

## Введение

Язык формирует наш способ мышления и определяет, чем мы можем мыслить.

Б.Л. Ворф

Визуальный язык программирования формирует наше воображение и определяет, что мы можем себе представить.

В традиционной процедурной модели программирования выполнение программы начинается с первой строки и следует стандартным путем с вызовом, по мере необходимости, тех или иных процедур. В объектно-ориентированных языках реализуется событийно-управляемая модель программирования, т.е. программа выполняется, не следуя строго определенным путем, а в зависимости от того, какие наступают события.

Эти события вызываются либо действиями пользователя, либо сообщениями от системы. Таким образом, последовательность выполнения запущенной программы определяется последовательностью событий. При последующем запуске программы последовательность может быть иной.

В мире существует несколько различных сред визуальной разработки: Visual Basic, Delphi, Borland C++ Builder и другие. Каждая рассчитана на свой язык программирования, от которого зависят правила записи программ. Готовая программа состоит из операторов этого языка, причем часть этих операторов записывают вручную, а часть операторов система программирования подставляют автоматически.

Системы визуального объектно-ориентированного программирования предоставляют разработчику множество средств для создания объектов и настройки их свойств и методов. Свойства объектов всегда имеют названия, процесс настройки можно выполнять с помощью специальных графических средств, не прибегая к ручному кодированию. Эта особенность визуальных систем разработки значительно повышает производительность программиста и делает создание больших программ достаточно простым.

Мы будем использовать одну из самых мощных систем визуальной разработки программ — среду Delphi. Язык программирования, который использован в ней, называется Object Pascal (Объектный Паскаль).

## 1. Идеология объектно-ориентированного программирования

Объектно-ориентированное программирование (object-oriented programming) или событийно-управляемое программирование (event-driven programming) – программирование, направленное на объекты. Объект (object) – это все то, что нас окружает, и с чем мы можем взаимодействовать. Объекты могут быть информационными и материальными.

В программировании мы имеем дело с созданием моделей информационного типа, отражающих (описывающих) свойства и поведение некоторых реальных объектов. В визуальном объектно-ориентированном программировании в качестве объектов могут выступать и объекты самого языка.

В каждый момент времени объект характеризуется присущим именно ему набором свойств (properties) и методов (methods) – операций, совершаемых над другими объектами или данным объектом, а также реагирует на события (events).

Если взять набор объектов, как правило, одного и того же типа, хотя и не обязательно, то мы получим семейство или коллекцию (collection), которая в свою очередь, тоже является объектом. Чтобы обращаться к элементам коллекции, каждому объекту присваивается уникальное имя или номер.

Способность реагировать на определенные события – это разновидность свойства. При возникновении события производится его обработка. Объекты, используемые в компьютерных программах, реагируют на события. При возникновении события происходит автоматический запуск специального метода – обработчика данного события. С помощью событий обеспечивается взаимодействие программы с пользователем. События, связанные с действиями пользователя, называются пользовательскими. Кроме пользовательских событий в программах происходят программные события.

Объединение в объекте его свойств и методов называют инкапсуляцией (encapsulation). Инкапсуляция означает, что объект инкапсулирует (содержит) в себе свойства и методы, но описывать мы ничего не должны. Под термином «инкапсуляция» подразумевается то, что мы работаем (взаимодействуем) с объектом, совершенно не зная об его устройстве. Синонимом инкапсуляции может служить термин «сокрытие информации» (information hiding) о конструкции объекта. Итак, объект можно определить как продукт инкапсуляции данных вместе с кодом, предназначенным для их обработки.

### 1.1. Элементы интерфейса программы

Delphi – интегрированная среда разработки (Integrated Development Environment – IDE). Delphi дает возможность создавать программы в стиле визуального конструирования формы, разместив на ней какие-либо визуальные элементы. Delphi имеет сложный интерфейс.

В Delphi имеются 10 окон, но после загрузки появляются четыре окна:

- главное окно Delphi - <имя проекта>;
- окно с формой для проектирования приложения Form1 (окно проектировщика формы);
- окно инспектора объектов Object Inspector;
- окно редактора кода Unit1.pas.

В главном окне реализуются основные функции управления проектом создаваемой программы. Главное управляющее окно системы Delphi обычно располагается в верхней части экрана. Оно содержит средства управления созданием программы и выглядит наиболее загруженным из всех окон

Это окно содержит:

- строку заголовка;
- строку меню;
- панель инструментов;

- Строка заголовка главного окна отображает имя открытого в данный момент проекта Project1. Под строкой заголовка окна системы располагается строка меню, содержащая команды системы Delphi.



Под строкой меню располагаются *панели инструментов* с кнопками. Во многих программах имеется только одна такая панель, но в системе Delphi их несколько. Панели инструментов предназначены для выполнения некоторых команд, реализуемых главным меню. Кнопки панелей инструментов обеспечивают доступ к наиболее часто встречающимся командам. Чтобы узнать, как называется та или иная кнопка, надо навести на нее указатель мыши и подождать, пока рядом с ним появится всплывающая подсказка. На этой панели есть, в частности, кнопка сохранения проекта на диске, кнопка открытия проекта, кнопка запуска программы на выполнение.

Одна из панелей инструментов, имеющихся в главном окне системы Delphi, заметно отличается от остальных. Это *палитра компонентов*. Палитра компонентов устроена в виде набора пиктограмм. Палитра компонентов содержит множество вкладок. Каждая из них содержит свой набор компонентов. Общее число компонентов достигает нескольких сотен. Совокупность наборов составляет библиотеку визуальных компонентов (Visual Components Library - VCL). Имеется несколько категорий компонентов, каждая из которых расположена на своей вкладке. С помощью палитры компонентов создаются экземпляры компонентов (объекты) на форме.



Окно *инспектора объектов* (Object Inspector) отображает свойства какого-либо компонента, активизированного щелчком мышью, или самой формы. Имя активизированного компонента находится под заголовком окна. Именно с помощью инспектора объектов настраивают объекты, используемые в программах. Это же окно используется для выбора и настройки событий, на которые будут реагировать объекты нашей программы. С его помощью создаются или выбираются нужные процедуры обработки.

Окно *проектировщика формы* – главное место, где происходит сборка программы из компонентов, содержащихся в палитре компонентов. Сама форма – это уже готовая к выполнению программа. Вначале окно формы пустое, но, создавая программу, в указанное место формы добавляются объекты – экземпляры компонента выбранного типа. Сетка из точек в окне формы поможет разместить объекты ровно и аккуратно. При работе программы ее не видно.

Форма — это заготовка окна будущей программы. Каждая программа содержит хотя бы одно окно и, следовательно, одну форму. Поэтому при работе над программой окно формы мы видим на экране всегда. Заготовка первого окна называется Form1. Если в программе будет два окна, то заготовка второго будет называться Form2 и так далее. Возможна замена стандартного названия более подходящим для программы.



Рис. 3.

Последнее из открытых окон содержит *код программы*. Часть программы система Delphi формирует автоматически. Даже «пустая» программа для Windows собирается из нескольких тысяч операторов языка Pascal. Добавлять нужные операторы система Delphi начинает еще до того, как программист приступил к созданию своей программы.

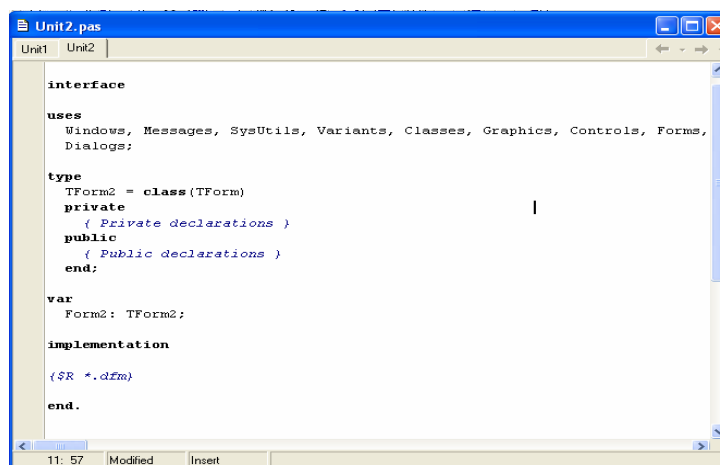


Рис. 4.

Добавляются операторы именно в этом окне. Некоторые операторы система Delphi добавит сама, другие мы введем вручную. Именно поэтому окно кода никогда не бывает пустым.

Для системы Delphi каждая незавершенная программа – это проект. Проект включает в себя множество файлов. Наиболее важными являются три файла: файл формы, файл кода и файл проекта.

Проект, состоит из:

- файла проекта Project1.dpf,
- файла параметров проекта Project1.dof,
- файла ресурсов проекта Project.res,
- файла настроек проекта Project1.cfg,
- файла описания формы Unit1.dfm,
- файла модуля формы Unit1.pas.

Файл модуля формы доступен для редактирования, именно он отображается в Редакторе Кода. Остальные файлы создаются Delphi автоматически. В процессе компиляции программы файлы преобразуются в исполняемый exe-файл, который, по умолчанию, создается в той же папке, в которой расположен файл проекта.

В проекте могут быть задействованы несколько форм, а также дополнительные модули и файлы ресурсов, при этом схема компиляции остается похожей.

Для сохранения проекта нужно воспользоваться пунктом главного меню Save Project As...

Чтобы не смешивать файлы разных проектов, желательно каждый новый проект сохранять в отдельную папку. Сначала предлагается ввести имя для модуля формы, а затем имя проекта.

Когда программа сохранена, можно начать написание новой программы. Для этого выбирается пункт главного меню File → New Application.

В результате на экране появляется пустая заготовка формы. Если же необходимо загрузить старую программу, нужно выбрать в главном меню Open Project либо воспользоваться кнопкой на панели инструментов и загрузить соответствующий файл проекта.

Вместе со средой Delphi поставляется более сотни различных компонентов, а в Интернете можно найти тысячи. Они отличаются по своему внешнему виду, функциям, способам реализации, требованиям к аппаратной части и т.п., но, тем не менее, все они построены на едином фундаменте, единой объектной основе.

Несмотря на различие форм и функций компонентов, их можно разбить на группы и классифицировать. Все компоненты можно разделить на визуальные и невидимые. Визуальные, в свою очередь, делятся на оконные элементы управления и графические элементы управления.

Визуальные компоненты являются элементами управления, которые добавляются на форму: это кнопки, поля ввода, картинки, панели и т.д. Те из них, что предназначены для ввода данных, могут принимать фокус ввода, обеспечивая управление с клавиатуры.

Невидимые компоненты служат в основном для обеспечения более удобного доступа к определенным функциям системы. При добавлении их на форму на ней появляется иконка с изображением компонента, невидимая во время исполнения программы. В этом и заключается их «невизуальность». Несмотря на это, некоторые из них служат для обеспечения визуальных по природе эффектов. Например, к таким относятся компоненты меню и диалоги.

Визуальные компоненты составляют интерфейс программы. Визуальные компоненты имеют много общих свойств и событий, связанных, прежде всего, с их визуальным отображением на форме.

Часть свойств отвечает за положение на форме: отступ слева, отступ сверху, высота, ширина, выравнивание.

Некоторые определяют внешний вид: цвет, шрифт, заголовок. Некоторые задают поведение компонента во время исполнения программы: доступен, виден, курсор, подсказка.

Механизм событий позволяет связать действия пользователя с необходимыми действиями программы. События определяют всю структуру программы, поскольку фактически весь код, то есть, собственно работа программиста, пишется в обработчиках событий.

В Windows действия пользователя обычно сводятся к нажатиям мыши в определенных местах экрана, нажатию клавиш клавиатуры, переключению между окнами и компонентами. В соответствии с этим существуют и события, реагирующие на нажатие мыши и клавиатуры, переключение фокуса, активацию окон.

Кроме того, для каждого компонента могут существовать характерные для него события, на которые может потребоваться реакция и обработка, например, событие изменения текста в поле ввода.

Некоторые конкретные компоненты и их предназначение.

- Для вывода надписей предназначен компонент Label.
- Для ввода текста пользователем используется компонент Edit.
- Для работы с несколькими строками используют Memo. Строки хранятся в свойстве Lines.
- Кнопка реализуется компонентом Button.
- Для работы с пунктами используется CheckBox.
- Для выбора текста из нескольких альтернатив предназначен компонент ComboBox. Альтернативы хранятся в свойстве Items, аналогичном свойству Lines у компонента Memo.
- Для множественного выбора используется RadioGroup. Чтобы добавить в него пункты, щелкните на кнопку свойства Items и введите названия пунктов. Номер выделенного пункта соответствует свойству ItemIndex, нумерация с нуля.
- Простейшим контейнером компонентов может служить компонент Panel. На него можно поместить другой компонент, например, кнопку. При перемещении контейнера передвигаются все находящиеся в нем компоненты.
- Для работы с картинкой используется компонент Image. Картинку можно загрузить, щелкнув на кнопку свойства Picture и выбрав сохраненную картинку. Если загруженная картинка не совпадает по размерам с размерами компонента, ее можно масштабировать, установив свойство Stretch, либо изменить размеры компонента, установив свойство AutoSize. Задний фон картинки можно убрать, используя свойство Transparent.
- Для графического оформления используется компонент Bevel. Его вид можно настроить с помощью свойства Shape.
- Ввод целых чисел с помощью мыши можно обеспечить с помощью компонента TrackBar. Его основное свойство – Position.
- А для отображения процесса длительных расчетов можно использовать компонент ProgressBar. Его свойство Position отображает процент выполненной работы.

## 1.2. Инспектор объектов

Каждый элемент управления, как и сама форма, обязательно имеет название. Название по умолчанию формирует сама система. Кроме того, практически все элементы управления также характеризуются текстовой информационной строкой - надписью, которая первоначально совпадает с названием элемента. Например, кнопка на форме получает название и надпись Button1, а поле-надпись - название и надпись Label1. Эти надписи можно изменять.

Инспектор - одно из важнейших окон при работе в дизайнера. С его помощью настраиваются всевозможные свойства объектов, используемых в программе, а также возможные реакции программы на различные действия пользователя с этими элементами

управления. Многие составляющие части программы (прежде всего элементы управления) представляют собой объекты. Они характеризуются наборами свойств и методами, вызываемыми в различных ситуациях. У каждого элемента управления есть обязательно свойство Name, с помощью которого можно определить название этого объекта (на самом деле это название будет названием переменной в тексте программы, по которому к нему можно обращаться!). А текстовая надпись элемента задается с помощью свойства Caption. Некоторые свойства существуют практически у всех элементов управления, а некоторые - специфичны, так как предназначаются для определения конкретных особенностей работы определенного элемента управления.

В Инспекторе показывается список свойств, связанный с текущим элементом, выделенным на форме. Как только изменяется значение какого-нибудь свойства в окне, внесенные изменения сразу же будут отражены и в главном окне дизайнера.

## **2. Три основных принципа ООП: наследование, инкапсуляция, полиморфизм**

В объектно-ориентированном программировании объекты объединены в классы. Класс – это проект, план строения конкретных объектов. Класс определяет, какие у данных объектов будут переменные состояния, и как они будут изменяться, какие будут методы и как они будут реализовываться. Класс нематериален, он не соответствует каким-то данным. В классе описано, какие переменные и с какими значениями могут быть у объекта. Могут быть, а не есть в данный момент. Ведь у разных объектов, относящихся к этому классу, значения этих переменных в данный момент, возможно, различны.

Объект же, наоборот, материален. Он занимает место в памяти компьютера, все его переменные имеют в каждый момент вполне определенные значения. Именно для конкретных объектов вызываются методы, хотя сами методы описаны в классе, к которому принадлежит данный объект.

Для этого служит механизм наследования – один из трех основных принципов объектно-ориентированного программирования.

Наследование дает огромные преимущества и значительно облегчает программирование. Оно позволяет не только экономить время при создании новых классов, но и дает возможность неограниченного расширения и совершенствования уже существующих классов. Можно придать новые функции классу, даже не зная деталей реализации самого объекта класса.

Некоторые свойства и методы доступны внешним объектам, а некоторые целесообразно спрятать для внутреннего пользования.

Этот принцип сокрытия – второй принцип объектно-ориентированного программирования, называемый инкапсуляцией.

Одни переменные и методы, определяемые в разделе *Интерфейс*, доступны для чтения, изменения и вызова, называются опубликованными, а другие, определяемые в разделе *Реализация*, служат исключительно нуждам самого объекта и недоступны извне, они называются приватными.

При наследовании все параметры и методы, описанные в родительском классе, переходят без изменений в класс-потомок. Но не всегда требуется такое буквальное копирование. Иногда просто необходимо переопределить уже существующий метод или параметр. Для того чтобы при вызове одинакового метода различные потомки вели себя по-разному, требуется замещение метода, описанного в классе-предке.

Замещать можно и переменные. В подобном разноликом поведении и заключается третий принцип объектно-ориентированного программирования – Полиморфизм.

С помощью полиморфизма потомки классов получают возможность переопределять действия, заложенные в предках.

Например, графические компоненты управления в Delphi все наследуют от своего предка метод Перерисовки, но каждый из них его *замещает*. В результате, когда Windows

решает, что пора обновить окно и вызывает для них метод Перерисовать, каждый компонент перерисовывается по-своему.

В Delphi все классы наследуются от одного предка – базового класса TObject. Этот класс не только дает всем своим потомкам ряд нужных всем им методов, например, методы создания и уничтожения экземпляров класса, но и обеспечивает базовую совместимость любых объектов. Все компоненты в Delphi – объекты. Можно создавать новые компоненты в Delphi.

События используются в Delphi для обеспечения определения программистом реакции на те или иные действия или запросы со стороны компонента.

Хотя описание обработчиков событий, по сути, является всего лишь полиморфным замещением методов, события в Delphi играют очень важную роль. Фактически вся программа – это описание обработчиков событий.

Программа Delphi не является сплошным последовательным выполнением кода, как в Turbo Pascal. Здесь программа реализует некоторый код только как реакцию на события – какие-то действия пользователя (нажатие кнопок, движение мыши, закрытие окон и т.п.). Когда код, описанный в обработчике, заканчивается, программа не завершается. Для завершения требуется, чтобы пользователь обычным в Windows способом закрыл главное окно приложения либо, к примеру, нажал на сделанную кнопку, в обработчике нажатия которой предусмотрен вызов процедуры Close.



### 3. Использование справки

Чтобы получить справку, например, по кнопке Button, надо выделить его и нажать F1.

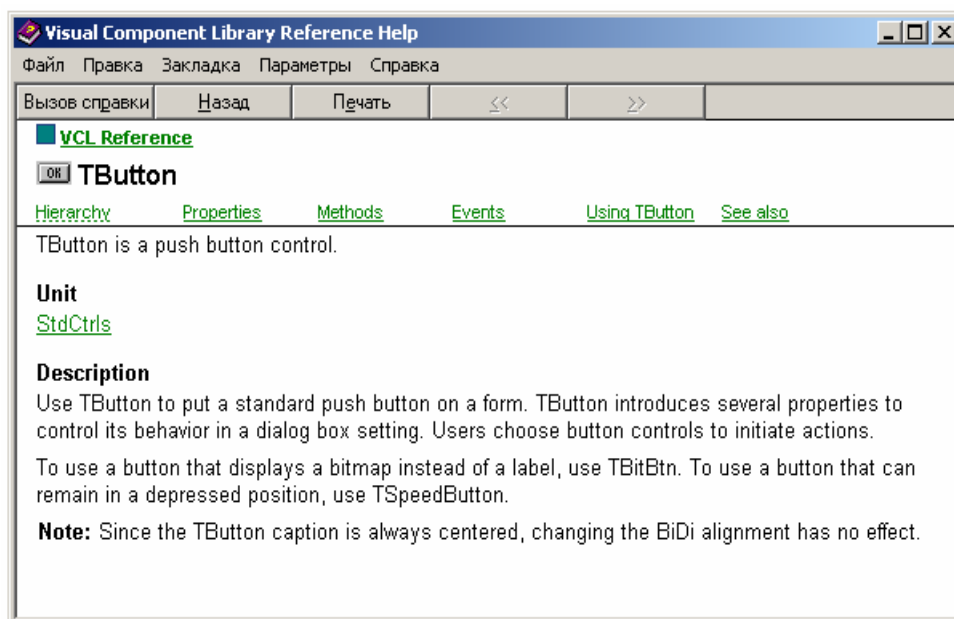


Рис. 5.

Откроется окно справки, в котором дано краткое описание компонента. Главное в справке – это описания свойств, методов, процедур и функций.

При щелчке на ссылку «Иерархия» появляется дерево объектов, являющихся предками данного объекта. Как уже отмечалось, самым базовым классом для любых объектов является TObject. От него наследуется TPersistent, промежуточный класс, обеспечивающий всем своим потомкам возможность работы с потоками. Его потомком является класс TComponent, который является предком абсолютно всех компонентов и т.д.

