ЛАБОРАТОРНАЯ РАБОТА №1. СРЕДА LAZARUS

1. ЦЕЛЬ РАБОТЫ

Целью лабораторной работы является приобретение умений и навыков практического освоения современных технологий визуального объектно-ориентированного программирования в свободно распространяемой среде разработки приложений `Lazarus`.

2. ОПИСАНИЕ СРЕДЫ РАЗРАБОТЧИКА ПРИЛОЖЕНИЙ LAZARUS

Чтобы установить Lazarus на ваш компьютер нужно зайти на http://www.freepascal.ru и закачать последнюю стабильную версию Lazarus. В процессе установки следует согласиться с условиями лицензионного соглашения. После установки в меню программ появится новая группа Lazarus.

Запустить интегрированную среду разработки (ИСР) `Lazarus` можно с помощью команды Главного меню Windows `Пуск` → `Все Программы` → `Lazarus` → `Lazarus` (рис.1).

Рис. 1. Группа Lazarus

После запуска `Lazarus` на экране компьютера появляется основное окно ИСР (рис. 2)

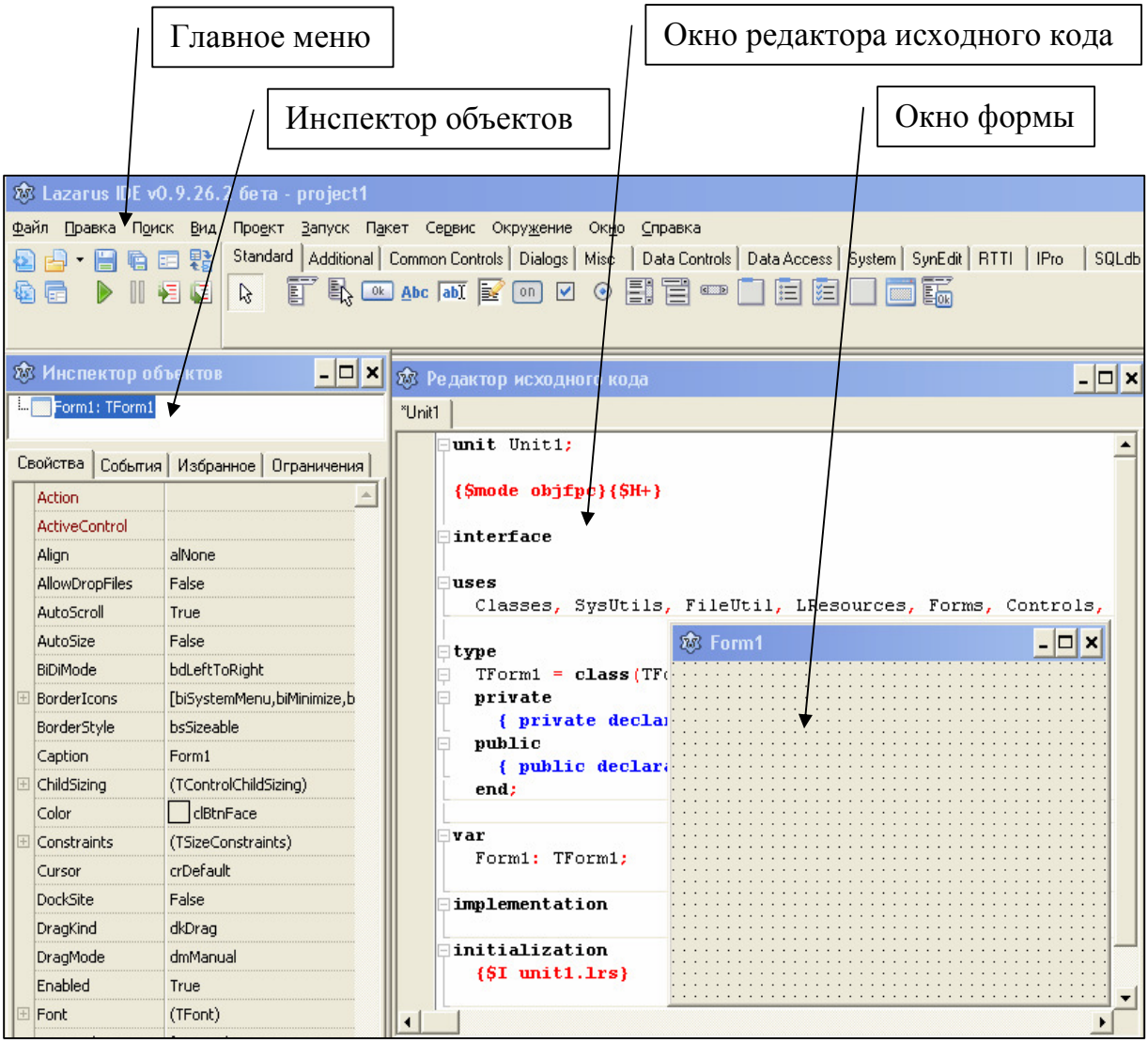


Рис. 2. Среда Lazarus

В верхней части окна ИСР отображается полоса главного меню. Ниже две инструментальные панели:

- Левая панель содержит два ряда кнопок, дублирующих некоторые наиболее часто используемые команды меню.

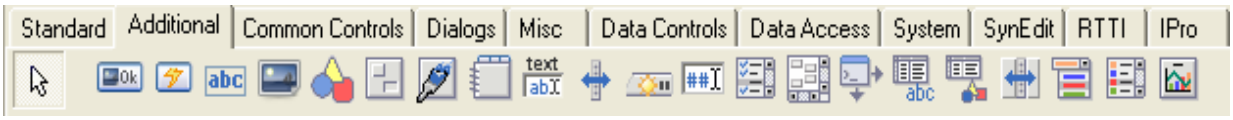
- Правая панель содержит панель библиотеки визуальных компонентов (`Visual Component Library – VCL`), в дальнейшем просто *палитра компонентов*.

Палитра компонентов позволяет выбрать с помощью иконок визуальные и другие компоненты, из которых, как из «строительных блоков», собирается разрабатываемое `Lazarus`-приложение. Назначение некоторых, наиболее часто используемых при проектировании форм компонентов представлено в приложении 1.

Палитра компонентов содержит ряд страниц, закладки которых видны в ее верхней части. Наиболее употребляемые из них: `Standard` (стандартные компоненты) и `Additional` (дополнительные компоненты).

Стандартные компоненты `Standard` реализуют интерфейсные элементы среды Windows. Среди них: главное меню `TMainMenu`, всплывающее меню `TPopupMenu`, метка `TLabel`, текстовое поле `TEdit`, флажок `TCheckBox`, переключатель `TRadioButton` и другие компоненты, позволяющие реализовать интерфейсные элементы среды Windows.

Дополнительные компоненты `Additional` представляют собой различные дополнительные интерфейсные элементы – графические кнопки `TBitBtn`, окно для вывода графических файлов `TImage`, таблица `TStringGrid` и другие.



На основном окне интегрированной среды разработки расположены еще три окна:

Окно формы `Form1` (рис. 3) представляет собой заготовку (макет) окна разрабатываемого приложения.

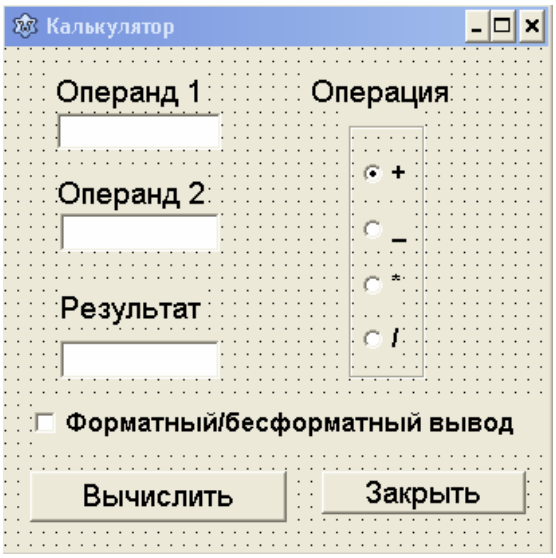


Рис. 3. Пример окна формы проекта

Слева в среде разработки Lazarus расположено *Окно Инспектора объектов*. В верхней его части (рис. 4) отображается иерархия компонентов приложения с точки зрения их принадлежности друг другу.

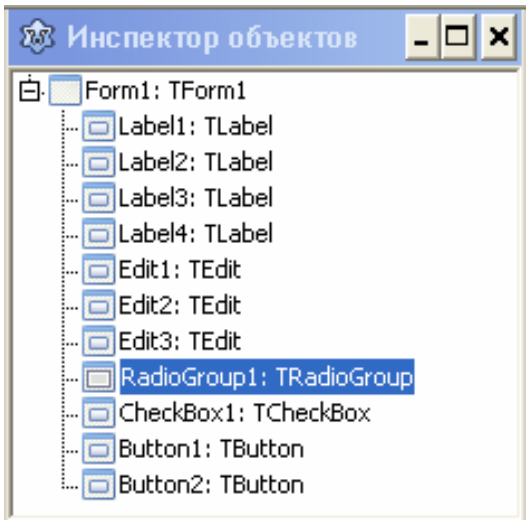


Рис. 4. Верхняя часть *Инспектора объектов*

В нижней части *Окна Инспектора объектов* расположены четыре закладки: Свойства (Property), События (Events), Избранное и Ограничения (рис. 5).

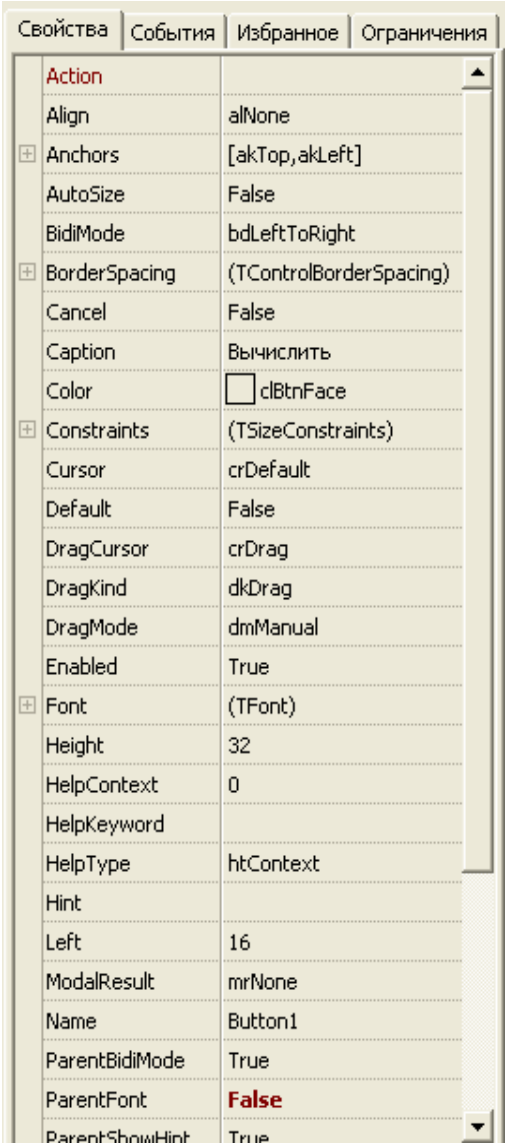


Рис. 5. Нижняя часть *Инспектора объектов*

Страница `События` (рис. 7) используется для задания реакции на событие. Состоит из двух частей. В левой – название события, а в правой – название процедуры, которая обрабатывает данное событие.

