Муниципальное бюджетное общеобразовательное учреждение
«Средняя школа № 37» города Смоленска

Методическое пособие по теме «Программирование в среде Lazarus»

Филиппенкова Оксана Александровна,
учитель информатики
МБОУ «СШ № 37»

Пояснительная записка

Методическое пособие по теме «Программирование в среде Lazarus» предназначено для обучающихся профильных (физико-математического, технологического) 11 классов при изучении темы «Объектно-ориентированное программирование», а также для обучающихся 10-11 общеобразовательных классов при изучении элективного курса «Программирование в среде Lazarus»

Цель и задачи пособия направлены на удовлетворение потребностей обучающихся в интеллектуальном совершенствовании и мотивации к творческой деятельности при получении универсальных учебных навыков в области программирования на Lasarus.

Объектно-ориентированное программирование постоянно развивается и используется в современных языках программирования при разработке различных программных продуктов. Lazarus наиболее близок для понимания школьниками.

Актуальность. Программирование является одной из самых востребованных сфер деятельности.

Выбор программы Lazarus не случаен. Большинство школ не могут приобретать новейшие средства разработки программ. Поэтому используют свободное программное обеспечение (СПО). В России принята Концепция развития разработки и использования свободного программного обеспечения. Достоинством СПО является общедоступность и бесплатность. Программа «Программирование в среде Lazarus» находится в полном соответствии с Концепцией развития разработки и использования свободного программного обеспечения в Российской Федерации.

Место программы в системе государственных программ. Рабочая программа элективного курса «Программирование в среде Lazarus» для 11 класса разработана на основе федерального государственного образовательного стандарта среднего общего образования (ФГОС СОО); требований к результатам освоения основной образовательной программы (личностным, предметным, метапредметным); основных подходов к развитию и формированию универсальных учебных действий (УУД) для среднего общего образования,

основной образовательной программы.

Особенность и новизна программы заключается в реализации поливариантного подхода к организации образовательного процесса, использовании системы взаимосвязанных занятий, выстроенных в логической последовательности и направленных на активизацию познавательных и творческих способностей школьников

Педагогическая целесообразность программы заключается в том, что подросток 10-11 класса имеет большой опыт в работе с информацией. Именно поэтому *педагогически целесообразно* обучение начинать в среднем школьном возрасте – наиболее благоприятном периоде, когда подросток приступает к систематическому овладению основами наук. Умение использовать ИКТ в качестве инструмента в профессиональной деятельности, обучении и повседневной жизни во многом определяет успешность современного человека.

Методы обучения и содержание **программы** отвечают возрастным особенностям старших школьников. Данная общеобразовательная общеразвивающая программа «Программирование в среде Lazarus» может быть использована в любом учреждении основного или дополнительного образования.

Адресат программы: Учащиеся 10-11 классов общеобразовательных учреждений, владеющие начальными знаниями в области информационных технологий. В этом возрасте учащиеся способны освоить программу по данному направлению, так как начинает активно развиваться логическая память, творческое воображение, алгоритмическое, критическое мышление, и память. Развивается способность к обобщённому и абстрактному мышлению. Работа в Lazarus позволяет учащимся создавать значимый для них продукт, исходя из интересов, потребностей и возможностей. Программа направлена на профессиональное самоопределение обучающихся.

Обучение школьников основам алгоритмического мышления базируется на понятии исполнителя, которое рассматривается при изучении курса информатики в среднем звене общеобразовательной школы в различных темах, так или иначе связанных с алгоритмической обработкой информации.

При этом, опыт структурного программирования, полученный детьми на уроках информатики на данной ступени образовательного процесса может являться фундаментом при изучении темы «Объектно-ориентированное программирование» в 10-11-х профильных (физико-математических, технологических) классах, что позволяет усложнить изучаемый материал и приложить практическое применение программирования в курсе информатики.

Условия реализации программы

1. Наличие профильных (технологических) классов в старшем звене общеобразовательной организации с достаточным количеством академических часов по учебному плану (168 часов в год / 4 часа в неделю) или возможность ведения внеурочной деятельности по предмету «Информатика»

2. Наличие современной материально-технической базы в кабинете информатики (современный компьютер, проектор, возможность доступа к ресурсам сети интернет)

Цель и задачи программы

Данный курс, а также разработанная система практических занятий преследует следующую *цель*: приобрести навыки разработки программных приложений в свободной визуальной среде программирования Lazarus.

Задачи курса:

1. Формировать у обучающихся умения продуктивно выполнять мыслительные операции (анализ, синтез, сравнение, абстрагирование) посредством выполнения практических заданий.

2. Реализовать положительный опыт создания проектов в области объектно-ориентированного программирования.

Планируемые результаты

Личностные результаты:

• сформированность мировоззрения, соответствующего современному уровню развития науки и техники;

• осознанный выбор будущей профессии и возможностей реализации собственных жизненных планов

Метапредметные результаты:

• владение навыками познавательной, учебно-исследовательской и проектной деятельности, навыками разрешения проблем;

• способность и готовность к самостоятельному поиску методов решения практических задач, применению различных методов познания

Предметные результаты:

иметь представление о :

основных возможностях среды программирования Lazarus;

инструментарии среды Lazarus, необходимом для разработки полнофункционального приложения;

принципах работы основных компонентов Lazarus;

основах объектно-ориентированного подхода в программировании;

конструировании интерфейсов приложений и разработке их дизайна;

владеть:

приёмами организации и самоорганизации работы по созданию приложений в среде программирования Lazarus.

Оглавление

Краткая теория

Среда визуального программирования Lazarus

Lazarus - это *среда визуального программирования*. Здесь программист получает возможность не просто создавать программный код, но и наглядно показывать системе, что бы он хотел увидеть.

Технология визуального программирования позволяет строить интерфейс будущей программы из специальных компонентов, реализующих нужные свойства. Количество таких компонентов достаточновелико.

Среда визуального программирования Lazarus сочетает в себе компилятор, объектно-ориентированные средства визуального программирования и различные технологии, облегчающие и ускоряющие создание программы.

После запуска Lazarus появляется окно. В верхней части этого окна размещается *главное меню* и *панель инструментов*. Слева расположено окно *инспектора объектов,* а справа окно *редактора исходного кода*. Если свернуть или сдвинуть окно редактора, то станет доступным *окно формы*, представленное на рис. 1.

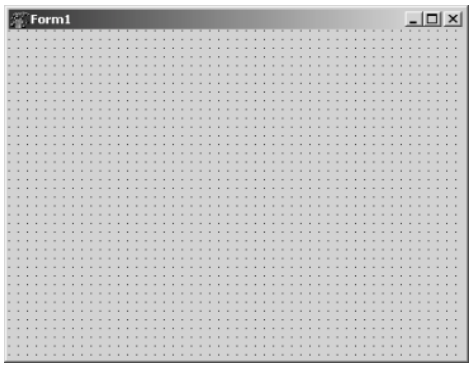


Рисунок 1.: Окно формы программирования

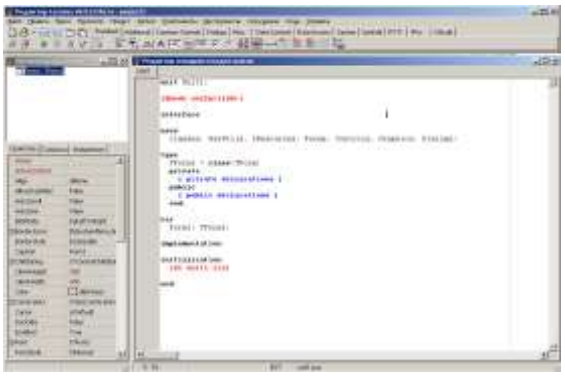


Рисунок 2: Среда визуального Lazarus

Работу над программой в среде визуального программирования условно можно разбить на две части. Первая это создание внешнего вида (интерфейса) будущей программы, вторая — написание программного кода. Итак, *инспектора объектов* и *окно формы* нужны для создания интерфейса программы, а *редактор исходного кода* — для работы с ее текстом. Файлы, из которых в результате получается программа, называют *проектом*.

Окно формы

Окно формы (рис. 1) — это проект интерфейса будущего программного продукта.

Вначале это окно содержит только стандартные элементы – строку заголовка и кнопки развертывания, свертывания и закрытия. Рабочая область окна заполнена точками координатной сетки (Координатная сетка отображается только на этапе создания программы).

Задача программиста – используя *панель компонентов*, заполнить форму различными интерфейсными элементами, создавая тем самым внешний вид своей программы.

Окно редактора Lazarus

Окно редактора тесно связано с окном формы (рис. 3) и появляется вместе с ним при создании нового проекта.

Окно редактора по умолчанию находится на первом плане и закрывает окно формы.

Окно редактора предназначено для создания и редактирования текста программы, который создается по определенным правилам и описывает некий алгоритм. Если окно формы определяет внешний вид будущей программы, то программный код, записанный в окне редактора, отвечает за ее поведение.

Вначале окно редактора содержит текст, обеспечивающий работу пустой формы. Этот программный код появляется в окне редактора автоматически, а программист в ходе работы над проектом вносит в него дополнения, соответствующие функциям программы.

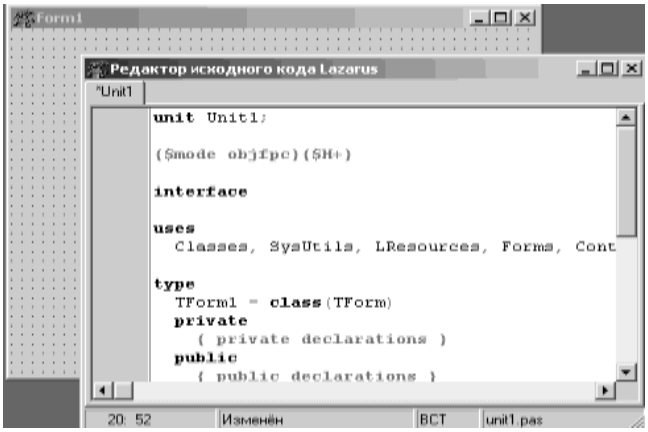


Рисунок 3: Окно формы и окно редактора

Обратите внимание, что при загрузке Lazarus автоматически загружается *последний проект*, с которым работал пользователь. Происходит это благодаря установке **Открывать последний проект при запуске**, которая находится на вкладке **Сервис.** Если убрать метку рядом с командой **Открывать последний проект при запуске**, то при загрузке Lazarus будет создавать *новый проект*.

Панель компонентов

Панель компонентов расположена под главным меню (рис. 4). Она состоит из большого числа групп, в которых располагаютсясоответствующие компоненты.

Рисунок 4: Панель компонентов

Компонент – это некий функциональный элемент интерфейса, обладающий определенными свойствами. Размещая компоненты на форме, программист создает внешний вид своей будущей программы - окна, кнопки, переключатели, поля ввода и т.п.

Для *внедрения нового компонента* на форму нужно сделать два щелчка мышкой:

- в панели компонентов, для выбора компонента;
- в рабочем пространстве формы, для указания положения левого верхнего угла компонента.

Компоненты объединяются в группы по функциональному при- знаку. После создания проекта по умолчанию открывается список группы **Standard**, содержащий основные элементы диалоговых окон. Просмотреть другие группы

можно, раскрывая их щелчком по соответствующей вкладке.

Инспектор объектов

Окно *инспектора объектов* располагается слева от окна редактирования. Как правило, оно содержит информацию о выделенном объекте. На рис. 5 представлен инспектор объектов с информацией о вновь созданной форме. Окно инспектора объектов имеет три вкладки: **Свойства**, **События**, **Избранное.** Эти вкладки используются для редактирования свойств объекта и описания событий, на которые будет реагировать данный объект. Совокупность *свойств* отображает внешнюю сторону объекта, совокупность *событий* – его поведение.

Вкладки инспектора объектов представляют собой таблицу. В левой колонке расположены названия свойств или событий, в правой – конкретные значения свойств или имена подпрограмм, обрабатывающих события. Чтобы выбрать свойство или событие, необходимо щелкнуть левой кнопкой мыши по соответствующей строке.

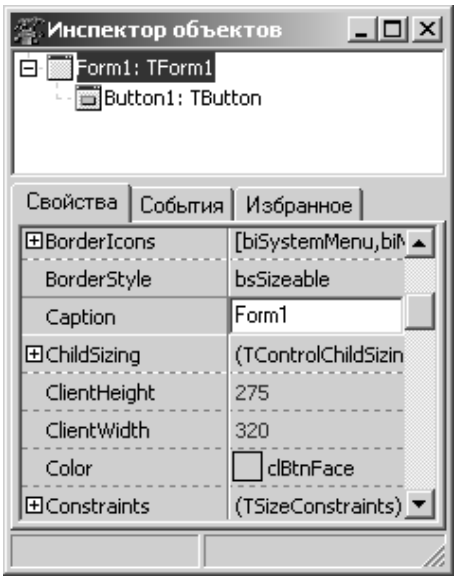


Рисунок 5: Окноинспектора объектов

Этапы создания приложения в Lazarus

Создание приложения в Lazarus состоит из следующих шагов:

1. Подготовка
2. Размещение компонентов интерфейса пользователя
3. Создание процедур-обработчиков событий
4. Тестирование и отладка приложения

Подготовка

Подготовка состоит из следующих шагов.

1. Создать отдельную папку для каждого проекта.

Без пробелов! Без русских букв!

2. Запустить Lazarus.
3. Создать новый проект: Приложение
4. Сохранить проект в созданной специально для него папке:

Проект/Сохранить проект как...

5. Проверить успешность компиляции и запуска нового пустого проекта
6. Закрыть запущенный пустой проект.
7. Продолжить сборку проекта.
8. Регулярно сохранять проект в процессе сборки

Работа 1. Первая программа в Lazarus

Процесс создания программы в Lazarus состоит из двух этапов: формирование внешнего вида программы, ее интерфейса и написание программного кода на языке программирования Free Pascal, заставляющего работать элементы интерфейса.

Для создания интерфейса программы существует *окно формы*, а для написания программного кода – *окно редактора*. Эти окна тесно связаны между собой, и размещение *компонентов* на форме приводит к автоматическому изменению программного кода.

Создадим простую программу, в результате которой на форме будет создана кнопка, при нажатии на нее произойдет смена фраз

Начнем с *создания нового проекта*

Проект — **Создать проект...**. В появившемся диалоговом окне (рис. 6) выберем из списка слово **Приложение** и нажмем кнопку **Создать**. Результатом этих действий будет появление*окна формы* и *окна редактора программного кода*.

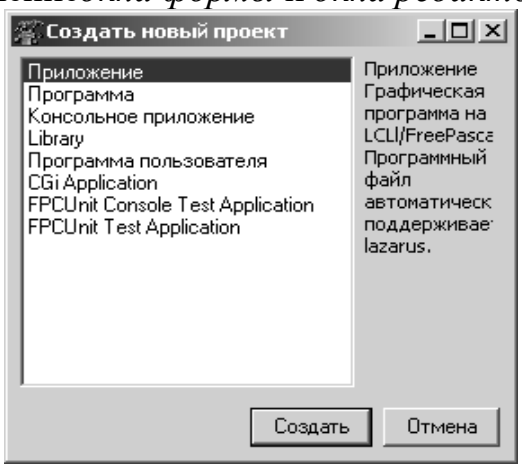


Рисунок 6: Создание нового проекта

Сохраним созданный проект, воспользовавшись командой **Проект** — **Сохранить проект как….** , предварительно создав папку для сохранения проекта

Теперь можно приступить к визуальному программированию. У нас есть один объект – форма Form1. Изменим некоторые его свойства с помощью инспектора объектов. Перейдем в окно инспектора объектов и найдем там свойство Caption (Заголовок). По умолчанию это свойство имеет значение Form1. Изменим его на слово ПРИМЕР 1. Это изменение сразу же отразится на форме – в поле заголовка появится надпись – ПРИМЕР 1 (рис. 7). Аналогично можно изменить размеры формы, присвоив новые значения свойствам Height (Высота) и Width (Ширина), еще проще это сделать, изменив положение границы формы с помощью кнопки мыши, какэто делается со всеми стандартными окнами

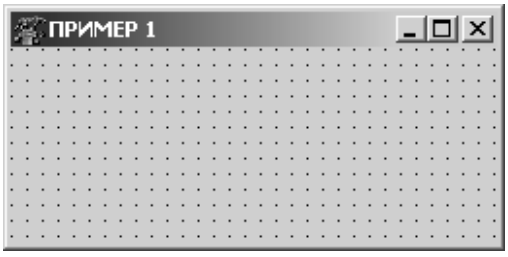


Рисунок 7: Измененииезаголовка формы

Поместим кнопку на форму. Для этого обратимся к панели компонентов (рис.

4) и найдем там компонент TButton. Чтобы *разместить компонент* на форме, нужно сделать два щелчка мышкой: первый — по компоненту, второй — по окну формы. В результате форма примет вид, представленный на рис. 8.

Теперь у нас два объекта: форма Form1 и кнопка Button1. Выделим объект Button1 и изменим его заголовок Caption и размеры Height, Width. В заголовке напишем фразу «Привет!» *Сохраним* проект (**Проект — Сохранить проект**).

