МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение
высшего профессионального образования
«Казанский (Приволжский) федеральный университет»

ИНСТИТУТ МАТЕМАТИКИ И МЕХАНИКИ им. Н.И. ЛОБАЧЕВСКОГО

КАФЕДРА ТЕОРИИ И ТЕХНОЛОГИЙ ПРЕПОДАВАНИЯ МАТЕМАТИКИ И ИНФОРМАТИКИ

Направление: 050202.65 информатика с дополнительной специальностью

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

Создание дистанционного курса «Визуальное программирование в Delphi» для средней школы

Работа завершена:

«___» _____ 2015 г. _____ (Л.Р. Хузеева)

Работа допущена к защите:

Научный руководитель

к.т.н., доцент

«___» _____ 2015 г. _____ (Т.Ю. Гайнутдинова)

Заведующий кафедрой

д.п.н., профессор

«___» _____ 2015 г. _____ (Л.Р. Шакирова)

Казань - 2015

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Основная причина бурного развития сред визуального проектирования — необходимость освободить разработчика программ от рутинной работы по созданию стандартизированных элементов интерфейса разрабатываемого проекта. Число таких элементов в настоящее время довольно велико и продолжает расти, причем внутренняя структура многих из этих элементов, весьма сложная.

Базовые элементы интерфейса (кнопки, меню, строки ввода и тому подобное) уже вряд ли будут принципиально совершенствоваться. Их состав, свойства, принципы использования являются практически промышленным стандартом и одинаковы в любой среде разработки современных программ.

Актуальность. В основе школьного курса информатики изучают язык программирования - Delphi. Для использования его графических возможностей необходимо использование технологии визуализации, которые позволят существенно сократить время разработки и облегчить процесс создания приложений — программ, для среды Windows.

Целью данной выпускной квалификационной работы является – разработка учебного курса по информатике «Визуальное программирование в Delphi» для школьников.

Для достижения поставленной цели необходимо решить следующие **задачи*:***

- проанализировать программу школьного курса информатики по разделу «Программирование»;

- систематизировать учебно-методическую и научно-педагогическую литературу по теме исследования;

- разработать учебный курс и написать учебное пособие по теме исследования.

Методы исследования:

1) изучение и анализ научно-педагогической и методической литературы, программы, учебников, учебных и методических пособий по теме;

2) посещение уроков и наблюдение за работой учащихся;

3) опытное преподавание.

Практическая значимость работы состоит в том, что методические рекомендации могут быть использованы учителями школ при проведении практических работ по предмету «Информатика и ИКТ»

Работа состоит из введения, двух глав, заключения и списка литературы. В первой главе представлены основы визуального программирования в Delphi. Во второй – разработаны практические задания для трех уровней: вводной, основной и повышенной сложности. Объем работы 102 страницы.

ГЛАВА I

ОСНОВЫ ВИЗУАЛЬНОГО ПРОГРАММИРОВАНИЯ В DELPHI

§1.1. Пользовательский интерфейс среды Delphi

Для запуска среды Delphi выполните следующую команду:

Пуск → Все программы → Borland Delphi 7 → Delphi 7

После загрузки интерфейс Delphi 7 имеет вид, показанный на рис. 1, и имеет 5 основных окон:

- главное окно (Delphi 7 - Project 1);
- окно редактора свойств объектов (Object Inspector);
- окно формы (Form 1);
- окно редактора кода (Unit1.pas);
- окно дерева объектов (Object TreeView).

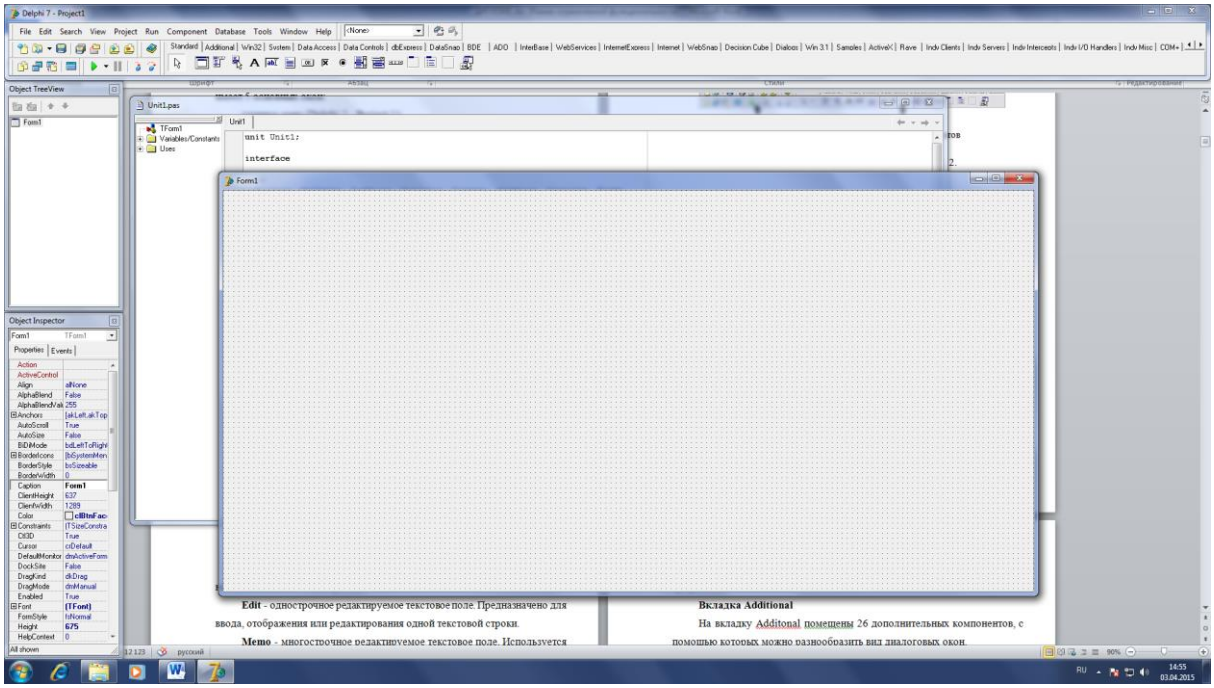


Рис.1 Интерфейс Delphi 7

Окно редактора кода почти полностью закрыто окном стартовой формы. Для переключения между окном формы и окном редактора кода используется клавиша F12.

Главное окно находится в верхней части экрана. В нём расположены:

- строка заголовка;
- строка меню;
- панель инструментов;
- панель палитры компонентов.

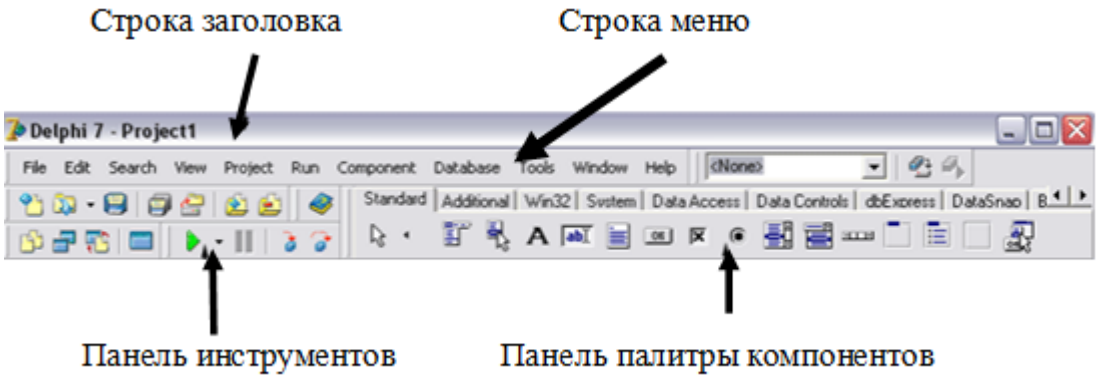


Рис. 2 Главное окно

Программирование в Delphi производится при помощи помещения объектов, находящихся на палитре компонентов, на форму. Опишем наиболее часто используемые компоненты и вкладки, на которых они расположены [5].

Вкладка Standard

На данной вкладке палитры компонентов сосредоточены стандартные для Windows интерфейсные элементы, без которых не обходится практически ни одна программа.

Frame – фрейм. Наравне с формой служит контейнером для размещения других компонентов. В отличие от формы, может размещаться в палитре компонентов, создавая заготовки компонентов.

MainMenu — главное меню программы. Компонент способен создавать и обслуживать сложные иерархические меню.

PopupMenu - контекстное, или локальное, меню. Обычно это меню появляется после щелчка правой кнопки мыши,

Label — метка. Этот компонент используется для размещения в окне не очень длинных однострочных надписей.

Edit - однострочное редактируемое текстовое поле. Предназначено для ввода, отображения или редактирования одной текстовой строки.

Memo - многострочное редактируемое текстовое поле. Используется для ввода и/или отображения многострочного текста.

Button - кнопка. Обработчик события OnClick этого компонента обычно используется для реализации некоторой команды.

CheckBox - флажок. Щелчок мышью на этом компоненте в работающей программе изменяет его логическое свойство Checked.

RadioButton - переключатель. Обычно объединяется как минимум еще с одним таким же компонентом в группу. Щелчок на переключателе приводит к автоматическому освобождению ранее выбранного переключателя в той же группе.

ListBox — список. Содержит список предлагаемых вариантов (пунктов списка) и дает возможность проконтролировать текущий выбор.

ComboBox — комбинированный список. Представляет собой комбинацию списка и однострочного текстового поля.

ScrollBar - полоса прокрутки. Представляет собой вертикальную или горизонтальную полосу, напоминающую полосы прокрутки по бокам окна программы.

GroupBox — панель группировки. Этот компонент используется для группировки нескольких связанных по смыслу компонентов.

RadioGroup — группа переключателей. Содержит специальные свойства для обслуживания нескольких связанных переключателей.

Panel — панель. Этот компонент, как и GroupBox, служит для объединения нескольких компонентов. Содержит внутреннюю и внешнюю кромки, что позволяет создать эффекты «вдавленности» и «выпуклости».

ActionList — список действий. Служит для централизованной реакции программы на действия пользователя, связанные с выбором одного из

группы однотипных управляющих элементов, таких как команды меню, графические кнопки и т. п.

Вкладка Additional

На вкладку Additonal помещены 26 дополнительных компонентов, с помощью которых можно разнообразить вид диалоговых окон.

BitBtn — кнопка с изображением.

Speed Button — кнопка панели инструментов. Обычно используется для быстрого доступа к тем или иным командам главного меню.

MaskEdit — поле с маской ввода. Этот компонент способен фильтровать вводимый текст, например, для правильного ввода даты.

StringGrid — текстовая таблица. Этот компонент обладает мощными возможностями для представления текстовой информации в табличном виде.

DrawGrid — произвольная таблица. В отличие от StringGrid, ячейки этого компонента могут содержать произвольную информацию, в том числе и рисунки.

Image — изображение. Этот компонент предназначен для отображения рисунков, в том числе значков и метафайлов.

Shape — стандартная фигура. С помощью этого компонента вы можете вставить в окно правильную геометрическую фигуру — прямоугольник, эллипс, окружность и т. п.

Bevel — кромка. Служит для выделения отдельных частей окна трехмерными рамками или полосами.

ScrollBox — панель с полосами прокрутки. В отличие от компонента Panel, автоматически вставляет полосы прокрутки, если размещенные в нем компоненты отсекаются его границами.

CheckListBox — список флажков. Отличается от стандартного компонента ListBox наличием рядом с каждым пунктом списка флажка CheckBox, что дает возможность выбора сразу нескольких пунктов,

Splitter — вешка разбивки. Этот компонент размещается на форме между двумя другими видимыми компонентами и дает возможность

пользователю во время прогона программы перемещать границу, отделяющую компоненты друг от друга.

StaticText - текстовая метка, Отличается от стандартного компонента Label наличием собственного оконного ресурса, что позволяет обводить текст рамкой или выделять его в виде «вдавленной» части формы.

ControlBar - контейнер для панелей инструментов. Служит контейнером для «причаливаемых» (Drag&Dock) компонентов.

Application Events - обработчик сообщений Windows, Если этот компонент помещен на форму, он будет получать все предназначенные для программы сообщения Windows (без этого компонента сообщения принимает глобальный объект-программа Application).

ValueListEditor - специализированный редактор списков, содержащих пары имя = значение. Пары такого типа широко используются в Windows, например, в файлах инициализации, в системном реестре и т. п.

Labeled Edit - комбинация однострочного поля и метки.

ColorBox - список выбора цвета. Специальный вариант компонента ComboBox для выбора одного из системных цветов.

Chart - диаграмма. Этот компонент облегчает создание специальных панелей для графического представления данных.

ActionManager — менеджер действий. Совместно с тремя следующими компонентам обеспечивает создание приложений, интерфейс которых (главное меню и инструментальные кнопки) может настраиваться пользователем.

ActionMainMenuBar — строка меню для действий. Команды меню создаются с помощью компонента ActionManager.

ActionToolBar - панель инструментов для действий. Служит контейнером для кнопок, создаваемых с помощью компонента ActionManager.

CustomizeDlg - диалоговое окно настройки. С помощью этого компонента пользователь может по своему вкусу настроить интерфейс работающей программы.

XPColorMap - совместно с тремя следующими компонентами впервые введен в версии 7 для настройки цветов и наполнения панелей ActionToolBar [5].

Вкладка System

На вкладке System представлены компоненты, которые имеют различное функциональное назначение, в том числе компоненты, поддерживающие стандартные для Windows технологии межпрограммного обмена данными OLE (Object Linking and Embedding — связывание и внедрение объектов) и DDE (Dynamic Data Exchange — динамический обмен данными).

Timer — таймер. Этот компонент служит для отсчета интервалов реального времени.

PaintBox — окно для рисования. Создает прямоугольную область, предназначенную для прорисовки графических изображений.

Media Player — медиаплейер. С помощью этого компонента можно управлять различными мультимедийными устройствами.

OteContainer — OLE-контейнер. Служит приемником связываемых или внедряемых объектов.

Вкладка Dialogs

Компоненты вкладки Dialogs реализуют стандартные для Windows диалоговые окна.

Open Dialog — окно открытия файла. Реализует стандартное диалоговое окно открытия файла.

SaveDialog — окно сохранения файла. Реализует стандартное диалоговое окно сохранения файла.

OpenPictureDialog - окно открытия изображения. Реализует специальное окно выбора графических файлов с возможностью предварительного просмотра изображений.

SavePictureDialog — окно сохранения изображений. Реализует специальное окно сохранения графических файлов с возможностью предварительного просмотра рисунков.

FontDialog - окно выбора шрифта. Реализует стандартное диалоговое окно выбора шрифта.

ColorDialog — окно выбора цвета. Реализует стандартное диалоговое окно выбора цвета.

PrintDialog — окно настройки параметров печати. Реализует стандартное диалоговое окно выбора параметров для печати документа.

PrinterSetupDialog - окно настройки параметров принтера. Реализует стандартное диалоговое окно для настройки печатающего устройства.

FindDialog — окно поиска. Реализует стандартное диалоговое окно поиска текстового фрагмента.

ReplaceDialog — окно поиска и замены. Реализует стандартное диалоговое окно поиска и замены текстового фрагмента.

Object Inspector – окно редактора свойств объектов или инспектор объектов (рис. 3).

Любой размещаемый на форме компонент характеризуется некоторым набором параметров: положением, размером, цветом и т. д. Часть этих параметров, например, положение и размеры компонента, можно изменять, манипулируя с компонентом в окне формы. Для изменения других параметров предназначено окно редактора свойств объектов (или, что, то же самое, Инспектора объектов). Это окно содержит две вкладки – **Properties** (Свойства) и **Events** (События). Вкладка **Properties** служит для установки нужных свойств компонента, вкладка **Events** позволяет определить реакцию компонента на то или иное событие. Совокупность свойств отображает видимую сторону компонента:

- положение относительно левого верхнего угла рабочей области формы;

- его размеры и цвет;

- шрифт и текст надписи на нем;

- совокупность событий – его поведенческую сторону: будет ли компонент реагировать на щелчок мыши или на нажатие клавиш, как он будет вести себя в момент появления на экране или в момент изменения размеров окна и т. п.