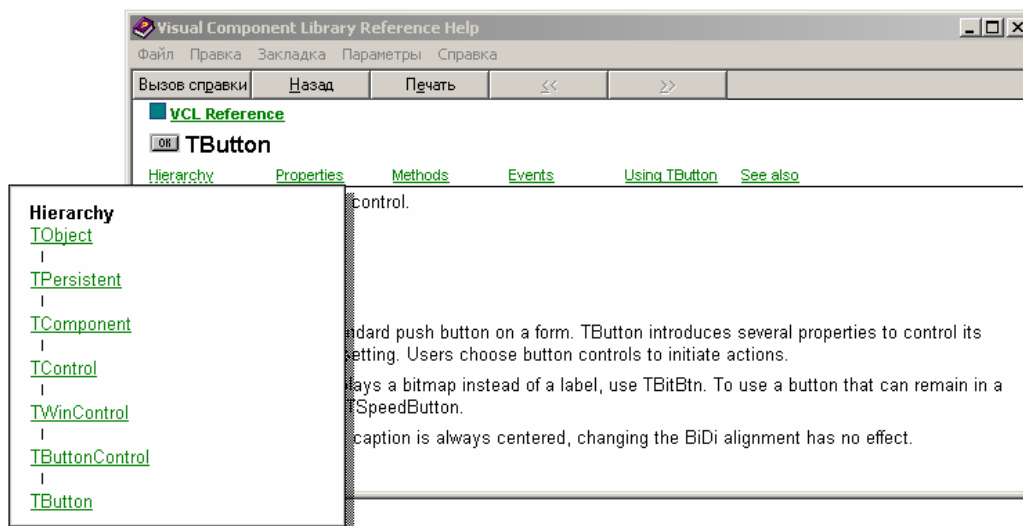


Рис. 6.

При щелчке по ссылке «Properties» откроется окно, в котором приведен список всех свойств компонента. В Инспекторе Объектов высвечиваются только помеченные зеленым квадратиком свойства.

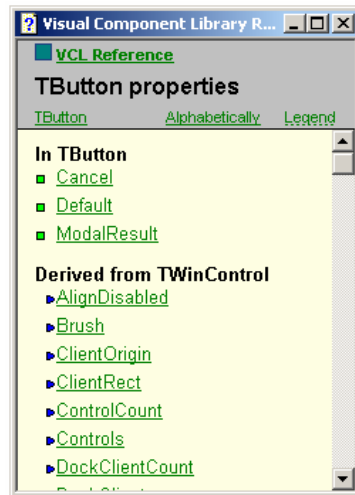


Рис. 7.

При щелчке по какому-нибудь свойству получаем его описание. Чтобы вернуться к описанию компонента, нажимают на ссылку.

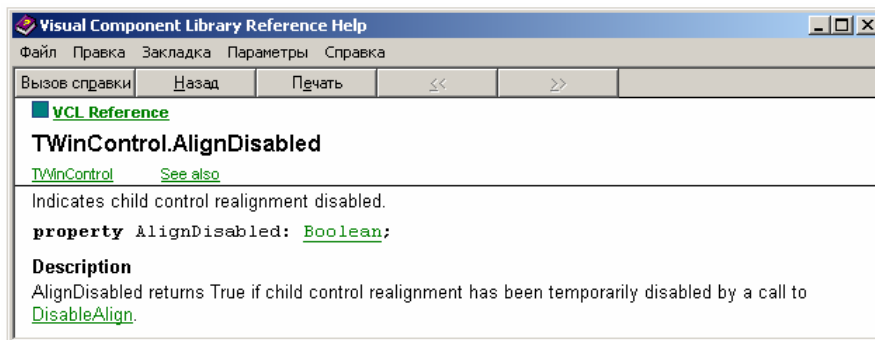


Рис. 8.

Аналогично можно получить и информацию о методах и событиях компонента, щелкнув по соответствующей ссылке.

Если нужно получить справку по какому-то ключевому слову, типу, процедуре, классу и т.п., необходимо поставить текстовый курсор посередине интересующего слова в коде и нажать F1.

Для функции дается кратное описание, указывается модуль, в котором она объявлена, даются варианты использования, часто можно посмотреть пример – Example.

Наконец, помощь в написании программы может оказать комбинация клавиш Ctrl+"пробел". Эта комбинация клавиш подсказывает, какие методы, свойства, процедуры и т.п. можно написать дальше.

При выборе из списка и нажатии Enter текст автоматически вводится.

#### 4. Структура программы

Модуль кода (unit) делится на две части – **интерфейс** (interface) и реализацию (implementation).

Раздел **interface**:

- **uses**: подключаемые модули, в которых содержатся используемые процедуры, функции, классы и т.п. Их список формируется автоматически в зависимости от добавленных в форму компонентов.
- **type**: описания типов. Автоматически в этом разделе описан класс вашей формы.

- **var**: описание глобальных переменных. Автоматически тут описана переменная типа описанного класса формы. В этой переменной во время работы программы хранится указатель на экземпляр формы.
- **const**: описание констант. Этот раздел автоматически не создается, однако может быть легко добавлен.

#### Раздел **implementation**:

- Здесь также можно добавить **uses**.
- С помощью **{SR}** подключаются файлы ресурсов. Автоматически подключен файл **dfm**, имеющий то же имя, что и файл модуля и потому подключаемый как **\*.dfm**. В этом файле хранится информация о форме – расположение и настройки компонентов, помещенных в форму, и т.п.
- Раздел содержит описание реализаций процедур и функций. Автоматически тут создаются обработчики событий. Здесь описывается реализация необходимых в программе процедур, функций и методов классов.

Все то, что касается объявления доступных из других модулей переменных, констант, процедур, типов, классов и т.п., должно быть описано в разделе **interface**. В этой части программы, которая создается автоматически, в разделе **uses** перечисляются используемые модули, в разделе **type** описывается класс формы – потомок класса **TForm**, а в разделе **var** описывается глобальная переменная, в которой будет храниться указатель на объект-экземпляр формы. Соответственно, если надо подключить какой-то свой модуль, то необходимо добавить его в раздел **uses**. При описании нового типа его добавляют в раздел **type** до или после класса формы. Если нужно описать глобальную переменную, то ее добавляют в раздел **var**, а если нужна константа – создается дополнительно раздел **const**. В свою очередь, в разделе **implementation** сначала указывается ссылка на файлы ресурсов, а затем описываются реализации процедур и функций – как создаваемых автоматически обработчиков событий, так и добавляемых по необходимости. Вначале эта часть пуста, но она будет заполняться во время написания программы.

При добавлении компонента в форму автоматически модифицируется код модуля – в описании класса формы появляется переменная, хранящая ссылку на добавленный компонент.

При создании обработчика события (при нажатии на компонент, помещенный в форму, создается обработчик основного события, а остальные создаются с помощью Инспектора Объектов) в коде происходят следующие изменения:

- в разделе **implementation** создается пустой обработчик, который вы потом заполняете;
- в классе вашей формы создается описание этого метода (обработчики событий являются методами);
- если не создавать обработчик через страницу **Events** Инспектора Объектов (или двойным щелчком по компоненту), а просто написать в коде процедуру-обработчик, то он не будет должным образом связан с компонентом, и событие не сработает.

Если изменить имя компонента, то изменятся имена и обработчиков событий, которые с ним связаны.

Если необходимо изменить название обработчика, то надо воспользоваться Инспектором Объектов. Переименовывая здесь имя процедуры обработчика, корректно автоматически изменяются также и ссылки на эту процедуру.

Если необходимо удалить какой-то обработчик, то удалите весь код внутри процедуры. При следующем запуске программы все пустые процедуры автоматически корректно удаляются. Компилятор автоматически удаляет пустые процедуры при запуске. При удалении всей процедуры описание и ссылки на нее останутся, а удаление их всех требует времени и аккуратности.

При удалении компонента не удаляются процедуры-обработчики его событий. Это связано с тем, что один и тот же обработчик может использоваться для разных компонентов.

В случае, если нужно удалить эти оставшиеся обработчики, удалите код внутри них и запустите программу.

Проблема обеспечения гарантированного выполнения некоторых действий решается наличием специального события OnCreate. Это событие формы. Оно возникает тогда, когда форма создается, а поскольку главная форма создается в самом начале работы приложения, то и код, описанный в обработчике данного события, выполнится один раз и в самом начале. В обработчике события OnCreate можно описать процедуры и функции инициализации, присваивать нужные значения глобальным переменным и т.п.

Раздел implementation предназначен не только для обработчиков событий. В этот раздел добавлять в него свои собственные функции и процедуры. Необходимо помнить, что они должны идти раньше процедур и обработчиков, из которых вызываются, или объявить их в разделе interface. В последнем случае эти функции и процедуры будут доступны для вызова из других модулей.

Существует также важная тонкость использования собственных функций: все обработчики событий имеют в своем названии имя формы:

**procedure** TForm1.FormCreate(Sender: TObject);

Это указывает на то, что работают с классом формы TForm1 и описывают методы для нее. Из обработчика просто обращаются к компонентам формы (класс формы «знает» о наличии компонентов на ней и имеет к ним непосредственный доступ), например:

MyGrid.RowCount := Num;

Когда описывается собственная процедура, то она «ничего не знает» о компонентах формы, и поэтому нужно обращаться к компонентам через форму, а, не минуя ее.

## 5. Методы отладки и борьбы с ошибками

От ошибок никто не застрахован, и они случаются даже у опытных программистов. Таким образом, все сводится к тому, чтобы их поскорее обнаружить. Допустим, сделана опечатка и неправильно написано какое-нибудь ключевое слово. В этом случае при попытке запуска программы внизу появляется окно, в котором отображены сообщения об ошибках. При этом программа не запустится. Благодаря подсказке такую ошибку легко найти и исправить.

Частой ошибкой начинающих является пропуск конструкции begin – end в цикле. При этом имеется в виду, что в цикле должны выполняться, например, обе команды, но на самом деле в цикле, конечно, будет выполняться только первая, а вторая выполнится только один раз – потом, когда программа выйдет из цикла. При попытке запуска появляется сообщение. Но это не ошибка, а предупреждение. В нем обращается внимание на то, что параметр цикла после выполнения цикла может быть неопределенным (он присутствует во второй команде).

Однако, несмотря на предупреждение, программа может запуститься. Если вводятся какие-либо данные, то появляется сообщение об исключительной ситуации – exception. При этом программа приостанавливается, переходя из режима исполнения в режим отладки. Чтобы перейти к обычному редактированию кода, лучше остановить программу. Это можно сделать с помощью команды Program Reset. Затем можно поправить ошибку и вновь запустить программу.

В большие программы всегда закрадываются ошибки. Их надо быстро и квалифицированно найти и исправить. Механизм исключительных ситуаций (exception) – одно из больших достоинств Delphi. С их помощью вы можете контролировать возникновение ошибок и создавать в результате устойчивые к ошибкам программы.

По мере знакомства с языком и средой программист проходит несколько этапов. На первом этапе он, по незнанию, путает типы, забывает ставить знаки препинания (например, точку с запятой в конце строки), некорректно использует операторы и т.п. В результате написанный им код в принципе невозможно исполнить. И это хорошо –

поскольку допущенные им ошибки оказываются автоматически выявленными на этапе компиляции, более того, часто среда программирования сама подсказывает, какая ошибка допущена, и, что важно, указывает строку, которую нужно поправить. По мере изучения языка и борьбы с синтаксическими ошибками программист плавно переходит к следующему этапу. Теперь он уже не делает таких простейших ошибок, но, поскольку сложность его программ возрастает, возрастает и вероятность совершения им ошибки, при которой программа все равно запустится. Поскольку, с точки зрения компилятора, явной ошибки нет, а некоторые странности кода, по-видимому, являются замыслом программиста. Однако компилятор все-таки сообщает об этих странностях с помощью предупреждений (Warning). Советуем всегда обращать на них внимание, проверять при их появлении, нет ли ошибки, и вообще стараться писать код так, чтобы не было предупреждений.

Опасность таких скрытых ошибок состоит:

- 1) в том, что они таятся в той части кода, которую программист написал и уверен, что она правильная (программа запустилась), а значит, и не очень внимательно будет искать ошибку;
- 2) в том, что проявляется эта ошибка совсем в другом месте кода – не в том, в котором допущена. А это приводит к долгим поискам ее по всей программе.

Наконец, когда программист становится уже опытным и приступает к сложным проектам, связанным с использованием объектов, указателей и т.п., на его пути стоят еще более опасные ошибки, о которых компилятор не выдает даже предупреждений. А ошибки эти весьма серьезные, поскольку они, в основном, связаны с некорректной записью в память и могут привести к непредсказуемым последствиям.

При возникновении исключительной ситуации можно проигнорировать ее и запустить исполнение программы дальше, нажав F9. В этом случае программа выдает сообщение об ошибке. Необходимо найти ошибку – понять, в какой строке и почему происходит сбой. Для этого можно воспользоваться трассировкой. Для того чтобы определить первую строчку, начиная с которой будет проводиться трассировка, нужно поставить Breakpoint – точку останова.

Когда исполняемый код доходит до точки останова, исполнение программы приостанавливается, и надо перейти в режим отладки. В режиме отладки можно исполнять последовательно программу по шагам, контролировать и изменять значения переменных и т.п. Этот режим служит для обнаружения и ликвидации ошибок. В этом случае появляется возможность просмотреть или изменить текущие значения переменных, однако изменение кода во время отладки невозможно. Измененный текст заработает только после перезапуска программы.

Чтобы выполнить текущую строку, на которой стоит курсор отладки, нажмите F7 или F8. Строка выполнилась, и курсор сместился. Если необходимо перейти к следующей строке, то можно нажать F8, если нужно зайти в какую-либо функцию, то нажимают клавишу F7 и продолжают трассировку.

Для того чтобы в ходе отладки узнать значение той или иной переменной, нужно просто подвести к ней курсор. Появится hint со значением этой переменной.

Это, однако, работает не со всеми переменными, а только с доступными в данный момент. Если нужно постоянно контролировать значение переменной, то еще проще добавить ее в список Watch. Для более основательного слежения за значениями можете воспользоваться Списком Наблюдения (Watch List, Ctrl+F5).

Таким образом, при программировании среда Delphi может находиться в различных режимах:

- **Режим редактирования** – режим, в котором редактируется код проекта, модифицируется форма, добавляя на нее компоненты и настраивая их свойства. Это основной режим.
- **Режим исполнения** программы – режим, в который среда переходит, как только нажата клавиша F9 и был построен exe-файл. Фактически, в этом режиме происходит

как раз исполнение получившегося exe-файла проекта. Программа выполняется так, как если бы ее вызвали не из Delphi, а просто из Windows.

- **Режим отладки** – в этот режим можно перейти из режима исполнения программы. При этом программа будет приостановлена (но не остановлена совсем).

Чтобы продолжить трассировку (последовательный переход от команды к команде), можете воспользоваться клавишами:

- **F9 (Run)** – продолжить программу, не трассируя ее.
- **F8 (Step over)** – выполняется текущая строка кода, и переходят к следующей строке.
- **F7 (Trace Into)** – то же, что и F8, с тем отличием, что если в текущей строчке содержится вызов какой-либо функции или процедуры, то попадают внутрь этой процедуры и трассируют ее до конца, затем из нее возвращаются и переходят к следующей строке (на которую перешли бы сразу, если бы нажали F8).
- **F4 (Run to Cursor)** – переход в режим исполнения программы до тех пор, пока не должна будет выполнена строка, на которой стоит текстовый курсор (аналогично тому, как если бы была установлена точка останова)
- **Shift+F8 (Run Until Return)** – процедура выполняется до конца.
- **Ctrl+F2 (Program Reset)** – остановка трассировки и переход в режим редактирования кода. (Иногда целесообразнее, если это не грозит ошибками, продолжить исполнение программы (F9) и выйти из нее нормальным образом, закрыв главную форму).

При работе в Delphi сообщение об ошибке фактически появляется дважды: сначала выводится окно об исключительной ситуации и программа приостанавливается, а потом, если нажать F9 (F8, F7 и т.п.), – возникает стандартное сообщение об ошибке Windows.

Итак:

1. Произошла ошибка.
2. Программа приостанавливается.
3. Выводится сообщение об exception. Это сообщение для программиста. Среда Delphi сообщает, что программа не в состоянии выполнить какую-то свою команду. Программист не предусмотрел возможность исключительной ситуации. Среда Delphi приостанавливает программу, чтобы программист разобрался, где и в чем ошибка. Отключить приостановку (2)–(3) можно, сняв флажок Menu => Tools => Debugger Options => Language Exceptions => Stop on Delphi Exceptions.
4. Нажатие клавиши F9 (F8, F7 или др.).
5. Выводится сообщение об ошибке. Это сообщение для пользователя программы (ситуация запуска приложения не из Delphi, а через exe-файл из Windows, т.е. не существует пунктов 2, 3, 4).
6. Программа продолжается (при этом она не сумела выполнить ту часть, в которой произошла ошибка, и значит, если эта часть важная, продолжение может сопровождаться дальнейшими ошибками).

Механизм обработки исключительных ситуаций заключается в том, что если произошла ошибка (1) и не надо выводить (5), предпринимаются действия, чтобы (6) исполнялось корректно. Для этого «опасная» команда (или целый блок) помещается внутрь конструкции try..except..end или try..finally..end.

Блок try..finally..end используется аналогично try..except.., но с тем отличием, что блок команд между finally и end выполняется в любом случае, вне зависимости от того, было исключение между try и finally или нет.

## Практическая работа № 1 «Моя первая программа»

**Цель работы** - создать программу, выполняющую следующие действия:

1. После запуска программы появляется окно.

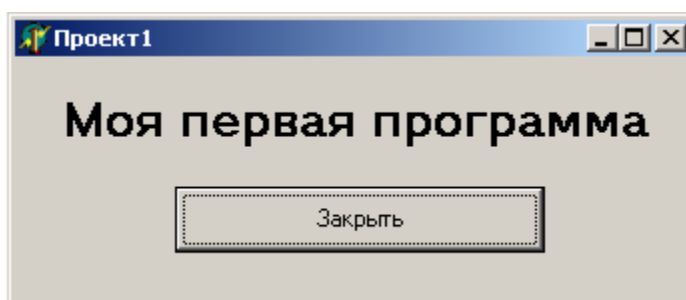


Рис. 9.

2. Для выхода из программы необходимо щелкнуть мышью на кнопке «Закреть».

### Описание плана разработки программы

1. Открыть новый проект.
2. Разместить на форме экземпляры компонентов: метку **Label** и кнопку **Button**.
3. Выполнить следующие действия:

Таблица 1.

Выделенный объект	Вкладка окна Object Inspector	Имя свойства/ имя события	Действие
<b>Form1</b>	Properties	Caption (надпись)	Установка имени формы «Проект1»
<b>Label1</b> (Вкладка Standard)	Properties	Caption	Ввод текста надписи «Моя первая программа»
		AutoSize (Автоподбор)	Ввод значения свойства: True
		Font → Color	Выбрать цвет: clPurple
<b>Button1</b> (Вкладка Standard)	Properties	Caption	Установка имени кнопки «Закреть»
	Events	OnClick	Close;

4. Сохраните проект, запустите и протестируйте его.

## Практическая работа № 2

### Создание консольного приложения

**Цель работы** - создать консольную программу.

1. Дать команду главного меню File > New > Other (Файл > Новый > Другое), и в диалоговом окне на закладке New выбрать значок Console Application (Консольное приложение).

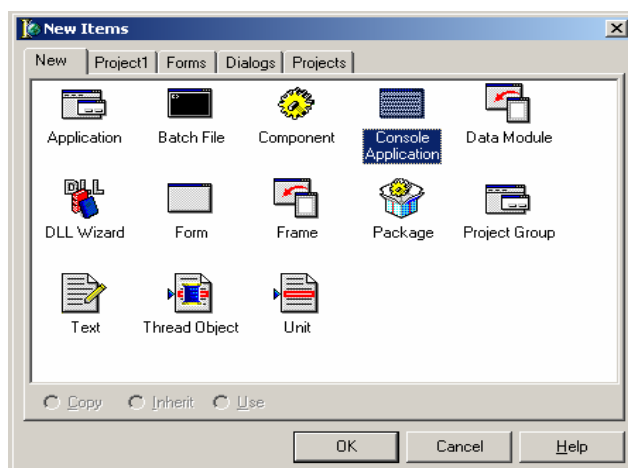


Рис. 10.

2. После нажатия на кнопку ОК экран примет следующий вид:

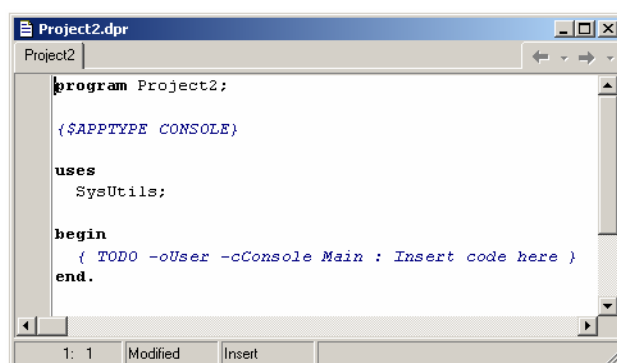


Рис. 11.

3. Текст был сгенерирован Delphi автоматически. Он представляет собой шаблон создаваемого приложения. Ничего изменять в этом тексте в принципе не надо - он соответствует готовой программе. Но внесем некоторые изменения.

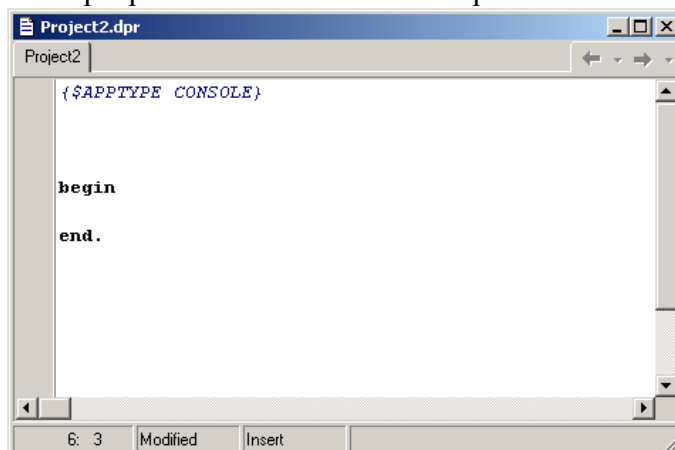


Рис. 12.