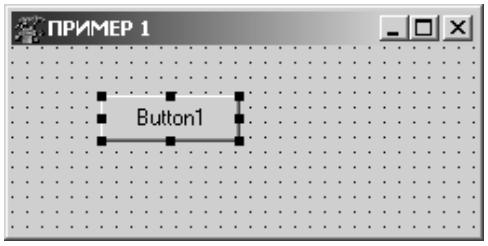


Рисунок 8: Размещение кнопки на форме

Для того чтобы посмотреть, как работает наша программа, ее необходимо *запустить на выполнение.* Сделать это можно кнопкой **Запуск** в панели инструментов (рис. 9). Созданное окно с кнопкой должно выглядеть, как на рис. 10

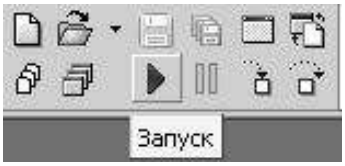


Рисунок 9: Кнопка Запуск впанели инструментов

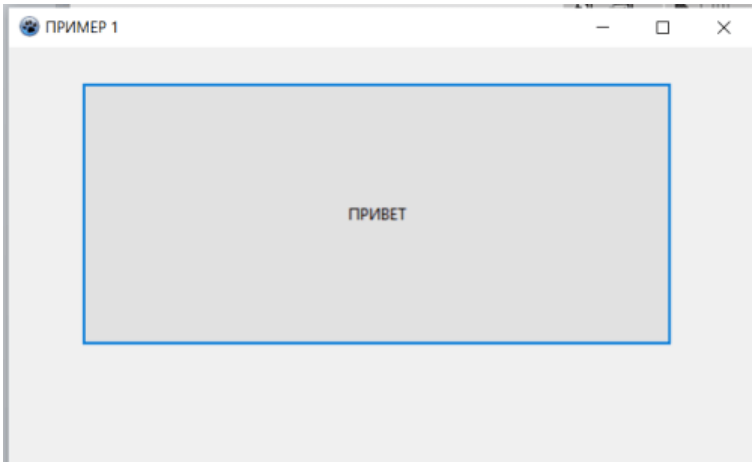


Рисунок 10: Результат работы *программы*

Теперь щелкните по кнопке и убедитесь, что ничего не произойдет. Несмотря на это, в окне редактора появился текст программы:

```
unit Unit1;
{$mode objfpc}{$H+}
interface
uses
  Classes, SysUtils, FileUtil, Forms, Controls, Graphics, Dialogs, StdCtrls;
type
  { TForm1 }
  TForm1 = class(TForm)
    Button1: TButton;
```

```
  private
    { private declarations }
  public
    { public declarations }
  end;
var
  Form1: TForm1;
implementation
{$R *.lfm}
end.
```

После запуска программы на экране появилось окно, на котором расположена кнопка с надписью «ПРИВЕТ!».

Пусть при нажатии на нее появится фраза: «ЗДРАВСТВУЙТЕ!».

Для воплощения этой идеи завершим работу программы. Выделим кнопку Button1 и обратимся к вкладке **События** инспектора объектов. Выберем событие OnClick – *обработка щелчка мыши* и дважды щелкнем в поле справа от названия (рис. 11).

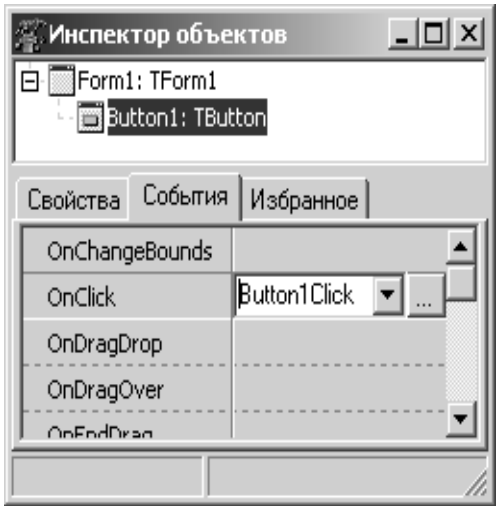


Рисунок 11: Выбор события OnClick

В результате описанных действий в окне редактора появиться следующий текст:

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

Теперь установим курсор между словами begin и end созданной подпрограммы и напишем:

```
Button1.Caption:='ЗДРАВСТВУЙТЕ!';
```

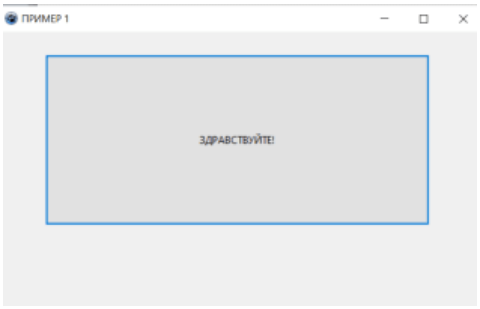


Рисунок 12: Окончательный результат работы программы

Сохраним созданную программу, запустим ее на выполнение и убедимся, что кнопка реагирует на щелчок мыши (рис. 12). *Закрыть проект* можно командой **Проект — Закрыть проект**.

Работа 2. Форма и ее основные свойства

Цели:

познакомить обучающихся с формой и ее основными свойствами и методами; развивать умения выполнять действия с формой, воспитывать эстетические навыки при оформлении формы и компонентов.

Задание.

Создать графическое приложение, которое при загрузке формы, будет выводить название в заголовке формы «Мое первое приложение», а по щелчку на форме в заголовке формы появится вопрос «Зачем нажал?».

Ход работы

1. В заголовке формы измените название (Caption)

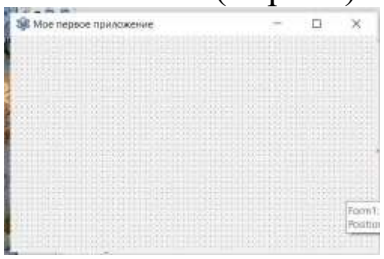


Рисунок 13: Название формы

2. Задайте основные свойства TForm (по своему усмотрению):

•Color (например Form1.Color:=clLime)

•BorderIcons (например Form1.BorderIcons:= [biSystemMenu,biMinimize,biMaximize])

•BorderStyle (например Form1.BorderStyle:= bsSizeable)

•Cursor (например Form1.Cursor:= crHourGlass)

•FormStyle (например Form1.FormStyle:= fsStayOnTop)

•Hint (например Form1.Hint:=’моя подсказка’)

•Position (например Form1.Position:= poMainFormCenter)

• Width и Height (например Form1.Width:=153)

3. Реализовать, чтобы по щелчку на форме в заголовке формы появился вопрос «Зачем нажал?».

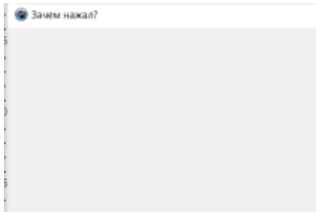


Рисунок 14:После выполнения щелчка

Фрагмент программного кода

```
procedure TForm1.FormCreate(Sender: TObject);
begin
Form1.Caption:='Моё первое приложение';
end;
```

```
procedure TForm1.FormClick(Sender: TObject);
begin
Form1.Caption:='Зачем нажал?';
end;
```

4. Реализовать закрытие формы по нажатию на кнопку –кнопка TButton. Разместите кнопку, создайте событие onClick. В методе обработчика напишите Form1.Close. В случае если закрывается главная форма – закрывается приложение.

Работа 3. Приветствие

Цели:

1. Показать разницу в подходах в программировании «стандартном» / «консольном» и «визуальном» или «объектно-ориентированном».
2. Познакомить с некоторыми Объектами Управления и Контроля (ОУК) и принципами их использования в Объектно Ориентированном Программировании (ООП).
3. Продемонстрировать возможные варианты настройки проектируемого приложения.

Задание.

Написать программу, которая предлагает оператору ввести своё имя. Итогом работы будет обращение программы: «Привет, «имя»!»

Ход работы.

1. Консольный вариант.

I. Составляем алгоритм программы:

1. Вывести на экран приглашение программы: «Введи свое имя:»
2. Получить с клавиатуры введенное пользователем имя.
3. Вывести на экран обращение программы.

II. Запускаем Lazarus.

В меню «Файл» выбираем «Создать» и в открывшемся окне выбираем:

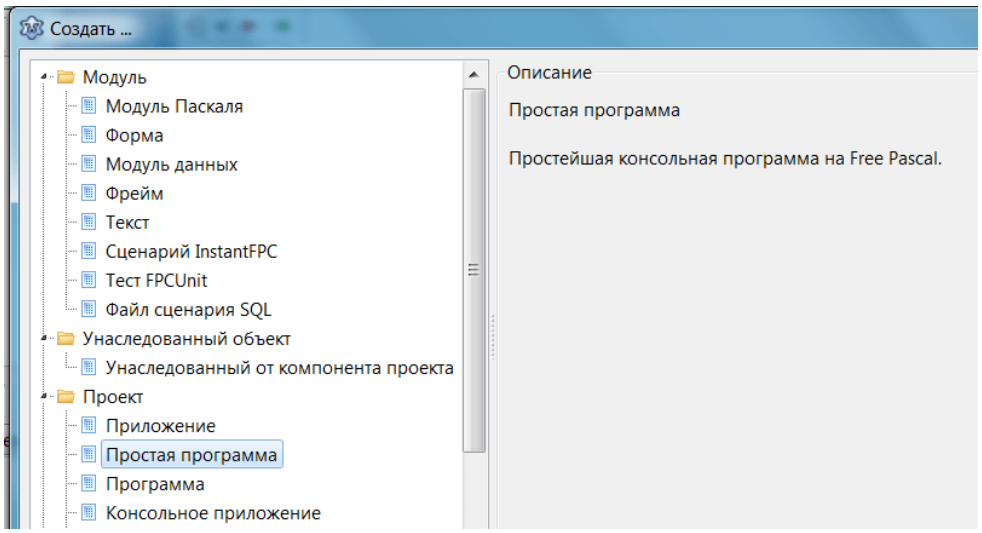


Рисунок 15:Создание программы

Простая программа, т.к. наша программа действительно будет простейшей.

И открывшемся окне определяем необходимую переменную в разделе Var и в блоке описания действий записываем код программы на языке Pascal

Запускаем программу на выполнение <F9> и выполняем её. В итоге получаем Окно с выполненной программой:

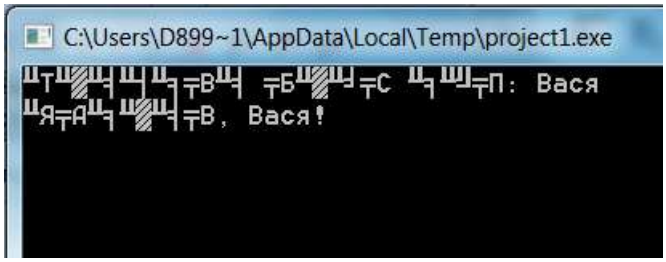


Рисунок 16:Окно с выполненной программой

Рассмотрев полученный результат, мы убеждаемся, что программа работает.

Имя исполняемого файла project1.exe.

И существует проблема с выводом на экран русских букв.

Дело в том, что Lazarus использует кодировку UTF8, и если у вас Windows использует кодировку Ansi, то возникает вот такая проблема...

Проблема легко решается ПЕРЕКОДИРОВКОЙ русских символов из UTF8 в Ansi с использованием функции Utf8ToAnsi(русский текст).

Например:

Рисунок 17: Окно с выполненной программой (русский текст)

Сохраняем проект в папке «Консоль»:

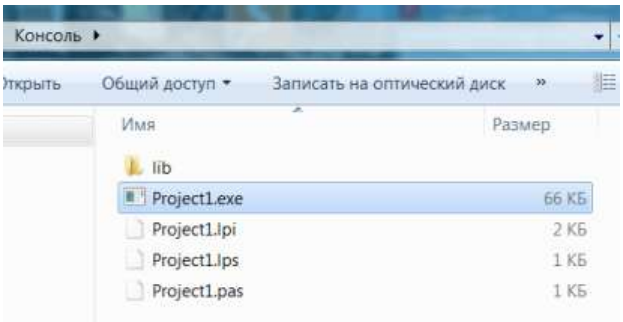


Рисунок 18:Сохранение проекта

Как видим, текст программы на диске занимает всего 1 Кбайт, Программа «весит» уже гораздо больше - целых 66 Кбайт т.к. в неё включены и коды процедур и функций используемых в программе.

Размер исполняемого (.exe) файла можно уменьшить, если отключить запись в исполняемый файл отладочной информации.

Проект --> Параметры проекта

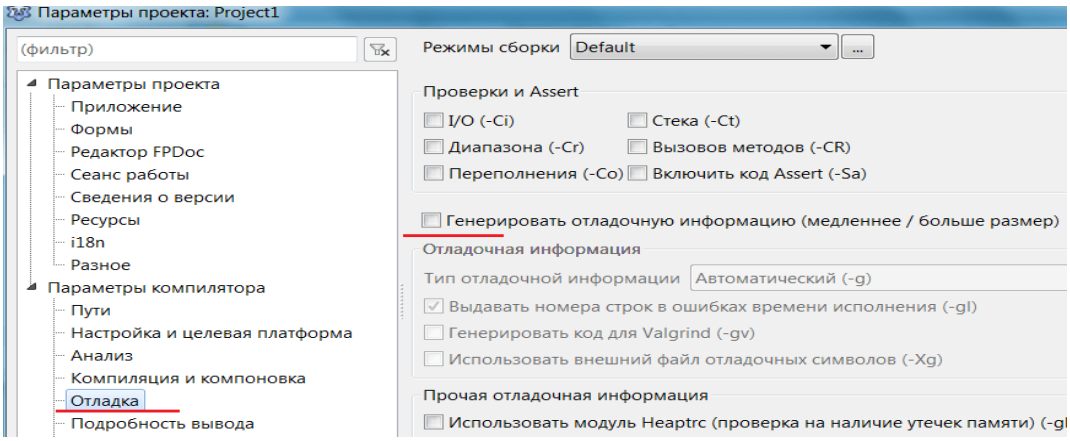


Рисунок 19:Отладка

После компиляции (сборки) программы:

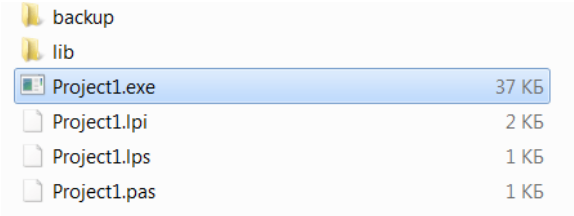


Рисунок 20:Размер файла

Размер уменьшился вдвое.

Так же можно изменить имя исполняемого файла в Свойствах Проекта.

Проект --> Параметры проекта --> Параметры компилятора --> Пути

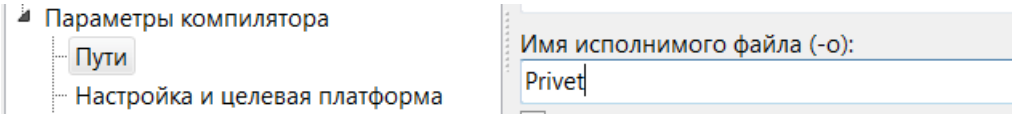


Рисунок 21: Изменение имени в свойствах

И придать программе привлекательный значок добавив в проект графическое изображение - файл типа .ico.

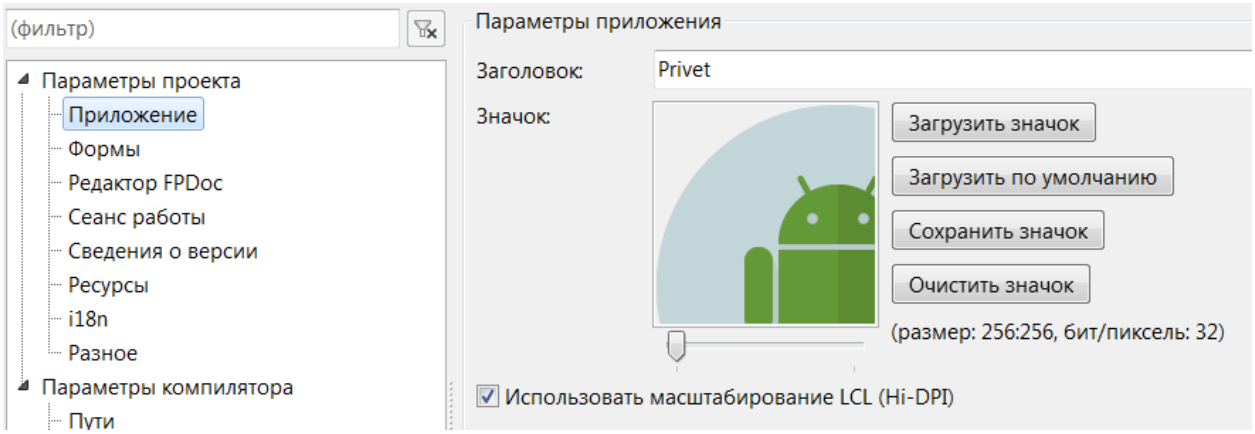


Рисунок 22:Добавление значка

Но это несколько увеличит размер исполняемого файла, т.к. Графический файл будет включен в тело исполняемого файла.

2. Приложение WINDOWS

Для создания приложения под Windows следует, загрузив Lazarus поступить следующим образом:

1. По умолчанию вы уже начали работу над новым проектом. Если же у вас открылся предыдущий проект, то следует, открыв меню Сервис → Параметры, убрать галочку «Открывать последние проект и пакеты при запуске» и, предварительно закрыв, повторить запуск Lazarus.

