1.dpr. Как на него посмотреть и исправить его? Нажмите Ctrl+F12 и выберете окно с проектом:

```
program Project1;

uses
  Forms,
  Unit1 in 'Unit1.pas' {Form1},
  Unit2 in 'Unit2.pas' {Form2};

{$R *.res}

begin
  Application.Initialize;
  Application.CreateForm(TForm1, Form1);
  Application.CreateForm(TForm2, Form2);
  Application.Run;
end.
```

Тут указано, что программа использует модули: Forms, Unit1 и Unit2. В тексте программы первым идет необязательный метод Application.Initialize, который выполняет соответствующие задачи инициализации OLE Automation. С использованием данной технологии построены очень многие популярные приложения, включая Microsoft Office, Microsoft Visio, программы семейства Corel Draw, интегрированная оболочка Visual

Studio. Если вы свое приложение не собираетесь расширять таким функционалом, то можно спокойно удалить строку "Application.Initialize;" из исходного кода вашего проекта. Следующие две строки обеспечивают создание двух форм нашего проекта и, наконец, последняя строка предназначена для запуска приложения. При этом для пользователя доступным становится только первая форма.

Для того, чтобы вторая форма при запуске приложения также была доступна, внесите в текст программы незначительное изменение в виде одной дополнительной команды: `Form2.Show`.

```
Begin
  Application.Initialize;
  Application.CreateForm(TForm1, Form1);
  Application.CreateForm(TForm2, Form2);
  Form2.Show;
  Application.Run;
end.
```

Запустите программу, форма 2 запустится вместе с формой 1. Испытайте работоспособность Button1, расположенной на Form2.

Краткие выводы:

1) наиболее распространенным типом модуля в Delphi является модуль со связанным с ним окном (формой), который создается автоматически при создании новой формы проекта;

2) интерфейсная часть модуля, связанного с формой, обычно содержит объявление нового класса (наследника TForm) и автоматически обновляется Delphi в ходе конструирования окна.

3) привязка модуля к определенной форме проекта обеспечивает обработку соответствующих событий;

4) при программировании в Delphi следует учитывать, что отдельные участки программного кода находятся не в тексте основной программы, а рассредоточены по отдельным модулям;

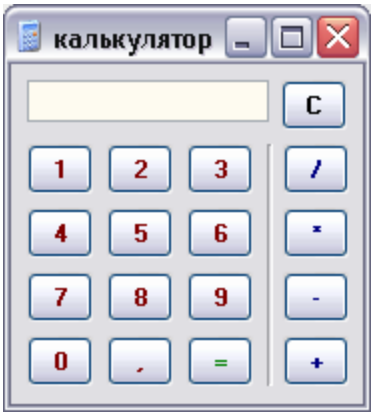
5) синтаксис Delphi позволяет обеспечить различный уровень доступа к подпрограммам модуля, расширяющим функционал предка TForm:

– в закрытый раздел класса (наследника TForm) private помещают объявления переменных, функций и процедур, включаемых в класс новой формы, но не доступных для других модулей

– в открытый раздел класса (наследника TForm) public помещают объявления переменных, функций и процедур, включаемых в класс новой формы и доступных для других модулей.

2. СОЗДАЕМ ПЕРВОЕ ОСМЫСЛЕННОЕ ПРИЛОЖЕНИЕ – КАЛЬКУЛЯТОР

самостоятельно выполняемая и контролируемая преподавателем часть работы



Создадим программу – простой калькулятор с основными арифметическими действиями: «+», «–», «*», «/»; с наличием кнопки «Сброс», «=» и возможностью оперировать с числами вещественного типа, то есть с числами с «,».