4. Введите текст программы в окно кода.

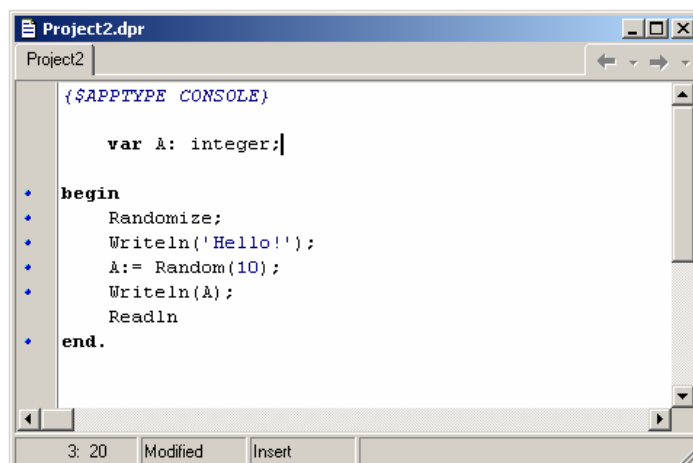


Рис. 13.

5. Выбрать команду Run. Получить результат. Нажать Enter для выхода в окно кода.

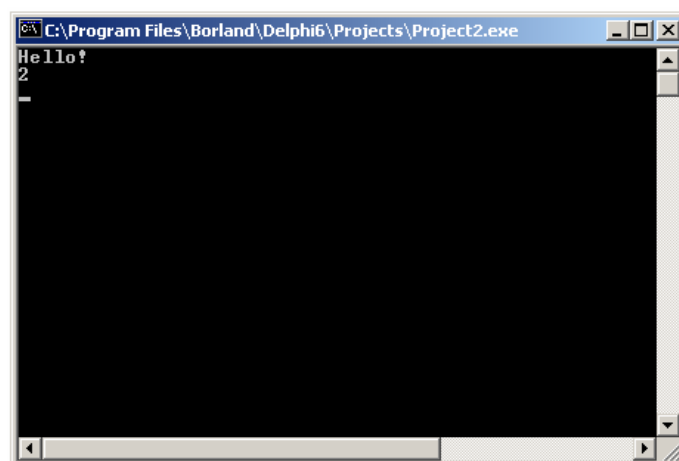


Рис. 14.

### Практическая работа № 3 «Приветствие»

**Цель работы** - создать программу, выполняющую следующие действия:

1. После запуска программы по щелчку мышью на кнопке «Приветствие» появляется сообщение «Первые успехи».

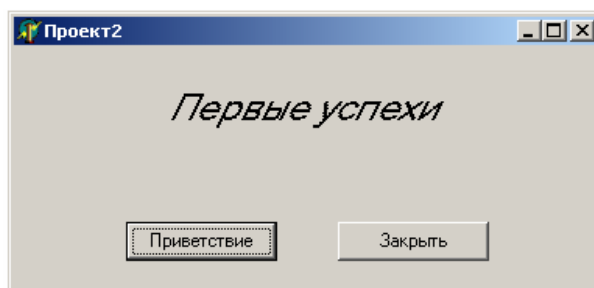


Рис. 15.

2. Для выхода из программы необходимо щелкнуть мышью на кнопке «Закреть».

#### **Описание плана разработки программы**

1. Открыть новый проект.
2. Разместить на форме экземпляры компонентов: метку **Label** и две кнопки **Button**.
3. Выполнить следующие действия:

Таблица 2.

Выделенный объект	Вкладка окна Object Inspector	Имя свойства/ имя события	Действие
<b>Form1</b>	Properties	Caption	Установка имени формы «Проект2»
<b>Label1</b>	Properties	Caption	Ввод текста надписи «Первые успехи»
<b>Button1</b>	Properties	Caption	Установка имени кнопки «Приветствие»
	Events	OnClick	Label1.Caption:='Первые успехи'
<b>Button2</b>	Properties	Caption	Установка имени кнопки «Закреть»
	Events	OnClick	Close;

4. Сохраните проект, запустите и протестируйте его.

#### **Задание для самостоятельного выполнения**

1. Сделать шрифт выводимой реплики «Первые успехи!» отличным от стандартного по виду, цвету и размеру.

*Подсказка.* В **Object Inspector** дважды щелкнуть справа от названия свойства **Font**.

2. Заменить вид кнопки «Выход» на более привлекательный.

*Подсказка.* Для замены кнопки удалить существующую, а другую **BitBtn** найти в палитре компонентов на вкладке **Additional**. Затем изменить ее вид с помощью свойства **Kind**.

3. Сделать так, чтобы после нажатия кнопки «Приветствие» на экране появлялось сообщение «Первые и не последние!».

*Подсказка.* Изменить значение свойства **Caption** метки **Label1** при реакции кнопки **Button1** на событие **OnClick**.

## Листинг программы

```
unit Unit1;  
  
interface  
  
uses  
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs, StdCtrls, Buttons;  
  
type  
  TForm1 = class(TForm)  
    Label1: TLabel;  
    Button1: TButton;  
    BitBtn1: TBitBtn;  
    procedure Button1Click(Sender: TObject);  
  
    private  
      {Private declarations }  
  
    public  
      {Public declarations }  
  end;  
  
  var  
    Form1: TForm1;  
  
  implementation  
    {$R *.DFM}  
  
    procedure TForm1.Button1Click(Sender: TObject);  
    begin  
      Label1.Caption:='Первые и не последние!';  
    end;  
  
  end.
```

## Практическая работа № 4 «Случайный выбор»

**Цель работы** - создать программу, выполняющую следующие действия:

1. После запуска программы появляется надпись «Брось кубик».

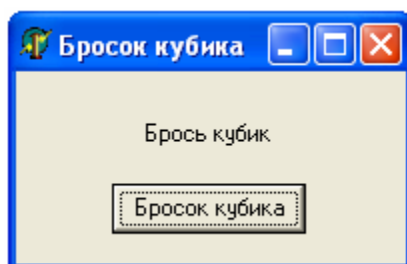


Рис. 16.

2. По щелчку мышью на кнопке «Бросок кубика» появляется сообщение, выдающее число в диапазоне 0 - 6.

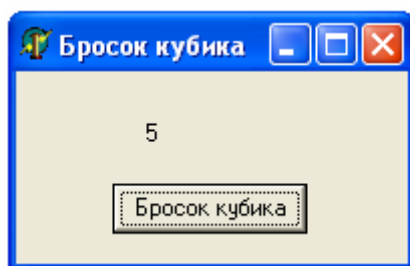


Рис. 17.

3. Для выхода из программы необходимо щелкнуть мышью на закрывающей кнопке в строке заголовка.

### Описание плана разработки программы

1. Открыть новый проект.
2. Разместить на форме экземпляры компонентов: метку **Label** и кнопку **Button**.
3. Выполнить следующие действия:

Таблица 3.

Выделенный объект	Вкладка окна Object Inspector	Имя свойства/ имя события	Действие
<b>Form1</b>	Properties	Caption	Установка имени формы «Бросок кубика»
<b>Label1</b>	Properties	Caption	Ввод текста надписи «Брось кубик»
<b>Button1</b>	Properties	Caption	Установка имени кнопки «Бросок кубика»
	Events	OnClick	Label1.Caption:= IntToStr(n);

4. Сохраните проект, запустите и протестируйте его.

### Фрагмент программы

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

var n: Integer;
```

```
begin  
  n := random (6) + 1 ;  
  Label1.Caption := IntToStr (n) ;  
end;  
end.
```

5. Создать функцию, отображающую числовые значения граней.  
Function Kubic (Sides:Integer): Integer;

```
Begin  
  If Sides >= 1 then  
    begin  
      Result := random (Sides) +1 ;  
    end  
  else  
    begin  
      Result :=0;  
    end;  
End;
```

## Практическая работа № 5

### Изменение заголовка формы

**Цель работы** - создать программу, выполняющую следующие действия:

1. После запуска программы ввести текст в текстовом поле.

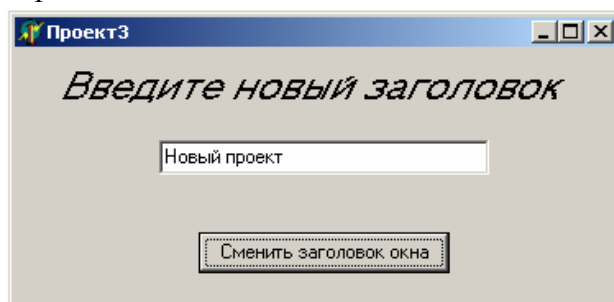


Рис. 18.

2. По щелчку мышью на кнопке «Сменить заголовок окна» изменяется заголовок окна.

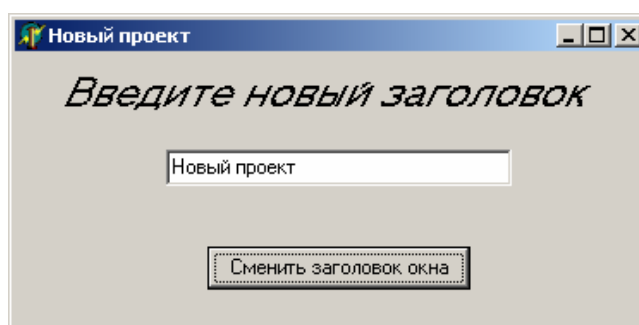


Рис. 19.

3. Ввести новый текст в текстовом поле.
4. Изменить название заголовка окна по нажатию клавиши Enter.
5. Для выхода из программы необходимо щелкнуть мышью на закрывающей кнопке в строке заголовка.

#### Описание плана разработки программы

1. Открыть новый проект.
2. Разместить на форме экземпляры компонентов: метку **Label**, кнопку **Button**, текстовое поле **Edit**.
3. Выполнить следующие действия:

Таблица 4.

Выделенный объект	Вкладка окна Object Inspector	Имя свойства/ имя события	Действие
<b>Form1</b>	Properties	Caption	Установка имени формы «Проект3»
<b>Label1</b> (Вкладка Standard)	Properties	Caption	Ввод текста надписи «Введите новый заголовок:»
<b>Edit1</b> (Вкладка Standard)	Properties	Text	Очистить значение свойства Text
<b>Button1</b> (Вкладка Standard)	Properties	Caption	Установка имени кнопки «Сменить заголовок окна»
		Default	Выбрать в раскрывающемся списке значение True
	Events	OnClick	Form1.Caption := Edit1.Text;

4. Сохраните проект, запустите и протестируйте его.

### Задание для самостоятельного выполнения

Создать программу, выполняющую следующие действия:

1. После запуска программы отображаются: две строки для ввода текущих курсов для евро и доллара; строка для ввода денежной суммы в рублях; две строки для вывода эквивалента в евро, долларах.



Рис. 20.

2. Ввести текущий курс для евро и доллара.
3. Ввести денежную сумму в рублях.
4. По щелчку мышью на кнопке «Подсчитать эквивалент» выводится денежная сумма в евро и долларах.
5. Ввести новый текущий курс для евро и доллара.
6. Ввести новую денежную сумму в рублях.
7. По щелчку мышью на кнопке «Подсчитать эквивалент» выводится новая денежная сумма в евро и долларах.
8. Для выхода из программы необходимо щелкнуть мышью на закрывающей кнопке в строке заголовка.

## Практическая работа № 6 «Двигающаяся кнопка»

**Цель работы** - создать программу, выполняющую следующие действия:

1. По щелчку мышью на кнопке кнопка либо останавливается, либо двигается.

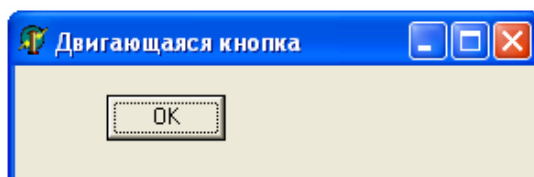


Рис. 21.

2. Для выхода из программы необходимо щелкнуть мышью на закрывающей кнопке в строке заголовка.

### Описание плана разработки программы

1. Открыть новый проект.
2. Разместить на форме экземпляры компонентов: кнопку **Button**, таймер **Timer**. Кнопка включает и выключает таймер, а таймер двигает кнопку.
3. Выполнить следующие действия:

Таблица 5.

Выделенный объект	Вкладка окна Object Inspector	Имя свойства/имя события	Действие
<b>Form1</b>	Properties	Caption	Установка имени формы «Двигающаяся кнопка»
<b>Timer1</b>	Properties	Enabled	Установить значение свойства Enabled = false Свойство Enabled определяет, включен или выключен таймер (по умолчанию, он включен).
		Interval	Interval = 100 Свойство Interval определяет интервал в миллисекундах между возникновением событий OnTimer (по умолчанию интервал равен 1 секунде).
	Events	OnTimer	Button1.Left := Button1.Left - 5; if Button1.Left < 10 then Button1.Left := 100;
<b>Button1</b>	Properties	Caption	Установка имени кнопки «Сменить заголовок окна»
		Default	Выбрать в раскрывающемся списке значение True
	Events	OnClick	Timer1.Enabled := <b>not</b> Timer1.Enabled;

4. Сохраните проект, запустите и протестируйте его.



## Практическая работа № 7 «Альбом»

**Цель работы** - создать программу, выполняющую следующие действия:

1. После запуска программы в окне изображается рисунок.



Рис. 22.

2. По щелчку мышью на рисунке появляется диалоговое окно.
3. Выбрать в открывшемся диалоговом окне любой другой рисунок.

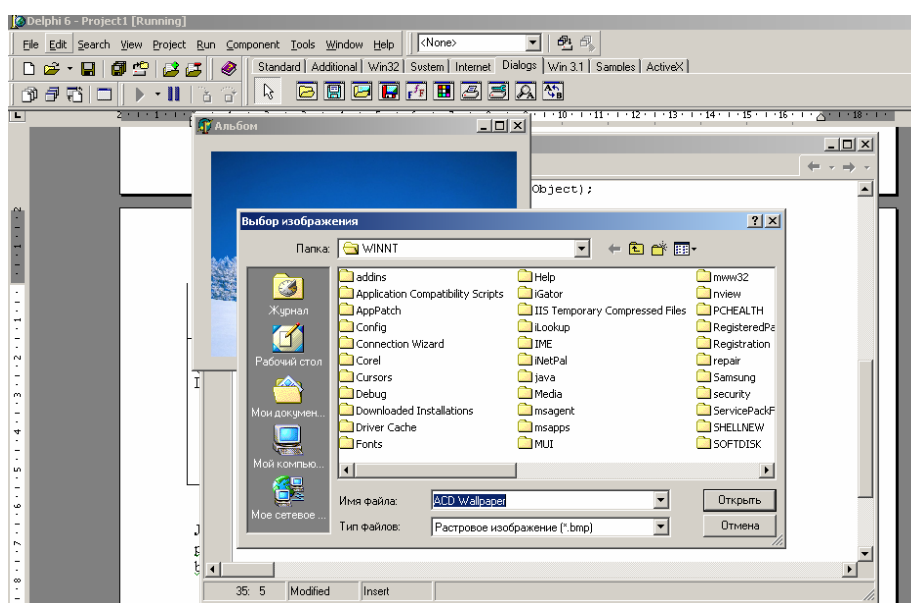


Рис. 23.

4. Для выхода из программы необходимо щелкнуть мышью на закрывающей кнопке в строке заголовка.

### Описание плана разработки программы

1. Открыть новый проект.
2. Разместить на форме экземпляры компонентов: панель **Panel**, рисунок **Image**, диалоговое окно **OpenDialog**.
3. Выполнить следующие действия:

Таблица 6.

Выделенный объект	Вкладка окна Object Inspector	Имя свойства/ имя события	Действие
<b>Form1</b>	Properties	Caption	Установка имени формы «Альбом»
<b>Panel1</b> (Вкладка Standard)	Properties	Caption	Очистите значение свойства Caption
		BevelOuter	Выбрать в раскрывающемся списке значение bvLowered
		BevelInner	Выбрать в раскрывающемся списке значение bvNone
		BewelWidth	Присвоить значение 2
		Width	Присвоить значение 241
		Height	Присвоить значение 185
<b>Image1</b> (Вкладка Additional)	Properties	Left	Присвоить значение 2
		Top	Присвоить значение 2
		Width	Присвоить значение 237
		Height	Присвоить значение 181
		Stretch	Включить свойство True
		Picture	С помощью кнопки-построителя открыть диалоговое окно Picture Editor (Редактор изображений). Щелкнуть на кнопке Load (Загрузить) – откроется диалоговое окно Load Picture (Загрузка рисунка). Открыть папку C:\Windows и выбрать файл Лес.bmp, щелкнуть на кнопке Открыть. Вернуться в окно Редактора изображений, щелкнуть на кнопке OK.
			Image1.Picture.LoadFromFile (OpenDialog1.FileName);
<b>OpenDialog1</b> (Вкладка Dialogs)	Properties	OnClick	OpenDialog1.Execute;
		Title	Ввести текст: «Выбор изображения»
		FileName	Ввести полный путь доступа к файлу: C:\Windows\Лес.bmp
		Filter	Ввести текст: Растровое изображение (*.bmp) *.bmp
		DefaultExt	Присвоить свойству значение: .bmp
		Options	Подсвойству ofFileMustExit (Файл должен существовать) присвоить значение True (Да)

4. Сохраните проект, запустите и протестируйте его.

#### Листинг подпрограммы

```

procedure TForm1.Image1Click (Sender: TObject);
begin
  OpenDialog1.Execute;
  Image1.Picture.LoadFromFile (OpenDialog1.FileName);
end;

```

## Практическая работа № 8

### Работа с меню

#### Цель работы -

Создать программу, выполняющую следующие действия:

1. После запуска программы в окне изображается строка меню (Файл, Выход).
2. При выборе пункта меню Файл появляются пункты меню (Рисунки, Выход).
3. При выборе пункта меню Рисунки появляется вложенное меню, состоящее из двух пунктов (Облака, Лес).

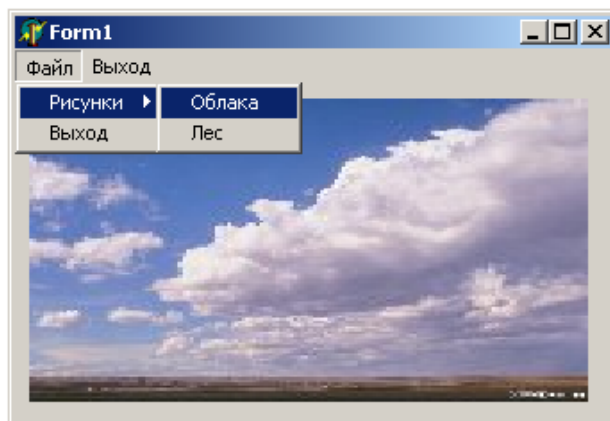


Рис. 24.

4. По щелчку правой кнопки мыши появляется контекстное меню.
5. Выбрать по пункту другой рисунок

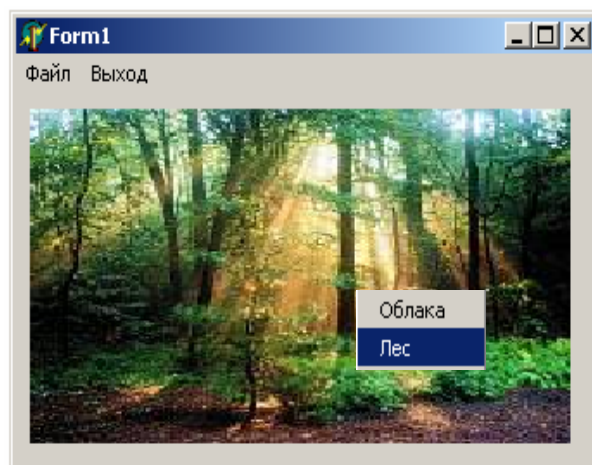


Рис. 25.

6. Для выхода из программы необходимо щелкнуть мышью на закрывающей кнопке в строке заголовка.
7. Если выбрать любой из пунктов Выход, работа программы завершается.

#### Описание плана разработки программы

1. Открыть новый проект.
2. Разместить на форме экземпляры компонентов: панель **Panel**, рисунок **Image**, диалоговое окно **OpenDialog**.
3. Выполнить следующие действия:

Таблица 7.

Выделенный объект	Вкладка окна Object Inspector	Имя свойства/ имя события	Действие
<b>Form1</b>	Properties	Caption	Установка имени формы «Мое меню»
	Events	OnMouseDown	<pre>var p:TPoint; begin   p.X :=X;   p.Y :=Y;   p := ClientToScreen (p);   PopupMenu1.Popup (p.X, p.Y); end;</pre>
Запустить редактор меню (дважды щелкнуть на значке меню на форме)			
<b>Form1.MainMenu1</b>	Properties (в окне Object Inspector не выбран никакой объект)	Caption	Ввести текст пункта меню – Файл, и нажать Enter. Система присвоит ему имя N1
Между существующими и будущими пунктами меню можно переключаться с помощью щелчка мыши или курсорных клавиш.			
<b>Form1.MainMenu1</b>	Properties	Caption	Ввести текст пункта меню – Выход, и нажать Enter. Система присвоит ему имя N2.
	Events (щелкнуть на пункте Выход в строке меню)	N2Click	Close;
Щелкните на пункте Файл. Редактор меню создал еще одну заготовку под этим пунктом. Это заготовка для меню, которое откроется при выборе пункта Файл в работающей программе. Используя заготовки, создайте в этом меню два пункта: Рисунки (система присвоит ему имя N3) и Выход (N4). Выберите в редакторе меню пункт Рисунки и нажмите комбинацию клавиш Ctrl + Вправо.			
<b>N4: TMenuItem</b>	Events	OnClick	Выберем из раскрывающегося списка уже существующую процедуру-обработчик N2Click
<b>Form1.MainMenu1</b>	Properties	Caption	Ввести текст пункта меню – Облака, и нажать Enter. Система присвоит ему имя N5.
<b>N5: TMenuItem</b>	Events (выбрать в строке меню на форме пункт Облака)	OnClick	Image1.Picture.LoadFromFile ('C:\Windows\Облака.bmp');
<b>Form1.MainMenu1</b>	Properties	Caption	Ввести текст пункта меню – Лес, и нажать Enter. Система присвоит ему имя N6.
<b>N6: TMenuItem</b>	Events (выбрать в строке меню на форме пункт Лес)	OnClick	Image1.Picture.LoadFromFile ('C:\Windows\Лес.bmp');
Закройте окно редактора меню и убедитесь, что теперь строка меню появилась в основной форме программы.			

Продолжение таблицы 7.

Выделенный объект	Вкладка окна Object Inspector	Имя свойства/ имя события	Действие
<b>PopupMenu</b> (Вкладка Standard)	Properties	Caption	Ввести текст пункта меню – Облака, и нажать Enter. Система присвоит ему имя N7.
		Caption	Ввести текст пункта меню – Лес, и нажать Enter. Система присвоит ему имя N8.
<b>N7</b>	Events	OnClick	Выберем из раскрывающегося списка уже существующую процедуру-обработчик N5Click
<b>N8</b>	Events	OnClick	Выберем из раскрывающегося списка уже существующую процедуру-обработчик N6Click
<b>Image</b> (Вкладка Additional)	Properties	Stretch	Присвоить значение True

4. Сохраните проект, запустите и протестируйте его.

#### Листинг подпрограммы