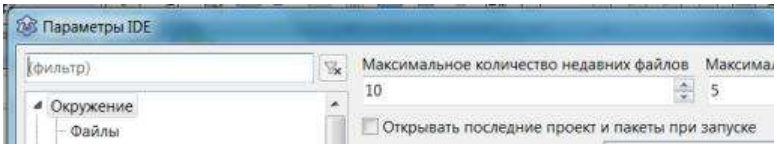


Рисунок 23:Создание приложения

2. либо создать новый проект. Меню Файл → Создать → Приложение

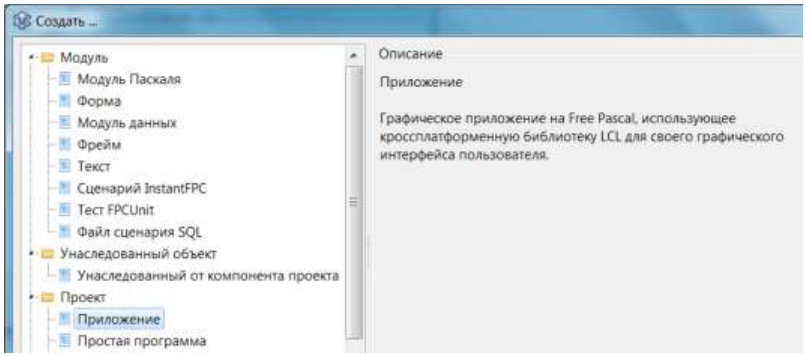


Рисунок 24: Создание приложения

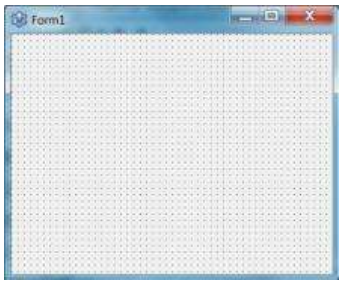
В визуальном программировании коренным образом изменяется алгоритм создания графического приложения в сравнении с консольной программой.

Этапы:

1. На предлагаемой «Форме» следует создать Интерфейс будущей программы. Для этого нужно воспользоваться «Вкладками» и Объектами Управления и Контроля (ОУК) размещенными на них. При этом никакой «алгоритм работы программы» писать не нужно! Главный критерий – удобство работы с будущей программой, т.е. удобный интерфейс.
2. Для тех ОУК, для которых это необходимо и возможно определяем «События», выбирая их из списка предлагаемых.
3. Для каждого «События», используя процедуры и функции, пишем «Обработчик» данного события, т.е. программный код.
4. Проводим проверку правильности работы программы.
5. При необходимости, создаем иконку программы, изменяем имя исполняемого файла, уменьшаем размер конечного продукта. Аналогично этапу работы с консольной программой.

ВНИМАНИЕ

Для начинающих пользователей изменять имена ОУК не рекомендуется!



Этап 1.

В окне будущей программы размещаем ОУК:

Label1 – метка. Текст приглашение к диалогу.

Edit1 – однострочный редактор. Для ввода имени.

Label2 – метка. Для вывода результата работы.

Button1 – кнопка. Для окончания работы программы.

Рисунок 25:Создание формы

Начинаем формировать интерфейс программы:

- на вкладке «Standard» совершаем двойной клик левой клавишей мыши по ОУК «TLabel». На форме появляется объект «Label1». Аналогичный результат получим и при активации мышью «TLabel» с последующим кликом мышью в окне «Form1».

Затем переходим к изменению свойств ОУК «Label1».

Аналогично размещаем на форме ОУК «TEdit».

«Align» с alNone на alTop – привязываем объект вверх.

В этом ОУК нет свойства «Caption», но есть «Text» - удаляем его значение.

Устанавливаем не только «BorderSpacing» - «Around»=10, но и «BorderSpacing» - «Left» и «BorderSpacing» - «Right» - отступы слева и справа по 100 пикселей.

--

Размещаем на форме кнопку «TButton».

Для ОУК Button1 устанавливаем следующие свойства:

«Align» с alNone на alBottom – привязываем объект вниз.

«Caption» с Button1 на Завершение:

«BorderSpacing» - «Around» устанавливаем значение 10, «Right» и «Left» по 100.

На свободное поле формы помещаем объект Label2. И устанавливаем для него следующие свойства:

«Align» с alNone на alClient – заполняем всё свободное место.

«Caption» - удаляем всё.

«BorderSpacing» - «Around» устанавливаем значение 10, «Right» и «Left» по 50.

«Alignment» изменяем на taCenter (для выравнивания текста по центру).

--

- меняем заголовок (Caption) формы (Form1) на «Привет!».

Можно изменить размер используемого в форме шрифта: Свойства (формы) → Font → Name (Times New Roman) и Size (16).

Итоговый вид формы:

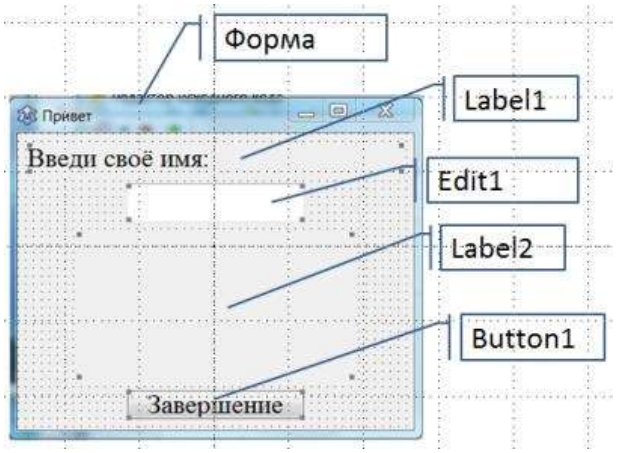


Рисунок 26:Форма

Определяем СОБЫТИЯ.

- По нажатию кнопки «Завершение», должна завершаться программа.

Выделить кнопку щелчком мыши и на вкладке События выбрать OnClick.

Для создания обработчика этого события, достаточно кликнуть по кнопке […].

Затем в блоке действий обработчика ввести оператор Halt.

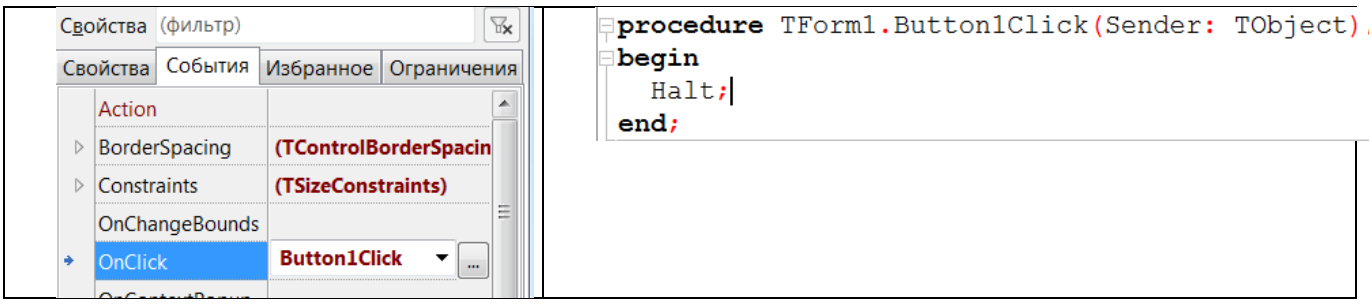


Рисунок 27:Свойства

- Ввод вашего имени завершается нажатием клавиши <Enter>. Поэтому, выделив ОУК Edit1 кликом мыши, выбираем событие OnKeyPress и создаём обработчик. В обработчике записываем то, что должно произойти по нажатию клавиши <Enter> - вывод в свойство Caption Label2 соответствующего текста:

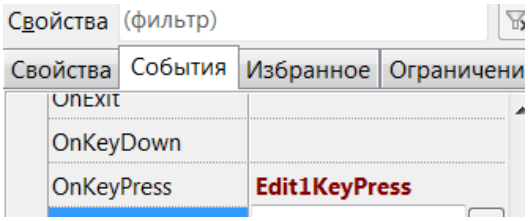


Рисунок 28:Форма

С последующей очисткой поля ввода.

```
procedure TForm1.Edit1KeyPress(Sender: TObject; var Key: char);
begin
  If Key=Chr(13) then   // если нажат символ с кодом 13 (<Enter>), то
   begin
    Form1.Label2.Caption:='Привет, '+Form1.Edit1.Text+'!';
    Form1.Edit1.Clear;
   end;
```

Завершающим этапом работы является сохранение проекта.

В выбранном для сохранения каталоге вы обнаружите следующие папки и файлы:

BACKUP	папка с предыдущими версиями файлов
LIB	папка с библиотечными данными
projectl.exe	скомпилированный файл проекта, программа
projectl.ico	файл иконки проекта
projectl.lpi	информационный файл проекта
projectl.lpr	стартовый файл проекта
projectl.lps	конфигурация проекта в виде xml-кода
projectl.res	файл ресурсов
unitl.lfm	информация о всех объектах на форме
unitl.pas	файл процедур и функций, код, подключаемый модуль

Запуск проекта для дальнейшей работы с ним производится выбором файла **projectl.lpr** – это, собственно, и есть ПРОГРАММА.

Работа 4. Сложение и вычитание чисел

Цель: научиться создавать простейшие математические модели в программе Lazarus., познакомить с тем, как выбирать компоненты (объекты) с панели инструментов, а также научить преобразовывать типы данных.

Задание.

Создать приложение «Счеты», которое может складывать и вычитать два числа.

Ход работы.

1. Запустить Lazarus

2. Разместить на форме три метки класса TLabel, три текстовых поля класса TEdit и две кнопки класса TButton. Все эти объекты можно брать с закладки Standart:

Рисунок 28:Выбор объектов на панели Стандартная

3. С помощью окна Инспектор объектов выделяем объекты и меняем их свойства

4. В результате наша форма вместе с другими объектами на ней принимает следующий вид:

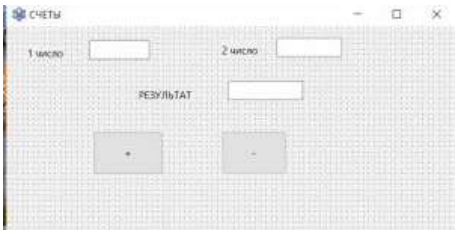


Рисунок 29:Вид формы

5. Напишем процедуру-обработчик нажатия по кнопке сложения (Button1). Для этого в окне инспектора объектов выбираем закладку События и находим событие OnClick (это и есть нажатие по кнопке).

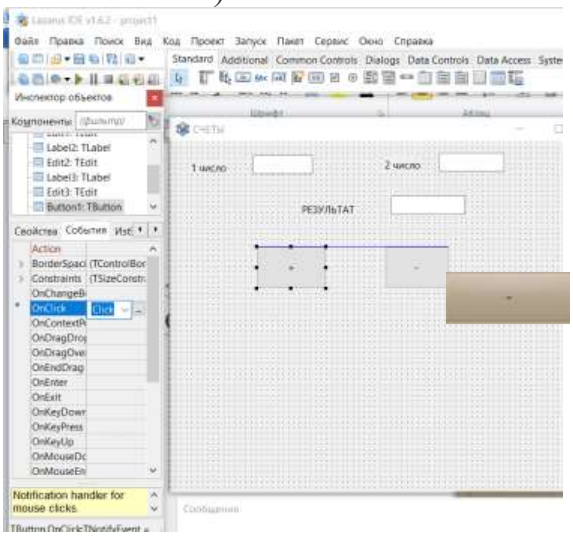


Рисунок 30:Выбор события

6. Наберем следующий текст внутри процедуры-обработчика:

```
procedure TForm1.Button1Click(Sender: TObject);
var a,b,result:real; // Объявление локальных переменных вещественного типа
begin
```

a:=StrToFloat(edit1.Text);*{Переводим текст, находящийся в поле ввода edit1 в вещественное число. Если необходимо работать с целыми числами, то используется функция StrToInt}*

b:=StrToFloat(edit2.Text);

result:=a+b;

edit3.Text:=FloatToStr(result); *{В текстовое поде ввода edit3 выводим полученную сумму, не забыв при этом сделать обратное преобразование вещественного числа result в строку. Если необходимо работать с целыми числами, то используется функция IntToStr}*

end;

7. Аналогично напишем процедуру-обработчик нажатия по кнопке вычитания
8. Сохраним
9. Выполним программу

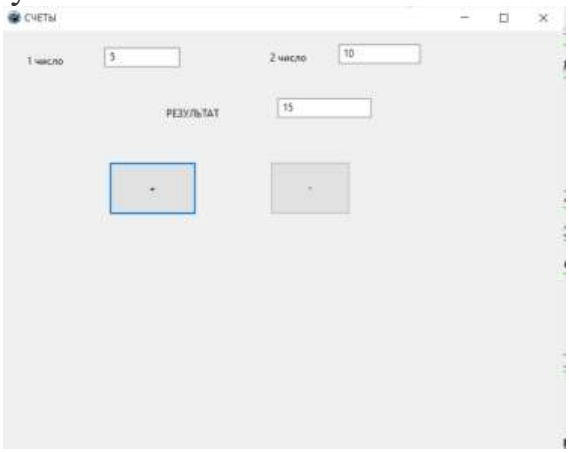


Рисунок 30:Итоговая форма

Работа 5. Умножение и деление чисел

Цели: научиться создавать простейшие математические модели в программе Lazarus, познакомить с тем, как выбирать компоненты (объекты) с панели инструментов, а также научить преобразовывать типы данных.

Задание.

Создать приложение «Простейший Калькулятор», используя приложение «Счеты» (см. Работу 4), которое может складывать, вычитать, умножать и делить два числа.

Ход работы.

1. Открыть созданное приложение «Счеты»
2. Изменить имя формы
3. Добавить на форму две кнопки класса TButton.
4. С помощью окна Инспектор объектов выделяем объекты и меняем их свойства
5. В результате наша форма вместе с другими объектами на ней принимает следующий вид:

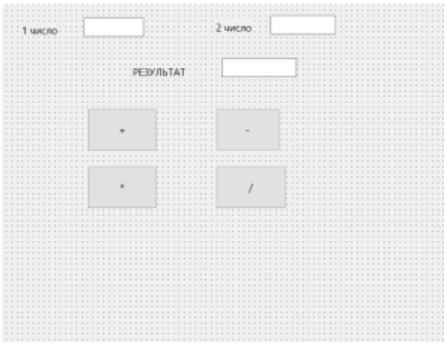


Рисунок 31:Вид формы Простейший Калькулятор

6. Напишем процедуру-обработчик нажатия по кнопке умножения (Button1). Для этого в окне инспектора объектов выбираем закладку События и находим событие OnClick (это и есть нажатие по кнопке).
7. Наберем следующий текст внутри процедуры-обработчика:

```
procedure TForm1.Button1Click(Sender: TObject);
var a,b,result:real; // Объявление локальных переменных вещественного типа
begin
   a:=StrToFloat(edit1.Text);
   b:=StrToFloat(edit2.Text);
   result:=a*b;
   edit3.Text:=FloatToStr(result);
end;
```

7. Аналогично напишем процедуру-обработчик нажатия по кнопке деления
8. Сохраним
9. Выполним программу

Работа 6. Времена года

Цели:

1. Изучить компоненты TTimer, TImage и TShape
2. Создать формы, используя компоненты TTimer, TImage и TShape

Задания

1. На форме расположить две фигуры Shape1 и Shape2 разной формы.

2. Изобразить с помощью компонентов TShape времена года.

Ход работы:

1. На форме расположить две фигуры Shape1 и Shape2 разной формы.

Задать их цвета с помощью ColorBox1 и ColorBox2 соответственно.

Пример интерфейса

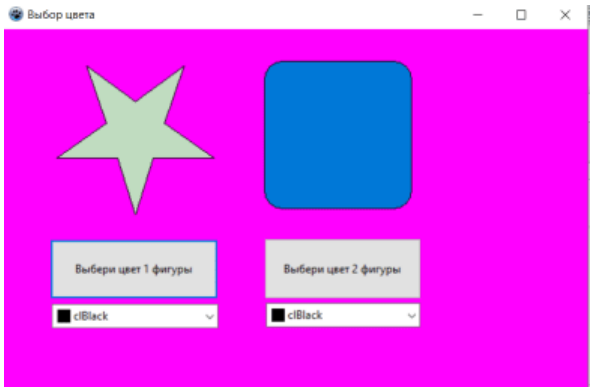


Рисунок 32:Фигуры на форме

Пример программы.

Создайте процедуры для кнопок Button1 и Button2, вставьте в них следующие строки.

```
Shape1.Brush.Color := ColorBox1.Selected;
```

2. Изобразить с помощью компонентов TShape времена года.

Создать четыре кнопки: Зима, Осень, Весна, Лето, использовать картинки:

 .

Рисунок 33:Рисунки

Зима: добавить солнце.
Весна: добавить солнце и белое облако.
Лето: солнце.
Осень: добавить луну, звезды.

