МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«САМАРСКИЙ ГОСУДАРСТВЕННЫЙ АЭРОКОСМИЧЕСКИЙ
УНИВЕРСИТЕТ имени академика С.П. КОРОЛЕВА
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)» (СГАУ)

РАБОТА В СРЕДЕ ВИЗУАЛЬНОГО ПРОГРАММИРОВАНИЯ LAZARUS

Рекомендовано редакционно-издательским советом федерального государственного автономного образовательного учреждения высшего образования «Самарский государственный аэрокосмический университет имени академика С.П. Королева (национальный исследовательский университет)» в качестве методических указаний

Составитель М.С. Стенгач

С А М А Р А
Издательство СГАУ
2016

УДК 004.9(075)
ББК 22.18я7

Составитель ***М.С. Стенгач***

Рецензент д-р техн. наук, проф. С. А. Ишков

Работа в среде визуального программирования Lazarus: метод. указания / сост. *М.С. Стенгач*. – Самара: Изд-во СГАУ, 2016. – 28 с.

В методических указаниях приводятся начальные сведения о работе в среде визуального программирования Lazarus.

Предназначены для выполнения курсовых и лабораторных работ по курсу «Информатика» для студентов всех специальностей.

Разработаны на кафедре суперкомпьютеров и общей информатики.

УДК 004.9(075)
ББК 22.18я7

СОДЕРЖАНИЕ

1. Начало работы

Среда визуального программирования Lazarus позволяет с минимальными затратами времени создавать графические и консольные приложения при помощи компилятора Free Pascal. Free Pascal – это компилятор языков Pascal и Object Pascal, работающий под операционными системами Windows, Linux, Mac OS X, FreeBSD.

Поскольку в основе Lazarus лежит концепция быстрого создания приложений, особое внимание в методических указаниях уделено объектно-ориентированной и визуальной технологиям программирования.

После запуска Lazarus на экране появляются четыре основных окна, представленные на рис.1 (версия v1.0.4): *Главное окно* (на рис. 1 это окно, имеющее заголовок *Lazarus IDE v1.0.4 – project 1*), *Форма* (окно, имеющее заголовок *Form1*), *Инспектор объектов* и *Редактор исходного кода*. Переключаться между окнами *Редактор исходного кода* и *Форма* можно с помощью клавиши F12.

1.1. Главное окно

Главное окно Lazarus осуществляет основные функции управления создаваемым проектом. Здесь располагается главное меню Lazarus, набор пиктографических командных кнопок и *Палитра компонентов*. Все элементы главного окна располагаются на специальных панелях.

Пиктографические кнопки занимают левую часть *Главного окна* (рис.1) и открывают быстрый доступ к наиболее важным опциям меню. Наиболее часто используются следующие кнопки:

 - *Сохранить* и *Сохранить всё,*

 - *Запуск программы* (клавиша F9).

Палитра компонентов Lazarus занимает правую часть *Главного окна* (рис.1) и имеет набор страниц, содержащих компоненты.

Standard. Большинство компонентов на этой странице являются широко применяемыми экранными элементами – меню, кнопки, полосы прокрутки.

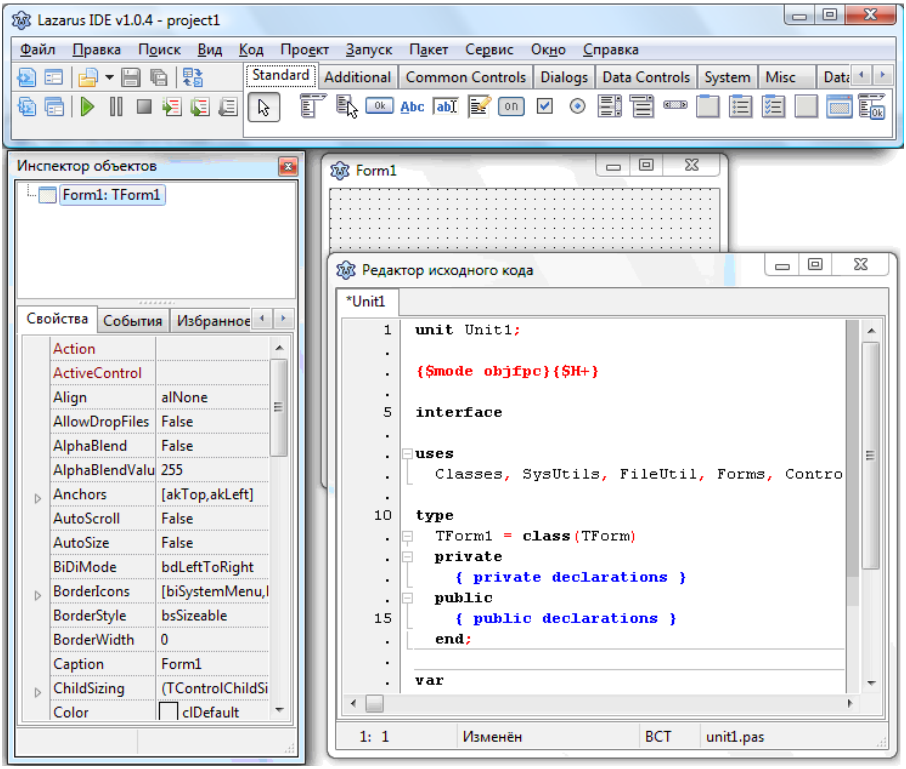


Рис. 1. Основные окна Lazarus

Additional. Эта страница содержит более развитые компоненты. Например, компонент TSpeedButton является специальной кнопкой для инструментальных панелей. Данная страница также содержит компоненты, главное назначение которых – отображение графической информации. Компонент TImage загружает и отображает растровые изображения, а компонент TShape – стандартные фигуры: окружность, квадрат и т.д.