Вкладка `Свойства` (`Properties`) (рис. 6) позволяет задавать и изменять свойства объектов. Основные общие свойства компонентов приведены в приложении 2.

Вкладка `События` (рис. 7) позволяет выбирать событие (`Events`), чтобы затем задать реакцию компонента на это событие. Наиболее часто используемые события приведены в приложении 3.

Вкладки `Избранное` и `Ограничения` облегчают проектирование интерфейса.

Каждая страница разделена на две части. Например, на странице свойств в левой части находится название свойства, а в правой – его значение.

Значок «+» слева от названия указывает на то, что свойство состоит из нескольких значений. Чтобы просмотреть их все, достаточно «щелкнуть» по этому значку. Значениями свойств могут быть слова, числа, а также значения из раскрывающегося списка (рис. 6).

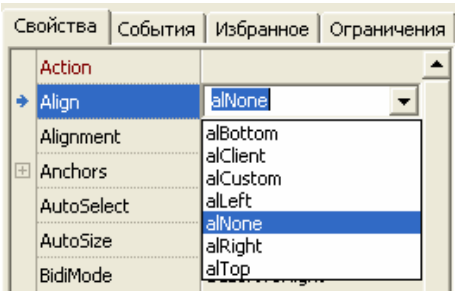


Рис. 6. Список свойств `Align`

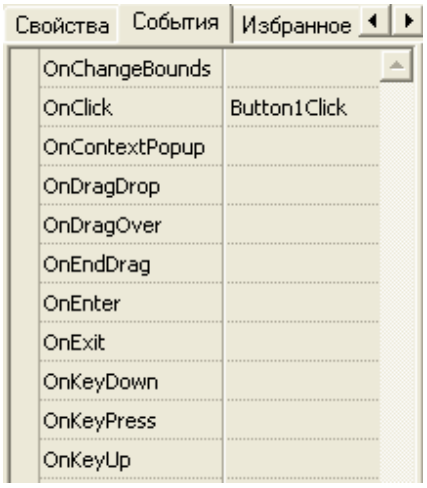


Рис. 7. Вкладка События

Если в правой части страницы События ничего не написано, то программа на данное событие не реагирует.

Для создания реакции на событие необходимо дважды щелкнуть в правой части напротив нужного события. Появится *Окно редактора исходного кода.* При этом курсор будет расположен в том месте исходного кода, где необходимо будет написать код обработки этого события (событийную процедуру).

Название событийной процедуры Lazarus создает автоматически. Оно состоит из двух частей: названия объекта + название события, например

```
procedure TForm1.Button2Click (Sender: TObject);
     begin

     end;
```

Здесь TForm1 – имя формы, на которой расположен объект; Button2 – название компонента кнопка; Click – событие «щелчок левой кнопкой мыши».

В Окне *Редактора исходного кода* (*Code Editor*) набираются тексты программ. Многие функции и возможности этого редактора совпадают с возможностями обычных текстовых редакторов, но он еще обладает рядом возможностей для комфортной работы. В частности выделенный фрагмент текста можно сдвигать вправо и влево на количество позиций, указанных в настройках редактора Окружение → Параметры → Редактор → Общие → Отступ блока, что очень удобно для форматирования с целью структурирования кода. Удобна подсветка блоков Begin … End, помогающая при отладке программ. Заголовки процедур вместе с операторными скобками Begin и End Lazarus формирует автоматически. Между операторными скобками можно печатать инструкции языка программирования Free Pascal, реализующие процедуру обработки выбранного события, а после заголовка процедуры перед

`Begin` описать, при необходимости, локальные переменные. Пример исходного кода приведен на рис. 8.

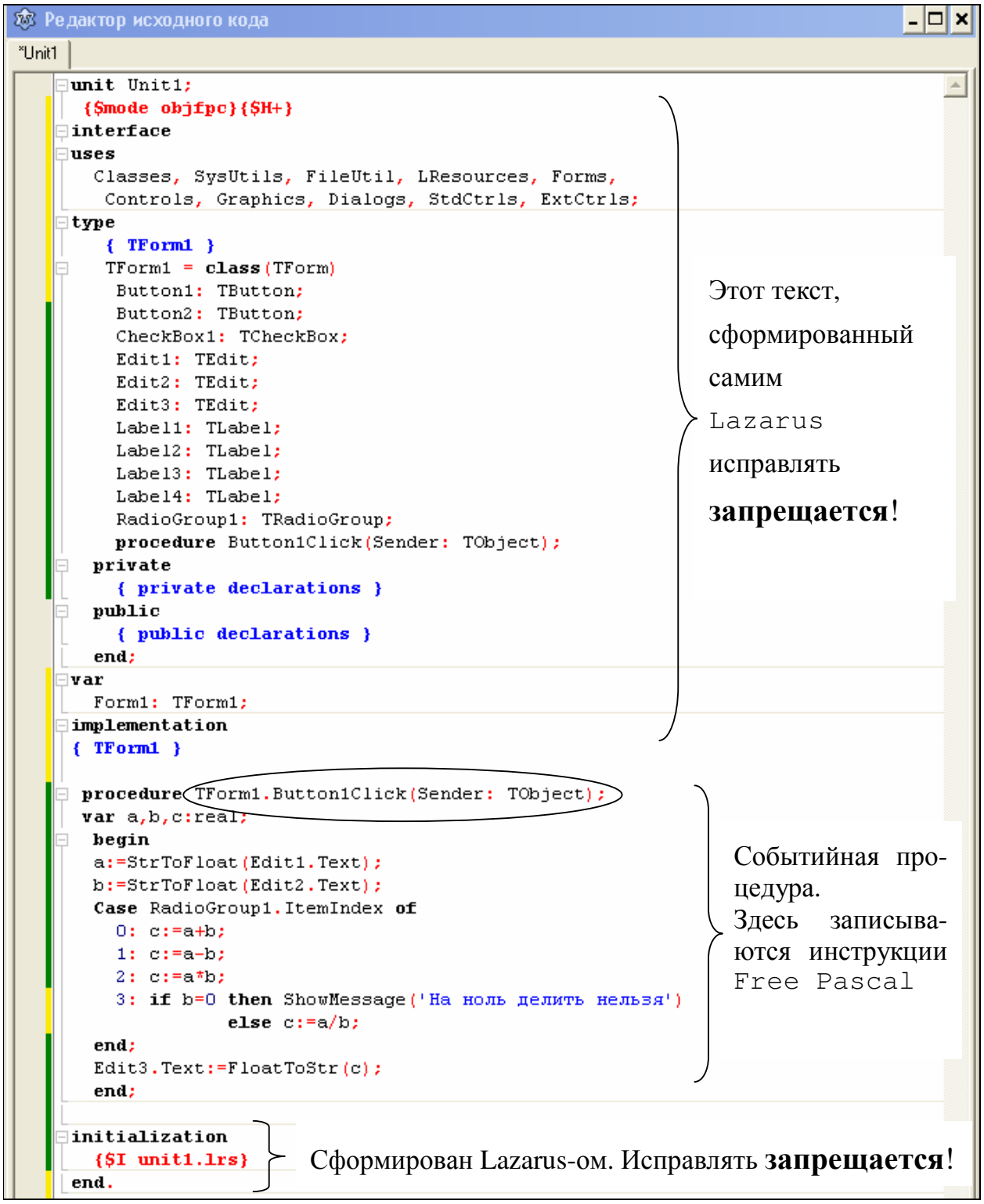


Рис. 8. Редактор исходного кода

С помощью команды меню `Окружение → Параметры...` страница `Редактор → Отображение` (рис. 9) можно изменить шрифт текста программного кода и другие параметры редактора исходного кода.

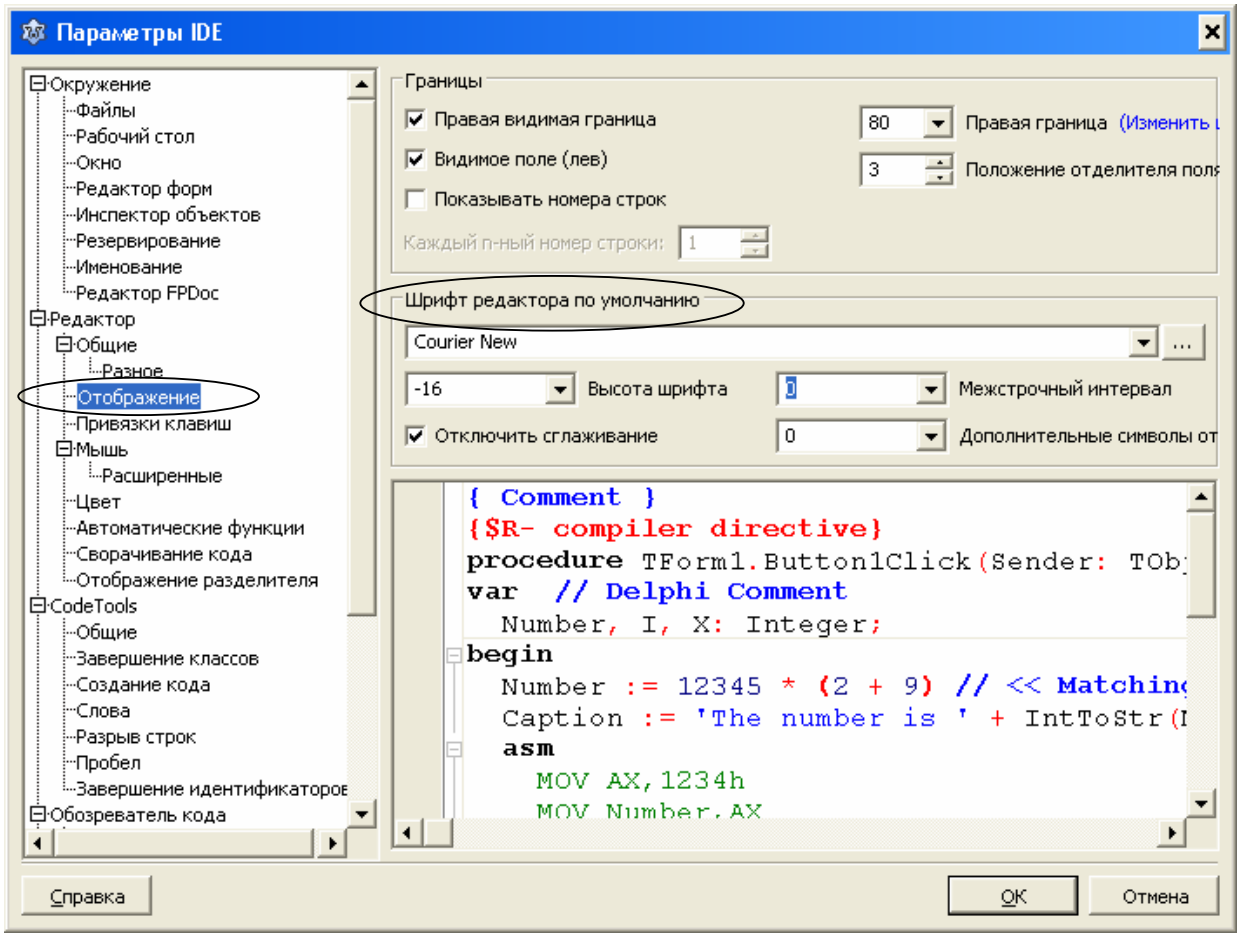


Рис. 9. Параметры IDE

Сообщения компилятора, компоновщика и отладчика Lazarus выводит сообщения в `Окне Сообщений`.