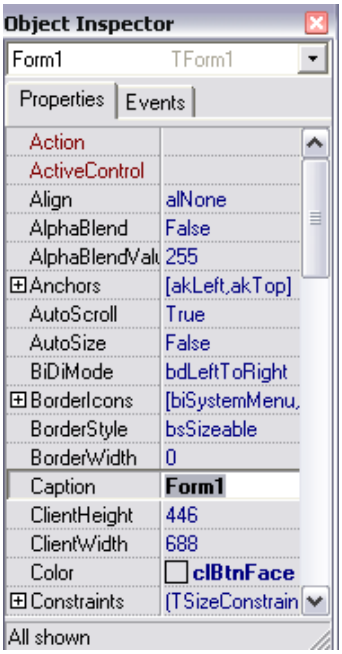


Рис. 3 Окно редактора свойств объектов

Таблица 1

Свойства формы

Свойство	Описание
Name	Имя формы. В программе имя формы используется для управления формой и доступа к компонентам формы.
Caption	Текст заголовка
Left	Расстояние левой границы формы до левой границы экрана
Top	Расстояние от верхней границы формы до верхней границы

	экрана
Height	Высота формы
Width	Ширина формы
BorderStyle	Вид границы. Граница может быть обычной (bsSizeable), тонкой (bsSingle) или отсутствовать (bsNone). Если у окна обычная граница, то во время работы программы пользователь может при помощи мыши изменить размер окна. Изменить размер окна с тонкой границей нельзя. Если граница отсутствует, то на экран во время работы программы будет выведено окно без заголовка. Положение и размер такого окна во время работы программы изменить нельзя.
BorderIcons	Кнопки управления окном. Значение свойства определяет, какие кнопки управления окном будут доступны пользователю во время работы программы. Значение свойства задается путем присвоения значений уточняющим свойствам biSystemMenu, biMinimaze, biMaximaze, biHelp. Свойство biSystemMenu определяет доступность кнопки *свернуть,* и кнопки системного меню,biMaximaze –кнопки *свернуть,* biMaximaze - *развернуть,* biHelp – *кнопки вывода справочной информации.*
Icon	Значок в заголовке диалогового окна, обозначающий кнопку вывода системного меню.
Color	Цвет фона. Цвет можно задать, указав название цвета или привязку к текущей цветовой схеме операционной системы. Во втором случае цвет определяется текущей цветовой схемой, выбранным компонентом привязки и меняется при изменении цветовой схемы операционной системы.
Font	Шрифт. Шрифт, используемый «по умолчанию»

	компонентами, находящимися на поверхности формы. Изменение свойства Font формы приводит к автоматическому изменение свойства Font компонента, располагающегося на поверхности формы. То есть компоненты наследуют свойство Font от формы (имеется возможность запретить наследование).

Object TreeView – окно дерева объектов (рис. 4) предназначено для наглядного отображения связей между отдельными компонентами, размещенными на активной форме или в активном модуле данных. Щелчок по любому компоненту в этом окне активизирует соответствующий компонент в окне формы и отображает свойства этого компонента в окне Инспектора объектов [5].

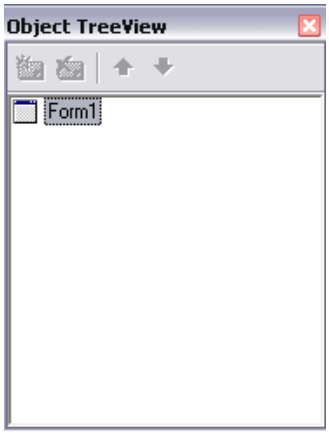


Рис.4 Окно дерева объектов

Окно редактора кода, можно увидеть, отодвинув в сторону окно формы либо, как уже говорилось выше, нажать клавишу F12. В него следует набирать текст программы. В начале работы над новым проектом это окно редактора кода содержит сформированный Delphi шаблон программы.

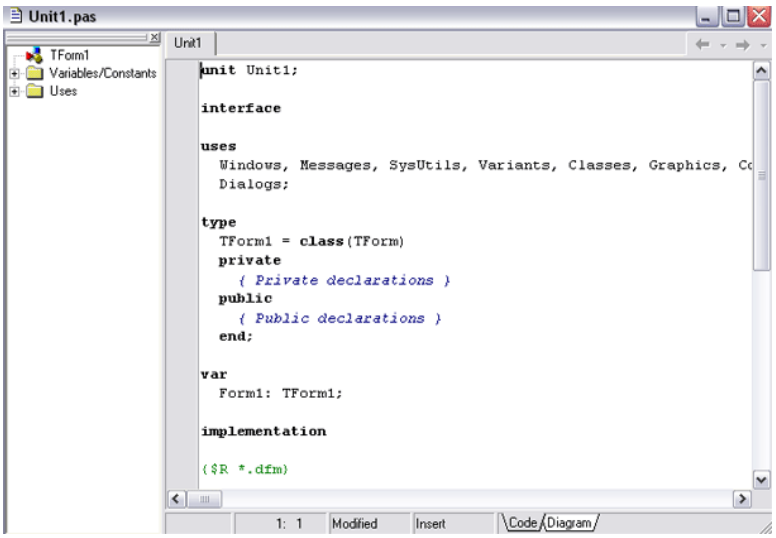


Рис. 5 Окно кода программы

Окно формы представляет собой проект Windows-окна будущей программы. Вначале это окно содержит стандартные для Windows интерфейсные элементы - кнопки вызова системного меню, максимизации, минимизации и закрытия окна, строку заголовка и очерчивающую рамку. Вся рабочая область окна заполнена точками координатной сетки, служащей для упорядочения размещаемых на форме компонентов.

Свойства формы определяют ее внешний вид: размер, положение на экране, текст заголовка, вид рамки.

В ходе работы над новым проектом, так в Delphi называется разрабатываемое приложение, программист изменяет значения свойств формы Form1 и добавляет к форме необходимые компоненты (поля ввода и вывода текста, командные кнопки), при этом он в любой момент времени контролирует содержание окна создаваемой программы и может внести в него необходимые изменения.[6]

§1.2. Компоненты и палитра компонентов

Палитра компонентов – это каталог, состоящий из визуальных и невизуальных компонентов. Компонент – это структурная единица Delphi. Основу ООП в Delphi составляет набор компонентов, который позволяет Delphi с помощью компонентов использовать множество возможностей, присущих Windows.

Окно формы – это окно Windows. Всё рабочее пространство окна – это рабочая область, размеченная сеткой для удобства расположения на ней компонентов из Палитры компонентов. Сама форма также является компонентом. Новая форма, которая создаётся при загрузке Delphi или при создании нового проекта, является главной формой приложения.

Палитра компонентов расположена в правой части главного окна и имеет вид многостраничного блокнота, где на каждой странице размещён набор пиктограмм её компонентов. Активизировать группу компонентов требуемой страницы надо щелчком мыши на её закладке.

Для добавления какого-либо компонента в Окно формы надо выбрать его пиктограмму на панели компонентов, щёлкнуть на нём левой кнопкой мыши, а затем возможны два варианта действий:

- если надо точно позиционировать компонент, надо подвести курсор мыши в нужное место Окна формы и щёлкнуть левой кнопкой мыши один раз; левый верхний угол компонента совпадёт при этом с положением конца стрелки курсора мыши;

- если надо поместить компонент в центр Окна формы, используется двойной щелчок на пиктограмме.

§1.3. Структура кода модуля

Нажмите клавишу F12 для просмотра кода программы. Можно заметить, что в окне кода программы на вкладке **Unit1** уже существует некоторый код, сформированный Delphi. Этот код выглядит следующим образом:

```
unit Unit1;
interface  // раздел интерфейса
{ Здесь находятся описания процедур и функций модуля, которые могут использоваться другими модулями}
{ Список подключаемых модулей}
```

uses

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

Type {раздел объявления типов}

TForm1 = class(TForm)

Private {Закрытый раздел класса}

{Private declarations}

{Сюда помещаются объявления переменных, функций и процедур, включаемых в класс формы, но не доступных для других модулей}

Public {Открытый раздел класса}

{Public declarations}

{Сюда могут помещаться объявления переменных, функций и процедур, включаемых в класс формы и доступных для других модулей}

end;

var {раздел объявления переменных}

{Здесь находятся объявления глобальных переменных модуля, которые могут использоваться процедурами и функциями модуля}

Form1: TForm1;

Implementation {раздел реализации}

{Описания (текст) процедур и функций модуля. Могут помещаться предложения ***uses****, объявления типов, констант, переменных, к которым не будет доступа из других модулей. Тут же должны быть реализации всех объявленных в разделе* ***interface*** *функций и процедур, реализации любых дополнительных, не объявленных ранее функций и процедур }*

{$R *.dfm}

end.

Модуль состоит из последовательности разделов. Каждый раздел начинается ключевым словом и продолжается до начала следующего раздела.

Модуль начинается с ключевого слова **Unit**, после которого пишется имя модуля. Оно совпадает с именем файла, в котором вы сохранили свой модуль. По умолчанию для первого модуля имя равно **Unit1**, для второго **Unit2** — и т.д.

Текст модуля состоит из двух основных разделов: **Interface** - открытый интерфейс модуля, и **Implementation** - реализация модуля. Все, что помещается непосредственно в раздел **Interface** (типы, переменные, константы, функции, процедуры), может быть использовано другими модулями программы. Все, что помещается в раздел **Implementation** — внутреннее дело модуля. Внешние модули не могут видеть типы, переменные, константы, функции и процедуры, размещенные в разделе реализации.

В разделе **Interface** после предложения **Uses**, содержащего список подключаемых библиотечных модулей, можно видеть объявление класса вашей формы, подготовленное Delphi. Имя класса вашей формы — TForm1. В класс включены те объекты, которые размещены на форме.

В классе предусмотрено также два раздела: **Private -** закрытый раздел класса, и **Public** - открытый раздел класса. То, что вы или Delphi объявите в разделе **Public,** будет доступно для других классов и модулей. То, что объявлено в разделе **Private,** доступно только в пределах данного модуля.

§1.4. Проект приложения. Файлы Delphi для приложения

Проект Delphi состоит из форм, модулей, установок параметров проекта, его ресурсов (битовые файлы, пиктограммы) и т.д. На этапе проектирования Delphi создаёт совокупность файлов, из которых состоит приложение. Результатом проектирования приложения, разработанного в Delphi, является EXE-файл для выполнения в среде Windows. Он формируется на базе проекта приложения. Один проект соответствует одному приложению.

Проект должен иметь определённую структуру каталогов для хранения файлов проекта. Имена файлов, форм, компонентов и переменных должны отражать их содержание. *Проект* – это набор взаимосвязанных форм и модулей, образующих приложение. Модули проекта используют другие модули, определённые их оператором **Uses**.

В состав проекта входят следующие файлы:

- файл проекта (расширение .DPR – Delphi PRoject); в проекте приложения он единственный;

- описание всех форм, входящих в проект; каждая форма проекта имеет файл;

- формы (расширение DFM – от Delphi ForM);

- модули форм (расширение PAS); файл создаётся автоматически для каждой - формы проекта;

- файлы с параметрами проекта (расширение DOF, от Delphi Option File);

- файлы с описаниями ресурсов (расширение RES, от Delphi Component RESourse).

§1.5. Реализация ввода и вывода

Наиболее распространённые компоненты для ввода и вывода текстовой информации **Label** и **Edit** расположены на странице **Standard.**

Компонент Label (Метка)

Текст, отображаемый в компоненте **Label,** определяется свойством **Caption**. Это свойство можно задать на вкладке **Properties** окна **Object Inspector** во время проектирования формы либо изменять его программно во время выполнения приложения. Например:

Label1.Caption:= ‘Текстовая строка’;

Свойства, которые можно изменить для компонента Label:

- Color (Цвет фона)

- Font (Шрифт)

При изменении свойства **Font** через окно **Object Inspector** появляется диалоговое окно, в котором можно изменить вид шрифта, цвет надписи и размер шрифта.

Таблица 2

Свойства компонента Label

Свойство	Описание
Name	Имя компонента. Используется в программе для доступа к компоненту и его свойствам
Caption	Отображаемый текст
Font	Шрифт, используемый для отображения текста
ParentFont	Признак наследования компонентом характеристик шрифта формы, на которой находится компонент. Если значение свойства равно True, текст выводится шрифтом, установленном для формы
AutoSize	Признак того, что размер поля определяется его содержимым
Left	Расстояние левой границы поля вывода до левой границы формы
Top	Расстояние от верхней границы поля вывода до верхней границы формы
Height	Высота поля вывода
Width	Ширина поля вывода
WordWrap	Признак того, что слова, которые не помещаются в текущей строке, автоматически переносятся на следующую строку.

Следует обратить внимание на свойство **AutoSize** и **WordWrap**. Эти свойства нужно использовать, если поле вывода должно содержать

несколько строк текста. После добавления к форме компонента **Label** значение свойства **AutoSize** равно *True,* т.е. размер поля определяется автоматически в процессе изменения значения свойства ***Caption*.** Если вы хотите, чтобы находящийся в поле вывода текст занимал несколько строк, то надо сразу после добавления к форме компонента *Label* присвоить свойству **AutoSize** значение *False,* свойству **WordWrap**- значение *True*. Затем изменением значений свойств *Width* и *Height* нужно задать требуемый размер поля. Только после этого можно вывести в свойство **Caption** текст, который должен быть выведен в поле.

Компонент Edit (Поле редактирования)

Данный компонент может использоваться как для ввода, так и для отображения текстовой строки. Вводимый и выводимый текст задаётся свойством **Text.**

Свойства надписи, отображаемой в компоненте **Edit**, аналогичны свойствам компонента **Label.**

Кроме компонентов **Label** и **Edit** существует большое количество различных компонентов, позволяющих осуществлять ввод и вывод текстовой информации. Эти компоненты перечислены в таблице "Компоненты ввода и отображения текстовой информации".

Группа подпрограмм реализует простые типовые (стандартные) оконные диалоги, не входящие в палитру компонентов: выдача сообщения пользователю, получение от него одного из ряда возможных ответов, ввод строки. Эти средства удобны для отображения на экран сообщения или получения от пользователя текстовой информации.

Таблица 3

Компоненты ввода и отображения текстовой информации

Процедура	Назначение
ShowMessage	Вывод сообщения в диалоговом окне
ShowMessagePos	

MessageDlg	Вывод сообщения в диалоговом окне с получением ответа от пользователя
MessageDlgPos	
InputBox	Ввод текстовой строки в поле ввода диалогового окна
InputQuery	

Например, выдать пользователю сообщение можно с помощью процедур:

ShowMessage (promt);

ShowMessagePos (promt, [, xpos] [, ypos]);

Обе процедуры выдают на экран окно с сообщением **prompt** и кнопкой **ОК**. В первом случае окно помещается в центре экрана, во втором – верхний левый угол окна помещается в точку с координатами, заданными выражениями **xpos и ypos.**

Для того чтобы после выдачи сообщения получить от пользователя ответ, можно использовать одну из следующих функций:

MessageDlg (promt, type, buttons, helpCtx);

MessageDlgPos (promt, type, buttons, helpCtx, [, xpos] [, ypos]);

Функции отличаются явным заданием координат окна диалога.

type – тип сообщения. Сообщение может быть информационным, предупреждающим или сообщением о критической ошибке. Каждому типу сообщения соответствует определенный значок. Тип сообщения задается именованной константой.

buttons – список кнопок, отображаемых в окне сообщения. Список может состоять из нескольких разделенных запятыми именованных констант. Весь список заключается в квадратные скобки.

helpCtx – контекст справки, параметр, определяющий раздел справочной системы, который появится на экране, если пользователь нажмет клавишу <F1>. Если вывод справки не предусмотрен, то значение параметра helpCtx должно быть равно нулю.

