Государственное бюджетное профессиональное образовательное учреждение «Арзамасский коммерческо-технический техникум»

**ПРОГРАММИРОВАНИЕ В СРЕДЕ DELPHI**

ЧАСТЬ I

ОСНОВНЫЕ КОМАНДЫ

ПЕРВЫЕ ПРОЕКТЫ

**Саблукова Наталья Геннадьевна**

УДК 004 (075.4)

ББК 74.261.63

В85

|  |  |
| --- | --- |
|  | Наталья Геннадьевна Саблукова |
| В85 | Программирование в среде Delphi. Ч.1. Основные команды. Первые проекты. Арзамас: Изд-во АГПИ, 2017. – 115 с. |

Учебное пособие включает в себя теоретический материал, разобранные примеры и задания по одной из наиболее популярных в настоящее время систем программирования Delphi. Весь теоретический материал и практические задания ориентированы на разработку сначала простых программных проектов, а затем все более и более сложных.

Пособие может быть также использовано для самостоятельного изучения системы программирования Delphi.

© Саблукова Н.Г., 2017

УДК 004 (075.4)

ББК 74.261.6

**Содержание**

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | Введение. | 5 |
| 1 |  | Основные этапы разработки проектов. | 6 |
| 2 |  | Среда программирования Delphi. | 8 |
|  | 2.1 | Главное окно программы. | 9 |
|  | 2.2 | Окно формы. | 10 |
|  | 2.3 | Окно Инспектора объектов. | 11 |
|  | 2.4 | Окно программного кода. | 11 |
|  | 2.5 | Структура проекта Delphi. | 12 |
|  | 2.6 | Основные понятия визуального программирования. | 13 |
| 3 |  | Проектирование графического интерфейса. | 15 |
|  | 3.1 | Форма и ее основные свойства. | 15 |
|  | 3.2 | Компоненты TEdit, TLabel, TButton. | 21 |
|  | 3.3 | Графическое оформление окон. | 24 |
|  | 3.4 | Реакция на события. | 28 |
|  | 3.5 | Изменение свойств компонентов через программный код | 30 |
| 4 |  | [Элементы языка программирования Delphi](#элементы_языка). | 33 |
|  | 4.1 | [Алфавит](#алфавит). | 33 |
|  | 4.2 | [Зарезервированные слова и идентификаторы](#зарезервированные_слова). | 34 |
|  | 4.3 | [Типы данных](#типы_данных). | 34 |
|  | 4.4 | [Константы](#константы). | 36 |
|  | 4.5 | Структура программного модуля. | 36 |
| 5 |  | [Основные операторы языка программирования Delphi](#основные_операторы). | 38 |
|  | 5.1 | [Оператор присваивания](#оператор_присваивания). | 38 |
|  | 5.2 | Логический тип данных. Создание проектов с использованием оператора присваивания и логических типов данных. | 41 |
|  | 5.3 | Математические операции Delphi. | 44 |
|  | 5.4 | Линейная алгоритмическая структура. Ввод и вывод информации в Delphi. | 46 |
|  | 5.5 | Создание проектов с использованием ввода/вывода информации. | 51 |
|  | 5.6 | Алгоритмическая структура «Ветвление». Условный оператор. | 62 |
|  | 5.7 | Создание проектов с использованием условного оператора и генератора случайных чисел. | 66 |
|  | 5.8 | Оператор выбора. | 71 |
|  | 5.9 | Создание проекта «Калькулятор» с использованием условного оператора и оператора выбора. | 79 |
|  | 5.10 | Реализация условного оператора и оператора выбора компонентами TRadioButton и TRadioGroup. | 79 |
|  | 5.11 | Реализация условного оператора и оператора выбора компонентом TCheckBox. | 84 |
|  | 5.12 | Создание проекта «Тест» с использованием условного оператора и переключателей. | 87 |
|  | 5.13 | Алгоритмическая структура «Цикл». Операторы повторений. | 93 |
|  | 5.14 | Компонент TTimer. | 100 |
|  | 5.15 | Создание проектов с использованием таймера. | 102 |
|  |  | Литература. | 110 |
|  |  | Приложения. |  |

**Введение**

Обучение школьников программированию на протяжении всего периода становления курса информатики являлось едва ли не основной задачей. Дело в том, что линия алгоритмизации и программирования всегда доминировала в школьном курсе информатики и лишь в последнее время уступила первенство линии информационных технологий. Но проблема подготовки школьников по программированию существовала всегда. Связано это с тем, что данный раздел информатики является самым сложным для восприятия школьников. Существует, по крайней мере, пять языков программирования, на которых традиционно ведется обучение. Кроме того, материал этого раздела достаточно объемный и на него часто не хватает учебного времени. Поэтому у авторов возникла идея вынести обучение линии алгоритмизации и программирования за рамки школьного курса информатики и ИКТ в область дополнительного образования. В городе Арзамасе была создана Школа программистов, в которой на протяжении более, чем десяти лет, шла апробация учебных программ и учебных пособий. Результатом этой работы и является данное учебное пособие.

Выбор в качестве базового языка программирования пал на Delphi не случайно. Delphi – это современная технология визуального программирования, которая может сделать процесс создания программ наглядным и увлекательным. Она прячет все трудности программирования и берет на себя большую часть рутинной работы. Но, вместе с этим, с помощью Delphi можно создавать достаточно сложные и профессиональные приложения.

Данное учебное пособие написано для школьников, которые до этого не умели программировать. Поэтому в теоретической части пособия подробно описываются основные объекты Delphi и алгоритмические конструкции. Чтобы Вы лучше усвоили материал, в каждой теме разбираются примеры проектов. Но если Вы хотите по-настоящему научиться программировать, нужно не только последовательно пройти весь учебный материал, но и выполнить самостоятельные задания и проекты. Для получения максимальной пользы от книги Вы должны работать с ней активно. Не занимайтесь просто чтением примеров, реализуйте их с помощью вашего компьютера. Не бойтесь экспериментировать – вносите изменения в программы. Чем больше выполните заданий, тем лучше научитесь программировать в современной визуальной среде Delphi.

Желаем Вам успехов в освоении одного из наиболее важных и интересных разделов современной информатики.

**§1 Основные этапы разработки проектов**

Современное визуальное программирование в среде Delphi еще называют визуальным конструированием программ. Главное отличие этого вида программирования состоит в том, что здесь программа не пишется в виде текста, а собирается из отдельных кусочков (компонентов), которые уже были созданы разработчиками языка программирования. Создание внешней видимой части программы чем-то напоминает конструирование, например в Lego. Написание кода программы нужно, в основном, для того, чтобы заставить эти компоненты реагировать на какие-то управляющие воздействия, для ввода, обработки и вывода данных, а также для задания параметров компонентов. Собственно, благодаря этому данный подход в программировании и получил название визуального программирования.

Среда программирования Delphi является профессиональной средой программирования и предназначена для разработки достаточно сложных и производительных программных комплексов, получивших название проектов. Создание проектов происходит в несколько этапов:

1. **Постановка задачи.** На этом этапе сначала анализируется, затем подробно описывается, что должен представлять собой проект. Здесь следует определить, как будут введены исходные данные, и в каком виде должен получиться результат. На первом этапе также выделяются компоненты, которые будут располагаться на форме проекта. Точность и четкость в постановке задачи – это половина дела.
2. **Разработка графического интерфейса проекта.** На этом этапе создается окно будущего проекта. Для этого на форме размещаются нужные компоненты и устанавливаются их свойства.
3. **Разработка алгоритма**.Здесь описывается последовательность действий, которая приведет к нужному результату, определяется, какие события будут происходить в проекте и как будут связаны между собой отдельные компоненты.
4. **Создание программного кода**.Это и есть собственно написание программы на языке Delphi.
5. **Отладка и тестирование программы**.Отладка программы – это устранение ошибок и неточностей, которые были допущены на предыдущих этапах. Как правило, при создании проекта именно отладка программы требует львиную долю времени. При тестировании выясняется, выполняет ли программа то, что от нее требуется и не выполняет ли то, что не нужно.
6. **Сохранение проекта и компиляция проекта.** На этом этапе проект превращается в исполняемое приложение. Такое приложение способно работать самостоятельно за пределами среды проектирования.

Одним из важнейших этапов создания проекта является этап разработки алгоритма. Если на этом этапе допущены ошибки, то устранить их далее достаточно сложно. Начиная с 50-х годов для наглядного изображения алгоритмов, программисты стали использовать графические схемы, которые получили название блок-схем. Существует несколько видов блок-схем, наибольшее распространение получили блок-схемы и структурограммы Насси-Шнейдермана.

Блок-схемы строятся по определенным правилам и содержат геометрические фигуры (блочные символы), соединенные между собой стрелками (линиями). Стрелки показывают порядок выполнения операций.

Наиболее часто используемые блочные символы приведены в таблице 1.

**Таблица 1. Основные блочные символы**

|  |  |  |
| --- | --- | --- |
| **Наименование** | **Обозначение** | **Функции** |
| Процесс |  | Выполнение операции или группы операций, в результате которых изменяются значения данных или их расположение. |
| Решение |  | Выбор направления выполнения алгоритма в зависимости от условия. |
| Предопределенный процесс (подпрограмма) |  | Использование ранее созданных и отдельно описанных алгоритмов. |
| Ввод-вывод |  | Преобразование данных в форму, пригодную для обработки (ввод) или отображение результатов обработки (вывод) |
| Модификация |  | Выполнение операций, меняющих команды, или группы команд, меняющих программу. |
| Пуск-остановка |  | Начало, конец, прерывание процесса обработки данных или выполнения программы. |
| Комментарий | 7 | Связь между элементами схемы и пояснением. |
| Внутристраничный соединитель |  | Указание на наличие связки между прерванными линиями алгоритма обработки данных. |
| Межстраничный соединитель |  | Указание на наличие связи между разъединенными частями схем, расположенными на разных страницах. |

Например, блок-схема алгоритма вычисления суммы двух чисел выглядит следующим образом:

Ввод a, b

S:=a+b

Вывод S

Рис. 1. Блок-схема алгоритма, вычисляющего сумму двух чисел.

Изучение Delphi начнем с рассмотрения её среды программирования.

**§2** **Среда программирования Delphi**

После вызова программы командой [Пуск - Программы - Borland Delphi – Delphi] экран ПК приобретает вид, показанный на рис. 2. При работе в Delphi Вы будете постоянно обращаться к четырем основным окнам:

1) **Главное окно программы.** На нём находятся основное меню, панели инструментов и палитра компонентов.

2) **Инспектор объектов.** Он предназначен для управления объектами и их свойствами.

3) **Окно формы.** Это готовая визуальная форма будущей программы.

4) **Окно программного кода**. В этом окне записывается сама программа.

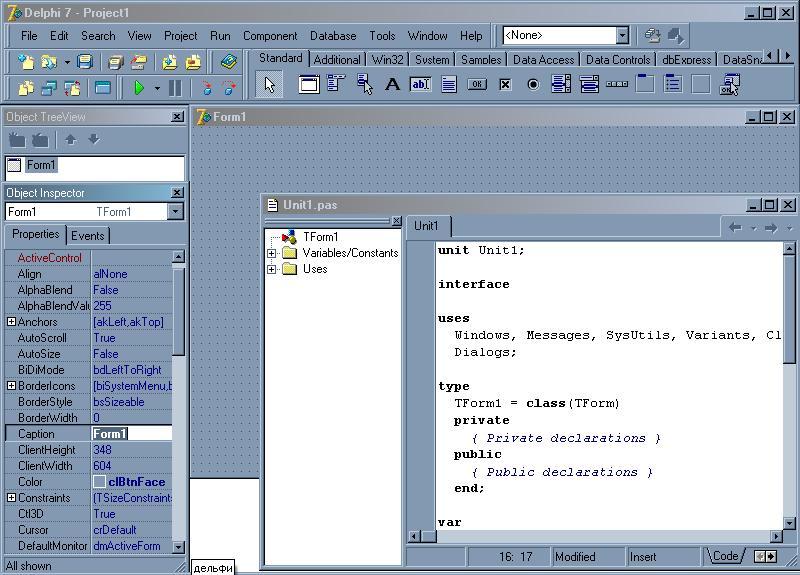
Главное окно

Инспектор объектов

Окно формы

Окно программного кода

Проводник



.

Рис. 2. Главное окно Delphi.

**2.1** **Главное окно программы**

**Главное окно** (рис. 3) выполняет основные функции управления проектом. Оно постоянно присутствует на экране и занимает его самую верхнюю часть.

В главном окне располагается главное меню Delphi, набор пиктографических командных кнопок и палитра компонентов.

Главное меню

Пиктографические кнопки

Палитра компонентов



Рис. 3. Главное окно программы Delphi

**Главное меню**

Главное меню содержит все необходимые команды для управления проектом.

**Пиктографические кнопки**

Пиктографические кнопки предоставляют быстрый доступ к наиболее важным командам главного меню. По выполняемым функциям они разделены на 4 группы (см. приложение №1). Каждая группа занимает отдельную панельку.

Набор пиктографических кнопок можно изменять, удаляя из него редко используемые кнопки или добавляя новые. Чтобы изменить набор кнопок нужно щелкнуть по любой из них правой кнопкой мыши и в появившемся вспомогательном меню выбрать команду Customize (Модифицировать). Если убрать флажок слева от названия группы, то она исчезнет с экрана. Чтобы удалить отдельную кнопку, нужно левой кнопкой мыши «вытащить» её за границу главного окна.

**Палитра компонентов**

На вкладках Палитры компонентов расположены все объекты, которые можно использовать для проектирования визуального окна будущей программы.

палитра компонентов

Рис. 4 Палитра компонентов Delphi

Палитра компонентов может настраиваться с помощью специального редактора. Чтобы его вызвать, нужно щелкнуть правой кнопкой мыши на любой пиктограмме в Палитре компонентов и выбрать команду Properties.

**2.2.** **Окно формы**

**Окно формы** – это визуальное окно будущей программы. Чтобы вызвать это окно, нужно выбрать в меню Delphi команду View, и далее щелкнуть по строчке Forms. Вначале это окно пусто, вся рабочая область заполнена точками координатной сетки. Сетка служит для упорядочения компонентов, размещаемых на форме.

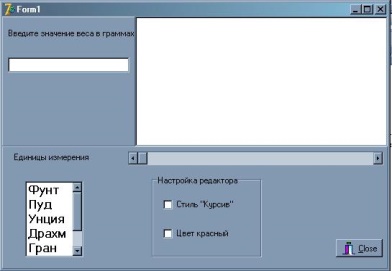
 

Рис. 5 Пустая форма и форма с компонентами

При разработке интерфейса программ на форму просто переносят нужные готовые компоненты из палитры компонентов. Именно в этом наполнении формы и заключается главное удобство визуального программирования.

**2.3.** **Окно Инспектора объектов**

Любой компонент в Delphi имеет определенный набор свойств, например, положение, размер, цвет и т.д. Для изменения этих параметров служит окно **Инспектора Объектов**. Оно содержит две страницы – Properties (Свойства) и Events (События).

Список объектов

Страница свойств

Страница событий

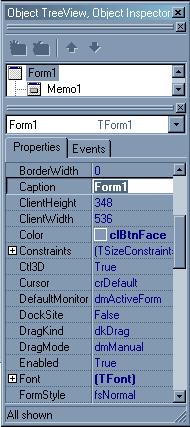


Рис. 6 Окно Инспектора объектов

На странице Properties можно установить нужные свойства у компонента. Некоторые свойства можно выбирать из раскрывающегося списка значений. Например, из списка можно выбрать значение цвета (свойство Color). На странице Events определяется реакция компонента на то или иное событие.

В верхней части окна Инспектора объектов располагается список объектов, в котором указаны все помещенные на форму компоненты. Поскольку форма сама является компонентом, то ее имя также присутствует в этом списке.

**2.4.** **Окно программного кода**

**Окно программного кода** предназначено для создания и редактирования текста программы. Чтобы вызвать это окно, нужно выбрать в меню Delphi команду View, и далее щелкнуть по строчке Units.

Несмотря на то, что визуальная среда Delphi берет на себя многие рутинные операции программирования, знание языка Delphi является непременным условием для работы в этой среде.

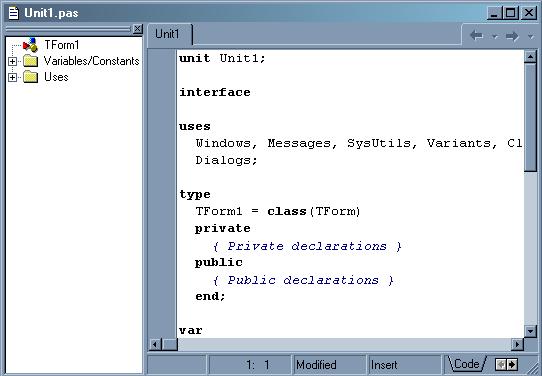


Рис. 7. Окно программного кода

Первоначально окно кода содержит минимальный исходный текст. Он обеспечивает нормальную работу пустой формы. Работая над проектом, Вы будете вносить в этот текст необходимые дополнения.

**2.5.** **Структура проекта Delphi**

Программа в Delphi разделяется на отдельные связанные между собой файлы – модули. Поэтому программирование в Delphi еще называют модульным. При создании каждой новой формы (окна) Delphi автоматически создает новый модуль (unit) программы, соответствующий этому окну.

Проект, созданный в Delphi, имеет следующую структуру:

**1. Главный файл** **проекта** с расширением .dpr изначально называется Project1.dpr. Файл проекта не предназначен для редактирования пользователем, он создается автоматически самой системой программирования Delphi. Чтобы увидеть содержимое файла проекта, нужно выполнить команду Project→View Source. Содержимое файла проекта может быть, например, таким, как на рис. 8:

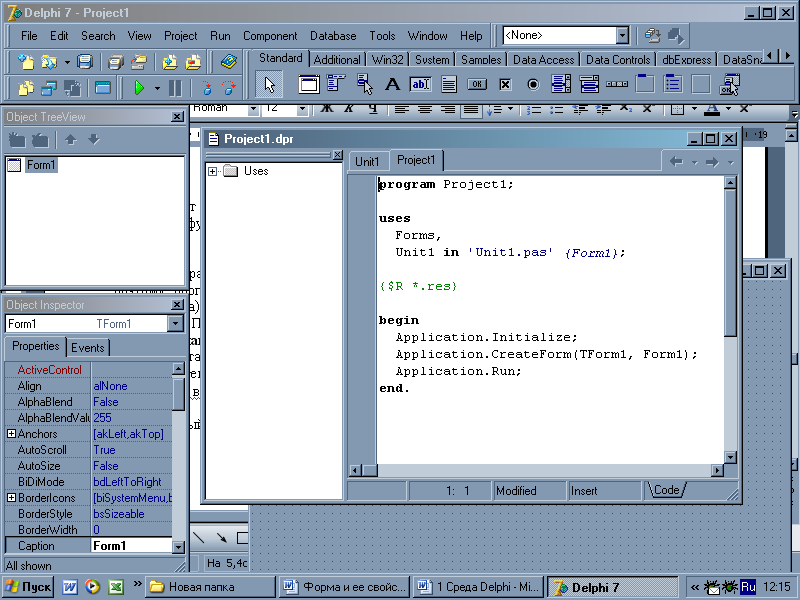


Рис. 8 Содержимое главного файла проекта.

Файл проекта (главный модуль) начинается словом *program*, за которым идет имя программы, совпадающее с именем проекта. Далее за словом *uses* следуют имена используемых модулей: стандартного модуля Forms и модуля формы Unit1. Похожая на комментарий директива *{$R\*.RES}* показывает файл ресурсов (Project1.res), который создается автоматически. В этом файле хранится описание ресурсов программы (первоначально он содержит иконку для проекта). Исполняемая часть главного модуля находится между операторными скобками *begin; end*. Инструкции исполняемой части обеспечивают инициализацию приложения и вывод на экран стартового окна.

**2. Файл программного модуля** имеет расширение .pas и по умолчанию называется Unit1.pas. В нем содержится программный код проекта.

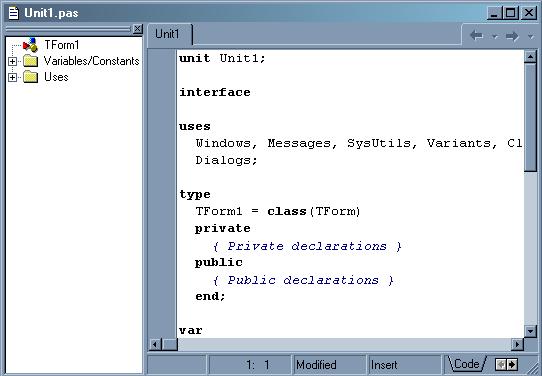


Рис. 9 Содержимое файла программного модуля

**3. Файл главной формы** используется для сохранения информации о внешнем виде формы, имеет расширение .dfm (Unit1.dfm).

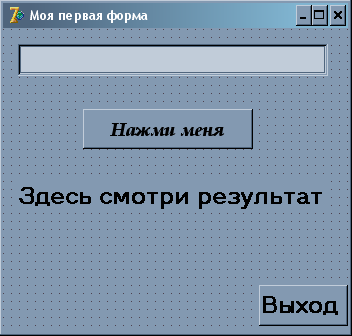
**4. Исполняемый файл** с расширением .exe создается при запуске проекта.

**2.6.** **Основные понятия визуального программирования**

Основной единицей в визуальном программировании является **объект**. Он имеет свойства, способен совершать какие-то действия и реагировать на внешние события. Объектом может выступать сама форма, или любой компонент с Палитры компонентов.



Рис. 10 Объекты панели Standard



Объект Форма

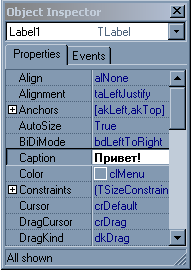
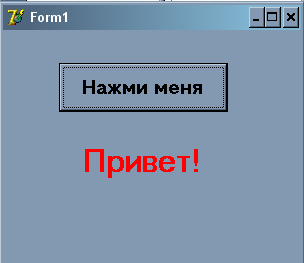
Объект Кнопка

Объект Надпись

Объект Текстовое поле

Рис. 11 Примеры объектов Delphi

**Свойства объектов** устанавливаются с помощью окна Инспектора объектов или в тексте самой программы.

** **

а) б)

Рис. 12 Установка свойства Caption объекта Надпись

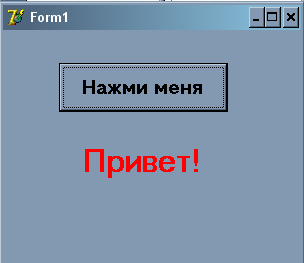
а) с помощью Инспектора объектов

б) вид окна

Чтобы присвоить свойству объекта новое значение в тексте программы нужно в левой части строки указать имя объекта, затем поставить разделитель – точку и далее записать название свойства. Здесь среда Delphi обычно пытается помочь, предлагая выбрать из списка доступных свойств нужное свойство. В правой части строки (после знака присваивания) нужно записать конкретное значение свойства:

**Объект. Свойство: = Значение свойства;**

Например, запись Label1.Caption: = ‘Привет!’ в программном коде означает, что при запуске программы у объекта Надпись с именем Label1 свойство Caption (Заголовок) станет равным Привет!

** **

а) б)

Рис. 13 Установка свойства Caption объекта Надпись

а) с помощью программного кода

б) вид окна

При разработке графического интерфейса окна в среде Delphi используется идеология объектно-ориентированного программирования. Все объекты, помещенные на форму, объединяются средой Delphi в единый **класс объектов**. Также Delphi включает в класс описание процедур и функций, которые выполняются над этими объектами.

Процедуры и функции, заявленные в описании класса, называются **методами**. Метод показывает, что объект умеет делать.

На рис. 14 показано описание класса TForm1. В состав этого класса входят объекты Label1 (Надпись) и Button1 (Кнопка), а также процедура Button1Click, которая является методом.

Описание класса TForm1 (формы) и объектов, размещенных на форме

Метод класса TForm1

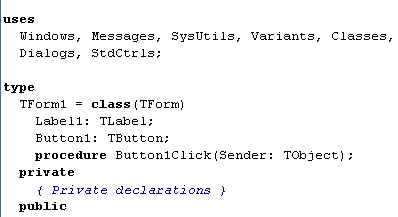


Рис. 14 Описание классов и методов в Delphi

Каждый из компонентов, а также форма описываются соответствующими классами Delphi. Например, форма (Form) описывается классом TForm, кнопка (Button) – классом TButton, надпись (Label) – классом TLabel и так далее.

Компоненты Delhi могут реагировать на внешние **события**, например, на щелчок мышью или нажатие клавиши на клавиатуре. Для каждого события можно запрограммировать его обработчик. Обработчик события – это процедура, которая вызывается при наступлении этого события.

Подробнее основные события Delphi мы рассмотрим ниже.

Обработка события, наступающего при щелчке на кнопке.

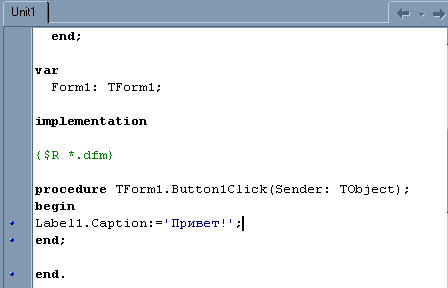


Рис. 15 Описание события в Delphi

Перед тем как составлять программу на языке Delphi нужно спроектировать графический интерфейс окна будущего проекта.

**§3** **Проектирование графического интерфейса**

**3.1.** **Форма и ее основные свойства**

Когда запустите программу Delphi, она автоматически предложит пустое окно – форму. Также создать новую форму можно с помощью команды [File – New – Application] (Файл – Создать – Приложение).

Перед тем, как сохранить проект, создайте отдельную папку, специально отведенную для него. В этой же папке в будущем будет создано и исполняемое приложение. Чтобы сохранить текущий проект, надо выполнить команду [File - Save All] (Файл - Сохранить все) или щелкнуть по одноименной командной кнопке. Сначала Вам будет предложено сохранить файл с исходным текстом модуля (Unit1.pas), а затем – файл проекта (Project1*.*dpr)**.**

Чтобы открыть уже сохраненную программу, нужно выбрать файл с расширением *.dpr*. Если открыть файл с расширением *.pas*, то появится лишь составляющая часть проекта, которая ничего не даст.

Для запуска программы на выполнение необходимо нажать кнопку F9.

Также в Delphi программу обычно компилируют. Компиляция – это преобразование исходной программы в исполняемую. Процесс компиляции состоит из двух этапов. На первом этапе выполняется проверка текста программы на отсутствие ошибок, на втором – создается исполняемая программа (ехе-файл). Для компиляции программы необходимо выбрать команду меню [Project – Compile Progect] (Проект – Компилировать проект)**.**



Рис. 16 Окно формы

Рассмотрим основные свойства формы. Многие из этих свойств имеются и у других компонентов Delphi.

**Основные свойства формы**

**1.** **Свойство Name (Имя).**

Форма является компонентом Form класса TForm. Новой форме автоматически присваивается имя Form1 типа TForm1. Форме и компонентам следует давать такие имена, которые помогают понять, что именно будет делать тот или иной компонент.

Изменить имя формы можно с помощью свойства **Name (Имя)**. Это свойство есть у всех объектов Delphi без исключения. Именно по имени любой объект доступен в тексте программы. В свойстве Name (Имя) можно использовать только латинские буквы и цифры, так как содержимое этого свойства соответствует названию переменной Delphi – идентификатору.

Изменим имя формы с Form1 на **MyForm**.

Свойство Name

Текущее имя объекта

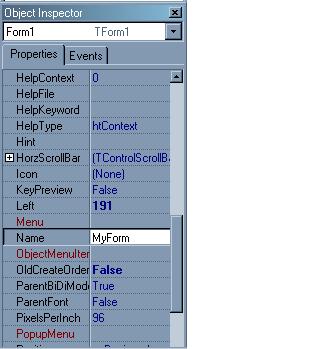


Рис. 17 Изменение имени формы с помощью окна Инспектора объектов

После этого заголовок формы изменится с Form1 на MyForm. Произойдут изменения и в окне кода программы: тип изменится с TForm1 на TMyForm, имя переменной Form1 изменится на MyForm. Все эти перемены выполнятся автоматически.

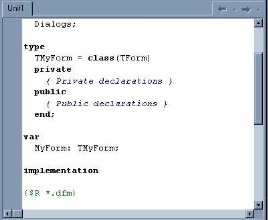
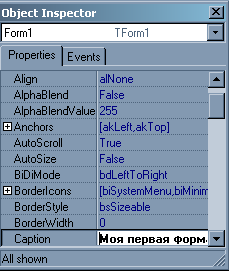
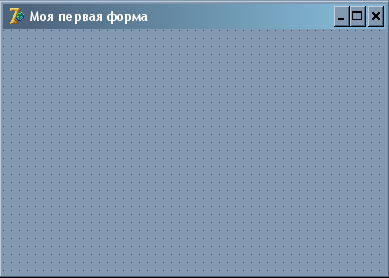


Рис. 18 Окно кода программы с новым именем формы

**2. Свойство Caption (Заголовок)** – это заголовок формы или компонента. Здесь можно ввести строчку **Моя первая форма**, при этом вместо заголовка формы **MyForm** автоматически появится заголовок **Моя первая форма**. Заголовок – это произвольная строка, не имеющая пря­мого отношения к программированию, поэтому в ней можно использовать любые символы.

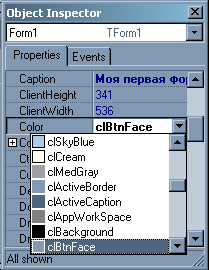
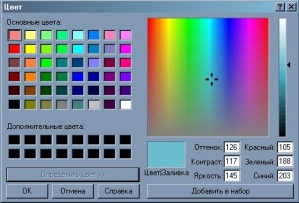


Свойство Caption

Заголовок объекта

Рис. 19 Изменение заголовка формы с помощью Инспектора объектов

**3. Свойство Color (цвет)** – это цвет рабочей области окна или компонента. В списке выбора есть все системные цвета, которые можно использовать. Для получения других цветов нужно дважды щелкнуть по списку выбора, после этого откроется стандартное окно выбора цвета.