Common Controls. Страница содержит компоненты, позволяющие просматривать древовидные структуры, списки, панель состояния.

Dialogs. Эта страница предоставляет программам Lazarus простой доступ к стандартным диалоговым окнам для операций над файлами, выбором шрифтов и цвета.

System. Компонент TTimer страницы System может генерировать события через определенные, заранее установленные промежутки времени.

Misc. Эту страницу можно рассматривать как продолжение страниц Standard и Additional. Содержит полезные компоненты, такие как, например календарь.

1.2. Форма

Форма представляет собой фундамент программы. Приложение может иметь несколько форм, каждая из которых выполняет свое особое предназначение. Изначально окно формы пустое. Точнее, оно содержит стандартные для Windows элементы интерфейса – кнопки максимизации, минимизации и закрытия окна, полосу заголовка и очерчивающую рамку.

Для размещения компонента на форме требуется щелкнуть на нужной закладке *Палитры компонентов*, а затем на кнопке с пиктограммой соответствующего компонента и после этого щелкнуть в окне формы.

1.3. Инспектор объектов

Размещаемые на форме компоненты характеризуются некоторым набором параметров. Для изменения этих параметров предназначено окно *Инспектора объектов*. Это окно содержит три закладки – *Свойства*, *События* и *Избранное*. Закладка *Свойства* служит для установки нужных свойств компонента, закладки *События* позволяет определить реакцию компонента на то или иное событие. Очень удобна закладка *Избранное*, так как она отображает основные и наиболее часто используемые свойства и события выбранного компонента.

Каждая страница *Инспектора объектов* состоит из двух колонок (см. рис. 1). Левая колонка содержит название свойства или события, а правая – конкретное значение свойства или имя подпрограммы, обрабатывающей соответствующее событие. Если слева от названия свойства стоит значок "+", это означает, что данное свойство определяется совокупностью значений. Щелчок по значку "+" приведет к раскрытию списка составляющих данного свойства.

В верхней части окна *Инспектора объектов* располагается список всех помещенных на форму компонентов.

1.4. Редактор исходного кода

Редактор исходного кода предназначен для создания и редактирования текста программы. Первоначально окно *Редактора исходного кода* содержит минимальный исходный текст, обеспечивающий нормальное функционирование пустой формы в качестве полноценного Windows-окна.

Сразу после создания нового проекта в *Редакторе исходного кода* будет открыт программный модуль, содержащий следующие строки:

```
unit Unit1;

{$mode objfpc}{$H+}

interface

uses
  Classes, SysUtils, FileUtil, Forms, Controls, Graphics,
  Dialogs, StdCtrls;

type
  { TForm1 }

  TForm1 = class(TForm)
    Label1: TLabel;
  private
    { private declarations }
  public
    { public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.lfm}

end.
```

Рис. 2. Исходный текст программного модуля в *Редакторе исходного кода*

При размещении компонентов на форме и при создании событий Lazarus автоматически добавляет нужный код в текст модуля.

1.5. Размещение компонентов на форме

Размещать компоненты на форме очень просто. Требуется только щелкнуть на нужной вкладке *Палитры компонентов*, затем на кнопке с пиктограммой соответствующего компонента и после этого щелкнуть в окне формы. Можно также щелкнуть на компоненте, а затем нарисовать прямоугольник с помощью мыши на форме – компонент появится внутри этого прямоугольника. Если размеры компонента поддаются изменению, то при появлении на форме он заполнит собой прямоугольник.

Если щелкнуть на каком-либо компоненте в *Палитре компонентов*, то его кнопка окажется нажатой. При выборе другого компонента кнопка предыдущего компонента вернется в исходное состояние, так как одновременно не может быть выбрано несколько компонентов. Для того чтобы все кнопки оказались в исходном состоянии и было восстановлено нормальное использование мыши, следует щелкнуть на кнопке со стрелкой выбора, которая появляется с левой стороны каждой страницы *Палитры компонентов* (см. рис. 1).