Пример интерфейса

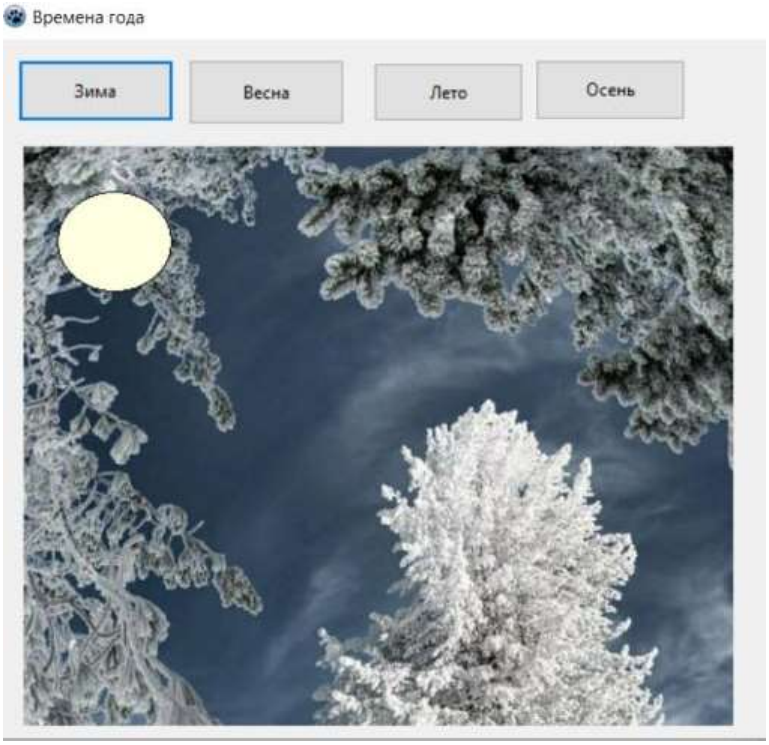


Рисунок 34:Пример интерфейса

Пример программы.

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

   image2.Visible:=false;
……
   image1.Visible:=True;
  Shape1.Visible := True;
……
end;
```

3.* Создать дополнительно две кнопки: Ветер, Штиль во вкладке Весна.

Ветер: облако плывет по небу.
Штиль: облако останавливается.
Указание к выполнению: использовать компонент TTimer.

Работа 7. Блокнот

Цель

закрепление пройденного материала, показать на практике реализацию текстового редактора - прототипа стандартного Блокнота Windows. Закрепление работы с диалогами, модулями и компонентом TMemo.

Задание. Создайте приложение Блокнот.

В приложении должно быть меню, содержащее пункты: Файл -Открыть, Файл-Сохранить, Шрифт. Приложение открывает текстовый файл на диске с использованием компонента TOpenDialog, записывает текст из файла в объект Memo на форме. После редактирования и форматирования текст необходимо сохранить. Для выбора файла для сохранения использовать компонент TSaveDialog, при форматировании шрифта — компонент TFontDialog.

При работе с текстовыми файлами следует иметь в виду, что при выводе в TMemo файла с русским текстом, в Windows буквы могут отображаться некорректно. Это вызвано тем, что текстовые файлы в ОС Windows имеют кодировку CP-1251. Перед выводом текст необходимо преобразовать в кодировку UTF8.

Для преобразования символов из одной кодировки в другую можно использовать функции: SysToUTF8() и UTFToSys().

Ход работы

1. Создайте форму и разместите на ней компоненты MainMenu, Memo1. Компонент MainMenu – невизуальный компонент, его значок можно поместить в произвольном месте.

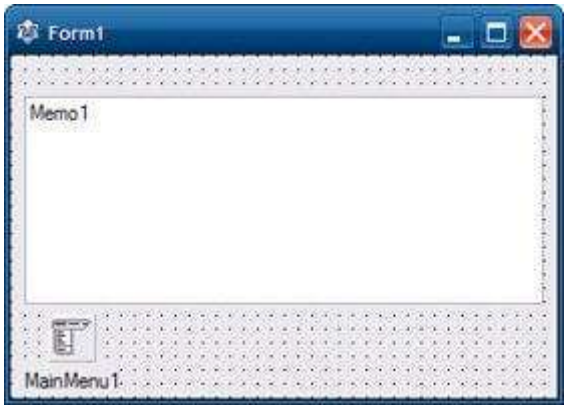


Рисунок 35:Компоненты MainMenu, Memo1

2. Чтобы начать формирование пунктов меню, дважды щелкните по компоненту TMainMenu1. Откроется специальное окно Редактор меню.

Определите пункты меню.

Меню первого уровне содержит два пункта: New Item1 и New Item2 . Первый пункт создается автоматически. Чтобы на этом уровне добавить второй пункт, откройте контекстное меню существующего пункта и выполните команду Вставить новый пункт (после). Появиться новый пункт New Item2.

Меню New Item1 содержит пункты : New Item3 и New Item4. Откройте контекстное меню пункта New Item1 и выполните команду Создать подменю. В

подменю появиться пункт New Item3. Для него откройте контекстном меню и выполните команду Создать новый пункт (после). Появиться пункт New Item4.

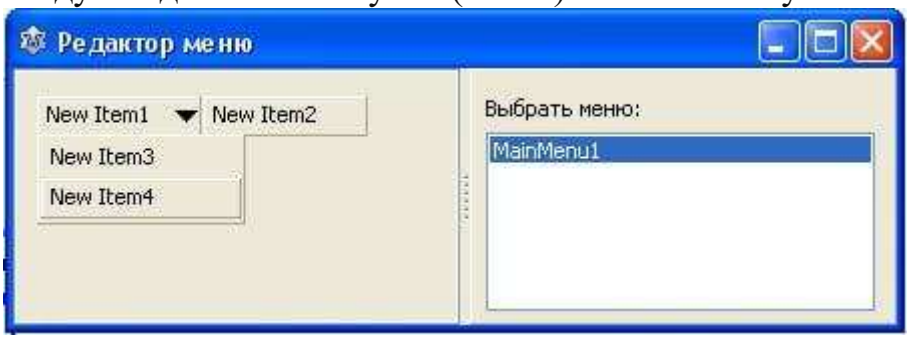


Рисунок 36:Редактор меню

3. Значения свойств установите в соответствии с таблицей.

Компонент	Свойство	Значение
MenuItem1	Caption	Файл
MenuItem2	Caption	Шрифт
MenuItem3	Caption	Открыть
MenuItem4	Caption	Сохранить
Form1	Caption	Текстовый редактор
Memo1	ScrollBars	ssVertical

4. Удалите текст «Memo1» из окна Memo1.

Для этого:

- выберите в окне Инспектор объектов объект Memo1;
- на странице Свойства в строке Lines дважды щелкните на значении String или на кнопке с многоточием для формирования и редактирования текста;
- в окне Диалог ввода строк удалите текст «Memo1» и щелкните мышью на кнопке ОК.

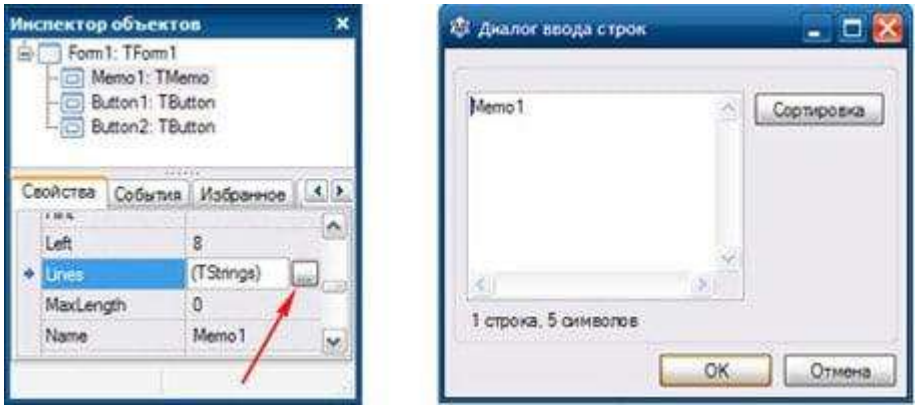


Рисунок 36:Инспектор объектов. Диалог ввода строк

5. Выберите в палитре компонентов вкладку Dialogs и поместите на форму компоненты OpenDialog, SaveDialog, FontDialog.

Эти компоненты невизуальные, разместить их в нижней части формы рядом со значком TMainMenu.

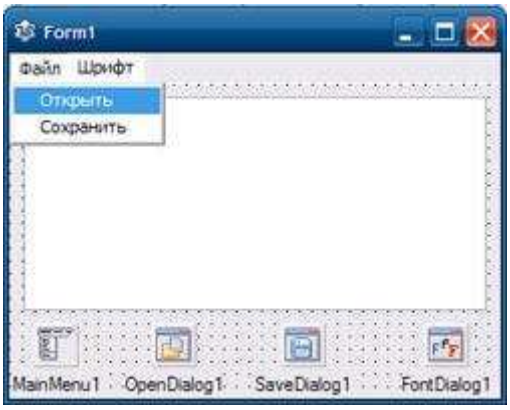


Рисунок 37:Компоненты OpenDialog, SaveDialog, FontDialog

6. Настройте свойства компонентов OpenDialog, SaveDialog

Компонент	Свойство	Значение
OpenDialog1	Title	Открыть
SaveDialog1	Title	Сохранить

7. Чтобы реализовать выбор типа файла при открытии файла в окне диалога:

- выберите в окне Инспектор объектов объект OpenDialog1;
- на странице свойства дважды щелкните мышью по списку значений свойства Filter.
- в окне Редактор фильтра (Filter Edit) задайте фильтры для выбора типа и расширения файла.

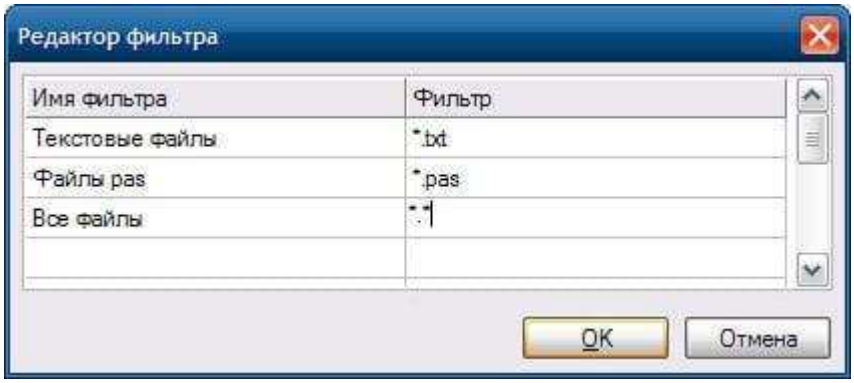


Рисунок 38:Редактор фильтра

8. Чтобы установить в качестве расширения файла первый вариант (.txt), задайте для свойства OpenDialog1.FilterIndex значение 1.

9. Для диалога SaveDialog задайте свойство DefaultExt = txt (чтобы расширение txt автоматически добавлялось к создаваемому файлу).

10. В разделе implementation после директивы { TForm1 } запишите программный код процедуры Ansi_Memo, реализующей загрузку текста из файла с системной кодировкой (кодировка Ansi) в memo-поле (кодировка UTF8), а также процедуры Memo_Ansi, сохраняющей текст из memo-поля в файле на диске.

```
procedure Ansi_Memo(File_Ansi: string);
// Загрузка текста из файла в memo-поле на форме
var
tfile: TStringList;
```

```
str: string;
begin
tfile:= TStringList.Create; // создать список строк
// загрузить текст из файла в список строк
tfile.LoadFromFile(File_Ansi);
str:= tfile.Text;
// загрузить текст из списка в memo-поле
Form1.Memo1.Lines.Add(str);
tfile.Free;
end;
procedure Memo_Ansi(File_Ansi: string);
// Сохранение текста из memo-поля в файле на диске
var
tfile: TStringList;
str: string;
begin
tfile:= TStringList.Create; // создать список строк
str:=Form1.Memo1.text;
// преобразовать текст в системную кодировку
str:= UTF8ToSys(str);
tfile.Add(str);
// сохранить в файле
tfile.SaveToFile(File_Ansi);
tfile.Free;
end;
```

11. Напишите программный код для процедуры обработчика щелчка на пункте меню Файл – Открыть. Для этого дважды щелкните на данном пункте меню.

```
procedure TForm1.MenuItem3Click(Sender: TObject);//Файл - открыть
var
File_Ansi:string;
begin
if OpenDialog1.Execute then
begin
File_Ansi:= OpenDialog1.FileName;
File_Ansi:= UTF8ToSys(File_Ansi);
Ansi_Memo(File_Ansi);
end;
end;
```

После выбора пользователем файла в свойстве OpenDialog1.FileName будет находиться имя файла вместе с путем к нему.

Обратите внимание на оператор:

fname:= UTF8ToSys(fname).

Если имя файла, а также путь содержит кириллицу, то необходимо строку с именем файла преобразовать в системную кодировку.

12. Напишите программный код процедуры обработчика щелчка на пункте меню Файл-Сохранить:

```
procedure TForm1.MenuItem4Click(Sender: TObject); //Файл - Сохранить
var
File_Ansi:string;
begin
if SaveDialog1.Execute then
begin
File_Ansi:=SaveDialog1.FileName;
File_Ansi:= UTF8ToSys(File_Ansi);
Memo_Ansi(File_Ansi);
end;
end;
```

Приложение открывает диалоговое окно «Сохранить», в котором задается имя файла. Имя файла из свойства SaveDialog1.FileName запоминается в переменной FName. В заключительной части процедуры оператор Memo1.Lines.SaveToFile(FName); используется для записи в файл содержимого свойства Lines объекта Memo1

13. Напишите программный код процедуры обработчика щелчка на пункте меню Шрифт:

```
procedure TForm1.MenuItem2Click(Sender: TObject);
begin
FontDialog1.Font:= memo1.Font;
if FontDialog1.execute=true then
begin
Form1.Memo1.Font := FontDialog1.Font;
end;
end;
```

14. Сохраните, откомпилируйте и запустите на выполнение созданное приложение.

15. Щелкните мышью на кнопке Открыть, в диалоговом окне Открыть текстовый файл выберите папку, задайте тип файла и выберите текстовый файл, после чего нажмите кнопку Открыть.

16. Отредактируйте текст в окне приложения и нажмите кнопку Сохранить.

После этого в диалоговом окне сохранить текстовый файл выберите в поле тип файла расширение для сохраняемого файла, задайте его имя и щелкните мышью по кнопке Сохранить.

17. Открыв в окне Проводника папку, в которой был сохранен файл, убедитесь, что файл с указанным вами именем в ней есть.

18. Измените размер шрифта и цвет в диалоговом окне Шрифт.

Работа 8. Создание теста

Цели

научить использовать объекты ввода-вывода данных панели Standart, организовывать диалог с пользователем, оформлять внешний вид приложения.

Задание. Создать приложение Тест

Ход работы

1. Запускаем Lazarus
2. Изменим заголовок окна
3. Сделаем заголовок теста (вопрос).

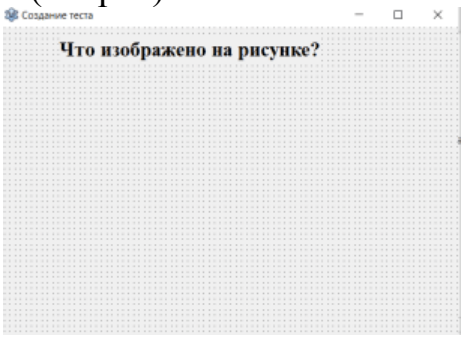


Рисунок 38:Заголовок формы и вопрос теста

4. Разместим изображение устройства вывода информации. Для этого:

найдем сверху на вкладке Additional значок

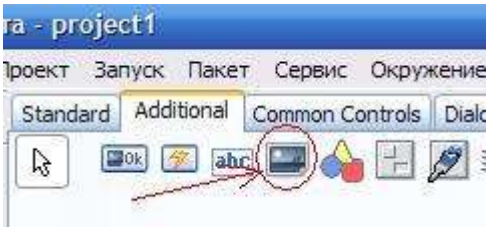


Рисунок 39:Панель для размещения изображения

щелкнем левой кнопкой мыши по этому значку и на нашей форме ниже размещенного вопроса

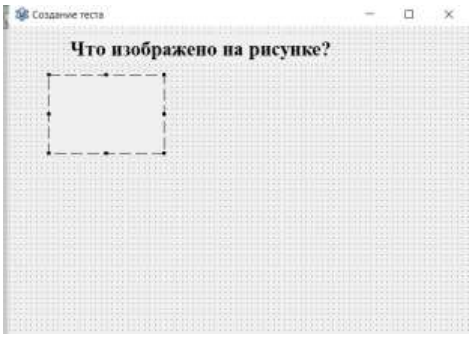


Рисунок 40:Отводим место под изображение

укажем изображение, которое будет внутри: щелкнем по размещенной только что рамке и слева в окне “Инспектор объектов” находим свойство Picture (изображение), щелкаем по нему, нажимаем кнопочку справа от него

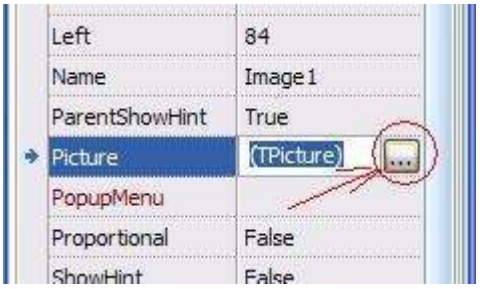


Рисунок 41:Выбор изображения

Рисунок 42:Форма с изображением

5. Разместим варианты ответов. Для этого найдем сверху на вкладке Standard значок

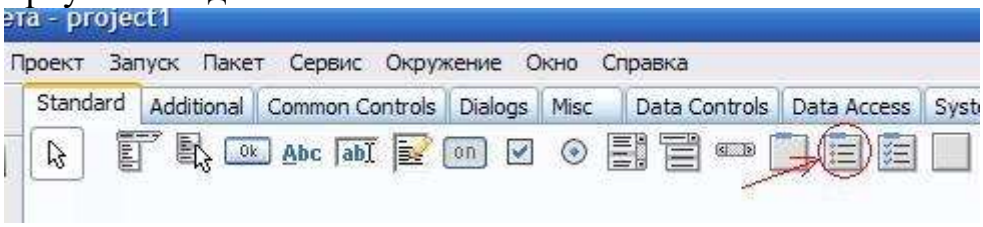


Рисунок 43:Кнопка на панели для размещения вариантов ответа

щелкнем по нему левой кнопкой мыши, а затем щелкнем на нашей форме правее рисунка, получим:

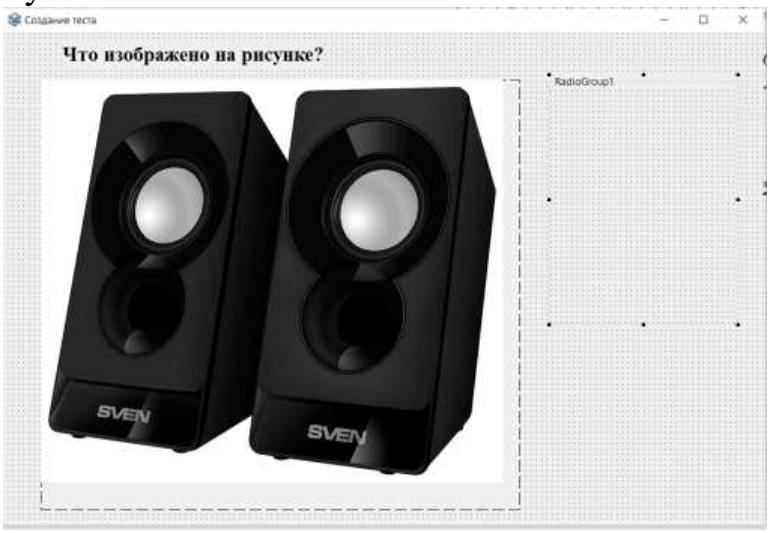


Рисунок 43:Выделение области для размещения вариантов ответа

найдем слева в окне “Инспектор объектов” свойство Caption и введем заголовок “Варианты ответов”

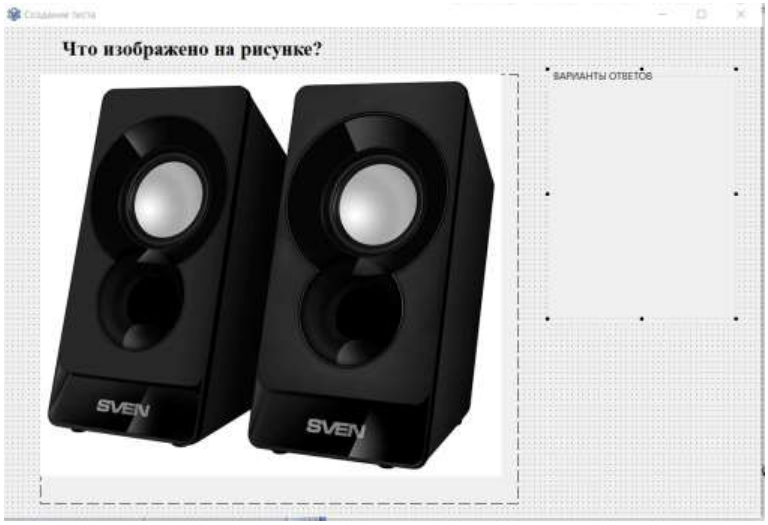


Рисунок 44:Появилась область ВАРИАНТЫ ОТВЕТОВ

введем сами варианты ответов: найдем свойство Items, щелкнем по нему, а затем по кнопочке справа от него

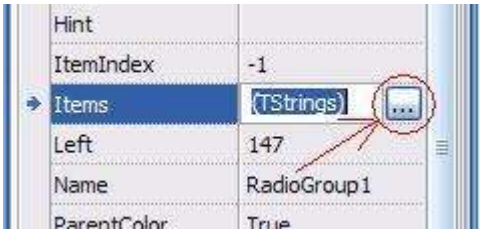


Рисунок 45:Свойство для ввода ответов Items

в появившемся окне вводим варианты ответов – каждый на отдельной строке

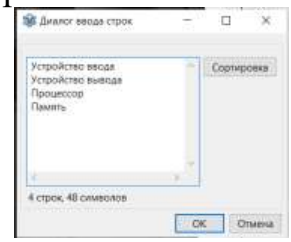


Рисунок 46:Ввод вариантов ответов

нажмем ОК и получим набор строк, из которые после запуска программы сможет быть выбрана только одна

Рисунок 47:Вопрос с вариантами ответов

6. Сделаем кнопку, нажав на которую пользователь покажет программе, что выбрал вариант ответа

Рисунок 48:Итоговый вид формы

8. Сохраним наш проект. Для этого нужно

9. Теперь нужно написать обработку нажатия на кнопку «Я выбрал правильный ответ». Для этого дважды щелкнем по этой кнопке мышкой. В тексте программы появится обработчик нажатия на кнопку:

```
    Image1: TImage;
    Label1: TLabel;
    RadioGroup1: TRadioGroup;
    procedure Button1Click(Sender: TObject);
  private
    { private declarations }
  public
    { public declarations }
  end;

var
  Form1: TForm1;

implementation

{ TForm1 }

procedure TForm1.Button1Click(Sender: TObject);
begin

end;

initialization
  {$I unit1.lrs}

end.
```

Введем внутри обработчика такой текст:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  if RadioGroup1.ItemIndex=-1 then
       ShowMessage('Нет ответа')
  else if RadioGroup1.ItemIndex=1 then
            ShowMessage('Правильно')
  else ShowMessage('Неправильно');
end;
```

Работа 9. Создание нескольких форм

Цели:

познакомить учащихся с формой и компонентами, их основными свойствами и методами; развивать умения выполнять действия с формой и компонентами; воспитывать эстетические навыки при оформлении формы и компонентов.

Задание. Создать приложение с тремя формами: «Главная», «Калькулятор» и «О программе»

Форму Опции вызывать в обычном окне. Для вызова формы «О программе» использовать модальное окно. На рисунке показаны главная форма и подформы нашего нового проекта.

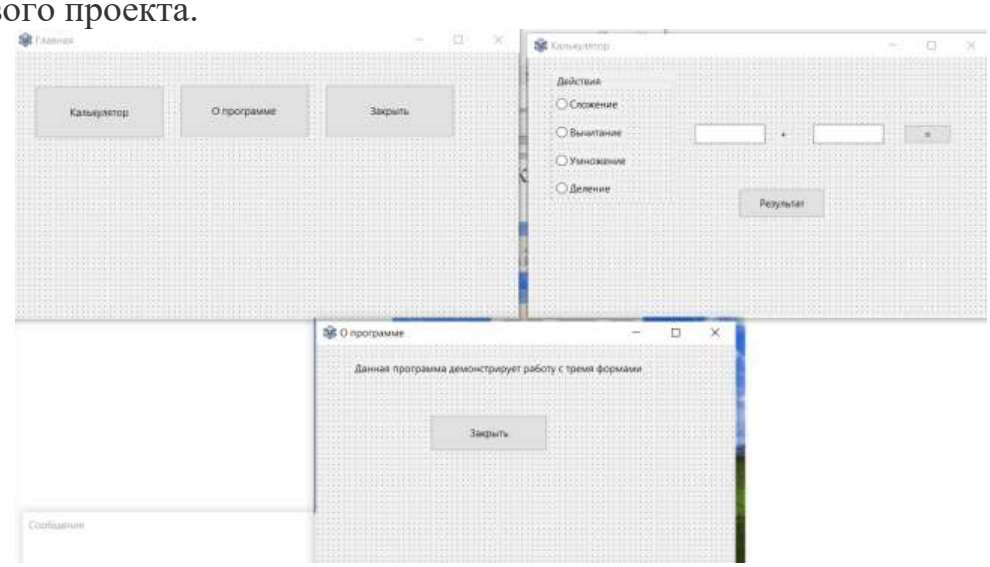


Рисунок 49: Формы и подформы проекта

Ход работы

Форма является объектом, отсутствующим на палитре компонентов. Чтобы добавить новую форму в проект, нужно выбрать команду **Файл → Создать форму** или щелкнуть кнопку **Создать форму** на панели инструментов.

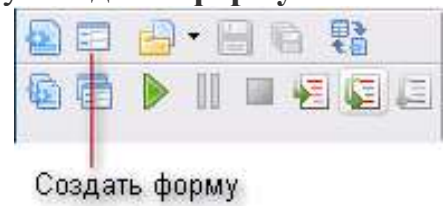


Рисунок 50: Создание форм

Появится новая пустая форма. Называться она будет Form2, а соответствующий ей файл с исходными текстами добавиться в Редактор кода на новую вкладку Unit2.

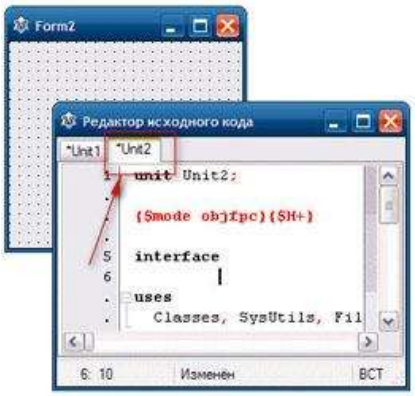


Рисунок 51:Редактор кода

После добавления новой формы, проект нужно сохранить.

Каждую форму, включаемую в приложение, необходимо, прежде всего, создать описанным способом. Создание формы не означает ее немедленного отображения, а только выделение и инициализацию памяти для нее. Все формы создаются неявно при запуске приложения, но отображается автоматически только главная форма, а остальные остаются скрытыми. Впоследствии их можно отобразить вызовом метода **Show**, например:

Form2.Show;

Если же требуется отобразить окно как модальное, то есть так, чтобы, не закрыв его, пользователь не мог переключиться на другое окно этого же приложения, то вместо метода Show следует использовать метод ShowModal:

Form2.ShowModal;

Окна, открытые с помощью методов *Show* и *ShowModal*, можно вновь скрыть при помощи метода **Hide**:

Form2.Hide;

Форма окна, скрытого методом Hide, не уничтожается, и это окно в любой момент можно снова отобразить. Все объекты форм уничтожаются (освобождают память) автоматически при завершении работы приложения.

Основные методы формы:

1. **Close** — закрывает форму;
2. **Hide** — форма становится невидимой;
3. **Show** — показать форму;
4. **ShowModal** — показать форму в модальном режиме. Когда форма показана в модальном режиме, приложение не может выполняться, пока форма не будет закрыта.

Т.е. для показа форм можно использовать один из двух методов: Show или ShowModal. Метод Show предназначен для показа формы в обычном окне, а ShowModal - для показа формы в модальном окне.

Различие между этими двумя видами окон состоит в том, что между обычными окнами можно перемещаться произвольным способом, а перейти в другое окно из модального окна можно только после его закрытия.

Модальные окна хорошо подходят для задания всевозможных настроек, выполнения ввода промежуточных значений, отображения результатов.

1. Создайте новое приложение
2. Разместите на форме Form1 3 кнопки

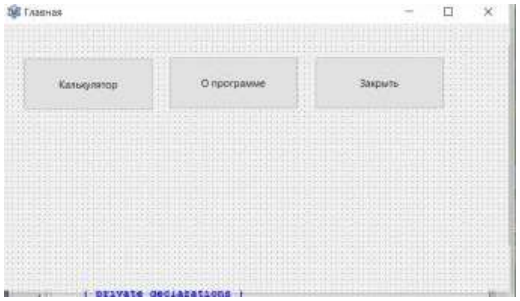


Рисунок 52:Форма Главная

3. Создайте формуForm2. Для этого выберите в меню **Файл →Создать форму**. На экране появиться новая форма Form2, в редакторе кода – новая вкладка Unit2.

4. Измените заголовок формы Form2 на «Калькулятор». Установите на форме компонент RadioGroup. Создайте интерфейс программы, так как показано на рисунке. Т.е. RadioGroup для выбора математического действия, 2 метки, 2 поля вывода, две кнопки.

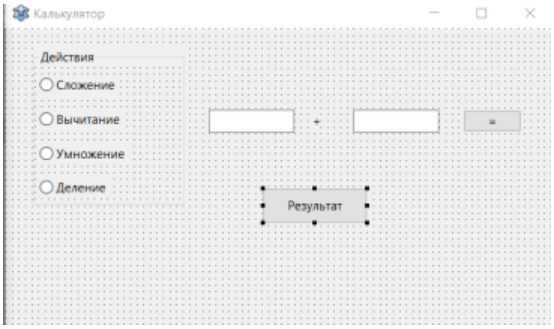


Рисунок 52:Форма Калькулятор

5. После размещения на форме компонента RradioGroup входящие в него переключатели задаются перечислением их названий. Эти названия вводятся в свойство Items. Так как требуется ввести не одну строку, а несколько, для их ввода предусмотрен специальный редактор, который вызывается щелчком на специальной кнопке  , расположенной справа в строке, описывающей свойство Items.

6. Создайте еще одну форму - Form3 – О программе

7. Разместите на Form3 объекты Надпись и Кнопка.

Рисунок 53:Форма О программе

8. Написать обработчики событий для кнопок формы Главная. Первая кнопка формы Главная (кнопка Калькулятор) вызывает форму Калькулятор в обычном окне с помощью метода Show.

```
procedure TForm1.Button1Click(Sender: TObject);
begin
Form2.Show;
end;
```

9. Вторая кнопка формы Главная (кнопка О программе) вызывает форму О программе в модальном окне с помощью метода ShowModal.

```
procedure TForm1.Button2Click(Sender: TObject);
begin
Form3.ShowModal;
end;
```

10. Третья кнопка формы Главная (кнопка Закрыть) закрывает главное окно.

procedure TForm1.Button3Click(Sender: TObject);
begin
Close;
end;

11. Открываем программный код формы Калькулятор (модуль Unit2). В модуле Unit2 в разделе **implementation** записать директиву **uses:**

uses Unit1;

Это необходимо для того чтобы главный модуль Unit1 формы Главная был видим в этом модуле.

12. Калькулятор и форма Главная становиться вновь видимой.

procedure TForm2.Button2Click(Sender: TObject);
begin
Form2.Close;
Form1.Hide;
end;

13. Переходим в программный код формы О программе (модуль Unit3).В модуле Unit3 в разделе **implementation** записать директиву **uses**.

uses Unit1;

Модуль Unit1 формы Главная должен был видим в этом модуле.

14. Кнопка ОК формы О программе закрывает окно.

procedure TForm3.Button1Click(Sender: TObject);
begin
Close;
end;

15. Проект готов. Сохраните проект и проверьте его работу.

Работа 10. Обработка массива.

Цель:. работа с массивами, знакомство с компонентами Memo, StringGrid.

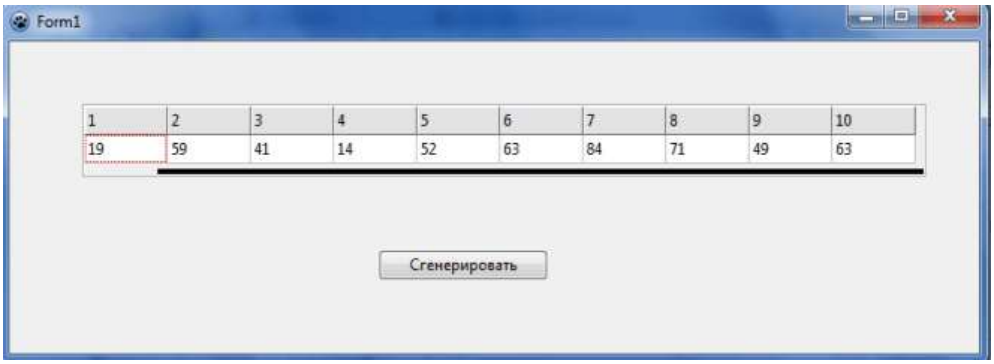


Рисунок 54:Форма, содержащая массив из 10 чисел

Задания.

1. Создать программу, генерирующую ряд из десяти чисел

Ход работы:

1. Откройте новый проект.
2. Разместите на форме экземпляры компонентов: кнопку Button, объект StringGrid из вкладки Additional.
3. Переименуйте кнопку согласно картинке выше.
4. Выделите объект StringGrid, перейдите в Object Inspector на вкладку Properties, найдите свойство ColCount и задайте ему значение 10.
   Задается количество столбцов в таблице, где будут генерироваться числа.
5. Найдите свойство RowCount и задайте ему значение 2.
   Задается количество строк в таблице, где будут генерироваться числа.
6. Дважды щелкните на объекте StringGrid и пронумируйте столбцы таблицы.
7. Выделите объект Button. Перейдите в окно редактора кода и пропишите следующий код:
   ArrayCreate;

   Данный код запускает генератор случайных чисел в массиве.
8. Дважды кликните на форму и пропишите следующий код:
   ArrayCreate;
9. Перейдите в редактор исходного кода для формы ArrayCreate и пропишите следующий код:

```
Randomize;
for i := 0 to 9 do
begin
a[i + 1] := Random(100);
StringGrid1.Cells[i, 0] := FloatToStrF(i + 1, ffFixed, 2, 0);
StringGrid1.Cells[i, 1] := FloatToStrF(a[i + 1], ffFixed, 2, 0);
```

end;

10. В окне редактора исходного кода в операторе описания переменных Var пропишите переменную i и массив a и присвойте им тип Integer.
11. Выделите форму, в свойстве Caption окна Object Inspector замените слово Form1 на «Массив».
12. Сохраните проект в новой папке, запустите и протестируйте его.