```

procedure TForm1.N2Click (Sender: TObject);
begin
    Close;
end;
procedure TForm1.N5Click (Sender: TObject);
begin
    Image1.Picture.LoadFromFile ('C:\Windows\Облака.bmp');
end;
procedure TForm1.N6Click (Sender: TObject);
begin
    Image1.Picture.LoadFromFile ('C:\Windows\Лес.bmp');
end;
procedure TForm1.FormMouseDown
    (Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
var p:TPoint;
begin
    p.X :=X;
    p.Y :=Y;
    p := ClientToScreen (p);
    PopupMenu1.Popup (p.X, p.Y);
end;

```

## Практическая работа № 9 Случайный выбор из списка

**Цель работы** - создать программу, выполняющую следующие действия:

1. После запуска программы в окне изображается три поля.

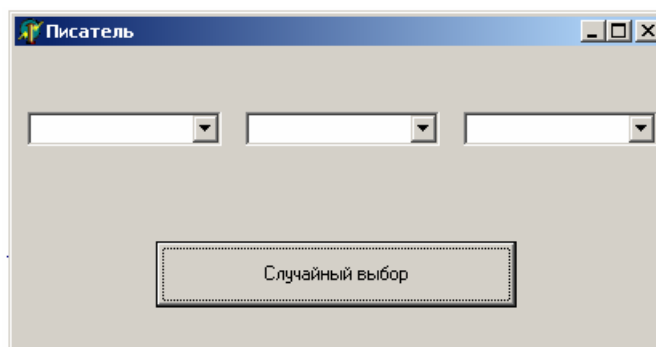


Рис. 26.

2. По щелчку мышью на кнопке «Случайный выбор» из трех слов составляется предложение случайным образом.

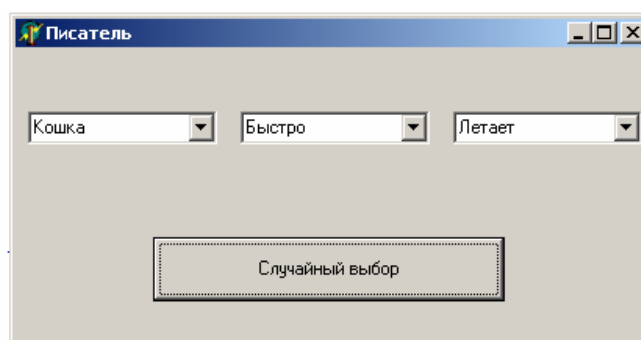


Рис. 27.

3. Для выхода из программы необходимо щелкнуть мышью на закрывающей кнопке в строке заголовка.

### Описание плана разработки программы

1. Открыть новый проект.
2. Разместить на форме экземпляры компонентов: поле со списком **ComboBox**, командная кнопка **Button**.
3. Выполнить следующие действия:

Таблица 8.

Выделенный объект	Вкладка окна Object Inspector	Имя свойства/ имя события	Действие
<b>Form1</b>	Properties	Caption	Установка имени формы «Сочинитель»
	Events	OnCreate	ComboBox1.ItemIndex :=0; ComboBox2.ItemIndex :=0; ComboBox3.ItemIndex :=0;
<b>ComboBox1</b> (Вкладка Standard)	Properties	Style	Выберите значение cSDropDownList из раскрывающегося списка
		Items	Щелкните на кнопке построителя. Откроется окно String List Editor (Редактор списка строк). Ввести пункты списка по одному в каждую строчку, завершая ввод нажатием клавиши Enter. После того как список готов, щелкнуть на кнопке ОК.

Продолжение таблицы 8.

Выделенный объект	Вкладка окна Object Inspector	Имя свойства/ имя события	Действие
<b>ComboBox2</b> (Вкладка Standard)	Properties	Style	Выберите значение cSDropDownList из раскрывающегося списка
		Items	Щелкните на кнопке строителя. Откроется окно String List Editor (Редактор списка строк). Ввести пункты списка по одному в каждую строчку, завершая ввод нажатием клавиши Enter. После того как список готов, щелкнуть на кнопке ОК.
<b>ComboBox3</b> (Вкладка Standard)	Properties	Style	Выберите значение cSDropDownList из раскрывающегося списка
		Items	Щелкните на кнопке строителя. Откроется окно String List Editor (Редактор списка строк). Ввести пункты списка по одному в каждую строчку, завершая ввод нажатием клавиши Enter. После того как список готов, щелкнуть на кнопке ОК.
<b>Button1</b> (Вкладка Standard)	Properties	Caption	Установка имени кнопки «Случайный выбор»
	Events	OnClick	ComboBox1.ItemIndex := Random(ComboBox1.ItemIndex.Count); ComboBox2.ItemIndex := Random(ComboBox2.ItemIndex.Count); ComboBox3.ItemIndex := Random(ComboBox3.ItemIndex.Count);

4. Сохраните проект, запустите и протестируйте его.

Таблица 9.

## Список существительных

Кошка  
Змея  
Кузнечик  
Дельфин  
Черепаша  
Ласточка

## Список наречий

Быстро  
Высоко  
Медленно  
Сильно  
Хорошо  
Плохо

## Список действий

Плавает  
Бегаёт  
Летает  
Ползает  
Прыгает  
Прячется

## Листинг подпрограммы

```
procedure TForm1.FormCreate (Sender: TObject);
```

```
begin
```

```
    Randomize;
```

```
    ComboBox1.ItemIndex :=0;
```

```
    ComboBox2.ItemIndex :=0;
```

```
    ComboBox3.ItemIndex :=0;
```

```
end;
```

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

```
begin
```

```
    ComboBox1.ItemIndex := Random(ComboBox1.ItemIndex.Count);
```

```
    ComboBox2.ItemIndex := Random(ComboBox2.ItemIndex.Count);
```

```
    ComboBox3.ItemIndex := Random(ComboBox3.ItemIndex.Count);
```

```
end;
```

## Практическая работа № 10

### Простейший плеер

**Цель работы** - создать программу, выполняющую следующие действия:

1. После запуска программы в окне изображается музыкальный проигрыватель.
2. По щелчку мышью на кнопке «Play» воспроизвести выбранную мелодию.
3. Для выхода из программы необходимо щелкнуть мышью на закрывающей кнопке в строке заголовка.

**Описание плана разработки программы**

1. Открыть новый проект.
2. Разместить на форме экземпляры компонентов: медиаплеер **MediaPlayer**, рисунок **Image**.

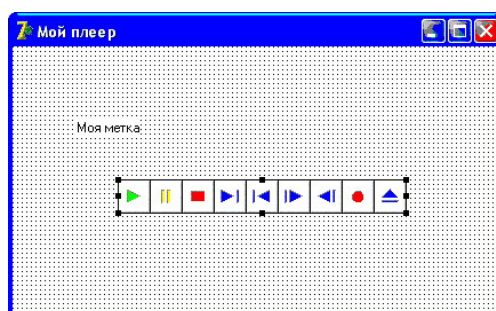


Рис. 28.

3. Выполнить следующие действия:

Таблица 10.

Выделенный объект	Вкладка окна Object Inspector	Имя свойства/имя события	Действие
<b>Form1</b>	Properties	Caption	Установка имени формы «Музыкальный проигрыватель»
<b>MediaPlayer1</b> (Вкладка System)	Properties	AutoOpen	Выберите значение True из раскрывающегося списка
		FileName	Указать не полный путь к файлу, а относительно местоположения программы. Например, “Prim.wav”, то программа будет этот файл искать в папке, где сама находится.

4. Сохраните проект, запустите и протестируйте его.
5. Создать проигрывателю картинку-фон, например:



Рис. 29.

6. Сохранить картинку-фон в файле, например, Фон.bmp
7. Вставить в плеер в качестве фона.



Продолжение таблица 10.

Выделенный объект	Вкладка окна Object Inspector	Имя свойства/ имя события	Действие
<b>Image1</b>	Properties	Picture	С помощью кнопки Load выберите ваш файл Фон.bmp и нажмите ОК. В результате на месте пунктирной каемки на форме появится этот рисунок из файла.
		AutoSize	Выбрать значение True

8. Подобрать размер формы под размер изображения, само изображение установите в верхний левый угол, панель с кнопками медиаплеера поставить так, чтобы гармонировала с рисунком фона, например:



Рис. 30.

## Практическая работа № 11 «Прыгающая кнопка»

**Цель работы** - создать программу-игру, выполняющую следующие действия:

1. После запуска программы в окне изображается беспорядочно прыгающая кнопка.
2. Необходимо успеть щелкнуть по ней.
3. Кнопка перепрыгивает из одного места в другое по сигналу, полученному от таймера.
4. Для выхода из программы необходимо щелкнуть мышью на закрывающей кнопке в строке заголовка.

**Описание плана разработки программы**

1. Открыть новый проект.
2. Разместить на форме экземпляры компонентов: командная кнопка **Button**, таймер **Timer**.

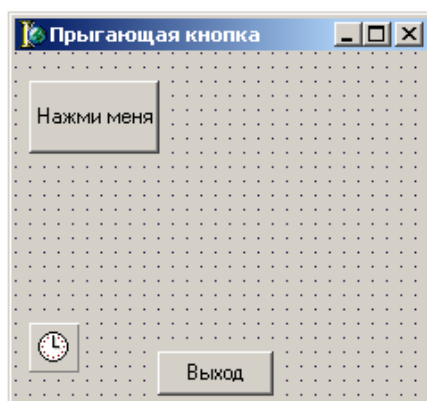


Рис. 31.

3. Выполнить следующие действия:

Таблица 11.

Выделенный объект	Вкладка окна Object Inspector	Имя свойства/ имя события	Действие
<b>Form1</b>	Properties	Caption	Установка имени формы «Прыгающая кнопка»
		ClientWidth (Внутренняя ширина)	Присвоить значение 300
		ClientHeight (Внутренняя высота)	Присвоить значение 200
		BorderStyle (тип границы)	Выбрать значение bsSingle (тонкая)
	Events	OnCreate	Randomize;
<b>Button1</b> (Вкладка Standard)	Properties	Caption	Ввести надпись «Нажми меня»
		TabStop	Присвоить значение False. Это свойство разрешает выбрать данный элемент управления клавишей Tab. Клавиатурой пользоваться запрещается.
		Visible	Присвоить значение False. Сначала кнопка невидима.
		Height	Присвоить значение 30
		Width	Присвоить значение 80
	Events	OnClick	Button1.Caption := 'Готово'; Button1.Enabled := False; Timer1.Enabled := False;

Выделенный объект	Вкладка окна Object Inspector	Имя свойства/ имя события	Действие
<b>Button2</b> (Вкладка Standard)	Properties	Caption	Ввести надпись «Выход»
		Default (по умолчанию)	Выбрать значение True
		Left (слева)	Присвоить значение 110
		Top (сверху)	Присвоить значение 160
		Width (ширина)	Присвоить значение 80
		Height (высота)	Присвоить значение 30
	Events	OnClick	Close;
<b>Timer1</b> (Вкладка System)	Properties	Interval (интервал)	Присвоить значение 500 (промежуток времени измеряется в миллисекундах)
	Events	Timer	var i: Integer; begin i:=Random(9); Button1.Visible := True; Button1.Top := 10 + 50 * ( i div 3); Button1.Left := 10 + 100 * ( i mod 3); end;

4. Сохраните проект, запустите и протестируйте его.

#### Листинг подпрограммы

```

procedure TForm1.Button2Click (Sender: TObject);
begin
  Close;
end;

procedure TForm1.Timer1Timer (Sender: TObject);
var i: Integer;
begin
  i:=Random(9);
  Button1.Visible := True;
  Button1.Top := 10 + 50 * ( i div 3);
  Button1.Left := 10 + 100 * ( i mod 3);
end;

procedure TForm1.Button1Click (Sender: TObject);
begin
  Button1.Caption := 'Готово';
  Button1.Enabled := False;
  Timer1.Enabled := False;
end;

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

```

#### Задание для самостоятельного выполнения

1. Измените игру так, чтобы скорость можно было настраивать в процессе игры.
2. Создайте две кнопки: Медленнее и Быстрее. Щелчок на одной из них будет увеличивать или уменьшать значение свойства Timer1.Interval на 100 миллисекунд.

## Практическая работа № 12 «Таблица умножения»

**Цель работы** - создать программу, выполняющую следующие действия:

1. После запуска программы в окне изображается два движка.
2. Необходимо выбрать два числовых значения и найти их произведение.
3. Если выбирается одно число, то находится его квадрат.

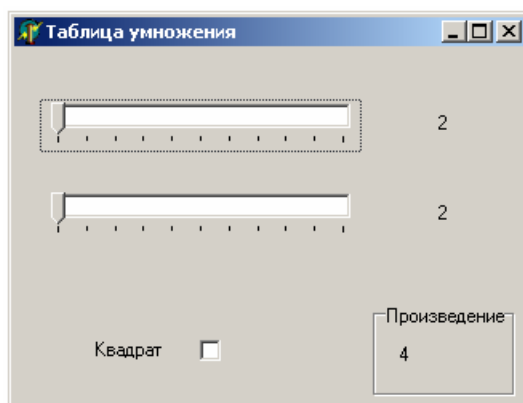


Рис. 32.

4. Для выхода из программы необходимо щелкнуть мышью на закрывающей кнопке в строке заголовка.

### Описание плана разработки программы

1. Открыть новый проект.
2. Разместить на форме экземпляры компонентов: командная кнопка **Button**, движок **TrackBar**, рамка **GroupBox**, надпись **Label**, флажок **CheckBox**.
3. Выполнить следующие действия:

Таблица 12.

Выделенный объект	Вкладка окна Object Inspector	Имя свойства/ имя события	Действие
<b>Form1</b>	Properties	Caption	Установка имени формы «Таблица умножения»
<b>TrackBar1</b> (Вкладка Win32)	Properties	Min (Минимум)	Присвоить значение 2
		Max (Максимум)	Присвоить значение 99
		Position (Положение)	Присвоить значение 2
		LineSize (Малое изменение)	Присвоить значение 1
		PageSize (Постраничное изменение)	Присвоить значение 7
		Frequency (Частота засечек)	Присвоить значение 7
	Events	OnChange	Label1.Caption := IntToStr(TrackBar1.Position); Label3.Caption := IntToStr(TrackBar1.Position * TrackBar2.Position); if CheckBox1.Checked then TrackBar2.Position :=TrackBar1.Position;

Продолжение таблицы 12.

Выделенный объект	Вкладка окна Object Inspector	Имя свойства/ имя события	Действие
<b>TrackBar2</b> (Вкладка Win32)	Properties	Min (Минимум)	Присвоить значение 2
		Max (Максимум)	Присвоить значение 99
		Position (Положение)	Присвоить значение 2
		LineSize (Малое изменение)	Присвоить значение 1
		PageSize (Постраничное изменение)	Присвоить значение 7
		Frequency (Частота засечек)	Присвоить значение 7
	Events	OnChange	Label2.Caption := IntToStr(TrackBar2.Position); Label3.Caption := IntToStr(TrackBar1.Position * TrackBar2.Position); if CheckBox1.Checked then TrackBar1.Position := TrackBar2.Position;
<b>GroupBox1</b> (Вкладка Standard)	Properties	Caption	Ввести подпись «Произведение»
<b>Label1</b> (Вкладка Standard)	Properties	AutoSize (Автоподбор)	Установить значение False
		Caption	Присвоить значение 2
		Alignment (Выравнивание)	Установить значение taRightJustify (Выравнивание по правому краю)
<b>Label2</b> (Вкладка Standard)	Properties	AutoSize	Установить значение False
		Caption	Присвоить значение 2
		Alignment	Установить значение taRightJustify (Выравнивание по правому краю)
<b>Label3</b> (Вкладка Standard)	Properties	AutoSize	Установить значение False
		Caption	Присвоить значение 4
		Alignment	Установить значение taRightJustify (Выравнивание по правому краю)
<b>CheckBox1</b> (Вкладка Standard)	Properties	Caption	Ввести подпись «Квадрат»
		Alignment	Установить значение taLeftJustify (Выравнивание по левому краю)
	Events	OnClick	TrackBar2.Position := TrackBar1.Position;

4. Сохраните проект, запустите и протестируйте его.

#### Листинг подпрограммы

**procedure** TForm1.TrackBar1Change (Sender: TObject);

**begin**

  Label1.Caption := IntToStr(TrackBar1.Position);

  Label3.Caption := IntToStr(TrackBar1.Position \* TrackBar2.Position);

  if CheckBox1.Checked then TrackBar2.Position := TrackBar1.Position;

```

end;
procedure TForm1.TrackBar2Change (Sender: TObject);
begin
    Label2.Caption := IntToStr(TrackBar2.Position);
    Label3.Caption := IntToStr(TrackBar1.Position * TrackBar2.Position);
    if CheckBox1.Checked then TrackBar2.Position := TrackBar1.Position;
end;
procedure TForm1.CheckBox1Click (Sender: TObject);
begin
    TrackBar2.Position := TrackBar1.Position;
end;

```

#### **Задание для самостоятельного выполнения**

1. Изменить программу так, чтобы находить произведения не только двузначных, но и трехзначных чисел от 2 до 199.
2. Изменить программу так, чтобы находить сумму двух чисел.

## Практическая работа № 13

### Применение полос прокрутки

**Цель работы** - создать программу, выполняющую следующие действия:

1. После запуска программы в окне изображается две полосы прокрутки. Вертикальная полоса будет управлять движением по вертикали, горизонтальная – по горизонтали.
2. Наводя указатель мыши на одну из двух фигур, можно выбирать, какая из этих фигур связана с полосами прокрутки.



Рис. 33.

3. Требуются дополнительные объекты, с помощью которых ограничивается область движения фигур в окне.
4. Если полоса прокрутки активная, то она должна реагировать на клавиши ВВЕРХ, ВНИЗ, ВЛЕВО, ВПРАВО, PAGE UP, PAGE DOWN.
5. Для выхода из программы необходимо щелкнуть мышью на закрывающей кнопке в строке заголовка.

#### Описание плана разработки программы

1. Открыть новый проект.
2. Разместить на форме экземпляры компонентов: панель **Panel**, полоса прокрутки **ScrollBar**, фигура **Shape**.
3. Ввести дополнительную переменную логического типа `num`. Если она принимает значение `True` (Да), то текущей считается первая фигура. Значению `False` (Нет) соответствует вторая фигура. Эта переменная должна быть доступна во всех процедурах.
4. Выполнить следующие действия:

Таблица 13.

Выделенный объект	Вкладка окна Object Inspector	Имя свойства/ имя события	Действие
<b>Form1</b>	Properties	Caption	Установка имени формы «Перемещение фигур»
<b>Panel</b> (Вкладка Standard)	Properties	Height	Присвоить значение 161
		Width	Присвоить значение 161
		Caption	Оставить значение свойства пустым
<b>ScrollBar1</b> (Вкладка Standard)	Properties	Min	Присвоить значение 5
		Max	Присвоить значение 145
		Position	Присвоить значение 76
		SmallChange	Присвоить значение 2

Продолжение таблицы 13.

Выделенный объект	Вкладка окна Object Inspector	Имя свойства/ имя события	Действие
		LargeChange (Большое изменение)	Присвоить значение 20
	Events	OnChange	if num then Shape1.Left := ScrollBar1.Position else Shape2.Left := ScrollBar1.Position
<b>ScrollBar2</b> (Вкладка Standard)	Properties	Kind	Выбрать значение sbVertical. Горизонтальная полоса прокрутки станет вертикальной.
		Min	Присвоить значение 5
		Max	Присвоить значение 145
		Position	Присвоить значение 76
		SmallChange (Малое изменение)	Присвоить значение 2
		LargeChange	Присвоить значение 20
	Events	OnChange	if num then Shape1.Top := ScrollBar2.Position else Shape2.Top := ScrollBar2.Position
<b>Shape1</b> (Вкладка Additional)	Properties	Height	Присвоить значение 11
		Width	Присвоить значение 11
		Left	Присвоить значение 76
		Top	Присвоить значение 76
		Shape (Форма)	Выбрать значение stCircle (Круг)
		Brush (Кисть)	Выбрать для подсвойства Color (Цвет кисти) значение clAqua (голубой цвет)
	Events	OnMouseMove (При движении мыши)	Shape1.Brush.Color := clAqua; Shape1.Brush.Color := clFuchsia; Num := True; ScrollBar1.Position:= Shape1.Left; ScrollBar2.Position:= Shape1.Top;
<b>Shape2</b> (Вкладка Additional)	Properties	Height	Присвоить значение 11
		Width	Присвоить значение 11
		Left	Присвоить значение 76
		Top	Присвоить значение 76
		Shape	Выбрать значение stSquare (Квадрат)
		Brush	Выбрать для подсвойства Color (Цвет кисти) значение clFuchsia (фиолетовый цвет)
	Events	OnMouseMove	Аналогично Shape2



5. Сохраните проект, запустите и протестируйте его.

**Листинг подпрограммы**