Пример размещения компонента на форме:

1. Выберите страницу **Standard** *Палитры Компонентов*.
2. Поместите указатель мыши на кнопку  . После того как мышь побудет в недвижимом состоянии секунду или две, появится поле помощи, идентифицирующее этот компонент как `TLabel`.
3. Щелкните на кнопке, затем щелкните на вашей форме. На форме появится надпись `Label1`.
4. Обратите внимание, что поле надписи на форме имеет небольшие квадратные маркеры (угловые и боковые), позволяющие изменять размер компонента (но не размер самой надписи).
5. Для изменения стандартной надписи `Label1` в строке *Caption* на закладке *Свойства* окна *Инспектора объектов* введите надпись «Привет!».
6. Для изменения цвета и шрифта надписи щелкните мышью по свойству *Font* на закладке *Свойства* окна *Инспектора объектов* и с помощью кнопки в правой части строки раскройте диалоговое окно настройки шрифта. В списке *Размер* этого окна

выберите высоту шрифта 24 пункта, а с помощью списка *Цвет* выберите нужный цвет.

Щелкните левой кнопкой мыши на надписи «Привет!» и перетащите ее в новое место. Видно, что при перемещении и изменении размера компоненты выравниваются по точкам координатной сетки формы. Такая возможность помогает поддерживать порядок размещения компонентов на форме.

Не каждый компонент виден на форме во время запуска программы. Например, размещение на форме компонента *MainMenu* приводит к появлению в разрабатываемом приложении меню, но соответствующая пиктограмма во время запуска программы не отображается. Компоненты, представляющие диалоговые окна общего назначения, вообще никак не визуализируются во время работы программы. Размеры невидимого компонента в процессе разработки не изменяются – он всегда отображается в виде пиктограммы.

2. Объектно-ориентированное программирование

Двумя фундаментальными концепциями всех объектно-ориентированных языков (Object Pascal, C++) являются объекты и классы.

Объект – фрагмент программы, предназначенный для решения отдельной задачи.

Объектно-ориентированный язык Object Pascal, который является основным инструментом Lazarus, разработан на основе популярного языка программирования Pascal. По этой причине все основные элементы этих двух языков практически совпадают. Это относится к операторам языка, типам данных, выражениям, операциям, процедурам и функциям.

2.1. Классы

Классы предназначены для создания объектов.

Структура описания класса:

```
type
  <имя класса> = class
  <описание переменных и подпрограмм>
  end;
```

Взаимосвязь класса с объектом подобна взаимосвязи типа данных с переменной. Например, в следующем примере создается целая переменная с именем `data`:

```
var data: integer;
```

Аналогичная связь существует между классом и объектом. Объявление переменной типа `class` в блоке описания переменных называется созданием объекта:

```
var <имя_переменной>: <имя_класса>;
```

Пример:

```
type
  TMyClass = class
    a: integer;
  end;                    // описание класса
var  myobj: TMyClass;     // создание объекта
```

Следует помнить, что все рабочие данные хранятся в объекте. Класс не содержит никаких данных – он лишь описывает общую структуру объекта.

По используемому в Object Pascal соглашению все имена классов начинаются с буквы T.

2.1.1. Основные понятия класса

В основе классов лежат три фундаментальных принципа, которые называются *инкапсуляция, наследование* и *полиморфизм*.

Инкапсуляция. Класс представляет собой единство трех сущностей – полей (переменных), методов (подпрограмм) и свойств. Объединение этих сущностей в единое целое и называется инкапсуляциой. Инкапсуляция позволяет сделать класс самодостаточным для решения конкретной задачи. Например, класс `TForm` содержит в себе (инкапсулирует в себе) всё необходимое для создания Windows-окна. Класс `TMemo` представляет собой полноценный текстовый редактор, класс `TTimer` обеспечивает работу программы с таймером.

Наследование. Любой класс может быть порожден от другого класса. Для этого при его объявлении указывается имя класса-родителя:

```
TChildClass = class(TParentClass)
```

Порожденный класс автоматически наследует поля, методы и свойства своего родителя и может дополнять их новыми.

Все классы Object Pascal порождены от единственного родителя – класса `TObject`. Этот класс не имеет полей и свойств, но включает в себя методы самого общего назначения – от создания до уничтожения объектов.

Все новые классы будут дочерними классу `TObject`. Следующие два объявления идентичны:

```
TMyClass = class(TObject)
TMyClass = class
```

Принцип наследования привел к созданию в Lazarus дерева классов, где каждый потомок дополняет возможности своего родителя новыми.

Полиморфизм – это свойство классов решать сходные по смыслу задачи разными способами. Можно изменить (*перекрыть*) метод

родителя в потомке. Для этого необходимо объявить в потомке одноименный метод и реализовать в нем нужные свойства. В результате в родителе и потомке будут действовать два одноименных метода, но с разными свойствами.

2.1.2. Составляющие класса

Поля. Полями называются инкапсулированные в классе данные. Поля могут быть любого типа, в том числе и классами, например:

```
type TMyClass = class
     a:   integer;
     obj: TObject;
     end;
```

Методы. Инкапсулированные в классе процедуры и функции называются методами.