Имя константы	Тип диалога	Значок
mtWarning	Внимание	
mtError	Ошибка	
mtInfirmation	Информация	
mtConfirmation	Подтверждение	
mtCustom	Обычное	Без значка

Рис.6. Константы, определяющие отображаемые значки

Таблица 4

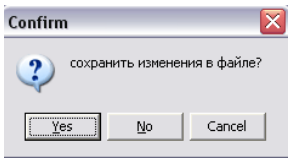
Константы, определяющие отображаемые кнопки

Имя кнопки	Кнопка	Имя результата	Значение результата
mbOK	OK	mrOK	1
mbCancel	Cancel	mrCancel	2
mbAbort	Abort	mrAbort	3
mbRetry	Retry	mrRetry	4
mbIgnore	Ignore	mrIgnore	5
mbYes	Yes	mrYes	6
mbNo	No	mrNo	7
mbHelp	Help		
mbHelp	All		

Примеры вызова функции MessageDlg:

MessageDlg('Ошибка', mtError, [mbYes], 0);

MessageDlg('сохранить изменения в файле?', mtConfirmation, [mbYes, mbNo, mbCancel], 0);



Для ввода текста в окно в процессе диалога служит функция:

InputBox (caption, promt, default);

caption – текст заголовка окна.

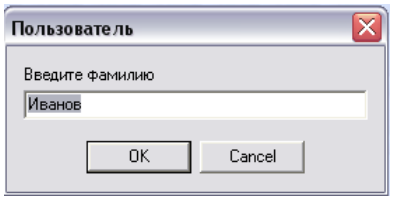
promt – подсказка, приглашающая набрать текст в поле ввода.

default – строковое выражение, изначально отображаемое в поле ввода. Если этот аргумент опущен, поле ввода изображается пустым.

Функция возвращает строку из поля ввода на момент закрытия диалога. При нажатии кнопки **Cancel** функция возвращает текст из **default.**

Пример вызова функции InputBox:

x := InputBox('Пользователь','Введите фамилию','Иванов');



Переменной x будет присвоено строковое значение, введённое в поле редактирования, если пользователь завершит диалог, не введя никакого значения, то переменной будет присвоено значение **default.**

Аналогичные действия выполняет функция:

InputQuery (caption, promt, value);

Набранный пользователем текст содержится в строке **value**. Функция возвращает булевское значение:

True – если пользователь нажал OK или Enter;

False – если Cancel или Esc.

Пример вызова функции InputQuery:

if InputQuery('Пользователь','Введите фамилию', Name)

then Label1.Caption:= Name *{при нажатии на кнопку OK}*

else Label1.Caption:= ‘Иванов’ *{при нажатии на кнопку Cansel}*

§1.6. События и процедуры обработки события

Программам можно придать функциональность. Функциональность программы определяется совокупностью ее реакций на те, или иные события. Каждый компонент, кроме свойств, характеризуется также набором событий, на которые он может реагировать. Событие (Event) – это то, что происходит во время выполнения программы. У каждого события есть имя. Например, щелчок кнопкой мыши – это событие Click, двойной щелчок кнопкой – событие DblClick, нажатие клавиши клавиатуры – событие KeyPress. [7]

Таблица 5

События

Событие	Когда происходит
OnClick	При щелчке кнопкой мыши
OnDblClick	При двойном щелчке кнопкой мыши
OnMouseDown	При нажатии кнопки мыши
OnMouseUp	При опускании кнопки мыши
OnMouseMove	При перемещении мыши
OnKeyPress	При нажатии клавиши клавиатуры
OnKeyDown	При нажатии клавиши клавиатуры. Событие OnKeyDown и OnKeyPress – это чередующиеся, повторяющиеся события, которые происходят до тех пор, пока не будет опущена удерживаемая клавиша (в этот момент происходит событие OnKeyUp)
OnKeyUp	При опускании нажатой клавиши клавиатуры
OnCreate	При создании объекта (формы, элемента управления). Процедура обработки этого события обычно используется для инициализации переменных, выполнения подготовительных действий.

OnPaint	При появлении окна на экране в начале работы программы, после появления части окна, которая, например, была закрыта другим окном, и в других случаях
OnEnter	При получении элементом управления фокуса
OnExit	При потере элементом управления фокуса

Реакцией на событие должно быть какое-либо действие. В Delphi реакция на событие реализуется как процедура обработки события. Для того, чтобы программа выполняла некоторую работу в ответ на действия пользователя, программист должен написать процедуру обработки соответствующего события. Следует обратить внимание на то, что значительную часть обработки событий берет на себя компонент. Поэтому разрабатывать процедуру обработки события только в том случае, если реакция на событие отличается от стандартной или не определена. Например, если по условию задачи ограничений на символы, вводимые в поле Edit, нет, то процедуру обработки событий **OnKeyPress** писать не надо, т.к. во время работы программы будет использована стандартная процедура обработки этого события.

Комментарии

Существует несколько способов задания комментариев:

// позволяет закомментировать текущую строку;

{…} комментируемый фрагмент заключён между скобками и может располагаться на нескольких строках;

(*…*) аналогично предыдущему варианту.

§1.7 .Структура проекта

Проект Delphi представляет собой набор программных единиц – модулей. Один из модулей – главный, содержит инструкции, с которых

начинается выполнение программы. Главный модуль приложения полностью формируется Delphi.

Главный модуль представляет собой файл с расширением dpr. Для того чтобы увидеть текст главного модуля приложения, нужно из меню **Project** выбрать команду **View Source.**

Начинается главный модуль словом **program**, за которым следует имя программы, совпадающее с именем проекта. Имя проекта задается в момент сохранения проекта, и оно определяет имя создаваемого компилятором исполняемого файла программы. Далее за словом **uses** следует имена используемых модулей: библиотечного модуля **Forms** и модуля формы. [9]

Строка {$R *.RES}, которая похожа на комментарий, - это директива компилятору подключить файл ресурсов. Файл ресурсов содержит ресурсы приложения: пиктограммы, курсоры, битовые образы и др. Звездочка показывает, что имя файла ресурсов такое же, как и у файла проекта, но с расширением res.

Файл ресурсов не является текстовым файлом, поэтому просмотреть его с помощью редактора текста нельзя. Для работы с файлами ресурсов используют специальные программы, например, **Resource Workshop.** Можно также применить входящую в состав Delphi утилиту **Image Editor,** доступ к которой можно получить выбором из меню **Tools** команды **Image Editor**.

Исполняемая часть главного модуля находится между инструкциями begin и end. Инструкции исполняемой части обеспечивают инициализацию приложения и вывод на экран стартового окна.

Помимо главного модуля, каждая программа включает в себя еще как минимум один модуль формы, который содержит описание стартовой формы приложения и поддерживающих ее работу процедур. В Delphi каждой форме соответствует свой модуль. [7]

Начинается модуль словом **Unit**, за которым следует имя модуля. Именно это имя упоминается в списке используемых модулей в инструкции uses главного модуля приложения. Модуль состоит из следующих разделов:

- Интерфейса;
- Реализации;
- Инициализации.

Раздел интерфейса (начинается словом **interface)** сообщает компилятору, какая часть модуля является доступной для других модулей программы. В этом разделе перечислены (после слова uses) библиотечные модули, используемые данным модулем. Также здесь находится сформированное Delphi описание формы, которое следует за словом type.

Раздел реализации открывается словом **implementation** и содержит объявление локальных переменных, процедур и функций, поддерживающих работу формы.

Начинается раздел реализации директивой **{$R *.DFM},** указывающей компилятору, что в процессе генерации выполняемого файла надо использовать описание формы. Описание формы находится в файле с расширением dfm, имя которого совпадает с именем модуля. Файл описания формы генерируется средой Delphi на основе внешнего вида формы.

За директивой {$R *.DFM} следует процедуры обработки событий для формы и ее компонентов.

Раздел инициализации позволяет выполнить инициализацию переменных модуля. Инструкции раздела инициализации располагаются после раздела реализации (описание всех процедур и функций) между begin и end. Если раздел инициализации не содержит инструкций, то слово begin не указывается [5].

Сохранение проекта

Проект – это набор файлов, используя которые компилятор создает исполняемый файл программы (EXE - файл). В простейшем случае проект состоит из файла описания проекта (DOF - файл), файла главного модуля (DPR - файл), файла ресурсов (RES - файл), файла описания формы (DFM - файл), файла модуля формы, в котором находится основной код приложения,

в том числе функции обработки событий на компонентах формы (PAS - файл), файл конфигурации (CFG - файл).

Чтобы сохранить проект, нужно из меню **File** выбрать команду **Save Project As.** Если проект еще ни разу не был сохранен, то Delphi сначала предложит сохранить модуль (содержимое окна редактора кода), поэтому на экране появится окно **Save Unit1 As**. В этом окне надо выбрать папку, предназначенную для файлов проекта, и ввести имя модуля. После нажатия кнопки *Сохранить*, появится следующее окно, в котором необходимо ввести имя файла проекта

Обратите внимание на то, что имена файлов модуля (pas - файл) и проекта (dpr - файл) должны быть разными. Имя генерируемого компилятором исполняемого файла совпадает с именем проекта. Поэтому файлу проекта следует присвоить такое имя, которое, по вашему мнению, должен иметь исполняемый файл программы, а файл модуля – какое-либо другое имя, например, полученное путем добавления к имени файла проекта порядкового номера модуля [5].

ГЛАВА II

ПРАКТИЧЕСКАЯ ЧАСТЬ

2.1. Вводный курс визуального программирования

2.1.1. Практическая работа на тему «Среда Delphi»

Основой визуальной разработки программ в Delphi является понятие компонента. Они могут быть визуальными, т.е. видимыми в окне на экране: строка ввода, текстовая метка, кнопка, список, компонента для отображения картинок, таблица и др. Существуют также невизуальные компоненты, т.е. выполняющие какие-то функции, но не имеющие прямого графического представления на экране. Это могут быть компоненты для создания меню, связи с базами данных, таймер и др.

Цель работы:

Изучить визуальную среду программирования Delphi.

Задачи:

1. Рассмотреть основные визуальные элементы Delphi: формы и ее свойства; компоненты: метки, кнопки, строки ввода.

2. Создать программы-калькулятора для вычисления суммы и произведения двух чисел.

Описание работы:

При работе в среде Delphi создается достаточно большое количество различных файлов. Поэтому перед началом разработки программ создать на диске отдельную папку для каждой практической работы. Например – Delphi, с вложенными папками: Labl, Lab2 и т.д.

1. Запуск программы Delphi. На экране появляется среда разработки, по умолчании состоящая из следующих окон:

- меню: панель инструментов и палитра компонентов;
- Object Inspector (инспектор объектов), предназначенное для изменения различных свойств редактируемых компонентов проекта Delphi;

- Form1, для автоматического создания формы проекта, с возможностью размещения компонентов, определяющих интерфейс разрабатываемого приложения;

- текстовый редактор, для ввода программы.

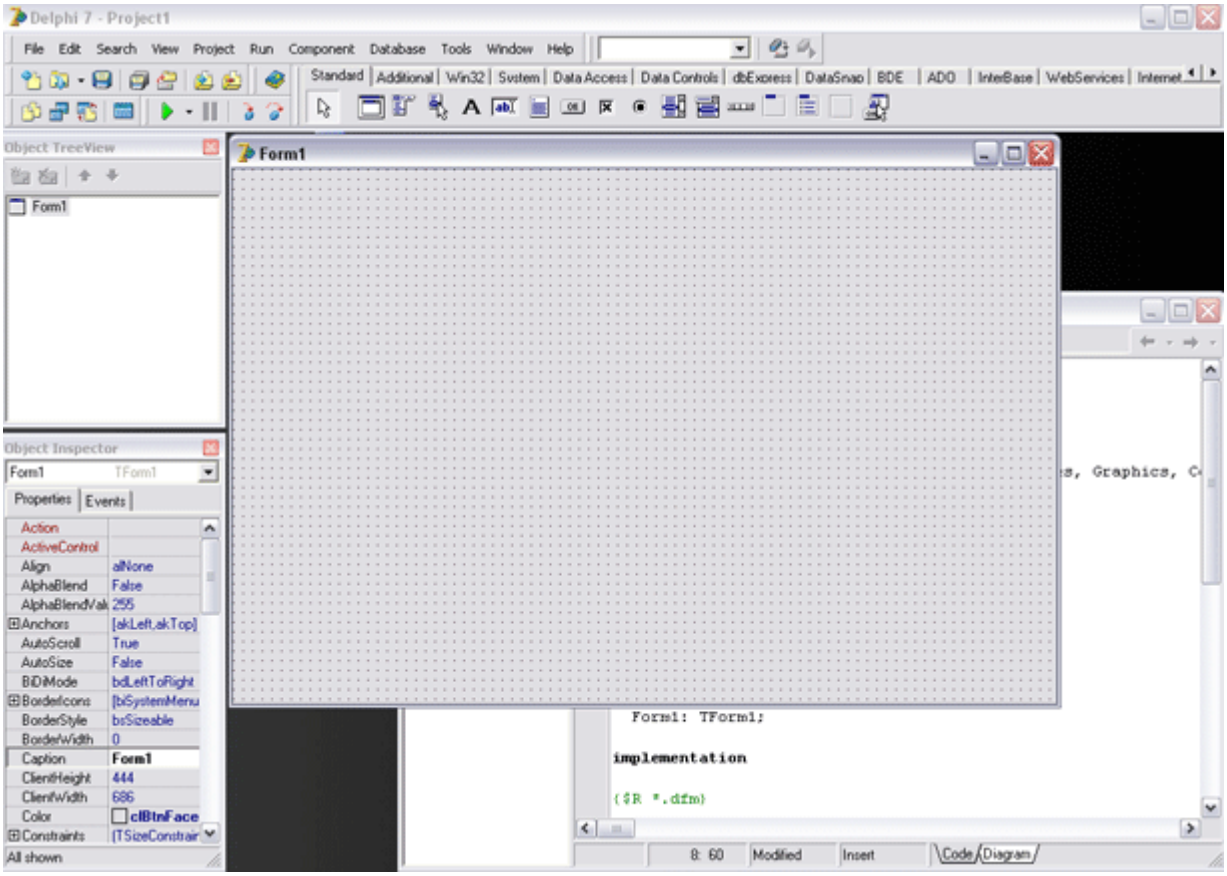


Рис. 7 Интерфейс Delphi 7

2. Создать с помощью команды меню **File / New Application** - новый проект и его сохранить **File / Save** в рабочую папку

Delphi сохраняет модуль с формой под именем **Unitl.pas,** а проект - **Projectl.dpr**. Можно оставить или изменить имена, например **MainUnit.pas** и **Labl.dpr.**

3. Разместить в форме необходимые компоненты, изменяя их параметры в редакторе кода программы.

Для начала попробуйте поменять различные свойства формы в инспекторе объектов на закладке **Properties** (другая закладка Events будет использована нами позднее для создания обработчиков событий). В первую очередь необходимо указать свойство **Name** формы. Оно будет

использоваться в качестве имени переменной, ссылающейся на объект, представляющий форму в коде программы. Например, установите его значение в **MainForm**. Теперь измените свойство **Caption**. Оно устанавливает текст заголовка окна. Так как мы собираемся в данной работе создать калькулятор, то установите значение **Caption,** например, равным слову «Калькулятор».

Замечания:

1. Не путайте свойства Name и Caption. Свойство Name определяет имя переменной для ссылки в программе на объект, а Caption - текст заголовка визуального компонента.

Каждый компонент имеет множество свойств. **Color** определяет цвет, **Width** - ширину, **Heigh** - высоту, **Name** - имя и т.д.

Многие свойства являются составными, при этом в инспекторе объектов слева от названия свойства отображается знак «+».

2. Не забывайте, что ваши приложения должны быть похожи на остальные программы Windows. Поэтому не меняйте без особой нужды стандартные значения визуальных свойств (цвета, шрифты, размеры).

При соблюдении всех этих рекомендаций экранные формы созданного вами приложения сами будут менять цвета, шрифты, размеры своих компонентов в зависимости от системных установок, сделанных пользователем централизованно для всех приложений Windows.

На рис. 8 показан внешний вид формы во время разработки. Необходимо разместить на форме 5 текстовых меток с надписями «Введите два числа:», «Сумма», «Произведение» и двумя «0», две строки ввода и две кнопки «Вычислить» и «Выход». Для этого надо выбрать на палитре компонентов необходимые элементы, щелкнуть по ним мышкой и затем щелкнуть мышкой в место их расположения на форме. При необходимости компоненты можно будет переместить на новое место и изменить их размер.