```
procedure TForm1.ScrollBar1Change (Sender: TObject);  
begin  
    if num then Shape1.Left := ScrollBar1.Position  
        else Shape2.Left := ScrollBar1.Position  
end;  
procedure TForm1.ScrollBar2Change (Sender: TObject);  
begin  
    if num then Shape1.Top := ScrollBar2.Position  
        else Shape2.Top := ScrollBar2.Position  
end;  
procedure TForm1.Shape1MouseMove  
    (Sender: TObject; Shift: TShiftState; X, Y: Integer);  
begin  
    Shape1.Brush.Color := clAgua;  
    Shape1.Brush.Color := clFuchsia;  
    Num := True;  
    ScrollBar1.Position:= Shape1.Left;  
    ScrollBar2.Position:= Shape1.Top;  
end;  
procedure TForm1. Shape2MouseMove  
    (Sender: TObject; Shift: TShiftState; X, Y: Integer);  
begin  
    Shape2.Brush.Color := clFuchsia;  
    Shape2.Brush.Color := clAgua;  
    Num := False;  
    ScrollBar1.Position:= Shape2.Left;  
    ScrollBar2.Position:= Shape2.Top;  
end;  
procedure TForm1. FormCreate (Sender: TObject);  
begin  
    num := True;  
end;
```

## Практическая работа № 14 «Светофор»

**Цель работы** - создать программу, выполняющую следующие действия:

1. После запуска программы в окне изображается светофор с тремя лампочками, способными реагировать на наведение указателя мыши.
2. Когда указатель мыши наведен на лампочку, она меняет свой цвет.

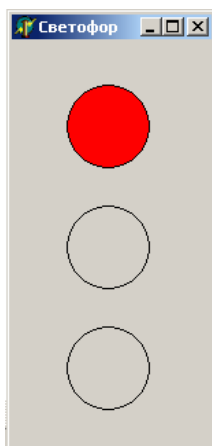


Рис. 34.

3. Для выхода из программы необходимо щелкнуть мышью на закрывающей кнопке в строке заголовка.

### Описание плана разработки программы

1. Открыть новый проект.
2. Разместить на форме экземпляры компонентов: фигура **Shape**.
3. Выполнить следующие действия:

Таблица 14.

Выделенный объект	Вкладка окна Object Inspector	Имя свойства/ имя события	Действие
<b>Form1</b>	Properties	Caption	Установка имени формы «Светофор»
		Height	Присвоить значение 300
		Width	Присвоить значение 120
		BorderIcons (Служебные кнопки)	Выбрать для подсвойства biMinimize (Сворачивание) и biMaximize (Разворачивание) значение False
		Color	Задать стандартный серый цвет
		BorderStyle (Стиль рамки)	Выбрать значение bsSingle
	Events	OnMouseMove	В процедуру передаются дополнительные параметры: Shift – указывает, не была ли при перемещении нажата клавиша SHIFT, CTRL или ALT; X - горизонтальная координата указателя мыши; Y - вертикальная координата указателя.

Продолжение таблицы 14.

Выделенный объект	Вкладка окна Object Inspector	Имя свойства/ имя события	Действие
<b>Shape1</b> (Вкладка Additional)	Properties	Height	Присвоить значение 61
		Width	Присвоить значение 61
		Shape (Форма)	Выбрать значение stCircle (Круг)
		Pen (Контур)	Выбрать для подсвойства Color (Цвет) значение clRed (красный цвет)
		Brush (Кисть)	Выбрать для подсвойства Style (стиль) значение bsClear (прозрачный)
		Enabled (Включен)	Выбрать значение False (Нет)
<b>Shape2</b> (Вкладка Additional)	Properties	Height	Присвоить значение 61
		Width	Присвоить значение 61
		Shape	Выбрать значение stCircle (Круг)
		Pen	Выбрать для подсвойства Color (Цвет) значение clYellow (желтый цвет)
		Brush	Выбрать для подсвойства Style значение bsClear
		Enabled (Включен)	Выбрать значение False (Нет)
<b>Shape3</b> (Вкладка Additional)	Properties	Height	Присвоить значение 61
		Width	Присвоить значение 61
		Shape	Выбрать значение stCircle (Круг)
		Pen	Выбрать для подсвойства Color (Цвет) значение clLime (ярко-зеленый цвет)
		Brush	Выбрать для подсвойства Style значение bsClear
		Enabled (Включен)	Выбрать значение False (Нет)

4. Написать функцию OnShape, которая вызывается из процедуры FormMouseMove.
5. Работа функции определяет следующие события:  
 Если указатель мыши не наведен на лампочку, то ее цвет будет прозрачным.  
 Если указатель мыши наведен на лампочку, то ее цвет будет соответствовать цветам светофора.  
 $r := sh.Width \div 2;$  {r - радиус фигуры}  
 $cx := sh.Left + r;$  {cx, cy – координаты центра фигуры}  
 $cy := sh.Top + r;$   
 $d2 := (X - cx) * (X - cx) + (Y - cy) * (Y - cy);$  {d2 – квадрат расстояния от центра}  
 OnShape := bsClear; {указатель мыши не наведен на лампочку, т.е. находится за пределами фигуры}  
**if** d2 > r\*r **then** ObShape := bsSolid; {указатель мыши наведен на лампочку, т.е. находится внутри фигуры}
6. Выравнивание элементов:
  - 1) Выделить все три фигуры.
  - 2) Выбрать команду Edit → Align (Правка → Выровнить).
  - 3) Откроется диалоговое окно Alignment (Выравнивание).
  - 4) Установить слева переключатель Center in Window (Центрировать в окне).

- 5) Установить справа переключатель Space Equally (С равными промежутками).
- 6) Щелкнуть на кнопке ОК.
7. Сохраните проект, запустите и протестируйте его.

#### Листинг подпрограммы

```
Function OnShape (sh: TShape; X, Y: Integer): TBrushStyle;  
  var r, cx, cy, d2: Integer;  
  begin  
    r := sh.Width div 2;  
    cx := sh.Left + r;  
    cy := sh.Top + r;  
    d2 := (X - cx) * (X - cx) + (Y - cy) * (Y - cy);  
    OnShape := bsClear;  
    if d2 > r*r then ObShape := bsSolid;  
  end;  
procedure TForm1.FormMouseMove (Sender: TObject, Shift: TShiftState; X, Y: Integer);  
begin  
  Shape1.Brush.Color := clRed;  
  Shape1.Brush.Style := OnShape (Shape1, X, Y);  
  Shape2.Brush.Color := clYellow;  
  Shape2.Brush.Style := OnShape (Shape2, X, Y);  
  Shape3.Brush.Color := clLime;  
  Shape.Brush.Style := OnShape (Shape3, X, Y);  
end;
```

## Практическая работа № 15 «Ханойские башни»

**Цель работы** - создать компьютерную версию игры-головоломки «Ханойские башни». Игровая доска содержит семь клеток, расположенных в ряд. На трех левых клетках стоят белые фишки, на трех правых – черные. Центральная клетка пуста. Задача заключается в том, чтобы, делая допустимые ходы, поменять фишки местами. Правила ходов такие.

1. Белые фишки могут ходить только вправо, а черные – только влево.
2. Ходить разрешается только на свободное поле.
3. Разрешены перемещения на соседнее поле и «прыжок» через одну фишку.

В роли фишек выступают кнопки с изображением, например, цветных треугольников, направленных так, чтобы они указывали допустимое направление ходов. Кнопки будут передвигаться по игровой доске, созданной при помощи панели.



Рис. 35.

### Описание плана разработки программы

1. Открыть новый проект.
2. Разместить на форме экземпляры компонентов: панель **Panel**, кнопка с изображением **BitBtn**.
3. Выполнить следующие действия:

Таблица 15.

Выделенный объект	Вкладка окна Object Inspector	Имя свойства/ имя события	Действие
<b>Form1</b>	Properties	Caption	Установка имени формы «Головоломка»
<b>Panel1</b> (Вкладка Standard)	Properties	Height	Присвоить значение 65
		Width	Присвоить значение 401
		Caption	Очистить значение
<b>BitBtn1</b> (Вкладка Additional)	Properties	Height	Присвоить значение 49
		Width	Присвоить значение 49
		Left	Присвоить значение 8
		Top	Присвоить значение 8
<b>BitBtn2</b>	Properties	Height	Присвоить значение 49
		Width	Присвоить значение 49
		Left	Присвоить значение 64
		Top	Присвоить значение 8
<b>BitBtn3</b>	Properties	Height	Присвоить значение 49
		Width	Присвоить значение 49
		Left	Присвоить значение 120
		Top	Присвоить значение 8
<b>BitBtn4</b>	Properties	Height	Присвоить значение 49
		Width	Присвоить значение 49
		Left	Присвоить значение 232
		Top	Присвоить значение 8
<b>BitBtn5</b>	Properties	Height	Присвоить значение 49
		Width	Присвоить значение 49
		Left	Присвоить значение 288
		Top	Присвоить значение 8

Продолжение таблицы 15.

Выделенный объект	Вкладка окна Object Inspector	Имя свойства/ имя события	Действие
<b>BitBtn6</b>	Properties	Height	Присвоить значение 49
		Width	Присвоить значение 49
		Left	Присвоить значение 344
		Top	Присвоить значение 8

4. Сохраните проект.
5. Создать рисунки при помощи стандартной программы Paint. Задайте размеры рисунков 45х45 пикселей (в режиме увеличения).
6. Создать две картинки с именами Left.bmp и Right.bmp, поместить их в папку проекта. Рисунок с именем Left.bmp предназначен для левых кнопок, рисунок с именем Right.bmp – для правых кнопок.
7. Назначить рисунки кнопкам.

Продолжение таблицы 15.

<b>BitBtn1, BitBtn2, BitBtn3, BitBtn4, BitBtn5, BitBtn6</b>	Properties	Caption	Очистить значение
		Glyph (Значок)	Щелкнуть на кнопке строителя. Открыть окно Picture Editor. Щелкните на кнопке Load. В диалоговом окне выберите файл с изображением кнопки и щелкните на кнопке Открыть. Щелкнуть на кнопке Ok, и выбранная картинка появится на кнопке.

8. Необходимо хранить два числа: номер клетки, в которой находится кнопка; признак «цвета» фишки», т.е. в какую сторону может ходить кнопка.
9. Номер клетки умножить на два. Если фишка черная, прибавить к числу единицу.
10. Задать свойство Tag (Тег) для всех кнопок.

Таблица 16.

Значение свойства Tag	Комментарий
2	Первая клетка, белая фишка
4	Вторая клетка, белая фишка
6	Третья клетка, белая фишка
11	Четвертая клетка, черная фишка
13	Пятая клетка, черная фишка
15	Шестая клетка, черная фишка

11. Создать дополнительную переменную для пустой клетки  
var n: Integer = 4;
12. Создать глобальную переменную  
win : Integer = 24; {все фишки в сумме должны переместиться на 24 клетки}
13. Программируем выполнение хода фишки.

Таблица 17.

Выделенный объект	Вкладка окна Object Inspector	Вмя события	Действие
<b>BitBtn1</b>	Events	OnClick	В процедуру BitBtn1Click передается параметр – объект Sender. Он представляет именно тот объект, который вызвал событие, и может дать доступ к его свойствам. Надо специально указать, что объект Sender в данном случае следует считать кнопкой. Чтобы обратиться к свойству Tag данного объекта, записывают (Sender as TBitBtn).Tag.

14. Сохраните проект, запустите и протестируйте его.

### Листинг подпрограммы

```

procedure TForm1.BitBtn1Click (Sender: TObject);
var i, c, k, ak: Integer;
begin
  with Sender as TBitBtn do {объект Sender используется многократно}
  begin {свойство Tag указывает, какая именно кнопка нажата}
    i := Tag div 2; {переменная i хранит номер клетки, где находится кнопка}
    c := Tag mod 2; {переменная c определяет цвет фишки}
    k := n - i; {k – величина перемещения}
    ak := Abs (k); {ak – длина хода}
    if ak < 3 then {проверка допустимости хода}
      if ((c = 0) and (k > 0)) or ((c = 1) and (k < 0)) {белые фишки могут ходить только
        вправо, черные - влево}
      then {делаем ход}
        begin
          Tag := Tag + 2 * k;
          Left := Left + 56 * k;
          win := win - ak; {при каждом ходе уменьшается значение
            переменной win на длину хода}
          n := i; {пустая клетка находится там, где раньше была фишка}
        end;
      end;
    end;
    if win = 0 then {проверка завершения решения}
      begin
        Caption := 'Победа!';
        Panel1.Color := clFuchsia;
        Panel1.Enabled := False; {отключаем панель, и одновременно все расположенные на
          ней объекты}
      end;
    end;
  end;

```

## Практическая работа № 16 «Электронный альбом»

**Цель работы** - создать программу, выполняющую следующие действия:

1. После запуска программы в окне изображается рисунок, выбранный первым переключателем из раскрывающегося списка.
2. Выбираем необходимый переключатель, рисунок из соответствующего раскрывающегося списка.
3. Для выхода из программы необходимо щелкнуть мышью на закрывающей кнопке в строке заголовка.

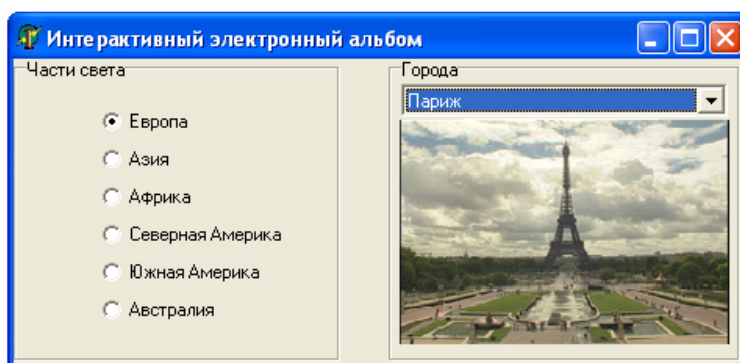


Рис. 36.

### Описание плана разработки программы

1. Открыть новый проект.
2. Разместить на форме экземпляры компонентов: рамка **GroupBox**, переключатель **RadioButton**, раскрывающийся список **ComboBox**.
3. Выполнить следующие действия:

Таблица 18.

Выделенный объект	Вкладка окна Object Inspector	Имя свойства/имя события	Действие
<b>Form1</b>	Properties	Caption	Установка имени формы «Интерактивный электронный альбом»
	Events	OnCreate	RadioButton1Click(RadioButton1);
<b>GroupBox1</b> (Вкладка Standard)	Properties	Caption	Введите название «Части света»
<b>RadioButton1</b> (Вкладка Standard)	Properties	Caption	Введите название «Европа»
		Checked (Включен)	Задайте значение True
		Tag	Присвоить значение 0
	Events	OnClick	В процедуре описать локальную переменную i. Описать действия переключателей, используя цикл с параметром i от 0 до 5.

Для создания еще пяти переключателей можно использовать метод копирования через буфер обмена. Пока объект **RadioButton1** остается выделенным, нажмите комбинацию клавиш **CTRL+C**. Произойдет его копирование в буфер обмена. Теперь нажмите комбинацию **CTRL+V** – это команда вставки объекта из буфера обмена и Delphi автоматически изменит имя объекта – новый объект получит имя **RadioButton2**.



Продолжение таблицы 18.

Выделенный объект	Вкладка окна Object Inspector	Имя свойства/ имя события	Действие
<b>RadioBatton2</b>	Properties	Caption	Введите название «Азия»
		Tag	Присвоить значение 5
<b>RadioBatton3</b>	Properties	Caption	Введите название «Африка»
		Tag	Присвоить значение 10
<b>RadioBatton4</b>	Properties	Caption	Введите название «Северная Америка»
		Tag	Присвоить значение 15
<b>RadioBatton5</b>	Properties	Caption	Введите название «Южная Америка»
		Tag	Присвоить значение 20
<b>RadioBatton6</b>	Properties	Caption	Введите название «Австралия»
		Tag	Присвоить значение 25
Выделите все шесть переключателей. Дайте команду Edit → Align (Правка → Выравнивание). Откроется окно Alignment (выравнивание). Установить слева переключатель Left Sides (Левые края), а справа Space Equally (Равные промежутки). Щелкните на кнопке ОК.			
<b>GroupBox2</b> (Вкладка Standard)	Properties	Caption	Введите название «Города»
<b>ComboBox1</b> (Вкладка Standard)	Properties	Style	Выбрать значение csDropDownList (Раскрывающийся список)
	Events	OnChange	Image1.Picture.LoadFromFile (ListBox2.Items [ComBox1.Tag + ComboBox1.ItemIndex]);
<b>Image1</b> (Вкладка Additional)			Поместить объект на рамку
<b>ListBox1</b> (Вкладка Standard)	Properties	Visible	Задайте значение False
		Items (Пункты)	Щелкнуть на кнопке строителя. Введите в список тридцать названий городов - по пять для каждой части света.
<b>ListBox2</b> (Вкладка Standard)	Properties	Visible	Задайте значение False
		Items	Щелкнуть на кнопке строителя. Введите в список тридцать имен файлов, содержащих иллюстрации.

4. Сохраните проект, запустите и протестируйте его.

### Листинг подпрограммы

```

procedure TForm1. RadioButton1Click (Sender: TObject);
var i: Integer;
begin
    ComboBox1.Clear;
    ComboBox1.Tag := (Sender as TRadioButton).Tag;
    for i := 0 to 5 do
        ComboBox1.Items.Add (ListBox1.Items [ComboBox1.Tag + i]);
        ComboBox1.ItemIndex := 0;

```

```
        Image1.Picture.LoadFromFile (ListBox2.Items [ComboBox1.Tag]);  
end;  
  
procedure TForm1.FormCreate(Sender: TObject);  
begin  
    RadioButton1Click(RadioButton1);  
end;  
  
procedure TForm1.ComboBox1Change (Sender: TObject);  
begin  
    Image1.Picture.LoadFromFile (ListBox2.Items [ComboBox1.Tag + ComboBox1.ItemIndex]);  
end;
```

## Практическая работа № 17 «Вычисление процентов»

**Цель работы** - создать программу, выполняющую следующие действия:

1. После запуска программы в окне изображается три текстовых поля.

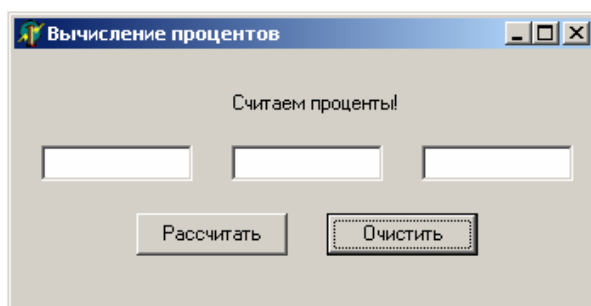


Рис. 37.

2. В первое поле вводится число. Во второе поле – проценты. При нажатии кнопки «Рассчитать» в третье поле выводятся вычисленные проценты от числа.

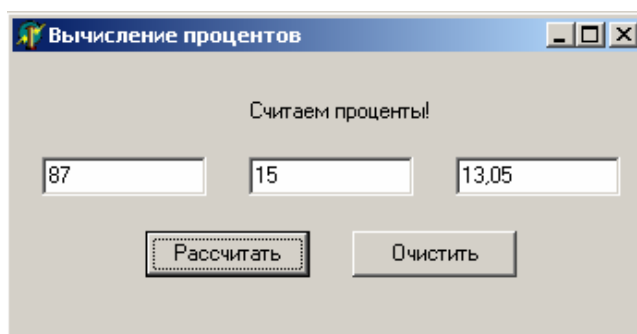


Рис. 38.

3. При нажатии кнопки «Очистить» очищаются значения полей. Далее вводятся новые значения в поля.
4. Для выхода из программы необходимо щелкнуть мышью на закрывающей кнопке в строке заголовка.

### Описание плана разработки программы

1. Открыть новый проект.
2. Разместить на форме экземпляры компонентов: командная кнопка **Button**, надпись **Label**, текстовое поле **Edit**.
3. Выполнить следующие действия:

Таблица 19.

Выделенный объект	Вкладка окна Object Inspector	Имя свойства/ имя события	Действие
<b>Form1</b>	Properties	Caption	Установка имени формы «Вычисление процентов»
<b>Label1</b>	Properties	Caption	Введите название «Считаем проценты!»
<b>Edit1</b>	Properties	Text	Очистить значение свойства
<b>Edit2</b>	Properties	Text	Очистить значение свойства
<b>Edit3</b>	Properties	Text	Очистить значение свойства
<b>Button1</b>	Properties	Caption	Введите название «Рассчитать»
	Events	OnClick	Описать локальные Number, Procent, Prn

Продолжение таблицы 19.

Выделенный объект	Вкладка окна Object Inspector	Имя свойства/ имя события	Действие
<b>Button2</b>	Properties	Caption	Введите название «Очистить»
	Events	OnClick	Edit1.Text:=""; Edit2.Text:=""; Edit3.Text:="";

4. Сохраните проект, запустите и протестируйте его.

#### Фрагмент программы