3. КРАТКАЯ ХАРАКТЕРИСТИКА КОМПОНЕНТОВ, ИСПОЛЬЗУЕМЫХ ПРИ СОЗДАНИИ ПРОСТЫХ ПРИЛОЖЕНИЙ

Ниже перечислены компоненты, их основные свойства и события, используемые при создании простых приложений.

Компонент ***Form*** (экранная форма) – макет окна разрабатываемого приложения:

Основные свойства компонента **Form:**

Name	Задает имя компонента формы
Align	Задает режим выравнивания объектов внутри формы.
BorderStyle	Задает стиль обрамления формы, а также поведение формы (возможность менять размеры окна). Значение по умолчанию bsSizeable – стандартная рамка, позволяющая изменять размер окна.
Caption	Задает заголовок окна формы.
Color	Задает цвет формы.
Font	Задает атрибуты шрифта формы (размер, цвет).
Icon	Определяет пиктограмму, отображаемую в заголовке окна формы
Position	Положение формы на экране. Значения этого свойства не вводятся вручную, а выбираются из списка. Если выбрать свойство poScreenCenter, то форма всегда будет появляться в центре экрана.

Основное событие компонента **Form:**

OnCreate	Возникает при начальном создании формы. В его событийной процедуре удобно проводить инициализацию (задание) начальных параметров.

Основные методы компонента **Form:**

Hide	Скрыть форму
Show	Показать форму

Примечание. Основные методы компонентов Lazarus вы можете найти в приложении 4.

Компонент ***Label*** (надпись или метка)  . Назначение – нести на себе надпись. Можно использовать для вывода ответа или пояснения вводимых данных. Относится к группе Standard.

Основные свойства компонента **Label:**

Name	Задает имя компонента Label
Caption	Задает заголовок надписи, выводимой на экран
Alignment	Задает режим выравнивания текста метки.
AutoSize	Позволяет автоматически менять размеры метки, чтобы соответствовать размерам надписи (значение True).
Font	Задает шрифт, используемый для отображения текста
Visible	Задает видимость надписи на экране. Имеет два значения. Если значение True, то надпись видна, False – нет.
WordWrap	Если значение True, то разрешается разбивка и перенос непомещающегося текста на следующую строку. При этом свойство AutoSize должно иметь то же значение True

Основное событие компонента **Label:**

OnClick	Щелчок левой кнопкой мыши на компоненте Label

Компонент ***Edit*** (поле редактирования)  . Используется для ввода/вывода чисел и текста в программу. Относится к группе Standard.

Основные свойства компонента Edit:

Name	Задает имя компонента Edit
AutoSize	Позволяет изменять размер компонента при изменении размера шрифта (значение True).
BorderStyle	Задает стиль обрамления поля.

Text	Содержит текст, отображаемый и редактируемый в строке редактирования.
MaxLength	Ограничивает число вводимых в поле символов.
ReadOnly	Запрещает редактировать отображаемый текст
Font	Задает шрифт, используемый для отображения текста

Основное событие компонента Edit:

OnChange	Происходит, когда пользователь изменяет текст

Компонент **Button** (командная кнопка) 

. Используется для задания реакции на событие. Относится к группе Standard.

Основные свойства компонента Button:

Name	Задает имя компонента Button
Caption	Задает надпись на кнопке.
Height	Задает высоту кнопки.
Width	Задает ширину кнопки.
Left	Задает расстояние от левой границы кнопки до левой границы формы
Top	Задает расстояние от верхней границы кнопки до верхней границы формы
Visible	Если это свойство имеет значение True, то кнопка будет видна на форме, а если – False, то она будет невидимой
Enabled	Если это свойство имеет значение False, то щелчок мыши по кнопке не даст эффекта, при этом текст, размещенный на ней, будет серым.

Основное событие компонента Button:

OnClick	Происходит, когда пользователь щелкает основной (левой) кнопкой мыши на объекте.

Задание 1. Ознакомитесь с основными свойствами и событиями, используемыми при создании простых приложений.

Следуйте рекомендациям, описанным ниже.

1. Запустите ИСР `Lazarus` с помощью команды Главного меню Windows `Пуск` → `Программы` → `Lazarus` → `Lazarus`.
2. Чтобы при запуске `Lazarus` всегда открывался новый проект, выберите пункт меню `Окружение` → `Параметры...` В открывшемся окне уберите метку рядом с командой `Открывать последний проект при запуске` (рис. 10). В дальнейшем при загрузке `Lazarus` всегда будет создаваться *новый* проект.

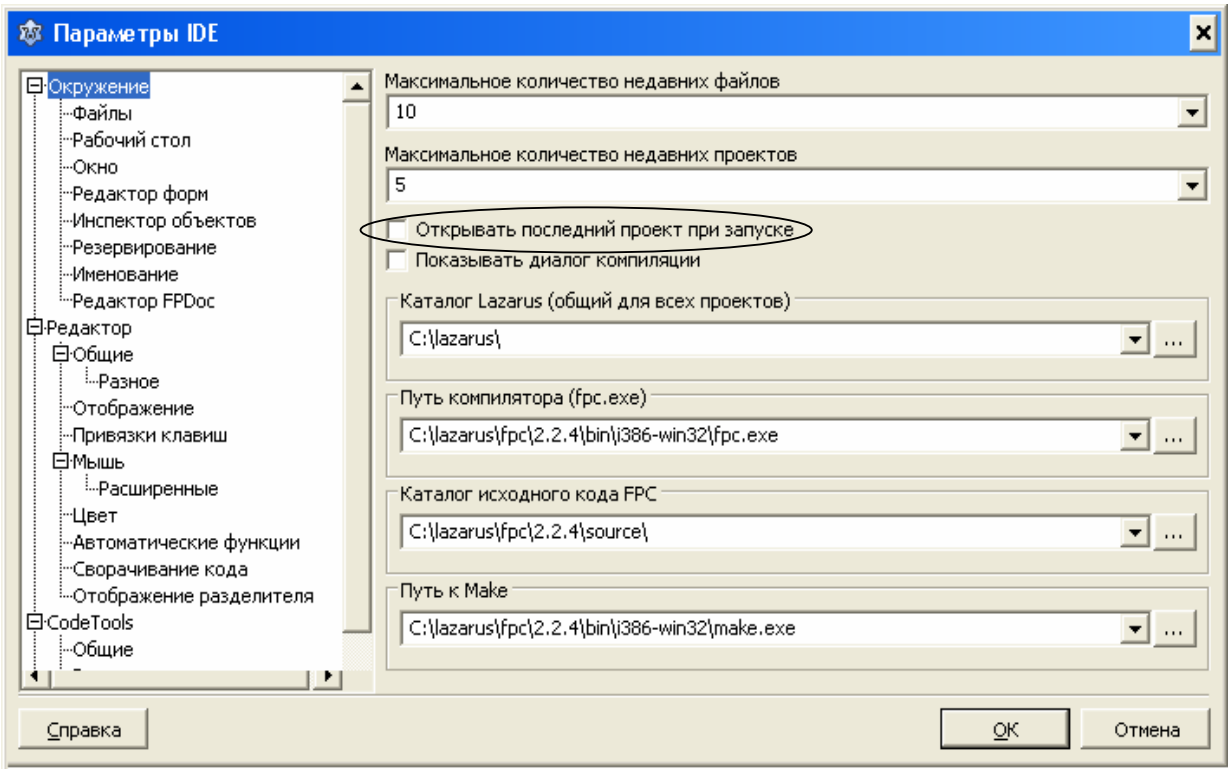


Рис. 10. Команда `Окружение`→`Параметры`

Замечание. Если сразу после инсталляции `Lazarus` при первом запуске проекта выдается сообщения типа «`Не найден каталог исходного кода`» проверьте в окне `Окружение` → `Параметры` путь компилятора (`C:\lazarus\fpc\2.2.4\bin\i386-win32\fpc.exe`) и путь исходного кода FPC (`C:\lazarus\fpc\2.2.4\source\`). Если в этих полях пусто, найдите и «пропишите» эти пути вручную.

В *Инспекторе объектов* найдите и ознакомьтесь с описанными выше основными свойствами и событиями формы. Попробуйте менять размер формы с помощью кнопки мыши, как это делается со всеми стандартными окнами. Посмотрите, какие свойства при этом меняются.

3. Поместите на форму компоненты `Button`, `Edit`, `Label`. Для этого щелкните левой кнопкой мыши по соответствующей компоненте на *Панели компонентов*, затем перейдите на форму и снова щелкните левой кнопкой мыши в окне формы в том месте, где хотите ее расположить (рис. 11).

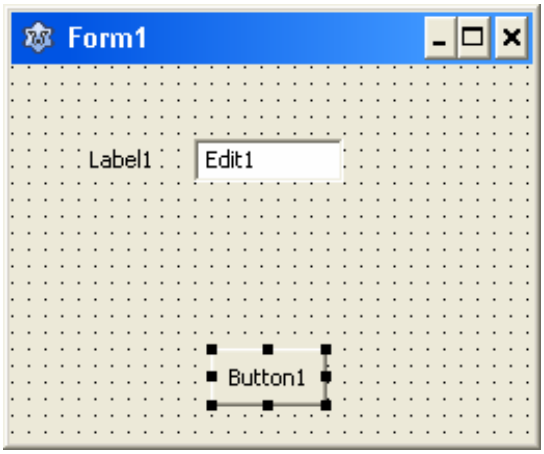


Рис. 11. Форма `Form1`

Выделенный компонент формы окружен восемью маркерами. Именно его свойства отображаются в окне `Инспектора объектов`. Выделенный компонент можно перемещать по форме, меняя его местоположение. Можно также менять его размеры, потянув за один из маркеров. Размер компонента `Label` изменяется автоматически (свойство `AutoSize`).