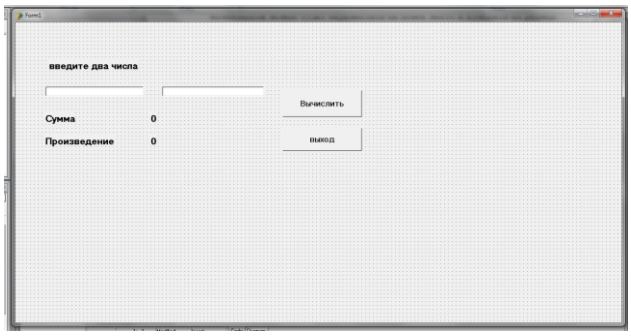


Рис.8 Внешний вид формы во время разработки

После размещения компонентов им следует задать необходимые свойства. Для этого надо мышкой выделить на форме нужный компонент, после чего указать его свойства в инспекторе объектов. Для изменения свойств формы нужно указать мышкой в форму мимо всех других компонент, либо нажать клавишу Esc.

Следующим шагом при разработке нашей программы будет написание обработчиков событий различных компонентов. Примерами событий могут быть, например, перемещение мышки, нажатие кнопок мышки или клавиш на клавиатуре, открытие и закрытие окна и т.д.

Для создания обработчика какого-либо события необходимо выбрать мышкой нужный компонент, выбрать в инспекторе объектов закладку Events, найти требуемое событие и дважды щелкнуть на нем мышкой.

При создании нового обработчика Delphi создает в программном модуле, соответствующем данной форме, пустую заготовку процедуры обработки события примерно в таком виде:

procedure
T<Имя_формы>.<Имя_компонента><Имя_события>(Sender: TObject);

begin

end;

3. Не пишите пустые заготовки сами - они не будут вызываться при выполнении программы!

Пока эта процедура пустая, но в дальнейшем она должна быть заполнена необходимым кодом, который будет вызываться в ответ на

указанное событие. При этом в форме в качестве специального свойства компонента будет установлено соответствие между выбранным событием и именем созданной процедуры. Поэтому, если самостоятельно написать в тексте кода соответствующие процедуры-обработчики без инспектора объектов, они никогда не будут вызваны. Однако, если у вас уже написаны нужные процедуры, то вы можете в инспекторе объектов указать это имя для нужного события с помощью выпадающего списка, появляющегося при нажатии кнопки справа от имени события (рис. 9).

Рис. 9 Инспектор объектов

В нашем приложении необходимо создать два обработчика события в ответ на нажатия кнопок **ButtonCalc** («Вычислить») и **ButtonExit** («Выход»). Такое событие для компонента типа TButton называется OnClick. В ответ на двойной щелчок в инспекторе объектов на месте значения обработчика этого события Delphi сформирует следующие заготовки:

```
procedure TMainForm.ButtonCalcClick(Sender: TObject);
begin
end;
procedure TMainForm.ButtonExitClick(Sender: TObject);
begin
end;
```

Формальный параметр **Sender** является указателем на породивший событие компонент. Он может использоваться в случае, если один обработчик событий назначен для нескольких разных компонентов. В данной работе он нам не понадобится, но будет использоваться в последующих.

4. ***Не создавайте все пустые заготовки сразу. При сохранении формы Delphi найдет все пустые обработчики событий, удалит их, и вам придется их создавать заново.***

Заполняем полученные заготовки обработчиков необходимым кодом.

Текст программного модуля MainUnit.pas

```
unit MainUnit;
interface
uses Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs, StdCtrls;
type
TMainForm = class(TForm)
Labell: TLabel;
Edit1: TEdit;
Edit2: TEdit;
ButtonCalc: TButton;
Label2: TLabel;
Label3: TLabel;
LabelSum: TLabel;
LabelProduct: TLabel;
procedure ButtonCalcClick(Sender: TObject);
procedure ButtonExitClick(Sender: TObject);
end;
var
MainForm: TMainForm;
implementation
{$R *.DFM}
procedure TMainForm.ButtonCalcClick(Sender: TObject);
var A, B: double;
begin // Нажата кнопка "Вычислить"
```

```
try
A:=StrToFloat(Edit1.Text); // Преобразовать текст в число
except
Editl.SetFocus;
ShowMessage ('Ошибка в первом числе');
exit;
end;
try B:=StrToFloat(Edit2.Text); // Преобразовать текст в число
except
Edit2.SetFocus;
ShowMessage('Ошибка во втором числе');
exit;
end;
LabelSum.Caption:=FloatToStr(A+B);
LabelProduct.Caption:=FloatToStr(A*B);
end;
procedure TMainForm.ButtonExitClick(Sender: TObject);
begin // Выход из программы
Close;
end;
end.
```

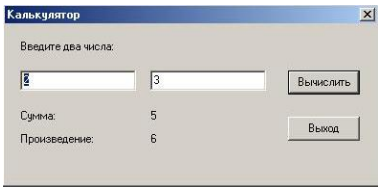


Рис. 10 Внешний вид готового приложения

2.1.2. Практическая работа на тему «Создание модальных форм»

Основным строительным блоком в Delphi является форма. Любая программа имеет как минимум одну связанную с ней форму, которая называется «главной». Главная форма появляется на экране в момент старта программы. Программа может иметь сколько угодно форм, каждая из которых решает какую-то локальную задачу и появляется на экране по мере надобности.

Современные многооконные приложения чаще всего строятся в стиле SDI (Single Document Interface), который не накладывает ограничений на положение и размеры вспомогательных форм. В стиле SDI реализована сама среда Delphi. В данной лабораторной работе вы познакомитесь с одним из способов использования вспомогательной формы в качестве модального диалогового окна.

Задача для программирования взята та же самая, что и в первой практической работе. Но для ввода двух целых чисел должна использоваться вспомогательная форма, а обработка введенных чисел и вывод результатов должен проводиться в модуле, связанном с главной формой.

Цель работы:

Создать приложение с несколькими формами.

Задачи:

1. Ознакомление с понятиями модальной и немодальной формы.

2. Создание программы-калькулятора для вычисления суммы и произведения двух чисел с использованием диалогового окна (модальной формы).

Описание работы:

Создаем отдельную папку для сохранения всех файлов данной работы, например, Lab2. После запуска Delphi и создания нового проекта необходимо сохранить в созданную папку главный модуль под именем Main Unit.pas (вместо предлагаемого Delphi имени Unit 1.pas), а проект под именем

Lab2.dpr (вместо Projectl.dpr). После этого можно начинать заполнять главную форму необходимыми компонентами в соответствии с рис. 11.

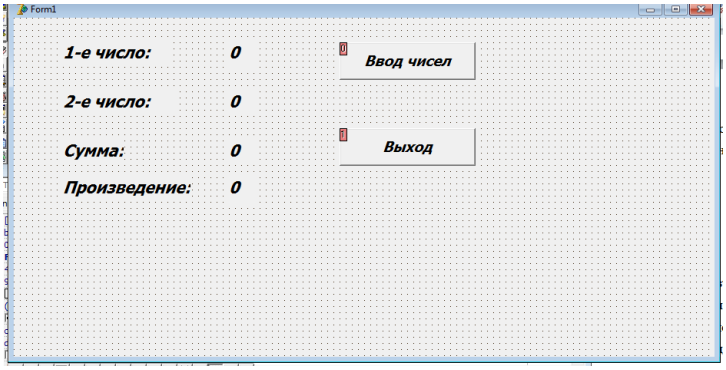


Рис.11 Главная форма

Создадим вторую (вспомогательную) форму для ввода двух чисел. Необходимо выбрать в меню команду **File / New Form**. В результате будет создана новая форма с соответствующим программным модулем. Модуль необходимо сохранить (командой File / Save) в рабочем каталоге Lab2 в файл Input.pas.

Заполнить созданную вспомогательную форму необходимыми компонентами в соответствии с рис. 12. После того как обе формы спроектированы, необходимо написать соответствующие обработчики событий для связи форм друг с другом.

В Delphi вспомогательные окна можно выводить на экран в двух режимах с помощью методов формы Show и ShowModal соответственно.

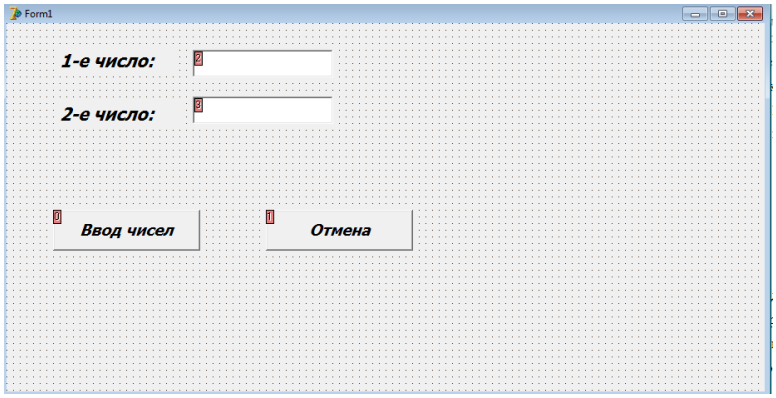


Рис.12 Вспомогательная форма

Метод **Show** открывает «немодальное» окно, которое появляется на экране и работает одновременно с вызвавшим его окном. Управление программой передается оператору, стоящему за оператором вызова метода **Show.** Этот режим обычно не используется для ведения диалога с пользователем. Примером такого рода окон являются окна Редактора кода и Инспектора объектов в Delphi.

Метод **ShowModal** открывает «модальное диалоговое» окно, которое полностью берет на себя дальнейшее управление программой. Модальные окна требуют от пользователя принятия какого-то решения. Для того, чтобы пользователь мог сообщить о принятом решении, в модальное окно вставляются необходимые интерфейсные элементы. Чаще всего это стандартные кнопки (с надписями «ОК», «Отмена», «Да», «Нет»), которые автоматически выполняют действия закрытия модального окна и помещения значения результата диалога (информации о нажатой кнопке) в свойство формы **ModalResult.** Свойство **ModalResulz** может иметь значения **mrNone, mrOk, mrCancel, mrYes, mrNo** и другие. Вызывающая программа получает это значение после выполнения метода **ShowModal** и может анализировать его для определения нужного направления своей дальнейшей работы.

Замечания:

1. Вывод диалоговых окон (модальных форм) на экран удобен, если они появляются по центру экрана. Для этого необходимо значение свойства Position формы равным psScreenCenter.

Необходимо создать обработчик событий в ответ на нажатие кнопки **ButtonInput** («Ввод чисел...») главной формы, который бы вызывал вспомогательную форму в модальном режиме. Следует указать, что вспомогательный МОДУЛЬ **Input.pas** будет использоваться в главном модуле **MainUnit.pas**.

-открыть модуль **MainUnit.pas**;

-выбрать команду меню **File / Use Unit**...

-указать в диалоге модуль **Input.pas.**

Результат: в главном модуле в секции **implementation** появится строка «uses Input».

Главным в создаваемом обработчике является метод **ShowModal** формы InputForm, который останавливает выполнение основной программы и передает управление вспомогательной форме. В зависимости от результата выполнения метода ShowModal (нужно выполнить проверку на равенство результата значению mrOk) необходимо выполнить действия по вычислению суммы и произведения чисел и вывести полученные значения в соответствующих метках главной формы.

Во вспомогательной форме необходимо написать обработчик событий для кнопки Buttor.Ok («OK»), который должен проверить правильность введения чисел и, если все верно, то закрыть форму с результатом mrOk. Для второй кнопки BuLtonCancel («Отмена») обработчик писать не нужно, т.к. у этой кнопки уже выставлено свойство ModalResult (форма должна быть закрыта).

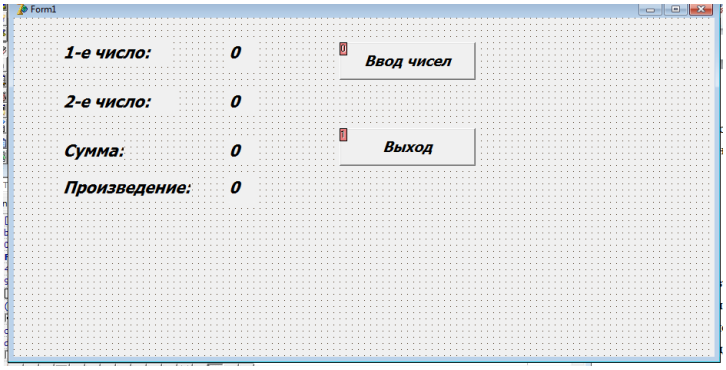


Рис.13 Главная форма

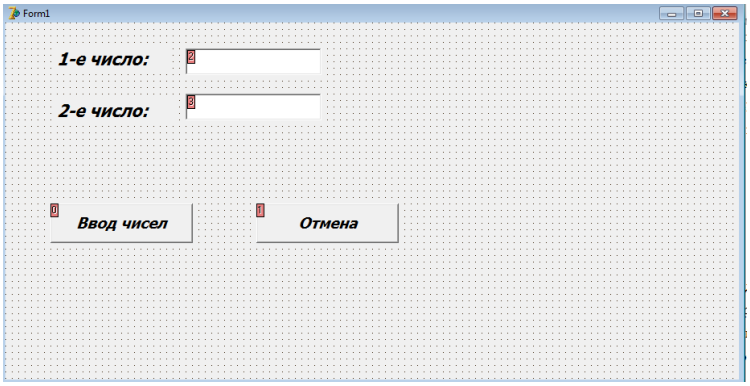


Рис. 14 Вспомогательная форма

Текст файла проекта Lab2.dpr

```
program Lab2;
uses
Forms,
Input in 'Input.pas' {InputForm},
MainUnit in 'MainUnit.pas' {MainForm};
{$R *.RES}
begin
Application.Initialize;
Application.CreateForm (TMainForm, MainForm);
Application.CreateForm (TInputForm, InputForm);
Application.Run;
end.
```

Текст главного модуля MainUnit.pas

```
unit MainUnit;
interface
uses
Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
StdCtrls;
type
TMainForm = class(TForm)
Labell: TLabel;
Label2: TLabel;
Label3: TLabel;
Label4: TLabel;
LabelNumberl: TLabel;
LabelNumber2: TLabel;
LabelSum: TLabel;
```

```
LabelProduct: TLabel;
ButtonInput: TButton;
ButtonExit: TBatton;
procedure ButtonExitClicktSender: TObject);
procedure ButtonInputClick(Sender: TObject);
end;
var
MainForm: TMainForm;
inplementation
uses Input;
{$R *.DFM}
procedure TMain.Form.ButtonExitClick (Sender: TObject);
begin // Выход из программы
Close;
end;
procedure TMainForm.ButtonInputClick(Sender: TObject);
begin // Нажата кнопка "Ввод чисел..."
if InputForm.ShowModal=mrOk then
with InputForm do
begin
LabelNumberl.Caption:=Edit1.Text;
LabelNumber2.Caption:=Edit2.Text;
try
labelSum.Caption:=FloatToStr(Number1+Number2);
except
LabelSum.Caption:= 'Ошибка сложения';
end;
try
LabelProduct.Caption:=FloatToStr(Number1*Number2);
except
```

```
LabelProduct.Caption:='Ошибка умножения';
end;
end;
end;
end.
```

Текст вспомогательного модуля Input.pas

```
unit Input;
interface
uses
Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
StdCtrls;
type
TInputForm = class(TForm)
Labell: ILabel;
Label2: TLabel;
Editl: TEdit;
Edit2: TEdit::
ButtonOk: IButton;
ButtonCancel : TButton;
procedure ButtonOkClick (Sender : TObject);
public
Number1, Number2: double;
end;
var
InputForm: TInputForm;
implementation
{$R *.DFM}
procedure TInputForm.ButtonOkClick (Sender: TObject);
begin
```

```
try
Numberl:=StrToFloat(Editl.Text);
except
Edit1.SetFocus;
ShowMessage ('Ошибка в первом числе');
exit;
end;
try
Number2:=StrToFloat(Edit2.text);
except
Edit2.SetFocus;
ShowMessage ('Ошибка во втором числе');
end;
ModalResult:=mrOk; {Числа введены правильно, поэтому форму можно закрывать}
end; end.
```

2.1.3. Задачи для самостоятельного решения

Пример 1

Создать проект, при запуске которого на метку выводится сообщение «Здравствуй, Delphi!».