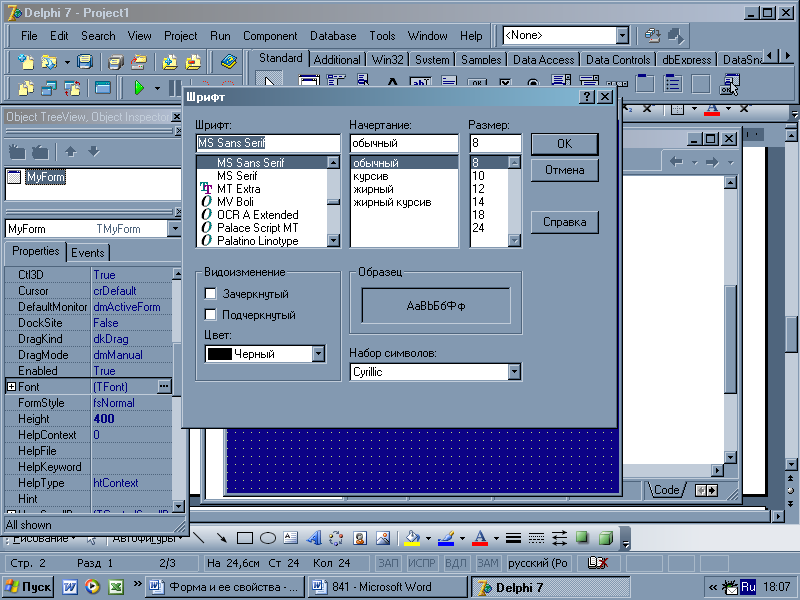
** **

Свойство Color

Стандартное окно выбора цвета

Рис. 20 Выбор цвета

**4. Свойства Height (Высота) и Width (Ширина)** – задают высоту и ширину формы или компонента. Конечно, размеры объектов можно задать и с помощью мышки, перетаскивая границы, но для установки точных значений удобно пользоваться этими свойствами.

**5. Свойство Font (Шрифт)** – задает параметры шрифта, используемого при выводе текста на форме. Чтобы изменить параметры шрифта нужно щелкнуть по кнопке , после этого откроется стандартное окно выбора шрифта.

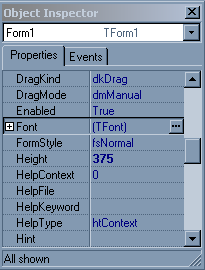
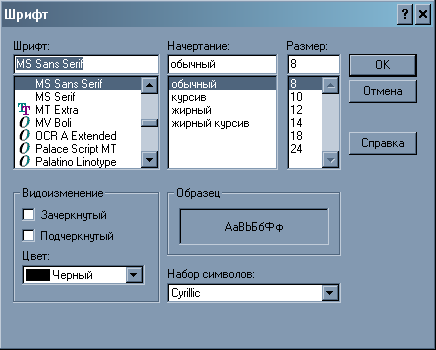
 

Рис. 21 Выбор шрифта

**6. Свойство Left (Левая граница) –** левая позиция окна или компонента. **Свойство Top (Верхняя граница) –** верхняя позиция окна или компонента.

**7. Свойство ClientHeight (рабочая высота)** показывает высоту рабочей области окна. **Свойство ClientWidth (рабочая ширина)** показываетширину рабочей области окна. Рабочая высота и ширина – это область без ширины границы и без системного меню.

**8. Свойство Align (Выравнивание)** – выравнивание компонента. Любой компонент может быть выровнен по одной из сторон формы или другого компонента. Свойство Align может иметь следующие значения:

*alNone* – нет выравнивания.

*alBottom* – выравнивание по нижнему краю.

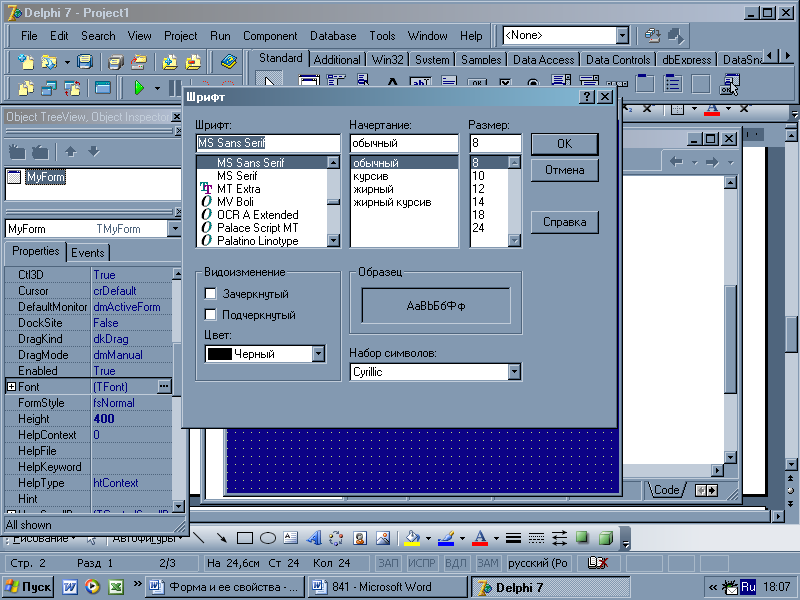
*alLeft* – выравнивание по левому краю.

*alRight* – выравнивание по правому краю.

*alTop –* выравнивание по верхнему краю.

*alClient* – форма занимает всю область окна.

*alCustom* – компоненты выравниваются относительно формы, а форма выравнивается относительно окна.

**9. Свойство Icon (Иконка)** – добавляет иконку, которая отображается в заголовке окна. Если щёлкнуть по кнопке  этого свойства, то появится окно загрузки иконки (рис. 22).

В этом окне есть следующие команды:

*Load* – загрузить иконку из файла.

*Save* – сохранить иконку в файл.

*Clear* – очистить текущую иконку.

*OK* – выполнить.

*Cancel* – отменить.

*Help* – помощь.

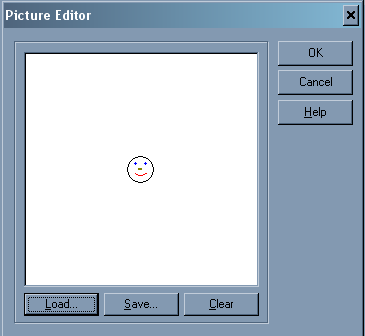


Рис. 22 Окно загрузки иконки

Для создания своей иконки применяется Редактор Картинок. Его можно открыть командой меню [Tools - Image Editor]. В появившемся окне Image Editor (Редактор картинок) следует выбрать команду File – New – Icon File и далее задать размер иконки.



Рис. 23 Окно Редактора картинок

Нарисованную иконку необходимо сохранить командой File – Save as в ту же папку, в которой располагается проект. Чтобы добавить иконку в заголовок формы используется команда Load в окне загрузки иконки (рис. 22).

**10. Свойство Anchors (Закрепление)** – это свойство есть и у формы, и у компонентов. Оно показывает, как происходит закрепление к форме или другому объекту. При щелчке по значку + слева от имени свойства раскрывается список из четырех дополнительных свойств:

*akLeft –* прикреплять левый край (по умолчанию true).

*akTop –* прикреплять верхний край (по умолчанию true).

*akRight –* прикреплять правый край (по умолчанию flase).

*akBottom –* прикреплять нижний край (по умолчанию false).

По умолчанию прикрепление происходит по левому и верхнему краю.

**11. Свойство BorderIcons (Кнопки границы)** – это свойство, которое определяет, какие кнопки должны присутствовать у окна. Возможны следующие кнопки:

*biSystemMenu* – показать меню (иконка слева в строке заголовка окна) и другие кнопки заголовка окна.

*biMinimize* – кнопка минимизации окна.

*biMaximize* – кнопка максимизации окна.

*biHelp* – кнопка помощи.

*biSystemMenu*

*biMinimize*

*biMaximize*

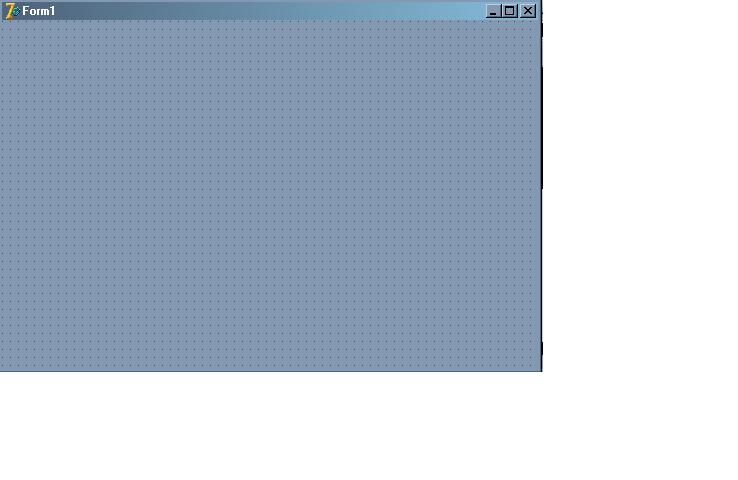


Рис. 24 Кнопки границы

**12. Свойство BorderStyle (Стиль границы) –** свойство формы, которое отвечает за вид обрамления или границы окна:

*bsSizeable* – установлено по умолчанию. Стандартное окно, с нормальным обрамлением, которое может изменять свои размеры.

*bsDialog* – окно выглядит в виде диалога.

*bsNone* – окно без границы.

*bsSingle* – окно с фиксированным размером и изменять его мышкой нельзя. Изменить размер можно только кнопкой Maximize.

*bsSizeToolWin* – окно с тонкой границей.

*bsToolWindow* – окно похоже на предыдущее, только у него нельзя изменять размеры.

**13. Свойство BorderWidth (Ширина границы) –** ширина обрамления окна.

**14. Свойство TransparentColor (Прозрачный цвет)**– показывает, является ли форма или компонент прозрачными. Если установлено значение True, то форма или компонент будут невидимыми. При значении False объекты видны. Это свойство не позволяет создавать полупрозрачные элементы.

**15. Свойство AlphaBlend** – также показывает, является ли форма или компонент прозрачным. Если установлено значение True, то форма или компонент будут прозрачными. Но здесь также можно задать степень прозрачности компонента через свойство AlphaBlendValue.

[**Задания**](#форма_свойства)

1. Измените имя и заголовок формы. Новое имя – MyForm, заголовок – Моя первая форма. Какие изменения произошли при этом в окне кода программы?
2. Измените цвет формы и параметры шрифта.
3. Установите следующие размеры формы: высота окна – 400, ширина окна – 500.
4. Создайте с помощью Редактора картинок иконку и добавьте ее в заголовок формы.

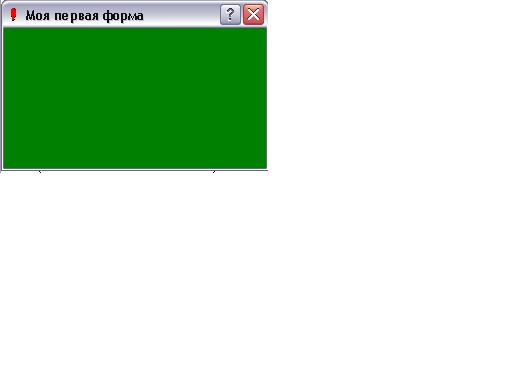


Рис. 25 Образец окна

1. Измените значение прозрачности формы с помощью свойств TransparentColor, AlphaBlend, AlphaBlendValue. Запустите программу на выполнение с различными значениями параметров.
2. Задайте с помощью свойства Border Style различные границы окна и запустите программу на выполнение с каждым значением свойства.
3. Создайте новую папку *Первая программа* и сохраните в ней сначала файл с исходным кодом модуля (Unit1.pas), затем – файл проекта (Project1.dpr). Откройте папку *Первая программа*, в ней появились также файлы с содержимым формы Unit1.dfm, с параметрами проекта Project1.dof, с описанием ресурсов Project1.res и исполняемое приложение (Project1.exe).
4. Откройте каждый файл из папки *Первая программа.* В чем их разница?

**3.2** **Компоненты TEdit, TLabel, TButton**

Рассмотрим некоторые компоненты панели Standard и их свойства. С помощью этих компонентов уже можно создавать простейшие приложения.

С панели компонентов Standard поместим на нашу форму MyForm следующие объекты:

* **TEdit (Текстовое поле)** компоненты **-** часто используется для ввода исходных данных.
* **TButton (Кнопка)** компоненты **-** по щелчку на кнопке выполняются какие-либо действия.
* **TLabel (Надпись)** компоненты **-** служит для отображения текста на экране.

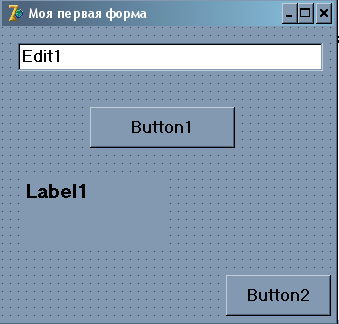
****

Рис. 26 Форма с компонентами

Если поместить на форму еще одну кнопку, то первая созданная кнопка будет иметь имя Button1, а вторая Button2. По такому же принципу даются имена и всем остальным компонентам Delphi.

Черные маркеры по контуру объекта указывают, что он выделен. Эти маркеры служат для изменения размеров объекта с помощью мышки.

В окне кода программы автоматически указываются типы созданных объектов.

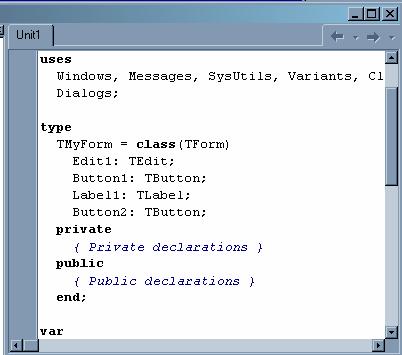


Рис. 27 Окно кода программы с объявлением типов созданных объектов

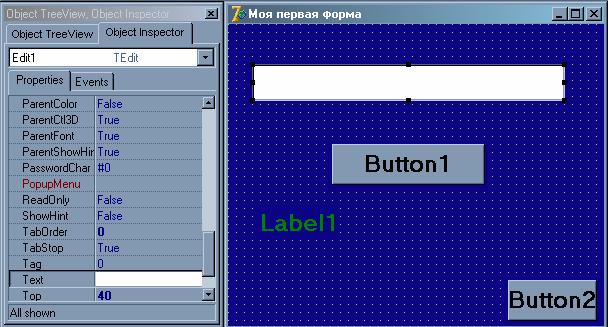
**Основные свойства компонентов TEdit, TButton и TLabel**

Свойства, рассмотренные для формы, в основном применяются и для компонентов TEdit, TButton и TLabel. Но у них также есть и свои особенности.

***Свойства TEdit***

У компонента TEdit свойства Caption (Заголовок) нет. Вместо него активно используется свойство Text (Текст).

**Свойство Text (Текст) –** содержит строку, которая вводится в текстовое поле. Это свойство имеет строковый тип **string** (о типах данных, используемых в Delphi, более подробно будет говориться позже).Первоначально свойство Text содержит строку, совпадающую с именем текстового поля (Editl, Edit2). При проектировании графического интерфейса лучше это имя удалить.



Текстовое поле пусто

Содержимое свойства Text удалено

Рис. 28 Изменение свойства Text с помощью Инспектора объектов

***Свойства TLabel***

**1. Свойство Autosize (Авторазмер)** – автоматически изменяет размеры компонента Label в зависимости от текста надписи. Это свойство логического типа Boolean. Если установлено значение True, то компонент Label изменяет свои размеры в соответствии с длиной текста, заданной в свойстве Caption (Заголовок).

**2. Свойство Alignment (Выравнивание)** – выравнивает тест внутри надписи. Это свойство может принимать одно из следующих значений:

* taLeftJustify – выравнивание по левому краю;
* taCenter – выравнивание по центру;
* taRightJustify – выравнивание по правому краю.

**3. Свойство WordWrap (Автоматический перенос)** – это свойство позволяет автоматически переносить слова на другую строчку надписи. По умолчанию свойство WordWrap имеет значение False и текст в надписи записывается в одну строку. Для длинных надписей удобно установить значение True этого свойства, чтобы можно было вводить текст в несколько строчек.

**Задания**

1. Поместите на форму компоненты TEdit, TLabel, TButton, измените имена и заголовки установленных объектов:
   * для текстового поля Edit1 задайте имя TEnter, текст из поля удалите.
   * для кнопки Button1 задайте имя TPress, заголовок – Нажми меня; для кнопки Button2 – имя TClose, заголовок – Выход.
   * для надписи задайте имя TPrint, заголовок – Здесь смотри результат.
2. Установите размеры, параметры шрифта, тип выравнивания, цвет компонентов в соответствии с образцом.

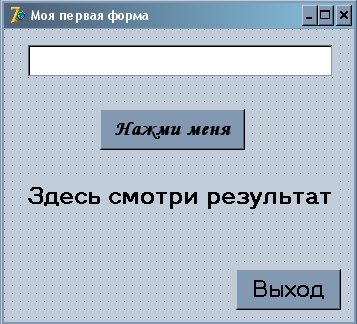


Рис. 29 Образец окна

1. Измените значение свойства формы Align. Запустите программу при различных значениях выравнивания.
2. Задайте различные способы закрепления объектов:

* кнопку *Выход* закрепите к правому нижнему краю.
* надпись к верхнему краю;
* текстовое поле к левому, верхнему краю;
* кнопку *Нажми меня* к левому, правому и нижнему краям.

1. Измените значение прозрачности надписи.
2. С помощью свойства Cursor измените внешний вид курсора.
3. Поэкспериментируйте с остальными свойствами формы и рассмотренных компонентов.

**3.3** **Графическое оформление окна**

Система программирования Delphi предоставляет большие возможности для графического оформления проектов. Рассмотрим компонент TImage с панели Additional.

**Компонент TImage (Изображение)** комп23 **-** добавляет картинку на форму. Нужно выбрать этот компонент и поместить его на форму.

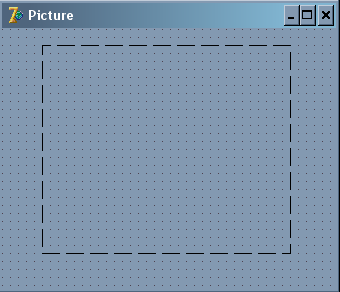


Рис. 30 Форма с компонентом TImage

**Основные свойства TImage**

**1. Свойство Picture (Картинка)** загружает в компонент TImage готовый рисунок. При обращении к этому свойству открывается окно выбора графического изображения. Чтобы выбрать необходимый рисунок, нужно нажать кнопку Load.

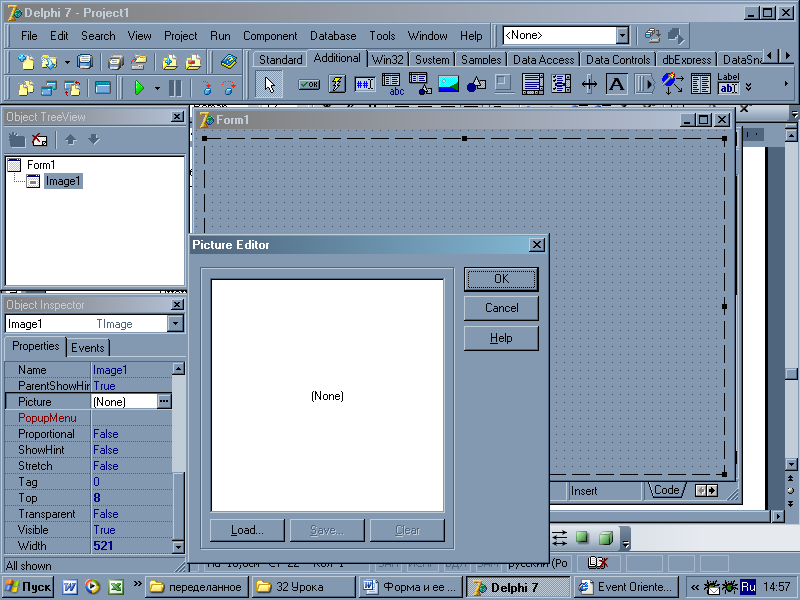


Рис. 31 Свойство Picture у компонента Image.

Рисунок, помещаемый на форму должен иметь форматы bmp, wmf или jpg. Размеры рисунка предварительно желательно изменить до нужных (например, до размера формы или компонента).

Для создания собственных изображений достаточно использовать простейший графический редактор Paint. Но при стандартном сохранении рисунка командой [Файл – Сохранить как] сохраняется и белая область вокруг рисунка, которая часто оказывается ненужной. Чтобы получить только один рисунок без белой области нужно выделить рисунок и выполнить команду [Правка – Копировать в файл]. Все графические объекты, используемые в проекте, желательно помещать в ту же папку, в которой находится сам проект.

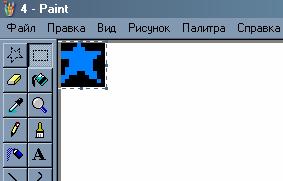


Рис.32 Создание графических объектов в программе Paint

**2. Свойство Center** **(Центр)** – если установлено значение True, то центр изображения будет совмещаться с центром объекта TImage.

**3. Свойство Stretch (Растяжение)** – если установлено значение True, то изображение будет сжиматься или растягиваться так, чтобы заполнить весь объект TImage.

**4. Свойство AutoSize (Авторазмер)** – автоматически изменяет размер компонента Image в соответствии с реальным размером рисунка.

**5. Свойство Proportional (Пропорция) –** пропорционально изменяет размер изображения в соответствии с размером компонента TImage.

**6. Свойство Visible (Видимость)** – если установлено значение True, то изображение будет видимым; если значение False, то невидимым.

**Создание графического интерфейса проекта в Delphi**

Каждый новый проект лучше сразу сохранять в специально отведенную для него папку командой File - Save All.

Создадим проект, в котором на форму помещается изображение солнца.

Для этого с помощью графического редактора Paint нарисуем солнце и сохраним его в папке с нашим проектом, с именем Sun.bmp. Всем файлам, используемым в Delphi, лучше давать имена, записанные латинскими буквами.

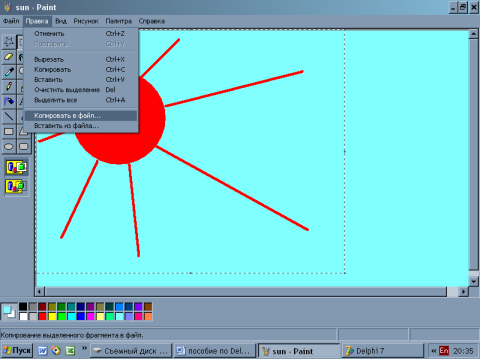


Рис. 33 Сохранение рисунка командой [Правка – Копировать в файл].

Если имеется лишняя пустая область, то выделим нужное изображение солнца и выполним команду [Правка – Копировать в файл].

На форму поместим компонент TImage и загрузим в свойство Picture созданный нами рисунок Sun.bmp. Установим соответствующий фон формы и зададим свойство Transparent (прозрачность) у компонента Image1 равным true.

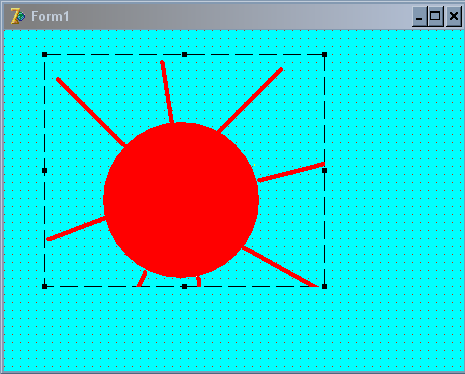


Рис. 34 Форма с компонентом TImage.

Наш рисунок не поместился в компонент, поэтому поэкспериментируем со свойствами Stretch, AutoSize или Proportional объекта TImage. При этом получится одно из следующих оформлений окна.

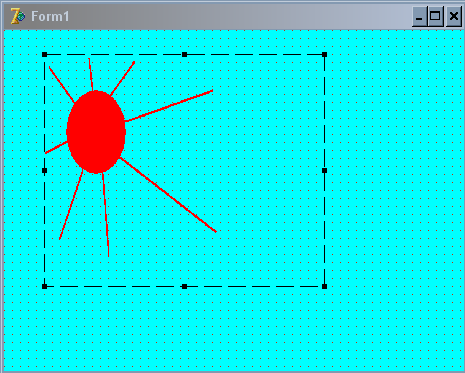
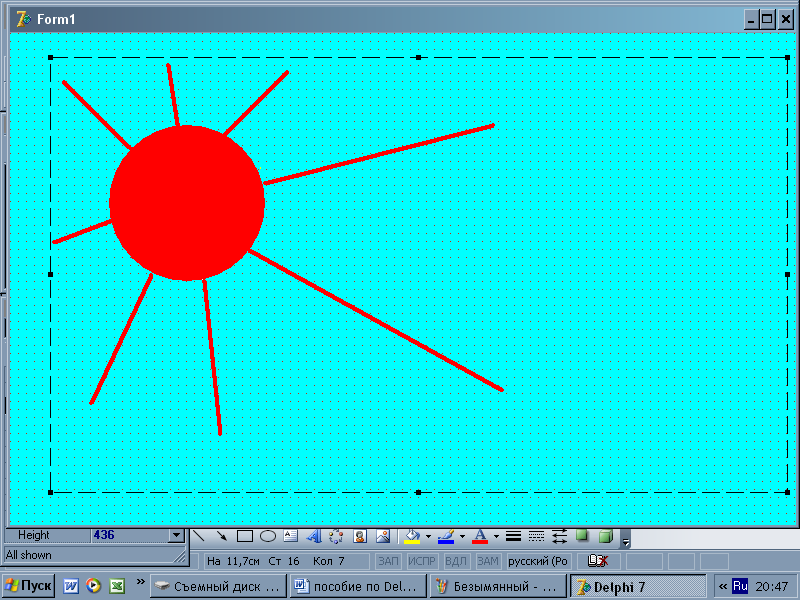
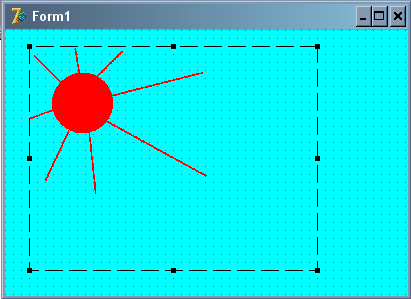
а) б) в)

Рис. 35 Изменение размеров изображения различными способами

а) устанавливая свойству Stretch значение True.

б) устанавливая свойству AutoSize значение True.

в) устанавливая свойству Proportional значение True.

[**Задания**](#графическое_оформление)

1. Создайте форму с изображением домика, нарисованного с помощью графического редактора Paint. Измените свойства Stretch, AutoSize или Proportional так, чтобы поместилась вся картинка.

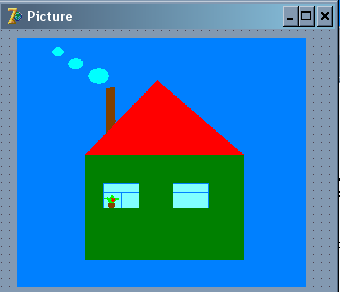


Рис. 36 Графический интерфейс окон

1. Создайте графический интерфейс калькулятора по образцу, используя имеющееся графическое изображение.
2. Создайте графический интерфейс поздравительной открытки по образцу, используя имеющееся графическое изображение.

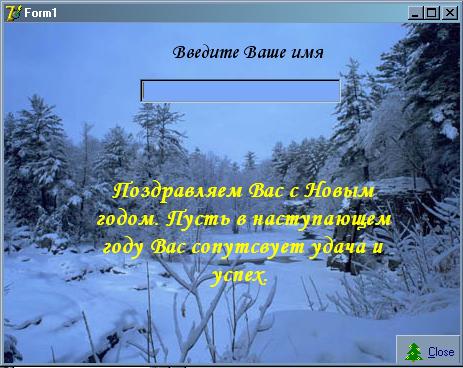
** **

Рис. 37 Графический интерфейс окон

**3.4** **Реакция на события**

Созданные нами окна никак не реагировали на какие-либо действия. Для создания «работоспособного» окна нужно вносить изменения в программный код.

Каждый компонент или элемент управления может реагировать на различные события. Однако есть события, например, Click (щелчок), на которые реагирует большинство элементов управления. Рассмотрим работу этого события.

|  |
| --- |
| **Пример.** Составить программу, которая завершает работу по щелчку на кнопке. |

Создадим графический интерфейс проекта, для этого поместим на форму компонент TButton (Кнопка). При щелчке именно по этой кнопке окно должно закрываться. С помощью Инспектора объектов изменим имя кнопки с Button1 на TClose, заголовок с Button1 на Выход. Можно также поменять параметры шрифта, чтобы заголовок был хорошо виден.

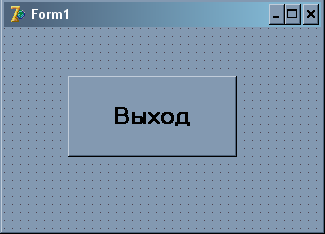


Рис. 38 Графический интерфейс окна

В Инспекторе объектов перейдем на закладку Events (События). За щелчок по любому компоненту, в том числе и по кнопке отвечает событие OnClick. Щёлкнем дважды по этому событию у кнопки Выход (можно также дважды щелкнуть по самой кнопке Выход). После этого Delphi создаст в редакторе кода процедуру – обработчик события OnClick.

|  |
| --- |
| **procedure** TForm1.TCloseClick(Sender: TObject); |
| **Begin** |
|  |
| **end;** |

По умолчанию имя процедуры начинается с имени класса TForm1, к которому относится данная процедура. Далее через точку записывается имя компонента TClose и имя события Click. В описании класса эта процедура объявляется автоматически (рис. 39).

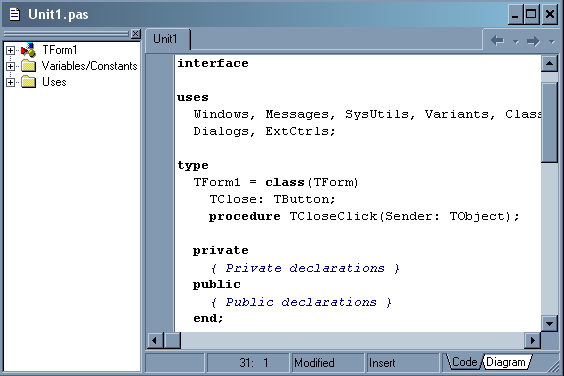


Рис. 39 Объявление процедуры в описании класса TForm1

В окне Инспектора объектов напротив строки OnClick также должно появиться имя процедуры обработчика.