4. Выделяя по очереди каждый из компонентов ознакомьтесь с описанными выше их основными свойствами и событиями в *Инспекторе объектов.*

4. СТРУКТУРА ПРОЕКТА LAZARUS

4.1. СОСТАВ ПРОЕКТА

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

- *Файл с расширением* `.lpr` **–** файл проекта. Это исходный код основной программы. Несмотря на специфичное для Lazarus расширение на самом деле это обычный Pascal-код (рис. 12):

```
program project1;

{$mode objfpc}{$H+}

uses
  {$IFDEF UNIX}{$IFDEF UseCThreads}
  cthreads,
  {$ENDIF}{$ENDIF}
  Interfaces, // this includes the LCL widgetset
  Forms
  { you can add units after this }, Unit1, LResources;

{$IFDEF WINDOWS}{$R project1.rc}{$ENDIF}

begin
  {$I project1.lrs}
  Application.Initialize;
  Application.CreateForm(TForm1, Form1);
  Application.Run;
end.
```

Рис. 12. Файл с расширением `.lpr`

- *Файл с расширением* `.lpi` **–** файл описания проекта. Сохраняется в XML формате.

> ***Замечание***. Если Lazarus не запущен, то можно открыть существующий проект дважды щелкнув по имени файла с расширением *.lpi или *.lpr.

- *Файл с расширением* `.lfm` – файл формы. Содержит информацию о внешнем виде формы. Информация в нем закодирована.
- *Файл с расширением* `.Pas` – это программный модуль формы. В нем хранится текст программы на языке `Free Pascal`. По умолчанию этот файл называется `Unit1.pas`. Это единственный тип файла, который можно редактировать начинающим.
- *Файл с расширением* `.Exe` – исполняемый файл приложения. После компиляции программы из всех файлов проекта создается единый выполняемый файл, имя этого файла совпадает с именем проекта.

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

Файлы с расширениями .pas и .lfm имеют одинаковое имя (по умолчанию Unit1). У файлов с расширениями .lpr, .Exe, lpi – также одинаковое имя (по умолчанию Project1).

Замечание. В Lazarus *файл проекта* и *файлы проекта* – разные вещи. Файл проекта – это файл головной программы с расширением .lpr, файлы проекта – это набор *всех* файлов приложения.

На рис. 13 приведен пример файлов проекта, состоящего из трех форм. В этом проекте все имена (Project1, Unit1, Unit2, Unit3) сохранены по умолчанию.

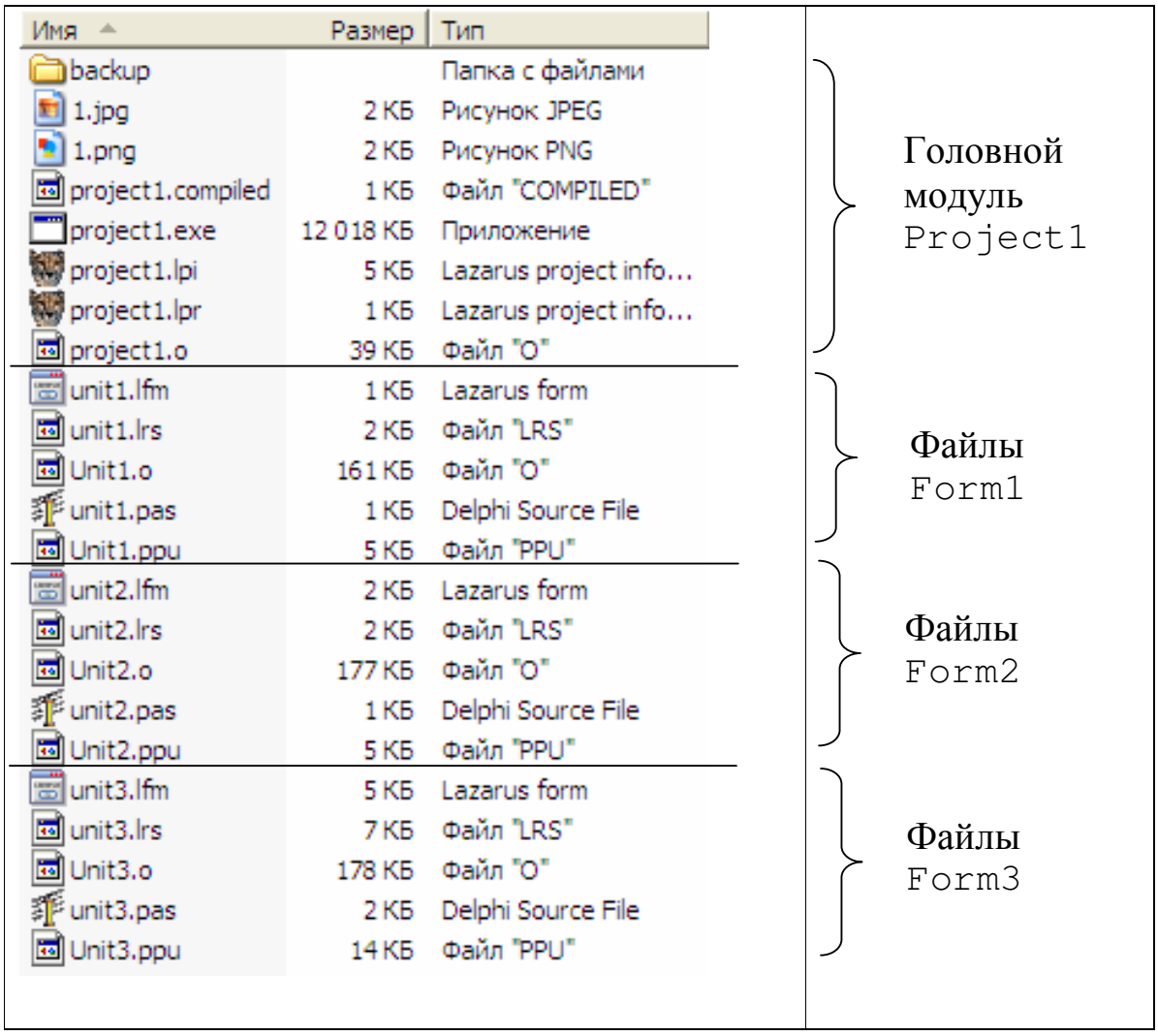


Рис. 13. Файлы проекта, состоящего из трех форм

Задание 2. Сохраните ранее созданный в *Задании 1* проект.

Следуйте рекомендациям, описанным ниже.

1. Выберите пункт меню `Проект → Сохранить проект как...` для сохранения проекта (рис. 14).

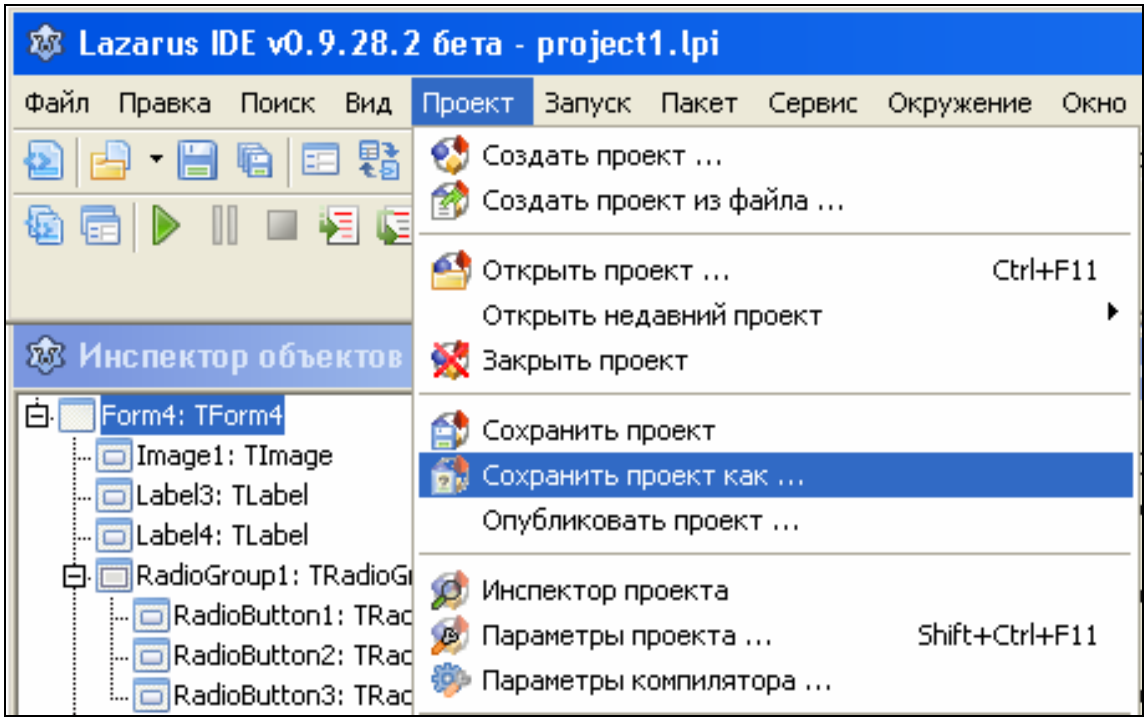


Рис. 14. Команда `Сохранить проект как...`

2. В первом появившемся окне (рис. 15) задайте имя модуля формы. Для простых приложений имя модуля формы можно оставить по умолчанию `unit1`. Если папка для проекта не была создана ранее, создайте ее в открывшемся окне с помощью кнопки  или используя контекстное меню), после чего откройте ее.

   После сохранения модуля формы сразу же появится следующее диалоговое окно, в котором нужно задать имя проекта (рис. 16). Для простых приложений его можно оставить по умолчанию `project1`.

   После этих манипуляций на самом деле сохраняется более чем два файла. В указанном каталоге есть еще файл с текстом программы `Unit1.pas`, файл `Unit1.lfm` со сведениями о форме `Form1`, файлы `Project1.lpr`, `Project1.lpi` и другие файлы проекта.

