Лабораторная работа №1.

Событийное программирование в Delphi.

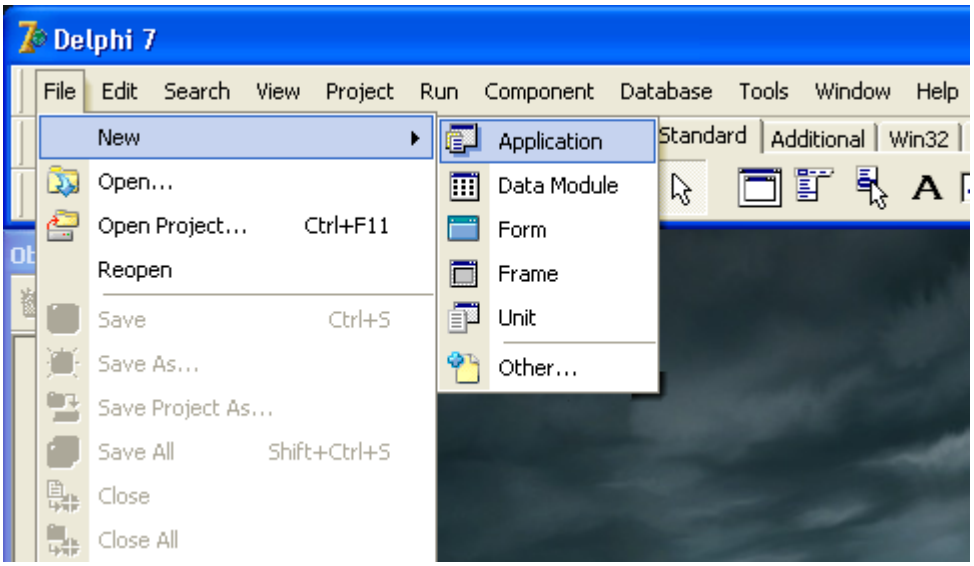
Время: 180 мин.

Что нужно освоить / контрольные вопросы:

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;
6) какие события мыши обрабатываются в Delphi;
7) как использовать обработчики событий мыши.

1. Приложение и его компоненты.

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.

Для того чтобы правильно сохранить разрабатываемое вами приложение, включая все модули и формы, следует выбрать пункт меню File / Save Project As и последовательно сохранить все модули приложения и сам проект в одну папку.

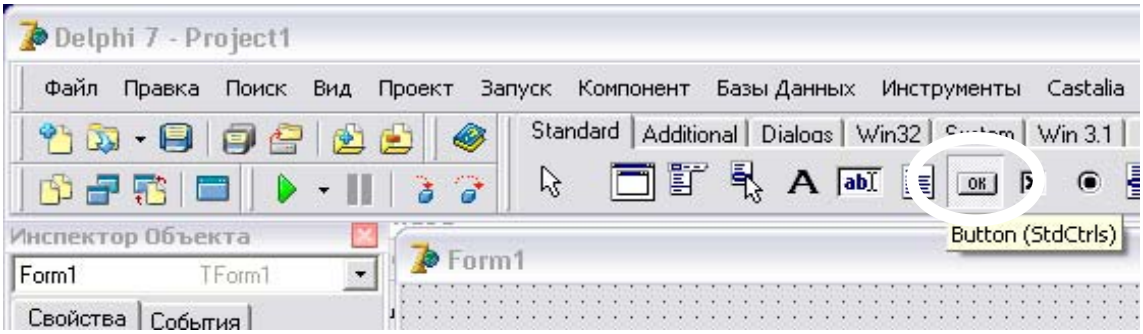
Обратим внимание на раздел подключения модулей. Ключевое слово Uses определяет начало списка модулей, которые используются текущим модулем, программой или библиотекой. Следует заметить, что при создании нового приложения Delphi самостоятельно размещает несколько модулей в разделе подключения модулей 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.