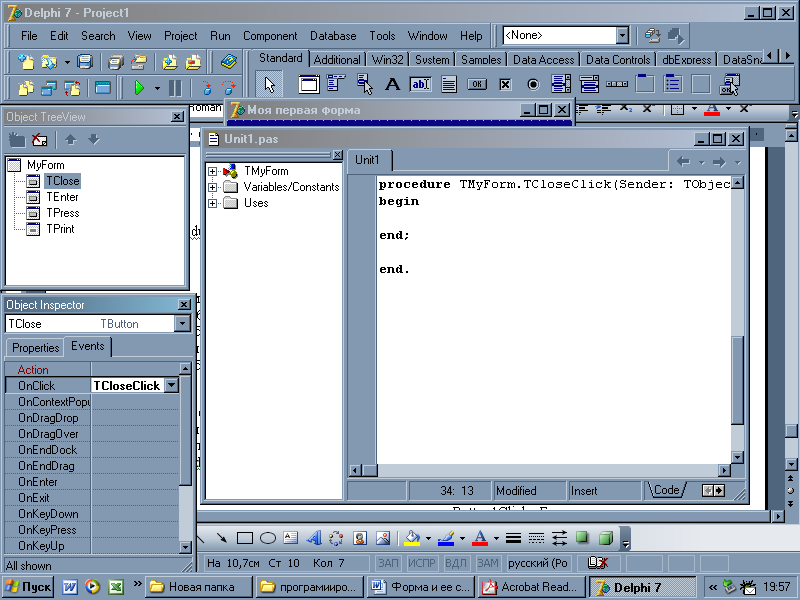


Рис. 40 Закладка Events

с именем обработчика события.

Созданная процедура имеет только один параметр Sender объектного типа. В начале выполнения процедуры в переменной Sender будет находиться указатель на объект, который вызвал этот обработчик. В данной программе в переменной Sender будет находиться указатель на кнопку TClose.

Напишем внутри процедуры (между *begin* и *end*) команду *Close*. Эта команда закрывает окно. Теперь процедура обработчика будет выглядеть так:

|  |
| --- |
| **procedure** TForm1.TCloseClick(Sender: TObject); |
| **Begin** |
| Close; |
| **end;** |

После запуска программы окно может закрываться как по щелчку на кнопке Выход, так и по щелчку на крестике в системном меню.

В таблице №2 приведены основные события, на которые реагирует форма и большинство компонентов Delphi.

**Таблица 2. Основные события Delphi**

|  |  |
| --- | --- |
| **Событие** | **Описание** |
| OnActivate | Применяется, когда приложение становится активным. |
| OnClick | Применяется, когда происходит щелчок по компоненту. |
| OnClose | Применяется, когда окно закрывается. |
| OnCreate | Применяется, когда окно создается. |
| OnKeyDown | Применяется, когда нажата клавиша на клавиатуре. |
| OnKeyPress | Применяется, когда нажата и отпущена клавиша на клавиатуре. |
| OnKeyUp | Применяется, когда отпущена клавиша на клавиатуре. |
| OnMouseDown | Применяется, когда нажата кнопка мыши. |
| OnMouseMove | Применяется, когда двигается мышка. |
| OnMouseUp | Применяется, когда отпускается кнопка мыши. |
| OnPaint | Применяется, когда надо перерисовать окно. |
| OnShow | Применяется, когда показывается окно, но до фактической прорисовки. В этот момент окно уже создано и готово к отображению, но ещё не прорисовалось на экране. |

**3.5 Изменение свойств компонентов**

**через программный код**

В данной главе рассматривались различные свойства формы и компонентов TEdit, TLabel, TButton и TImage, а также изменение их свойств с помощью окна Инспектора объектов. Задать новые значения свойств можно также через программный код. Рассмотрим это на примере.

|  |
| --- |
| **Пример.** Создать проект, в котором по щелчку на кнопке в надписи должен появиться текст «Я учусь программировать в Delphi»**.** |

Разработаем графический интерфейс проекта. Из условия задачи следует, что на форму необходимо поместить компоненты TButton (Кнопка) и TLabel (Надпись).

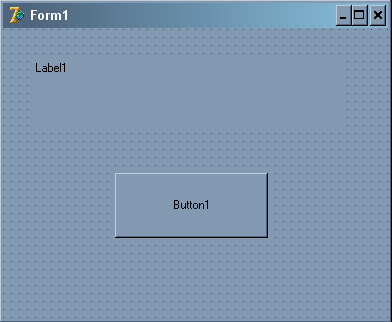


Рис. 41 Форма с Надписью и Кнопкой.

У кнопки Button1 изменим заголовок на «OK», а у надписи Label1 заголовок удалим совсем. Можно также поменять параметры шрифта у кнопки и надписи.

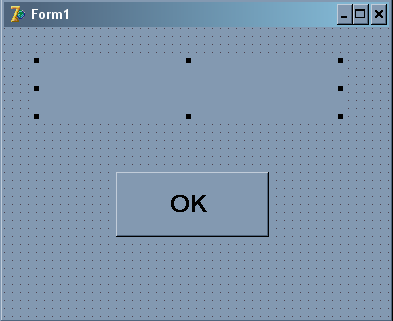


Рис. 42 Графический интерфейс проекта.

После того, как графический интерфейс создан, можно начинать программировать. В данном проекте текст в надписи должен появиться при щелчке мышью по кнопке «ОК». Таким образом, событием здесь является щелчок мыши по кнопке. Поэтому необходимо создать обработчик события OnClick для кнопки Button1. Для этого дважды щелкнем по кнопке, при этом откроется окно программного кода с процедурой обработки события OnClick.

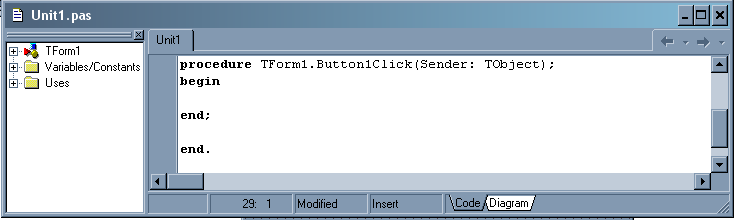


Рис. 43. Окно программного кода с обработчиком события OnClick.

Чтобы в надписи появился текст нужно записать команду, которая изменяет свойство Заголовок (Caption) у компонента Label1 на строку «Я учусь программировать в Delphi». Другими словами, необходимо присвоить свойству Caption новое значение в тексте программы.

Чтобы присвоить свойству новое значение в тексте программы нужно в левой части строки указать имя объекта (Label1), затем поставить разделитель – точку и далее записать название свойства (Caption). В правой части строки (после знака присваивания :=) нужно записать конкретное значение свойства:

**Label1.Caption: = ‘Я учусь программировать в Delphi’;**

Таким образом, текст в обработчике события будет следующим:

|  |
| --- |
| **procedure** TForm1.Button1Click(Sender: TObject); |
| **Begin** |
| Label1.Caption: = ‘Я учусь программировать в Delphi’; |
| **end;** |

Свойство Caption – это свойство строкового типа string. Значения строкового типа заключаются в апострофы.

После запуска проекта нажмем на кнопку «ОК» и в надписи появится фраза «Я учусь программировать в Delphi».

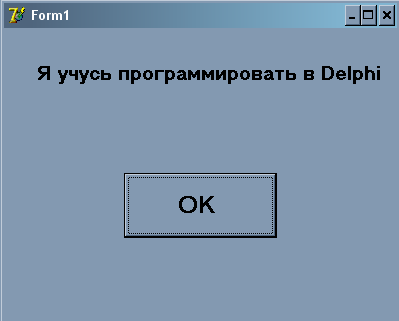


Рис. 44 Работающее приложение

**Задания**

1. Создать проект, в котором по щелчку на кнопке в текстовом поле должно появиться слово «Привет»**.**
2. Добавить в предыдущий проект кнопку «Очистить», по щелчку на которой содержимое текстового поля должно удаляться.

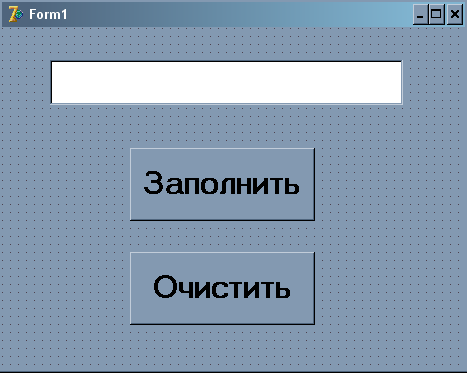


Рис. 45 Графический интерфейс проекта

1. Создать проект, в котором требуется ввести имя в текстовое поле. При нажатии на кнопку в надписи должна появиться фраза «Привет, *имя*. Как дела?».

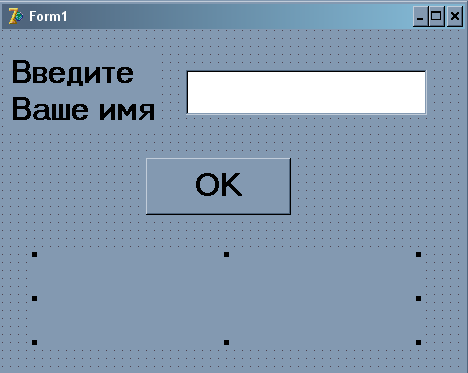
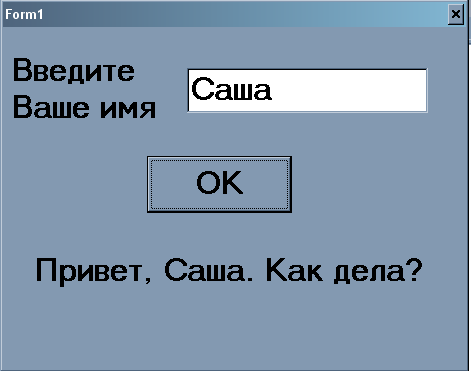
а)  б) 

Рис. 46 а) Графический интерфейс проекта

б) Работающее приложение

1. Создать проект, в котором в текстовые поля требуется ввести фамилию, имя и отчество. По щелчку на кнопке полное ФИО должно появиться в надписи.

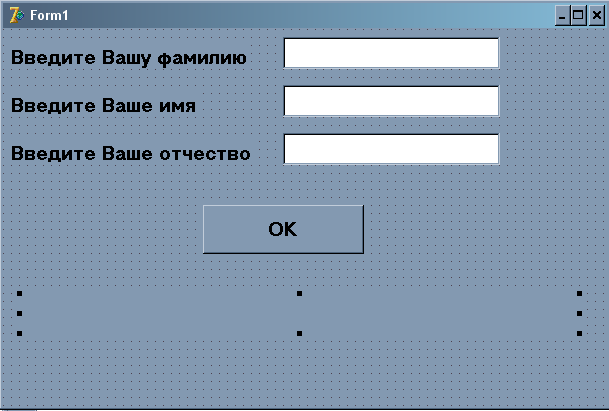
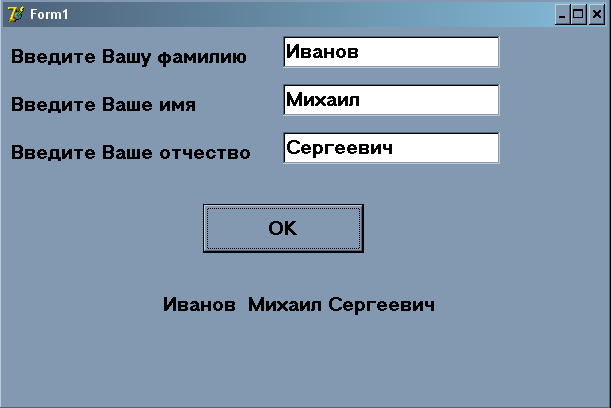
а)  б) 

Рис. 47 а) Графический интерфейс проекта

б) Работающее приложение

1. Создать графический интерфейс проекта по образцу (рис. 48). При нажатии на каждую из кнопок в текстовом поле должно появляться соответствующее слово, причем уже имеющиеся слова удаляться не должны. Соберите из этих слов известную пословицу. По щелчку на кнопке «Очистить» текстовое поле очищается.

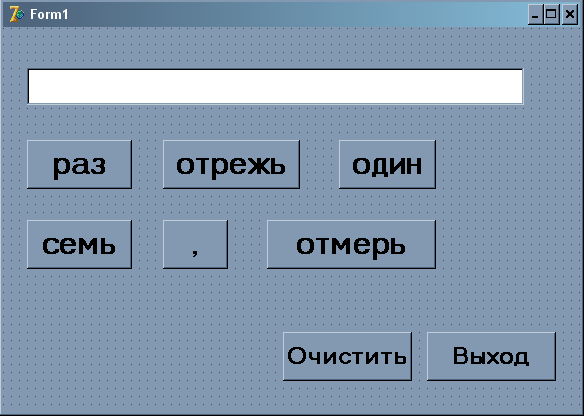


Рис. 48 Графический интерфейс проекта

**§4** **Элементы языка программирования Delphi**

**4.1** **Алфавит**

Алфавит языка Delphi включает буквы, цифры, шестнадцатеричные цифры, специальные символы, пробел, знак подчеркивания и зарезервированные слова. Буквы – это буквы латинского алфавита от а до z и от А до Z. В Delphi нет разницы между строчными и прописными буквами алфавита, если только они не входят в символьные и строковые выражения.

Цифры – арабские цифры от 0 до 9.

Шестнадцатеричные цифры имеют значения от 0 до 15. Первые 10 обозначаются арабскими цифрами от 0 до 9, остальные – это A, B, C, D, E, F.

Специальные символы Delphi – это символы:

+ - \* / = , ‘ . : ; <> [ ] ( ) { } ~ ^ @ $ #

К специальным символам также относятся составные символы, которые воспринимаются компилятором как единое целое:

<= >= := (\* \*) (. .) // ..

**4.2** **Зарезервированные слова и идентификаторы**

В Delphi имеются следующие **зарезервированные слова**:

And Except Library Set

Array Exports Mod Shl

As File Nil Shr

Asm Finalization Not String

Begin For Object Then

Case Function Of Threadvar

Class Goto Or To

Const If Out Try

Constructor Implementation Packed Type

Destructor In Procedure Unit

Dispinterface Inherited Program Until

Div Initialization Property Uses

Do Inline Raise Var

Downto Interface Record While

Else Is Repeat With

End Label Resourcestring Xor

Зарезервированные слова не могут использоваться в качестве идентификаторов.

**Идентификаторы** в Delphi – это имена переменных, констант, меток, типов, объектов, классов, свойств, процедур. Идентификатор всегда начинается буквой, за которой могут следовать буквы и цифры. Пробелы и специальные символы алфавита не могут входить в идентификатор.

**Переменная** в Delphi – это величина, которая может изменяться в процессе выполнения программы. Любая переменная перед использованием обязательно должна быть объявлена, т.е должен быть указан ее тип в разделе описания переменных (после зарезервированного слова Var).

Например, **Var** *a: integer* задает переменную *а* целого типа.

**4.3** **Типы данных**

Все данные, используемые в программе (константы, переменные, значения функций или выражений), характеризуются своими типами*.* Тип определяет множество допустимых значений, которые может иметь тот или иной объект, а также множество допустимых для него операций. Кроме того, тип определяет и формат внутреннего представления данных в памяти компьютера.

В Delphi используются следующие типы данных: простые, структурированные, указатели, процедурные, варианты.

По мере изучения Delphi ознакомимся со всеми этими типами данных, но начнем с простых типов.

К **простым типам** **данных** относятся порядковые, вещественные типы и тип дата-время.

**1.** **Порядковые типы**отличаются тем, что каждый из них имеет конечное число возможных значений. Эти значения можно определенным образом упорядочить и каждому сопоставить некоторое целое число – порядковый номер значения.

К порядковым типам относятся целый, логический, символьный, перечисляемый типы и тип-диапазон. Диапазон возможных значений целых типов зависит от их внутреннего представления, которое может занимать один, два, четыре или восемь байтов.

**Таблица 3. Целые типы данных**

|  |  |  |
| --- | --- | --- |
| **Целые типы** | | |
| **Название типа** | **Длина, байт** | **Диапазон допустимых значений** |
| Byte | 1 | 0 ... +255 |
| ShortInt | 1 | -128 ... +127 |
| SmallInt | 2 | -32 768 ... +32 767 |
| Word | 2 | 0 ... +65 535 |
| Integer | 4 | -2 147 486 648 ... +2 147 487 647 |
| LongInt | 4 | -2 147 483 648 ... +2 147 483 647 |
| LongWord | 4 | 0 ... + 4 294 967 295 |
| Int64 | 8 | -9∙1018 ... +9∙1018 |
| Cardinal | 8 | 0 ... 2 147 483 647 |

**2.Вещественные типы**, строго говоря, тоже имеют конечное число значений, которое определяется форматом внутреннего представления вещественного числа. Однако количество возможных значений вещественных типов настолько велико, что сопоставить с каждым из них целое число не представляется возможным. Значения вещественных чисел определяются лишь с определенной степенью точности, зависящей от формата вещественного числа.

**Таблица 4. Вещественные типы данных**

|  |  |  |  |
| --- | --- | --- | --- |
| **Длина, байт** | **Название типа** | **Количество значащих цифр** | **Диапазон допустимых значений** |
| 8 | Real | 15 … 16 | 5,0∙10-324 … 1,7∙10308 |
| 4 | Single | 7 … 8 | 1,5∙10-45 … 3,4∙1038 |
| 8 | Double | 15 … 16 | 5,0∙10-324 … 1,7∙10308 |
| 10 | Extended | 19 … 20 | 3,4∙10-4951 … 1,1∙104932 |
| 8 | Comp | 19 … 20 | -263 … +263 -1 |
| 8 | Currency | 19 … 20 | ±922 337 203 685 477, 5807 |

**3. Тип дата-время** предназначен для хранения даты и времени. Фактически для этой цели он использует вещественный формат.

**4.4** **Константы**

При разработке программы часто возникают ситуации, когда нужно использовать лишь одно конкретное значение переменной. В этом случае удобно использовать константы. Константы отличаются от переменных тем, что им присваивают определенное значение в разделе описаний. Поэтому в теле программы уже нет необходимости вводить операцию присвоения значений. Но, тем не менее, в теле программы менять значение константы можно.

Константы описываются после ключевого слова **const.** Тип у константы можно указы­вать, а можно и не указывать. Значение константы задается после знака равенства.

Например, **const** PI = 3.14;

Е:Real = 2.87;

Названия констант принято записывать прописными буквами, чтобы легко отличать их от переменных.

Помимо числовых констант, в Delphi существуют строковые и символьные константы.Символы имеют тип **Char**и записываются в одиночных кавычках: ‘Язык программирования Delphi’.

Например, **const** Prog = ‘Язык программирования Delphi’;

**4.5 Структура программного модуля**

Программа в Delphi записывается в файле программного модуля. Первоначально программный код содержит минимальный исходный текст (рис. 49).

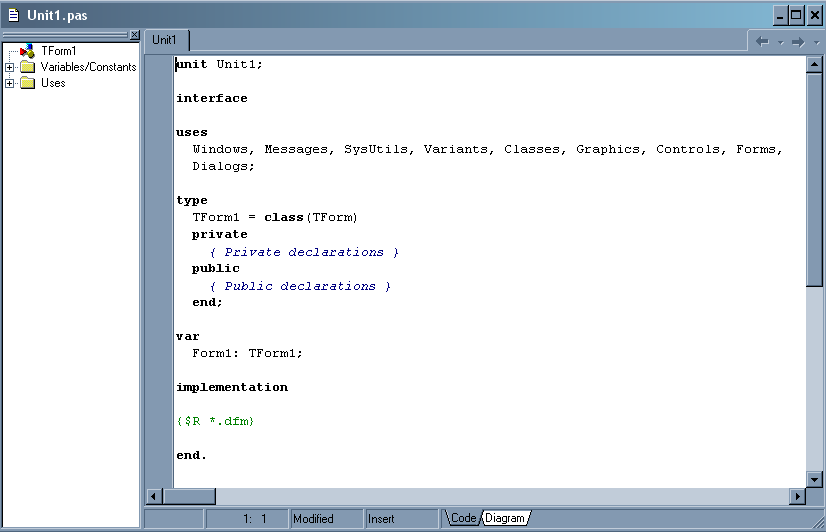


Рис. 49 Окно программного кода

Минимальный исходный текст программного кода имеет следующий вид:

//заголовок программного модуля

**unit** Unit1;

//зарезервированным словом **interface** открывается секция интерфейсных объявлений**.** Здесь описываются программные элементы: типы, классы, процедуры и функции, которые будут «видны» другим программным модулям.

**interface**

//после зарезервированного слова Uses перечисляются подключенные модули.

**uses**

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

//после зарезервированного слова Type в программе объявляются используемые типы.

**type**

**//**описание нового объекта TForm1, который происходит от объекта *TForm*. Это значит, что TForm1 будет обладать всеми возможностями *TForm.*

TForm1 = **class**(TForm)

//Далее записываются компоненты и события

**Private**

**//** после Private записываются закрытые данные объекта. Здесь можно описывать переменные и методы, доступные только для объекта TForm1.

*{ Private declarations }*

**Public**

// после Public записываются открытые данные объекта. Здесь можно описывать переменные и методы доступные из любого другого модуля

*{ Public declarations }*

**end;**

//объявление глобальных переменных.

**var**

Form1: TForm1;

//зарезервированным словом **implementation** открывается секция реализаций. В ней описывается работа программных элементов с помощью процедур и функций.

**implementation**

//директива, предназначенная для связывания программного модуля с описанием соответствующей ему формы (.dfm файла).

{$R \*.dfm}

// end с точкой означает конец модуля

**end.**

**§5** **Основные операторы языка программирования Delphi**

**5.1** **Оператор присваивания**

С оператором присваивания мы уже работали, когда задавали новые значения у свойств объектов в тексте программы (см. п. 3.5).

**Оператор присваивания** служит для присвоения нового значения некоторой переменной или свойству объекта.

Формат записи оператора присваивания:

<Имя переменной>: = <Выражение>;

или

<Объект. Свойство>: = <Новое значение>;

Оператор присваивания работает следующим образом. Переменная, стоящая в левой части оператора, становится равной той величине, которая находится справа. Типы этих величин должны совпадать.

Например:

а:= 1;

b:= 0.5;

Form1.Height:=500;

Эта запись означает, что переменной *а* присваивается число 1, переменной *b* – число 0.5. С помощью третьей операции присваивания свойство Height (Высота) у формы станет равным 500.

|  |
| --- |
| **Пример.** Составить программу, в которой по щелчку на кнопке меняются её размеры. |

Поместим на форму компонент TButton. С помощью окна Инспектора объектов установим размеры кнопки: Height (Высота) – 80, Wight (Ширина) – 160. Графический интерфейс окна будет иметь следующий вид (рис. 50):

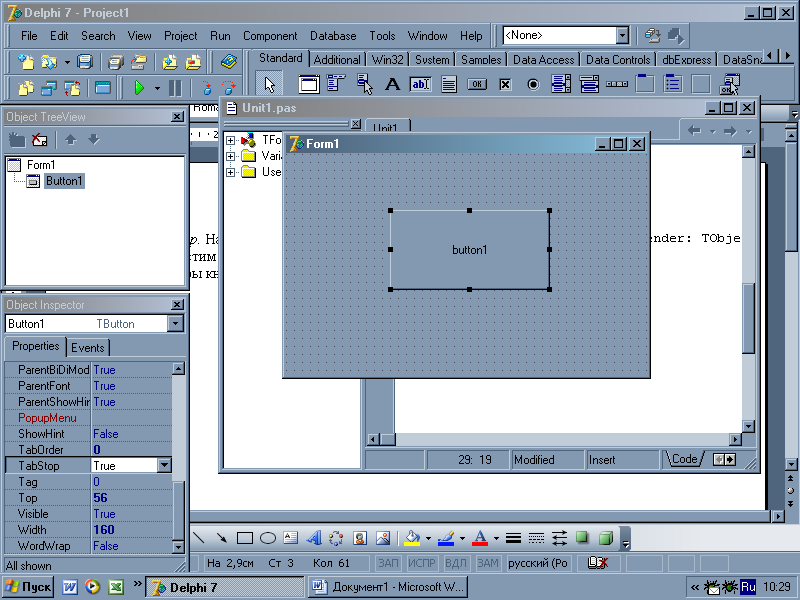


Рис. 50 Окно с кнопкой

Щелкнем дважды по созданной кнопке, при этом в коде программы откроется обработчик события OnClick.

Чтобы изменить размеры кнопки нужно присвоить свойствам Height и Wight компонента Button1 новые значения. Текст программы в процедуре будет следующий:

|  |
| --- |
| **Procedure** TForm1.Button1Click(Sender: TObject); |
| **Begin** |
| Button1.Height:=30; |
| Button1.Width:=80; |
| **end;** |

[**Задания**](#оператор_присваивания)

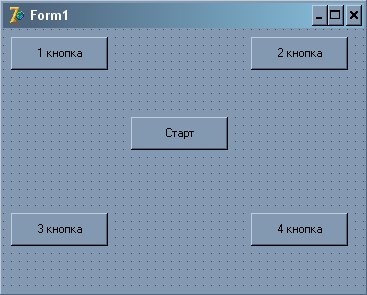
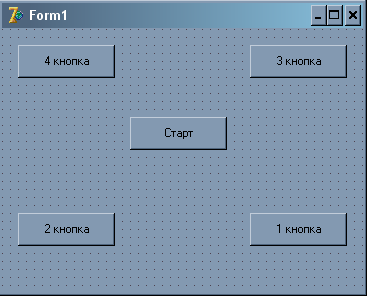
1. Создать проект, в котором по щелчку на кнопке «Старт» кнопка с заголовком «Учебная» меняет свое расположение на форме:

а) левый верхний угол на правый верхний;

б) левый верхний угла на левый нижний;

в) левый верхний угла на правый нижний.

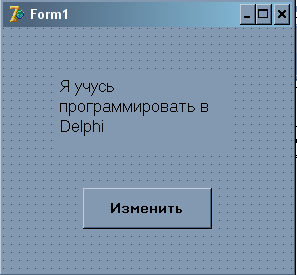
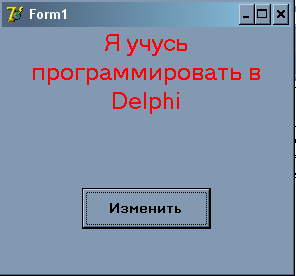
1. Написать программу, в которой используется следующий сценарий: на форму помещаются 5 кнопок (рис. 51а); при щелчке на центральной кнопке «Старт» меняются местами кнопки 1 и 4, 2 и 3 (рис. 51б).

а б

Рис. 51 Графический интерфейс проекта

1. Поместить на форму компоненты Надпись и Кнопка. В надписи набрать текст в несколько строк (рис. 52а). По щелчку на кнопке должен изменяться размер, цвет, начертание и выравнивание шрифта в надписи (рис. 52б).

а б

Рис. 52 а) Графический интерфейс проекта

б) Работающее приложение

1. Создать программу «Обмен значениями» между текстовыми полями.
2. Составить программу, которая по щелчку на кнопке копирует текст из текстового поля в надпись.
3. Составить программу, которая запрашивает у пользователя пароль и записывает его звездочками (свойство PasswordChar). При нажатии кнопки должен отобразиться текст пароля.



Рис. 53 Графический интерфейс окна

1. Создать проект, в котором меняется цвет формы при нажатии на соответствующую кнопку.

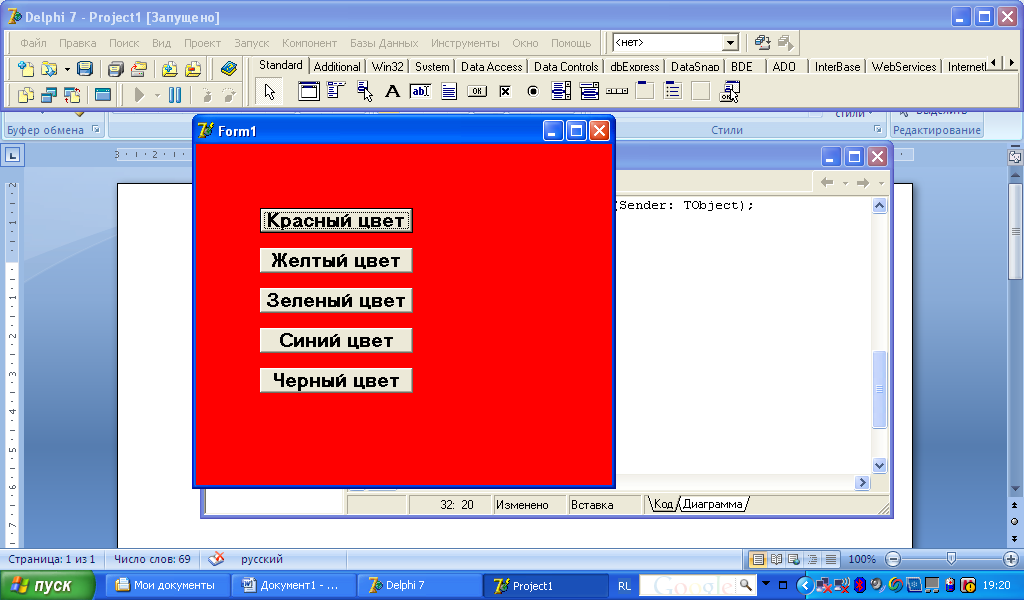


Рис. 54 Графический интерфейс проекта

1. Создать проект, в котором на форму помещается графический объект. Он должен перемещаться при каждом нажатии на соответствующие кнопки (вправо, влево, вниз и вверх) и увеличивать свои размеры при каждом нажатии на кнопку Расти (рис.55).