Программный код

```
unit Pr1;
interface
uses
Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms, Dialogs, StdCtrls;
type
TForm1=class (TForm)
```

```
Label1:TLabel;
Button1:TButton;
Procedure Button1Click (Sender:TObject);
private
{ Private declarations }
public
{Public declarations}
end;
Var
Form1:TForm1;
implementation
{$R*.dfm}
procedure TForm1.Button1Click (Sender: TObject);
begin
Form1.Close;
end; end.
```

Результат:

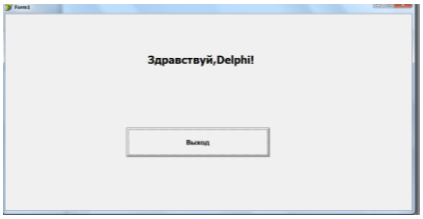


Рис. 15 Результат выполнения программы

Пример 2.

Создать проект, который выводит на метку формы вопрос «Как Вас зовут?». Однострочный редактор – поле Edit используется для ввода данных пользователем.

Программный код

```
Unit Znakom;
Interface
```

```
Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
Dialogs, StdCtrls;
type
TForm1=class (TForm)
Label1:TLabel;
Button1:TButton;
Edit1:TEdit;
Procedure Button1Click (Sender:TObject);
Procedure FormCreate (Sender:TObject);
private
{Private declarations}
public
{Public declarations}
end;
Var
Form1:TForm1;
implementation
{$R*.dfm}
procedure TForm1.Button1Click (Sender: TObject);
begin
Form1.Close;
end;
procedure TForm1.FormCreate (Sender:TObject);
begin
Edit1.Text:='';
end;
end.
```

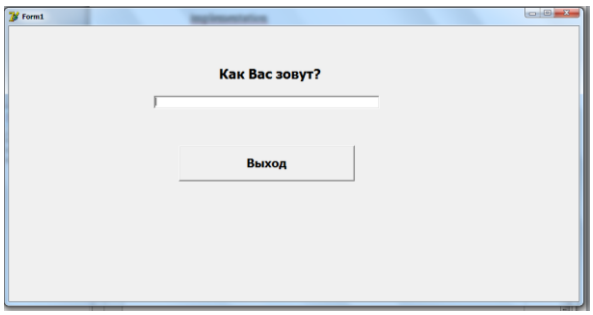


Рис. 16 Результат выполнения программы

Пример 3

Создать проект, в результате работы которого можно изменять цвет различных объектов, сообщений (надписей) на них, записать процедуры обработки событий.

```
unit Unit1;
interface
uses
Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
Dialogs, StdCtrls;

type
TForm1 = class(TForm)
label1: TLabel;
btn1: TButton;
btn2: TButton;
btn3: TButton;
btn4: TButton;
btn5: TButton;
btn6: TButton;
btn7: TButton;
procedure btn1Click(Sender: TObject);
procedure btn2Click(Sender: TObject);
procedure btn3Click(Sender: TObject);
procedure btn5Click(Sender: TObject);
```

```
procedure btn6Click(Sender: TObject);
procedure btn7Click(Sender: TObject);
procedure btn4Click(Sender: TObject);
private
{Private declarations}
public
{Public declarations}
end;
var
Form1: TForm1;
implementation
{$R *.dfm}
procedure TForm1.btn1Click(Sender: TObject);
begin
close;
end;
procedure TForm1.btn2Click(Sender: TObject);
begin
Lbl1.Color:=clYellow;
Lbl1.Font.Color:=clBlue;
end;
procedure TForm1.btn3Click(Sender: TObject);
begin
Lbl1.Color:=clWhite;
Lbl1. Font. Color:=clRed;
end;
procedure TForm1.btn5Click(Sender: TObject);
begin
Form1.Color:=clRed;
end;
```

```
procedure TForm1.btn6Click(Sender: TObject);
begin
Form1.Color:=clYellow;
end;
procedure TForm1.btn7Click(Sender: TObject);
begin
Form1.Color:=clSilver;
end;
procedure TForm1.btn4Click(Sender: TObject);
begin
Lbl1.Color:=clAqua;
Lbl1.Font.Color:=clFuchsia;
end;
end.
```

Результат:

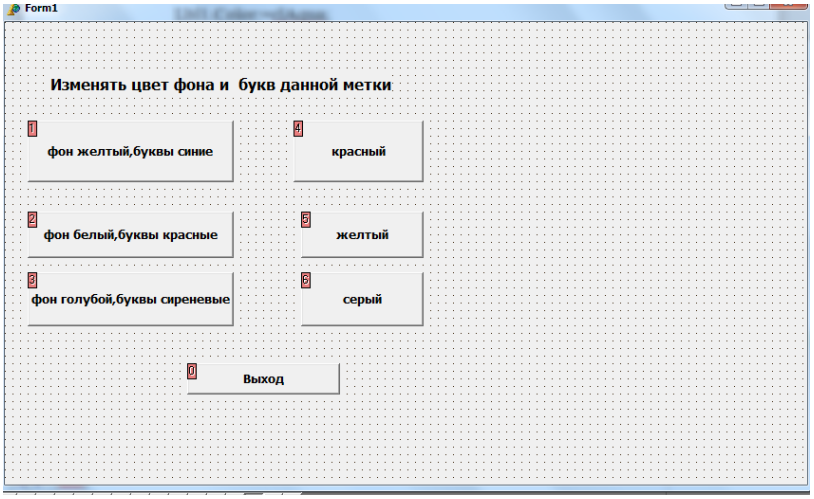


Рис.17 Результат выполнения программы

Пример 4

Разработать проект, при выполнении которого текст из строки ввода Edit будет копироваться без изменений в метку Label и многострочный редактор Memo.

Программный код.

```
unit Uch_prog
```

```
interface
uses
Windows, Messagee, SysDtils, Variants, Classes, Graphics, Controls, Forms,
Dialogs, StdCtrls, Buttons, ExtCtrls;
type
TForml = class (TForm)
Panell: TPanel;
BitBtnl: TBitBtn;
BitBtn2: TBitBtn;
Editl: TEdit;
Memol: TMemo;
Labell: TLabel;
procedure BitBtnlClick(Sender: TObject);
procedure FormActivate(Sender: TObject);
private
{Private declarations}
public
{Public declarations}
end;
var
Forml: TForml;
implementation
{$R *.dfm}
procedure TForml.BitBtnlClick(Sender: TObject);
begin
Labell.Caption:=Editl.Text;      {Повторяем  текст в  метке}
Memol.Lines.Add(Editl.Text); {и в многострочном редакторе}
Editl.Text:='';        {Очищаем строку ввода}
Editl.SetFocus;      {Передаем  ей  фокус ввода}
end;
```

```
procedure TForml.FormActivate (Sender: TObject);
begin
Editl.SetFocus;
end;
end.
```

Пример 5.

Разработать проект, в котором скорость ветра задается в метрах в секунду (м/с) , а результат — в километрах в час (км/ч). Скорость вводить только как целые числа.

Код процедуры обработки события кнопки Пересчет.

```
procedure TForml.ButtonlClick(Sender: TObject);
var
ms: integer; km: real;
begin
ms:=StrToInt (Editl.Text);
km:=ms*3.6;
Label3.Caption:=IntToStr(ms)+' м/с - это '+FloatToStr(km) + ‘км/ч’;
end;
end.
```

2.1.4. Контрольные задачи

1. Создать проект, в результате работы которого на метку Labell выводится одно из сообщений в зависимости от того, на какой кнопке пользователь щелкнул мышью. Установить значение свойства метки ***WordWrap = true.***

Наименование	Отображаемый текст на метке Labell
Delphi	Объектно-ориентированный язык
Forml	Заготовка главного окна
Object	Окно редактора свойств объекта

Object Tree	Окно просмотра списка объекта
Unitl.pas	Окно редактора кодов объекта

2. Разработать проект, в процессе выполнения которого над рисунком, первоначально расположенным в центре формы, выполняются действия, указанные в надписях на кнопках.(Центр, вверх вправо, уменьшить, вниз вправо, вниз влево, увеличить, вверх влево, выход)

3. Разработать проект, выполняющий указанные далее действия. После запуска приложения на форме пользователь видит две командные кнопки – Quiz (загадка) и Exit (выход). Командная кнопка Answer (отгадка) скрыта, чтобы пользователь не увидел рисунка с отгадкой до тех пор, пока не прочитает тест загадки. Щелкнув кнопкой мыши на кнопке Quiz, он прочитает на форме текст загадки, затем, щелкнув на кнопке Answer, увидит картинку с отгадкой на эту загадку. По щелчку мыши на кнопке Exit проект завершит работу. Текст метки, его размер, шрифт, начертание можно задавать с помощью Инспектора Объектов. Мы предлагаем воспользоваться другим способом: в процедуру обработки щелчка на кнопке Quiz (загадка) включить следующий программный код.

```
Label1.Show;
Label1.Font.Name:=’Arial’;
Label1.Font.Size:=14;
Label1.Font.Color:=00006200;
Label1.Caption:=’What is that  is sometimes with a head and sometimes without a head?’;
```

4. Разработать проект, вычисляющий скорость, с которой спортсмен пробежал дистанцию.

5. Разработать проект, в котором первоначально заданы две переменные. Используя третью переменную С, поменять местами значения переменных А и В.

2.2. Основной курс

2.2.1. Практическая работа на тему «Компонент Shape» на примере задачи «Светофор»

Нарисовать светофор с тремя лампочками, которые реагируют на наведение указателя мыши.

Для рисования лампочек светофора использовать компонент *Shape*. Компонент *Shape* – стандартная фигура (страница Additional), рисует одну из простейших геометрических фигур. Вид фигуры определяется свойством *TShapeType*. Это свойство может принять следующие значения (прямоугольник, квадрат, эллипс, окружность).

Контур фигуры определяется свойством *Pen*. Это составное свойство. Раскрыть это свойство. Свойство *Pen.Style* определяет способ вычерчивания линий. Цвет контура определяется свойством Color: *ClRed* – ярко красный, *CiYellow* – ярко-желтый, *ClLime* – ярко-зеленый. Мы хотим, чтобы первоначально фигуры были не закрашенными, т.е. прозрачными. Прозрачность задается свойством *Style-bsClear*.

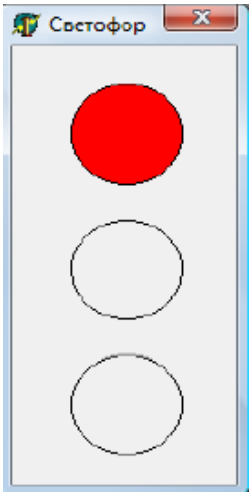


Рис.18 Готовое приложение

Фигура полностью занимает все пространство компонента. Если задан круг или квадрат, а размеры компонента по горизонтали и по вертикали отличаются, фигуры чертится с размером меньшего измерения. Параметры закраски определяются свойством *Brush*. Свойство *Brush.Color* определяет цвет заполнения области. Свойство *Brush.Style* определяет стиль заполнения области.

На окне формы разместить 3 компонента *Shape*. Ширина и высота каждого компонента по 65. Форма фигуры задается свойством *Shape*. Для каждой фигуры из раскрывающегося списка, выбрать значение *StCircle*.

Фигуры могут реагировать на наведение мыши. Событие MouseMove можно было бы обработать для каждой фигуры. Фигуры круглые, но маркеры изменения размера располагаются по сторонам прямоугольника. Система Delphi продолжает считать фигуры прямоугольниками. Это означает, что при наведении указателя мыши на уголок фигуры, лампочки будут зажигаться. Чтобы этого не произошло, лампочки надо «выключить», т.е. свойством *Enabled* (включен) каждой фигуры задать значение false (нет).

Для формы *Form1* установить свойства:

Caption - светофор, Height – 300, Width – 120.

Окно станет таким узким, что кнопки управления размером окна, присутствующие в строке заголовка, будут мешать увидеть заголовок окна целиком. Отображение стандартных кнопок задается составным свойством *BorderIcons* (служебные кнопки). Открыть свойство и для *biMinimize* (сворачивание формы) и *biMaxmize* (разворачивание формы на весь экран) установить значение false. На форме ничего не изменится, но при выполнении программы эти кнопки (сворачивание и разворачивание) исчезнут.

Выделить все три фигуры нажимая на Shift, выполнить команду Edit – *Align* – (Правка - выровнять). Открывается диалоговое окно *Aligment* (Выравнивание), где установить:

по горизонтали Center on window (центрировать в окно),

по вертикали Space Equally (сравнение промежутками).

Для формы обработать событие OnMouseMove. Если указатель наведен на 1-ю лампочку, она зажигается, если нет, то гаснет. То же самое для остальных лампочек, т.е. для каждой лампочки выполняются одни и те же действия. Их можно оформить в виде отдельной функции:

Function Onshape (Sh:Tshape; X,Y:integer); TbrushStyle;

В функцию передается следующие параметры: имя фигуры, координаты указателя мыши.

Текст модуля

```
unit Unit1;
interface
uses
Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
Dialogs, ExtCtrls;
type
TForm1 = class(TForm)
shp1: TShape;
shp2: TShape;
shp3: TShape;
procedure FormMouseMove(Sender: TObject; Shift: TShiftState; X,
Y: Integer);
private
{ Private declarations }
public
{ Public declarations }
end;
var
Form1: TForm1;
implementation
{$R *.dfm}
function OnShape(Sh:Tshape; X,Y:integer):TbrushStyle;
var r, cx, cy: integer;
begin
r:=sh.width div 2;
cx:=sh.left+r;
cy:=sh.Top+r;
onshape:=bsClear;
if (x-cx)*(x-cx)+(y-cy)*(y-cy)<r*r then Onshape:=bsSolid;
```

```
end;
procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y:
Integer);
begin
shp1.Brush.Color:=clred;
shp1.Brush.style:=OnShape(shp1,x,y);
shp2.Brush.Color:=clyellow;
shp2.Brush.style:=OnShape(shp2,x,y);
shp3.Brush.Color:=cllime;
 shp3.Brush.style:=OnShape(shp3,x,y);
end; end.
```

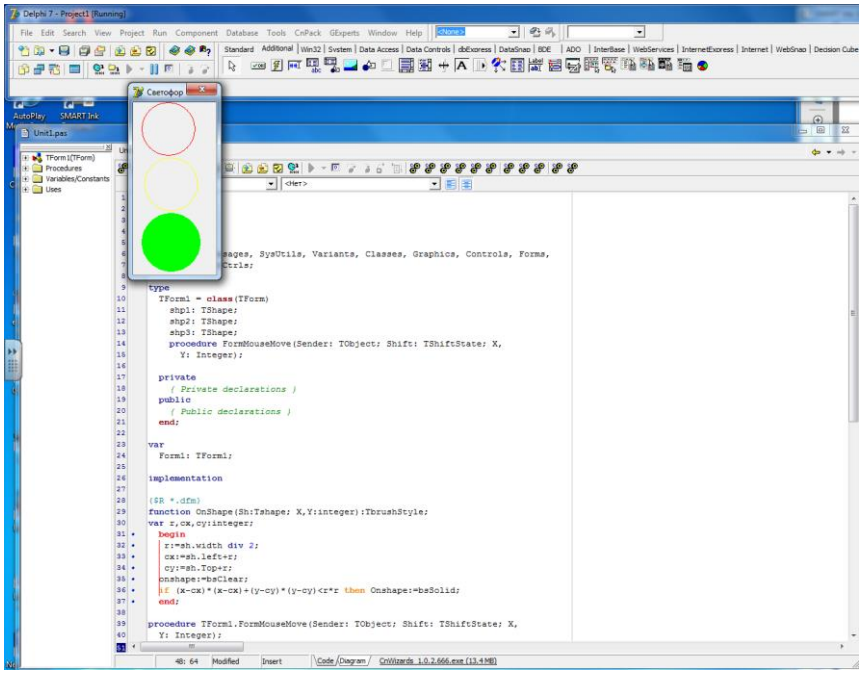


Рис. 19 Готовая программа

2.2.2. Практическая работа на тему «Ввод и обработка массивов»

В данной практической работе представлены некоторые компоненты Delphi для ввода и обработки числовых одномерных и двумерных массивов данных. Необходимо будет создать главную форму, позволяющую вывести на экран другие вспомогательные формы для ввода значений массивов.