2. Вычислить среднее арифметическое значение элементов одинарного массива C (5).

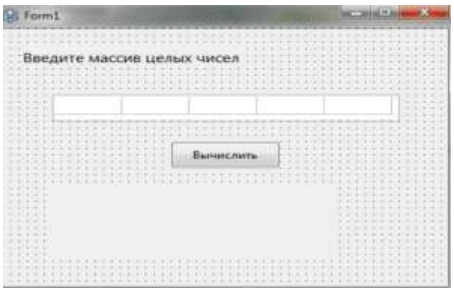


Рисунок 55:Форма, для вычисления среднего арифметического чисел

Для этого поместите на форму компоненты и укажите значения свойств StringGrid1

Свойство	Значение
ColCount	5
FixedCols	0
FixedRows	0
RowCount	1
DefaultRowHeight	24
Height	24
DefaultColWidth	64
Width	328
Options.goEditing	True
Options.AlwaysShowEditing	True
Options.goTabs	True

Рисунок 56: Свойства StringGrid1

Введите текст модуля для события OnClick компонента Button1:

```
procedure TForm1.Button1Click(Sender: TObject);
var a : array[1..5] of integer; // массив
summ: integer; // сумма элементов
sr: real; // среднее арифметическое
i: integer; // индекс
begin // ввод массива
```

```
For i:= 1 to 5 do
if Length(StringGrid1.Cells[i-1,0]) <> 0
then a[i] := StrToInt(StringGrid1.Cells[i-1,0])
else a[i] := 0; // обработка массива
summ := 0;
for i :=1 to 5 do
summ := summ + a[i];
sr := summ / 5;
// вывод результата
Label2.Caption :='Суммаэлементов: ' + IntToStr(summ) + #13+ 'Среднее арифметическое: ' + FloatToStr(sr); end;
```

Работа 11. Светосигнализатор

Цель: создать программу «Светосигнализатор», в которой светосигнализатор будет работать в двух режимах – ручном и автоматическом.

Задания.

Создать приложение моделирующее работу светофора в двух режимах – ручного и автоматического управления.

Предусмотреть настройки периодов действия сигналов.

Ход работы

Задание.

Создать приложение моделирующее работу светофора в двух режимах – ручного и автоматического управления.

Предусмотреть настройки периодов действия сигналов.

Рисунок 57: Результат выполнения работы

Шаг 1. Создание интерфейса программы.

1. Работа с формой. (Form1)

- Создать форму.
- Изменить заголовок на «Светосигнализатор».
- Установить шрифт Arial, размер – 14.

2. На форму поместить две панели:

- Panel1 – для размещения сигналов светофора.

Установить: привязку панели влево (alLeft), нейтральный (можно серый) цвет панели, удалить заголовок (Caption), установить отступ по периметру от 10 до 30 пикселей, соотношение ширины и высоты примерно 1:3.

- Panel2 – для размещения ОУК управления сигналами светофора в ручном и автоматическом режимах.

Установить: привязку – заполнение оставшегося места (alClient), удалить

заголовок, установить отступ по периметру до 30 пикселей. Font для Panel2 – 12 пунктов.

3. На Panel1 поместить три объекта типа TShape (вкладка Additional).

Привязку для: Shape1 – alTop, Shape3 – alBottom, Shape2 – alClient. Установить им форму – свойство Shape (квадрат или круг); цвет – Brush – Color; отступ по периметру для всех сигналов и вертикальные размеры ОУК Shape1 и Shape3 подобрать такими, чтобы все три сигнала светофора были равными по величине.

4. На Panel2 поместить:

- ОУК RadioGroup. Установить для RadioGroup1 следующие свойства: Caption (заголовок) – «Режим:» ; Columns (колонки) – 2; Items (коллекция строк) – «Ручной» и «Авто»; ItemIndex (№ выбранной строки) – 0.
- Panel3 для размещения кнопок ручного управления сигналами светофора. Удалить заголовок. Установить привязку – alTop. Свойство Height (вертикальный размер) – установить от 70 до 80 пунктов.
- Panel4 для размещения ОУК управления в автоматическом режиме сигналами светофора. Удалить заголовок. Установить привязку – alClient.
- Button1 (кнопку «Выход»), с привязкой alBottom, отступом по периметру и отступами слева и справа.

5. На Panel3 разместить три объекта SpeedButton1, SpeedButton2, SpeedButton3, вкладки панели компонент Additional.

- Установить привязку SpeedButton1 – alLeft; SpeedButton3 – alRight; SpeedButton2 – alClient.
- Выделить их и установить свойства: Flat – True, Transparent – False. ПОСЛЕ этого для каждой кнопки управления цветом сигналов светофора установить соответствующий цвет: Красный, Желтый, Зеленый.
- Изменяя свойства BorderSpacing для кнопок Panel3, добиться, чтобы их размеры были визуально примерно равными.

6. На Panel4 разместить три объекта типа SpinEdit вкладки Misc панели компонент.

- Установить привязку SpinEdit1 – alLeft; SpinEdit3 – alRight; SpinEdit2 – alClient.
- Установить каждому компоненту управления длительностью сигналов светофора в автоматическом режиме соответствующий цвет.
- Используя свойства BorderSpacing сделайте ОУК Panel4 визуально примерно равными.
- Для всех ОУК Panel4 устанавливаем свойства Increment = 1; MinValue = 2; MaxValue, соответственно – 50, 5, 50.

7. Поместить на форму ОУК Timer1 вкладки System панели компонент.

Шаг 2. Определение необходимых глобальных переменных, процедур и событий, на которые обязана реагировать Программа.

1. Глобальные переменные:

В принципе, свойства многих ОУК можно использовать в качестве переменных. Тем более, что они определяемы на любом этапе выполнения программы.

Но для режима «Авто» нужна переменная, которая однозначно будет определять режим работы светосигнализатора.

Цвет	К	Ж	З	Ж
R	0	1	2	3

Переменная R будет изменяться от 0 до 3 циклически определяя цвет сигнала.

Длительность сигналов будет считываться из значения свойства Value ОУК SpinEdit1, SpinEdit2, SpinEdit3.

Для контроля длительности сигнала нужен счетчик целого типа, например i.

```
var
  Form1: TForm1;
  R,i : integer;
Implementation
```

Нужна процедура, которая включая цвет соответствующий R, отключает все остальные цвета: Procedure Svet; Её можно вызывать всякий раз когда следует включить один из трёх сигналов светосигнализатора.

Включать сигналы можно в ручном режиме:

События и обработчики: SpeedButton1Click, SpeedButton2Click, SpeedButton3Click.

И в автоматическом режиме – срабатывание таймера – Событийная процедура (обработчик) Timer1Timer.

При переключении режимов Ручной/Авто, срабатывает процедура RadioGroup1Click.

Внимательно анализируя текст модуля Unit1, разберитесь с тем КАК в программе происходит переключение сигналов светосигнализатора и повторите события и их обработчики в своей программе.

ВНИМАНИЕ!

Процедура SVet НЕ ЯВЛЯЕТСЯ событийной – это процедура ПОЛЬЗОВАТЕЛЯ. Она должна располагаться (быть написанной) в модуле ВЫШЕ её вызовов из событийных процедур.

Событийные процедуры (обработчики) создаются ТОЛЬКО через «механизм» Lazarusa.

```
unit Unit1;

{$mode objfpc}{$H+}

interface

uses
  Classes, SysUtils, Forms, Controls, Graphics, Dialogs, ExtCtrls, Buttons,
  StdCtrls, Spin;
```

```
type

  { TForm1 }

  TForm1 = class(TForm)
    Button1: TButton;
    Panel1: TPanel;
    Panel2: TPanel;
    Panel3: TPanel;
    Panel4: TPanel;
    RadioGroup1: TRadioGroup;
    Shape1: TShape;
    Shape2: TShape;
    Shape3: TShape;
    SpeedButton1: TSpeedButton;
    SpeedButton2: TSpeedButton;
    SpeedButton3: TSpeedButton;
    SpinEdit1: TSpinEdit;
    SpinEdit2: TSpinEdit;
    SpinEdit3: TSpinEdit;
    Timer1: TTimer;
    procedure Button1Click(Sender: TObject);
    procedure RadioGroup1Click(Sender: TObject);
    procedure SpeedButton1Click(Sender: TObject);
    procedure SpeedButton2Click(Sender: TObject);
    procedure SpeedButton3Click(Sender: TObject);
    procedure Timer1Timer(Sender: TObject);
  private

  public

  end;

var
  Form1 : TForm1;
  R, i : integer;
implementation

{$R *.lfm}

Procedure Svet;
Begin
  Form1.Shape1.Brush.Color:=clSilver;
  Form1.Shape2.Brush.Color:=clSilver;
  Form1.Shape3.Brush.Color:=clSilver;
```

```
  case R of
    0 : Form1.Shape1.Brush.Color:=clRed;
  1,3 : Form1.Shape2.Brush.Color:=clYellow;
    2 : Form1.Shape3.Brush.Color:=clLime;
  end;
end;

{ TForm1 }

procedure TForm1.SpeedButton1Click(Sender: TObject);
begin       // красный
  R:=0;
  Svet;
end;

procedure TForm1.Button1Click(Sender: TObject);
begin     // остановка программы
  Halt;
end;

procedure TForm1.RadioGroup1Click(Sender: TObject);
begin       // переключение режимов
  case Form1.RadioGroup1.ItemIndex of   // ручной
    0 : begin
          Form1.Panel4.Enabled:=False;
          Form1.Panel3.Enabled:=True;
          Form1.Timer1.Enabled:=False;
        end;
    1 : begin                       // автоматический
          Form1.Panel4.Enabled:=True;
          Form1.Panel3.Enabled:=False;
          i:=0;
          Form1.Timer1.Enabled:=True;
        end;
  end;
end;

procedure TForm1.SpeedButton2Click(Sender: TObject);
begin
  R:=1;
  Svet;
end;

procedure TForm1.SpeedButton3Click(Sender: TObject);
begin
  R:=2;
```

```
  Svet;
end;

procedure TForm1.Timer1Timer(Sender: TObject);
begin
  Case R of
    0 : if i=Form1.SpinEdit1.Value then begin i:=0; inc(R); end;
    1 : if i=Form1.SpinEdit2.Value then begin i:=0; inc(R); end;
    2 : if i=Form1.SpinEdit3.Value then begin i:=0; inc(R); end;
    3 : if i=Form1.SpinEdit2.Value then begin i:=0; R:=0; end;
  end;
  Inc(i);
  Svet;
end;

end.
```

Работа 12. Редактор простых изображений

Цель

Рассмотреть примеры: использования Главного меню, объектов TSpeedButton, TPanel, TImage, сохранения файла изображения и распечатки на принтере – объекты вкладки Dialogs.

Задание.

Написать редактор для рисования картинок

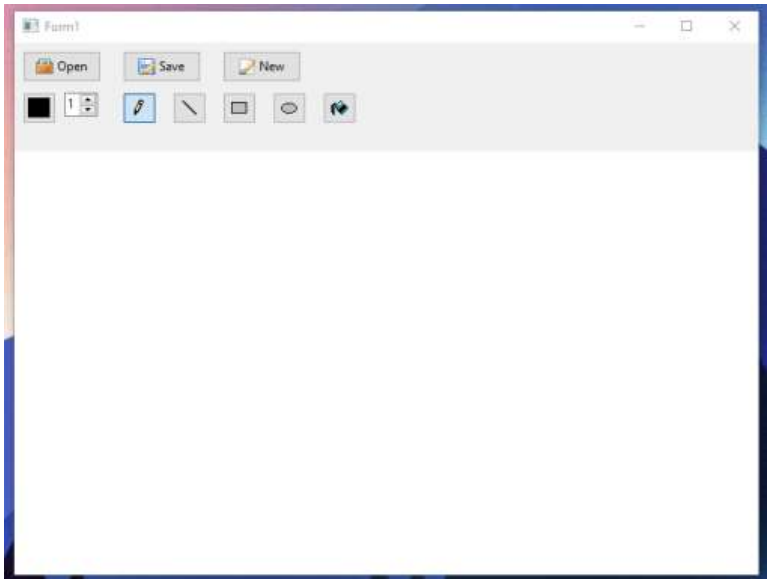


Рисунок 58: Результат выполнения работы

Ход работы

Шаг 1: **Создание интерфейса программы**

1. Работа с формой. (Form1)

- Создать форму.

2. На форму поместить:

- 3 TbitBtn. Нужны для работы с файлами. Меняем им caption на: Open, Save, New (для изменения текста на кнопке). А Name на btnOpen, btnSave, btnNew (для удобства). В свойстве Glyph добавляем им картинки. Располагаем их сверху слева по порядку.
- 1 TColorButton. Нужен для изменения цвета. Делаем его маленьким и располагаем под кнопкой Open.
- 1 TSpinEdit. Нужен для определения размеров кистей. Делаем его маленьким и располагаем чуть правее TColorButton. Меняем свойства: Value ставим 1, MinValue ставим 1, MaxValue ставим 100.
- 5 TSpeedButton. Нужны для выюора кисти. Свойство Name меняем им на ToolPencil, ToolLine, ToolRect, ToolOval, ToolFill. В свойстве Glyph добавляем им картинки. Располагаем их чуть ниже кнопок Save, New. Для всех кнопок меняем свойсто GroupIndex на 1. Для кнопки ToolPencil меняем свойство Down на True.
- 1 TOpenDialog. Нужен для того, чтобы открывать файлы. Располагаем где угодно. Меняем свойство Title на Открыть существующий файл. С свойстве Filter

прописываем Bitmap Files (*.bmp)|*.bmp (файлы с таким расширение он будет открывать).

- 1 TSaveDialog. Нужен для сохранения файлов. Распологаем где угодно. Меняем свойство Title на Сохранить файл как. С свойстве Filter прописываем Bitmap Files (*.bmp)|*.bmp (файлы с таким расширение он будет открывать). Свойство FileName можно написать имя файла по умолчанию.
- 1 TPaintBox. Это сама зона рисования. Свойства Height = 408, Left = 0, Top = 104, Width = 729. Меняем Name на MyCanvas.

Шаг 2: **Написание кода**

```
unit Unit1;
{$mode objfpc}{$H+}
interface
uses
  Classes, SysUtils, Forms, Controls, Graphics, Dialogs, Buttons, ExtCtrls,
  Spin;
type
  { TForm1 }
  TForm1 = class(TForm)
    btnNew: TBitBtn;
    btnSave: TBitBtn;
    btnOpen: TBitBtn;
    ColorButton1: TColorButton;
    OpenDialog1: TOpenDialog;
    MyCanvas: TPaintBox;
    SaveDialog1: TSaveDialog;
    SpinEdit1: TSpinEdit;
    ToolFill: TSpeedButton;
    ToolPencil: TSpeedButton;
    ToolLine: TSpeedButton;
    ToolRect: TSpeedButton;
    ToolOval: TSpeedButton;
    procedure btnNewClick(Sender: TObject);
    procedure btnOpenClick(Sender: TObject);
    procedure btnSaveClick(Sender: TObject);
    procedure ColorButton1ColorChanged(Sender: TObject);
    procedure FormClose(Sender: TObject; var CloseAction: TCloseAction);
    procedure FormCreate(Sender: TObject);
    procedure FormPaint(Sender: TObject);
    procedure MyCanvasMouseDown(Sender: TObject; Button: TMouseButton;
      Shift: TShiftState; X, Y: Integer);
    procedure MyCanvasMouseMove(Sender: TObject; Shift: TShiftState; X,
      Y: Integer);
    procedure MyCanvasMouseUp(Sender: TObject; Button: TMouseButton;
      Shift: TShiftState; X, Y: Integer);
    procedure MyCanvasPaint(Sender: TObject);
    procedure SpinEdit1Change(Sender: TObject);
```