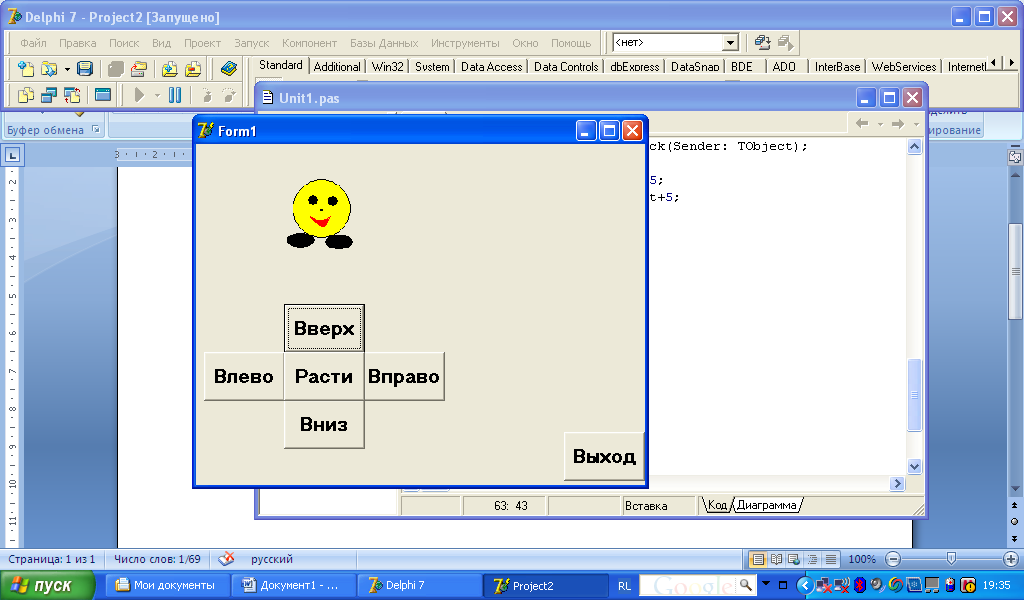


Рис. 55 Графический интерфейс проекта

1. Создать проект «Мозаика», в котором по щелчку на кнопке «Размешать» рисунок перемешивается случайным образом, при щелчке на кнопке «Собрать» восстанавливается целая картина.

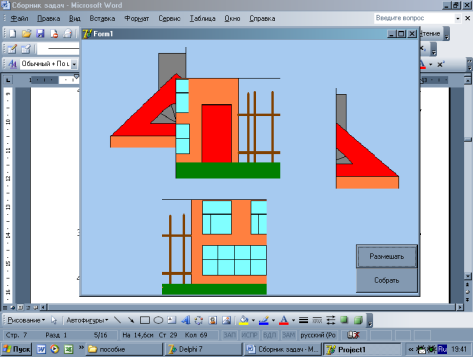
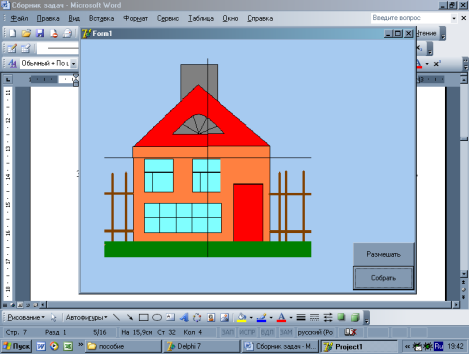
 

Рис. 56 Графический интерфейс проекта «Мозаика

**5.2.** **Логический тип данных. Создание проектов**

**с использованием оператора присваивания**

**и логических типов данных**

В Delphi многие свойства компонент заданы логическим типом данных Boolean. Переменная или свойство логического типа может принимать одно из двух значений: True или False. Если у свойства установлено значение True (истина), то это свойство выполняется. Если свойство имеет значение False (ложь), то оно не выполняется. Логический тип данных будет также применяться немного позже при рассмотрении условных операторов.

Рассмотрим пример проекта, в котором изменяется свойство логического типа.

|  |
| --- |
| **Пример.** Создать проект «Светофор». В этом проекте при щелчке на кнопке Запрещено должен загореться красный цвет светофора, при щелчке на кнопке Предупреждение – желтый цвет, при щелчке на кнопке Разрешено – зеленый цвет. |

В графическом редакторе Paint выполним изображение светофора, красного, желтого и зеленого кружков и сохраним все рисунки в разных файлах. На форму поместим:

* 3 кнопки с заголовками: Запрещено (Button1), Предупреждение (Button2) и Разрешено (Button3).
* 4 объекта TImage: в первый объект загрузим изображение светофора с не горящими кнопками (Image1), во второй – загрузим кружок красного цвета (Image 2), в третий – кружок желтого цвета (Image 3), в четвертый – кружок зеленого цвета (Image 4).
* Объект BitBtn с панели инструментов Additional, для которого установим свойство Kind (сорт) – bkClose (выход).

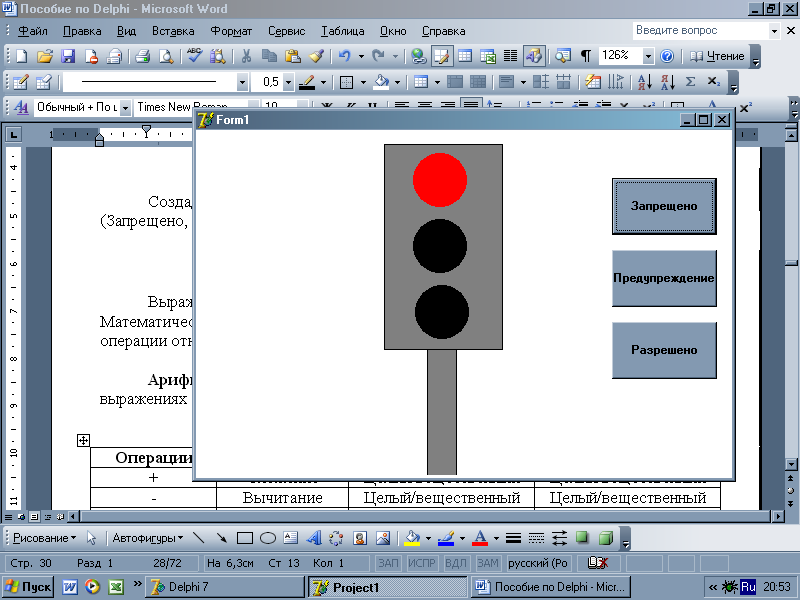
закрыть

Рис. 57 Графический интерфейс проекта «Светофор»

Установим для объектов Image 2, Image 3, Image 4 значение свойства Visible равным False, в результате в начале работы программы все цветные круги будут невидимыми.

Чтобы по щелчку на кнопке Запрещено загорелся красный сигнал светофора, нужно сделать объект Image2 видимым. Т.е. необходимо значение свойства Visible (Видимость) у Image2 изменить на True. Остальные объекты-круги при этом останутся невидимыми. Создадим следующий обработчик события для кнопки Button1:

|  |
| --- |
| **Procedure** TForm1.Button1Click(Sender: Tobject); |
| **Begin** |
| Image2.Visible:=true; |
| Image3.Visible:=false; |
| Image4.Visible:=false; |
| **end;** |

Для остальных кнопок текст события OnСlick будет подобным. Таким образом, полностью проект «Светофор» будет выглядеть следующим образом:

|  |
| --- |
| **Procedure** TForm1.Button1Click(Sender: Tobject); |
| **Begin** |
| Image2.Visible:=true; |
| Image3.Visible:=false; |
| Image4.Visible:=false; |
| **end;** |
| **Procedure** TForm1.Button2Click(Sender: Tobject); |
| **Begin** |
| Image2.Visible:= false; |
| Image3.Visible:= true; |
| Image4.Visible:=false; |
| **end;** |
| **Procedure** TForm1.Button3Click(Sender: Tobject); |
| **Begin** |
| Image2.Visible:= false; |
| Image3.Visible:= false; |
| Image4.Visible:= true; |
| **end;** |

[**Задания**](#оператор_присваивания)

1. Изменить проект «Светофор» таким образом, чтобы по щелчку на красном кружке появлялось сообщение «Движение запрещено», по щелчку на желтом кружке – «Приготовьтесь», по щелчку на зеленом кружке – «Движение разрешено».

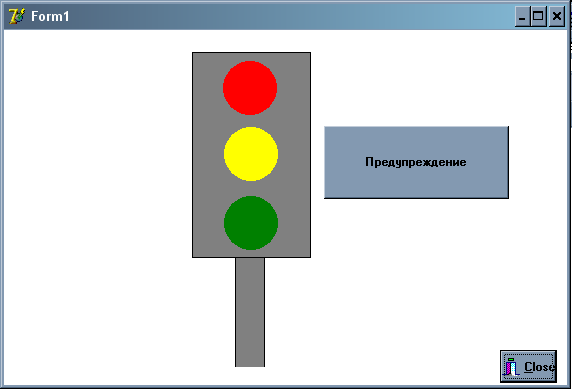
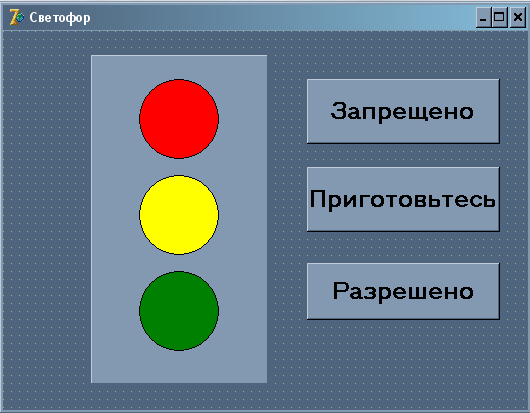


Рис. 58 Графический интерфейс проекта «Светофор»

1. Измените графический интерфейс проекта «Светофор» следующим образом: для создания лампочек светофора воспользуйтесь объектом TShape (Форма) с панели Additional. Чтобы залить компонент TShape цветом, используйте составное свойство Brush. Для придания объекту Shape формы окружности измените свойство Shape на stCircle.



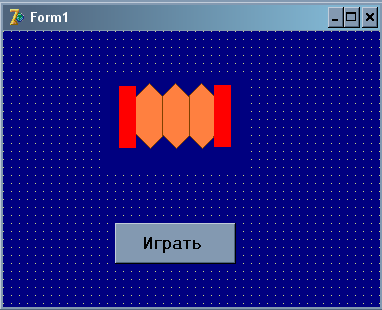
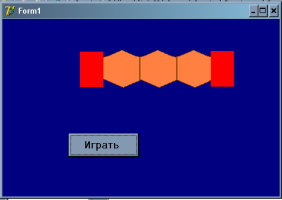
Компонент TPanel (Панель)

Компоненты TShape

Рис. 59 Графический интерфейс проекта «Светофор»

Сделайте вывод, для чего служат компоненты TShape и TPanel, используемые в этом проекте.

1. Создать программу, в которой используется следующий сценарий: на форму помещается изображение гармошки (рис. 60а); при щелчке на кнопке «Играть», гармошка растягивается (рис. 60б).

а б

Рис. 60 Графический интерфейс проекта

1. Создать программу «Видимость формы», в которой используется следующий сценарий: создаются 2 формы и на каждой из форм помещается кнопка; если щелкнуть по кнопке на первой форме, то появится вторая форма, а первая исчезает, и наоборот.
2. Создать проект «Телевизор», в котором при переключении каналов (щелчки на кнопках) появляются разные картинки. Кнопка ВЫКЛ – выключает все картинки.

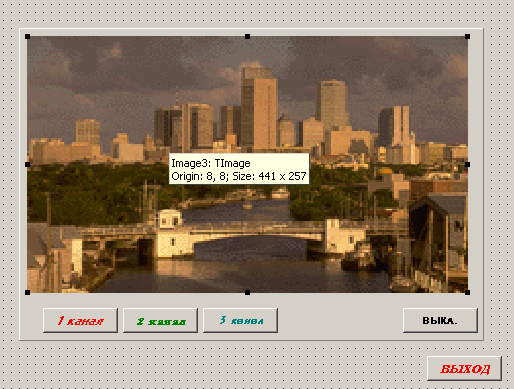


Рис. 61 Графический интерфейс проекта

**5.3** **Математические операции Delphi**

Для дальнейшего изучения языка Delphi рассмотрим математические операции. Они делятся на арифметические, логические операции и операции отношения.

**Арифметические операции** выполняют арифметические действия в выражениях над значениями переменных.

**Таблица 5. Арифметические операции**

|  |  |  |  |
| --- | --- | --- | --- |
| **Операция** | **Действия** | **Типы операндов** | **Тип результата** |
| + | Сложение | Целый/вещественный | Целый/вещественный |
| - | Вычитание | Целый/вещественный | Целый/вещественный |
| \* | Умножение | Целый/вещественный | Целый/вещественный |
| / | Деление | Целый/вещественный | Вещественный |
| Div | Деление нацело | Целый | Целый |
| Mod | Остаток от деления | Целый | Целый |

Особое внимание следует обратить на то, что тип результата деления – всегда вещественное число.

Оператор DIV позволяет получить целую часть результата деления одного числа на другое. Например, значение выражения 5 DIV 2 равно 2.

Оператор MOD (деление по модулю) позволяет получить остаток от деления одного числа на другое. Например, значение выражения 15 MOD 7 равно 1.

При вычислении значений выражений, как и в математике, учитывается порядок действий. Сначала выполняются операторы \*, /, DIV, MOD, а затем – операторы + и -. Для задания нужного порядка выполнения операций можно использовать скобки.

**Операции отношения** сравнивают два операнда и определяют истинность или ложность выражения. Результат операции отношения имеет тип Boolean, который принимает два значения: True (истина) и False (ложь).

**Таблица 6. Операции отношения**

|  |  |  |  |
| --- | --- | --- | --- |
| **Операция** | **Название** | **Выражение** | **Результат** |
| = | Равно | А=В | True, если А равно В |
| <> | Не равно | A<>B | True, если А не равно В |
| > | Больше | A>B | True, если А больше В |
| < | Меньше | A<B | True, если А меньше В |
| >= | Больше или равно | A>=B | True, если А больше или равно В |
| <= | Меньше или равно | A<=B | True, если А меньше или равно В |

**Логические операции** используются для создания составных операций отношения и часто используются, например, в условном операторе if…then…else.

**Таблица 7. Логические операции**

|  |  |  |
| --- | --- | --- |
| **Операция** | **Действие** | **Пример выражения** |
| Not | Логическое отрицание | Not A |
| And | Логическое и | A and B |
| Or | Логическое или | A or B |
| Xor | Исключающее или | A xor B |

**Стандартные математические функции Delphi**

В языке Delphi имеются следующие математические функции:

**Таблица 8. Математические функции**

|  |  |
| --- | --- |
| **Функция** | **Значение** |
| Abs (x) | Возвращает абсолютное значение (модуль) числа х |
| Sqrt (x) | Возвращает число, равное квадратному корню из х |
| Sqr (x) | Возвращает число, равное квадрату числа х |
| Sin (x) | Возвращает синус числа х, где х – угол в радианах |
| Cos (x) | Возвращает косинус числа х, где х – угол в радианах |
| Arctan (x) | Возвращает арктангенс числа х, где х – угол в радианах |
| Exp (x) | Возвращает число, равное е в степени х |
| Ln (x) | Возвращает число, равное натуральному логарифму от числа х |
| Pi | Число Пи |
| Random (x) | Возвращает случайное число от 0 до х-1. При использовании функции без параметров генерируются случайные числа от 0 до 1. |
| Trunc (x) | Возвращает число, равное целой части х (при округлении отбрасывается дробная часть). Усеченное число имеет тип LongInt. |
| Frac (x) | Возвращает число, равное дробной части числа х. Например, Frac(2.4) возвращает 0.4, Frac (3.99) возвращает 0.99. |
| Int (x) | Возвращает число, равное целой части числа х. Например, Int (2.4) возвращает 2.0, Int (0.99) возвращает 0.0. |
| Round (x) | Возвращает число, равное целой части числа х (округление происходит по правилам математики, т.е. к ближайшему целому). Процедура является универсальной, так как результат может быть как целым, так и вещественным числом. |
| Ord (x) | Определяет четность числа. Возвращает значение True, если число х нечетно. |

**5.4 Линейная алгоритмическая структура.**

**Ввод и вывод информации в Delphi.**

Программы, которые были ранее разработаны в системе Delphi, позволяли выводить информацию на экран компьютера или вводить информацию с клавиатуры с отображением ее на экране. Однако основными задачами, ради которых были разработаны компьютеры, являются задачи по обработке информации, в том числе задачи вычислительного характера. Поэтому дальнейший материал будет посвящен составлению программ для решения вычислительных задач.

Простейшими программами являются программы с линейной алгоритмической структурой, которая предполагает выполнение команд последовательно друг за другом. Программы линейной структуры с вычислениями обычно включают в себя следующие этапы:

1. ввод исходных данных;
2. вычисление требуемой величины;
3. вывод результата.

Соответствующая этим этапам блок-схема линейного алгоритма будет следующей:

Ввод исходных данных

Вычисление

Вывод результата

Рис. 62 Блок-схема линейного алгоритма

Простейшим оператором ввода и вывода является рассмотренный нами оператор присваивания. В системе программирования Delphi ввести исходные данные и вывести результат позволяют различные компоненты.

Для ввода данных удобно использовать компонент TEdit. Для этого необходимо с помощью мыши установить в текстовом поле курсор и ввести с клавиатуры текст. Вводимая строка отображается в свойстве Text компонента TEdit и воспринимается программой как набор символов. Цифровые символы можно преобразовать в числа с помощью соответствующих типу числа функций преобразования.

**Таблица 9. Функции преобразования**

|  |  |
| --- | --- |
| **Функция** | **Значение функции** |
| IntToStr (x) | Преобразует целое число в строку символов. |
| StrToInt (x) | Преобразует строку символов в целое число. |
| FloatToStr (x) | Преобразует вещественное число в строку символов. |
| StrToFloat (x) | Преобразует строку символов в вещественное число. |
| FloatToStrF(x, f, k, m) | Преобразует вещественное число в строку символов в соответствии с указанными параметрами: f – формат (способ изображения); k – точность (нужное количество цифр); m – количество цифр после десятичной точки |
| Chr (x) | Возвращает символ, соответствующий ASCII коду числа х. |

Для вывода информации можно использовать компонент TLabel. Чтобы увидеть результат работы программы, нужно присвоить свойству Caption компонента TLabel новое значение. Свойство Caption символьного типа. Поэтому выводимое число следует преобразовать в строку символов с помощью соответствующих типу числа функций преобразования.

Рассмотрим проект, в котором реализуется линейная алгоритмическая структура, и используются функции преобразования.

|  |
| --- |
| **Пример.**На рис. 62 приведен вид диалогового окна пересчета скорости из км/ч в м/с. Составить программу, которая по щелчку на кнопке выполняет этот перевод (исходная скорость выражается целым числом). |

Создадим графический интерфейс окна по образцу, изображенному на рис. 63. Для этого поместим на форму текстовое поле TEdit, кнопку TButton и 2 надписи TLabel. Удалим содержимое свойства Text у компонента Edit1. У надписи Label1 запишем в свойстве Caption сообщение «Введите скорость в км/ч». Для кнопки Button1 введем заголовок «Пересчет».

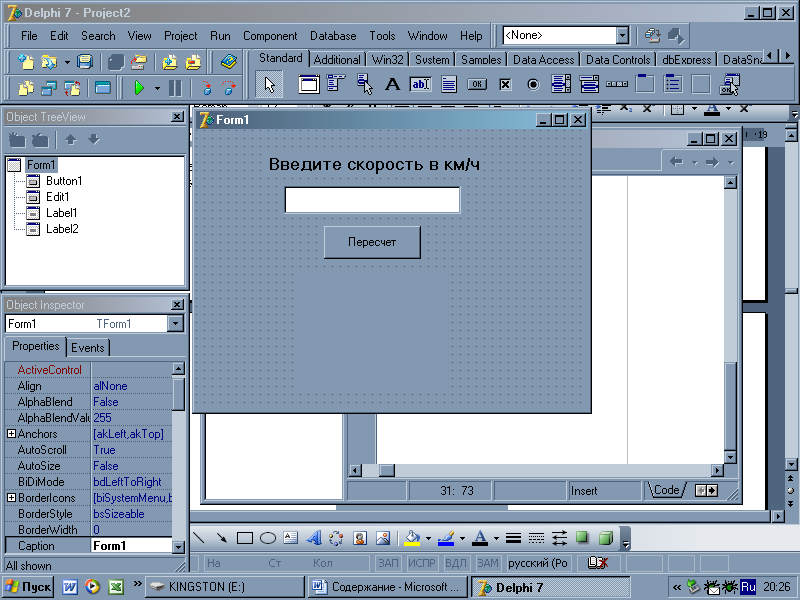
**

Рис. 63 Графический интерфейс окна

Исходные данные – скорость в км/ч – будем вводить в текстовое поле Edit1. Результат перевода выведем в надпись Label2.

В данном проекте можно выделить следующие этапы линейного алгоритма:

1. Ввести в поле Edit1 скорость в км/ч.
2. Перевести скорость из км/ч в м/с.
3. Вывести в надпись Label2 скорость в м/с.

Рассмотрим каждый этап линейного алгоритма.

1. Для обозначения исходной скорости в км/ч введем дополнительную переменную целого типа V1. Присвоим переменной V1 значение свойства Text компонента Edit1.

V1:=Edit1.Text;

Однако такая запись является неверной, так как типы данных до и после знака присваивания не совпадают: переменная V1 целого типа, а свойство Text компонента Edit1 имеет строковый тип данных. Чтобы далее производить вычисления с вводимыми в поле Edit1 цифрами, которые пока воспринимаются программой как текст, необходимо преобразовать их в числовой формат. В данном случае необходимо использовать функцию StrToInt (преобразует строку в целое число). Таким образом, текст команды ввода будет иметь следующий вид:

V1:= StrToInt (Edit1.Text);

2. Чтобы перевести скорость из км/ч в м/с нужно умножить значение скорости на 1000 и разделить 3600 (в 1 километре – 1000 метров, в 1 часе – 3600 секунд). Введем еще одну переменную V2 для обозначения скорости в м/с. Таким образом, для расчета скорости в м/с используется следующий оператор:

V2:=V1\*1000/3600;

Переменная V2 будет вещественного типа, так как в выражении для ее вычисления имеется операция деления.

3. Для вывода полученной скорости предназначен компонент Label2. Поэтому необходимо свойству Caption компонента Label2 присвоить значение переменной V2:

Label2.Caption:= V2;

Данная запись опять нарушает соответствие типов данных, так как в свойстве Caption у надписи могут отображаться только данные символьного типа, а переменная V2 – это вещественное число. Поэтому необходимо преобразовать переменную V2 вещественного типа в строку функцией FloatToStr (преобразовать вещественное число в строку). Текст команды вывода будет выглядеть следующим образом:

Label2.Caption:= FloatToStr(V2);

При выводе результата часто к полученным значениям добавляют комментарии. Например, в данном проекте можно к значению скорости добавить запись её единиц измерения ‘м/с’. В программе эта запись указывается в одиночных кавычках (апострофах), т.к. является символьной константой. Таким образом, вывод результата может иметь следующий вид: Label2.Caption:= FloatToStr(V2)+ ‘м/с’;

Так как мы ввели дополнительные переменные V1 и V2, то нужно указать их тип в разделе описаний переменных. Все переменные, используемые в Delphi, подразделяются на два основных вида. Это глобальныепеременные, которые могут использоваться во всей программе, и локальныепеременные, которые можно использовать только внутри определенной процедуры.

В данном случае опишем переменные как локальные внутри процедуры **TForm1.Button1Click**. Для этого после заголовка процедуры введем раздел описания переменных, который открывается зарезервированным словом **var**. Описание переменных будет выглядеть так:

**var** V1: integer; V2: real;

Таким образом, текст обработчика события для кнопки Пересчет будет выглядеть следующим образом:

|  |
| --- |
| **procedure** TForm1.Button1Click(Sender: TObject); |
| **var** V1: integer; V2: real; |
| **begin** |
| V1:= StrToInt (Edit1.Text); |
| V2:= V1\*1000/3600; |
| Label2.Caption:= FloatToStr(V2) + ‘м/с’; |
| **end;** |

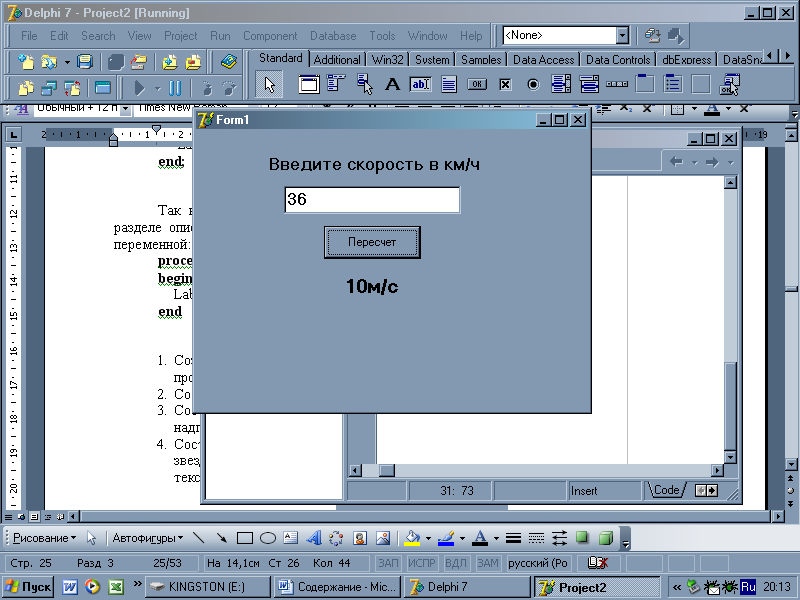
****

Рис. 64 Работающее приложение

Можно было написать программу и без ввода дополнительных переменных:

|  |
| --- |
| **procedure** TForm1.Button1Click(Sender: TObject); |
| **begin** |
| Label2.Caption:= FloatToStr((StrToInt (Edit1.Text))\*1000/3600) + ‘м/с’; |
| **end** |

Чтобы программа была готова к вводу новых данных, нужно очистить текстовое поле и поместить в него курсор. В Delphi для этого применяются методы:

* **Clear (Очистить)**
* **SetFocus (Передать фокус ввода)**

Учитывая эти методы, текст программы изменится на следующий:

|  |
| --- |
| **procedure** TForm1.Button1Click(Sender: TObject); |
| **var** V1: integer; V2: real; |
| **begin** |
| V1 := StrToInt (Edit1.Text); |
| V2:= V1\*1000/3600; |
| Label2.Caption:= FloatToStr(V2) + ‘м/с’; |
| Edit1.Clear; |
| Edit1.SetFocus; |
| **end;** |

**5.5** **Создание проектов с использованием**

**ввода/вывода информации**

Создадим проект «Инженерный калькулятор», позволяющий производить вычисления арифметических действий и стандартных математических функций над целыми числами.

Спроектируем графический интерфейс калькулятора подобно тому, как изображено на рис. 65.



Текстовые поля Edit1, Edit2 и Edit3

Надписи

Label1 и Label2

Рис.65 Графический интерфейс проекта «Инженерный калькулятор»

Для ввода исходных данных (двух чисел или аргумента функции) используем три текстовых поля (Edit1, Edit2 и Edit3). Результаты арифметических действий и стандартных математических функций будем выводить в надписи (Label1 и Label2). Чтобы выполнить какую-либо операцию нужно нажать соответствующую кнопку. Поэтому для каждой кнопки калькулятора опишем событие щелчка OnClick.

Для удобства введем дополнительные переменные: а – первое число, b – второе число, с – аргумент функции, S – результат действия или значение функции.

Начнем с описания обработчика события у кнопки Button1, при щелчке на которой выполняется операция сложения. Складываемые числа вводятся в текстовые поля Edit1 и Edit2, они хранятся в переменных *а* и *b*. Эти значения воспринимаются программой как набор символов, поэтому их нужно преобразовать в числа с помощью функции StrToInt. Результат сложения выводится в свойство Caption компонента Label1 и хранится в переменной S. Так как свойство Caption символьного типа, то результат действия следует перевести в строку с помощью функции преобразования IntToStr.

Так как в процедуре были использованы дополнительные переменные, то необходимо обязательно задать их тип в разделе описания переменных.