После ввода значений необходимо их обработать (вычислить сумму и произведение всех элементов) и вывести результаты в главной форме.

Цель работы:

Освоение приёмов работы с массивами.

Задачи:

1. Ознакомление с компонентами для представления списков и таблиц, которые можно использовать в Delphi для работы с одномерными и двумерными массивами.

2. Создание программы для ввода и обработки одномерных и двумерных массивов.

Описание работы:

Создать новый проект, сохранить его в отдельном каталоге, в соответствии с рекомендациями, приведенными в предыдущих практических работах.

Приложение будет состоять из трех модулей:

- первый модуль **UnitMain.pas**, соответствующий форме **MainForm**, предназначен для координации работ других модулей и обработки массивов.

- второй модуль **Listlnput.pas**, соответствующий форме **List Form**, предназначен для ввода одномерного массива.

- третий модуль **Gridlnput.pas,** соответствующий форме GridForm, предназначен для ввода двумерного массива. Проект необходимо сохранить под именем Lab3.dpr.

Для визуализации работы с одномерным массивом можно использовать компонент класса **TListBox** со страницы **Standard** палитры компонентов. Этот визуальный компонент представляет собой стандартное окно списка Windows, позволяющее работать с отдельными элементами.

Для визуализации работы с двумерным массивом можно использовать компонент **TStringGrid** со страницы Additional палитры компонентов, предназначенный для создания таблиц, в ячейках которых располагаются произвольные текстовые строки.

Замечания:

1. Индексация элементов таблицы производится с помощью двух чисел -

номеров колонки (столбца) и ряда (строки). Нумерация ведется, начиная с 0.

Объявление динамического массива выполняется аналогично обычному, но диапазон изменения индексов не указывается. При этом нумерация элементов массива производится, начиная с 0. Для изменения количества элементов в динамическом массиве используется функция SetLength с двумя параметрами: именем переменной динамического массива и количеством элементов в массиве, например:

```
var SomeArray: array of integer;
SetLength (SomeArray,10);
for i:=0 to 9 do
SomeArray [i] :=0;
```

2. Для вывода на экран форму, находящуюся в другом модуле, используются методы Show и ShowModal. При этом необходимо внести в список uses в секции implementation имена этих модулей с формами.

Для создания динамического двумерного массива необходимо объявить динамический массив динамических массивов, а для изменения его размера вызвать функцию SetLength с еще одним дополнительным параметром, указывающим размерность массива по второй координате, например:

```
var SomeArray2: array of array of integer;
Setlength(SomeArray2,10,5); for i:=0 to 9 do
For j:=0 to 3 do
SomeArray2 [i, j] : =0;
```

Ниже приведены варианты форм, которые могут быть использованы для выполнения практической работы.

На главной форме используется компонент типа TGroupBox для объединения других компонентов, логически связанных друг с другом.

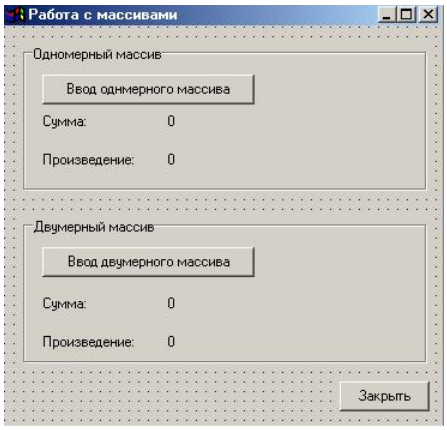


Рис. 20 Главная форма

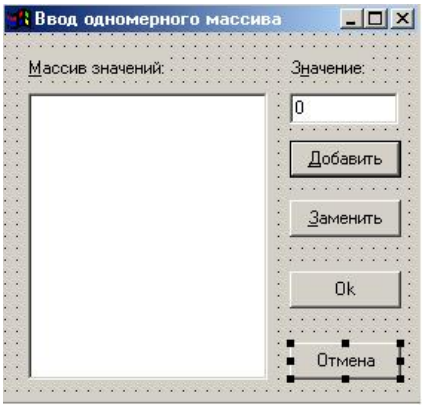


Рис. 21 Форма для ввода одномерного массива

На форме для ввода одномерного массива основным является компонент **TListBox**. В метках, размещенных на этой форме, свойство FocusedControl позволяет их использовать для быстрого перемещения на указанный компонент с помощью метки. Перемещение можно осуществлять нажатием мышкой на метке либо нажатием на клавиатуре комбинации Alt+<Подчеркнутая буква метки>. Для того, чтобы буква метки оказалась подчеркнутой, необходимо перед этой буквой в свойстве Caption поставить символ «&».

Например, для метки «Массив значений» необходимо установить свойство Caption равным «&Массив значений», а свойство FocusedControl — ListBoxArray.

На форме для ввода двумерного массива основным является компонент **TStringGrid** для ввода значений массива. Компонент **TPanel** используется для отображения сообщений об ошибке. У компонента TPanel необходимо

установить свойства **BevelOuter.AJM** изменения типа рамки. **Alignment** равным **alLeft**, для отображения текста, прижатого к левой стороне панели**.**

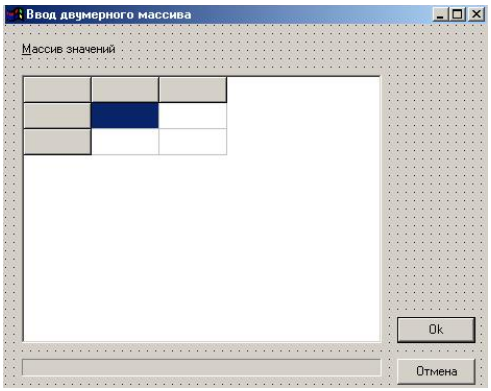


Рис. 22 Форма для ввода двумерного массива

При создании модуля **Gridlnput.pas** необходимо написать обработчик события формы **OnCreate**. В нем следует установить начальные значения ячеек таблицы.

В данном модуле необходимо создать метод формы UpdateGrid, используемый для автоматического сокращения количества ячеек в таблице. Delphi для такого рода методов не умеет создавать заготовки, поэтому это надо сделать полностью самостоятельно, описав этот метод в секции **private** и разместив тело метода в любом месте секции **implementation.**

Текст файла проекта Lab3.dpr

```
program Lab3;
uses
Forms,
MainUnit in 'MainUnit.pas' {MainForm},
ListInput in 'ListInput.pas' {ListForm},
GridInput in 'GridInput.pas' {GridForm};
{$R *.RES}
begin
Application.Initialize;
Application.CreateForm (TMainForm, MainForn);
Application.CreateForm (TListForm, ListForm);
```

```
Application.CreateForm(TGridForm, GridForm);
Application.Run;
end.
```

Текст главного модуля MainUnit.pas

```
unit MainUnit;
interface
uses
Windows, Messages, SysUtiis, Classes, Graphics, StdCtrls, Forms, Dialogs;
type
TMainForm = class(TForm)
GroupBox1: TGroupBox;
ButtonList: TButton;
Label1: TLabel;
Label2: TLabel;
LabelProduct1: TLabel;
LabelSum1: TLabel;
GroupBox2: TGroupBox;
Label3: TLabel;
Label4: TLabel;
LabelProduct2: TLabel;
LabelSum2: TLabel;
ButtonGrid: TButton;
ButtonClose: TBLtton;
procedure ButtonListClick(Sender: TObject);
procedure ButtonCloseClick(Sender: Tabject);
procedure ButtonGridClick(Sender: TObject);
end;
var
MainForm: TMainForm;
```

```
implementation
uses GridInput, ListInput;
{$R *.DFM}
procedure TMainForm.ButtonListClick (Sender: TObject);
var
a: array of double; {Объявление динамического массива вещественных чисел}
i: integer; s, p: double;
begin
with ListForm do
if ShowModal=mrOk then
begin
SetLength(a, ListBoxArray.Items.Count); {Установка длины массива}
for i:=0 to ListBoxArray.Items.Count-1 do
a[i]:=StrToFloat(ListBoxArray.Items[i]); {Копирование значений в массив}
try
s:=0;
for i:=0 to Length(a)-1 do
s:=s+a[i]; {Суммирование всех чисел}
LabelSum1.Caption:=FloatToStr(s);
except
LabelSum1.Caption := 'Ошибка вычисления суммы';
end;
try
p:=1;
for i:=0 to Length(a)-1 do
P:=P*a[i]; {Вычисление произведения всех чисел}
LabelProduct1.Caption:=FloatToStr(p);
except
LabelProduct1.Caption:='Ошибка вычисления произведения';
end;
```

```
end;
end;
procedure TMainForm.ButtonCloseClick(Sender: TObject);
begin
Close; {выход из программы}
end;
procedure TMainForm.ButtonGridClick (Sender: TObject);
var
a: array of array of double; {Объявление двумерного динамического массива}
x,y: integer; s, p: double;
begin
with GridForm do
if ShowModal=mrOk then
begin
SetLength(a, GridArray.ColCount-2); {Установка длины массива по столбцам}
for x:=0 to Length(a)-1 do
SetLength (a[x], GridArray.RowCount-2); {Установка длины массива по
рядам}
for x:=0 to Length (a)-1 do
for y:=0 to Length (a[x])-l do
a[x, y]:=StrToFloat (GridArray.Ce11s[x+1,y+1]); {Копирование значений}
try
s:=0;
for x:=0 to Lengeh(a)-1 do
for y:=0 to length (a[x])-1 do
s:=s+a [x, y]; {Суммирование всех чисел}
LabelSum2.Caption:=FloatToStr(s);
except
LabelSum2.Capticn:= 'Ошибка вычисления суммы';
end;
```

```
try
p:=1;
for x:=0 to Length(a)-1 do
for y:=0 to Length(a[x])-1 do
p:=p*a[x, y];
LabelProduct2.Caption:=FloatToStr(p);
except
LabelProduct2.Caption:= ‘Ошибка вычисления произведения’;
end;
end;
end;
end.
```

Текст вспомогательного модуля ListInput.pas.

```
unit ListInput;
interface
uses
Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
StdCtrls;
type
TListForm = class(TForm)
Label1: TLabel;
ListBoxArray: TListBox;
EditValue: TEdit;
Label2: TLabel;
ButtonAdd: TButton;
ButtonReplace: TButton;
ButtonOk: TButton;
ButtonCancel: TButton;
procedure ButronAddClick (Sender: TObject);
```

```
procedure LisiBoxArrayClick(Sender: TObject);
end;
var
ListForm: TListForm;
implementation
{$R *.DFM}
procedure TListForm.ButtonAddClick(Sender:TObject);
var
V: Double; S: string;
begin { Нажаты "Добавить" или "Заменить"}
EditValue.SetFocus;  {Сделать строку ввода активной}
EditValue.SelectAll;  {выделить все символы в строке}
try
V:=StrToFloat (EditVaiue.Text);
except
ShowMessage ('Неверное вещественное значение');
exit;
end;
S:=FloatToStr(V);
if (Sender=ButtonReplace) and (ListBoxArray.ItemIndex>=0) then {Проверка
нажатой кнопки}
ListBoxArray.Items[ListBoxArray.Itemlndex]:=S; {Нажата кнопка 'Заменить'}
else
begin
ListBoxArray.Items.Add(S); {Нажата кнопка "Добавить"}
ListBoxArray.ItemIndex:=ListBoxArray.Items.Count-1;
end;
end;
procedure TListForm.ListBoxArrayClick (Sender: TObject);
begin
```

```
if ListBoxArray.ItemIndex>=0 then
EditValue.Text:=ListBoxArray.Items[ListBoxArray.ItemIndex];
end;
end.
```

Текст вспомогательного модуля GridInput.pas.

```
unit Gridlnput;
interface
uses
Windows, Messages, SysUtils, Classes, Graphics, Controls, Fonts, Dialogs,
Grids, StdCtrls, ExtCtrls;
type
TGridForm = class(TForm)
GridArray: TStringGrid;
ButtonOk: TButton;
ButtonCancel: TButton;
PanelStatus: TPanel;
Label1: TLabel;
procedure FornCreate(Sender: TObject);
procedure GridArraySetEditText (Sender: TObject; ACol, ARow: Integer; const
Value: String);
procedure ButtonCkCiick(Sender: TObject};
procedure GridArray'KeyDown(Sender: TObject; var Kev: :Word; Shift:
TShiftState);
private
procedure UpdateGrid; {Рассчитать количество строк в таблице}
end;
var
GridForm: TGridForm;
implementation
```

```
{$R *.DFM}
procedure TGridForm.FormCreate (Sender: TObject);
begin
with GridArray do
begin
ColWidths[0]:=35; {Установка широты первой колонки }
Cells[0,1]:='1'; {Занесение значений в ячейки по умолчанию}
Cells[0,2]:='2';
Cells[1,0]:='1';
Cells[2,0]:='2';
Cells[1,1]:='0';
end;
end;
procedure TGridForm.GridArraySetEditText (Sender: TObject; ACol,
ARow: Integer; const Value: String); {Параметры ACol и ARow задают номер
измененной ячейки, а Value - новее значение}
begin
with GridArray do
if Value='' then
UpdateGrid
else
begin
if ARow=RowCount-l then  {Добавить новый ряд }
begin
RowCount:=ARow+2;
Cells[0,ARow+l]:=IntToStr(ARow+l);
end;
if ACol=ColCount-l then  {Добавить новую колонку}
begin
ColCount:=ACol+2;
```

```
Cells[ACol+1,0]:=IntToStr(ACol+1);
end;
try
StrToFloat(Value);
except
PanelStatus.Caption: =' Неверное вещественное значение';
exit;
end;
end;
PanelStatus:='';
end;
procedure TGridForm.UpdateGrid;
var x,y,maxx,maxy: Integer;
begin
with GridArray do
maxx:=1; maxy:=1;  {Вычисляем максимальный номер заполненной строки}
for x:=1 to ColCount-1 do
for y:=1 to RowCount-1 do
if Cells[x,y] <> then
begin
if x>maxx then maxx:=x;
if y>maxy then maxy:=y;
end;
ColCount:=maxx+2;
RowCount:=maxy+2;
end;
end;
procedure TGridFormGridArrayKeyDown(Sender: TOblect; var Key: Word; Shift:
TShiftState); {Нажата клавиша на клавиатуре, когда фокус ввода находится
в таблице}
```

```
begin  if Key=VK_DELETE then
with GridArray do
if not EditorMode and (( Col<>1) or (Rov<>l)) then
begin
Cells[Col,Row]:='';
UpdateGrid;
end;
end;
procedure TGridForm.ButtonOkClick (Sender: TObject);
var x,y: integer;
begin {Перед закрытием формы необходимо проверить правильность
введенных значений}
with GridArray do
for y:=l to RowCount-2 do
for x:=1 to ColCount-2 do
if Cells[x, y] = '' then
begin
Col:=x; Row:=y;   {Сделать незаполненную ячейку текущей}
SetFocus;  {Сделать таблицу активным элементом в форме}
EditorMode :=True;  {Включить режим редактирования значений ячейки}
PanelStatus.Caption: = ‘Значение не введено’;
exit;
end
else try
StrToFloat (Cells [x,y]) ;
except
Col:=x; Row:=y;
SetFocus;
EditorMode:=True;
PanelStatus.Caption: = ‘Неверное вещественное значение’;
```

end;

ModalResuit:=mrOk; *{Форму надо закрыть, т.к. сшибок не найдено}*

end;

end.

2.2.3. Задачи для самостоятельной работы

1. «Если звезды зажигают…»

Предварительно разработать интерфейс проекта, подобрать готовые рисунки звезд или нарисовать их в графическом редакторе, использовать свойство видимости объектов, установленных на форме, разместить на форме отрывок из стихотворения Владимира Маяковского «Послушайте!»:

Послушайте!

Ведь, если звезды зажигают-

значит - это кому-нибудь нужно?

Значит – это необходимо,

чтобы каждый вечер

над крышами

загоралась хоть одна звезда?!

После запуска приложения на форме, цвет который похож на цвет ночного неба, должны быть отображены три метки: звезды (по щелчку на этой метке появляются звезды разной величины), стихотворение (по щелчку на этой метке выводятся стихотворные строки), выход (завершение работы приложения). Метки организовать в виде гиперссылок.