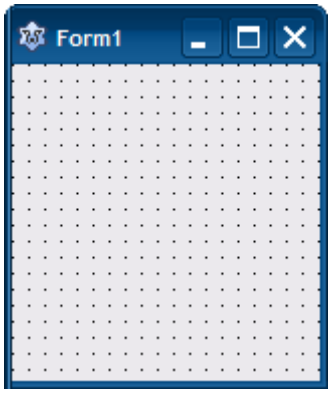
```
  private
  public
  end;
var
  Form1: TForm1;
  paintbmp: tbitmap;
  MouseIsDown: boolean;
  PrevX, PrevY: integer;
implementation
{$R *.lfm}
{ TForm1 }
procedure TForm1.FormPaint(Sender: TObject);
begin
end;
procedure TForm1.FormClose(Sender: TObject; var CloseAction: TCloseAction);
begin
  paintbmp.Free;
end;
procedure TForm1.btnSaveClick(Sender: TObject);
begin
  SaveDialog1.Execute;
  if SaveDialog1.Files.Count > 0 then begin
       //Добавляем .bmp
    if RightStr(SaveDialog1.FileName, 4) <> '.bmp' then
      SaveDialog1.FileName:=SaveDialog1.FileName+'.bmp';
          //Сохраняем
    paintbmp.SaveToFile(SaveDialog1.FileName);
  end;
end;
procedure TForm1.btnNewClick(Sender: TObject);
begin
  if paintbmp <> nil then
     paintbmp.Destroy;
    paintbmp := TBitmap.Create;
    paintbmp.SetSize(Screen.Width, Screen.Height);
    paintbmp.Canvas.FillRect(0,0,paintbmp.Width,paintbmp.Height);
    paintbmp.Canvas.Brush.Style:=bsClear;
    MyCanvas.Canvas.Brush.Style:=bsClear;
    paintbmp.Canvas.Pen.Color:=ColorButton1.ButtonColor;
    paintbmp.Canvas.Pen.Width:=SpinEdit1.Value;
    MyCanvasPaint(Sender);
end;
procedure TForm1.btnOpenClick(Sender: TObject);
begin
   OpenDialog1.Execute;
  if (OpenDialog1.Files.Count > 0) then begin
```

```
      paintbmp.LoadFromFile(OpenDialog1.FileName);
      MyCanvasPaint(Sender);
    end;
  end;
  procedure TForm1.ColorButton1ColorChanged(Sender: TObject);
  begin
    MyCanvas.Canvas.Pen.Color := ColorButton1.ButtonColor;
    paintbmp.Canvas.Pen.Color := ColorButton1.ButtonColor;
  end;
  procedure TForm1.FormCreate(Sender: TObject);
  begin
    btnNewClick(Sender);
  end;
  procedure TForm1.MyCanvasMouseDown(Sender: TObject; Button: TMouseButton;
    Shift: TShiftState; X, Y: Integer);
  begin
    if Button = mbLeft then MouseIsDown := true;
    PrevX := X;
    PrevY := Y;
  end;
  procedure TForm1.MyCanvasMouseMove(Sender: TObject; Shift: TShiftState; X,
    Y: Integer);
  begin
    if MouseIsDown then begin
      if ToolPencil.Down = true then begin
        //кисть
        paintbmp.Canvas.Line(PrevX, PrevY, X, Y);
        MyCanvas.Canvas.Line(PrevX, PrevY, X, Y);
        PrevX:=X;
        PrevY:=Y;
      end else if ToolLine.Down = true then begin
        //линия
        MyCanvasPaint(Sender);
        MyCanvas.Canvas.Line(PrevX, PrevY, X, Y);
      end else if ToolRect.Down = true then begin
        //прямоугольнирк
        MyCanvasPaint(Sender);
        MyCanvas.Canvas.Rectangle(PrevX, PrevY, X, Y);
      end else if ToolOval.Down = true then begin
        //круг
        MyCanvasPaint(Sender);
        MyCanvas.Canvas.Ellipse(PrevX, PrevY, X, Y);
      end;
    end;
  end;
  procedure TForm1.MyCanvasMouseUp(Sender: TObject; Button: TMouseButton;
```

```
  Shift: TShiftState; X, Y: Integer);
var
  TempColor: TColor;
begin
  if MouseIsDown then begin
  if ToolLine.Down = true then begin
      paintbmp.Canvas.Line(PrevX, PrevY, X, Y);
  end else if ToolRect.Down = true then begin
      paintbmp.Canvas.Rectangle(PrevX, PrevY, X, Y);
    end else if ToolOval.Down = true then begin
      paintbmp.Canvas.Ellipse(PrevX, PrevY, X, Y);
    end else if ToolFill.Down = true then begin
      TempColor := paintbmp.Canvas.Pixels[X, Y];
      paintbmp.Canvas.Brush.Style := bsSolid;
      paintbmp.Canvas.Brush.Color := ColorButton1.ButtonColor;
      paintbmp.Canvas.FloodFill(X, Y, TempColor, fsSurface);
      paintbmp.Canvas.Brush.Style := bsClear;
      MyCanvasPaint(Sender);
    end;
  if Button = mbLeft then MouseIsDown := false;
end;
end;
procedure TForm1.MyCanvasPaint(Sender: TObject);
begin
       if MyCanvas.Width <> paintbmp.Width then begin
    MyCanvas.Width:=paintbmp.Width;
    Exit;
  end;
 if MyCanvas.Height<>paintbmp.Height then begin
    MyCanvas.Height:=paintbmp.Height;
   Exit;
  end;
  MyCanvas.Canvas.Draw(0,0,paintbmp);
end;
procedure TForm1.SpinEdit1Change(Sender: TObject);
begin
  MyCanvas.Canvas.Pen.Width := SpinEdit1.Value;
  paintbmp.Canvas.Pen.Width:=SpinEdit1.Value;
end;
end.
```

Краткий справочник

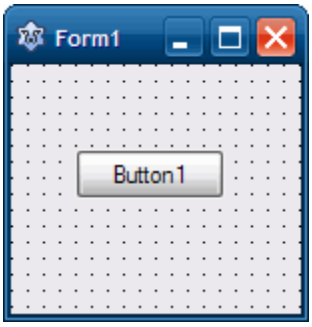
Компонент Form, является основой программы. Свойства формы определяют вид окна программы.



Основные свойства формы

Свойство	Описание
Name	Имя формы. В программе имя формы используется для управления формой и доступа к компонентам формы.
Caption	Текст заголовка окна.
Top	Расстояние от верхней границы формы до верхней границы экрана.
Left	Расстояние от левой границы формы до левой границы экрана.
Width, Height	Ширина, высота формы.
Icon	Значок в заголовке диалогового окна, обозначающий кнопку вывода системного меню.
Color	Цвет фона.
Font	Шрифт. Шрифт, используемый по «умолчанию» для компонентов, находящимися на поверхности формы.
Canvas	Поверхность, на которую можно вывести графику.

Компонент Button, вкладка панели компонентов Additional



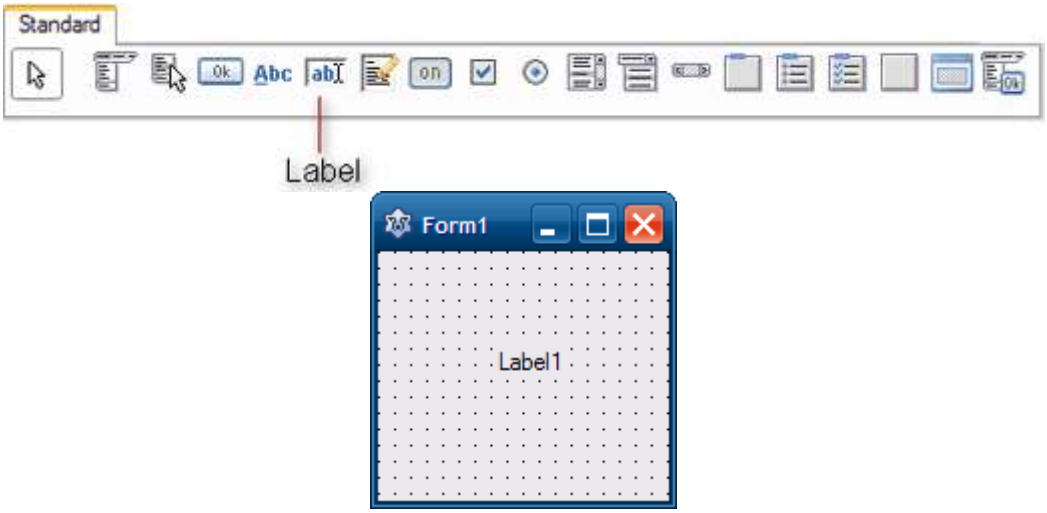
Компонент Button (Кнопка) – командная кнопка, с помощью которой пользователь может вызывать выполнение какого-либо действия.

Основные свойства

Свойство	Описание
Name	Имя компонента. Используется в программе для доступа к компоненту и его свойствам.
Caption	Текст на кнопке.
Left	Расстояние от левой границы кнопки до левой границы формы.
Top	Расстояние от верхней границы кнопки до верхней границы формы.
Width, Height	Ширина, высота кнопки.
Enabled	Признак доступности кнопки. True —кнопка доступна False — кнопка недоступна. Например, в результате щелчка на кнопке событие Click не возникает.
Visible	Позволяет скрыть текст. False – текст видим. True – текст невидим.
Hint	Контекстная подсказка – текст, который появляется рядом с указателем мыши при наведении указателя (для того чтобы текст появился, надо чтобы значение свойства ShowHint было True).
ShowHint	Разрешает (True) или запрещает (False) отображение подсказки при наведении указателя на кнопку.

Компонент Label, вкладка панели компонентов Additional

Компонент Label (Надпись) используется для вывода на форму текста, который пользователь не может изменить во время выполнения программы.

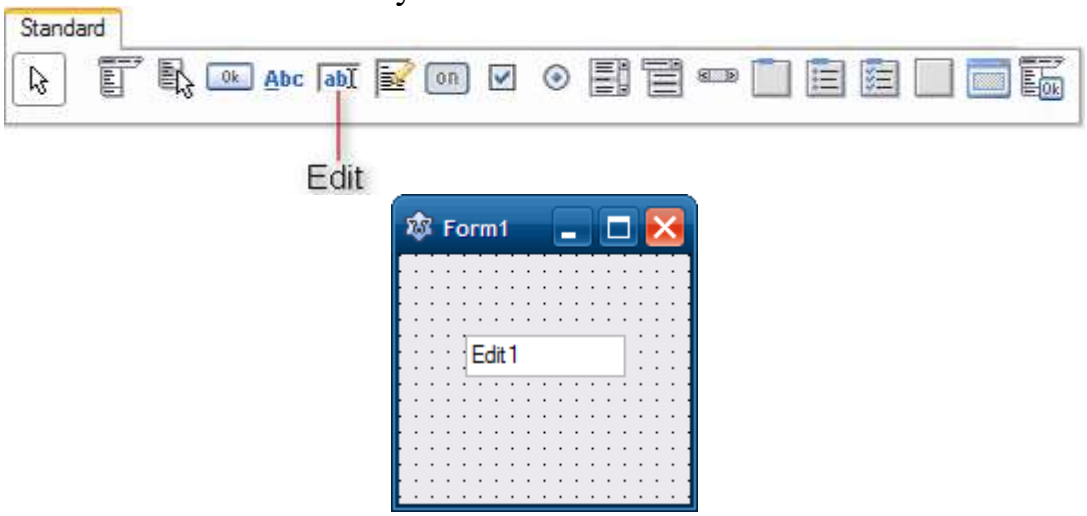


Основные свойства

Свойство	Описание
Name	Имя компонента. Используется в программе для доступа к компоненту и его свойствам.
Caption	Отображаемый в поле надписи текст.
Left	Расстояние от левой границы поля вывода до левой границы формы.
Top	Расстояние от верхней границы поля вывода до верхней границы формы.
Width,Height	Ширина, высота поля вывода.
AutoSize	Признак того, что размер поля определяется его содержимым.
WordWrap	Признак того, что слова, которые не помещаются в текущей строке, автоматически переносятся на следующую строку (значение свойства AutoSize должно быть False).
Alignment	Задает способ выравнивания текста внутри поля:taLeftJustify — выровнивание по левому краю;taCenter – выравнивание по центру;taRightJustify — Выравнивание по правому краю
Font	Параметры шрифта, используемые для отображения текста: Font.Name — вид шрифта;Font.Size – размер шрифта;Font.Color – цвет шрифта.
ParentFont	Признак наследования компонентом характеристик шрифта формы, на которой находиться компонент. Если значение свойства равно True, то текст выводиться шрифтом, установленным для формы.

Color	Цвет фона области вывода текста.
Transparent	Управляет отображением фона области вывода текста. Значение True делает область вывода текста прозрачной, (область не закрашивается цветом, заданным свойством Color).
Visible	Позволяет скрыть текст (False) или сделать его видимым (True).

Компонент Edit,
вкладка панели компонентов System



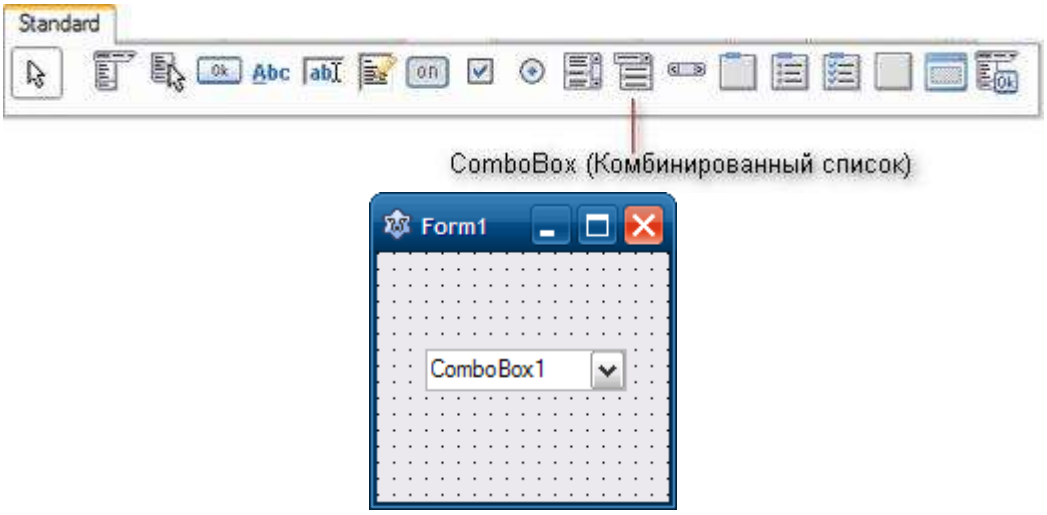
*Компонент (*TEdit*) представляет из себя поле ввода-редактирования строки символов.*

Основные свойства

Свойство	Описание
Name	Имя компонента. Используется в программе для доступа к компоненту и его свойствам, в частности для доступа к тексту, введенному в поле редактирования.
Text	Текст, находящийся в поле ввода и редактирования.
Left	Расстояние от левой границы компонента до левой границы формы.
Top	Расстояние от верхней границы компонента до верхней границы формы.
Width, Height	Ширина, высота поля.
Font	Шрифт, используемый для отображения вводимого текста.
ParentFont	Признак наследования компонентом характеристик шрифта формы, на которой находиться компонент.Если значение свойства

	равно True, то при изменении свойства Font формы автоматически меняется значение свойства Font компонента.
Enabled	Используется для ограничения возможности изменить текст в поле редактирования. Если значение свойства равно False , то текст в поле редактирования изменить нельзя.
Visible	Позволяет скрыть текст (False) или сделать его видимым (True).

Компонент TColorBox,
вкладка панели компонентов Additional



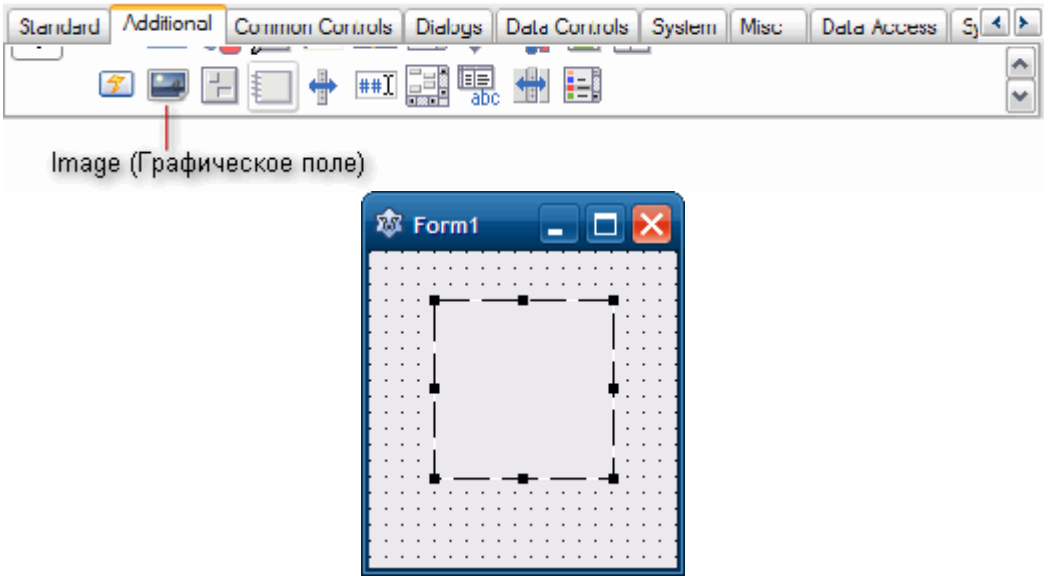
*Компонент (*TComboBox*) дает возможность ввести данные в поле редактирования путем набора на клавиатуре или выбором из списка.*

Основные свойства

Свойство	*Описание*
Name	*Имя компонента. Используется в программе для доступа к компоненту и его свойствам.*
Text	*Текст, находящийся в поле ввода-редактирования*
Items	*Элементы списка – массив строк*
Count	*Количество элементов списка*
Sorted	*Признак необходимости автоматической сортировки (*True*) после добавления очередного элемента.*
ItemIndex	*Номер выбранного элемента. Элементы списка нумеруются с нуля. Если в списке ни один из*

	элементов не выбран, то значение равно минус 1.
DropDownCount	*Количество отображаемых элементов в раскрытом списке. Если количество элементов списка больше чем* DropDownCont , *то появляется вертикальная полоса прокрутки.*
Left	*Расстояние от левой границы компонента до левой границы формы.*
Top	*Расстояние от верхней границы компонента до верхней границы формы.*
Width	*Ширина компонента.*
Height	*Высота компонента (поля ввода-редактирования).*
Font	*Шрифт, используемый для отображения элементов списка.*
ParentFont	*Признак наследования свойств шрифта родительской формы.*

Компонент TImage,
вкладка панели компонентов Additional



Компонент (TImage) обеспечивает вывод на поверхность формы иллюстраций, представленных в bmp–формате (чтобы компонент можно было использовать для отображения иллюстраций в формате JPG, надо подключить модуль JPEG – указать имя модуля в директиве uses).