Текст обработчика события у Button1 будет следующим:

|  |
| --- |
| **procedure** TForm1.Button1Click(Sender: TObject); |
| **var** a, b, S:integer; |
| **begin** |
| a:=StrToInt(Edit1.Text); |
| b:=StrToInt(Edit2.Text); |
| S:=a + b; |
| Label1.Caption:=IntToStr(S); |
| **end;** |

Обработчики событий у других кнопок будет подобными. Для кнопки Button2, выполняющей вычитание, текст программы обработчика события будет выглядеть таким образом:

|  |
| --- |
| **procedure** TForm1.Button2Click(Sender: TObject); |
| **var** a, b, S:integer; |
| **Begin** |
| a:=StrToInt(Edit1.Text); |
| b:=StrToInt(Edit2.Text); |
| S:=a - b; |
| Label1.Caption:=IntToStr(S); |
| **end;** |

Для кнопки Button3, выполняющей умножение, обработчик события примет вид:

|  |
| --- |
| **procedure** TForm1.Button3Click(Sender: TObject); |
| **var** a, b, S:integer; |
| **begin** |
| a:=StrToInt(Edit1.Text); |
| b:=StrToInt(Edit2.Text); |
| S:=a\*b; |
| Label1.Caption:=IntToStr(S); |
| **end;** |

Кнопка Button4 делит число, вводимое в поле Edit1 на число, вводимое в поле Edit2. Результат деления является вещественным числом. Поэтому для преобразования числа в строку используем функцию FloatToStr. Текст обработчика события для кнопки Button4 будет следующим:

|  |
| --- |
| **procedure** TForm1.Button4Click(Sender: TObject); |
| **var** a, b: integer; S: real; |
| **begin** |
| a:=StrToInt(Edit1.Text); |
| b:=StrToInt(Edit2.Text); |
| S:=a/b; |
| Label1.Caption:= FloatToStr(S); |
| **end;** |

Кнопка Button5 определяет остаток от деления числа, введенного в поле Edit1 на число, введенное в поле Edit2. Кнопка Button6 показывает целую часть при делении первого числа на второе. Текст обработчиков событий для этих кнопок примет следующий вид:

|  |
| --- |
| **procedure** TForm1.Button5Click(Sender: TObject); |
| **var** a, b, S: integer; |
| **begin** |
| a:=StrToInt(Edit1.Text); |
| b:=StrToInt(Edit2.Text); |
| S:=a **mod** b; |
| Label1.Caption:=IntToStr(S); |
| **end;** |
| **procedure** TForm1.Button6Click(Sender: TObject); |
| **var** a, b, S:integer; |
| **begin** |
| a:=StrToInt(Edit1.Text); |
| b:=StrToInt(Edit2.Text); |
| S:=a **div** b; |
| Label1.Caption:=IntToStr(S); |
| **end;** |

Остальные кнопки служат для вычисления стандартных математических функций для числа, введенного в текстовое поле Edit3. Кнопка Button7 определяет абсолютное значение или модуль числа. Для вычисления абсолютного значения используем стандартную математическую функцию abs (см. таблицу №8). Текст обработчика события для кнопки Button7 будет выглядеть таким образом:

|  |
| --- |
| **procedure** TForm1.Button7Click(Sender: TObject); |
| **var** с, S:integer; |
| **begin** |
| c:=StrToInt(Edit3.Text); |
| S:=abs(c); |
| Label2.Caption:= IntToStr(S); |
| **end;** |

Для кнопки Button8, извлекающей корень из числа, текст программы обработчика события будет следующим:

|  |
| --- |
| **procedure** TForm1.Button8Click(Sender: TObject); |
| **var** с: integer;S:real; |
| **begin** |
| c:=StrToInt(Edit3.Text); |
| S:=sgrt(c); |
| Label2.Caption:= FloatToStr (S); |
| **end;** |

Кнопка Button9 возводит введенное число в квадрат.

|  |
| --- |
| **procedure** TForm1.Button9Click(Sender: TObject); |
| **var** с, S: integer; |
| **Begin** |
| c:=StrToInt(Edit3.Text); |
| S:=sgr(c); |
| Label2.Caption:= IntToStr (S); |
| **end;** |

Кнопки Button10 и Button11 вычисляют соответственно синус и косинус числа, введенного в поле Edit3. Кнопка Button12 определяет натуральный логарифм введенного числа. Кнопка Button13 возводит число е в степень, заданную аргументом. Текст обработчиков событий у этих кнопок будет следующим:

|  |
| --- |
| **procedure** TForm1.Button10Click(Sender: TObject); |
| **var** с: integer;S:real; |
| **Begin** |
| c:=StrToInt(Edit3.Text); |
| S:=sin(c); |
| Label2.Caption:= FloatToStr (S); |
| **end;** |
| **procedure** TForm1.Button11Click(Sender: TObject); |
| **var** с: integer;S:real; |
| **Begin** |
| c:=StrToInt(Edit3.Text); |
| S:=cos(c); |
| Label2.Caption:= FloatToStr (S); |
| **end;** |
| **procedure** TForm1.Button12Click(Sender: TObject); |
| **var** с: integer;S:real; |
| **begin** |
| c:=StrToInt(Edit3.Text); |
| S:=ln(c); |
| Label2.Caption:= FloatToStr (S); |
| **end;** |
| **procedure** TForm1.Button13Click(Sender: TObject); |
| **var** с: integer;S:real; |
| **Begin** |
| c:=StrToInt(Edit3.Text); |
| S:=exp(c); |
| Label2.Caption:= FloatToStr (S); |
| **end;** |

Текст проекта «Инженерный калькулятор» можно записать и без использования дополнительных переменных. В этом случае запись программы получается более компактной.

|  |
| --- |
| **procedure** TForm1.Button1Click(Sender: TObject); |
| **Begin** |
| Label1.Caption:=IntToStr(StrToInt(Edit1.Text)+StrToInt(Edit2.Text)); |
| **end;** |
| **procedure** TForm1.Button2Click(Sender: TObject); |
| **Begin** |
| Label1.Caption:=IntToStr(StrToInt(Edit1.Text)-StrToInt(Edit2.Text)); |
| **end;** |
| **procedure** TForm1.Button3Click(Sender: TObject);  **begin**  Label1.Caption:=IntToStr(StrToInt(Edit1.Text)\*StrToInt(Edit2.Text));  **end;** |
| **procedure** TForm1.Button4Click(Sender: TObject);  **begin**  Label1.Caption:=FloatToStr(StrToInt(Edit1.Text)/StrToInt(Edit2.Text));  **end;** |
| **procedure** TForm1.Button5Click(Sender: TObject);  **begin**  Label1.Caption:=IntToStr(StrToInt(Edit1.Text) **mod** StrToInt(Edit2.Text));  **end;** |
| **procedure** TForm1.Button6Click(Sender: TObject);  **begin**  Label1.Caption:=IntToStr(StrToInt(Edit1.Text) **div** StrToInt(Edit2.Text));  **end;** |
| **procedure** TForm1.Button7Click(Sender: TObject);  **begin**  Label2.Caption:= IntToStr(abs(StrToInt(Edit3.Text)));  **end;** |
| **procedure** TForm1.Button8Click(Sender: TObject);  **begin**  Label2.Caption:=FloatToStr(sqrt(StrToInt(Edit3.Text)));  **end;** |
| **procedure** TForm1.Button9Click(Sender: TObject);  **begin**  Label2.Caption:=IntToStr(sqr(StrToInt(Edit3.Text)));  **end;** |
| **procedure** TForm1.Button10Click(Sender: TObject);  **begin**  Label2.Caption:= FloatToStr(sin(StrToInt(Edit3.Text)));  **end;** |
| **procedure** TForm1.Button11Click(Sender: TObject);  **begin**  Label2.Caption:= FloatToStr(cos(StrToInt(Edit3.Text)));  **end;** |
| **procedure** TForm1.Button12Click(Sender: TObject);  **begin**  Label2.Caption:= FloatToStr(ln(StrToInt(Edit3.Text)));  **end;** |
| **procedure** TForm1.Button13Click(Sender: TObject);  **begin**  Label2.Caption:= FloatToStr(exp(StrToInt(Edit3.Text)));  **end;** |
| **End.** |

[**Задания**](#ввод_вывод)

1. Изменить проект «Инженерный калькулятор» следующим образом:

* Добавить кнопку возведения числа в заданную степень  и создать для нее обработчик события. Для возведения числа *х* в степень *y* используется формула: xy = ey∙ln(x).
* Изменить проект так, чтобы вводимые числа были вещественного типа.
* Запрограммировать кнопку Сброс. При нажатии на эту кнопку должны обнуляться исходные данные и результат.



Рис. 66 Графический интерфейс проекта «Инженерный калькулятор»

1. Составить проект «Параллелепипед», в котором требуется ввести длину, ширину и высоту параллелепипеда. В проекте должны вычисляться объем и площадь полной поверхности параллелепипеда. Для вывода результатов вычисления используйте компоненты TEdit.

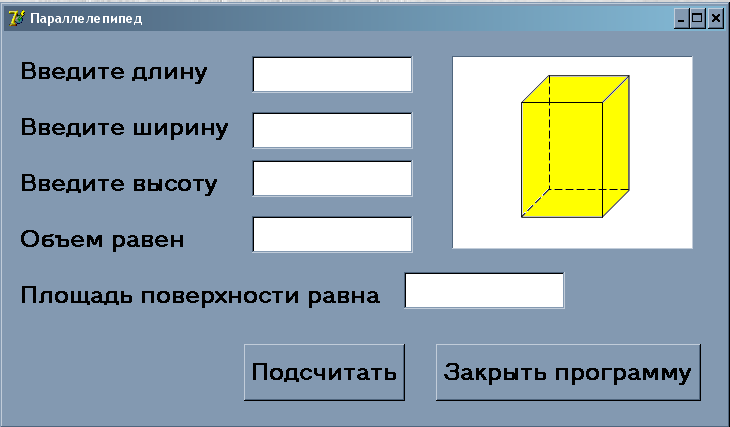


Рис. 67 Графический интерфейс проекта «Параллелепипед»

1. Составить проект, в котором нужно определить среднее арифметическое и среднее геометрическое двух чисел, введенных с клавиатуры. Среднее арифметическое двух чисел *а* и *b* рассчитывается по формуле: , среднее геометрическое двух чисел *а* и *b* – по формуле: .

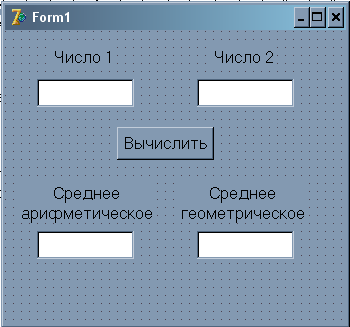


Рис. 68 Графический интерфейс проекта

1. Написать программу, которая вычисляет силу тока в электрической цепи. По закону Ома сила тока равна , где U – напряжение, R – сопротивление.

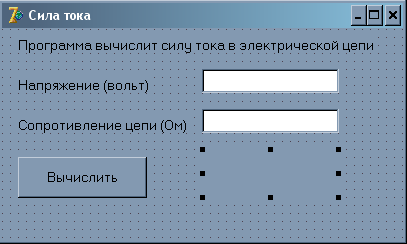


Рис. 69 Графический интерфейс проекта

1. Составить проект вычисления гипотенузы и площади прямоугольного треугольника, если известны длины его катетов. Гипотенуза прямоугольного треугольника вычисляется по формуле , где а и b – длины катетов. Площадь прямоугольного треугольника равна .

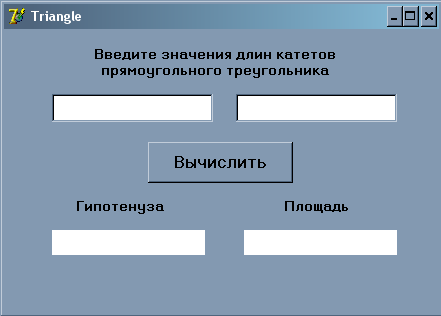


Рис. 70 Графический интерфейс проекта

1. Составить проект «Поездка», в котором вычисляется стоимость поездки до дачи и обратно. Требуется ввести расстояние до дачи, количество бензина, которое потребляет машина в среднем на 100 км и стоимость 1 л. бензина.

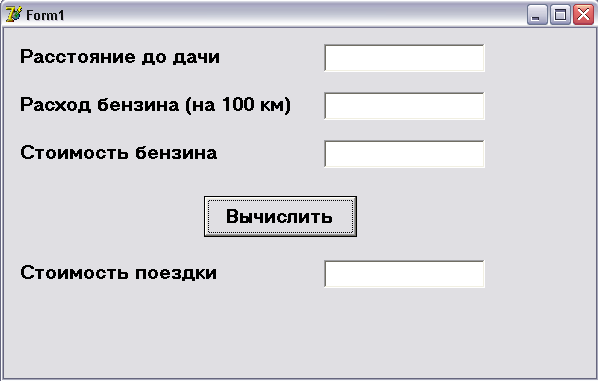


Рис. 71 Графический интерфейс проекта «Поездка»

1. Написать программу, которая вычисляет силу тока в электрической цепи. Цепь состоит из двух параллельно соединенных сопротивлений, общее сопротивление определяется по формуле: . Силу тока найти по закону Ома.

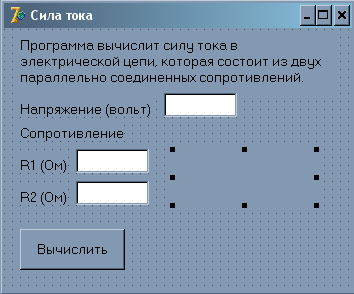


Рис. 72 Графический интерфейс проекта

1. Написать программу, которая вычисляет скорость (км/ч), с которой бегун бежал дистанцию. Скорость находится по формуле: , где S – длина дистанции, t – время в секундах.

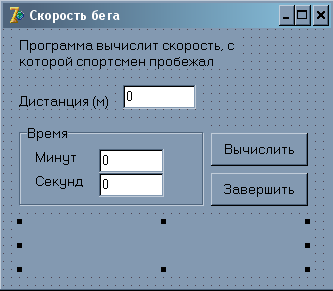


Рис. 73 Графический интерфейс проекта

1. Составить проект «Цилиндр», который вычисляет объем и площадь поверхности цилиндра по вводимым значениям высоты и радиуса основания.
2. Составьте проект «Вычисление работы», совершенной при равномерном подъеме груза массой m кг на высоту h м. Ускорение свободного падения описать как константу g = 9,81. Работа силы тяжести равна: A = mgh.
3. Составить программу, которая выполняет операции умножения и деления двух обыкновенных дробей.
4. Создать проект «Конвектор», определяющий размер денежного вклада через t лет после его внесения, если заданы размер начального взноса n и банковский процент b. Для расчета размера вклада использовать формулу: S = n∙(1+b)t.
5. Составить проект, в котором требуется ввести х и вычислить значение функции . Результат вычисления вывести в отдельном диалоговом окне. Для вывода информации в отдельном окне используется процедура ShowMessage().

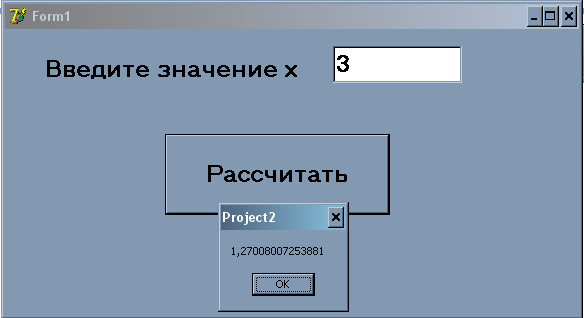


Рис. 74 Работающее приложение

1. Создать проект «Начисление заработной платы». Требуется ввести фамилию, имя, отчество, количество отработанных часов и стоимость одного часа. Компьютер выдает в отдельном диалоговом окне результат вида «Уважаемый, …, Вы заработали … рублей». Чтобы найти зарплату, нужно количество отработанных часов умножить на стоимость одного часа.

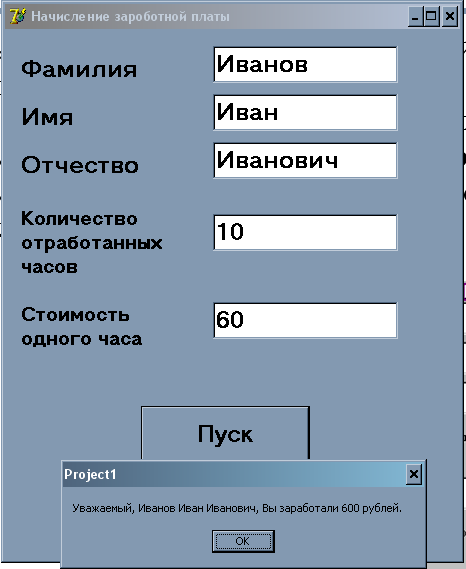


Рис. 75 Окно проекта «Начисление заработной платы»

1. Составить проект, который вычисляет целую часть от деления вводимого числа на 5 и остаток от деления этого же числа на 5.
2. Создать проект «Банкомат», в котором требуется ввести желаемую сумму. Банкомат должен рассчитать, сколько купюр по 1000, 500, 100 и 50 рублей ему нужно выдать.

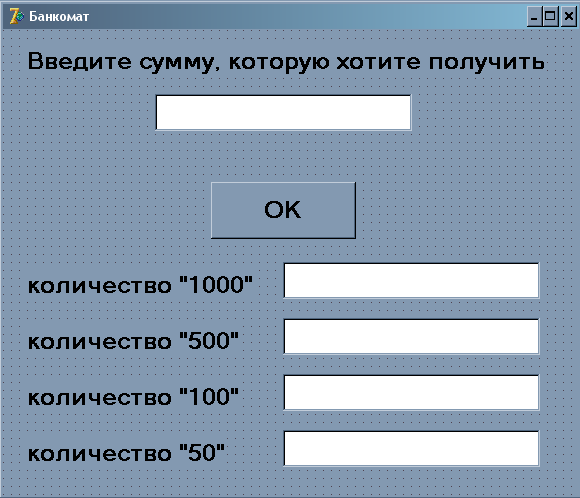


Рис. 76 Графический интерфейс проекта «Банкомат»

1. Создать проект «Хронометр», который измеряет интервал, прошедший от одного момента времени до другого. Моменты времени задаются в часах и минутах. Интервал между этими моментами также вычисляется в часах и минутах.

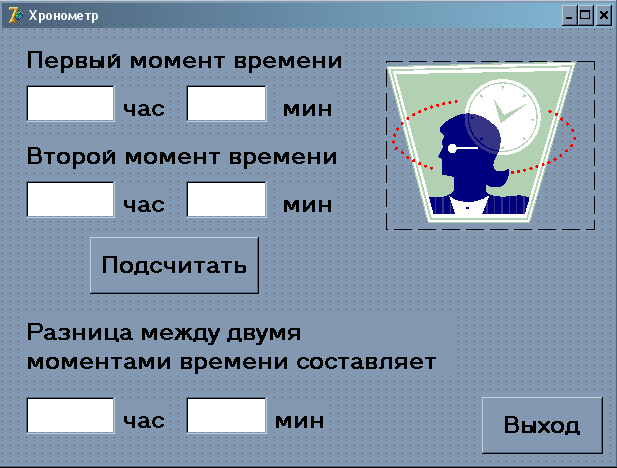


Рис. 77 Графический интерфейс проекта «Хронометр»

* 1. **5.6 Алгоритмическая структура «Ветвление».**

**Условный оператор.**

В отличие от линейных алгоритмов, в которых команды выполняются последовательно одна за другой, в алгоритмическую структуру «ветвление» входит условие, в зависимости от выполнения или невыполнения которого реализуется та или иная последовательность команд.

Блок-схема алгоритмической структуры «ветвление» имеет следующий вид:

Условие

Оператор 1

Оператор 2

Да

Нет

Рис. 78 Блок-схема алгоритмической структуры «ветвление»

Средством ветвления в программеявляется условный оператор. **Условный оператор** проверяет заданное условие и в зависимости от результата проверки выполняет то или иное действие.

Структура условного оператора имеет следующий вид:

**If** <условие> **then** <оператор 1> **else** <оператор 2>,

где If, then, else – зарезервированные слова (если, то, иначе);

<условие> - произвольное выражение логического типа;

<оператор 1>, <оператор 2> - любые операторы языка Delphi.

Условный оператор if работает следующим образом. Сначала проверяется условие. Условие – выражение логического типа, оно может принимать следующие значения: True – истина или False – ложь. Если условие истинно, то выполняется оператор, следующий за словом then. На этом выполнение оператора if заканчивается, то есть инструкции, следующие за else, не будут выполнены. Если условие ложно, то выполняется оператор, следующий за словом else.

На практике часто используется укороченная структура условного оператора, в которой не предусмотрено действие при невыполнении условия:

**If** <условие> **then** <оператор 1>

Рассмотрим пример проекта, в котором применяется условный оператор.

|  |
| --- |
| **Пример.**Составить проект, в котором требуется ввести два вещественных числа. Программа сравнивает эти числа и выводит результат сравнения. |

Создадим графический интерфейс окна подобно тому, как изображено на рис. 79. Для ввода чисел используем два текстовых поля Edit1 и Edit2. Сравнение осуществляется при нажатии на кнопку Button1. Результат сравнения будем выводить в надпись Label3.

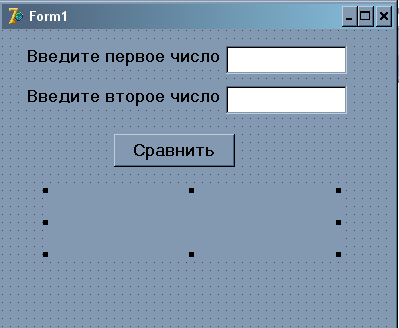


Рис. 79 Окно проекта сравнения двух чисел.

Введем следующие дополнительные переменные: а – первое число, b – второе число. Если первое число больше второго, то нужно присвоить свойству Caption y надписи строчку ‘Первое число больше второго’. Если это условие не выполняется, то в надпись выводится строка ‘Второе число больше первого’.

Блок-схема алгоритма сравнения двух чисел будет следующей:

Ввод а, b

a>b

Второе число больше первого

Первое число больше второго

Да

Нет

Рис. 80 Блок-схема условного оператора

Условный оператор в данном проекте запишется следующим образом:

**If** a>b **then** Label3.Caption:=’Первое число больше второго’

**else** Label3.Caption:=’Второе число больше первого’;

Текст обработчика события OnClick для кнопки «Сравнить» будет таким:

|  |
| --- |
| **procedure** TForm1.Button1Click(Sender: TObject); |
| **var** a,b: real; |
| **begin** |
| a:= StrToFloat (Edit1.Text); |
| b:= StrToFloat (Edit2.Text); |
| **if** a>b **then** Label3.Caption:=’Первое число больше второго’ |
| **else** Label3.Caption:=’Второе число больше или равно  первому’; |
| **end;** |

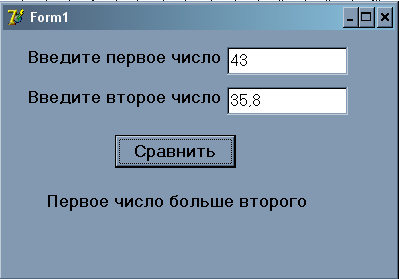


Рис. 81 Работающее приложение

Данный проект можно реализовать и с помощью укороченной структуры условного оператора:

|  |
| --- |
| **procedure** TForm1.Button1Click(Sender: TObject); |
| **var** a,b: real; |
| **begin** |
| a:= StrToFloat (Edit1.Text); |
| b:= StrToFloat (Edit2.Text); |
| **if** a>b **then** Label3.Caption:=’Первое число больше второго’; |
| **if** a<b **then** Label3.Caption:=’Второе число больше первого’; |
| **if** a=b **then** Label3.Caption:=’Числа равны’; |
| **end;** |

Если в зависимости от выполнения или невыполнения условия необходимо выполнить несколько операторов, то их заключают в операторные скобки begin … end.

**If** <условие> **then**

**Begin**

<оператор 1>;

<оператор 2>;

….

<оператор n>;

**End**

«Оператор» в условном операторе может быть любым, в том числе и условным оператором. В таких случаях возникают конструкции со вложенными условиями типа:

**If** <условие1> **then**

**If** <условие2> **then**

**If** <условие3> **then**

<оператор>;

В таких случаях удобно использовать составные условия с использованием логических операций.

|  |
| --- |
| **Пример.**Составить проект, в котором требуется ввести три целых числа. Программа должна опередить максимальное из этих чисел. |

Создадим графический интерфейс окна, подобно тому, как изображено на рис. 82.

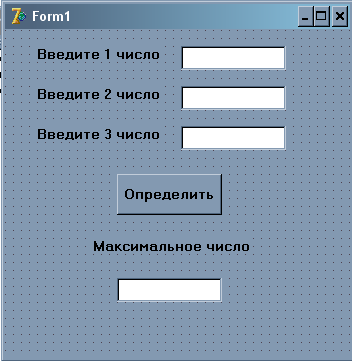


Рис. 82 Графический интерфейс проекта

Введем дополнительные переменные: X – 1 число, Y – 2 число, Z – 3 число. Текст обработчика события OnClick для кнопки будет следующим:

|  |
| --- |
| **procedure** TForm1.Button1Click(Sender: TObject); |
| **var** X, Y, Z: integer; |
| **begin** |
| X:=StrToInt(Edit1.Text); |
| Y:=StrToInt(Edit2.Text); |
| Z:=StrToInt(Edit3.Text); |
| **if** (X>Y) **and** (X>Z) **then** Edit4.Text:=IntToStr(X) + ‘- максимальное число’; |
| **if** (Y>X) **and** (Y>Z) **then** Edit4.Text:=IntToStr(Y) + ‘- максимальное число’; |
| **if** (Z>X) **and** (Z>Y) **then** Edit4.Text:=IntToStr(Z) + ‘- максимальное число’; |
| **end;** |

**5.7 Создание проектов с использованием**

**условного оператора и генератора случайных чисел**

|  |
| --- |
| **Пример.** Создать проект «Ромашка», в котором при щелчке на выбранном лепестке в надписи отображается случайное предсказание, и данный лепесток становится невидимым. |

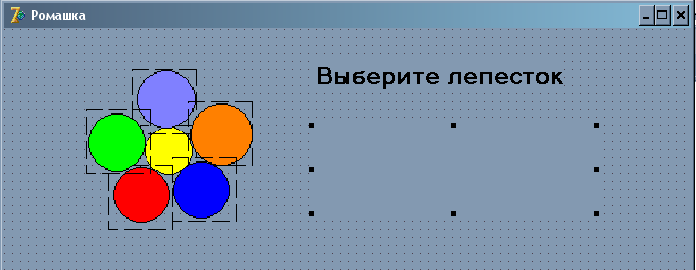


Рис. 83 Графический интерфейс проекта «Ромашка»

Создадим графический интерфейс проекта по образцу (рис. 83). Лепестки ромашки поместим в компоненты Image1-Image5.

В данном проекте необходимо ввести переменную целого типа, которая будет принимать случайные значения в определенном диапазоне.

Для задания случайных величин в Delphi используется генератор случайных чисел, который запускается процедурой Randomize. Эта процедура делает выбор случайных чисел еще более случайным. Случайное число в диапазоне от 0 до (Х-1) возвращает функция Random (Х). При этом сама граница в диапазон не входит. Например, функция Random(10) может выдать случайное число в интервале от 0 до 9, а число 10 не выдаст никогда.

Генератор случайных чисел запустим при открытии формы. Для этого на вкладке Events (События) Инспектора объектов у компонента Form1 выберем событие OnCreate и дважды щелкнем по нему. В появившемся обработчике события TForm1.FormCreate включаем генератор случайных чисел процедурой Randomize.

|  |
| --- |
| **procedure** TForm1.FormCreate(Sender: TObject); |
| **begin** |
| Randomize; |
| **end;** |

Введем переменную х, которая будет принимать одно из пяти случайных значений (по количеству лепестков ромашки). Чтобы эта переменная могла принимать пять случайных значений, ей необходимо выполнить следующее присвоение:

х:= random(5);

Если переменная х окажется равной 0, то в надписи должно появиться первое предсказание:

**if** x=0 **then** Label1.Caption:='Счастье';

Аналогично задаются остальные предсказания.

Чтобы затем сделать компонент Image1 невидимым нужно добавить оператор:

Image1.Visible:=false;

Обработчик события щелка OnClick у компонента Image1 будет следующим:

|  |
| --- |
| **procedure** TForm1.Image1Click(Sender: TObject); |
| **var** x:integer; |
| **begin** |
| х:=random(5); |
| **if** x=0 **then** Label1.Caption:='Счастье'; |
| **if** x=1 **then** Label1.Caption:='Любовь'; |
| **if** x=2 **then** Label1.Caption:='Удача'; |
| **if** x=3 **then** Label1.Caption:='Богатство'; |
| **if** x=4 **then** Label1.Caption:= 'Здоровье'; |
| Image1.Visible:=false |
| **end;** |

У остальных лепестков обработчики событий будут подобными.

Так как в данном проекте во всех процедурах используется одна и та же переменная х, то ее можно описать уже не как локальную, а как глобальную. Для этого уберем строку **Var x:integer;** из каждой процедуры и укажем тип переменной х в уже имеющемся разделе описания переменных, где пока имеется лишь описание формы.

|  |
| --- |
| **var** |
| Form1: TForm1; |
| х: integer; |
|  |

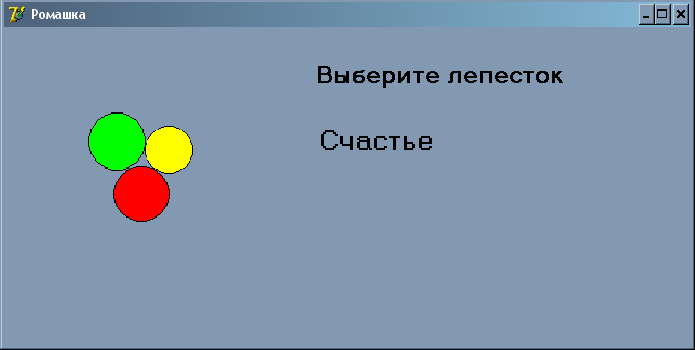
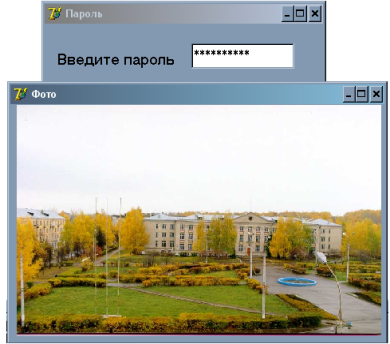


Рис. 84 Окно работающего приложения

**Задания**

1. Напишите проект анализа работы пожарного датчика в помещении. Пользователю предлагается ввести температуру в комнате. Если температура превысит 60ºС, то должно появиться сообщение «Пожарная ситуация».
2. Составить проект, в котором анализируется введенное число. Программа должна определить является ли это число положительным, отрицательным или нулем; четным или нечетным.
3. Составить проект, в котором вычисляется квадратный корень из введенного числа. Если введено отрицательное число, то должно появиться сообщение «Введите положительное число».
4. Составить проект вычисления функции y=1/x. Если введенное значение х принадлежит области определения функции, то выводится результат расчета, иначе выводится фраза «Введите другое значение».
5. Составить программу, которая запрашивает у пользователя пароль и записывает его звездочками. Ввод звездочек задается свойством PasswordChar. Если пароль введен правильно, то выводится окно с картинкой (рис. 85а), если неверно, то выводится сообщение «Введите другой пароль» (рис. 85б).

а) б)

Рис. 85 Работающее приложение

1. Написать программу, которая запрашивает возраст мужчины и сообщает, сколько лет ему осталось до пенсии, либо что он уже пенсионер.
2. Составить программу, которая запрашивает вес и рост человека, вычисляет оптимальное значение веса (рост (см) – 100) и сравнивает его с реальным весом. По результату сравнения выводится соответствующее сообщение.
3. Создать проект «Вычисление размера налога». Налоги рассчитываются по следующей схеме: если доход меньше 5000 руб, то налог составляет 12% от дохода, если доход больше 5000 руб, то налог составляет 30% от дохода.