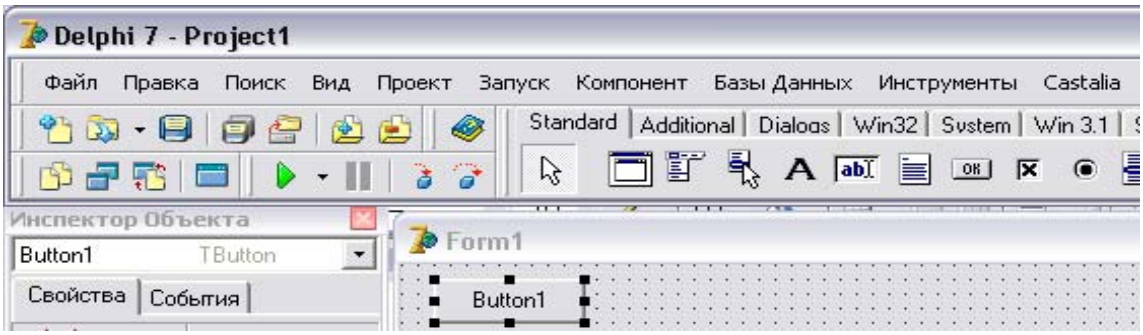
Подключаемые модули Delphi имеют расширение dcu (например, Forms.dcu) и представляют собой откомпилированный модуль проекта в промежуточном формате. Когда компилируется программа, все модули компилируются в файлы формата dcu, а потом собираются в один исполняемый exe файл. Если модуль не изменялся с 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 7 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.

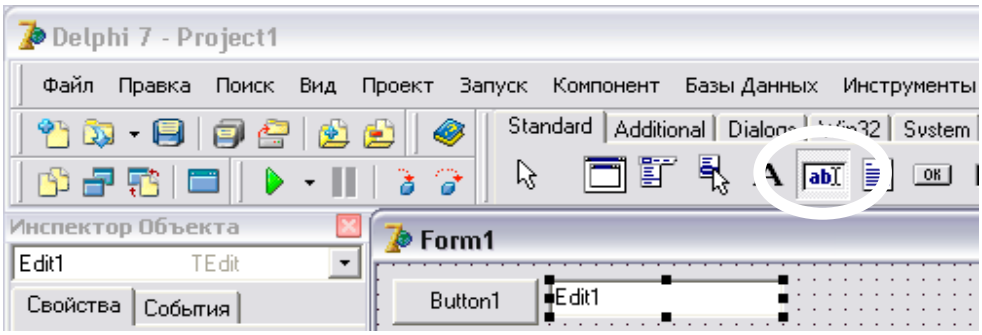




Запустите программу на выполнение – она работает, кнопка нажимается, но пока не выполняет никаких действий, так как мы и не задали никаких обработчиков событий. Однако все эти новые возможности стали доступны благодаря тому, что Delphi самостоятельно подключил еще несколько стандартных модулей – обратите 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 `Button1`, 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 в любом месте 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? Вернемся 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;
```

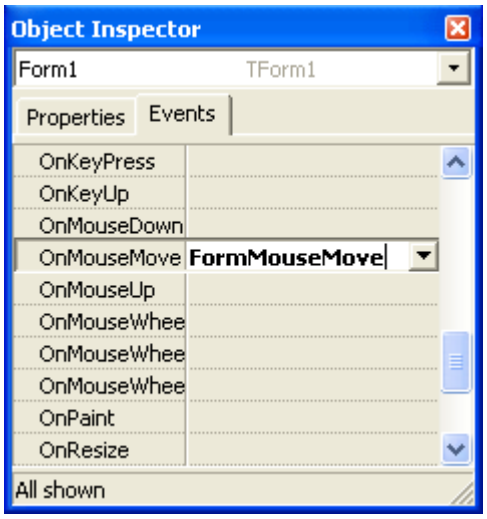
==========

2. События мыши.

Делфи позволяет обрабатывать различные события манипулятора «мышь»: одиночный клик левой, средней или правой клавишей; двойной клик левой клавишей мыши; движение указателя мыши; прокрутка колёсика мыши и т.п. Наличие обработчиков и их возможности зависят от компонентов, к которым привязаны обработчики событий.

Рассмотрим некоторые из событий мыши и попрактикуемся настраивать соответствующие обработчики на разрабатываемом нами приложении.

Создадим обработчик события – движение указателя по главной форме приложения. Пусть координаты указателя во время движения отображаются в текстовом поле Edit1.



Чтобы перейти к форме кликните по неё один раз. В инспекторе объектов (Object Inspector) станут доступными для редактирования свойства (Properties) и события (Events), доступные для формы Form1. Нас интересуют сейчас события мыши. Найдите в списке событий в левом столбце событие OnMouseMove и кликните два раза по правому (пустому) полю. Ваше действие приведет к генерации процедуры обработчика события движение мыши по форме:

```
procedure    TForm1.FormMouseMove(Sender:    TObject;    Shift:
TShiftState; X, Y: Integer);
begin

end;
```

Обратите внимание, что среди параметров процедуры есть X и Y – они содержат текущие координаты указателя мыши в любой момент времени. Их можно выводить на экран в момент движения мыши в пределах границ формы, для чего добавьте в процедуру код:

```
procedure    TForm1.FormMouseMove(Sender:    TObject;    Shift:
TShiftState; X, Y: Integer);
begin
    Edit1.Text:=IntToStr(X)+':'+IntToStr(Y);
end;
```

Теперь создадим процедуру, которая будет нам сообщать какой клавишей мыши кликнул пользователь по форме. Разместите на форме компонент Label с вкладки Standard и настройте его по своему усмотрению. Сгенерируйте обработчик события OnMouseDown. Обратите внимание на переменную Button – она содержит наименование нажатой клавиши мыши и имеет тип TMouseButton. Как же узнать какие именно значения она может приобретать? Для этого нажмите и удерживайте клавишу Ctrl и наведите указа-

тель мыши на описание типа в процедуре (TMouseButton), которое приобретет вид гиперссылки. Кликните на ней и Делфи откроет вам программный код с описанием модуля Controls, в котором вы и увидите перечисление значений типа в круглых скобках:

```
TMouseButton = (mbLeft, mbRight, mbMiddle);
```

Закройте модуль Controls, кликнув правой клавишей мыши по его заголовку и выбрав опцию Close Page.

Заполним процедуру программным кодом, который будет обеспечивать вывод наименования нажатой клавиши в метку Label1:

```
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMou-
seButton;
  Shift: TShiftState; X, Y: Integer);
begin
  with Label1 do
    case Button of
      mbLeft:   Caption:='левая';
      mbMiddle: Caption:='средняя';
      mbRight:  Caption:='правая';
    end;
end;
```

Оператор объединения with в процедуре позволяет избежать излишнего дублирования программного кода и не писать каждый раз Label1.Caption.

Событие OnMouseDown возникает в момент нажатия клавиши мыши, а событие OnMouseUp в момент её отпускания. Сгенерируйте процедуру `FormMouseUp` и заполните кодом:

```
procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouse-
Button;
  Shift: TShiftState; X, Y: Integer);
begin
  Label1.Caption:=Label1.Caption+' отпущена';
end;
```

Испытайте её следующим образом: кликните мышкой по форме и удерживайте её нажатой, оцените содержимое поля Label1, затем отпустите клавишу мыши и посмотрите на изменения. Доработайте код процедуры, используя оператор with.

Теперь поработаем немного с колесиком мыши. Пусть от вращения колесика зависит положение метки Label1 по вертикали. Сгенерируйте процедуры `FormMouseWheelDown` и `FormMouseWheelUp` и заполните их кодом:

```
procedure  TForm1.FormMouseWheelDown(Sender:  TObject;  Shift:
TShiftState; MousePos: TPoint; var Handled: Boolean);
begin
  if Label1.Top<Form1.Height then Label1.Top:=Label1.Top+1;
end;

procedure  TForm1.FormMouseWheelUp(Sender:  TObject;  Shift:
TShiftState; MousePos: TPoint; var Handled: Boolean);
begin
  if Label1.Top>0 then Label1.Top:=Label1.Top-1;
end;
```

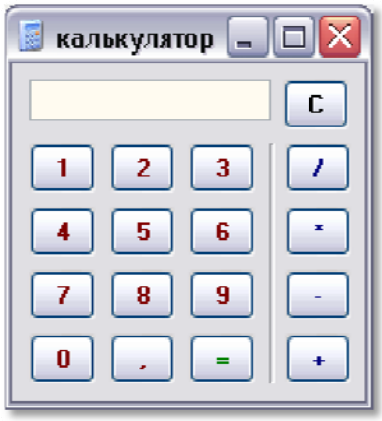
Доработайте процедуры, используя оператор with.

Обратите внимание, что процедуры содержат переменную `Shift`. Она содержит комбинацию нажатых управляющих клавиш (Ctrl, Alt, Shift) в момент вращения колёсиком мыши. Посмотрите самостоятельно описание типа `TShiftState` и определите, какие значения может принимать переменная такого типа. Использование управляющих клавиш позволяет разнообразить возможности разрабатываемой процедуры. Пусть вращение колесика мыши будет двигать метку Label1 слева-направо, если при этом удерживается в нажатом состоянии клавиша Ctrl; сверху-вниз при удержании клавиши Alt; по диагонали – при удержании сразу обеих клавиш.

```
procedure  TForm1.FormMouseWheelDown(Sender:  TObject;  Shift:
TShiftState;
  MousePos: TPoint; var Handled: Boolean);
begin
  with Label1 do
  begin
    if Shift=[ssCtrl] then
      if Left<Form1.Width then Left:=Left+1;
    if Shift=[ssAlt] then
      if Top<Form1.Height then Top:=Top+1;
    if Shift=[ssAlt,ssCtrl] then
    begin
      if Left<Form1.Width then Left:=Left+1;
      if Top<Form1.Height then Top:=Top+1;
    end;
  end;
end;
```

Аналогичным образом заполните процедуру `FormMouseWheelUp`.

3. СОЗДАЕМ ПЕРВОЕ ОСМЫСЛЕННОЕ ПРИЛОЖЕНИЕ – КАЛЬКУЛЯТОР



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

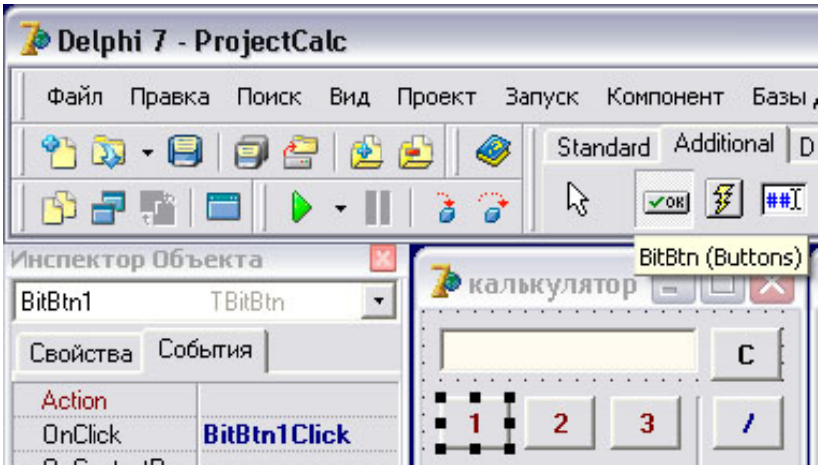
Создайте новое приложение в среде Delphi и сохраните его в отдельной папке.

Часть 1. Разработка интерфейса программы.

Необходимые компоненты:

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

– 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.

В этой части не нужно ничего набирать в текст программного кода. Сейчас мы обсуждаем структуру будущей программы.

Наиболее очевидным можно считать такой алгоритм работы калькулятора:

1) вводим первое число в поле Edit1;
2) нажимаем кнопку действия («+»,«–»,«*»,«/») – первое число из поля Edit1 и наименование действия из Caption только что нажатой экранной клавиши сохраняем для последующего использования в глобальных переменных;
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 процедуру обработчика 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 «+», «–», «/», «*».

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, подготовьтесь к ответам на контрольные вопросы.

Приложение I

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.