```

var
  Number, Procent, Prn: Real;
procedure TForm1.Button1Click(Sender: TObject);
begin
  Number:=StrToFloat(Edit1.Text);
  Procent:= StrToFloat(Edit2.Text);
  PrN:=0.01*Procent*Number;
  Edit3.Text:=FloatToStr(PrN);
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
  Edit1.Text:="";
  Edit2.Text:="";
  Edit3.Text:="";
end;

```

## Практическая работа № 18

### «Головоломка №1»

**Цель работы** - создать компьютерную версию одной из головоломок Самуэля Ллойда: из заданного набора чисел надо выбрать те, сумма которых составит 50. Числа, которые избрал Ллойд для своей головоломки: 25, 27, 3, 12, 6, 15, 9, 30, 21, 19.

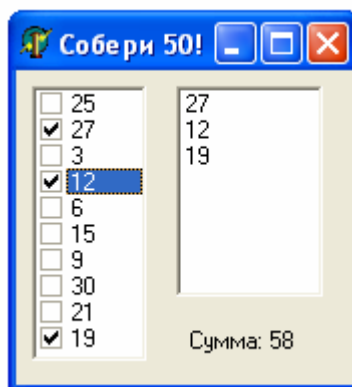


Рис. 39.

Выполнить следующие действия:

1. После запуска программы в окне изображается список чисел Ллойда.
2. Выбираем с помощью флажков числа и помещаем в правое окно.
3. Сумма выбранных чисел представлена в виде надписи.
4. Для выхода из программы необходимо щелкнуть мышью на закрывающей кнопке в строке заголовка.

#### Описание плана разработки программы

1. Открыть новый проект.
2. Разместить на форме экземпляры компонентов: список флажков **CheckBox**, надпись **Label**, список **ListBox**.
3. Выполнить следующие действия:

Таблица 20.

Выделенный объект	Вкладка окна Object Inspector	Имя свойства/ имя события	Действие
<b>Form1</b>	Properties	Caption	Установка имени формы «Головоломка»
		BorderStyle	Задайте значение bsSingle
<b>CheckBoxList1</b> (Вкладка Additional )	Properties	Items	Задаем состав списка. Щелкнуть на кнопке построителя. Откроется окно String List editor (Редактор списка строк). Введите в список заданные числа через Enter. Нажмите кнопку ОК.
		Height	Подобрать значение так, что все числа поместились в список (без полос прокруток).
	Events	OnClickCheck	Описать очистку списка. Проверить, установлен флажок или нет. После обновления списка необходимо подсчитать сумму выбранных чисел. Элементы списка выглядят как числа, но являются текстовыми строками (воспользоваться функцией StrToInt).

Продолжение таблицы 20.

Выделенный объект	Вкладка окна Object Inspector	Имя свойства/ имя события	Действие
<b>ListBox1</b> (Вкладка Standard)	Properties	Height	Подобрать значение так, что все числа поместились в список (без полос прокруток). Первоначально список пуст. Заполняться он будет при работе программы.
<b>Label1</b> (Вкладка Standard)	Properties	Caption	Введите текст: «Сумма: 0»
		Autosize	Задайте значение False
		Alignment	Задайте значение taCenter

4. Сохраните проект, запустите и протестируйте его.

#### Листинг подпрограммы

```

procedure TForm1.CheckListBox1ClickCheck (Sender: TObject);
var i, s: Integer;
begin
    ListBox1.Clear;
    for i := 0 to CheckListBox1.Items.Count-1 do
        if CheckListBox1.Checked[i] then
            ListBox1.Items.Add (CheckListBox1.Items[i]);
    s := 0;
    for i := 0 to ListBox1.Items.Count -1 do
        s := s + StrToInt(ListBox1.Items[i]);
    Label1.Caption := 'Сумма: ' + IntToStr(s);
    if s = 50 then
        begin
            Label1.Caption := 'Сумма:' + IntToStr (s);
            CheckListBox1.Enabled := False;
            ListBox1.Enabled := False;
        end;
end;

```

## Практическая работа № 19 «Головоломка № 2»

**Цель работы** - создать компьютерную версию головоломки: из изображенных пяти сброшенных флажков установить все. Но при выборе одного флажка меняется состояние двух следующих.

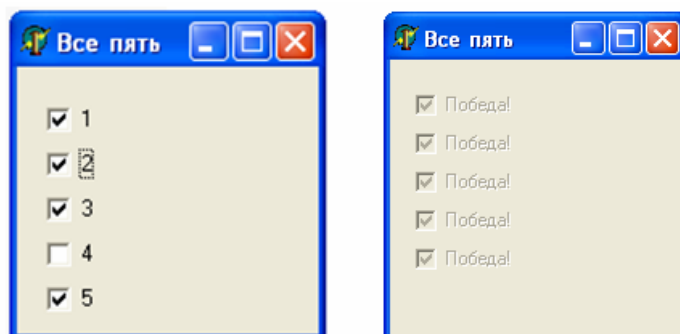


Рис. 40.

Создать программу, выполняющую следующие действия:

1. После запуска программы в окне изображаются пять сброшенных флажков.
2. Щелкать разрешено только на сброшенных флажках. Щелчок на установленном флажке не действует.
3. При установке какого-то флажка меняется состояние двух следующих флажков. При этом сброшенные флажки устанавливаются, а установленные - сбрасываются.
4. Для выхода из программы необходимо щелкнуть мышью на закрывающей кнопке в строке заголовка.

### Описание плана разработки программы

1. Открыть новый проект.
2. Разместить на форме экземпляры компонентов: список флажков **CheckBox**, надпись **Label**.
3. Выполнить следующие действия:

Таблица 21.

Выделенный объект	Вкладка окна Object Inspector	Имя свойства/имя события	Действие
<b>Form1</b>	Properties	Caption	Установка имени формы «Все пять»
	Events	OnCreate	Создать глобальную логическую переменную s. В то время, когда идет обработка события, переменная имеет значение True (да).
<b>CheckBox1</b> (Вкладка Standard)	Properties	Caption	Задать значение «1».
	Events	OnClick	Необходимо работать с флажками, как с массивом. Форма является контейнером для флажков. У объектов-контейнеров имеется свойство Controls (Элементы управления) – массив элементов управления, находящихся в данном контейнере: Controls [i] as TCheckBox.
Для создания еще пяти флажков можно использовать метод копирования через буфер обмена.			
<b>CheckBox2</b>	Properties	Caption	Задать значение «2».
<b>CheckBox3</b>	Properties	Caption	Задать значение «3».
<b>CheckBox4</b>	Properties	Caption	Задать значение «4».
<b>CheckBox5</b>	Properties	Caption	Задать значение «5».

Выделенный объект	Вкладка окна Object Inspector	Имя свойства/ имя события	Действие
<b>Label1</b> (Вкладка Standard)	Properties	Caption	Введите текст: «Сумма: 0»
		Autosize	Задайте значение False
		Alignment	Задайте значение taCenter

4. Сохраните проект, запустите и протестируйте его.

#### Листинг подпрограммы

```

procedure TForm1.FormCreate (Sender: TObject);
begin
    s := False;
end;

procedure TForm1.CheckBox1Click (Sender: TObject);
var Index, i, num : Integer;
    e: Boolean;
begin
    if s then Exit; {Если программа снова вызовет процедуру обработки, будет выполнен
                    оператор Exit – немедленный выход из процедуры}
    s := True;
    for Index := 0 to 4 do {Оператор break прерывает выполнение цикла}
        if Sender = Controls[Index] then break; {Определяется, какой флажок был переключен}
        {Когда выполнение цикла завершается, значение переменной Index соответствует
        переключенному флажку}
    {Если значение свойства Checked (Установлен) равно False (Нет), флажок сброшен, а если
    True (Да) – установлен. Номер флажка в массиве определяется переменной Index}
    if not (Controls[Index] as TCheckBox).Checked
        then (Controls[Index] as TCheckBox).Checked := True {Условие выполнено, если
        флажок сейчас сброшен, т.е. до щелчка он был установлен}
        else {Программирование изменения состояния «дополнительных флажков». Текущий
        флажок уже переключен}
        begin
            num := Index + 1; {Переключение двух следующих флажков}
            if Index = 4 then num := 0; {Изменение состояния нулевого флажка}
            {Состояние флажка надо поменять на противоположное}
            (Controls[num] as TCheckBox).Checked := not (Controls[num] as TCheckBox).Checked;
            {Выполнение проверки на выход за пределы массива}
            num := num + 1; if Index = 3 then num := 0;
            (Controls[num] as TCheckBox).Checked := not (Controls[num] as TCheckBox).Checked;
        end;
    e := True; {Головоломка решена, если установлены все пять флажков}
    for i := 0 to 4 do
        e := e and (Controls[i] as TCheckBox).Checked;
        {После цикла значение останется равным True, если все флажки установлены}
    if e then {Головоломка решена}
        for i := 0 to 4 do
            begin
                (Controls[i] as TCheckBox).Caption := 'Победа!';
                (Controls[i] as TCheckBox).Enabled := False; {Отключение флажков}
            end;
    s := False;
end;

```



## Практическая работа № 20 «Обычный калькулятор»

**Цель работы** - создать программу, выполняющую действия обычного калькулятора.

### Описание плана разработки программы

1. Открыть новый проект.
2. Разместить на форме экземпляры компонентов: **Label**, **Edit**, **Button**.

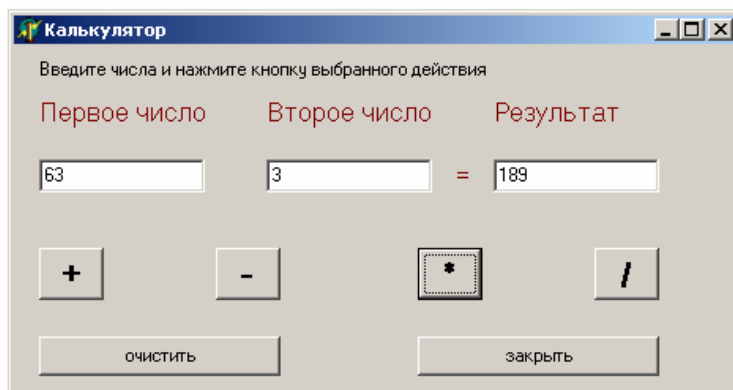


Рис. 41.

3. Сохраните проект, запустите и протестируйте его.

### Листинг подпрограммы

```
procedure TForm1.FormCreate(Sender: TObject);
begin
    Edit1.Text := "";
    Edit2.Text := "";
    Edit3.Text := "";
end;

procedure TForm1.Button5Click(Sender: TObject);
begin
    Edit1.Text := "";
    Edit2.Text := "";
    Edit3.Text := "";
end;

procedure TForm1.Button6Click(Sender: TObject);
begin
    close;
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
    a := StrToFloat (Edit1.Text);
    b := StrToFloat (Edit2.Text);
    c := a + b;
    Edit3.Text := FloatToStr (c);
end;

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

```

begin
  a := StrToFloat (Edit1.Text);
  b := StrToFloat (Edit2.Text);
  c := a - b;
  Edit3.Text := FloatToStr (c);
end;

procedure TForm1.Button3Click(Sender: TObject);
begin
  a := StrToFloat (Edit1.Text);
  b := StrToFloat (Edit2.Text);
  c := a * b;
  Edit3.Text := FloatToStr (c);
end;

procedure TForm1.Button4Click(Sender: TObject);
begin
  a := StrToFloat (Edit1.Text);
  b := StrToFloat (Edit2.Text);
  if b = 0 then Edit3.Text := 'division by zero'
  else
    begin
      c := a / b;
      Edit3.Text := FloatToStr (c);
    end;
end;

```

#### Задание для самостоятельного выполнения

1. Создать программу, выполняющую действия простого инженерного калькулятора.

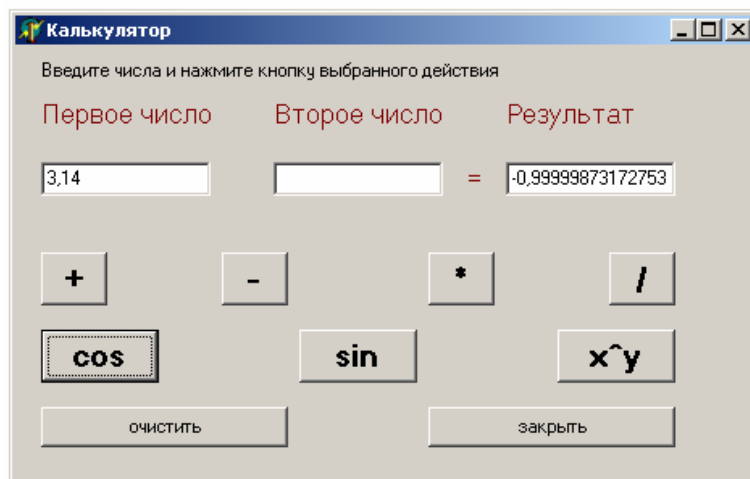


Рис. 42.

**Фрагмент подпрограммы** (для вычисления функции Cos(x))

```

if Edit1.Text <> " " then
  begin
    a := StrToFloat (Edit1.Text);
    c := cos (a);
  end;
if Edit2.Text <> " " then

```

```

begin
  b := StrToFloat (Edit2.Text);
  c := cos (b);
end;

```

```

Edit3.Text := FloatToStr (c);

```

Добавить на форму кнопки для вычисления функций  $Tg(x)$ ,  $\log_a b$ .

2. Создать программу, выполняющую перевод чисел из десятичной системы счисления в двоичную систему и обратно.

Рис. 43.

## Практическая работа № 21 «Строковый калькулятор»

**Цель работы** - создать программу, выполняющую действия строкового калькулятора.

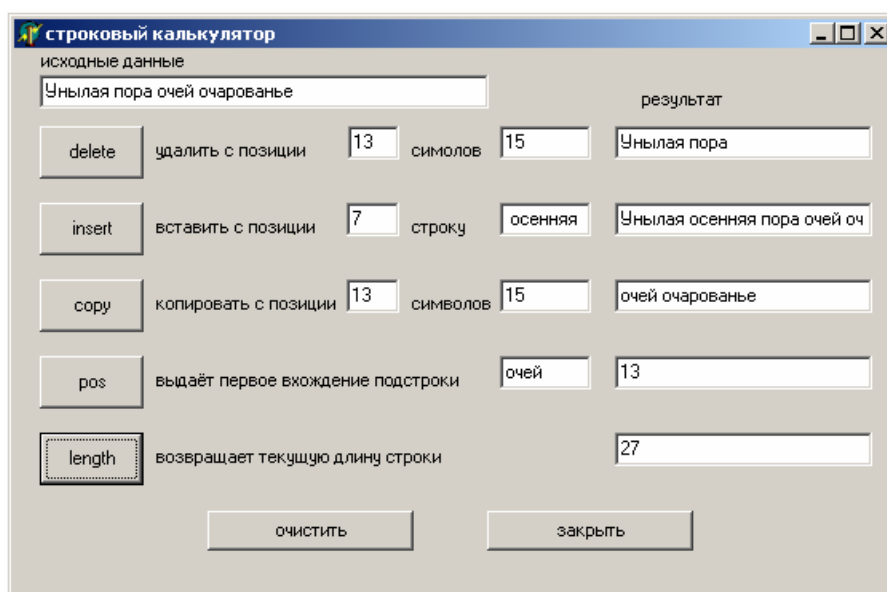


Рис. 44.

### Описание плана разработки программы

1. Открыть новый проект.
2. Разместить на форме экземпляры компонентов: **Edit, Label, Button**.

### Фрагмент программы

```
procedure TForm1.Button2Click(Sender: TObject);
begin
  s:=Edit1.Text;
  k:=StrToInt(Edit2.Text);
  t:=StrToInt(Edit3.Text);
  Delete(s,k,t);
  Edit10.Text:=s;
end;
```

```
procedure TForm1.Button3Click(Sender: TObject);
begin
  s:=Edit1.Text;
  q:=StrToInt(Edit4.Text);
  w:=Edit5.Text;
  Insert(w,s,q);
  edit11.Text:=s;
end;
```

```
procedure TForm1.Button4Click(Sender: TObject);
begin
  s:=Edit1.Text;
  p:=StrToInt(Edit6.Text);
  l:=StrToInt(Edit7.Text);
```

```

        Copy(s,p,l);
        s1:=copy(s,p,l);
        Edit12.Text:=s1;
    end;

procedure TForm1.Button5Click(Sender: TObject);
begin
    s:=Edit1.Text;
    r:=Edit9.Text;
    a:=Pos(r,s);
    edit13.Text:=IntToStr(a);
end;

procedure TForm1.Button6Click(Sender: TObject);
begin
    s:=edit1.Text;
    b:=Length(s);
    edit14.Text:=IntToStr(b);
end;

```

## Практическая работа № 22

### Нахождение индекса в массиве случайных чисел

**Цель работы** - создать программу, которая находит индекс числа в массиве случайных чисел.

#### Описание плана разработки программы

1. Открыть новый проект.
2. Разместить на форме экземпляры компонентов: **Button**, **Edit**, **Label**.
3. Выполнить следующие действия:

Таблица 22.

Выделенный объект	Вкладка окна Object Inspector	Имя свойства/ имя события	Действие
<b>Form1</b>	Properties	Caption	Установка имени формы «Новый проект»
	Events	OnCreate	Очистить значения свойств Text текстовых полей
<b>Button1</b>	Properties	Caption	Введите название «Очистить»
	Events	OnClick	Очистить значения свойств Text текстовых полей
<b>Button2</b>	Properties	Caption	Введите название «Закрыть»
	Events	OnClick	Обработка события закрытия формы
<b>Button3</b>	Properties	Caption	Введите название «Найти первый индекс»
	Events	OnClick	Обработка события нахождения индекса введенного числа
<b>Button4</b>	Properties	Caption	Введите название «Ввести случайным образом»
	Events	OnClick	Ввод массива случайным образом
<b>Edit1</b>	Properties	Caption	Очистить значение свойства Text
<b>Edit2</b>	Properties	Caption	Очистить значение свойства Text
<b>Edit3</b>	Properties	Caption	Очистить значение свойства Text

4. Введите переменные  $ik, k, i$  : integer;  $s$  : string;  $a$  : array [1..15] of integer.
5. Сохраните проект, запустите и протестируйте его.

#### Листинг программы

```

procedure TForm1.FormCreate(Sender: TObject);
begin
    Edit1.Text := '';
    Edit2.Text := '';
    Edit3.Text := '';
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
    Edit1.Text := '';
    Edit2.Text := '';
    Edit3.Text := '';
end;

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

```

```

end;
procedure TForm1.Button3Click(Sender: TObject);
begin
  k := StrToInt (Edit2.Text);
  For i:= 1 to 15 do
    if k = a[i] then ik := i;
  if ik = 0 then Edit3.Text := 'number ubsent'
    else Edit3.Text := IntToStr (ik);
end;
procedure TForm1.Button4Click(Sender: TObject);
begin
  randomize;
  s := '';
  For i := 1 to 15 do
    begin
      a[i] := random (26);
      s := concat (s, IntToStr (a[i]), #32);
    end;
  Edit1.Text := s;
end;
end.