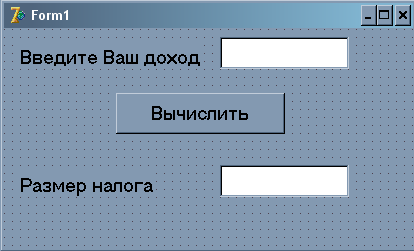


Рис. 86 Графический интерфейс проекта

1. Ввести двузначное число. Составить программу, которая определяет, какая из его цифр больше, первая или вторая.
2. Составить проект для вычисления значения функции .
3. Составить проект, в котором выясняется, имеет ли уравнение  действительные корни. Если корни есть, то нужно найти и вывести их. В противном случае должно появиться сообщение, что корней нет. Корни квадратного уравнения вычисляются по формулам: , .

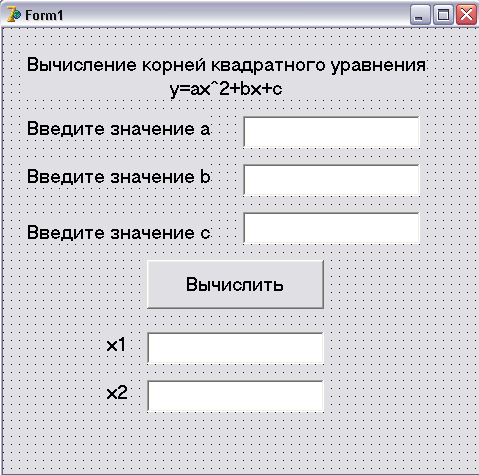
****

Рис. 87 Графический интерфейс проекта

1. Создать проект «Книжный магазин». Компьютер запрашивает стоимость книг, сумму денег, внесенную покупателем; если сдача не требуется, выводит «спасибо»; если денег внесено больше, чем необходимо, то выводит «возьмите сдачу» и указывает сумму сдачи; если денег недостаточно, то выводит сообщение об этом и указывает размер недостающей суммы.
2. Администрация одного магазина для привлечения большего числа клиентов ввела правило, согласно которому каждый покупатель, который приобрел товар на сумму более 1000 рублей, имеет право на трехпроцентную скидку со стоимости покупок. Требуется составить программу, которая в случае, если стоимость покупки превышает указанную сумму, должна подсчитывать величину скидки и ту сумму, которую должен оплатить покупатель с учетом скидки. В случае же если стоимость покупок меньше 1000 рублей, программа должна выдавать сообщение о том, что покупка должна быть оплачена полностью.
3. Составить проект для вычисления значения функции .
4. Создать проект «Ракета», который определяет траекторию движения ракеты. Ракета запускается с точки на экваторе и развивает скорость v км/с. Если v<=7.8 км/с, то ракета упадет на Землю, если 7.8<v<11.2, то ракета станет спутником Земли, если 11.2≤v≤16.4, то ракета станет спутником Солнца, если v>16.4, то ракета покинет Солнечную Систему.
5. Составить программу, которая требует ввести номер года и определяет, является ли этот год високосным или нет. Указание. В современном (григорианском) календаре каждый год, номер которого делится на 4, является високосным, за исключением тех, которые делятся на 100 и не делятся на 400. Например, 1900 год - не високосный, 2000 год - високосный.
6. Ставить программу, которая находит сумму цифр двузначного числа. Если же введенное число не является двузначным, то программа должна сообщить пользователю о его ошибке и прекратить на этом свою работу.
   1. **Оператор выбора**

Алгоритмическая структура «выбор» применяется для реализации ветвления со многими вариантами серий команд. В структуру выбора входят несколько условий, проверка которых осуществляется в строгой последовательности их записи в команде выбора. При истинности одного из условий выполняется соответствующая последовательность команд.

Условие 1

Серия 1

Да

Нет

Условие 2

Серия 1

Серия 1

Да

Нет

Рис. 88 Блок-схема алгоритмической структуры «выбор»

Если в программе необходимо выбрать одно действие из нескольких возможных, то используется **оператор выбора**. Структура оператора выбора такова:

**Case** <ключ выбора> **of**

<список выбора>: <оператор>

[**else** <оператор>]

**end**;

Здесь Case, of, else, end – зарезервированные слова (выбор, из, иначе, конец);

<ключ выбора> - ключ выбора;

<список выбора> - одна или несколько конструкций вида:

<константа выбора>: <оператор >;

<константа выбора> - константа того же типа, что и выражение <ключ выбора>;

<оператор> - произвольные операторы Delphi.

Оператор выбора работает следующим образом. Сначала вычисляется выражение <ключ выбора>. Затем в последовательности параметров из списка выбора отыскивается такой, который соответствует значению ключа выбора и выполняется оператор, соответствующий данной переменной. После этого оператор выбора завершает работу. Если в списке выбора не будет найдена константа, соответствующая значению ключа выбора, то управление передается операторам, стоящим за словом Else. Часть Else - <оператор> может опускаться. Тогда при отсутствии в списке выбора нужной константы ничего не произойдет, и оператор выбора просто завершит свою работу.

Рассмотрим пример работы оператора выбора.

|  |
| --- |
| **Пример.**Составить проект «Времена года», в котором по введенному номеру месяца определяется время года. |

Создадим графический интерфейс проекта подобно тому, как изображено на рис. 89.

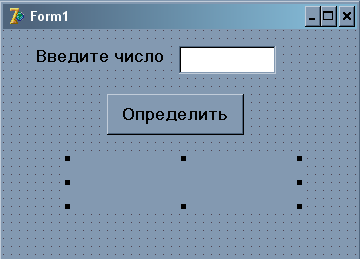


Рис. 89 Графический интерфейс проекта «Времена года»

Используем дополнительную переменную *а* целого типа, в которой будет храниться вводимое в поле Edit1 число. С помощью оператора выбора проверим, к какому диапазону относится число. Если введенное число имеет значения 1, 2 и 12, то свойство Caption надписи Label2 должно принять значение ‘Зима’; от 3 до 5 – значение ‘Весна’, от 6 до 8 – значение “Лето’, от 9 до 11 – значение ‘Осень’.

Текст обработчика события для кнопки Button1 будет следующим:

|  |
| --- |
| **procedure** TForm1.Button1Click(Sender: TObject); |
| **var** a:integer; |
| **begin** |
| a:=StrToInt(Edit1.Text); |
| **case** a **of** |
| **1,2,12:** Label2.Caption:=’Зима’; |
| **3..5:** Label2.Caption:=’Весна’; |
| **6..8:** Label2.Caption:=’Лето’; |
| **9..11:** Label2.Caption:=’Осень’; |
| **end;** |
| **end.** |

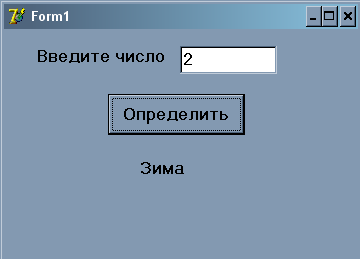


Рис. 90 Работающее приложение

**Задания**

1. Изменить проект «Времена года» таким образом, чтобы при вводе чисел меньше 1 или больше 12 выводилось сообщение: «Введите число в диапазоне от 1 до 12».
2. Составить проект, в котором используется следующий сценарий: в текстовое поле требуется ввести натуральное число, определяющее возраст человека в годах; по щелчку на кнопке это число должно появиться в надписи с добавлением, соответственно, «год», «года» или «лет». (1 год, 23 года, 45 лет).

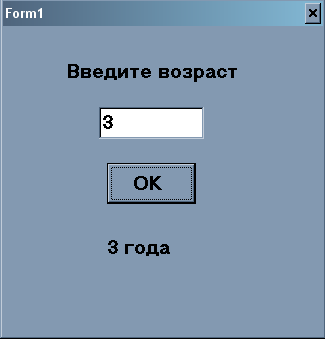


Рис. 91 Окно работающего приложения

1. Составить программу, которая запрашивает номер дня недели и выводит расписание уроков в Вашем классе в этот день.
2. Составить проект, в котором требуется ввести номер месяца. Программа должна вывести название этого месяца и указать число дней в нем. При этом сообщение о месяце должно выводиться на экран компьютера соответствующим сезону цветом. Для зимнего месяца сообщение должно выводиться белым цветом, для весеннего – зеленым цветом, для летнего – красным, для осеннего – желтым.
3. Создать проект, который запрашивает возраст ребенка и определяет в какой отряд лагеря его нужно распределить. При этом вместе с сообщением для каждого отряда выводится соответствующая эмблема.
4. Составить проект определения недельной зарплаты сотрудника предприятия за отработанные часы, если оплата за час равна 90руб. При расчете зарплаты учесть, сколько часов проработал сотрудник в неделю. В зависимости от этого количества часов его зарплата изменится в соответствии с коэффициентом k:

* первые 39 часов: k = 1;
* с 40-го по 44-й час: k = 1.2;
* с 45-го по 49-й час: k = 1.5;
* после 49-го часа: k = 1.8;

1. Составить проект, в котором требуется ввести значение х. Если введенное число от 0 до 10, то необходимо рассчитать квадрат этого числа, от 11 до 20 – куб числа, от 21 до 30 – четвертую степень. Если введенное число не входит ни в одни диапазон, то оно должно остаться без изменений.
2. Создать проект «Мини-калькулятор». В этом проекте требуется ввести два числа и выбрать номер арифметической операции. В зависимости от выбранного номера над числами должна производиться соответствующая арифметическая операция: сложение, вычитание, умножение или деление. Предусмотреть ситуацию, когда при выполнении деления, в качестве делителя будет введен ноль. В этом случае необходимо вывести сообщение «На ноль делить нельзя».

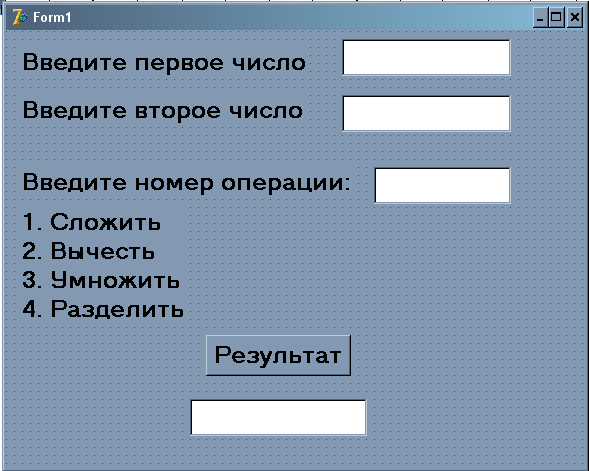


Рис. 92 Графический интерфейс проекта

**5.9** **Создание проекта «Калькулятор» с использованием**

**условного оператора и оператора выбора**

Создадим проект «Калькулятор», демонстрирующий работу стандартного калькулятора. Разработаем графический интерфейс проекта подобно тому, как изображено на рис. 93. Для этого поместим на форму следующие компоненты:

* текстовое поле Edit1, в котором будут отображаться нажимаемые цифры и результат операций;
* кнопки ввода цифр: Button1-Button9 с заголовками от 1 до 9 соответственно, Button10 с заголовком 0;
* кнопки выполнения арифметических действий: Button11 с заголовком «+», Button12 – «-», Button13 – «\*», Button14 – «/», Button15 – «=».
* кнопка очистки поля ввода: Button16 с заголовком «С».

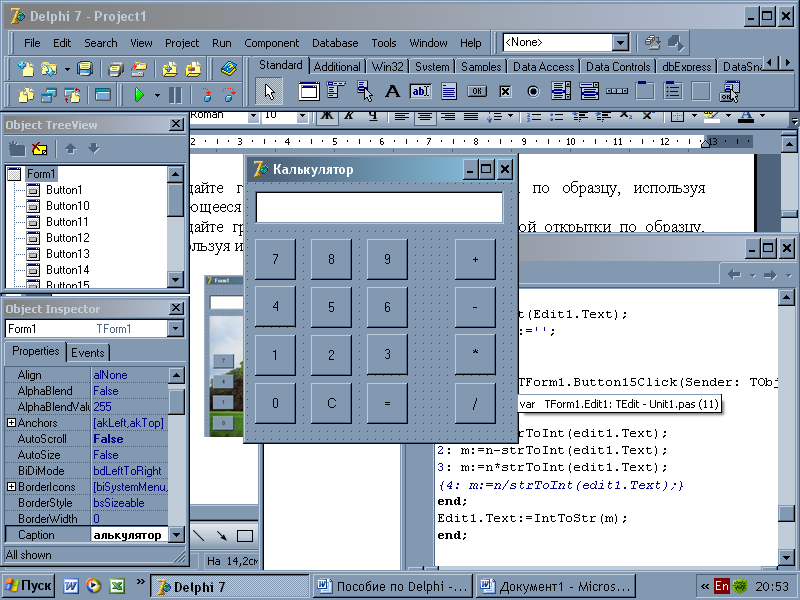


Рис. 93 Графический интерфейс проекта «Калькулятор»

Сначала составим обработчики событий для кнопок ввода цифр Button1-Button10.

При нажатии на кнопку с цифрой 1 в текстовом поле должна появиться единица. Для этого свойству Text поля Edit1 нужно присвоить значение ‘1’: Edit1.Text:=’1’. Но данная запись не позволяет вводить числа, состоящие из нескольких цифр (двузначные, трехзначные и т.д.). Чтобы задать ввод многозначных чисел, следует заголовок кнопки прибавить к уже имеющемуся в поле ввода тексту: Edit1.Text:= Edit1.Text + ’1’.

Таким образом, текст обработчика события для кнопки Button1 будет следующим:

|  |
| --- |
| **procedure** TForm1.Button1Click(Sender: TObject); |
| **begin** |
| Edit1.Text:= Edit1.Text+’1’; |
| **end;** |

Для кнопки Button2 обработчик события будет иметь похожий вид. Здесь в текстовое поле записывается заголовок кнопки Button2.

|  |
| --- |
| **procedure** TForm1.Button1Click(Sender: TObject); |
| **begin** |
| Edit1.text:= Edit1.Text+’2’; |
| **end;** |

Аналогично составляются обработчики для остальных кнопок ввода цифр.

Для описания работы кнопок, выполняющих арифметические действия, введем переменную-индикатор *i*. Эта переменная будет содержать номер операции. Переменная i = 1 задает операцию сложения; i = 2 – вычитания; i = 3 – умножения; i = 4 – деления. Текст, находящийся в поле ввода, необходимо преобразовать в число с помощью функции StrToInt. Результат преобразования поместим в дополнительную переменную *n:* n:=StrToInt(Edit1.Text). Эта переменная будет содержать первое число в операции. После выполнения операции текстовое поле нужно очистить, чтобы оно было готово к вводу второго числа.

Дополнительные переменные i и n опишем как глобальные, так как они используются в нескольких процедурах. Для этого укажем их тип в уже имеющемся разделе описания переменных.

**var**

Form1: TForm1;

i,n: integer;

Текст обработчика события для кнопки Button11, выполняющей сложение, будет следующим:

|  |
| --- |
| **var** |
| Form1: TForm1; |
| i,n:integer; |
| **…** |
| //обработчик события для кнопки «+» |
| **procedure** TForm1.Button11Click(Sender: TObject); |
| **begin** |
| i:=1; |
| n:=StrToInt(Edit1.Text); |
| Edit1.Text:=' '; |
| **end;** |

Для остальных кнопок, отвечающих за выполнение операций, обработчики событий будут аналогичными, изменится лишь значение индикатора:

|  |
| --- |
| //обработчик события для кнопки «-» |
| **procedure** TForm1.Button12Click(Sender: TObject); |
| **begin** |
| i:=2; |
| n:=StrToInt(Edit1.Text); |
| Edit1.Text:=' '; |
| **end;** |
| //обработчик события для кнопки «\*» |
| **procedure** TForm1.Button13Click(Sender: TObject); |
| **begin** |
| i:=3; |
| n:=StrToInt(Edit1.Text); |
| Edit1.Text:=' '; |
| **end;** |
| //обработчик события для кнопки «/» |
| **procedure** TForm1.Button14Click(Sender: TObject); |
| **begin** |
| i:=4; |
| n:=StrToInt(Edit1.Text); |
| Edit1.Text:=' '; |
| **end;** |

В обработчике события для кнопки «=» следует проверить значение переменной-индикатора i.

Если i = 1, то будет выполняться операция сложения; если i = 2 – вычитания; если i = 3 – умножения; если i = 4 – деления. После проверки условия нужно задать второе число, для записи которого используем еще одну дополнительную переменную k. При выполнении операций сложения, вычитания и умножения преобразование символов, выражающих второе число, в числовое представление осуществляется с помощью функции StrToInt. У деления преобразование выполняется функцией StrToFloat. После этого, в зависимости от номера индикатора, запишем для каждой операции соответствующую формулу:

m:=n + k – для сложения;

m:=n – k – для вычитания;

m:=n\*k – для умножения;

m:=n/k – для деления;

Переменные m и k также опишем как глобальные переменные вещественного типа.

Текст обработчика события для кнопки Button15будет следующим:

|  |
| --- |
| **var** |
| Form1: TForm1; |
| i,n: integer; m, k: real; |
| … |
| **procedure** TForm1.Button15Click(Sender: TObject); |
| **begin** |
| **if** i = 1 **then** |
| **begin** |
| k:= StrToInt(Edit1.Text); |
| m:=n + k; |
| **end;** |
| **if** i = 2 **then** |
| **begin** |
| k:= StrToInt(Edit1.Text); |
| m:=n - k; |
| **end;** |
| **if** i = 3 **then** |
| **begin** |
| k:= StrToInt(Edit1.Text); |
| m:=n\*k; |
| **end;** |
| **if** i = 4 **then** |
| **begin** |
| **If** k=0 **then** ShowMessage (‘На ноль делить нельзя’) **else** |
| **begin** |
| k:= StrToFloat(Edit1.Text); |
| m:=n/k; |
| **end;** |
| **end;** |
| Edit1.Text:=FloatToStr(m); |
| **end;** |

При выполнении операции деления здесь предусмотрена ситуация, когда в качестве делителя введен ноль.

Так как в рассмотренном выше обработчике используется несколько условий, то их удобно представить с помощью оператора выбора case. Также можно обойтись без переменной *k,* отвечающей за второе число. Это значительно сокращает запись программы обработчика:

|  |
| --- |
| **procedure** TForm1.Button15Click(Sender: TObject); |
| **Begin** |
| **case** i **of** |
| **1**: m:=n+StrToInt(Edit1.Text); |
| **2**: m:=n-StrToInt(Edit1.Text); |
| **3**: m:=n\*StrToInt(Edit1.Text); |
| **4**: |
| **begin** |
| **If** k=0 **then** ShowMessage (‘На ноль делить нельзя’) |
| **else** m:=n/StrToFloat(Edit1.Text); |
| **end;** |
| **end;** |
| Edit1.Text:=FloatToStr(m); |
| **end;** |

**5.10 Реализация условного оператора и оператора выбора компонентами TRadioButton и TRadioGroup**

Условный оператор и оператор выбора применяются при описании работы компонента **TRadioButton** **(Переключатель)** компоненты с панели **Standard**. Переключатели всегда используются группами (см. рис. 94). В группе переключателей может быть включен только один элемент, а остальные находятся в отключенном состоянии. Работа группы переключателей аналогична работе исправного светофора, на котором всегда горит только одна лампочка, а остальные погашены.

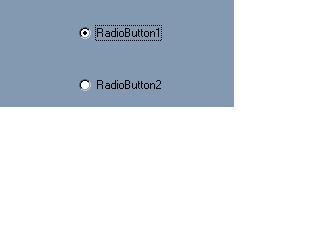


Рис. 94 Компонент Переключатель

**Основные свойства компонента TRadioButton**

**1. Свойство Caption (Заголовок)** задает подпись к переключателю.

**2. Свойство Checked (проверка)** проверяет состояние переключателя. Если он включен, то свойство приобретает значение True; если выключен, то False. Его можно изменить и на этапе проектирования, и на этапе разработки программы. Для проверки этого свойства в программе используется условный оператор.

**3. Свойство Alignment (выравнивание)** определяет расположение подписи к переключателю:

* + значение taRightJustify – расположение подписи справа;
  + значение taLeftJustify – расположение подписи слева.

На примере следующей задачи рассмотрим работу переключателей.

|  |
| --- |
| **Пример.**Составить проект, в котором реализуется следующий сценарий: с помощью переключателя выбирается время суток, в зависимости от выбранного времени на экран выводится соответствующее приветствие (доброго утра, дня, вечера или спокойной ночи). |

Разместим на форме 2 метки и четыре переключателя (см. рис. 95). Пожелание пользователю будем выводить во вторую метку Label2.

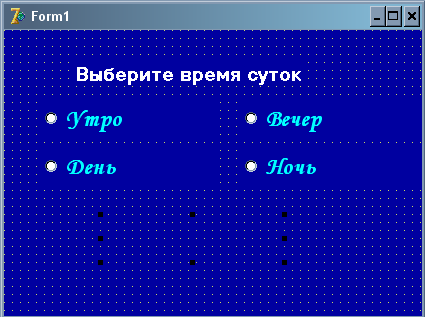


Рис. 95 Окно программы выбора времени суток

Обработчик события OnClick для первого переключателя RadioButton1 будет следующим:

|  |
| --- |
| **procedure** TForm1.RadioButton1Click(Sender: TObject); |
| **Begin** |
| Label2.Caption:='C добрым утром'; |
| **end;** |

Аналогично записываются программные коды остальных переключателей

|  |
| --- |
| **procedure** TForm1.RadioButton2Click(Sender: TObject); |
| **Begin** |
| Label2.Caption:='Добый день'; |
| **end**; |
| **procedure** TForm1.RadioButton3Click(Sender: TObject); |
| **Begin** |
| Label2.Caption:='Добрый вечер'; |
| **end;** |
| **procedure** TForm1.RadioButton4Click(Sender: TObject); |
| **Begin** |
| Label2.Caption:='Спокойной ночи'; |
| **end;** |

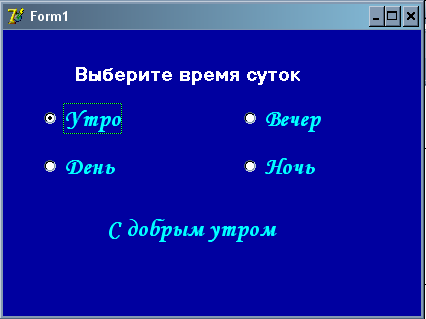
****

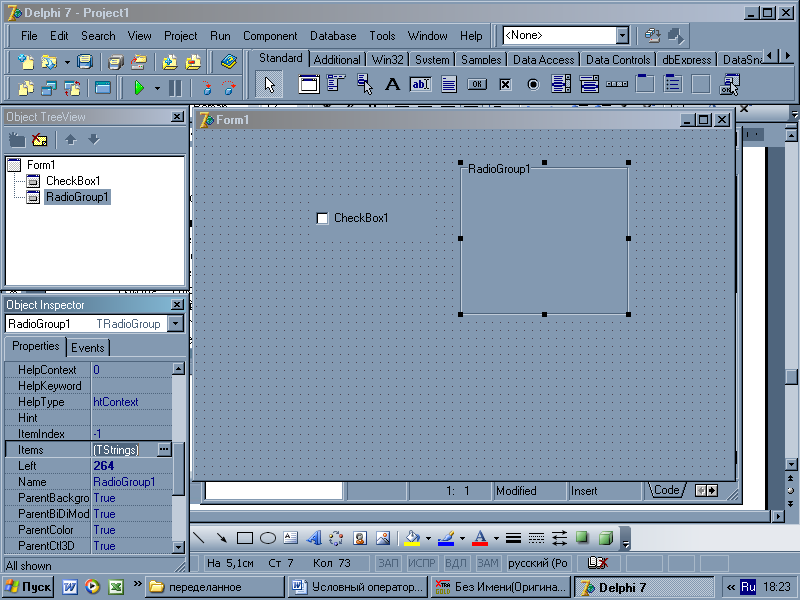
Рис. 96 Работающее приложение

Иногда в программе требуется использовать несколько групп переключателей. Например, одну группу – для указания пола человека, а другую – для выбора возраста. Для этого можно применить один из двух подходов.

Первый способ состоит в том, чтобы у каждой группы была отдельная панель. Эта панель создается с помощью объекта **Panel** **(Панель)** компоненты.

Второй подход заключается в использовании компонента **TRadioGroup** **(Группа переключателей)** компоненты. Он объединяет свойства и методы для работы с группой переключателей.

**Основные свойства TRadioGroup**

**1.** **Свойство** **Items (Список строк)** задает заголовки переключателей, входящих в компонент TRadioGroup. Это свойство имеет тип TString. Для ввода названий предусмотрен специальный редактор, который вызывается щелчком по кнопке .

Ввод заголовков переключателей при помощи редактора списка строк

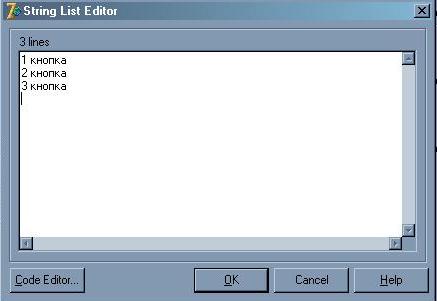


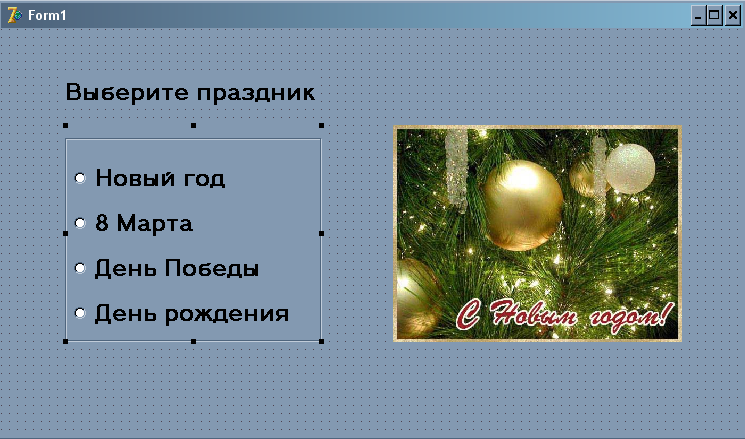
Рис. 97 Ввод заголовков переключателей

**2.** **Свойство Columns (столбцы)** задает число столбцов, образованных переключателями.

**3.** **Свойство ItemIndex (номер элемента)** содержит номер выделенного переключателя. Исходное значение этого свойства равно -1. Число -1 показывает, что ни один переключатель не выбран. Значение этого свойства изменяется автоматически, когда выбирается один из переключателей группы. Это свойство также можно менять программно: при занесении нового значения в свойство ItemIndex, изменится и текущий выбранный переключатель.

|  |
| --- |
| **Пример.** Составить проект «Поздравление», в котором при выборе с помощью переключателя определенного праздника на форме отображается соответствующая открытка-поздравление. |

Создадим графический интерфейс окна подобно тому, как изображено на рис. 97. В компоненты Image1-Image4 поместим изображения с соответствующими открытками. У всех компонентов TImage установим свойство Visible равным false.



Компонент

RadioGroup1

Компоненты

Image1-Image4

Рис. 98 Графический интерфейс окна

В данном проекте открытка с поздравлением «С Новым годом!» должна появиться при щелчке на первом переключателе компонента RadioGroup1. Первому переключателю соответствует номер 0 в свойстве ItemIndex, второму переключателю – номер 1, третьему – номер 2, четвертому – номер 3. Следовательно, необходимо с помощью условного оператора или оператора выбора проверить, какой из переключателей выбран, и отобразить соответствующую картинку на форме.

Текст обработчика события OnClick для группы переключателей RadioGroup1 будет следующим:

|  |
| --- |
| **procedure** TForm1.RadioGroup1Click(Sender: TObject); |
| **Begin** |
| **case** RadioGroup1.ItemIndex **of** |
| 0: **begin** |
| Image1.Visible:=true; |
| Image2.Visible:=false; |
| Image3.Visible:=false; |
| Image4.Visible:=false; |
| **end;** |
| 1: **begin** |
| Image2.Visible:=true; |
| Image1.Visible:=false; |
| Image3.Visible:=false; |
| Image4.Visible:=false; |
| **end;** |
| 2: **begin** |
| Image3.Visible:=true; |
| Image1.Visible:=false; |
| Image2.Visible:=false; |
| Image4.Visible:=false; |
| **end;** |
| 3: **begin** |
| Image4.Visible:=true; |
| Image1.Visible:=false |
| Image2.Visible:=false |
| Image3.Visible:=false |
| **end;** |
| **end;** |
| **end;** |

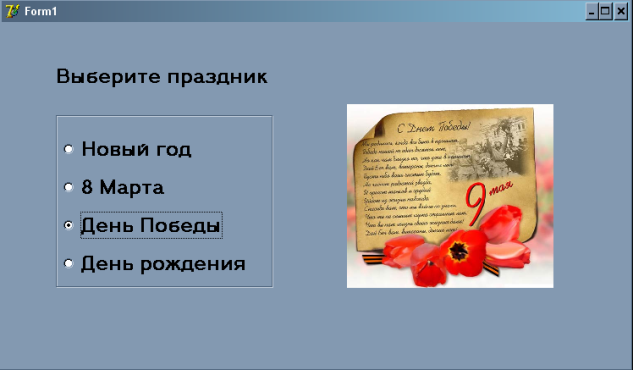


Рис. 99 Окно работающего приложения

* 1. **Реализация условного оператора и оператора выбора**

**компонентом TCheckBox**

Операторы условия и выбора также используются при описании работы компонента **TCheckBox (Флажок)** компоненты с панели **Standard**. Этот компонент служит для фиксации включенного или выключенного состояния.

**Основные свойства компонента TCheckBox**

**1.** **Свойство Checked (Проверка).** Это свойство принимает значение True, если флажок включен и False, если он сброшен.

**2. Свойство Alignment (Выравнивание)** задает расположение подписи к флажку.

**3.** **Свойство AllowGrayed.** Если для этого свойства задано значение True, то флажок при последовательных щелчках на нем будет поочередно принимать состояния «сброшен», «установлен частично», «установлен». При «частично установленном» состоянии флажок отображается приглушенным цветом. Такая возможность нужна, когда требуется сообщить пользователю о неполном соответствии указанному статусу.