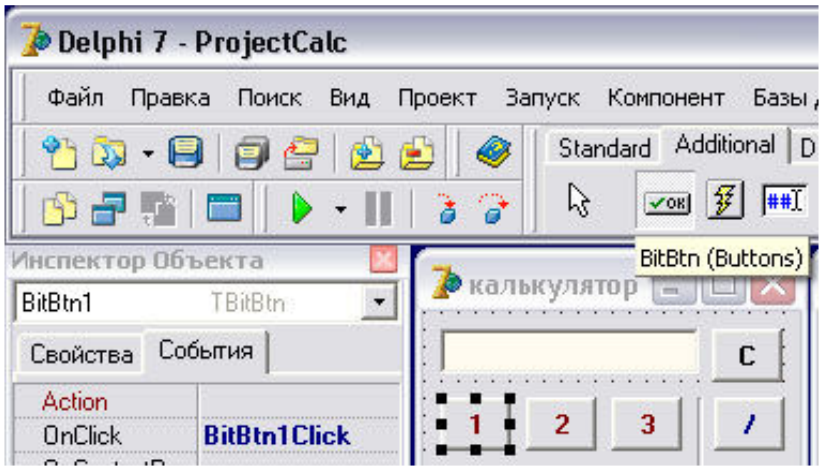
Создайте новое приложение в среде Delphi.

Часть 1. Разработка интерфейса программы.

Необходимые компоненты:

– Edit – текстовое поле для организации ввода чисел и вывода результата арифметического дйствия;

– BitBtn – кнопка из вкладки Additional, в отличие от кнопки Button с вкладки Standart, у BitBtn настраивается цвет надписи – нам это пригодится для цветового выделения кнопок (группа цифр и запятая, группа действий, сброс и равно).



В инспекторе объектов задайте свойства формы:

Name: calc

Caption: калькулятор

Color: цвет по желанию

BorderIcons.biMaximize: False;

Разместите на форме компонент Edit.

В инспекторе объектов задайте его свойства:

Text: пустая строка;

Color: цвет по желанию.

Разместите на форме необходимое число кнопок:

– цифры от 0 до 9,

– кнопка «,»,

– знаки арифметических операций,

– знак «=»,

– кнопка Сброс.

В инспекторе объектов задайте их свойства:

Name: пока можно оставить по умолчанию, но для последующего усовершенствования программы лучше сразу дать осмысленные имена,

Caption: соответствующее выполняемой функции название (цифра, запятая, знак действия),

Font: шрифт и цвет шрифта по желанию.

Часть 2. Разработка алгоритма работы калькулятора.

Наиболее очевидным можно считать такой алгоритм работы:

1) вводим первое число в поле Edit1;
2) нажимаем кнопку действия («+»,«–»,«*»,«/») – первое число и наименование действия сохраняем для последующего использования;
3) вводим второе число в поле Edit1;
4) нажимаем кнопку «=» – результат выводим в поле Edit1;

Часть 3. Разработка процедур обработчиков событий.

Обработчики событий, естественно, зависят от задуманного алгоритма работы программы.

А. Напишите процедуру добавления цифры (`0..9`) к текстовому полю Edit1. В самом примитивном варианте следует создать обработчик события «клик на клавише» для каждой клавиши. В обработчик события следует добавить одну строку, например, для клавиши «1» – `Edit1.Text:=Edit1.Text+'1';`

Аналогично напишите процедуры ввода остальных цифр и символа «,»

В дальнейшем мы рассмотрим методы, позволяющие существенно сократить дублирование полученного сейчас программного кода.

Б. Создайте процедуру очистки поля `Edit1` кнопкой «C», используя один из двух вариантов:

- `Edit1.Text:='';`
- `Edit1.Clear.`

В. Создайте процедуры обработки нажатий кнопок «+», «–», «/», «*» и «=».

Процедура обработчика события такой кнопки действия («+»,«–»,«*»,«/») не предполагает собственно выполнения арифметического действия, ведь ещё не введён второй аргумент, достаточно реализовать следующую последовательность:

- преобразовать строку из свойства Text поля Edit1 в число с использованием функции StrToFloat,
- сохранить результат в глобальную (если переменная будет локальной, то она не будет доступна в других процедурах) переменную X типа Real
- очистить поле Edit1 для ввода второго аргумента,
- зафиксировать в глобальной переменной N (тип можете выбрать сами) наименование выполняемого арифметического действия («+»,«–»,«*»,«/»).

Г. Создайте процедуру обработки нажатия кнопки «=».