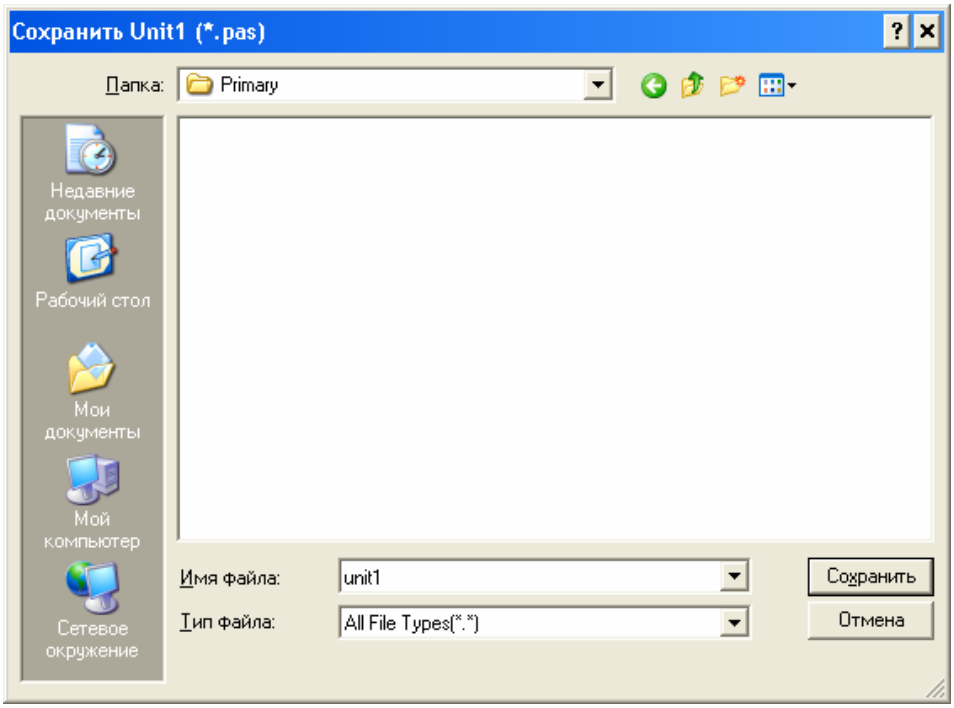


Рис. 15. Сохранить имя модуля формы

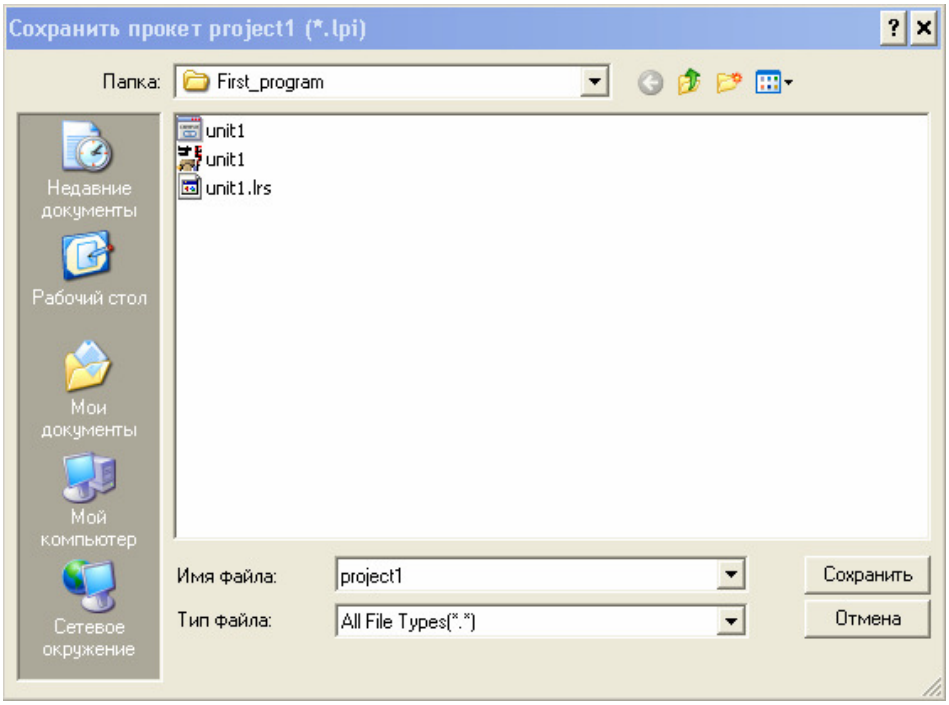


Рис. 16. Сохранить имя проекта

Замечание. После компиляции создается исполняемый файл с расширением .exe, который записывается на диск в пробный каталог, который может быть по той или иной причине недоступен, поэтому проект перед компиляцией следует сохранить, тогда исполняемый файл будет записываться в ту же папку, где находятся все файлы проекта.

Итак, проект сохранен. Все дальнейшие изменения в проекте сохраняйте командой `Проект → Сохранить проект` или щелкнув на значок дискеты в главном меню (рис. 17) .

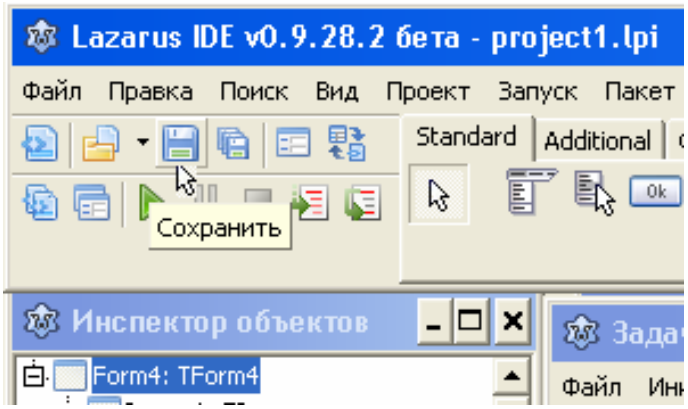


Рис. 17. Сохранение проекта

Замечание. Копирование файлов проекта в другую папку или на внешний носитель выполняется ***только*** с помощью файлового менеджера Windows (например, проводника). В среде `Lazarus` это делать ***запрещается***. Файл с расширением `.exe` можно не копировать.

3. Далее приступаем к визуальному программированию. Ранее на форму `Form1` были помещены компоненты `Button1`, `Edit1`, `Label1`. Выделяя каждый объект, изменим некоторые их свойства с помощью инспектора объектов:

Объект	Свойство	Значение
`Form1`	`Caption`	ПРИМЕР
	`Position`	`poScreenCenter`
`Button1`	`Caption`	Щелкни мышкой
	`Font`	`Arial 16`
`Label`	`Caption`	Мой первый проект
	`Font`	`Arial 14`
`Edit1`	`Text`	Очистите это поле

4. Нажмите на кнопку Запуск  . После сообщения «Проект Project1 успешно собран» появится окно вашего проекта (рис. 18).

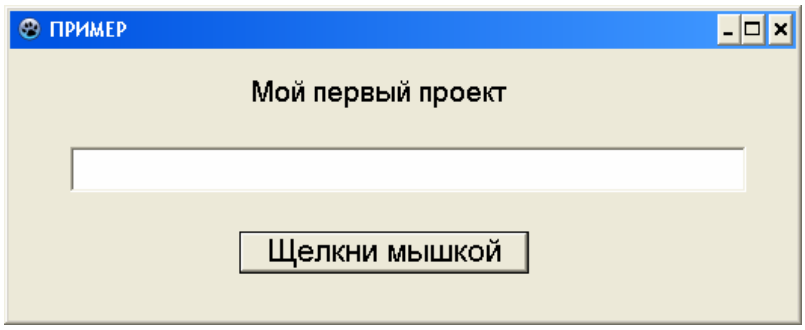


Рис. 18. Окно первого проекта

5. Теперь щелкните по кнопке и убедитесь, что ничего не произойдет. В этом нет ничего удивительного, ведь пока мы не написали ни одной строчки программного кода. Мы разработали только интерфейс.

   Закройте пока окно обычным способом (щелкните на крестик). Выделите объект Button1, и выберем событие OnClick во вкладке События. Дважды щелкните в поле справа от названия. В результате описанных действий в окне редактора кода появится пока пустая процедура обработки события OnClick:

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

 end;
```

   При этом курсор будет установлен между словами begin и end. Это означает, что при наступлении указанного события будут выполняться команды, расположенные между этими словами. Запишите туда, например, следующий код:

```
Edit1.Text:='Ура! Я – студент УГАТУ';
```

   Оно читается так: присвоить свойству Text объекта Edit1 значение 'Ура! Я – студент УГАТУ'. Так как присваиваемое значение – строка, оно заключено в одинарные кавычки.

Замечание. Событийную процедуру можно создать проще, если используется основное событие, установленное по умолчанию для данного объекта. Для этого достаточно дважды щелкнуть на этом объекте левой кнопкой мыши. В нашем случае достаточно дважды щелкнуть на кнопке `Button1`. В результате откроется окно `Редактора исходного кода`, в котором вы увидите пустую процедуру `procedure TForm1.Button1Click(Sender:TObject);`

6. Сохраните созданную программу, запустите ее на выполнение и убедитесь, что кнопка реагирует на щелчок мыши.

7. Закройте окно, щелкнув на крестик. Далее добавьте на форму еще одну кнопку `Button2`. Задайте свойству `Caption` значение `Выход`. Она будет предназначена для выхода из проекта. Щелкните дважды по ней, откроется `Редактор исходного кода`. Наберите в месте курсора `Close` (этот оператор закрывает проект).

Замечание. Перейти из `Окна формы` в `Окно редактора исходного кода` и обратно можно с помощью клавиши `F12`

8. Сохраните проект, запустите его на выполнение и убедитесь, что кнопка `Выход` работает.

9. Измените свойство `ShowHint` объекта `Button2` на значение `True`. Если это свойство имеет значение `True`, то всплывает подсказка – текст, содержащийся в свойстве `Hint`. Поэтому найдите свойство `Hint` и наберите в открывшемся окне фразу `'Кнопка завершает работу программы'`.

10. Запустите проект и убедитесь, что подсказка работает при наведении указателя мыши на кнопку `Выход`. Свойство `ShowHint` можно использовать и для любых других объектов.

11. Закройте проект командой `Проект → Закрыть проект`.

4.2. СТРУКТУРА МОДУЛЯ ПРИЛОЖЕНИЯ *.PAS

Текст программ каждой формы хранится в модулях с расширением `.Pas`. Программный модуль, или просто модуль представляет собой набор типов данных, констант, переменных, процедур и функций. Любой модуль `.Pas` имеет следующую структуру:

```
unit имя_модуля;  {заголовок модуля}
interface
 {раздел описаний}
implementation
 {раздел реализаций}
end.  {конец модуля}
```

Заголовок модуля – это зарезервированное слово `unit`, за которым следует имя модуля и точка с запятой.

Раздел описаний (интерфейса) открывается служебным словом `interface`:

```
interface
  uses   {список подключаемых модулей (библиотек)};
  type   {список_типов};
  const  {список_констант};
  var  {список_переменных};
  procedure имя_процедуры;
     …
  function имя_функции;
     …
```

Здесь размещаются списки подключаемых модулей, объявления типов, констант, переменных, *заголовки* функций и процедур, к которым будет доступ из других модулей, в том числе и автоматически созданные `Lazarus` объявления класса `Form1`. Иными словами, в этом разделе перечисляется, все то, что должно быть видимым из тех модулей, которые его используют.

Раздел реализации начинается ключевым словом `implementation`. Он содержит программный код, реализующий механизм работы описанных на форме программных элементов – тексты процедур обработки событий, а также процедуры и функции, созданные пользователем.

Здесь располагаются объявления типов, констант, переменных, доступных только для данной формы, т.е. те, к которым из других модулей доступа не будет. Здесь же располагаются все тексты процедур и функций, объявленных в разделе `Interface`. Заголовки процедур и функций могут полностью совпадать с заголовками из интерфейсной части или могут отличаться от них полным отсутствием параметров. Если в этой части набран текст функции или процедуры, не представленной в `Interface`, то данная функция или процедура будет локальной.

Все процедуры и функции (событийные и пользовательские) в `Lazarus` также построены по модульному принципу.

На рис. 19 приведен пример модуля `Unit1.pas`.

unit Unit1; {$mode objfpc}{$H+}	Название модуля формы

interface
uses
Classes, SysUtils, FileUtil, LResources, Forms,
Controls, Graphics, Dialogs, StdCtrls, ExtCtrls;
type
{ TForm1 }
TForm1 = class(TForm)
Button1: TButton;
Edit1: TEdit;
Edit2: TEdit;
Edit3: TEdit;
Label1: TLabel;
Label2: TLabel;
RadioGroup1: TRadioGroup;
procedure Button1Click(Sender: TObject);
private
{ private declarations }
public
{ public declarations }
end;
var
Form1: TForm1;
	Раздел интерфейса

implementation
{ TForm1 }
procedure TForm1.Button1Click(Sender: TObject);
var a,b,c:real;
begin
a:=StrToFloat(Edit1.Text);
b:=StrToFloat(Edit2.Text);
Case RadioGroup1.ItemIndex of
0: c:=a+b;
1: c:=a-b;
2: c:=a*b;
3: if b=0 then ShowMessage('На ноль делить нельзя') else c:=a/b; end;
Edit3.Text:=FloatToStr(c);
end;
initialization
{$I unit1.lrs}
	Раздел реализации

end.	Конец модуля

Рис. 19. Пример модуля `Unit1` формы `Form1`

4.3. СТРУКТУРА СОБЫТИЙНОЙ ПРОЦЕДУРЫ

На рис. 20 изображена структура событийной процедуры. Синтаксис заголовка событийной процедуры, написанной для объекта на форме, имеет вид