**4.** **Свойство** **State (Состояние)** определяет текущее состояние флажка. Оно может принимать следующие значения:

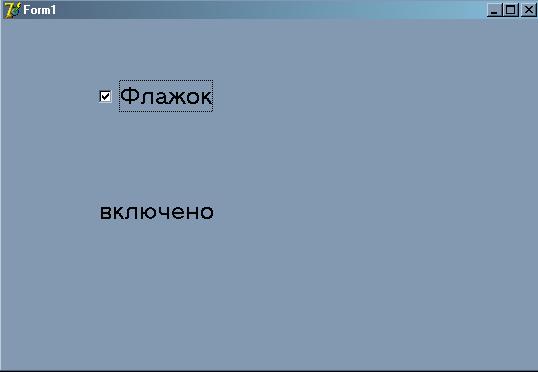
cbUncheked – сброшен;

cbGrayed – установлен частично;

cbChecked – установлен.

|  |
| --- |
| **Пример.** Составить программу, которая выводит в надпись текущее состояние Флажка. |

Разместим на форме элементы CheckBox и Label. Для Флажка свойство AllowGrayed изменим на True. Создадим обработчик события OnClick по Флажку и запишем в нем оператор выбора текущего состояния флажка.



Состояние флажка отображается в

надписи

Рис. 100 Вывод состояния флажка

|  |
| --- |
| **procedure** TForm1.CheckBox1Click(Sender: Tobject); |
| **Begin** |
| **case** CheckBox1.State of |
| cbUnchecked: Label1.Caption:=’выключено’; |
| cbGrayed: Label1.Caption:=’затемнено’; |
| cbChecked: Label1.Caption:=’включено’; |
| **end;** |
| **end;** |

**Задания**

1. Составить проект, в котором выводится расписание уроков в выбранный день недели. Дни недели задаются с помощью переключателей.
2. Составить проект, который закрашивает форму выбранным цветом из списка переключателей.
3. Написать проект, в котором требуется ввести массу в килограммах и перевести ее в выбранную единицу измерения (килограмм, миллиграмм, грамм, тонна, центнер).

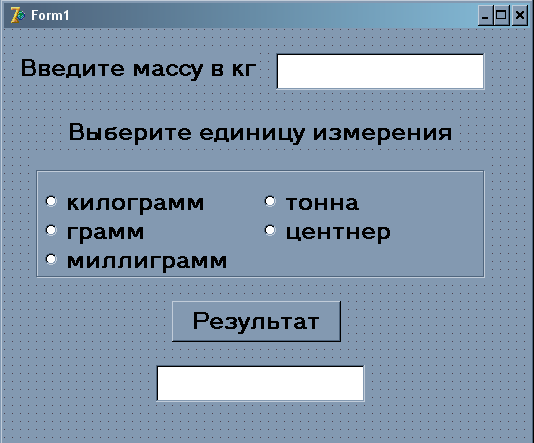


Рис. 101 Графический интерфейс проекта

1. Пусть элементами круга являются радиус, диаметр и длина окружности. Составить программу, которая по выбранному элементу запрашивала бы его соответствующее значение и вычисляла бы площадь круга.

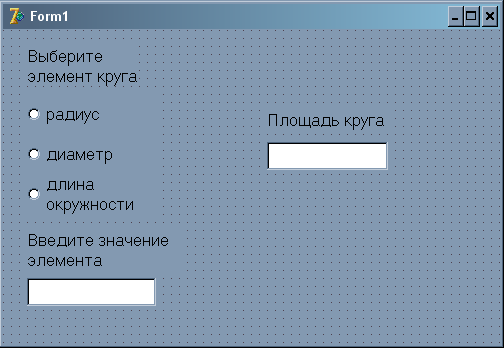


Рис. 102 Графический интерфейс проекта

1. Составить программу перевода заданного значения угла: из радианной меры в градусную и наоборот. Программа должна запрашивать, какой перевод нужно осуществить, и выполняет указанное действие.
2. Составить программу вычисления неизвестной величины по закону Ома:
   * при известных значениях силы тока и напряжения найти сопротивление;
   * при известных значениях напряжения и сопротивления найти силу тока;
   * при известных значениях силы тока и напряжения найти сопротивление.

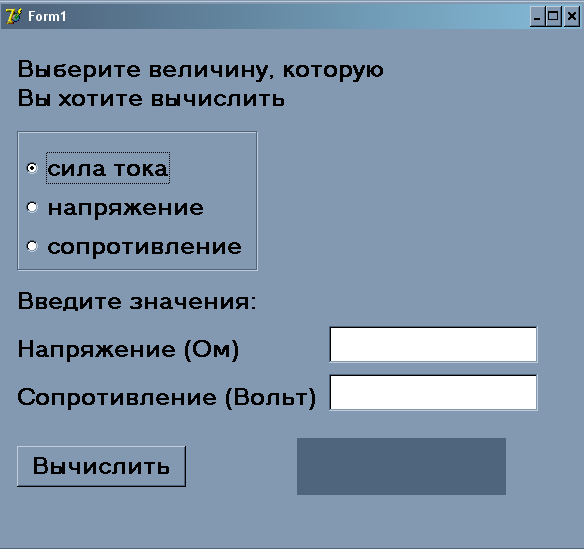


Рис. 103 Графический интерфейс проекта

1. Составить программу вычисления площадей основных геометрических фигур. Программа предлагает пользователю выбрать фигуру из имеющегося перечня и задать ее параметры, после чего выводит на экран результат расчета площади фигуры.
2. Создать проект «Электронный справочник». Справочник содержит разделы: кинотеатры, театры, музеи, концертные залы, выставочные залы. При выборе того или иного раздела выводится соответствующая афиша на выбранный день.
   1. **Создание проекта «Тест» с использованием условного оператора и переключателей**

|  |
| --- |
| **Пример.** Создать проект «Тест» по информатике. Пользователю предлагается несколько вопросов, каждый из которых имеет один вариант ответа. При щелчке на кнопке «Результат» должна появиться надпись с количеством правильных ответов. Оценка за тест выдается по щелчку на кнопке «Оценка». |

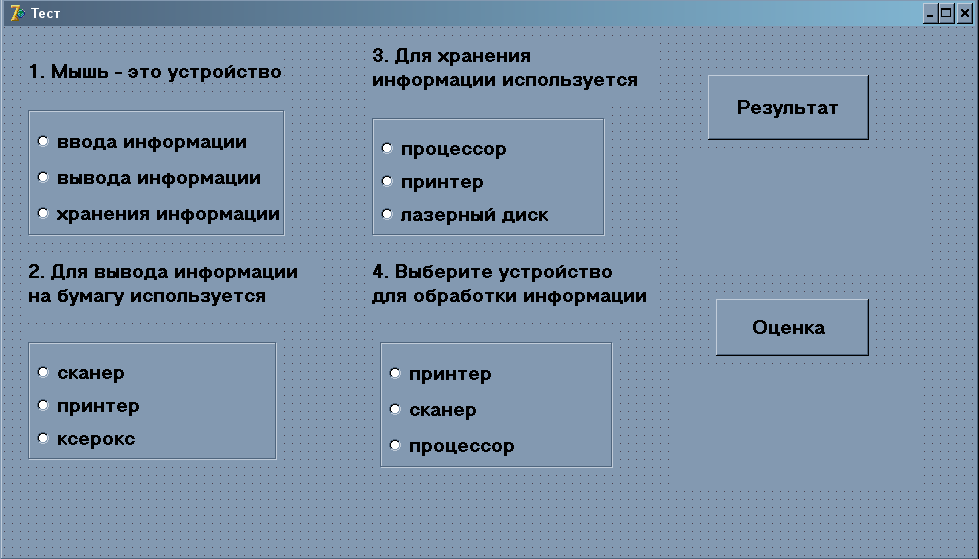


Рис. 104 Графический интерфейс проекта «Тест»

Создадим графический интерфейс проекта по образцу рис. 103.

В данном проекте необходимо ввести переменную целого типа S, в которой будет храниться количество правильных ответов. При каждом правильном ответе переменная S должна увеличиваться на единицу. В начале программы ее необходимо обнулить. Для этого, создадим обработчик события OnCreate у формы, в котором запишем один оператор S:=0.

|  |
| --- |
| **procedure** TForm1.FormCreate(Sender: TObject); |
| **Begin** |
| S:=0; |
| **end;** |

Для каждого компонента TRadioGroup необходимо создать обработчик события OnClick, в котором следует проверить, соответствует ли индекс выбранного ответа правильному. Если это условие выполняется, то переменная S должна увеличиваться на 1. Например, у первой группы переключателей RadioGroup1 будет следующий текст программного кода:

|  |
| --- |
| **var** |
| Form1: TForm1; |
| s:integer; |
| … |
| **procedure** TForm1.RadioGroup1Click(Sender: TObject); |
| **begin** |
| **if** RadioGroup1.ItemIndex=0 **then** S:=S+1; |
| **end;** |

Для вывода числа правильных ответов создадим следующий обработчик события OnClick у кнопки с заголовком «Результат».

|  |
| --- |
| **procedure** TForm1. Button1Click (Sender: TObject); |
| **begin** |
| Label5.Caption:= 'Количество правильных ответов '+IntToStr(S); |
| **end;** |

Чтобы вывести оценку, необходимо снова использовать условный оператор:

|  |
| --- |
| **procedure** TForm1. Button2Click (Sender: TObject); |
| **begin** |
| if s=4 then Label6.Caption:='Ваша оценка 5'; |
| if s=3 then Label6.Caption:='Ваша оценка 4'; |
| if s=2 then Label6.Caption:='Ваша оценка 3'; |
| if (s=1) or (s=0) then Label6.Caption:='Ваша оценка 2'; |
| **end;** |

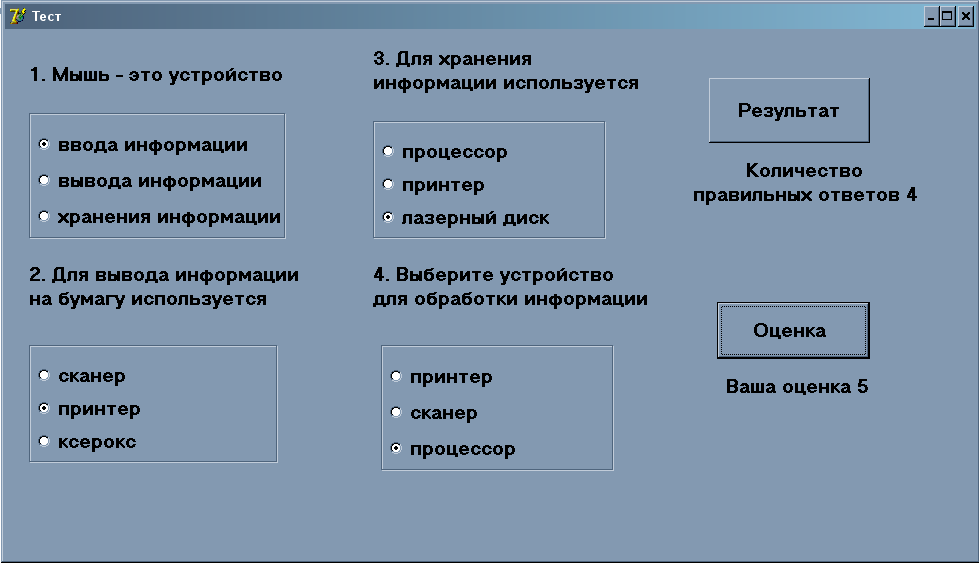
****

Рис. 105 Работающее приложение

В данном проекте использовалась только одна форма, на которой размещались все вопросы. Однако можно для создания тестов использовать несколько окон – форм. В этом случае тест получается более наглядным и красочным.

Рассмотрим создание теста, в котором каждый вопрос выводится в новой форме.

Первую форму сделаем титульной и разместим на ней заголовок теста, рисунок и кнопку «Начать».

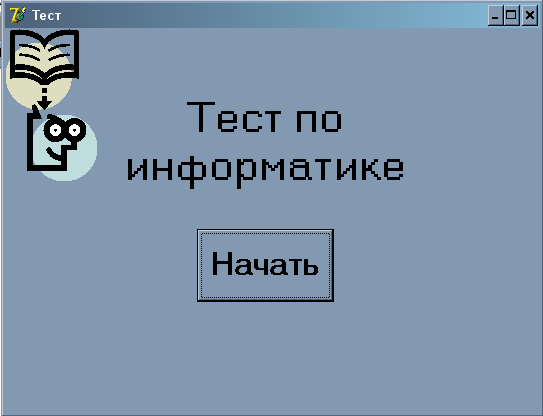


Рис. 106 Графической интерфейс первой формы проекта

Для подключения к проекту новой формы необходимо в главном меню выбрать команду [**File – New].** Если задержать указатель мыши на данной команде, то справа от нее появится список элементов, которые можно добавить в проект. Выберем из списка вариант **Form.** После этого в проекте появится дополнительная форма. В нашем случае это будет **Form2**.

Далее необходимо добавить оператор в основной программный модуль unit1, который открывал бы вторую форму. Для этого применяется процедура FormN.Show, где N — порядковый номер открываемого дополнительного окна.

В данном проекте вторая форма должна открыться при щелчке на кнопке «Начать», поэтому текст программного кода обработчика события OnClick у данной кнопки будет следующим:

|  |
| --- |
| **procedure** TForm1. Button1Click (Sender: TObject); |
| **begin** |
| Form2.Show; |
| Form1.hide; //скрываем первую форму |
| **end;** |

Далее необходимо разработать графический интерфейс подключаемого окна. Для того чтобы вывести на экран необходимую форму, нужно выбрать команду меню [**View-Forms].** После выполнения данной команды на экран компьютера выводится диалоговое окно, которое содержит список всех форм, используемых в данном проекте. Выбираем вторую из них, после чего данная форма будет выведена на передний план. Сам же процесс создания графического интерфейса аналогичен соответствующему процессу для основной формы.

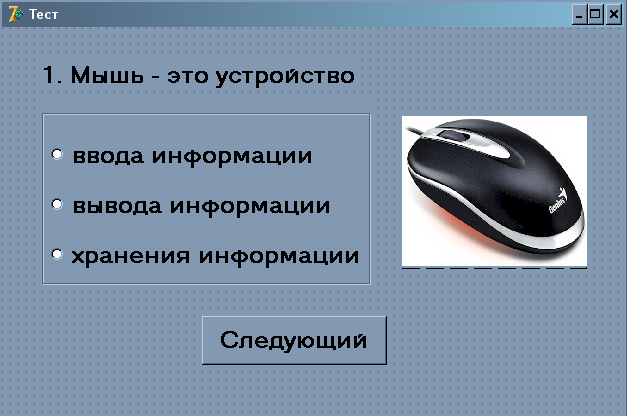


Рис. 107 Графический интерфейс дополнительного окна Form2

По завершении работы над графическим интерфейсом дополнительное окно можно закрыть, щелкнув стандартную закрывающую кнопку, расположенную в правом верхнем углу окна.

Код, описывающий действия, производимые пользователем в дополнительном окне, будет содержаться в дополнительном модуле unit2. Для того чтобы вывести на экран текст данного модуля, нужно выбрать команду меню [**View –** **Units].** После щелчка на этой команде открывается диалоговое окно со списком модулей, в котором выбираем нужный нам (в данном случае – это Unit2).

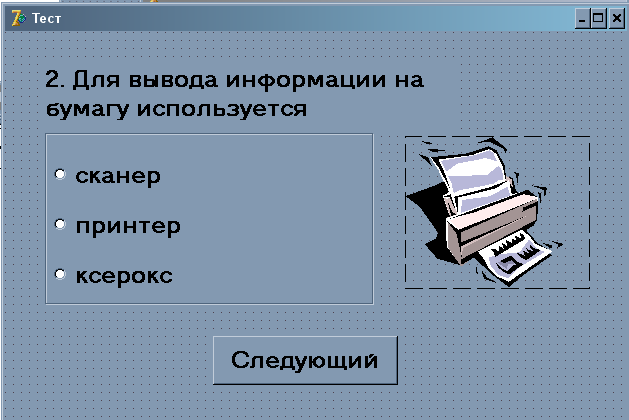
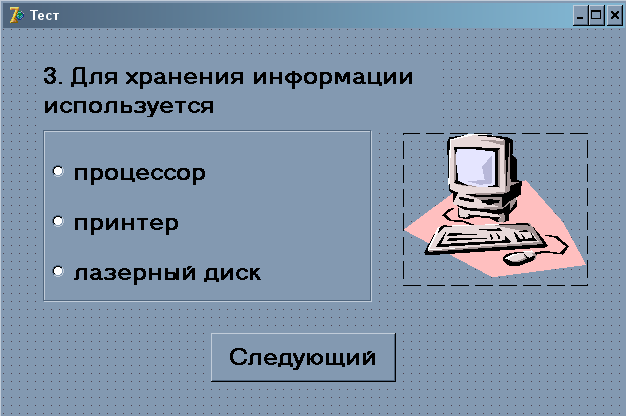
Сам по себе процесс написания кода аналогичен соответствующему процессу для основного окна, но нужно учесть, что модуль unit2 должен быть подключен к основному модулю unit1. Данное подключение производится так: в основной части программы после заголовка указывается список используемых в данной программе дополнительных модулей. Этот список начинается со служебного слова uses, после которого через запятую перечисляются модули. В этот список следует добавить модуль unit2. Тогда список будет выглядеть следующим образом:

**uses**

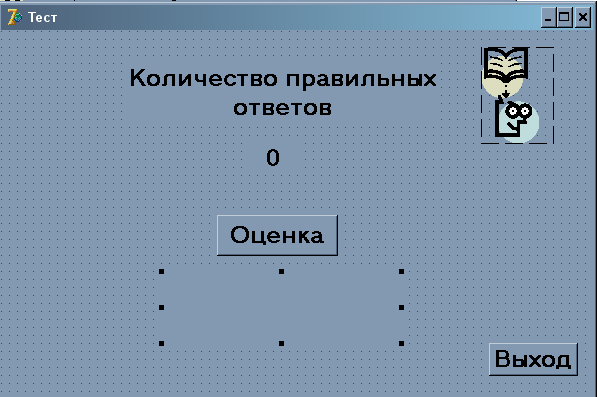
Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms, Dialogs, ExtCtrls, StdCtrls, Menus, Unit2;

После выполнения всех указанных ранее операций дополнительное окно становится полноценной частью проекта.

Аналогично добавляются остальные окна теста Form3-Form6.

** **

а) б)

в) г)

Рис. 108 Графический интерфейс дополнительных окон Form3-Form6

Разработаем код программного модуля unit2.pas для второй формы.

Сначала создадим обработчик события TForm2.RadioGroup1Click, в котором проверим, соответствует ли индекс выбранного ответа правильному варианту. При разработке теста с несколькими формами желательно не использовать дополнительных переменных:

|  |
| --- |
| **procedure** TForm2.RadioGroup1Click (Sender: TObject); |
| **begin** |
| **if** RadioGroup1.ItemIndex=0 **then** |
| Form6.Label1.Caption:=IntToStr(StrToInt(Form6.Label1.Caption)+1); |
| **end;** |

Количество правильных ответов выводится в компонент Label1 шестой формы.

При щелчке на кнопке «Следующий» третья форма должна появиться, а вторая скрыться.

|  |
| --- |
| **procedure** TForm2.Button1Click (Sender: TObject); |
| **begin** |
| Form3.show; |
| Form2.hide; |
| **end;** |

Так как в программном модуле unit2.pas были использованы объекты Form3 и Form6, то необходимо подключить их модули к модулю unit2.pas. Т.е. добавить их в список **Uses** к уже имеющимся модулям.

**uses**

Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms, Dialogs, StdCtrls, jpeg, ExtCtrls, Unit3, Unit6;

Аналогично разрабатываются программные модули unit3.pas-unit5.pas.

В программном модуле unit6.pas у шестой формы необходимо прописать обработчики событий для кнопок «Оценка» (выводит соответствующую оценку) и «Выход» (закрывает все формы).

|  |
| --- |
| **procedure** TForm6.Button1Click (Sender: TObject); |
| **begin** |
| **if** strtoint(label1.Caption)=4 **then** Label2.Caption:='Ваша оценка 5'; |
| **if** strtoint(label1.Caption)=3 **then** Label2.Caption:='Ваша оценка 4'; |
| **if** strtoint(label1.Caption)=2 **then** Label2.Caption:='Ваша оценка 3'; |
| **if** (strtoint(label1.Caption)=1) **or** (strtoint(label1.Caption)=0) **then** |
| Label2.Caption:='Ваша оценка 2' ; |
| **end;** |

|  |
| --- |
| **procedure** TForm6.Button2Click (Sender: TObject); |
| **begin** |
| Form1.close; |
| Form2.close; |
| Form3.close; |
| Form4.close; |
| Form5.close; |
| Form6.close; |
| **end;** |

**Задания**

* 1. Изменить проект «Тест» таким образом, чтобы при правильном ответе на экран выводилось сообщение «Верно». Если ответ неверный, то должно выводиться сообщение «Ошибка».
  2. Изменить проект «Тест» таким образом, чтобы вопросы могли иметь несколько правильных ответов.
  3. Дополнить проект «Тест» вопросами открытого типа, ответы на которые необходимо вводить в текстовое поле.

**5.13 Алгоритмическая структура «Цикл».**

**Операторы повторений.**

В алгоритмическую структуру «цикл» входит серия команд, выполняемая многократно. Такая последовательность команд называется телом цикла. Циклические алгоритмические структуры бывают двух типов:

* *циклы со счетчиком*, в которых тело цикла выполняется определенное количество раз;
* *циклы с условием*, в которых тело цикла выполняется, пока условие истинно.

В языке Delphi имеются три оператора, с помощью которых можно реализовать алгоритмическую структуру «цикл».

1. **Счетный оператор цикла For** имеет следующую структуру:

**For** <параметр цикла>:= <нач.знач> **to** <кон.знач> **do** <оператор>;

Здесь For, to, do – зарезервированные слова (для, до, выполнить);

<параметр цикла> - переменная любого порядкового типа;

<нач.знач> - начальное значение – выражение того же типа;

<кон.знач> - конечное значение – выражение того же типа;

<оператор> - произвольный оператор Delphi.

Блок схема цикла со счетчиком имеет вид рис. 109.

Счетчик:=начальное значение

Счетчик>Конечное

значение

Нет

Тело цикла

Счетчик:= Счетчик +1

Да

Рис. 109 Блок-схема выполнения цикла со счетчиком

Оператор For работает следующим образом. Сначала вычисляется выражение <нач.знач> и осуществляется присваивание параметру цикла начального значения. После этого циклически повторяется:

* проверка условия <параметр\_цикла>:=<кон\_знач>; если условие выполнено, оператор For завершает свою работу;
* выполнение оператора <оператор>;
* наращивание переменной <параметр цикла> на единицу.

Можно использовать изображение блок-схемы с блоком модификации.

Счетчик:=нач.знач, кон. знач

Тело цикла

Рис. 110 Блок-схема цикла со счетчиком

с блоком модификации.

Оператор For выполняет циклически только один оператор. Если нужно выполнить в цикле последовательность операторов, их необходимо заключить в операторные скобки begin .. end.

|  |
| --- |
| **Пример.** Составить проект, в котором подсчитывается сумма чисел от 1 до n. |

Создадим графический интерфейс проекта по образцу рис. 111.

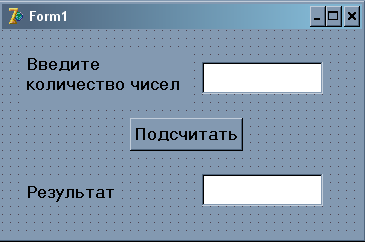


Рис. 111 Окно подсчета суммы

Алгоритм вычисления суммы представим в виде блок-схемы (рис. 112):

S:= 0

ввод n

i:= 1, n

S:= S + i

вывод S

Рис. 112 Блок-схема подсчета суммы чисел от 1 до n.

Текст обработчика события OnClick для кнопки будет следующим:

|  |
| --- |
| **procedure** TForm1.Button1Click(Sender: Tobject); |
| **var** i, S, n: Integer; |
| **Begin** |
| S:=0; |
| n:=StrToInt(Edit1.Text); |
| **for** i:= 1 **to** n **do** S:= S + i; |
| Edit2.Text:=IntToStr(S); |
| **End;** |

Цикл **for** выполняется от 0 до числа, введённого в компонент Edit1.

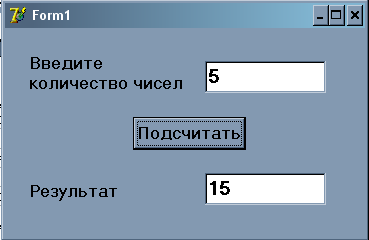


Рис. 113 Работающее приложение

Следует отметить еще два обстоятельства. Во-первых, условие, управляющее работой оператора For, проверяется перед выполнением оператора. Если условие не выполняется в самом начале работы оператора, то он не выполнится ни разу. Другое обстоятельство – шаг наращивания параметра строго постоянен и равен (+1).

Существует другая форма оператора:

**For** <параметр цикла>:= <нач.знач> **downto** <кон.знач> **do** <оператор>;

Замена зарезервированного слова to на downto означает, что шаг наращивания параметра цикла равен (-1), а управляющее условие приобретает вид <параметр цикла>:= <кон.знач>. Отличие оператора For..to..do от For..downto..do заключается в том, что первый оператор организует прямой цикл (начальное значение всегда должно быть меньше конечного), а второй – обратный (начальное значение должно быть больше конечного).

1. **Оператор цикла While с предпроверкой условия**:

**While** <условие> **do** <оператор>;

Здесь while, do – зарезервированные слова (пока [выполняется условие], делать);

<условие> - выражение логического типа;

<оператор> - произвольный оператор Delphi.

Блок-схема цикла с предусловием имеет вид рис. 114.

Условие истинно?

Тело цикла

Да

Нет

Рис. 114 Блок-схема цикла с предусловием

Оператор начинает свою работу с вычисления и проверки условия. Если условие выполняется, то выполняется <оператор>, после чего вычисление выражения <условие> и его проверка повторяются. Если <условие> не выполняется, то оператор цикла while прекращает работу.

При использовании оператора цикла While следует иметь в виду, что он выполняет циклически только один оператор. Если нужно выполнить в цикле последовательность операторов, их необходимо заключить в операторные скобки begin .. end. В теле цикла While должен быть хотя бы один оператор, влияющий на изменение значения параметра условия. В противном случае произойдет зацикливание программы.

Предыдущий пример с использованием цикла while будет выглядеть следующим образом:

|  |
| --- |
| **procedure** TForm1.Button1Click(Sender: Tobject); |
| **var** i, S, n: Integer; |
| **Begin** |
| S:= 0; |
| i:= 1; |
| n:= StrToInt(Edit1.Text); |
| **while** i<=n do |
| **Begin** |
| S:= S + i; |
| i:= I + 1; |
| **end;** |
| Edit2.Text:=IntToStr(S); |
| **end;** |

1. **Оператор цикла Repeat с постпроверкой условия**:

**Repeat** <тело цикла> **until** <условие>

Здесь repeat, until – зарезервированные слова (повторять [до тех пор], пока [не будет выполнено условие]);

<тело цикла> - произвольная последовательность операторов Delphi;

<условие> - выражение логического типа.

Оператор цикла с постпроверкой условия целесообразно использовать в тех случаях, когда тело цикла нужно выполнить не менее одного раза и заранее не известно общее количество повторений цикла.

Блок-схема цикла с постусловием представлена на рис. 114.

Тело цикла

Условие истинно?

Нет

Да

Рис. 115 Блок-схема цикла с постусловием

Операторы, заключенные между словами repeat и until, составляют тело цикла и выполняются до тех пор, пока не выполнится условие. Так как условие проверяется в конце цикла, то операторы цикла проверяются минимум один раз. В теле цикла может находиться произвольное число операторов без операторных скобок. Необходимо учитывать, что, по крайней мере, один из операторов тела цикла должен влиять на изменения значения параметра условия. В противном случае произойдет зацикливание программы.

C использованием цикла Repeat пример будет выглядеть так:

|  |
| --- |
| **procedure** Tform1.Button1Click(Sender: Tobject); |
| **var** i, S, n: Integer; |
| **begin** |
| S:= 0; |
| i:= 1; |
| n:= StrToInt(Edit1.Text); |
| **Repeat** |
| S:= S + i; |
| i:= i + 1; |
| **until** i>n; |
| Edit2.Text:=IntToStr(S); |
| **end;** |

**Задания**

1. Составить проект, в котором требуется ввести целое число n и рассчитать:
2. 2n;
3. n!;
4. ;
5. ;

n корней

1. Составить проект, в котором требуется ввести действительное число а, целое число n и рассчитать:
   * 1. а(а + 1)(а + 2)…(а + n - 1);
     2. ;
     3. sin(a) +sin2(a) + sin3(a) + … + sinn(a)
2. Составить проект, в котором нужно определить количество натуральных чисел, сумма кубов которых не превышает 50000.
3. Составить проект расчета значения функции y=sin(x) для х, изменяющегося от 0 до 10 с шагом 1. Для вывода значений функции использовать компонент TMemo (Текстовая область), который позволяет выводить текст в несколько строк. Инструкция добавления строки в компонент Memo1 имеет следующий вид: Memo1.Lines.Add (‘Новая строка’).
4. Составить проект, в котором в компонент TMemo выводятся числа от 1 до 100 и их квадраты.

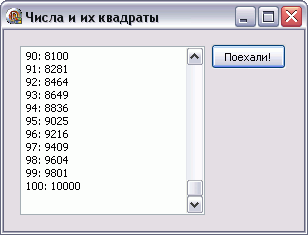


Рис. 116 Окно работающего приложения

1. Составить проект расчета значения функции y= для х, изменяющегося от 0 до 10 с шагом 0,5.
2. Радиус основания цилиндра равен 2, а высота его h. Составить программу, которая выводит на экран таблицу расчета значений объема цилиндра для значений высоты от 0,5 до 5 с шагом 0,5.
3. Составить проект расчета суммы всех четных чисел от 30 до 60.
4. Составить проект, в котором с помощью вложенных циклов выводится таблица умножения.