В процедуре обработки этой кнопки необходимо выполнить следующие действия:

- преобразовать строку из свойства Text поля Edit1 в число с использованием функции StrToFloat,
- сохранить это число в локальной переменной Y;
- в зависимости от наименования операции, выполнить арифметическое действие с переменными X и Y;
- результат преобразовать в строку с помощью функции FloatToStr и вывести в поле Edit1.

Испытайте программу, исправьте явные ошибки.

Часть 4. Усовершенствование программы.

1. Продумайте последовательность действий при возникновении ситуации когда поле Edit1 пустое, а пользователь нажимает клавишу арифметического действия или клавишу получения результата («=»).

2. Измените дизайн формы по вашему усмотрению (изменить цвет фона поля Edit1, можно использовать компоненты Panel и Bevel, подобрать размеры и расположение цифровых и управляющих кнопок). Для использования современного дизайна компонентов в раздел uses включите модуль XPMan – текст программы останется тем же, но внешний вид компонентов немного поменяется.

3. Посмотрите на стандартный калькулятор Windows, там есть меню с возможностью копировать результат в буфер обмена Windows и вставить результат из буфера обмена в текстовое поле Edit1. Эту возможность можно реализовать используя методы

– `Edit1.CopyToClipboard;` // копировать в буфер обмена
– `Edit1.PasteFromClipboard;` // вставить из буфера обмена

Пока вы не освоили порядок создания главного меню приложения можно реализовать эти возможности, привязав их к обработчикам событий все тех же Button.

4. Так же обычно в главном меню есть пункт «О программе». Это можно реализовать разными способами. Для начала попробуем использовать метод `MessageDlg`. Создайте отдельную кнопку с надписью «О программе» и привяжите к ней этот код:

```
MessageDlg('      Программа Калькулятор'+#13+
           '      Version 1.0, Freeware'+#13+
           '========================='+#13+
           '     '+#$A9+' Имя Фамилия, год',
           MtInformation,[mbOK],0);
```

Особенности использования `MessageDlg` можно узнать в справке Delphi.

5. Добавьте функции вычисления квадрата числа, любой степени числа, квадратного корня числа, абсолютного значения числа, максимума из двух чисел и некоторых других функций из Приложения I. При вычислении тригонометрических функций учесть, что тригонометрические функции в качестве аргументов принимают радианную меру угла, что не совсем удобно пользователю. Напомню, что переход от градусов к радианам можно организовать так: `r:=g*Pi/180`.

6. Добавьте функции перевода числа из десятичной системы счисления в двоичную и восьмеричную, из двоичной в восьмеричную и десятичную, из восьмеричной в двоичную и десятичную. Желательно аналогичные (повторяющиеся) участки программного кода оформить в виде подпрограмм (процедур, функций).

==========

Список требований к программе, которые вы должны будете соблюдать при разработке приложений по мере освоения системы Delphi:

1. Дружественный интерфейс программы: главное меню, горячие клавиши, наличие подсказок (Hint) на визуальных компонентах, иконка программы, пункт «О программе», наличие справки о функциях программы.

2. Грамотный дизайн формы (окна): компоненты должны быть сгруппированы в контейнеры (Panel, GroupBox, …); аккуратное размещение компонентов на форме, читаемость текстуальных и цифровых сообщений и т.п.

3. Структурированное написание текста модуля (модулей) программы: наличие отступов, комментариев, …

4. Использование оператора «case» вместо «if then else» при наличии выбора из нескольких вариантов.

5. Оформление повторяющихся блоков модуля в отдельные процедуры и функции.

6. Разбиение длинной процедуры на отдельные процедуры для лучшей читаемости.

Математические функции и процедуры модуля Math.

Функции и процедуры для работы с данными перечислимого типа.

Dec	Уменьшает значение переменной на заданную величину.
Inc	Увеличивает значение переменной на заданную величину.
Odd	Определяет четность аргумента.
Ord	Возвращает порядковый номер выражения перечислимого типа или код ASCII выражения символьного типа.
Pred	Возвращает значение, предшествующее аргументу.
Succ	Возвращает значение, следующее за аргументом.

Тригонометрические функции и процедуры.