Программный код

unit Unitl;

interface

uses

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

type

```
    TForml = class(TForm)
    Label1:TLabel;
    Label2:TLabel;
    Label3:TLabel;
    Label4:TLabel;
    Label5:TLabel;
    Label6:TLabel;
    Label7:TLabel;
    Label8:TLabel;
    Label9:TLabel;
    Image1:TImage;
procedure LabellMouseMove (Sender:TObject; Shift:TShiftState; X,Y:Integer);
    procedure LabellMouseLeave (Sender: TObject);
    procedure LabellClick(Sender: TObject);
procedure Label2MouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
    procedure Label2MoueeLeave(Sender: TObject);
procedure Label3MouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
    procedure Label3MouseLeave(Sender: TObject);
    procedure Label3Click(Sender: TObject);
    procedure Label2Click(Sender: TObject);
    private
    {Private declarations}
    public
    {Public declarations}
    end;
    var
    Forml: TForml;
    implementation
    {$R *.dfm}
    procedure TForml.LabellMouseMove(Sender: TObject; Shift: TshiftState);
```

```
X, Y: Integer;
begin
labell.Color:=clYellow;
labell.Font.Color:=clRed;
end;
procedure TForml.LabellHouseLeave(Sender: TObject);
begin
labell.Color:=clBlack; labell.Font.Color:=clMenuBar;
end;
procedure TForml.LabellClick(Sender: TObject);
begin
Label4.Caption: = ‘Послушайте! Ведь, если звезды зажигают – ’;
Label5.Caption: = ‘значит – это кому-нибудь нужно?’;
Label6.Caption:= ‘значит – это необходимо, ’;
labeL7.Caption:= ‘чтобы каждый вечер’;
Label8.Caption:= ‘над крышами’;
Label9.Caption:= ‘загоралась хоть одна звезда?!’;
End;
procedure TForml.Label2MouseMove(Sender: TObject; Shift: TshiftState);
X, Y: Integer;
begin
label2.Color:=clYellow;
label2.Font.Color:=clRed;
end;
procedure TForml.Label2MoueeLeave(Sender: TObject);
begin
label2.Color:=clBlack;
label2.Font.Color:=clMenuBar;
end;
procedure TForml.Label3MouseMove(Sender: TObject; Shift: TShiftState);
```

```
X, Y: Integer;
begin
label3.Color:=clYellow; label3.Font.Color:=clRed;
end;
procedure TForml.Label3MouseLeave(Sender: TObject);
begin
label3.Color:=clBlack; label3.Font.Color:=clMenuBar;
end;
procedure TForml.Label3Click(Sender: TObject);
begin
Form1.Close;
end;
procedure TForml.Label2Click(Sender: TObject);
begin
Image1.Picture.LoadFromFile('star.bmp');
end;
end.
```

2. Создать Windows-приложение для разрешения одной из проблем – вычисления площади всех стен комнаты, имеющей форму прямоугольного параллелепипеда.

Код процедуры обработки события кнопки Button 1.

```
procedure TForml.ButtonlClick(Sender: TObject); var
a, b, h, в: real; begin
a:=StrToFloat(Editl.Text);  bs=StrToFloat(Edit2.Text);
h:=StrToFloat (Edit3 .Text);
a: =2*(a+b)*h;
Edit4.Text:=FloatToStr(в);
end;
end.
```

Вставка картинки производится путем загрузки заранее приготовление го файла в графическом редакторе Paint.

3. Разработать проект, в котором на форме после щелчка пользователем на соответствующей командной кнопке будет распечатываться текст стихотворения А.С. Пушкина «Талисман» (константа типа String).

Программный код.

```
unit Stich;
Interface
uses
Windows, Messages, SysOtils, Variants, Classes, Graphics, Controls, Forms, Dialogs, jpeg, ExtCtrls, StdCtrls, Buttons;
type
TForml = class(TForm)
BitBtnl: TBitBtn;
Button1: TButton;
Image1: TImage;
Image2: TImage;
Image3: TImage;
Label1: TLabel;
Label2: TLabel;
Label3: TLabel;
Label4: TLabel;
Label5: TLabel;
Label6: TLabel;
Label7: TLabel;
Label8: TLabel;
Label9: TLabel;
Label10: TLabel;
Label11: TLabel;
```

```
Label12: TLabel;
Label13: TLabel;
Label14: TLabel;
Label15: TLabel;
Label16: TLabel;
Label17: TLabel;
Label18: TLabel;
Label19: TLabel;
Label20: TLabel;
procedure ButtonlClick(Sender: TObject);
private
{Private declarations}
public
{ Public declarations }
end;
var
Forml: TForml;
implementation
{$R *.dfm}
Procedure TForml.ButtonlClick(Sender: TObject);
const
str: string= ‘Храни меня мой талисман,’;
begin
Labell.Visible:=true;
Labell.Caption:=str;
Label2.Visible:=true;
Label2.Caption:= ‘Храни меня во дни гоненья,’
Label3.Visible:=true;
Label3.Caption:= ‘В дни раскаянья, волненья:’;
Label4.Visible:=true;
```

Label4.Caption: = ‘Ты в день печали был мне дан’;

Label5.Visible:=true;

Label6.Visible:=true;

Label6.Caption:= ‘Вокруг меня валы ревучи,’

Label7.Visible:=true;

Label7.Caption:= ‘Когда грозою грянут тучи -’ ;

Label8.Visible:=true;

Label8.Caption:=str;

Label9.Visible:=true;

Label9.Caption:= ‘В уединенье чуждых стран,’

Labell0.Visible:=true,

Labell0.Caption:= ‘На лоне скучного покоя,’

Labelll.Visible:=true;

Labelll.Caption:= ‘В тревоге пламенного боя,’

Labell2.Visible:=true;

Label12.Caption:=str;

Labell3.Visible:=true;

Labell3.Caption: = ‘Священный сладостный обман,’

Labell4.Visible:=true;

Labell4.Caption:= ‘Души волшебное святило…’

Labell5.Visible:=true;

Labell5.Caption:= ‘Оно сокрылось, изменило…’

Labell6.ViBible:=true;

Labell6.Caption:=str;

Labell7.Visible:=true;

Labell7.Caption:= ‘Пускай же в век сердечных ран’;

Label18.Visible:=true;

Label18.Caption:= ‘Не растравит воспоминанье.’

Labell9.Visible:=true;

Label19.Caption: = ‘Прощай, надежда; спи, желанье;’

```
Label20.Visible:=true;
Label20.Caption:=str;
End; end.
```

4. Поместить на форму метку и две кнопки — *Ввод данных* и *Close*. При щелчке на кнопке *Ввод данных* поочередно появляются четыре окна ввода. Пользователь должен ввести фамилию, имя, отчество и возраст. По окончании ввода анкетные данные выводятся на метку.

Программный код

```
Unit An;
interface
uses
Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms, Dialogs, Buttons, StdCtrls;
type
TForml = class(TForm)
Buttonl: TButton;
BitBtnl: TBitBtn;
Labell: TLabel;
procedure ButtonlClick(Sender: TObject);
private
{Private declarations }
public
{ Public declarations }
end;
var
Forml: TForml;
implementation
{$R *.dfm}
procedure TForml.ButtonlClick (Sender: TObject);
```

```
var
f, i, o, v: string;
begin
f:=InputBox (‘Ввод фамилии’, ’Введите фамилию’, ’’);
i:= InputBox (‘Ввод имени’,  ‘Введите имя’, ’’);
o:=InputBox (‘Ввод отчества’,  ‘Введите отчество’, ‘’);
v:= InputBox (‘Ввод возраста’,  ‘Сколько вам лет’, ‘’);
Label1.Caption:=f+#13+i+#13+o ’,’ +#13+’Вам ‘+v+’ лет’;
End;
End.
```

5. Написать программу, на поверхности, формы которой отображаются текущее время и текущая дата. На форме поместить таймер и метки.

Программный код.

```
unit Unitl;
interface
uses
Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms, Dialogs, StdCtrls, ExtCtrls;
Type
TForml = class(TForm)
Timer1: TTimer;
Labell: TLabel;
Label2: TLabel;
Label3: TLabel;
procedure FormCreate(Sender: TObject);
procedure TimerlTimer(Sender: TObject);
private
{ Private declarations }
public
```

```
{ Public declarations }
end;
var
Forml: TForml;
Implementation
{$R *.dfm}
procedure TForml.FormCreate(Sender: TObject);
begin
Timer1.Interval:=1000 ;
Timer1.Enabled:=true;
Label3.Caption:=FormatDateTime( ‘dd.mm.yy’, now( ));
end;
procedure TForml.TimerlTimer(Sender: TObject);
begin
Labell.Caption:='';
Labell.Caption:=FormatDateTime( ‘hh:mm:se’, now( ));
end;
end.
```

2.2.4. Контрольные задачи

1. В проекте установить на форму компонент Shape (например, эллипс). Компонент Shape размещен на вкладке палитры компонентов Additional, установить 4 кнопки: Вверх, Вниз, Вправо, Влево. Над ними установить метку Перемещение эллипса. В проекте установить над эллипсом еще один компонент — Shape2. Первоначально это будет прямоугольник (кирпич), в дальнейшем он будет изменять свою форму, а цвет его будет изменяться случайным образом. Установить на форму кнопку Удар, при щелчке на которой упадет кирпич на эллипс, а тот при этом сожмется в высоту и вытянется в длину. Далее кирпич должен отскочить назад, для этого на форму необходимо установить таймер Timer (вкладка палитры компонентов

System). Кирпич при отскоке назад поменяет цвет с помощью датчика случайных чисел. На форме установить компоненты RadioButton (вкладка палитры компонентов Standard), позволяющие выбирать фигуру падающего предмета.

2. Составить программу, выполняющую следующие действия. При щелчке мышью на кнопке Старт в первом окне ввода-вывода Edit выводится текущее время в момент щелчка на этой кнопке. Во втором окне выводится текущее время в момент щелчка на кнопке Стоп. При щелчке мышью на кнопке Время в третьем окне показывается, сколько времени прошло между щелчками на кнопках Старт и Стоп.

3. Создать программу, которая над двумя данными числами производит указанное арифметическое действие. Для указания знаков арифметических действий использовать компонент ComboBox. Компонент ComboBox (страница Standard) - комбинированный список выбора. Представляет собой комбинацию списка выбора и текстового редактора. С помощью списка выбора пользователь может выбрать один или несколько элементов выбора. Свойства:

Items (тип string) содержит набор строк, показываемых в компоненте;

Multiselect(тип boolean) разрешает, отменяет выбор нескольких элементов;

Itemindex(тип integer) содержит индекс сфокусированного элемента. Первый элемент имеет индекс 0. Если не выбрана ни одна строка

itemindex=-1.

На форме расположить компонент Panel, на панели расположить две строки ввода (Edit), компонент Combobox и две командные кнопки Ok и Close.

Над панелью расположить компонент Label и многострочный редактор Memo.

Itemindex(тип integer) содержит индекс сфокусированного элемента. Первый элемент имеет индекс 0. Если не выбрана ни одна строка

itemindex=-1.

На форме расположить компонент Panel, на панели расположить две строки ввода (Edit), компонент Combobox и две командные кнопки Ok и Close.

Над панелью расположить компонент Label и многострочный редактор Memo.

4. Дан числовой массив. Элементы массива ввести на нулевой строке многострочного редактора Memo. При вводе элементы массива разделить одним или несколькими пробелами. Найти среднее геометрическое положительных элементов массива.

5. Дан числовой массив. Массив ввести с помощью компонента Memo по одному элементу на каждой строке, начиная с нулевой. Найти среднее арифметическое отрицательных элементов массива. Исходный массив и результат вывести на многострочный редактор Memo.

2.3 Задачи повышенной сложности

Задача 1. Создать проект «Проигрыватель midi-файлов», используя компонент OpenDialog (вкладка палитры компонентов Dialogs).

Программный код.

```
unit Prjmusic2;
interface
uses
Windows, Messages, SysOtils, Classes, Graphics, Controls, Forms, Dialogs, StdCtrls, MPlayer;
type
TForml = class(TForm)
Button2: TButton;
Buttonl: TButton;
MediaPlayerl: TMediaPlayer;
OpenDialogl: TOpenDialog;
Labell: TLabel;
procedure Button2Click(Sender: TObject);
procedure ButtonlClick(Sender: TObject);
private
{ Private declarations }
public
{ Public declarations }
End;
var
Forml: TForml;
implementation
{$R *.DFM}
procedure TForml.Button2Click(Sender: TObject);
begin
```

```
MediaPlayerl.Close;
Application.Terminate;
end;
procedure TForml.ButtonlClick (Sender: TObject);
begin
OpenDialogl.Filter: = ‘Filter mid /  *.midi’ ;
if OpenDialogl.Execute and FileExists (OpenDialogl.FileName) then begin
MediaPlayerl.FileName:=OpenDialogl.FileName; MediaPlayerl.Open;
MediaPlayerl.Play;
end;
end;
end.
```

2. Создайте приложение, реализующее ввод двух целых чисел, по щелчку на кнопке с символом «=» вычисляющее результат операции вещественного деления и выводящее значение результата на экран. В программе перед операцией деления должна выполняться проверка делителя на равенство нулю. В случае равенства делителя нулю вместо выполнения деления должно выводиться сообщение «На ноль делить нельзя!» в отдельном окне сообщения.

Программный код.

```
unit Dell;
interface
uses
Windows, Messages, SystJtils, Variants, Classes, Graphics, Controls, Forms, Dialogs, StdCtrls;
type
TForml = class(TForm)
Labell: TLabel;
```

```
Editl: TEdit;
Label2: TLabel;
Label3: TLabel;
Edit2: TEdit;
Label4: TLabel;
Edit3: TEdit;
Buttonl: TButton;
Button2: TButton;
procedure ButtonlClick(Sender: TObject);
procedure Button2Click(Sender: TObject);
private
{ Private declarations }
public
{ Public declarations }
end;
var
Forml: TForml;
implementation
{$R *.dfm}
procedure TForml.ButtonlClick(Sender: TObject);
var
a, b: integer; c: real; begin
a:=StrToInt(Editl.Text); b:=StrToInt(Edit2.Text) ;
if b=0 then ShowMessage( ‘На ноль делись нельзя!!!’)
else
begin
c:=a/b;
Edit3.Text:=FloatToStrF(c, ffGeneral, 7, 4);
End;
end;
```

```
procedure TForml.Button2Click(Sender: TObject);
begin
close;
end;
end.
```

3. Поместить на форму два поля ввода и кнопку со стрелкой «>>». При щелчке на кнопке текст из левого поля ввода переписывается в правое, при этом стрелка на кнопке изменяет свое направление. Если теперь щелкнуть на кнопке еще раз, то текст из правого поля перепишется в левое, а стрелка снова изменит свое направление.

Программный код.

```
unit Pole;
interface
uses
Windows, Messages, SysOtils, Variants, Classes, Graphics, Controls, Forms, Dialogs, Buttons, StdCtrls;
type
TForml = class(TForm)
Editl: TEdit;
Edit2: TEdit;
Buttonl: TButton;
BitBtnl: TBitBtn;
procedure ButtonlClick(Sender: TObject);
private
{ Private declarations }
public
{ Public declarations }
end;
var
```

```
Forml: TForml;
implementation
{$R *.dfm}
procedure TForml.ButtonlClick(Sender: TObject);
begin
If Buttonl.Caption=’>>’  then
begin
Edit2.Text:=Edit1.Text;
Editl.Text: =’’;
Buttonl.Caption: =’<< ‘ ;
end
else
begin
Editl.Text:=Edit2.Text;
Edit2.Text: = ‘’;
Buttonl.Caption:=’>>’;
end;
end;
end.
```

4. В однострочном редакторе пользователь выбирает одну из закладок. На рабочем поле блокнота на метку выводится соответствующая надпись: год, месяц, день календаря или сообщение о памятной дате.