```
Procedure Имя_класса_формы.ОбъектСобытие(параметры);
```

Название процедуры состоит из имени класса формы, для которой пишется событийная процедура, и после точки имени процедуры. Имя процедуры `ОбъектСобытие` состоит из имени объекта, для которого написана процедура и далее без пробела названия выбранного события. В скобках после имени процедуры записываются ее параметры, которые могут и отсутствовать, например:

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

Текст программы начинается с раздела описания, в котором объявляются все константы, переменные и типы, используемые в данной процедуре. Раздел описания констант начинается со слова `Const`, раздел описания типов – со слова `Type`, раздел описания переменных – со слова `Var`.

Procedure <название процедуры>;	Заголовок процедуры. Название процедуры: *названия объекта + название события*
Const <имя константы> = <значение константы>; . . . <имя константы> = <значение константы>;	Раздел описания констант
Type <имя типа> = <тип>; . . . <имя типа> = <тип>;	Раздел описания типов
Var <имя переменной>:<тип>; . . . <имя переменной>:<тип>;	Раздел описания переменных
<Тексты локальных процедур и функций с заголовками>	Раздел описания процедур и функций

Begin
<операторы>
End;
	Раздел инструкций

Рис. 20. Структура событийной процедуры

За разделом описания следует раздел инструкций (операторов), который начинается со слова `Begin` и заканчивается словом `End`. В разделе инструкций записываются исполняемые операторы. В конце каждого оператора ставится символ «;».

5. ОШИБКИ ВЫПОЛНЕНИЯ ПРИЛОЖЕНИЯ

5.1. ОШИБКИ ВРЕМЕНИ КОМПИЛЯЦИИ

Во время компиляции текст программы проверяется на отсутствие синтаксических ошибок. Компилятор просматривает программу от начала. Если обнаруживается ошибка, то процесс компиляции приостанавливается, и в окне редактора кода выделяется строка с первой ошибочной конструкцией, а в окне сообщений (рис. 21) описываются все сообщения компилятора о синтаксических ошибках.

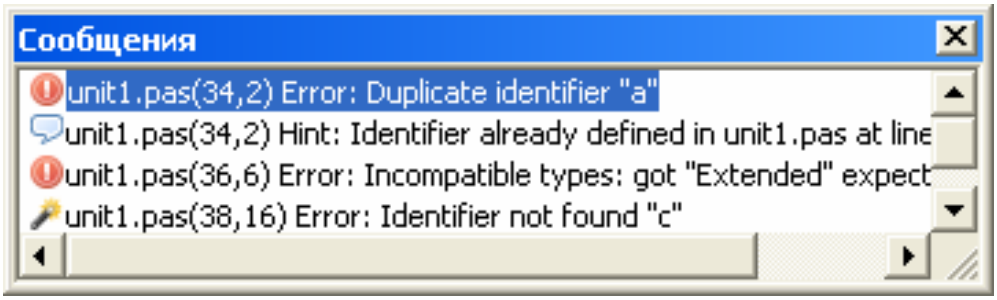


Рис. 21. Окно сообщений

Наиболее типичные сообщения компиляции приведены в таблице.

Сообщения компилятора	Вероятная причина
`Identifier not found` (Идентификатор не найден)	Используется переменная, не объявленная в разделе `var` программы; Ошибка при написании имени переменной; Ошибка при написании синтаксиса оператора; Не найден компонент `Lazarus`.
`String exceeds` (Незавершенная строка)	При записи строковой константы не поставлена завершающая кавычка.
`Incompatible types` (Несовместимые типы)	В операторе присваивания тип выражения не соответствует или не может быть приведен к типу переменной, получающей значение выражения.
“…” `Expected but` “…” `found` (“…” ожидается, но “…” найден)	Ошибка при написании синтаксиса оператора (чаще условного оператора). Не хватает завершающей операторной скобки `end`; Нет разделителя «;» между операторами
`Duplicate identifier` (Повторный идентификатор)	Идентификатор описан дважды

Примеры типичных ошибок начинающих приведены на рис. 22-27.

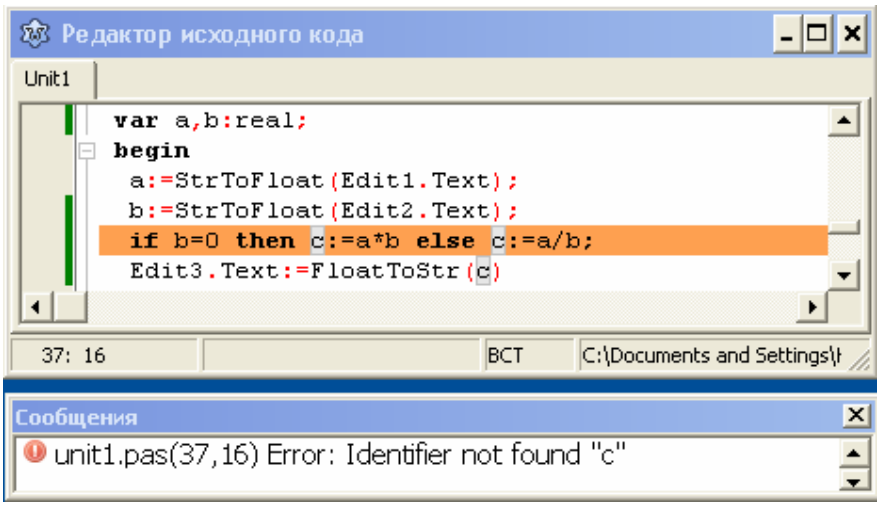


Рис. 22. Не описан идентификатор ***c***

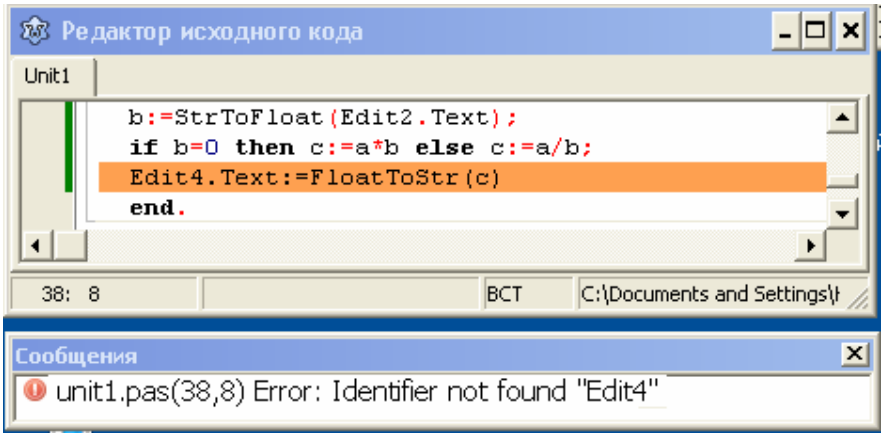


Рис. 23. Не найден объект `Edit4`

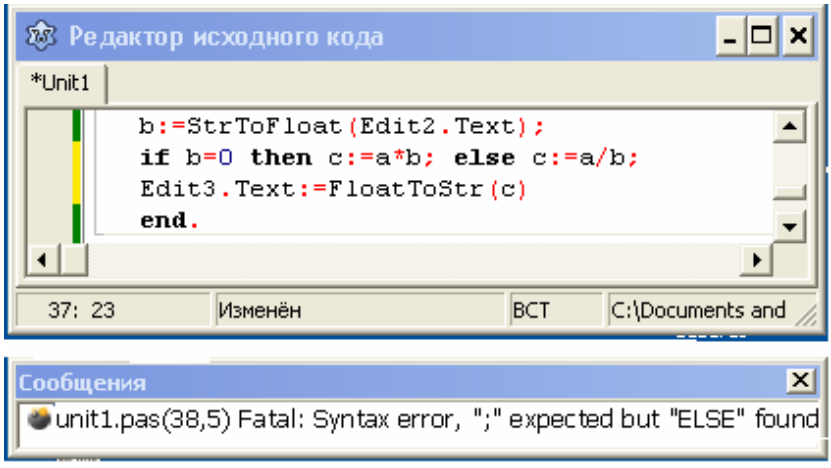


Рис. 24. Неправильно записан условный оператор

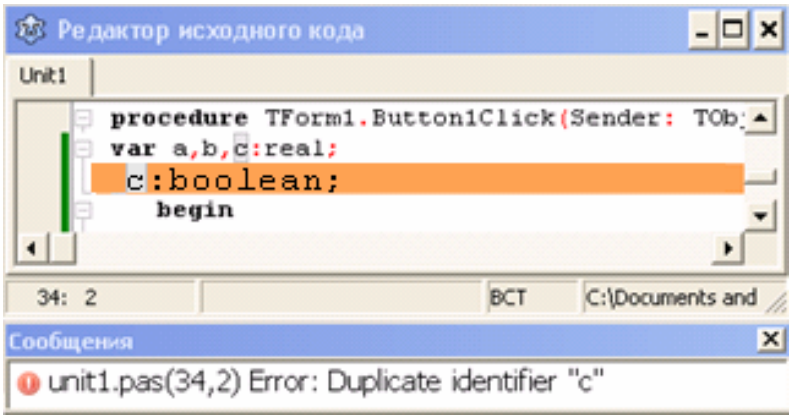


Рис. 25. Идентификатор описан дважды

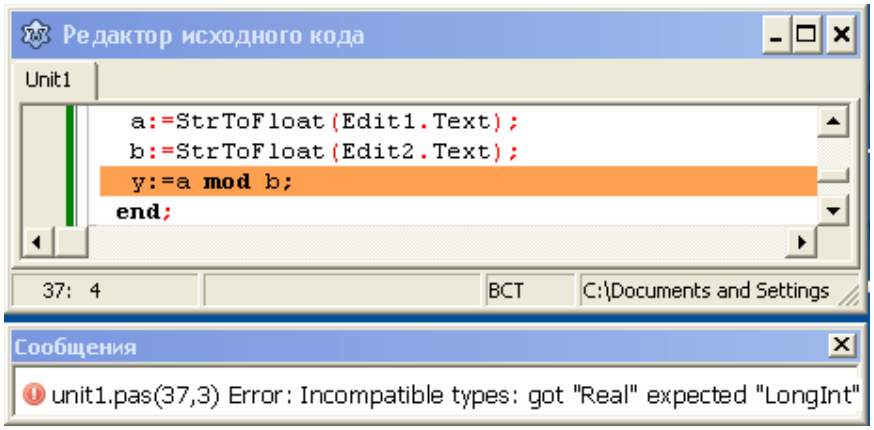


Рис. 26. Несоответствие типов данных

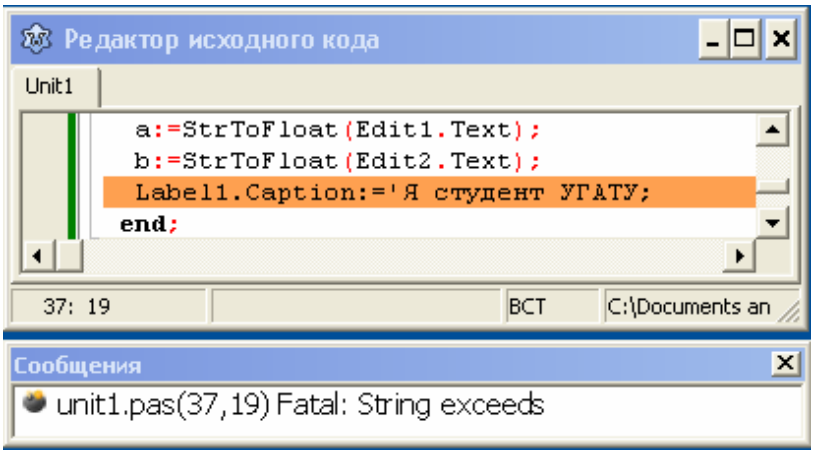


Рис. 27. Не завершена строковая константа

Строка, выделенная компилятором, не всегда содержит ошибку. Довольно часто ошибочным является оператор, находящийся в предыдущей строке (рис. 28).

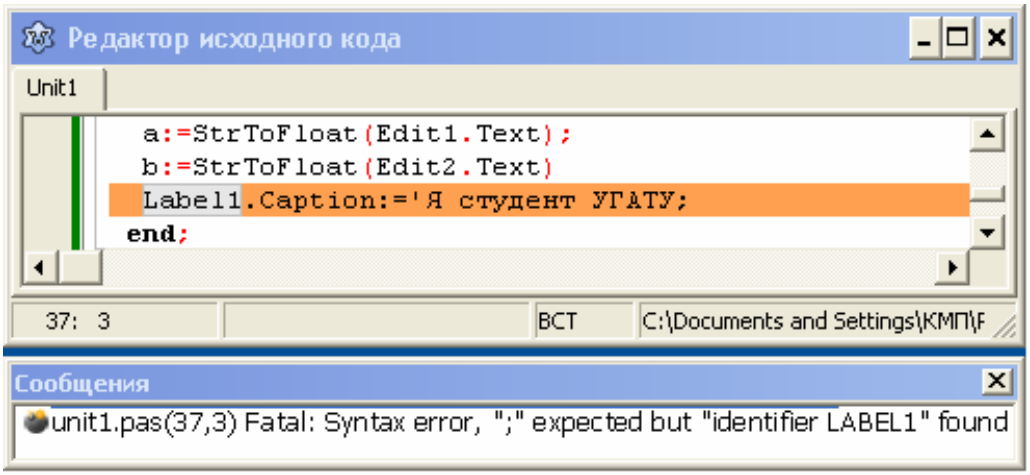


Рис. 28. Нет разделителя «;» между операторами

5.2. ОШИБКИ ВРЕМЕНИ ВЫПОЛНЕНИЯ

Во время работы приложения могут возникать ошибки, которые называются *ошибками времени выполнения*. В большинстве случаев причинами являются неверные исходные данные. Если программа запускается из среды `Lazarus` с помощью встроенного отладчика, то при возникновении ошибки появляется окно с сообщением об ошибке (рис. 29, 30), и выполнение программы приостанавливается.

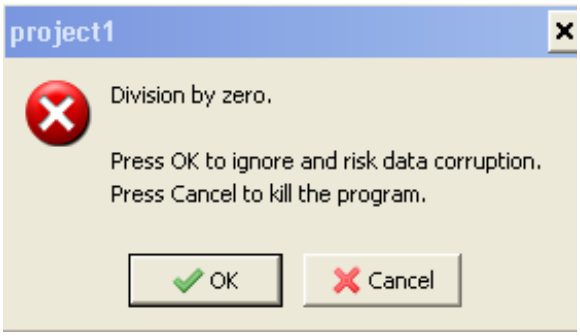


Рис. 29. Деление на ноль

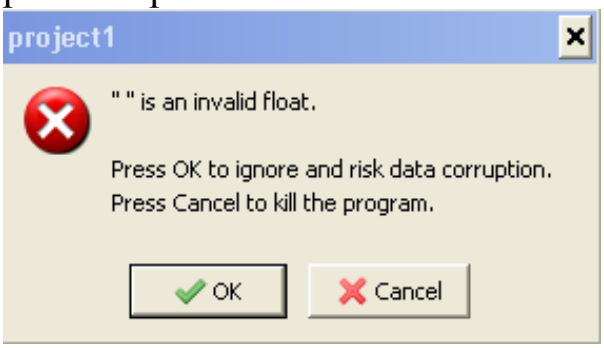


Рис. 30. Невозможность преобразования строки в число

Чтобы остановить работу программы, в которой возникла ошибка, нужно нажать на кнопку `OK` и выполнить команду меню `Запуск` → `Останов` (рис. 31) или просто щелкнуть на красный значок .

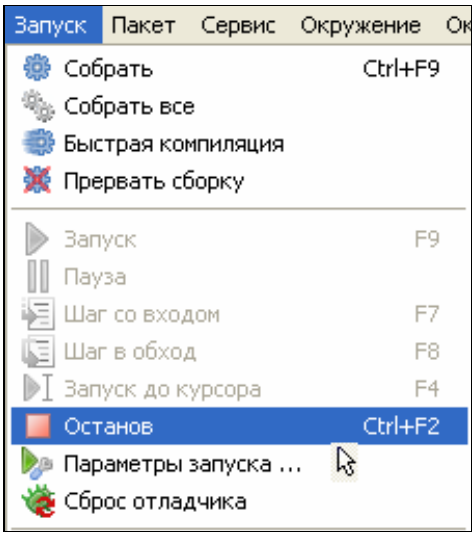


Рис. 31. Команда меню `Запуск` → `Останов`

С алгоритмическими ошибками дело обстоит иначе. Компиляция программы, в которой есть алгоритмическая ошибка, завершается успешно. При пробных запусках программа ведет себя нормально, однако при анализе результата выясняется, что он неверный. Для того чтобы устранить алгоритмическую ошибку, приходится анализировать алгоритм, вручную «прокручивать» его выполнение.

6. ЗАДАНИЯ К ЛАБОРАТОРНОЙ РАБОТЕ

1. Создайте новый проект Lazarus при помощи команды `Проект` → `Создать проект`.
2. Сохраните проект командой меню `Проект` → `Сохранить проект как...`
3. Размещайте на форме по очереди каждую компоненту палитры `Standard` (Приложение 1). Меняйте *свойства* компонентов (Приложение 2). Обратите внимание на то, что многие свойство разных объектов совпадают. Для каждой компоненты изучите основные *события* (Приложение 3) при совершении которых над соответствующим компонентом будет выполняться событийная процедура. Обратите внимание на *событие,* установленное по умолчанию.
4. Размещайте на форме по очереди каждую компоненту палитры `Additional`. Повторите задание 3 для этих компонентов.

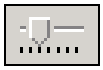
7. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Что такое объект?
2. Что такое метод?
3. Что такое свойство объекта, каким образом его можно изменять?
4. Что такое события? Каково назначение обработчика событий?
5. Каково назначение сообщений?
6. В чем заключается преимущества визуального программирования интерфейса?
7. Опишите структуру и назначение отдельных элементов головной программы приложения Lazarus.
8. Каково назначение модуля в проекте приложения Lazarus? Опишите назначение отдельных разделов модуля.
9. Как различается доступность объектов, описанных в разделах `interface` и `implementation`?
10. Как указать ссылку на свойства и методы объекта в тексте программы?
11. Какие компоненты входят в интегрированную среду разработки приложений Lazarus?
12. Перечислите основные компоненты окна среды Lazarus и укажите их назначение.
13. Как получить подсказку Lazarus по элементам окна и компонентам проектируемого графического интерфейса?
14. Каково назначение страниц `Свойства` и `События` в окне `Инспектора объектов`?
15. Как разместить компонент на форме?
16. Какими способами можно изменять свойства компонента? Приведите примеры.
17. Перечислите состав проекта Lazarus. Опишите назначение различных файлов.
18. Каково назначение обработчиков событий? Каким образом можно инициировать создание процедуры-обработчика событий?

ПРИЛОЖЕНИЕ 1. ОПИСАНИЕ КОМПОНЕНТОВ, НАИБОЛЕЕ ЧАСТО ИСПОЛЬЗУЕМЫХ ПРИ ПРОЕКТИРОВАНИИ ПРИЛОЖЕНИЙ

Компо-ент	Описание
Стандартные компоненты (Standard)	
	TMainMenu (Главное меню формы). Создает линейку меню для формы. После двойного щелчка мыши открывается дизайнер меню Menu Designer, с помощью которого можно установить обработчики событий по каждому пункту меню
	TPopupMenu (Контекстное меню). После двойного щелчка мыши открывается дизайнер менюMenu Designer
	TButton (Командная кнопка)
	TLabel (Надпись). Текст метки нельзя интерактивно изменить при работе приложения, его можно изменить только программно
	TEdit (Однострочное редактируемое текстовое поле). Предназначен для ввода и редактирования строки текста
	TMemo (Многострочное редактируемое текстовое поле)
	TToogleBox (Командная кнопка, похожая на TButton, которую можно «вжать»)
	TCheckBox (Флажок)
	TRadioButton (Переключатель)
	TListbox (Список)
	TComboBox (Раскрывающийся список)
	TScrollBar (Полоса прокрутки)
	TGroupBox (Контейнер с рамкой и надписью для группы компонентов)

Компо-ент	Описание
	`TRadioGroup` (Контейнер с рамкой и надписью для группы связанных переключателей)
	`TCheckGroup` (Контейнер с рамкой и надписью для группы связанных флажков)
	`TPanel` (Панель для размещения других компонентов)
	`TFrame` (Рамка-контейнер для размещения других компонентов)