```

Вид проекта по действиям:

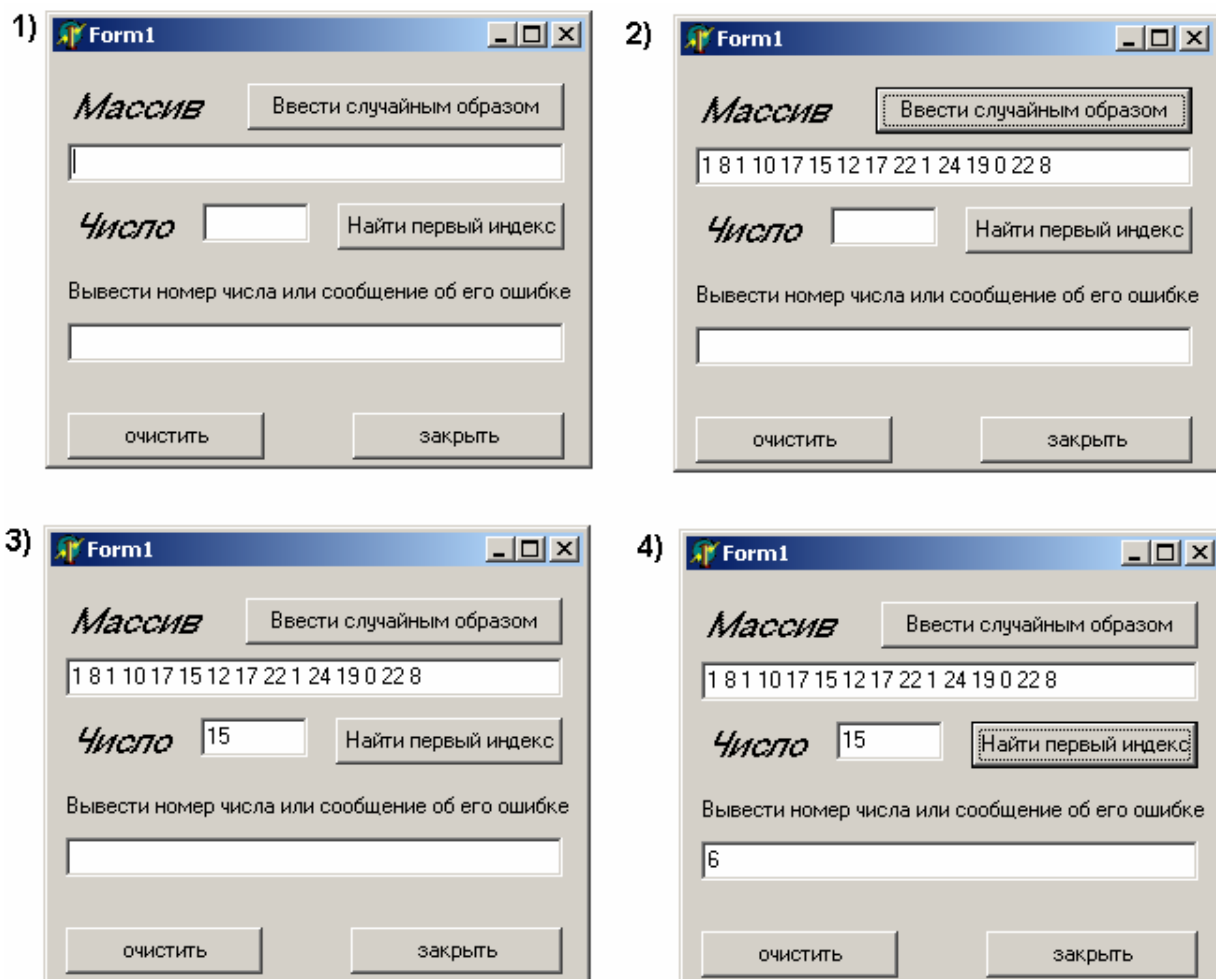


Рис. 45.

## Практическая работа № 23

### Нахождение минимального и максимального числа в массиве

**Цель работы** - создать программу, которая находит минимальное и максимальное числа в введенном массиве.

#### Описание плана разработки программы

1. Открыть новый проект.
2. Разместить на форме экземпляры компонентов: **Button**, **Edit**, **Label**.
3. Выполнить следующие действия:

Таблица 23.

Выделенный объект	Вкладка окна Object Inspector	Имя свойства/ имя события	Действие
<b>Form1</b>	Properties	Caption	Установка имени формы «Поиск»
	Events	OnCreate	Очистить значения свойств Text текстовых полей
<b>Button1</b>	Properties	Caption	Введите название «Очистить»
	Events	OnClick	Очистить значения свойств Text текстовых полей
<b>Button2</b>	Properties	Caption	Введите название «Заккрыть»
	Events	OnClick	Обработка события закрытия формы
<b>Button3</b>	Properties	Caption	Введите название «Поиск»
	Events	OnClick	Обработка события нахождения минимального и максимального чисел во введенном массиве
<b>Edit1</b>	Properties	Caption	Очистить значение свойства Text
<b>Edit2</b>	Properties	Caption	Очистить значение свойства Text
<b>Edit3</b>	Properties	Caption	Очистить значение свойства Text

4. Введите переменные  
s, ss: string ; a: array [1..15] of integer; I, j, k, max, min, p, code : integer.
5. Сохраните проект, запустите и протестируйте его.

#### Листинг подпрограмм

```

procedure TForm1.FormCreate(Sender: TObject);
begin
    Edit1.Text := '';
    Edit2.Text := '';
    Edit3.Text := '';
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
    Edit1.Text := '';
    Edit2.Text := '';
    Edit3.Text := '';
end;

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

```



**end;**

**procedure** TForm1.Button3Click(Sender: TObject);

**begin**

s := Edit1.Text;

s := concat (s, #32);

i := 0;

**while** Length(s) > 0 **do**

**begin**

i := i+1;

p := pos (#32,s);

ss := copy (s,1,p-1);

**Val** (ss,k,code);

a[i] := k;

**delete**(s,1,p);

**end;**

max := a[1];

**For** j := 1 **to** i **do**

**if** max < a[j] **then** max := a[j];

min := a[1];

**For** j := 1 **to** i **do**

**if** min > a[j] **then** min := a[j];

Edit3.Text := IntToStr (max);

Edit2.Text := IntToStr (min);

**end;**

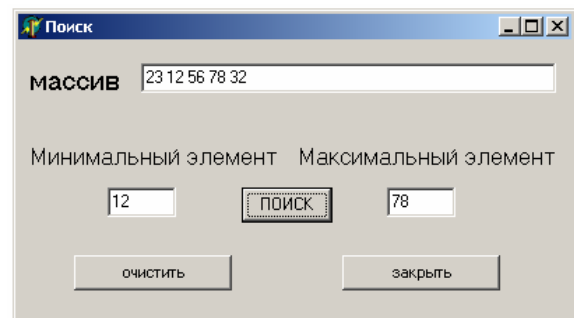
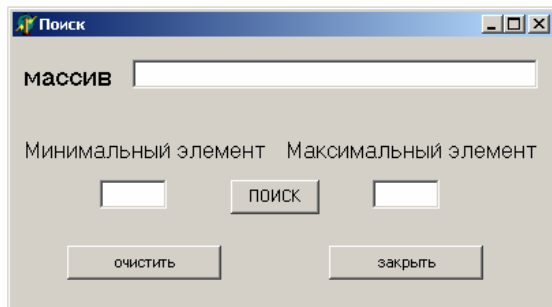


Рис. 46.

## Практическая работа № 24 «Текущее время и текущая дата»

**Цель работы** - создать программу, которая выводит текущее время и текущую дату.

### Описание плана разработки программы

1. Открыть новый проект.
2. Разместить на форме экземпляры компонентов: **Button**, **Edit**, **Label**.
3. Выполнить следующие действия:

Таблица 24.

Выделенный объект	Вкладка окна Object Inspector	Имя свойства/ имя события	Действие
<b>Form1</b>	Properties	Caption	Установка имени формы «Таймер»
<b>Button1</b>	Properties	Caption	Введите название «Текущее время»
	Events	OnClick	DateTime:=Time; Edit1.Text:=TimeToStr(DateTime);
<b>Button2</b>	Properties	Caption	Введите название «Текущая дата»
	Events	OnClick	Edit2.Text:=DateToStr(Date);
<b>Edit1</b>	Properties	Caption	Очистить значение свойства Text
<b>Edit2</b>	Properties	Caption	Очистить значение свойства Text

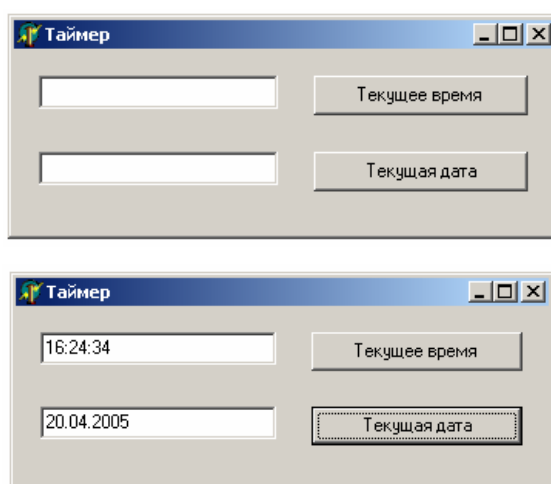


Рис. 47.

### Листинг подпрограмм

**var**

DateTime : TdateTime;

**procedure** TForm1.Button1Click(Sender: TObject);

**begin**

DateTime:=Time;

Edit1.Text:=TimeToStr(DateTime);

**end;**

**procedure** TForm1.Button2Click(Sender: TObject);

**begin**

Edit2.Text:=DateToStr(Date);

**end;**

**end.**

## Практическая работа № 25 «Электронные часы»

**Цель работы** - написать программу «Электронные часы», в окне которой отображается текущее время, дата и день недели.



Рис. 48.

### Описание плана разработки программы

1. Открыть новый проект.
2. Разместить на форме экземпляры компонентов: **Label, Timer**.
3. Выполнить следующие действия:
  - 3.1. Ввести константы, отвечающие за названия дней недели и месяцев.
  - 3.2. Объявить процедуру ShowTime вручную для доступа к компонентам формы напрямую.
  - 3.3. Процедура ShowTime отображает текущее время.
  - 3.4. Процедура FormTime обрабатывает событие Paint.
  - 3.5. Процедура Timer1Timer обрабатывает сигнал таймера.
  - 3.6. Процедура FormCreate обрабатывает событие OnCreate.

Листинг программы:

```
unit Unit1;
```

```
interface
```

```
uses
```

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

```
type
```

```
TForm1 = class(TForm)  
    Timer1: TTimer;  
    Label1: TLabel; // время  
    Label2: TLabel; // дата и день недели  
    procedure Timer1Timer(Sender: TObject);  
    procedure FormCreate(Sender: TObject);  
    procedure FormPaint(Sender: TObject);  
    procedure ShowTime;  
private  
    { Private declarations }  
public  
    { Public declarations }  
end;
```

```
var
```

```
Form1: TForm1;
```

```
implementation
```

```

const
  stDay : array[1..7] of string[11] =
    ('воскресенье','понедельник','вторник',
     'среда','четверг','пятница','суббота');

  stMonth : array[1..12] of string[8] =
    ('января','февраля','марта',
     'апреля','мая','июня','июля',
     'августа','сентября','октября',
     'ноября','декабря');

{$R *.dfm}
procedure TForm1.ShowTime;
var Time : TDateTime; // текущее время
begin
  Time := Now(); // получить системное время
  Label1.Caption := FormatDateTime('hh:mm:ss',Time);
end;

procedure TForm1.Timer1Timer(Sender: TObject);
begin
  ShowTime; // отобразить время
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  Present: TDateTime; // текущая дата и время
  Year, Month, Day : Word; // год, месяц и число, как
    // отдельные числа
begin
  Present:= Now; // получить текущую дату
  DecodeDate(Present, Year, Month, Day);
  Label2.Caption := 'Сегодня '+IntToStr(Day)+' ' +
    stMonth[Month] + ' '+ IntToStr(Year)+
    ' года, '+ stDay[DayOfWeek(Present)];

  // настроить и запустить таймер
  Timer1.Interval := 1000; // период сигналов таймера 1 с
  Timer1.Enabled := True; // пуск таймера
end;

procedure TForm1.FormPaint(Sender: TObject);
begin
  ShowTime; // отобразить часы
end;

end.

```

## Практическая работа № 26

### Графика

Для рисования статичных рисунков используется компонент PaintBox (панель System). Этот компонент размещается на форме в виде прозрачного пунктирного квадрата, и в его пределах можно рисовать. Рисование выполняется обращением к свойству Canvas (графическая канва) этого компонента: PaintBox1.Canvas. У него в свою очередь есть свойство Pixels (PaintBox1.Canvas.Pixels), которое представляет собой матрицу, двумерный массив заданного размера - поточечный образ канвы, каждый элемент - отдельная точка. В Pixels[] отсчет точек (пикселей экрана) начинается с 0.

Координаты x, y отсчитываются от верхнего левого угла, то есть он считается точкой с координатой (0, 0), увеличение по оси x идет слева направо, а по оси y - сверху вниз. Для конкретной точки указывается цвет. Функция RGB() формирует цвет комбинацией интенсивности красного, зеленого и синего (интенсивность задается числом от 0 до 255). Например, черный - RGB(0,0,0), красный - rgb(255,0,0), синий - rgb(0,0,255), белый - rgb(255,255,255).

**Цель работы** - создать программу, выполняющую следующие действия:

1. Разместить на форме компонент PaintBox.
2. Заполнить доступную канву 300 красными точками в случайных позициях по нажатиям на некоторую кнопку.
3. Для выхода из программы необходимо щелкнуть мышью на закрывающей кнопке в строке заголовка.
4. Записать код в обработчике нажатия.



Рис. 49.

```
procedure TForm1.Button1Click(Sender: TObject);  
  var i,x,y: Integer;  
  begin  
    randomize;  
    for i := 1 to 300 do  
      begin  
        x := random(100);  
        y := random(100);  
        PaintBox1.Canvas.Pixels[x,y] := RGB(255,0,0);  
      end  
    end;  
  end;
```

5. Изменить RGB(255,0,0) на RGB(random(255),random(255),random(255) ).

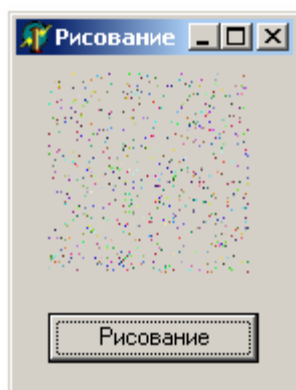


Рис. 50.

6. Заполнить фон черным цветом перед началом выполнения программы с помощью метода канвы FillRect. Метод вызывается с указанием прямоугольной области заливки цветом: FillRect(Rect(0, 0, 100, 100)) // координаты верхнего левого и правого нижнего углов  
Вложенное слово Rect формирует данное типа "прямоугольник". Перед вызовом FillRect надо указать цвет заливки: PaintBox1.Canvas.Brush.Color := RGB(0,0,0);

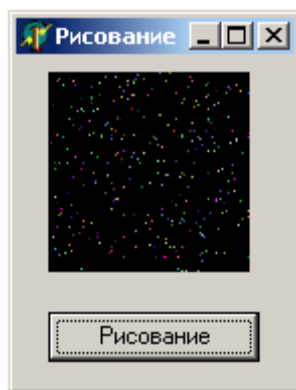


Рис. 51.

```

procedure TForm1.Button1Click(Sender: TObject);
var i,x,y: Integer;
begin
    PaintBox1.Canvas.Brush.Color := RGB(0,0,0);
    PaintBox1.Canvas.FillRect(Rect(0,0,100,100));
    for i := 1 to 300 do
        begin
            x := random(100);
            y := random(100);
            PaintBox1.Canvas.Pixels[x,y] := RGB(random(255),random(255),random(255));
        end
    end;

```

7. Заполнить канву случайными разноцветными линиями разной толщины. Линия рисуется с помощью методов (сначала задается начальная точка, потом конечная):  
PaintBox1.Canvas.MoveTo(10,10);  
PaintBox1.Canvas.LineTo(50,50);  
Цвет линии и толщина задаются свойством канвы Pen (карандаш). Pen.Color - цвет карандаша, Pen.Width - толщина линии в пикселах (по умолчанию - 1).

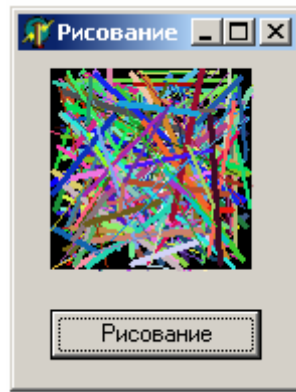


Рис. 52.

**for i := 1 to 300 do**

**begin**

PaintBox1.Canvas.Pen.Color := RGB(random(255),random(255),random(255));

PaintBox1.Canvas.Pen.Width := random(3)+1;

x := random(100);

y := random(100);

PaintBox1.Canvas.MoveTo(x,y);

x := random(100);

y := random(100);

PaintBox1.Canvas.LineTo(x,y);

**end**

8. Заполнить канву эллипсами случайным образом (круги, окружности - частный случай эллипса). Эллипсы рисуются методом `Ellipse()` с четырьмя параметрами - координатами верхнего левого и правого нижнего углов прямоугольника, в который эллипс вписывается. Кайма эллипса рисуется в соответствии с параметрами свойства `Pen` канвы, а заливается эллипс внутри цветом кисти `Brush` канвы.



Рис. 53.

**for i := 1 to 300 do**

**begin**

// цвет и ширина каймы будущего эллипса

PaintBox1.Canvas.Pen.Color := RGB(random(255),random(255),random(255)) ;

PaintBox1.Canvas.Pen.Width := random(3)+1;

// цвет заливки внутренней эллипса

PaintBox1.Canvas.Brush.Color := rgb(random(255),random(255),random(255)) ;

```

// координаты углов прямоугольника, в который вписывается эллипс
x := random(150);
y := random(150);
x2 := random(150);
y2 := random(150);
// вписываем эллипс
PaintBox1.Canvas.Ellipse(x, y, x2, y2);
end;

```

9. Сделать канву на весь экран. Заполнить разноцветными прямоугольниками в случайных позициях и случайных размеров с помощью FillRect().
10. Создать графический образ так называемого множества Жюлиа.

```

procedure TForm1.Button1Click(Sender: TObject);

```

```

var RE,IM,RE1,IM1: REAL ;

```

```

V,X,Y : INTEGER;

```

```

// вывод точки заданного цвета

```

```

procedure PUTPIXEL(x,y,c:Integer);

```

```

var cc: TColor;

```

```

begin

```

```

    case c mod 8 of

```

```

        0:cc:=clBlack;

```

```

        1:cc:=clRed;

```

```

        2:cc:=clLime;

```

```

        3:cc:=clYellow;

```

```

        4:cc:=clBlue;

```

```

        5:cc:=clFuchsia;

```

```

        6:cc:=clAqua;

```

```

        7:cc:=clWhite;

```

```

    end;

```

```

    PaintBox1.Canvas.Pixels[x,y] := cc;

```

```

end;

```

```

procedure QWA ;

```

```

begin

```

```

    RE1:=RE*RE-IM*IM;

```

```

    IM1:=2*RE*IM;

```

```

    RE:=RE1;

```

```

    IM:=IM1;

```

```

end;

```

```

procedure KUB;

```

```

begin

```

```

    RE1:=RE*(RE*RE-3*IM*IM);

```

```

    IM1:=IM*(3*RE*RE-IM*IM);

```

```

    RE:=RE1;

```

```

    IM:=IM1;

```

```

end ;

```

```

Begin

```

```

    PaintBox1.Canvas.Brush.Color := rgb(0,0,0);

```

```

    PaintBox1.Canvas.FillRect(Rect(0,0,640,480));

```

```

    X:=-320 ;

```

```

    REPEAT Y:=-240 ;

```

```

    REPEAT V:=0 ; RE:=-1+0.001*X ; IM:=0+0.001*Y ;

```

```

    REPEAT KUB; RE:=RE+1.00003 ; IM:=IM+1.01828201638 ;

```

```

    if RE*RE > 50 then break;

```



```

if IM*IM > 50 then break;
V:=V+1 ;
UNTIL V>40 ;
if ( ABS(RE) > 10 ) or ( ABS(IM) > 1000)
  then
    begin
      PUTPIXEL((X+320),(Y+240),TRUNC(V)) ;
    end
  else PUTPIXEL((X+320),(Y+240),0);
Y:=Y+1 ;
UNTIL Y > 241 ;
X:=X+1 ;
UNTIL X>320 ;
End;

```

## Практическая работа № 27 «Олимпийский флаг»

**Цель работы** - создать программу, которая на поверхности формы рисует олимпийский флаг.

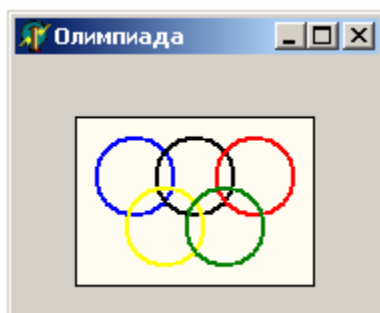


Рис. 54.

**Листинг программы:**

```
unit Unit1;  
  
interface  
  
uses  
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,  
  Dialogs;  
  
type  
  TForm1 = class(TForm)  
    procedure FormPaint(Sender: TObject);  
  private  
    { Private declarations }  
  public  
    { Public declarations }  
  end;  
  
var  
  Form1: TForm1;  
  
implementation  
  
{$R *.dfm}  
  
procedure TForm1.FormPaint(Sender: TObject);  
begin  
  with Canvas do  
    begin  
      // полотно  
      Canvas.Pen.Width := 1;  
      Canvas.Pen.Color := clBlack;  
      Canvas.Brush.Color := clCream;  
      Rectangle(30,30,150,115);  
  
      // кольца
```

```
Pen.Width := 2;  
Brush.Style := bsClear; // область внутри круга  
                        // не закрашивать  
Pen.Color := clBlue;  
Ellipse(40,40,80,80);  
Pen.Color := clBlack;  
Ellipse(70,40,110,80);  
Pen.Color := clRed;  
Ellipse(100,40,140,80);  
Pen.Color := clYellow;  
Ellipse(55,65,95,105);  
Pen.Color := clGreen;  
Ellipse(85,65,125,105);  
end;  
end;  
  
end.
```

## Практическая работа № 28 «Узоры»

**Цель работы** - создать программу, выполняющую следующие действия без участия пользователя:

1. После запуска программы в окне изображаются рисунки, созданные самой программой по заранее заданным правилам.
2. Картинка обновляется «сама собой». Интервал таймера может быть любым. Он зависит от компьютера, на котором работает программа.
3. Для выхода из программы необходимо щелкнуть мышью на закрывающей кнопке в строке заголовка.

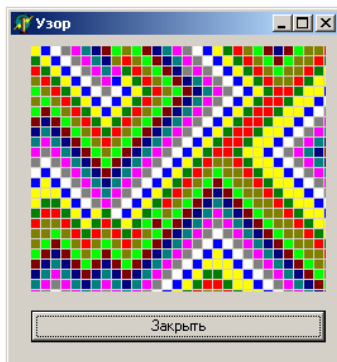


Рис. 55.