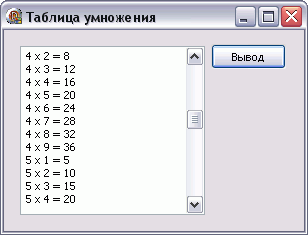


Рис. 117 Окно работающего приложения

1. Составить проект «Вклад», с помощью которого вкладчик, положивший деньги в банк, сможет рассчитать, какую сумму он получит по окончании срока действия договора с банком, если по условиям договора вклад положен в банк на определенное количество лет под определенный процент, который не должен изменяться до окончания действия договора.

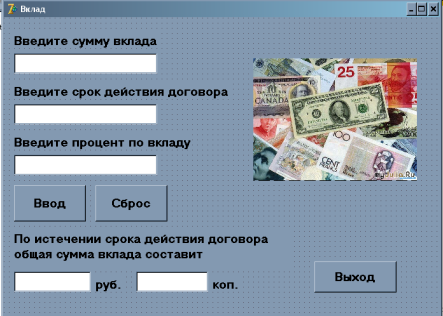


Рис. 118 Графический интерфейс проекта «Вклад»

1. Бизнесмен взял ссуду m тысяч рублей в банке под 20% годовых. Составить программу, вычисляющую через сколько лет его долг превысит s тысяч рублей, если за это время он не будет отдавать долг?
2. Создать проект «Моделирование биологических процессов», в котором определяется потомство одной бактерии через 6 часов после ее рождения (каждая бактерия живет 1 час, и каждые полчаса порождает одну новую – всего две за свою жизнь). Определить время, которое потребуется для увеличения бактерий до некоторого введенного количества.
3. Создать проект, в котором кнопка должна пропрыгать форму слева направо через 50 точек. Для задания паузы используйте команду sleep(t), где t – время задержки в миллисекундах.

**5.14** **Компонент TTimer**

Кроме рассмотренных циклических операторов в среде программирования Delphi для реализации повторения действий часто используется компонент **TTimer (Таймер)** л2 с панели System. Компонент **TTimer** можно рассматривать как циклический оператор, в котором цикл повторяется через задаваемый интервал времени. Компонент Timer является невизуальным, т.е. во время работы программы он не отображается на форме. Поэтому его значок можно поместить в любое место формы.

У таймера главным событием является OnTimer, которое возникает после заданного интервала времени.

**Основные свойства TTimer**

1. **Свойство Interval (Интервал)** – определяет промежуток времени в миллисекундах, через который периодически вызывается событие OnTimer. Период времени может составлять от 1 до 65535 мс.
2. **Свойство Enabled (Разрешение)** – разрешает (значение True) или запрещает (значение False) работу таймера.

Рассмотрим работу таймера на примере задачи №13 из заданий к параграфу 5.13

|  |
| --- |
| **Пример.**Создать проект, в котором кнопка должна пропрыгать форму слева направо через 50 точек. |

Для создания графического интерфейса проекта поместим на форму кнопку и таймер. У компонента таймер установим свойство Interval равным 250 (¼ секунды).

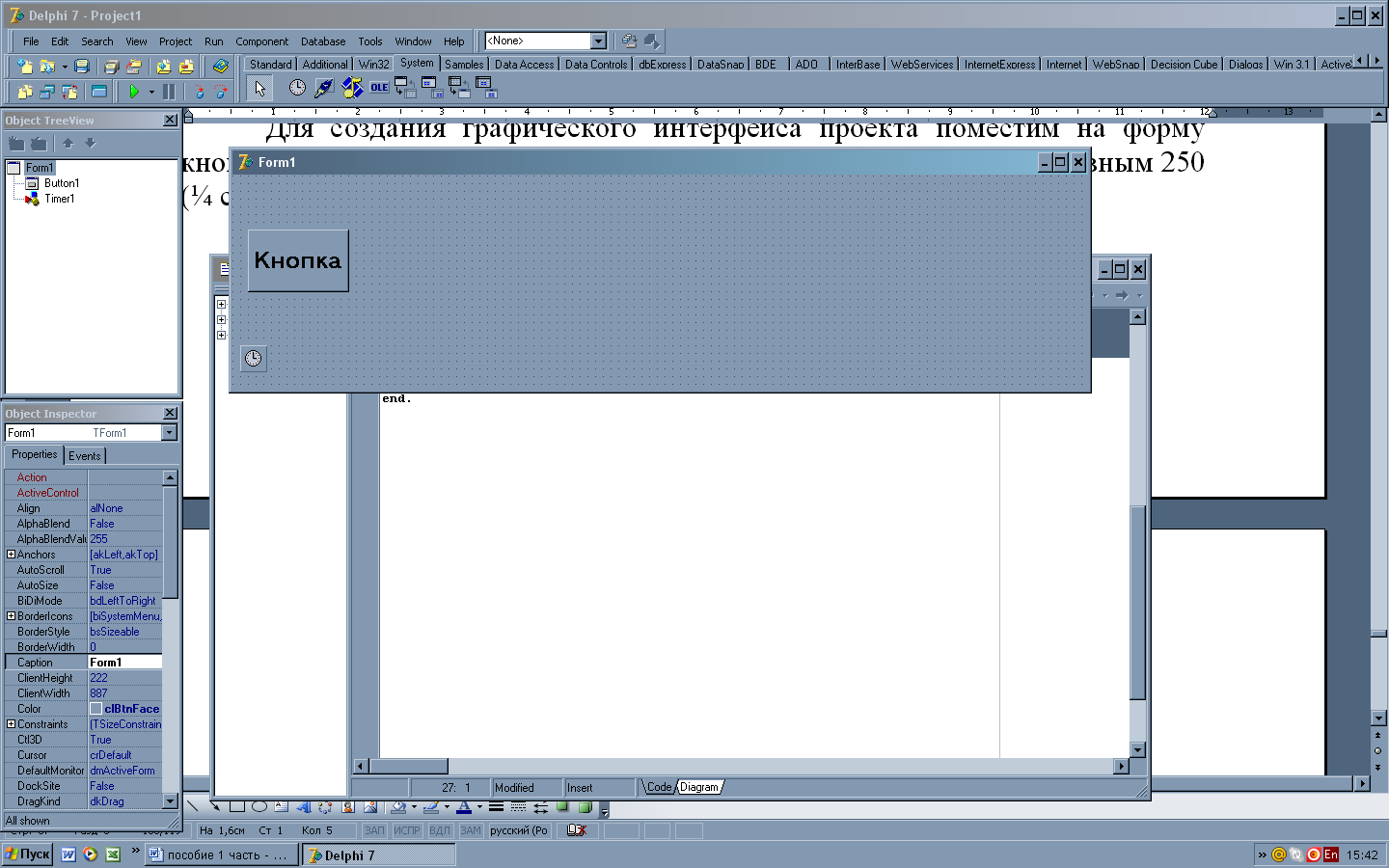


Рис. 119 Графический интерфейс проекта

Дважды щелкнем по таймеру, при этом откроется обработчик события OnTimer, в который запишем оператор, заставляющий кнопку двигаться слева направо с интервалом в 50 точек.

|  |
| --- |
| **procedure** TForm1.Timer1Timer(Sender: TObject); |
| **Begin** |
| Button1.Left:=Button1.Left+50; |
| **end;** |

С помощью таймера можно также вывести на форму текущее время.

|  |
| --- |
| **Пример.**Создать проект, показывающей работу электронных часов. |

Разместим на форме компоненты надпись и таймер, зададим интервал у таймера равным 1 секунде (значение 1000). Надпись будет нужна для вывода текущего времени.

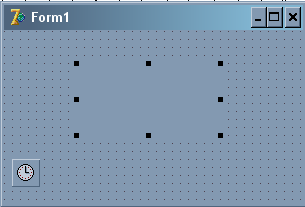


Рис. 120 Графический интерфейс проекта

Введем дополнительную переменную *а* типа TDateTime (дата/время). В эту переменную запишем текущее время, используя стандартную функцию Time. Затем преобразуем его в текстовое представление с помощью функции TimeToStr.

Обработчик события OnTimer запишется следующим образом:

|  |
| --- |
| **procedure** TForm1.Timer1Timer(Sender: TObject); |
| **var** а: TDateTime; |
| **Begin** |
| а:=Time; |
| Label1.Caption:=TimeToStr(а); |
| **end;** |



Рис. 121 Окно проекта «Электронные часы»

**5.15** **Создание проектов с использованием таймера**

|  |
| --- |
| **Пример.** Создать проект «Летающая тарелка», в котором по ночному небу перемещается летающая тарелка. При вылете тарелки за правую границу окна, она должна вновь появиться у его левой границы. |

Спроектируем графический интерфейс проекта подобно рис. 122, установив следующие свойства компонентов:

* для формы TForm: Color = clBlack, Height = 735, Width = 550, AutoScroll = false.
* для изображения тарелки TImage: Stretch = true, Proportional = true.
* для таймера TTimer: Interval = 200.

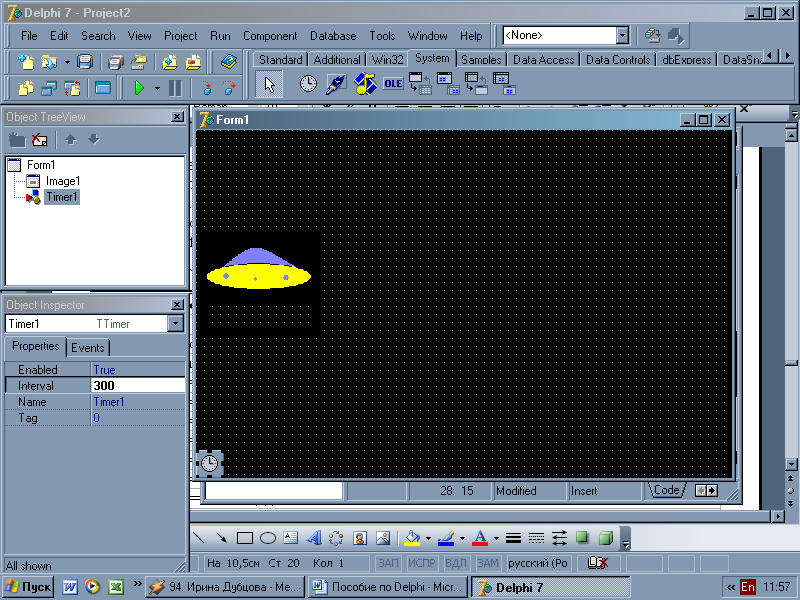


Рис. 122 Графический интерфейс проекта «Летающая тарелка»

При движении тарелки меняется её позиция относительно левого края окна. Для изменения этой позиции применим следующий оператор Image1.Left:=Image1.Left+h, где h – случайная величина. Если расстояние от левой границы до тарелки станет больше, чем размеры окна, то тарелку вернем на исходную позицию. Для этого используем условный оператор:

**if** Image1.Left>= Form1.Width **then** Image1.left:=1.

Текст программного кода обработчика события OnTimer будет следующим:

|  |
| --- |
| **procedure** TForm1.Timer1Timer(Sender: TObject); |
| **begin** |
| randomize; |
| //задаем для переменной h одно из 15 возможных состояний |
| h:=random(15); |
| Image1.Left:=Image1.Left+h; //движение тарелки |
| **if** Image1.Left>=Form1.Width **then** Image1.Left:=1; |
| **end;** |

Добавим в проект мерцающие звезды, которые появляются и исчезают случайным образом. Звезды помещаем в компоненты TImage 2 – Timage 16.

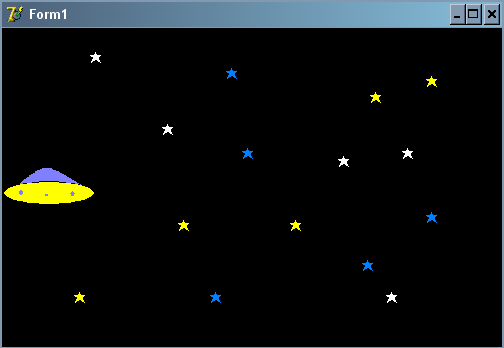
таймер

Рис. 123 Графический интерфейс проекта «Летающая тарелка»

Для создания эффекта мерцания используем генератор случайных чисел. Каждой звезде ставим в соответствие переменную, способную случайным образом принимать значения 0 или 1. Если переменная равна единице, то звезду делаем видимой с помощью свойства Visible (Image2.Visible: = true). Если в переменной окажется ноль, то звезда становится невидимой (Image2.Visible: = false).

|  |
| --- |
| **procedure** TForm1.Timer1Timer(Sender: TObject); |
| **begin** |
| randomize; |
| //задаем для переменных х1, х2 и т.д. одно из двух возможных состояний |
| x1:=random(2); |
| x2:=random(2); |
| x3:=random(2); |
| x4:=random(2); |
| x5:=random(2); |
| x6:=random(2); |
| x7:=random(2); |
| x8:=random(2); |
| x9:=random(2); |
| x11:=random(2); |
| x12:=random(2); |
| x13:=random(2); |
| x14:=random(2); |
| x15:=random(2); |
| x16:=random(2); |
| {Если переменная принимает значение, равное 1, то звезду видно, иначе – нет} |
| **if** x1=1 **then** image2.Visible:=true **else** image2.Visible:=false; |
| **if** x2=1 **then** image3.Visible:=true **else** image3.Visible:=false; |
| **if** x3=1 **then** image4.Visible:=true **else** image4.Visible:=false; |
| **if** x4=1 **then** image5.Visible:=true **else** image5.Visible:=false; |
| **if** x5=1 **then** image6.Visible:=true **else** image6.Visible:=false; |
| **if** x6=1 **then** image7.Visible:=true **else** image7.Visible:=false; |
| **if** x7=1 **then** image8.Visible:=true **else** image8.Visible:=false; |
| **if** x8=1 **then** image9.Visible:=true **else** image9.Visible:=false; |
| **if** x9=1 **then** image10.Visible:=true **else** image10.Visible:=false; |
| **if** x11=1 **then** image11.Visible:=true **else** image11.Visible:=false; |
| **if** x12=1 **then** image12.Visible:=true **else** image12.Visible:=false; |
| **if** x13=1 **then** image13.Visible:=true **else** image13.Visible:=false; |
| **if** x14=1 **then** image14.Visible:=true **else** image14.Visible:=false; |
| **if** x15=1 **then** image15.Visible:=true **else** image15.Visible:=false; |
| **if** x16=1 **then** image16.Visible:=true **else** image16.Visible:=false; |
| //задаем для переменной h одно из 15 возможных состояний |
| h:=random(15); |
| Image1.Left:=Image1.Left+h; //движение тарелки |
| **if** Image1.Left>=Form1.Width **then** Image1.Left:=1; |
| **end;** |

Данный проект можно усложнить, задав не просто мерцание звезд, а их развитие. Звезда появляется на небе, увеличивается в размерах, затем сжимается и гаснет.

Чтобы показать жизнь звезд, необходимо для каждой звезды создать свой таймер со следующими свойствами: Enabled = false, Interval = 1500. При этом у изображений Image2-Image16 следует установить свойства Visible = false, Stretch = true.

В обработчике события OnTimer для первой звезды Image2 введем переменную-индикатор *k,* которая будет показывать состояние звезды. Если k = 1, то звезда должна появиться на небе, то есть стать видимой: Image2.Visible:=True. При k = 2 звезда увеличивает свои размеры. Ширину и длину объекта Image2 нужно увеличить на определенное значение: Image2.Width:=Image2.Width+10; Image2.Height:=Image2.Height + 10. Если k = 3, то звезда сжимается. Для этого следует уменьшить размеры звезды. При k = 4 звезда гаснет, то есть становится невидимой: Image2.Visible:=false.

Текст программы обработчика события для компонента Timer2 будет следующим:

|  |
| --- |
| **procedure** TForm1.Timer2Timer(Sender: TObject); |
| **begin** |
| k:=k+1; |
| **if** k=1 **then** Image2.Visible:=true; |
| **if** k=2 **then** |
| **begin** |
| Image2.Width:=Image2.Width+10; |
| Image2.Height:=Image2.Height+10; |
| **end;** |
| **if** k=3 **then** |
| **begin** |
| Image2.Width:=Image2.Width-15; |
| Image2.Height:=Image2.Height-15; |
| **end;** |
| **if** k=4 **then** Image2.Visible:=false; |
| **if** k=5 **then** k:=1; |
| **end;** |

Аналогичные обработчики событий OnTimer будут у каждой звезды. Например, у второй звезды Image3 текст программы обработчика события для Timer3 будет следующим:

|  |
| --- |
| **procedure** TForm1.Timer3Timer(Sender: TObject); |
| **begin** |
| j:=j+1; |
| **if** j=1 **then** Image3.Visible:=true; |
| **if** j=2 **then** |
| **begin** |
| Image3.Width:=Image3.Width+10; |
| Image3.Height:=Image3.Height+10; |
| **end;** |
| **if** j=3 **then** |
| **begin** |
| Image3.Width:=Image3.Width-15; |
| Image3.Height:=Image3.Height-15; |
| **end;** |
| **if** j=4 **then** Image3.Visible:=false; |
| **if** j=5 **then** k:=1; |
| **end;** |

Звезды развиваются независимо друг от друга. Добавим на форму еще один компонент TTimer (Timer17), который включает предыдущие таймеры независимо друг от друга. Установим у этого таймера следующие свойства: Enabled = true, Interval = 500. В обработчике события для Timer17 с помощью случайной переменной n задается работа каждого из компонентов Timer2-Timer16.

Если n=1, то включается компонент Timer2 у первой звезды Image2: Timer2.Enabled:=true. После однократного выполнения обработчика события для Timer2, необходимо подготовить его к повторному действию. Для этого изображение Image2 должно стать видимым и вернуть свои начальные размеры: Image2.Visible:=true, Image2.Wight:=12, Image2.Height:=12. Аналогично включаются остальные таймеры.

Текст программы обработчика события для Timer17 будет следующим:

|  |
| --- |
| **procedure** TForm1.Timer17Timer(Sender: TObject); |
| **begin** |
| n:=random(20); |
| **if** n=1 **then** |
| **begin** |
| Timer2.Enabled:=true; |
| Image2.Visible:=true; |
| Image2.Wight:=12; |
| Image2.Height:=12; |
| **end;** |
| **if** n=2 **then** |
| **begin** |
| Timer3.Enabled:=true; |
| Image3.Visible:=true; |
| Image3.Wight:=12; |
| Image3.Height:=12; |
| **end;** |
| **if** n=3 **then** |
| **begin** |
| Timer4.Enabled:=true; |
| Image4.Visible:=true; |
| Image4.Wight:=12; |
| Image4.Height:=12; |
| **end;** |
| // аналогично для остальных n. |
| **end;** |

Полностью текст проекта «Летающая тарелка» с добавлением проекта «Жизнь звезд» выглядит следующим образом:

|  |
| --- |
| **procedure** TForm1.Timer1Timer(Sender: TObject); |
| **begin** |
| randomize; |
| h:=random(15); |
| Image1.Left:=Image1.Left+h; |
| **if** Image1.Left>=Form1.Width **then** Image1.Left:=1; |
| **end;** |
| **procedure** TForm1.Timer2Timer(Sender: TObject); |
| **begin** |
| k:=k+1; |
| **if** k=1 **then** Image2.Visible:=true; |
| **if** k=2 **then** |
| **begin** |
| Image2.Width:=Image2.Width+10; |
| Image2.Height:=Image2.Height+10; |
| **end;** |
| **if** k=3 **then** |
| **begin** |
| Image2.Width:=Image2.Width-15; |
| Image2.Height:=Image2.Height-15; |
| **end;** |
| **if** k=4 **then** Image2.Visible:=false; |
| **if** k=5 **then** k:=1; |
| **end;** |
| **procedure** TForm1.Timer3Timer(Sender: TObject); |
| **begin** |
| j:=j+1; |
| **if** j=1 **then** Image3.Visible:=true; |
| **if** j=2 **then** |
| **begin** |
| Image3.Width:=Image3.Width+10; |
| Image3.Height:=Image3.Height+10; |
| **end;** |
| **if** j=3 **then** |
| **begin** |
| Image3.Width:=Image3.Width-15; |
| Image3.Height:=Image3.Height-15; |
| **end;** |
| **if** j=4 **then** Image3.Visible:=false; |
| **if** j=5 **then** k:=1; |
| **end;** |
| // аналогично для компонентов Timer4-Timer16 |
| **procedure** TForm1.Timer17Timer(Sender: TObject); |
| **begin** |
| n:=random(20); |
| **if** n=1 **then** |
| **begin** |
| Timer2.Enabled:=true; |
| Image2.Visible:=true; |
| Image2.Wight:=12; |
| Image2.Height:=12; |
| **end;** |
| **if** n=2 **then** |
| **begin** |
| Timer3.Enabled:=true; |
| Image3.Visible:=true; |
| Image3.Wight:=12; |
| Image3.Height:=12; |
| **end;** |
| **if** n=3 **then** |
| **begin** |
| Timer4.Enabled:=true; |
| Image4.Visible:=true; |
| Image4.Wight:=12; |
| Image4.Height:=12; |
| **end;** |
| // аналогично для остальных n. |
| **end;** |
| **End.** |

**Задания**

1. Создать программу, в которой происходит запуск ракеты.

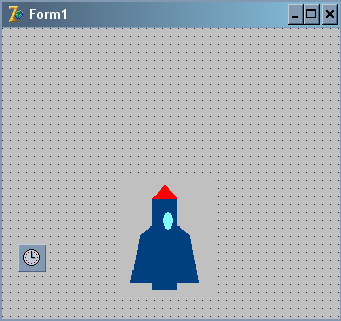


Рис. 124 Графический интерфейс проекта

1. Составить программу, в которой по дороге движется автомобиль. Если автомобиль уезжает за пределы окна, то он должна появиться снова у левой границы.
2. Создать проект, в котором один автомобиль едет за другим и обгоняет его.
3. Составить программу, в которой помещенный на форму квадрат постепенно растет. Если размеры квадрата превысят размеры окна, то квадрат должен вновь стать маленьким.
4. Создать программу, в которой навстречу друг другу летят два космических корабля. При столкновении этих кораблей происходит взрыв и от кораблей остаются только обломки.

**Литература**

1. Бадд Т. Объектно-ориентированное программирование в действии. – СПб.: изд-во [«Питер»](http://www.piter.com/), 2008
2. Бобровский С.И. Delphi 7. Учебный курс. – СПб.: изд-во Питер, 2008. – 736 с.
3. Боровский А.Н. Программирование в Delphi. – СПб.:«БХВ-Петербург», 2005 – 446 с.
4. Вострокнутов И.Е. Основы программирования в Турбо Паскаль. Ядро. Учебное пособие для Школы программистов СГА. – М.: Современная гуманитарная академия. 88 с.
5. Гейн А.Г. и др. Информатика, 10 – 11 кл. – СПб.: изд-во «Питер», 2005.
6. Голицына О.Л., Попова И.И. Основы алгоритмизации и программирования: учеб. пособие. – М.: Форум: ИНФРА-М, 2002.
7. Гофман В., Хомоненко А. Delphi. Быстрый старт. – СПб.:«БХВ-Петербург», 2003 – 288 с.
8. Каймин В.А. Информатика: учебник. – М.: изд-во «ИНФРА-М», 2000.
9. Лапчик М.П., Семакин И.Г., Хеннер Е.К. Методика преподавания информатики: учеб. пособие для студентов пед. вузов. – М.: Изд-й центр «Академия», 2005.
10. Макарова Н.В. Информатика, 10 кл. – СПб.: «Питер», 2005.
11. Пестриков В.М. Маслобоев А.Н. Delphi на примерах. – Спб.: .:«БХВ-Петербург», 2005 – 496 с.
12. Семакин И.Г., Хеннер Е.К. Информатика, 10 кл. – М.: изд-во «БИНОМ», 2005.
13. Семакин И.Г., Шестаков А.П. Основы программирования: Учебник. – М.: Высшая школа, 2001.
14. Угринович Н.Д. Информатика и информационные технологии: Учебник для 10-11 классов. – М.: БИНОМ. Лаборатория знаний, 2005. – 511 с.
15. Угринович Н.Д. Информатика и ИКТ: Учебник для 10 класса. Профильный уровень. – М.: БИНОМ. Лаборатория знаний, 2007. – 371 с.
16. Фаронов В.В. Delphi 4. Учебный курс. – М.: «Нолидж», 1998, - 464 с.
17. Фленов М. Е. Библия для программиста в среде Delphi. – СПб.:«БХВ-Петербург», 2008 – 799 с.
18. Фленов М. Е. Программирование в Delphi глазами хакера. - СПб.:«БХВ-Петербург», 2003 – 368 с.
19. Шауцукова Л.З. Информатика: учеб. пособие для 10–11 кл. общеобразоват. учреждений. – М.: «Просвещение», 2002.
20. Единая система программной документации. Схемы алгоритмов, программ, данных и систем. Условные обозначения и правила выполнения. ГОСТ 19.701-90 (ИСО 5807-85). – М.: Издательство стандартов, 1991.
21. Иллюстрированный самоучитель по Delphi 7 для начинающих. – СПб,: «Питер», 2001 – 496 с.
22. Конспекты уроков по информатике в 9-11 классах: практикум по программированию/авт.-сост. А.А. Чернов. – Волгоград: изд-во «Учитель», 2006.
23. Сборник задач по курсу информатики/под редакцией Л.И. Белоусовой. – М.: Издательство «Экзамен», 2007. – 253 с.
24. Языки программирования в школе – Викиучебник.

**Приложение 1**

**Таблица 10. Пиктографические кнопки**

|  |  |
| --- | --- |
| **Кнопка** | **Реализуемое кнопкой действие** |
| **Группа Standard** | |
| мал | Создает новый проект. Эквивалент опции File/New. |
|  | Открывает существующий файл. Эквивалент опции File/Open File |
|  | Сохраняет текущий модуль. Эквивалент опции File/Save |
|  | Сохраняет все модули проекта. Эквивалент опции File/Save All |
|  | Открывает созданный ранее проект программы. Эквивалент опции File/Open Project |
|  | Добавляет новый файл к проекту. Эквивалент опции File/Add File. |
|  | Удаляет файл из проекта. Эквивалент опции File/Remove File. |
| **Группа View** | |
|  | Выбирает модуль из списка модулей, связанных с текущем проектом. Эквивалент опции View/Units. |
|  | Выбирает форму из списка форм, связанных с текущим проектом. Эквивалент опции View/Forms. |
|  | Переключает активность между окном формы и окном кода программы. Эквивалент опции View/Toggle Form/Unit. |
|  | Создает новую форму и добавляет её к проекту. Эквивалент опции File/New Form. |
| **Группа Debug** | |
|  | Компилирует и выполняет программу. Эквивалент опции Run/Run. |
|  | Реализует паузу в работе отлаживаемой программы. Эквивалент опции Run/Program Pause. |
|  | Осуществляет пошаговую трассировку программы с прослеживанием работы вызываемых подпрограмм. Эквивалент опции Run/Trace Info. |
|  | Осуществляет пошаговую трассировку программы, но не прослеживает работы вызываемых подпрограмм. Эквивалент опции Run/Step Over. |
| **Группа Custome** | |
|  | Открывает доступ к встроенной справочной службе. Эквивалент опции Help/Contents. |

**Приложение 2**

**Таблица 11. Панель компонентов Standard**

|  |  |  |
| --- | --- | --- |
| **Компонент** | **Внешний вид** | **Назначение** |
| Надпись (TLabel) | компоненты | Предназначен для размещения на форме различного рода текстовых надписей. |
| Текстовое поле (TEdit) | компоненты | Предназначен для ввода однострочного текста. |
| Кнопка (TBotton) | компоненты | Используется для управления программой. |
| Текстовая область (TMemo) | компоненты | Предназначен для ввода, редактирования и отображения многострочного текста |
| Флажок (TCheckBox) | компоненты | Используется для фиксации включенного или выключенного состояния (одного из двух). |
| Переключатель (TRadioButton) | компоненты | Предназначен для выбора одного значения из ряда возможных. |
| Группа переключателей (TRadioGroup) | компоненты | Специальный контейнер для размещения переключателей |
| Список выбора (TListBox) | компоненты | Позволяет выбрать одну или несколько строк в списке. |
| Выпадающий список (TComboBox) | компоненты | Вариант списка с дополнительным полем, в котором отображается выбранный элемент списка. |
| Панель (TPanel) | компоненты | Контейнер общего назначения. |
| Группа компонентов (TGroupBox) | компоненты | Контейнер для размещения управляющих элементов с одинаковыми функциями. |
| Полоса прокрутки (TScrollBar) | компоненты | Используется как вспомогательный инструмент в других элементах управления, может применяться в качестве ползунка. |
| Главное меню (TMainMenu) | компоненты | Предназначен для создания главного меню |
| Контекстное меню (TPopupMenu) | компоненты | Предназначен для создания контекстного меню |

**Приложение 3**

**Таблица 12. Некоторые компоненты панелей Additional и System**

|  |  |  |
| --- | --- | --- |
| **Компонент** | **Внешний вид** | **Назначение** |
| **Панель Additional (дополнительные компоненты)** | | |
| Кнопка с картинкой (TBitBtn) | комп23 | Предназначен для создания кнопки с картинкой. |
| Быстрая кнопка (TSpeedButton) | комп23 | Используется для формирования панелей управления с «быстрыми» командными кнопками. |
| Шаблон ввода (TMaskEdit) | комп23 | Позволяет вводить данные в текстовое поле по заданному шаблону. |
| Рамка (TBevel) | комп23 | Используется для создания рамок и линий оформления. |
| Изображение (TImage) | комп23 | Предназначен для размещения на форме изображений. |
| Фигура (TShape) | комп23 | Предназначен для отображения на форме геометрических фигур. |
| **Панель System (системные компоненты)** | | |
| Таймер (TTimer) | л2 | Используется для отсчета интервалов реального времени. |
| Мультимедийный проигрыватель (TMediaPlayer) | л2 | Предназначен для воспроизведения музыкальных и видеоклипов. |
| Область рисования (TPainBox) | л2 | Окно с канвой для рисования изображений. |

Н.Г. Саблукова

Программирование в среде Delphi.

Ч.1 Основные команды. Первые проекты.

Подписано в печать 27.05.17

Формат 60×90/16. Бумага офсетная.

Усл. печ. листов 5,6.

Тираж 1000. Заказ № 108

Участок офсетной печати: 607220, г. Арзамас, Нижегородской области, ул. К.Маркса, 36