Методы в классе объявляются так же, как и обычные подпрограммы (процедуры и функции):

```
function <метод1(список параметров)>: <тип>;
procedure <метод2(список параметров)>;
```

Описание процедур и функций, реализующих методы, помещается после слова `implementation` (см. рис. 2) того модуля, где объявлен класс, и выглядит так:

```
function <имя_класса>.<метод1(список пар.)>:<тип>;
   begin
      <тело функции>;
   end;
procedure <имя_класса>.<метод2(список пар.)>;
   begin
      <тело процедуры>;
   end;
```

После описания переменной в программе можно обращаться к полям и методам класса аналогично обращению к полям структуры, используя оператор «`.`» (точка).

Например:

```
<имя_переменной>.<поле1>:=<выражение>;
<имя_переменной>.<метод1(список параметров)>;
```

Свойства. Свойства – это специальный механизм классов, регулирующий доступ к полям. Свойства объявляются с помощью зарезервированных слов `property`, `read` и `write`.

2.1.3. Объявления класса

Класс может содержать разделы: `private`, `public` (см. рис. 2), `published`, и `protected`. Внутри разделов сначала определяются поля, а затем – методы и свойства.

Разделы определяют области видимости элементов. Элементы раздела `public` можно вызвать в любом другом модуле программы. Элементы раздела `private` доступны только внутри методов данного класса и в подпрограммах, находящихся в том же модуле, где описан класс. В разделе `published` перечисляются свойства, доступные на этапе конструирования программы (т.е. в окне *Инспектора объектов*). Раздел `protected` доступен только методам самого класса, а также методам потомков этого класса.

2.1.4. Класс TForm и реакция на события

Проведем модернизацию нашей первой программы (см. пункт 1.5): поместим на ее форму еще один компонент – кнопку и заставим ее откликаться на событие, связанное с нажатием левой клавиши мыши. Компонент «кнопка» изображается пиктограммой на странице **Standard** палитры компонентов как кнопка 

(см. рис. 1). Щелкните по этой кнопке, а затем – на форме. Появится кнопка с надписью `Button1`.

При щелчке по кнопке мышью в работающей программе возникает событие `OnClick` (по щелчку). Пока это событие никак не обрабатывается программой, и поэтому «нажатие» кнопки не приведет ни к каким последствиям. Чтобы заставить программу реагировать на нажатие кнопки, необходимо написать на языке Object Pascal фрагмент программы, который называется *обработчиком события*. Этот фрагмент программы оформляется в виде процедуры.

Чтобы заставить Lazarus самостоятельно сделать заготовку для процедуры обработчика события `OnClick`, дважды щелкните мышью по вновь вставленному компоненту (кнопке). В ответ Lazarus активи-

зирует окно *Редактора исходного кода*, и вы увидите в нем такой текстовый фрагмент:

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

end;
```

Попробуем разобраться, что он содержит. Слово `procedure` извещает компилятор о начале подпрограммы-процедуры. За ним следует имя процедуры `TForm1.Button1Click`. Имя процедуры составное: оно состоит из имени класса `TForm1` и собственно имени процедуры `Button1Click.`

В состав Object Pascal входит несколько сотен стандартных классов. Каждый компонент принадлежит к строго определенному классу, а все конкретные экземпляры компонентов, вставляемые в форму, получают имя класса с добавленным числовым индексом.

Таким образом, имя `TForm1` означает имя класса, созданного по образцу стандартного класса `TForm`. Если посмотреть начало текста в окне кода, то мы увидим следующие строки:

```
type
  { TForm1 }
  TForm1 = class(TForm)
    Button1: TButton;
    Label1: TLabel;
    procedure Button1Click(Sender: TObject);
  private
    { private declarations }
  public
    { public declarations }
  end;
var
  Form1: TForm1;
```

Строка

```
TForm1 = class(TForm)
```

определяет новый класс `TForm1`, который *порожден от* (создан по образцу) стандартного класса `TForm`.

Строка

```
Form1: TForm1;
```

создает *объект* этого класса с именем `Form1`. Стандартный класс `TForm` описывает пустое Windows-окно, в то время как `TForm1` описывает окно с уже вставленными в него компонентами *метка* и *кнопка*. Описание этих компонентов содержат строки:

```
Button1: TButton;
Label1: TLabel;
```

Они указывают, что кнопка `Button1` представляет собой *экземпляр* компонента `TButton`, а метка `Label1` – экземпляр компонента `TLabel`.

Кроме того, класс `TForm1` содержит процедуру `Button1Click`, на что указывает строка:

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

За именем процедуры `Button1Click` в круглых скобках следует описание вызова `Sender: TObject` (параметр с именем `Sender` принадлежит классу `TObject`). С помощью параметра `Sender` подпрограмма `Button1Click` может при желании определить, какой именно компонент создал событие `OnClick`.

Само тело процедуры `TForm1.Button1Click(Sender: TObject);` (т.е. строки между словами `begin…end;`) пока еще не содержит описания каких-либо действий, т.к. Lazarus лишь создал заготовку для процедуры. Наполнить тело нужными операторами – задача программиста.

Каждый раз при нажатии кнопки `Button1` управление будет передаваться в тело процедуры, а значит, между словами `begin` и `end` мы можем написать фрагмент программы, который будет выполняться в ответ на это событие.

Чтобы убедиться в этом, напишите в пустой строке между словами `begin` и `end` следующее предложение:

```
Form1.Color:=clRed;
```

и сделайте прогон программы (кнопка  или клавиша F9). Теперь в ответ на нажатие кнопки форма окрасится в красный цвет.

2.2. Структура проекта Lazarus

Любой проекта в Lazarus – это совокупность файлов, из которых создается единый выполняемый файл. В простейшем случае список файлов проекта имеет вид:

• файл описания проекта (`.lpi`);

• файл проекта (`.lpr`);

• файл ресурсов (`.lrs`);

• модуль формы (`.lfm`);

• программный модуль (`.pas`);

После компиляции программы из всех файлов проекта создается единый выполняемый файл с расширением `.exe`, имя этого файла совпадает с именем проекта.

Программный модуль, или просто модуль (файл с расширением `.pas`), – это отдельно компилируемая программная единица, которая представляет собой набор типов данных, констант, переменных, процедур и функций. Любой модуль имеет следующую структуру:

```
unit <имя модуля>; //Заголовок модуля
interface
//Раздел описаний
implementation
//Раздел реализаций
end.
```

Заголовок модуля – это зарезервированное слово `unit`, за которым следует имя модуля и точка с запятой. В разделе описаний, который открывается служебным словом `interface`, описывают программные элементы – типы, классы, процедуры и функции.

Раздел `implementation` содержит программный код, реализующий механизм работы описанных программных элементов (тексты процедур обработки событий, а также процедуры и функции, созданные программистом).

Изначально проект Lazarus содержит один модуль (файл с именем `Unit1.pas`), исходный текст которого показан на рис. 2. При необходимости программист может добавлять новые модули к проекту.

Чтобы увидеть файла проекта (файл с расширением `.lpr`), необходимо в главном меню выбрать `Проект/Посмотреть исходный код проекта`. В окне *Редактора исходного кода* появится новая закладка `project1`, содержащая следующий текст:

```
program project1;
{$mode objfpc}{$H+}
uses
  {$IFDEF UNIX}{$IFDEF UseCThreads}
  cthreads,
  {$ENDIF}{$ENDIF}
  Interfaces, // this includes the LCL widgetset
  Forms, Unit1
  { you can add units after this };

{$R *.res}

begin
  RequireDerivedFormResource := True;
  Application.Initialize;
  Application.CreateForm(TForm1, Form1);
  Application.Run;
end.
```

В разделе `uses` мы видим несколько подключаемых модулей: `cthreads`, `Interfaces`, `Forms` – это стандартные модули. Модуль `Unit1` – это новый модуль, который создается автоматически при создании нового проекта Lazarus.

В теле файла проекта происходит обращение к методам объекта `Application`. В этом объекте собраны данные и подпрограммы, необходимые для функционирования Windows-программы. Метод `Create Form` создает и показывает на экране окно главной формы, а метод `Run` реализует бесконечный цикл получения и обработки поступающих от операционной системы сообщений о действиях пользователя.

3. Компоненты Lazarus

В этом параграфе мы рассмотрим наиболее популярные компоненты Lazarus.

3.1. Компоненты страницы Standard

TMainMenu – главное меню. После размещения компонента TMainMenu на форме необходимо задать его пункты. Для этого следует дважды щелкнуть по компоненту левой кнопкой мыши. На экране покажется окно редактора меню. Первый пункт меню создается автоматически и носит название `New Item1`. Чтобы создать новый пункт меню, нужно щелкнуть правой кнопкой мыши по пункту `New Item1` и выбрать *Вставить новый пункт*. В *Окне инспектора объектов* в строке `Caption` вводится нужное название пункта. Для создания у пункта раскрывающегося меню необходимо щелкнуть правой кнопкой мыши по этому пункту и выбрать *Создать подменю*. Если в свойстве `Caption` задать символ "-", то в списке подменю появится разделительная черта.

TPopupMenu – локальное меню, появляющееся после нажатия правой кнопки мыши. Может быть создано для любого компонента, для этого в свойстве `PopupMenu` этого компонента необходимо поместить имя компонента-меню.

TButton – кнопка. В пункте 2.1.4 уже был рассмотрен пример размещения кнопки на форме, а также создания отклика на событие, связанного с нажатием левой клавиши мыши. Чтобы событие `OnClick` возникало также и при нажатии клавиши Enter, необходимо свойство `Default` задать `True`.

В Windows часто используются *модальные окна*. Модальное окно – это окно, блокирующее работу с другими окнами вплоть до своего закрытия. Обычно в состав модального окна включается несколько кнопок. Если у кнопки свойство `ModalResult` не равно `mrNone`, нажатие на нее приведет к закрытию модального окна.

В отличие от большинства других компонентов кнопка является компонентом самой Windows и поэтому не может изменять свой цвет произвольным образом – она его меняет вместе с изменением палитры самой Windows.

TLabel – метка для отображения текста. В пункте1.5 мы уже подробно рассмотрели размещение этого компонента на форме.

TEdit – ввод и отображение строки. Компонент класса TEdit представляет собой однострочный редактор. Главным свойством компонента является свойство `Text`, отображающее строку. Так как строка имеет тип `String`, для ввода чисел необходимо воспользоваться следующими стандартными функциями:

`StrToFloat(s: String): real;` – преобразует строку `s` в вещественное число,

`StrToInt(s: String): integer;` – преобразует строку `s` в целое число.

Например:

```
var x: real;
x:= StrToFloat(Edit1.Text);
```

Чтобы отобразить в компоненте число, необходимо проделать обратную операцию:

```
Edit1.Text:= FloatToStr(x);
```

TMemo – ввод и отображение текста. Компоненты данного класса предназначены для ввода и редактирования длинного текста. Текст хранится в свойстве `Lines` и представляет собой пронумерованный набор строк. При работе с компонентом TMemo наиболее часто употребляются следующие операции:

`Memo1.Lines[i:integer];` – доступ к i-й строке (нумерация строк начинается с нуля);

`Memo1.Lines.Count;` – общее количество строк;

`Memo1.Lines.add(s: string);` – добавить новую строку `s` в список;

`Memo1.Clear;` – очистить содержимое компонента.

Компонент имеет также свойство `Text`, которое, в отличие от `Lines`, содержит текст в виде длинной строки.

ToggleBox – кнопка с фиксацией нажатого состояния. Состояние кнопки определяет логическое свойство `Checked`.

TCheckBox – независимый переключатель или флаг выбора. В составе диалогового окна их может быть несколько, и состояние любого из них не зависит от состояния остальных. Логическое свойство `Checked` определяет наличие флага выбора:

`if CheckBox1.Checked then …` – если флаг установлен,

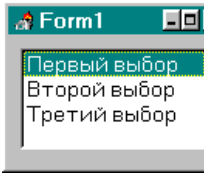
`if not CheckBox1.Checked then …` – если флаг не установлен.

Можно использовать свойство `State`:

```
case CheckBox1.State of
cbChecked:    … - да,
cbUnChecked: … - нет,
cbGrayed:     … - не совсем – окошко компонента полностью закрашивается.
```

TRadioButton – зависимые переключатели. На форме имеет смысл размещать минимум два компонента `TRadioButton.` Если в одном из компонентов свойство `Checked` принимает значение `True`, то у других компонентов оно принимает значение `False`.

TListBox – список выбора.



В списке предусмотрена возможность программной прорисовки элементов, поэтому список может содержать не только строки, но и произвольные изображения. Для ввода пунктов меню необходимо выбрать свойство Items. После того, как меню сформировано, двойным щелчком на компоненте создается реакция на событие:

```
procedure ListBox1Click(Sender: TObject);
```

Количество строк в списке определяется с помощью свойства `Count: integer;,` например:

```
M:=ListBox1.Items.Count;
```

В данном примере `M` будет равно 3.

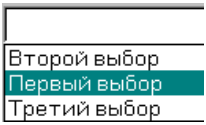
Номер выбранного пункта определяется свойством `ItemIndex`:

```
I:=ListBox1.ItemIndex;
```

Номер первой строки равен 0, второй – 1 и т.д. В случае выбора второго пункта списка возможна такая конструкция:

```
if ListBox1.ItemIndex=1 then … ;
```

TComboBox – раскрывающийся список выбора. Пункты меню также задаются в свойстве `Items`.



В свойстве `Style` можно задать следующие модификации компонента:

`CsDropDown` (по умолчанию) – раскрывается после нажатия кнопки;

`CsSimple` – список всегда раскрыт.

TScrollBar – управление значением величины. 
Свойства компонента:

`Max: integer;` – максимальное значение диапазона изменения числовой величины;

`Min: integer;` – минимальное значение диапазона изменения числовой величины;

`Position: integer;` – текущее значение числовой величины.

TGroupBox и **TPanel** – панели, предназначены для размещения в них групп элементов.

TRadioGroup – контейнер зависимых переключателей. Чтобы создать в контейнере переключатели, следует использовать свойство `Items`. Свойство `ItemIndex` по умолчанию имеет значение –`1` (не выбран ни один переключатель). Если значение `ItemIndex` равно `0`, то выбран первый зависимый переключатель, если равно `1`, то второй и т. д. В случае выбора второго зависимого переключателя возможна такая конструкция:

```
if RadioGroup1.ItemIndex=1 then…
```

3.2. Компоненты страницы Additional

TBitBtn – кнопка с изображением – разновидность кнопки `TButton`. Её отличительная особенность – свойство `Glyph`, с помощью которого определяется изображение, рисуемое на поверхности кнопки. Свойство `Kind` определяет одну из 13 стандартных разновидностей кнопки. Например:

– свойство `Kind` равно `bkOK;`

Нажатие любой из этих 13 кнопок (кроме `bkCustom` и `bkHelp`) закрывает модальное окно и возвращает в программу результат

mrИмякнопки, например: bkOk-mrOk, bkCancel-mrCancel. У кнопок bkOk или bkYes свойство Default автоматически устанавливается True. Кнопка bkClose закрывает главное окно программы. Кнопка bkHelp автоматически вызывает раздел справочной службы, связанный с HelpContext формы, на которую она помещена.

TSpeedButton – кнопка для инструментальных панелей.

Особенности этих кнопок:

1) могут фиксироваться в утопленном состоянии;
2) не могут закрывать модальное окно;
3) не могут быть умалчиваемыми (отсутствует свойство Default).

Для фиксации кнопки свойство GroupIndex должно быть неравным 0. Если у нескольких кнопок свойство GroupIndex имеет одно и то же значение, то эти кнопки принадлежат одной группе.

Логическое свойство AllowAllUp определяет поведение кнопки:

– если AllowAllUp=False - утопленная кнопка отпускается только при нажатии другой кнопки, входящей в ту же группу,

– если AllowAllUp=True - кнопку можно освободить повторным щелчком.

TStaticText – аналогичен метеке TLabel и используется для отображения различных надписей на форме. Размер меток TLabel и TStaticText может определяться свойством AutoSize. Если это свойство установлено в True, то вертикальный и горизонтальный размеры компонента определяются размером надписи. Если же AutoSize равно False, то выравнивание текста внутри компонента определяется свойством Alignment, которое позволяет выравнивать текст по левому краю, правому краю или центру клиентской области метки.

TImage – отображение картинок. Служит для размещения на форме изображений. Свойство Picture позволяет отыскать файл с нужным изображением и поместить это изображение на форме.

TBevel – декоративная рамка. Свойство Style определяет стиль рамки – вдавленная или выпуклая, а свойство Shape определяет её внешний вид.

TShape – прорисовка стандартных фигур. В свойстве `Shape` имеется следующий набор стандартных фигур:

`stCircle` – круг;

`stDiamond` – ромб;

`stEllipse` – эллипс;

`stRectangle` – прямоугольник;

`stRoundRect` – скругленный прямоугольник;

`stRoundSquare` – скругленный квадрат;

`setSquare` – квадрат;

`setTriangle` – треугольник.

Свойства `Brush` и `Pen` позволяют менять цветовое оформление компонента.

TMaskEdit – специальный редактор ввода текста по шаблону, задаваемому свойством `EditMask: String`.

TStringGrid – таблица строк, предназначена для создания таблиц. Таблица делится на две части – фиксированную (служит для показа заголовков столбцов и рядов) и рабочую.

Главное свойство компонента: `Cells` – двумерный массив ячеек, где первый номер соответствует столбцу, а второй – ряду. Например:

```
Cells[0,0]:='Левая верхняя ячейка';
```

Свойства компонента:

`ColCount` – общее количество столбцов в таблице;

`RowCount` – общее количество строк в таблице;

`FixedCols` – количество фиксированных столбцов;

`FixedRows` – количество фиксированных строк.

Чтобы можно было редактировать текст в таблице, необходимо к установленным опциям таблицы добавить опцию `goEditing`:

```
StringGrid1.Options:=StringGrid1.Options+[goEditing];
```

TScrollBox – панель с прокруткой. Служит контейнером для размещения других компонентов. Использование компонента не отличается сложностью: положите его на форму и разместите затем на нем другие компоненты. Если очередной компонент выйдет за пределы рабочей области, по сторонам контейнера возникнут полосы прокрутки.

3.3. Компоненты страницы Common Controls

TTrackBar – компонент представляет собой элемент управления в виде ползунка, который можно перемещать курсором мыши или клавишами во время выполнения. Таким образом можно управлять какими-то процессами: громкостью звука, размером изображения и т.п.

Основное свойство компонента `Position`. При перемещении ползунка можно прочитать значение `Position`, характеризующее позицию, в которую переместили ползунок. Для возможности такого чтения служит событие `OnChange`. В обработчике этого события можно прочитать значение `Position` и использовать его для управления каким-то компонентом.

Свойство `Position` целое, значение которого может изменяться в пределах, задаваемых свойствами `Min` и `Max`. По умолчанию `Min=0`, `Max=10`, так что `Position` может принимать только 11 значений – от 0 до 10. Если задать большее значение `Max`, соответственно увеличится количество возможных значений `Position` в диапазоне `Min` – `Max`.

Свойство `Orientation` определяет ориентацию ползунка: `trHorizontal` – горизонтальная, `trVertical` – вертикальная.

Свойство `TickMarks` указывает размещение шкалы относительно компонента и может принимать значения: `tmBottomRight` – снизу или справа в зависимости от ориентации компонента, `tmTopLeft` – сверху или слева в зависимости от ориентации компонента, `tmBoth` – с обеих сторон.

Свойство `TickStyle` определяет способ изображения шкалы.

TProgressBar – компонент предназначен для отображения хода выполнения длительного по времени процесса. Главные свойства компонента: `Position`, `Max` и `Min` аналогичны подобным свойствам компонента TTrackBar.

Следующий оператор демонстрирует работу компонента `TProgressBar`:

```
for i:=1 to 100 do ProgressBar1.Position:=i;
```

4. Пример работы в Lazarus

Напишем приложение, вычисляющее значение гипотенузы. Результат работы программы представлен на рис. 3.

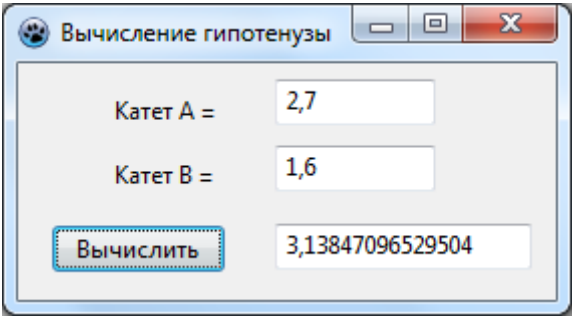


Рис. 3. Приложение Lazarus, вычисляющее значение гипотенузы

1. Запустите Lazarus.
2. Выполните команду `Создать проект...` из пункта главного меню `Проект`.
3. Выберите пункт `Приложение` и нажмите `OK`.
4. Выполните команду `Сохранить проект как...` из пункта главного меню `Проект`.
5. Нажмите кнопку `Сохранить`.
6. Ещё раз нажмите кнопку `Сохранить`.
7. Выберите страницу `Standard` на *Палитре компонентов*.
8. Поместите указатель мыши на кнопку компонента `TLabel` (кнопка в старых версиях).
9. Щелкните по этой кнопке, а затем щелкните на форме (переключение между формой и *Окном кода* – клавиша F12). На форме появится надпись `Label1`.
10. В строке `Caption` окна *Инспектора объектов* измените стандартную надпись `Label1` на надпись `Катет A =.`
11. Аналогично пунктам 2…4 создайте надпись `Катет B =.`
12. Поместите указатель мыши на кнопку на *Палитре компонентов* (кнопка компонента `TEdit`).

13. Щелкните по кнопке, затем щелкните на вашей форме. На форме появится поле ввода данных `Edit1`.
14. Удалите в свойстве `Text` окна *Инспектора объектов* текст `Edit1`.
15. Аналогично пунктам 6…8 разместите на форме компоненты `Edit2` и `Edit3`.
16. Щелкните по форме в любом месте вне размещенных на ней компонентов.
17. В свойстве `Caption` окна *Инспектора объектов* измените название формы `Form1` на `Вычисление гипотенузы`.
18. Поместите указатель мыши на кнопку 

на *Палитре компонентов* (кнопка компонента `TButton`).
19. Щелкните по этой кнопке, а затем щелкните на форме. На форме появится кнопка с надписью `Button1`.
20. Поменяйте название кнопки на `Вычислить` в свойстве `Caption` окна *Инспектора объектов*.
21. Перейдите в окне *Инспектора объектов* на страницу `События`.
22. Дважды щелкните мышью по пустому полю правее пункта `OnClick`. В ответ Lazarus автоматически создаст процедуру обработки события `OnClick`:

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

23. Напишите в процедуре фрагмент программы ввода исходных данных, вычисления и вывода на экран результатов расчета (вставляемые операторы выделены):

```
procedure TForm1.Button1Click(Sender: TObject);
var a,b,c:real;
begin
a:=StrToFloat(Edit1.Text);
b:=StrToFloat(Edit2.Text);
c:=sqrt(sqr(a)+sqr(b));
Edit3.Text:=FloatToStr(c);
end;
```

24. Запустите проект на выполнение (F9).

Список литературы

1. Алексеев, Е.Р. Самоучитель по программированию на Free Pascal и Lazarus / Е.Р. Алексеев, О.В. Чеснокова, Т.В. Кучер. – Донецк: ДонНТУ, Технопарк ДонНТУ УНИТЕХ, 2011. – 503 с.

2. Алексеев, Е.Р. Программирование на Microsoft Visual C++ и Turbo C++ Explorer / Е.Р. Алексеев. – М.: НТ Пресс, 2007. – 352 с.

3. Фаронов, В.В. Delphi. Программирование на языке высокого уровня: учебник для вузов / В.В. Фаронов. – СПб.; М.; Нижний Новгород: Питер: Питер принт, 2005. – 639 с.

4. http://www.beluch.ru/progr/100comp/3_3_3.htm

Учебное издание

РАБОТА В СРЕДЕ ВИЗУАЛЬНОГО ПРОГРАММИРОВАНИЯ LAZARUS

Методические указания

Составитель ***Михаил Сергеевич Стенгач***

Редактор Ю.Н. Л и т в и н о в а
Доверстка Т.С. З и н к и н а

Подписано в печать 11.04.2016. Формат 60×84 1/16.
Бумага офсетная. Печать офсетная. Печ. л. 1,75.
Тираж 100 экз. Заказ . Арт. - 44/2016.

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«САМАРСКИЙ ГОСУДАРСТВЕННЫЙ АЭРОКОСМИЧЕСКИЙ
УНИВЕРСИТЕТ ИМЕНИ АКАДЕМИКА С. П. КОРОЛЕВА
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)» (СГАУ)
443086, САМАРА, МОСКОВСКОЕ ШОССЕ, 34.

Изд-во Самарского государственного
аэрокосмического университета им. академика С.П. Королева.
443086, Самара, Московское шоссе, 34.