	`TActionList` (Список пользовательских команд, которые могут использоваться различными элементами приложения)
Дополнительные компоненты (Additional)	
	`TBitBtn` (Кнопка с изображением)
	`TSpeedBtn` (Кнопка-пиктограмма без заголовка, например, для кнопок панели инструментов)
	`TImage` (Графический образ). Объект для отображения или редактирования растрового рисунка
	`TShape` (Стандартная геометрическая фигура)
	`PaintBox` (Окно для рисования)
	`TNoteBook`
	`MaskEdit` (Редактор ввода по шаблону)
	`CheckListBox` (Группа флажков). Похож на `ListBox`, но перед каждым элементом списка имеется флажок
	`ScrollBox` (Панель-контейнер, изменяемого размера, с отображением, при необходимости, полосами прокрутки)

Компо-ент	Описание
	`StringGrid` (Текстовая таблица). Создает таблицу
	`TDrawGrid`
Дополнительные компоненты (Common Controls)	
	`TTrackBar` (Ползунок)
	`TProgressBar` (Индикатор прогресса)
	`TUpDown` (Счетчик)
	`TreeView` (Иерархическое дерево, например файловая структура диска)
	`TListView` (Иерархический список, например список файлов папки)
	`TTabControl` (Набор вкладок)
	`TPageControl`
Дополнительные компоненты (System)	
	`Timer` (Таймер). Способен инициировать события через регулярные промежутки времени. Не визуальный компонент. Для таймера главное событие – истечение заданного интервала времени. Используют для выполнения определенных действий через заданный интервал времени
Дополнительные компоненты (Dialogs)	
	`OpenDialog` (Окно открытия файла)
	`SaveDialog` (Окно сохранения файла)
	`FontDialog` (Панель выбора шрифтов)

Компо-ент	Описание
	`ColorDialog` (Панель настройки цветов)
	`FindDialog` (Панель поиска текста)
	`ReplaseDialog` (Панель поиска и замены текста)
	`PrintDialog` (Окно настройки печати)

ПРИЛОЖЕНИЕ 2. ОСНОВНЫЕ ОБЩИЕ СВОЙСТВА КОМПОНЕНТОВ LAZARUS

Одними и теми же свойствами могут обладать разные компоненты.

Свойство	Значение
`Height, Width`	Высота и ширина компонента в пикселях
`Left`	Положение левой кромки компонента относительно формы
`Top`	Положение верхней кромки компонента относительно формы
`Align`	Положение компонента относительно формы: `alTop` (сверху), `alBotton` (снизу), `alLeft` (слева), `alRight` (справа), `alNone` (определяется свойствами `Left` и `Top`)
`Color`	Цвет фона компонента
`Font`	Внешний вид (размер, цвет и т.д.) шрифта для надписей
`Caption`	Связывает текстовую строку с управляющим компонентом: надпись на кнопке, текст метки, заголовок окна для формы
`Text`	Содержимое поля для текстового окна, которое будет видно при загрузке формы
`Visible`	Если это свойство имеет значение `True`, то компонент будет видимым на форме, а если – `False`, то он будет невидимым
`Enabled`	Если это свойство имеет значение `False`, то компонент недоступен, т.е. щелчок мыши по этому элементу не даст эффекта, при этом текст, размещенный на нем, будет серым. Если это свойство имеет значение `True`, то компонент можно будет использовать в процессе работы программы
`AutoSize`	Если это свойство имеет значение `True`, то размеры оконного компонента будут автоматически меняться при изменении размера шрифта, или добавления в него неоконных компонентов

Hint ShowHint	Если свойство ShowHint имеет значение True, то всплывает подсказка – текст, содержащийся в свойстве Hint
Focused	Если это свойство имеет значение True , то данный компонент получил фокус. Поскольку элементов на форме много, то только один компонент на форме может получить фокус . Фокус объекту можно передать методом SetFocus в программном модуле. Компонента может получить фокус, только если оно видимо и доступно (его свойства Visible и Enabled имеют значения True).
TabStop	Если это свойство имеет значение True, то данный компонент получает фокус с помощью клавиши Tab, т.е. определяет, сможет ли пользователь переходить с помощью клавиши Tab к данному оконному элементу управления
TabOrder	Порядковый номер выбора компонента клавишей Tab

Примеры использования в программе:

Установка свойства в программе	Результат
`Edit1.AutoSize:=True;`	Автоматическое изменение размеров текстового окна
`Edit1.Text:=FloatToStr(a);`	Вывод в текстовое окно значения вещественной переменной a
`Button1.TabOrder:=3;`	После третьего нажатия клавиши Tab командная кнопка получит фокус
`Button1.Hint:='Кнопка завершает работу программы';`	После наведения курсора мыши на командную кнопку появится подсказка о ее назначении

ПРИЛОЖЕНИЕ 3. ПЕРЕЧЕНЬ СОБЫТИЙ

Все компоненты пользовательского интерфейса, которые размещаются на формах, обладают определенным перечнем *событий*, при совершении которых над соответствующим компонентом будут выполняться запрограммированные действия (событийная процедура).

События, инициируемые мышью:

Событие	Действие пользователя, которое вызывает данное событие
`OnClick`	Щелчок левой кнопкой мыши на объекте
`OnDblClick`	Двойной щелчок левой кнопкой мыши на объекте
`OnMouseDown`	Нажимается кнопка мыши при условии, что указатель мыши находится на объекте
`OnMouseUp`	Отпускается кнопка мыши при условии, что указатель мыши находится на объекте
`OnMouseMove`	Указатель мыши движется поверх объекта

Обработчики этих событий имеют следующие параметры:

Параметр	Описание
`Sender`	Объект, для которого выполняется действия мыши
`Button`	Указывает нажатую кнопку мыши: `mbLeft`, `mbMiddle`, `mbRight`
`Shift`	Указывает были ли нажаты клавиши `Alt`, `Shift`, `Ctrl`
`X,Y`	Координаты точки, в которой произошло событие

События, инициируемые клавиатурой:

Событие	Действие пользователя, которое вызывает данное событие
`OnKeyDown`	Нажимается клавиша при условии, что объект находится в фокусе
`OnKeyUp`	Отпускается клавиша при условии, что объект находится в фокусе
`OnKeyPress`	Нажимается и отпускается алфавитно-цифровая клавиша при условии, что объект находится в фокусе

Обработчики этих событий имеют следующие параметры:

Параметр	Описание
`Sender`	Объект для которого выполняется нажатие клавиши
`Key`	Содержит код ASCII нажатой клавиши
`Shift`	Указывает были ли нажаты клавиши `Alt`, `Shift`, `Ctrl`

События перемещения и сброса объектов:

Операции перемещения и сброса позволяют перемещать целые компоненты или отдельные их элементы из одного компонента в другой. Для этих операций определены следующие события:

Событие	Действие пользователя, которое вызывает данное событие
`OnDragDrop`	Отпущена кнопка мыши. Выполнен сброс объекта
`OnDragOver`	Пользователь перетаскивает объект на элемента управления. Событийная процедура пишется для этого элемента управления
`OnStartDrag`	Левая кнопка мыши нажата и удерживается. Процесс перемещения объекта начался
`OnEndDrag`	Процесс перемещения и сброса объекта завершен

Замечание. Для объектов, участвующих в этих событиях, должны быть установлены следующие свойства: свойство `DragMode` перемещаемого компонента равно `dmAutomatic`; свойство объекта `DragKind`, на который перемещаем должно быть равно `dkDrag`.

События, инициируемые для компонентов:

Событие	Описание
`OnChange`	Происходит изменение значения определенного свойства объекта. Например, для объекта `TEdit` это изменение свойства `Text`
`OnEnter`	Компонент получил фокус ввода
`OnExit`	Компонент потерял фокус ввода
`OnActivate`	Приложение или форма становится активной
`OnCreate`	Вызывается при создании (загрузке) формы
`OnPaint`	Вызывается при перерисовке формы
`OnClose`	Вызывается при закрытии формы
`OnTimer`	Вызывается через указанный промежуток времени

ПРИЛОЖЕНИЕ 4. ОСНОВНЫЕ МЕТОДЫ КОМПОНЕНТОВ LAZARUS

Одни и те же методы можно применить к разным объектам.

Метод	Описание метода
`Hide`	Скрывает компонент
`Show`	Показывает компонент
`SetFocus`	Передает фокус объекту, определенному в вызове метода. Поскольку клавиатура одна, а элементов на форме, которые могут ее использовать несколько, необходимо выделить элемент, которому в данный момент передается клавиатурный ввод. Компонент с фокусом ввода имеет значение `True` в своем свойстве `Focused`

Примеры использования в программе:

Использование метода	Результат воздействия метода на объект
`Edit1.Clear`	Очищается текстовое окно
`Button1.SetFocus`	Фокус передается командной кнопке
`Form5.Hide`	Форма `Form5` удаляется с экрана
`Form1.Show`	Форма `Form1` выводится на экран

СПИСОК ЛИТЕРАТУРЫ

1. Алексеев Е.Р., Чеснокова О.В., Кучер Т.В. Самоучитель по программированию на Free Pascal и Lazarus. - Донецк.: ДонНТУ, Технопарк ДонНТУ УНИТЕХ, 2009. - 503 с.
2. Культин, Н. Б. Основы программирования в Delphi 2010 / Никита Культин. – СПб: БХВ-Петербург, 2010. – 434 с.