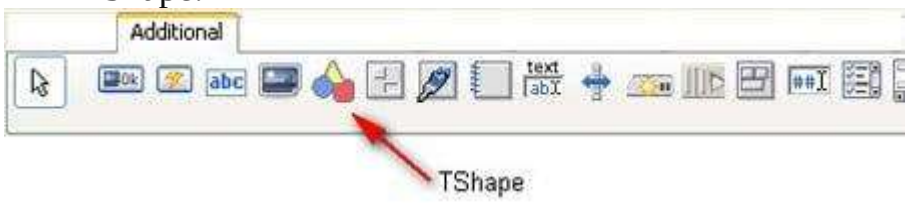
Основные свойства

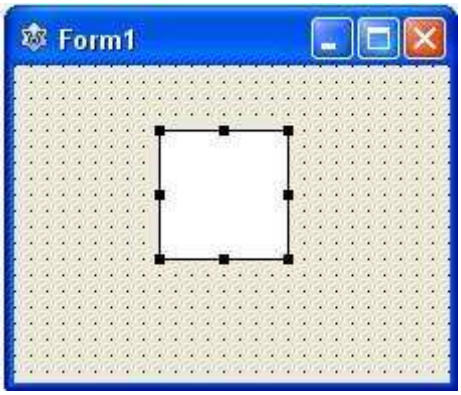
Свойство	***Описание***
Picture	*Иллюстрация, которая отображается в поле компонента.*

Width, Height	*Размер компонента. Если размер компонента меньше размера иллюстрации, и значение свойств AutoSize, Strech и Proportional равно False, то изображается часть иллюстрации.*
Proportional	*Признак автоматического масштабирования картинки без искажения. Чтобы масштабирование было выполнено, значение свойства AutoSize должно быть False.*
Strech	*Признак автоматического масштабирования (сжатия или растяжения) иллюстрации в соответствии с реальным размером компонента. Если размер компонента не пропорционален размеру иллюстрации, то иллюстрация будет искажена.Обратите внимание: свойство Stretch не влияет на файлы рисунков типа .ico.*
AutoSize	*Признак автоматического изменения размера компонента в соответствии с реальным размером иллюстрации.*
Center	*Признак определяет расположение картинки в поле компонента по горизонтали, если ширина картинки меньше ширины поля компонента. Если значение свойства равно False, то картинка прижата к правой границе компонента, если True – то картинка располагается по центру.*
Visible	*Отображается ли компонент и соответственно, иллюстрация на поверхности формы.*
Canvas	*Поверхность, на которую можно вывести графику.*

Компонент TShape,
вкладка панели компонентов Additional

Компонент  Фигура (TShape) предназначен для отображения на форме различных геометрических фигур. Конкретная форма геометрического объекта задается в свойстве Shape.



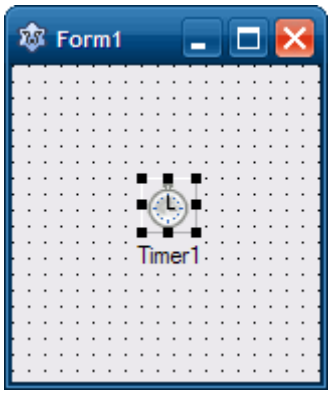


Возможны следующие значения свойства Shape:.

Значение	*Форма фигуры*
stCircle *stEllipse* *stRectangle* *stRoundRect* *stRoundSquare* *stSquare*	*Круг* *Эллипс* *Прямоугольник* *Прямоугольник с округленными краями* *Квадрат с округленными углами* *Квадрат*

Цвет фигуры определяется кистью объекта (свойство Brush), границы фигуры – пером (свойство Pen).

Компонент TTimer,
вкладка панели компонентов System



С помощью таймера (Timer) можно запрограммировать выполнение определенного кода через равные интервалы времени. Когда таймер установлен на форме, система периодически генерирует событие OnTimer. Для пользователя таймер невидим.

Основные свойства

Свойство	*Описание*

Name	*Имя компонента. Используется для доступа к компоненту.*
Interval	*Интервал времени между генерацией событий OnTimer, выраженный в миллисекундах (мс). Отсчет времени начинается с момента установки свойства* Enabled *в* True.
Enabled	*Разрешение работы. При значение* True *таймер включается,* False — выключается.

Чтобы отключить таймер, нужно присвоить свойству Enabled значение False или свойству Interval — значение 0

Настройка таймера Timer1

в окне Инспектор объектов

1. Остановить таймер

Timer1.Enabled = False

2. Задать интервал

Timer1.Interval = 200

Компоненты TOpenDialog и TSaveDialog

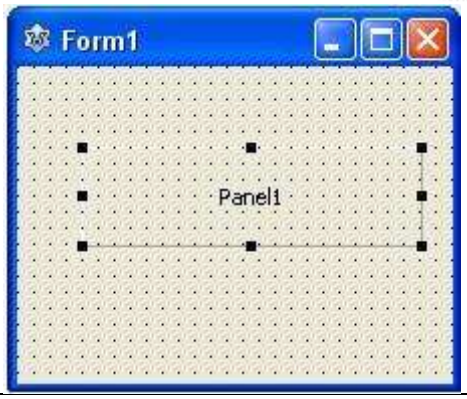
Компонент TOpenDialog предназначен для выбора файла с целью последующего открытия, а компонент TSaveDialog — для последующего сохранения файла.

Свойства приведены в таблице.

Свойства	Описание
DefaultExt	Расширение файла по умолчанию Добавляется в конец имени файла, если расширение не указано явно.
Title	Заголовок диалогового окна
FileName	Выбранное пользователем имя файла вместе с полным путем поиска.
Filter	Список расширений файлов, в соответствии с которыми отбираются имена файлов для отображения в диалоговом окне при открытии файла . При сохранении файла выбранное из списка расширение добавляется к имени файла.
FilterIndex	Порядковый номер строки в списке расширений.

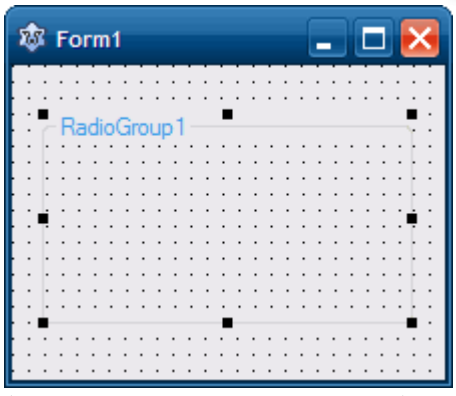
Компонент Panel

Компонент Панель (TPanel) предназначена для объединения произвольных элементов управления с возможностью их перемещения (перетаскивания) по форме вместе с родительской панелью.



Свойство	Название
BovelInner и BovelOuter	Задают стили оформления внутренней и внешней рамок панели. Они могут принимать одно из четырех значений: bvNone Отсутствует blLowered «Вдавленная» «рамка bvRaised «Выпуклая» рамка bvSpace «Плоская» рамка
BovelWidth	Определяет расстояние между внутренней и внешней рамками (в пикселях)
BorderWidth	Определяет ширину рамки вокруг панели в пикселях

Компонент RadioGroup (Группа переключателей) находится на вкладке Standard



Компонент TRadioGroup (Группа переключателей) представляет собой группу переключателей, в которой одновременно может быть выделен только один переключатель. Когда пользователь устанавливает один из переключателей

группы, все остальные автоматически сбрасываются.

Таким образом, какие-либо два переключателя могут быть установлены одновременно, только если они расположены в разных контейнерах, например в разных группах переключателей.

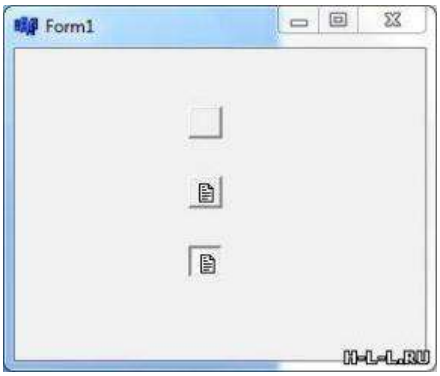
Радиокнопки в контейнере TRadioGroup создаются через свойство Items. В группе переключателей каждая строка свойства Items относиться к отдельному переключателю и служит его заголовком.

Свойство	Описание
Caption	Заголовок группы
Columns	Количество столбцов в группе переключателей. По умолчанию равно 1.
ItemIndex	Определяет номер (начиная с 0) переключателя, который является выделенным в группе. Если первоначально не выделен ни один переключатель, то значение свойства равно -1.
Item	Содержит список заголовков переключателей группы. Для их ввода заголовков предусмотрен специальный редактор, который вызывается щелчком на специальной кнопке  , расположенной справа в строке, описывающей свойство Items.

Свойство Name содержит префикс grp, сообщающий о том, что это группа переключателей.

Компонент SpeedButton

SpeedButton - кнопка, используемая для создания панелей инструментов или там, где нужна фиксация в зажатом состоянии.

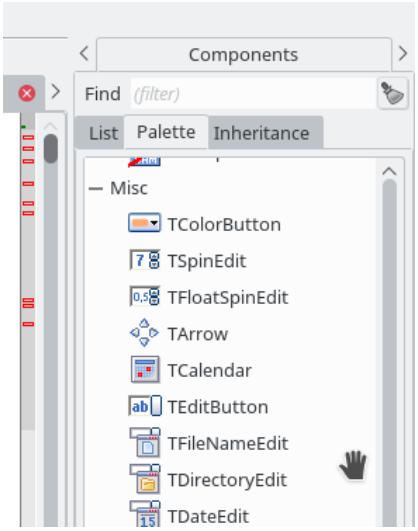


Свойство	Значение
Caption	Определяет название кнопки.

Down	Фиксация кнопки в зажатом положении, GroupIndex не должен быть равен 0.
Flat	Определяет стиль кнопки. Если установлено true, то очертания кнопки появляются при наведении на неё курсора мыши.
Glyph	С помощью него можно подгрузить рисунок кнопки в формате bmp. Очень много картинок для кнопок находятся в %\Program Files\Common Files\Borland Shared\Images\Buttons
GroupIndex	По умолчанию равен 0. Нужно изменить на отличное от нуля число, чтобы кнопку можно было зафиксировать нажатой.
Layout	Расположение изображения и названия кнопки относительно друг друга, blGlyphLeft - изображение слева, blGlyphBottom - снизу, blGlyphRight - справа, blGlyphTop - сверху.

Компонент SpinEdit

Элемент управления **TSpinEdit** находится на вкладке Misc палитры компонентов. Он полезен для установки числовых значений. Фактически, элемент **TSpinEdit** является комбинацией элемента TUpDown и связанного с ним элемента TEdit, предназначенного для ввода чисел.



Наиболее актуальными свойствами элемента **TSpinEdit** являются:

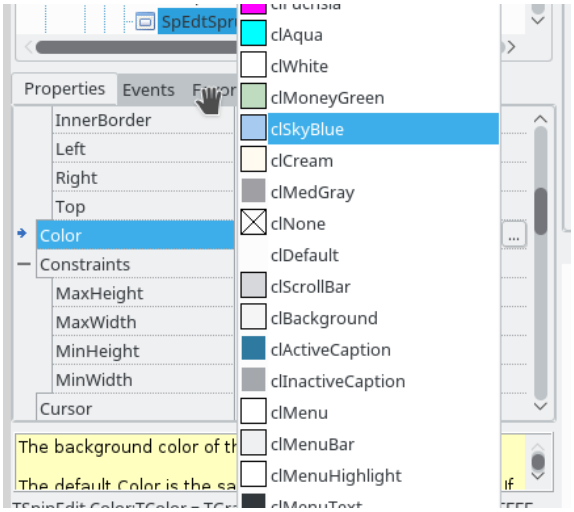
Value: текущее значение

MinValue: минимально допустимое значение

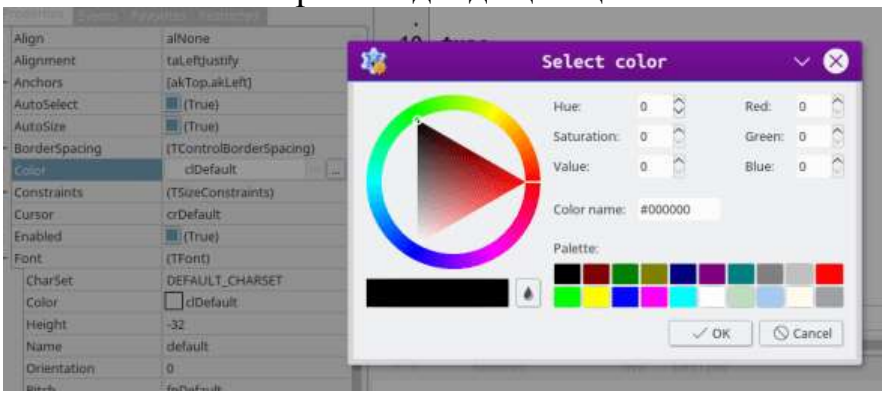
MaxValue: максимально допустимое значение

Increment: значение, на которое увеличивается/уменьшается значение при щелчках по кнопкам

Цвет фона элемента, цвет шрифта, стиль и другие подобные свойства доступны пользователю. Для изменения цвета фона элемента перейдите в Инспектор объектов и выберите из выпадающего списка любой из доступных цветов.



В списке представлен полный перечень цветов, но пользователь также может выбрать цвет, щелкнув по кнопке с тремя точками справа [...] от свойства за выпадающим списком. При нажатии на кнопку открывается диалоговое окно, в котором пользователь может выбрать подходящий цвет.

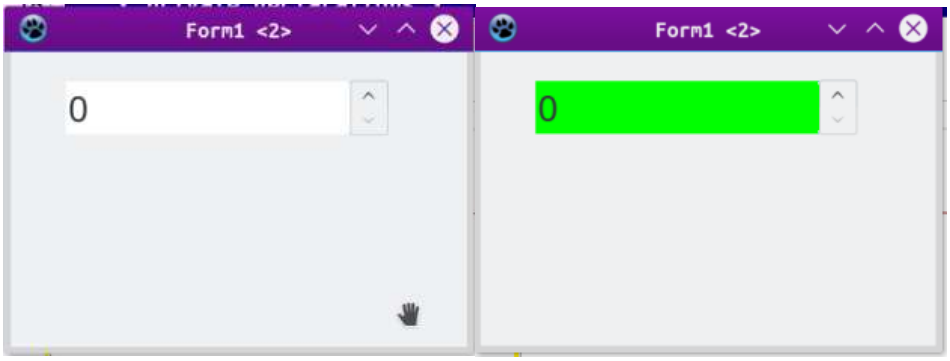


Аналогичным образом могут быть выбраны цвет шрифта, стиль и другие свойства.

Также возможно изменение этих свойств программным способом. В этом случае создается форма и на ней размещается элемент управления **TSpinEdit**. На вкладке *Events* Инспектора объектов выбирается событие *OnClick* с помощью нажатия на кнопку [...] с правой стороны. В результате этого создается код, в котором можно изменить цвет фона:

```
procedure TForm1.SpinEdit1Click(Sender : TObject) ;
begin
  SpinEdit1.Color := clLime;
end;
```

После этого нажмите клавишу F9 или выберите команду *Запустить* из меню *Запуск*. После компиляции появится форма. При щелчке внутри элемента **TSpinEdit** изменится его цвет:



Большинство других свойств могут быть изменены аналогичным образом.

Список литературы

1. Free Pascal и Lazarus: Учебник по программированию / Е. Р. Алексеев, О. В. Чеснокова, Т. В. Кучер — М. : ALT Linux ; Издательский дом ДМК-пресс, 2010. — 440 с. ил.
2. Акчурин Э.А. Программирование на языке FreePascal Часть 4. ЛР в ИСР Lazarus. Учебное пособие для студентов направления «Информатика и вычислительная техника» Самара 2008 г., стр. 150
3. Беляков А.Ю. Прикладное программирование в Лазарус. Учебное пособие, Пермь ФГБОУ ВО Пермский ГАТУ, 2019. – 114 с.
4. Крайнова Т.С. Разработка программных приложений. Методические указания по выполнению лабораторно-практических работ для студентов, обучающихся по направлениям 09.03.03 «Прикладная информатика», 38.03.05 «Бизнес-информатика» всех форм обучения, Екатеринбург 2014 г., стр.45
5. Мансуров К.Т. Основы программирования в среде Lazarus, 2010. – 772 с. ил.
6. Медведева О. А. Методические указания к лабораторным работам по дисциплине «Информатика» Работа с основными компонентами визуальной среды Lazarus для студентов технических специальностей всех форм обучения , 2013 г. Краматорск, ДГМА, 2013, стр.32
7. Программирование для начинающих/ Л..С. Цветкова. – Москва: Издательство БИНОМ. Лаборатория знаний. 2010. – 287 с.: ил.

Интернет-ресурсы

1. Lazarus для школьников и студентов - https://www.sites.google.com/site/studylazarus/
2. Программирование в Лазарус. Лабораторный практикум - https://www.sites.google.com/site/ifizmat/prog/lazarus
3. Программирование на Лазарус. - https://kryar.yartel.ru/files/inform/11k_ugl/lazarus.pdf