Программный код.

```
unit Unitl;
interface
uees
Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
Dialoge, ComCtrls, StdCtrls;
type
```

```
TForml = class (TForm)
TabControll: TTabControl;
Label2:TLabel;     {Надпись «Дата»}
Label3:TLabel;     {Отображение текущей даты компьютера}
Labell:TLabel;      {Надпись блокнота}
procedure FormCreate (Sender: TObject); procedure TabControllChange (Sender: TObject);
private
{ Private declarations }
public
{ Public declarations }
end;
var
Forml: TForml;
y, m, d: word; { Глобальные переменные для хранения даты календаря}
implementation
{$R *.dfm}
procedure TForml.FormCreate(Sender: TObject);
begin
Label3.Caption:=dateTostr(date);
DecodeDate(date, y, m, d);
Labell.Caption:=IntToStr(y);
end;
procedure TForml.TabControllChange (Sender: TObject);
begin
case TabControll.Tablndex of
0: begin
Labell.Caption:= '';
Labell.Caption:=IntToStr (y);
end;
```

```
1: begin
Label1.Caption “^”; case m of
1: Label1.Caption:='Январь';
2: Label1.Caption:='Февраль';
……
12: Label1.Caption:='Декабрь';
end;
end;
2: begin
Label1.Caption: = ' ';
Label1.Caption:=IntToStr(d);
end;
3: begin
Label1.Caption:= ‘’;
Label1.Caption:='год 70-летия Победы';
end; end; end; end.
```

5. Составить программу, которая пересчитывает рубли по курсу ММВБ в денежные эквиваленты разных стран.

Название страны	Курс валюты
Англия	1 фунт-52,8994 руб.
Бельгия	1 евро- 36,5066 руб.
Германия	1 евро – 36,5066 руб.
Индия	1 рупия - 6,4244 руб.
Исландия	100 исландских крон - 45,1380 руб.
Испания	1 евро - 36,5066 руб.
Италия	1 евро -36,5066 руб.
Китай	10 юаней -33,9328 руб.

Литва	1 лит -10,5812 руб.
Монголия	1000 тугриков - 23,172 руб.
США	1 доллар - 27,9359 руб.
Франция	1 евро -36,5066 руб.
Швейцария	1 швейцарский франк - 23,5945 руб.
Япония	100 йен -26,5097 руб.

Код кнопки Вычислить.

```
procedure TForml.ButtonlClick(Sender: TObject);
var
rubl: real; { Сумма в рублях }
vis: real; { Эквивалент валюты }
k: real;          { Коэффициент пересчета }
nazvl: string[20];
begin
case ListBoxl.Itemlndex of
0: begin
к: =52.8994;
nazvl: = ‘фунтов’;
end;
1, 2, 5, 6, 11:
begin
к:=36.5066;
nazvl:= ‘евро’;
end;
3: begin
к: =6.4244;
nazvl:= ‘рупий’;
end;
4: begin
```

```
к:=0.45138;
nazvl:= ‘исландских крон’;
end;
7: begin
k:=3.39328;
nazvl:= ‘юаней’;
end;
8: begin
k:=10.5812;
nazvl: = ‘лит’ ;
end;
9: begin
k:=0.02317;
nazvl: = ‘тугриков’;
end;
10: begin
k:=27.9359;
nazvl:= ‘долларов’;
end;
12: begin
k: =23.5945;
nazvl: = ‘швейцарских франков’
end
else
 begin
k:=0.2651;
nazvl:= ‘йен’;
end;
end;
rubl:=StrToFloat (Editl.Text);
```

```
vl:=rubl/k;
Label4 Caption:=Editl.Text+' рублей-'+FloatToStrF(vl, ffFixed, 6, 3) + tnazvl;
end;
end;
end.
```

6. Приложение показывает фоновый рисунок цветущего луга, над которым летают две бабочки. На форме разместить три объекта Image: один объект будет фоном приложения, два других объекта будут служить для вывода из файлов рисунков бабочек. Следует обратить внимание на то, что файлы с рисунками бабочек должны иметь формат bmp и обязательно левый нижний пиксель должен быть белого цвета для задания прозрачного фона рисунка. Бабочки летят навстречу друг другу. Вначале размещенная в нижнем правом углу бабочка находится ближе к зрителю (на переднем плане), поэтому она крупнее. В верхнем левом углу бабочка расположена как бы на втором плане, размеры ее меньше.

Программный код.

```
unit Unitl;
interface
usee
Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms, Dialogs, ExtCtrls, jpeg;
type
TForml = class(TForm)
Imagel: TImage;
Timerli TTimer;
Image2: TImage;
Image3: TImage;
procedure TimerlTimer(Sender: TObject);
```

```
private
{ Private declarations }
public
{ Public declarations }
end;
var
Forml: TForml;
Implementation
{$R *.dfm}
Procedure TForml.TimerlTimer (Sender: TObject);
var
x2, y2, x3, y3: integer;
begin
Tag:=Tag+l;
if Tag mod 2<>0 then
begin
Image2.Picture.LoadFromFile (‘Бабочка1.bmp’);
Image3.Picture.LoadFromFile (‘Бабочка.bmp’);
Image2.Width:=Image2.Width-5;
Image2.Height:=Image2.Height-5;
Image3.Width:=Image3.Width+7;
Image3.Height:=Image3.Height+7;
end
else
begin
Image2.Picture.LoadPromFile (‘Бабочка1 правая.bmp’);
Image3.Picture.LoadFromFile (‘Бабочка левая.bmp’);
Image2.Width:=Image2.Width-5;
Image2.Height:=Xmage2.Height-5;
Image3.Width:=Image3.Width+7;
```

```
Image3. Height: =Image3. Height+7; end;
Image2.Transparent:= true;
Image3.Transparent:=true;
x2:=Image2.Left;
y2:=Image2.Top;
Image2.Iie£t:=x2-40;
Image2.Top:=y2-20;
x3:=Image3.Left;
y3:=Image3.Top;
Image3.Left:=x3+40;
Image3.Top:=y3+20;
if  (Image2.Left<Imagel.Ieft+5) or
((Image3.Left+Image3.Width)>Imagel.Width) then
Begin
Timerl.Enabled:=false;
Close;
End;
End;
End.
```

7. Доработать проект (6) таким образом ,чтобы летающие над ним две бабочки махали крыльями.

```
procedure TForml.TimerlTimer(Sender: TObject); var
x2, y2, x3, y3: integer; begin
Tag:=Tag+l; if Tag mod 2<>0 then begin
Image2.Picture.LoadFromFile (‘Бабочка1.bmp’);
Image3. Picture. LoadFromFile (‘Бабочка.bmp’);
End;
else
begin
```

Image2.Picture.LoadFromFile (‘Бабочка1 правая.bmp’);

Image3.Picture.LoadFromFile (‘Бабочка левая.bmp’);

end;

Image2.Transparent:=true;

Image3.Transparent:=true;

x2:=Image2.Left;

y2:=Image2.Top;

Image2.Lef t:=x2-10;

Image2.Top:=y2-5;

x3:=Image3.Left;

y3:=Image3.Top;

Image3.Left:=x3+40;

Image3.Top:=y3+20;

if (Image2.Leftxlmagel.Left+5) or ((Image3.Left+Image3.Width) > Imagel.Width) then

begin

Timerl.Enabled:=false;

close;

end; end; end.

8. На форме изобразить термометр со шкалой. При вводе значений температуры в окно ввода на температурной шкале должен перемещаться бегунок.

Программный код.

unit Onitl;

interface

uses

Windows, Messages, SysOtils, Variants, Classes, Graphics, Controls, Forms, Dialogs, StdCtrls, ExtCtrls, ComCtrls;

type

TForml = class(TForm)

```
Shape1: Tshape;
Shape2: TShape;
ScrollBarl: TScrollBar;
Labell: TLabel;
Label2: TLabel;
Label3: TLabel;
Buttonl: TButton;
Editl: TEdit;
Button2: TButton;
Label4: TLabel;
Label5: TLabel;
Label6: TLabel;
Label7: TLabel;
Label8: TLabel;
Label9: Tlabel;
procedure Button2Click(Sender: TObject);
procedure ButtonlClick(Sender: TObject);
procedure FormActivate(Sender: TObject);
private
{ Private declarations }
public
{ Public declarations }
end;
var
Forml: TForml;
implementation
{$R *.dfm}
procedure TForml.Button2Click(Sender: TObject);
begin
close;
```

```
end;
procedure TForml.ButtonlClick(Sender: TObject);
var
t: integer;
begin
t:=StrToInt(Edit1.Text);
ScrollBarl.Position:=-t;
Editl.SetFocus;
end;
procedure TForml.FormActivate(Sender: TObject);
begin
Editl.SetFocus;
end; end.
```

9. Поместить на форму метку и две кнопки – ВВОД ДАННЫХ и CLOSE. При щелчке на кнопке ВВОД ДАННЫХ поочередно появляются четыре окна ввода. Пользователь должен ввести фамилию, имя, отчество и возраст. По окончании ввода анкетные данные выводятся на метку.

Программный код.

```
unit An;
interface
uses
Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms, Dialogs, Buttons, StdCtrls;
type
TForml = class(TForm)
Buttonl: TButton;
BitBtnl: TBitBtn;
Labell: TLabel;
procedure ButtonlClick(Sender: TObject);
private
```

```
{ Private declarations }
public
{ Public declarations }
End;
var
Forml: TForml;
implementation
{$R *.dfm}
procedure TForml.ButtonlClick(Sender: TObject);
var
f, i, o, V: string;
begin
f:=InputBox (‘Ввод фамилии’, ‘Введите фамилию’, ‘’);
i:=XnputBox ( ‘Ввод имени’, ‘Введите имя’, ‘’);
o:=InputBox (‘Ввод отчества’, ‘Введите отчество’, ‘’);
v:=XnputBox (‘Ввод возраста’, ‘Введите возраст’, ‘’);
Labell.Caption:=f+#13+i+#13+o+','+ #13+'Вам '+v+' лет' ;
end; end.
```

10. Внести изменения в предыдущий проект (9), чтобы каждое введенное в поле ввода значение выводилось на следующих формах с помощью InputBox.

Программный код.

```
unit Uaitl;
interface
uses
interface
uses
Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms, Dialogs, StdCtrls;
type
```

```
TForml = class(TForm)
BitBtn1:TBitBtn;
BitBtn2:TBitBtn;
Label1:TLabel;
Label2:TLabel;
procedure BitBtnlClick(Sender: TObject);
private
{ Private declarations }
public
{ Public declarations }
end;
var
Form1:TForm1;
Implementation
{$R *.dfm}
Var
Sf, si, so, sg: string;
procedure BitBtnlClick(Sender: TObject);
begin
sf:=InputBox (‘Ввод фамилии’, ‘Введите фамилию’, ‘’);
si:=XnputBox Ввод имени’,’Господин, ‘+sf+’ ‘Введите  свое имя’, ‘’);
so:=InputBox (‘Ввод отчества’, ‘Уважаемый, ‘si+’ ‘ +sf+’, ‘Введите отчество’, ‘’);
sg:=XnputBox (‘Ввод возраста’, ‘Уважаемый, ‘si+’ ‘ +so+’ ‘+sf+,’,’#13+’ ‘сколько Вам лет?’, ‘’);
Label1.Caption:=’Доброе время суток Вам, ‘+#13+sf+’ ‘ +si+’ ‘+so+’,’;
Label2.Caption:=’сегодня Вам - ’+sg+’ лет’;
End;
End.
```

2.3.1. Контрольные задачи

1. Человеку нужно съездить из Лондона в расположенный в 390 милях Эдинбург и вернуться назад. Он может съездить на автомобиле марки «Ролс-Ройс» либо на автомобиле марки «Форд Эскорт». «Ролс-Ройс» расходует 1 галлон бензина на каждые 15 миль пути. «Форд Эскорт» расходует 1 галлон на каждые 36 миль пути. Сколько будет стоить поездка в Эдинбург на «Ролс- Ройсе», если стоимость одного галлона бензина составляет *x* фунтов? Сколько денег человек сбережет, если вместо этого поедет на машине «Форд Эскорт»?

2. Реактивный аэробус летит со 100 пассажирами на борту из Лондона в Нью-Йорк. Три четверти пассажиров имеют билеты второго класса стоимостью *x* фунтов каждый. Остальные пассажиры имеют билеты первого класса, которые стоят вдвое дороже билетов второго класса. Напишите программу, которая выведет сумму денег, получаемую авиакомпанией от продажи билетов на этот рейс.

3. Программа расчета платы за электроэнергию требует ввода старого и нового значений показаний счетчика. Стоимость 1 квт/ч — 1 руб. 30 коп. На экран вывести сумму к оплате.

4. Вычислить площадь кольца (r1 – радиус внешней окружности, r2 – радиус внутренней окружности).

5. Составить программу перевода градусов Цельсия в градусы Фаренгейта по формуле f=9/5 *c+32.

6. Написать программу вычисления стоимости поездки на автомобиле на дачу (туда и обратно). Исходными данными являются расстояние до дачи (в км), количества бензина, которое потребляет автомобиль на 100 км пробега, цена одного литра бензина.

7. Программист Сидор Пентюхов пишет девушке, с которой познакомился в чате, 2 письма в день объемом по 4 Кб каждое, а «юзер» Вася Чайников – 5 писем по 2 Кб. Каков будет их суммарный трафик к тому

моменту, когда через n дней они обнаружат, что переписываются друг с другом? Проект сохранить в папке «Трафик».

8. На телеге покосилось колесо. Опытный кучер определил, что колесо может еще совершить x оборотов, пока не развалится. Определить, доедет ли кучер до своего поместья, если до него n км, а радиус колеса равен r см.

9. Вывести на экран номер четверти координатной плоскости, которой принадлежит точка с координатами (x,y), при условии, что x< >0, y< >0.

10. Написать программу вычисления стоимости покупки с учетом скидки. Скидка в 3% предоставляется в том случае, если стоимость покупки более 500 рублей, в 5% - если сумма больше 1000 рублей, в 10% - если сумма более 10000 рублей.

ЗАКЛЮЧЕНИЕ

В основе школьного курса информатики изучают язык программирования - Delphi. Для эффективного усвоения курса, было разработано данное методическое пособие, в котором:

- подробно описаны компоненты окна программной среды Delphi;
- рассмотрены простейшие примеры использования существующих инструментов;
- приведены примеры решения практических заданий с подробным пошаговым описанием действий на примерах задач из школьного курса информатики и задач повышенной сложности;
- приведен ряд практических заданий для самостоятельного решения;
- Разработано учебное пособие по теме «Визуальное программирование в Delphi».

Данная методическая разработка была апробирована в ходе прохождения педагогической практики и может быть использована в учебном процессе.

СПИСОК ЛИТЕРАТУРЫ

1. Архангельский А.Я. – Программирование в Delphi 7: М.: ООО «Бином-Пресс», 2003 г. — 1152 с.
2. Гайнанова Р.Ш. - Дополнительные главы программирования в Delphi: учебно-методическое пособие. – Казань: КФУ, 2012. – 74с.
3. Давыдова Н.А., Боровская Е.В. – Программирование: учебное пособие. Издатель: БИНОМ. Лаб. знаний, Москва, 2012.
4. Культин Н.Б. – Основы программирования в Delphi 7. – СПб.: БХВ - Петербург, 2004. – 608 с.
5. Культин Н.Б. –Delphi 6. Программирование на Object Pascal. – СПб.: БХВ - Петербург, 2004. – 528 с.
6. Культин Н.Б. –Delphi в задачах и примерах. – СПб.: БХВ - Петербург, 2012. – 288 с.
7. Мельников С.В. – Delphi и Turbo Pascal на занимательных примерах. : БХВ - Петербург, 2012.
8. Павлова И.М., Власова Н.Г. – Информатика в школе: Приложение к журналу «Информатика и образование». №6- 2006. – М.: Образование и Информатика, 2006. – 96 с.
9. Скворцов А.В., Поддубная Т.Н. – Введение в программирование в среде Delphi (лабораторный практикум). – Томск: Изд-во Том. Ун-та, 2000. – 112 с.
10. Фаронов В.В – Delphi 6: учебный курс. СПб.: Питер,2002, 512 с.
11. Шпак Ю. А. - Delphi 7 на примерах/Под ред. Ю. С. Ковтанюка — К.: Издательство Юниор, 2003. — 384 с.

Интернет источники:

1. http://ru.wikipedia.org/wiki
2. http://vit-prog.narod.ru
3. http://www.snkey.net/books/delphi/ch1_2.html