ArcCos	Вычисляет арккосинус аргумента.
ArcCosh	Вычисляет гиперболический арккосинус аргумента.
ArcSin	Вычисляет арксинус аргумента.
ArcSinh	Вычисляет гиперболический арксинус аргумента.
ArcTan	Вычисляет арктангенс аргумента.
ArcTan2	Вычисляет arctg(Y/X).
ArcTanh	Вычисляет гиперболический арктангенс аргумента.
Cos	Вычисляет косинус аргумента.
Cosh	Вычисляет гиперболический косинус аргумента.
Cotan	Вычисляет котангенс аргумента.
Hypot	Вычисляет длину гипотенузы прямоугольного треугольника.
Sin	Вычисляет синус аргумента.
SinCos	Вычисляет одновременно синус и косинус аргумента.
Sinh	Вычисляет гиперболический синус аргумента.
Tan	Вычисляет тангенс аргумента.
Tanh	Вычисляет гиперболический тангенс аргумента.

Арифметические функции и процедуры.

Abs	Возвращает абсолютное значение аргумента.
Ceil	Округляет значение аргумента в большую сторону.
Exp	Вычисляет значение ex.
Floor	Округляет значение аргумента в меньшую сторону.
Frac	Возвращает дробную часть аргумента.
Frexp	Возвращает мантиссу и экспоненту аргумента.
Int	Возвращает целую часть аргумента.
IntPower	Возводит аргумент X в целочисленную степень Y.
Ldexp	Вычисляет $X*2^{Y}$.
Ln	Вычисляет натуральный логарифм ln(x).
LnXP1	Вычисляет натуральный логарифм ln(x+1).
Log10	Вычисляет десятичный логарифм.
Log2	Вычисляет логарифм аргумента по основанию 2.
LogN	Вычисляет логарифм аргумента по основанию N.
Max	Возвращает большее из двух чисел.
Min	Возвращает меньшее из двух чисел.
Pi	Возвращает значение числа Пи.
Poly	Вычисляет однородный полином.
Power	Возводит X в степень Y.
Round	Округляет число к ближайшему целому.
Sqr	Вычисляет квадрат аргумента X.
Sqrt	Вычисляет квадратный корень аргумента X.
Trunc	Отсекает дробную часть числа.

Числовые типы данных в Delphi

Целочисленные типы данных

Целочисленные типы данных применяются для описания целочисленных данных. Для решения различных задач могут потребоваться различные целые числа. В одних задачах счет идет на десятки, в других – на миллионы. Соответственно в языке Delphi имеется несколько целочисленных типов данных, среди которых вы можете выбрать наиболее подходящий для своей задачи.

Фундаментальные типы данных:

Тип данных	Диапазон значений	Объем памяти (байт)
Byte	0..255	1
Word	0..65535	2
Shortint	–128..127	1
Smallint	–32768..32767	2
Longint	–2147483648..2147483647	4
Longword	0.. 4294967295	4
Int64	$-2^{63}..2^{63}-1$	8

Обобщенные типы данных:

Тип данных	Диапазон значений	Объем памяти (байт)
Byte	0..255	1
Cardinal	0.. 4294967295	4
Integer	–2147483648..2147483647	4

Переменные обобщенных типов данных могут храниться в памяти по-разному в зависимости от конкретной модели компьютера, и для работы с ними компилятор может генерировать наиболее оптимальный код.

Вещественные типы данных

(применяются для описания вещественных данных с плавающей или с фиксированной точкой):

Тип данных	Диапазон значений	Мантисса	Объем памяти (байт)
Real	$5.0*10^{-324}..1.7*10^{308}$	15–16	8
Real48	$2.9*10^{-39}..1.7*10^{38}$	11–12	6
Single	$1.5*10^{-45}..3.4*10^{38}$	7–8	4
Double	$5.0*10^{-324}..1.7*10^{308}$	15–16	8
Extended	$3.4*10^{-4932}..1.1*10^{4932}$	19–20	10
Comp	–9223372036854775808 .. 9223372036854775807	19–20	8
Currency	–922337203685477.5808 .. 922337203685477.5807	19–20	8

Тип Real является обобщенным типом данных.