### Описание плана разработки программы

1. Открыть новый проект.
2. Разместить на форме экземпляры компонентов: область рисования **PaintBox**, таймер **Timer**.
3. Область рисования представить в виде клеток (точек или пикселей). Каждая клетка покрашена в свой цвет. В ходе программы цвета меняются.
4. Описать предварительно константы в окне кода перед ключевым словом:  
`Const size = 40;`  
`csize = 12;`  
`Colors: array [1..16] of TColor =`  
`(clRed, clGreen, clYellow, clBlue, clWhite, clGray, clFuchsia, clTeal,`  
`clNavy, clMaroon, clLime, clOlive, clPurple, clSilver, clAqua, clBlack);`  
 Константа `size` задает число клеточек по каждому направлению, константа `csize` – число используемых цветов. Массив `Colors` (константа) определяет цвета.  
 В разделе описаний после ключевого слова `var` описывается массив клеток  
`Points: array[1..size, 1..size] of Integer;`
5. Выполнить следующие действия:

Таблица 25.

Выделенный объект	Вкладка окна Object Inspector	Имя свойства/ имя события	Действие
<b>Form1</b>	Properties	Caption	Установка имени формы «Узор»
	Events	OnCreate	В процедуре обеспечить неповторимость случайных чисел с помощью процедуры <b>Randomize</b> . Провести инициализацию массива пикселей.
<b>PaintBox1</b> (Вкладка System)	Properties	Height	Задать значение 320
		Width	Задать значение 320

Выделенный объект	Вкладка окна Object Inspector	Имя свойства/ имя события	Действие
		Canvas	Свойства и методы этого свойства обеспечивают рисование. Метод <b>Rectangle</b> рисует прямоугольник с заданными вершинами. Цвет контура задается свойством <b>Pen.Color</b> , а цвет заливки – свойством <b>Brush.Color</b> .
<b>Timer1</b> (Вкладка System)	Properties	Interval	Задать значение 100 (одна десятая секунды)
	Events	OnTimer	В созданной процедуре-заготовке <b>Timer1Timer</b> описать переменные <b>c (color)</b> , <b>up (up)</b> , <b>d (down)</b> , <b>l (left)</b> , <b>r (right)</b> . Новые значения цветов записываются в отдельном массиве <b>NewPoints</b> . Написать правила, по которым будут меняться цвета.

6. Сохраните проект, запустите и протестируйте его.

#### Листинг подпрограммы

```

procedure TForm1.FormCreate (Sender: TObject);
var i, j : Integer;
begin
    Randomize;
    for i := 1 to size do {инициализация массива пикселей}
        for j := 1 to size do
            Points[i, j] := 1 + Random (csize); {Минимально возможное значение элемента
                                                    массива равно 1}
end;

procedure TForm1.Timer1Timer (Sender: TObject);
var i, j : Integer;
    c, l, r, u, d : Integer;
    newPoints: array [1..size, 1..size] of Integer;
begin
    {Вычислить, какого цвета будет клетка на следующем шаге}
    for i := 1 to size do
        for j := 1 to size do
            begin
                c := Points[i, j] + 1; {Вычисляется «следующий цвет» и запоминается в переменной c}
                if c > csizes then c := 1; {После последнего цвета идет первый}
                {Вычисляются индексы для клеток, примыкающих к данной сверху, снизу, слева и
                справа. Края узора как бы «склеены» друг с другом}
                u := i - 1;
                if u = 0 then u := size;
                d := i + 1;
                if d > size then d := 1;
                l := j - 1;
                if l = 0 then l := size;
                r := j + 1;
                if r > size then r := 1;
            
```

```

newPoints [i, j] := Points [i, j]; {Если среди «соседей» цвет отсутствует, то клетка
                                  остается без изменений}
{Если хотя бы один из «соседей» имеет такой цвет, клетка перекрашивается}
if (Points [u, j] = c) or (Points [d, j] = c) or (Points [i, l] = c) or (Points [i, r] = c)
then newPoints [i, j] := c;
end;
c := 320 div size; {Выбирается размер клетки так, чтобы узор занимал, по возможности,
                   всю область рисования}
{Обновляется узор на экране}
for i := 1 to size do
  for j := 1 to size do
    begin
      Points [i, j] := newPoints [i, j];
      {Настроить цвет контура прямоугольника и цвет закрашки}
      PaintBox1.Canvas.Pen.Color := Colors[Points[i, j]];
      PaintBox1.Canvas.Brush.Color := Colors[Points[i, j]];
      {Выполнить рисование}
      PaintBox1.Canvas.Rectangle (c*(i - 1), c*(j - 1), c*i - 1, c*j - 1);
      {Параметры метода Rectangle подобраны так, чтобы между клетками оставался
       небольшой зазор. Чтобы клетки располагались вплотную друг к другу, заменить
       (c*(i - 1), c*(j - 1), c*i - 1, c*j - 1) на (c*(i - 1), c*(j - 1), c*i , c*j }
    end;
  end;
end;

```

## Практическая работа № 29

### Перемещение рисунка

**Цель работы** - создать программу, в которой на поверхности окна перемещается случайным образом изображение веселой рожицы. Пользователь должен сделать щелчок кнопкой мыши по изображению. Программа должна завершить работу после того, как пользователь сделает 10 щелчков кнопкой мыши.

Начало игры осуществляется по нажатию на кнопку *Ok*.

Свойство *WordWrap* компонента *Label* – признак того, что слова, которые не помещаются в текущей строке, автоматически переносятся на следующую строку (значение свойства *AutoSize* должно быть *False*).

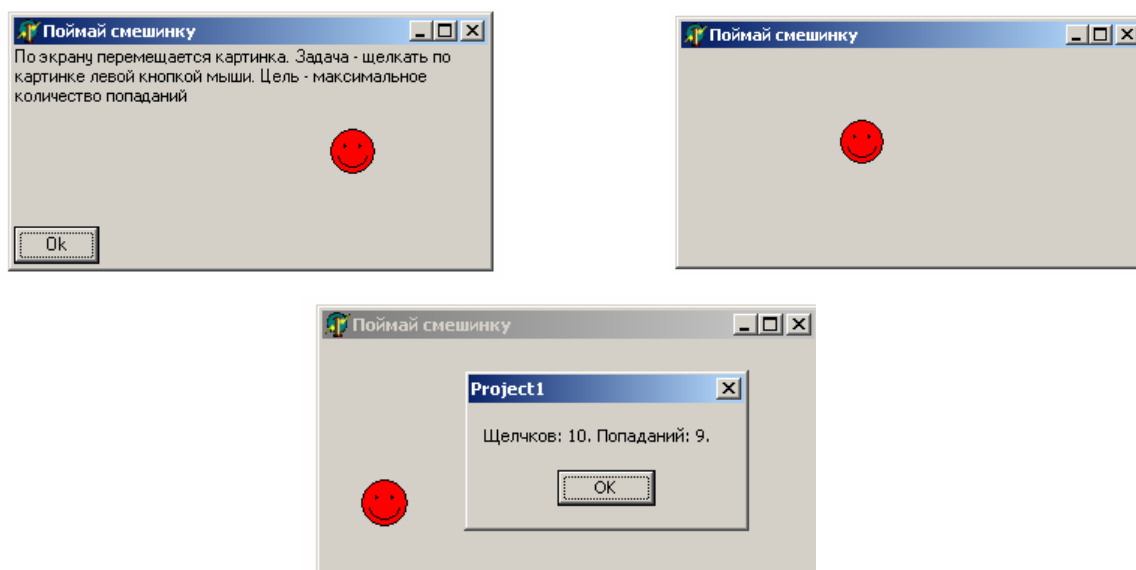


Рис. 56.

#### Листинг программы:

```
unit Unit1;
```

```
interface
```

```
uses
```

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

```
type
```

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

```
Timer: TTimer;
```

```
Label1: TLabel;
```

```
Button1: TButton;
```

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

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

```
procedure FormMouseDown(Sender: TObject; Button: TMouseButton;
```

```
Shift: TShiftState; X, Y: Integer);
```

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

```
private
```

```
{ Private declarations }
```

```
public
```

```
{ Public declarations }
```

```
{ объявление процедур помещено сюда,  
чтобы процедуры имели прямой доступ  
к форме, на которой они рисуют }
```

```
procedure PaintFace (x,y: integer); {рисует рожицу}  
procedure EraseFace(x,y: integer); {стирает рожицу}  
end;
```

```
var
```

```
Form1: TForm1;  
fx,fy: integer; { координаты рожицы}  
n: integer;     { количество щелчков кнопкой мыши}  
p: integer;     { количество попаданий}
```

```
implementation
```

```
{ рисует рожицу}  
procedure TForm1.PaintFace(x,y: integer);  
begin  
  Canvas.Pen.Color := clBlack;  { цвет линий}  
  Canvas.Brush.Color := clRed;  { цвет закрашки}  
  { рисуем рожицу}  
  Canvas.Ellipse(x,y,x+30,y+30); { лицо}  
  Canvas.Ellipse(x+9,y+10,x+11,y+13); { левый глаз}  
  Canvas.Ellipse(x+19,y+10,x+21,y+13); { правый глаз}  
  Canvas.Arc(x+4,y+4,x+26,y+26,x,y+20,x+30,y+20); {улыбка}  
end;
```

```
{ стирает рожицу}  
procedure TForm1.EraseFace(x,y: integer);  
begin  
  { зададим цвет границы и цвет закрашки, совпадающий с цветом формы.}  
  { По умолчанию цвет формы - clBtnFace }  
  Canvas.Pen.Color := clBtnFace; { цвет окружности}  
  Canvas.Brush.Color := clBtnFace; { цвет закрашки}  
  Canvas.Ellipse(x,y,x+30,y+30);  
end;
```

```
{ $R *.dfm }
```

```
procedure TForm1.TimerTimer(Sender: TObject);  
begin  
  EraseFace(fx,fy);  
  { новое положение рожицы}  
  fx:= Random(ClientWidth-30); { 30 - это диаметр рожицы}  
  fy:= Random(ClientHeight-30);  
  PaintFace(fx,fy);  
end;
```

```
procedure TForm1.FormCreate(Sender: TObject);
```



```

begin
    { исходное положение рожицы}
    fx:=100;
    fy:=100;
    Randomize; { инициализация генератора случайных чисел}
end;

procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton;
    Shift: TShiftState; X, Y: Integer);
begin
    inc(n); { кол-во щелчков}

    if (x > fx) and (x < fx+30) and (y > fy) and (y < fy+30)
    then begin
        { щелчок по рожице}
        inc(p);
        end;
    if n = 10 then
        begin
            { игра закончена}
            Timer.Enabled := False; { остановить таймер}
            ShowMessage('Щелчков: 10. Попаданий: ' + IntToStr(p)+'.');
            EraseFace(fx,fy);
            Label1.Visible := True;
            Button1.Visible := True;
            { теперь кнопка и сообщение снова видны}
        end;
    end;

procedure TForm1.Button1Click(Sender: TObject);
begin
    Label1.Visible := False; { скрыть сообщение}
    Button1.Visible := False; { скрыть кнопку}
    Timer.Enabled := True; { пуск таймера}
end;

end.

```

## Практическая работа № 30

### Рисунок

**Цель работы** - создать программу, выводящую на форму приведенный ниже рисунок. По нажатию на клавишу «Лучи» происходит смена направления лучей. Изменить программу так, чтобы лучи изменяли направления без нажатия клавиши.



Рис. 57.

1. Добавьте в форму компоненты Image и Button.
2. Создайте обработчик события нажатия. Важнейшим свойством компонента Image является свойство Canvas.TCanvas (холст). Это свойство само является объектом. У Canvas есть методы и свойства. Это свойства и методы рисования на холсте.
  - *Pen*: Tpen (перо): рисование линий, границ фигур и т.п. производится пером. Важнейшие свойства пера – Color: Tcolor (цвет), Width: Integer (ширина), Style: TpenStyle (стиль).
  - *Brush*: Tbrush (кисть): закрашка фигур, заднего фона надписей и т.п. производится кистью. Важнейшие свойства кисти – Color: Tcolor (цвет), Style: TbrushStyle (стиль).
  - *Font*: Tfont (шрифт): надписи на холсте выполняются с учетом значений его свойства Font. При этом задний фон за надписью закрашивается текущим значением кисти. Для того чтобы закрашки не было, нужно установить прозрачный стиль кисти: Brush.Style := bsClear, но затем не забудьте сделать кисть непрозрачной: Brush.Style := bsSolid, иначе фигуры, которые вы будете в дальнейшем рисовать, тоже окажутся незакрашенными. Важнейшие свойства шрифта – Name: string (имя), Size: Integer (размер), Color: Tcolor (цвет). Вы также использовали два метода холста:
    - *Rectangle*(X1, Y1, X2, Y2: Integer) (прямоугольник): рисует прямоугольник, закрашивая его кистью и обводя пером (в нашем случае, для того чтобы рамки не было видно, мы сделали цвет кисти и пера одинаковым). В качестве параметров задаются координаты верхнего левого X1, Y1 и нижнего правого X2, Y2 углов относительно верхнего левого угла холста.
    - *TextOut*(X, Y: Integer; const Text: string) (вывести текст): рисует текст согласно заданному шрифту и закрашивая задний фон согласно заданной кисти (в нашем случае эта закрашка незаметна, поскольку текст рисуется на том же фоне, что и текущее значение кисти). В качестве параметров задаются координаты верхнего левого угла текста X, Y относительно верхнего левого угла холста и текст надписи Text.

3. Нарисуйте градиентно закрашенное синее небо отдельными линиями, каждую следующую линию рисуя более светлой. Таким образом, рисование выполняется в цикле. Необходимо определить локальную переменную. В цикле меняются значения цветов, устанавливается перо в точку и чертится линия из этой точки в другую.
- *MoveTo*(X, Y: Integer) (передвинуть перо): передвигает позицию пера в точку холста в координаты X, Y. При этом на холсте ничего не рисуется;
  - *LineTo*(X, Y: Integer) (чертить линию): чертит линию из точки, в которой находилось перо, в точку X, Y, используя текущее перо. При этом перо тоже перемещается в эту точку.
- Местоположение пера хранится в свойстве холста *PenPos*: *Tpoint*. Запись в эту переменную полностью эквивалентна вызову метода *MoveTo*. Цвет можно задавать, используя константы цвета *clBlack*, *clNavy*, *clGreen* и т.д. – полный список констант находится в выпадающем списке свойства *Color* в Инспекторе Объектов. Цвет также можно задать, используя функцию *RGB* (*Red*, *Green*, *Blue*: byte), в параметрах *Red*, *Green* и *Blue* указываются значения интенсивности красной, зеленой и синей компоненты цвета (от 0 до 255 каждая).
4. Добавить на небо солнце. Нарисовать эллипс с линиями-лучами. Лучи нарисовать случайным образом, для этого необходимо использовать генератор случайных чисел.
- Круги и эллипсы рисуются с помощью метода холста ***Ellipse***(*X1*, *Y1*, *X2*, *Y2*: Integer). Как и прямоугольник, эллипс закрашивается текущей кистью и обводится текущим пером. В качестве параметров *X1*, *Y1*, *X2*, *Y2* необходимо указать левый верхний и нижний правый углы прямоугольника, в который «вписан» эллипс. Если  $Y2 - Y1 = X2 - X1$  (то есть прямоугольник является квадратом), эллипс рисуется как круг.
  - Датчик случайных чисел – функция *random* выдает число типа *double* в диапазоне от 0 до 1.
5. Нарисовать облака. Можно воспользоваться картинкой из файла. Нарисовать в стандартном редакторе *Paint* облако: а) установить атрибуты картинки – ширина 100, высота 50; б) закрасить картинку черным цветом – этот цвет в программе будет прозрачным; в) нарисовать облако, используя инструмент *Распылитель* и различные оттенки серого; г) проследите, чтобы левая нижняя точка картинки осталась черной (именно по этой точке автоматически определяется прозрачный цвет).
6. Сохранить картинку в файл.
7. Добавить в форму еще один *Image*, щелкнуть по нему мышкой и загрузить в него картинку. Сделать его прозрачным можно, установив свойство *Transparent*. Поскольку он вспомогательный, лучше, чтобы он не был виден во время работы программы, поэтому отключите ему свойство *Visible*.
8. Добавить компонент *Timer*. Установите его интервал 100.
9. На вкладке событий Инспектора Объектов напротив события *OnTimer* в выпадающем списке выберите имя обработчика события *Button1Click*. Таким образом устанавливается уже существующий обработчик для еще одного события.
10. В программе каждые сто миллисекунд будет вызываться событие и происходить перерисовка картинки, как если бы мы нажимали каждый раз кнопку. Замедлить интервал между перерисовкой картинки можно, увеличив интервал таймера.
11. При рисовании часто требуется использовать более сложные объекты, чем просто комбинации линий прямоугольников и эллипсов. Может возникнуть необходимость использования готового растрового изображения. Кроме того, может понадобиться копирование изображения из одного *Image* на другой. Для этого используется метод холста ***Draw***(*X*, *Y*: Integer; *Graphic*: *Tgraphic*). Этот метод практически копирует графический объект на холст в область с координатами верхнего левого угла X,Y. Графическим объектом *Graphic* могут быть битовые картинки, иконки и метафайлы. Для нас важным является его применение к копированию картинок из одного *Image* в другой. Это делается следующим образом:

*ImageA.Canvas.Draw(X, Y, ImageB.Picture.Bitmap)*

При этом все, что нарисовано на ImageB, будет скопировано на ImageA. Если ImageB.Transparent = true, то картинка копируется с прозрачным фоном.

Вместо ImageA и ImageB соответственно подставьте нужные вам имена компонентов, фигурирующие в вашей программе:

*Image1.Canvas.Draw(130 + random(2), 20 + random(2), Image2.Picture.Bitmap);*

*Image1.Canvas.Draw(100 + random(2), 50 + random(2), Image2.Picture.Bitmap);*

*Image1.Canvas.Draw(170 + random(2), 80 + random(2), Image2.Picture.Bitmap);*

Таким образом, для рисования можно использовать любые картинки из файла – облака, домики, персонажей, логотипы и т.п.

## Листинг программы

**procedure** TForm1.Button1Click(Sender: TObject);

**var** i:integer;

**begin**

Image1.Canvas.Pen.Color := clGreen;

Image1.Canvas.Brush.Color := clGreen;

Image1.Canvas.Rectangle(0, 200, 300, 300);

Image1.Canvas.Font.Color := clWhite;

Image1.Canvas.Font.Size := 30;

Image1.Canvas.Font.Name := 'Arial';

Image1.Canvas.TextOut(10, 250, 'Delphi');

**for** i := 0 to 200 **do**

**begin**

Image1.Canvas.Pen.Color := RGB(I, I, 255);

Image1.Canvas.MoveTo(0, i);

Image1.Canvas.LineTo(300, i);

**end;**

Image1.Canvas.Pen.Color := clRed;

Image1.Canvas.Brush.Color := clYellow;

Image1.Canvas.Ellipse(20, 20, 60, 60);

Image1.Canvas.Pen.Color := clYellow;

**for** i := 1 to 20 **do**

**begin**

Image1.Canvas.MoveTo(40, 40);

Image1.Canvas.LineTo(5 + random(70), 5 + random(70));

**end;**

**end;**

**end.**

## Практическая работа № 31

### Построение графика

**Цель работы** - создать программу построения графика, в которой устанавливается масштаб, в цикле осуществляется построение графика функции, рисуются оси координат и печатаются на них числовые шкалы.

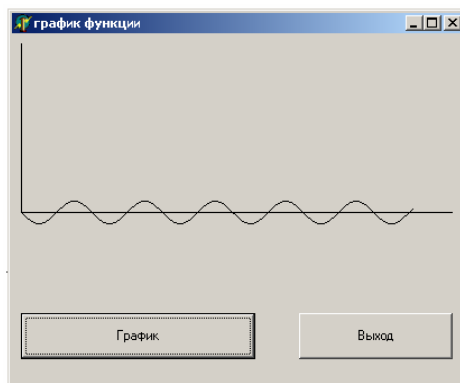


Рис. 58.

#### Описание плана разработки программы

1. Открыть новый проект.
2. Разместить на форме экземпляры компонентов: кнопка **Button**, область рисования **PaintBox**.
3. Выполнить следующие действия:

Таблица 26.

Выделенный объект	Вкладка окна Object Inspector	Имя свойства/имя события	Действие
<b>Form1</b>	Properties	Caption	Установка имени формы «График функции»
<b>Button1</b>	Properties	Caption	Введите название «График»
	Events	OnClick	Написать процедуру, рисующую график по точкам
<b>PaintBox1</b>	Properties	Canvas	Соединение линиями получаемых точек
<b>Button2</b>	Properties	Caption	Введите название «Выход»
	Events	OnClick	Close;

4. Сохраните проект, запустите и протестируйте его.

#### Листинг программы

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

```
function f(x:integer):integer;
var c:integer;
begin
    c:=round(10*sin(0.1*x));
    f:=c;
end;
```

```
var x:integer;
begin
    PaintBox1.Canvas.LineTo(0,150);
```

```
PaintBox1.Canvas.LineTo(410,150);
x:=0;
for x:=0 to 350 do
  begin
    PaintBox1.Canvas.LineTo(x, f(x)+150);
  end;
end;

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

## Практическая работа № 32 «Градусник»

**Цель работы** - создать программу, которая переводит значение температуры по Цельсию в значения температуры по Фаренгейту. Введите графический объект изображения значения температуры.

1. Поместите на форму два поля ввода Edit и четыре кнопки.

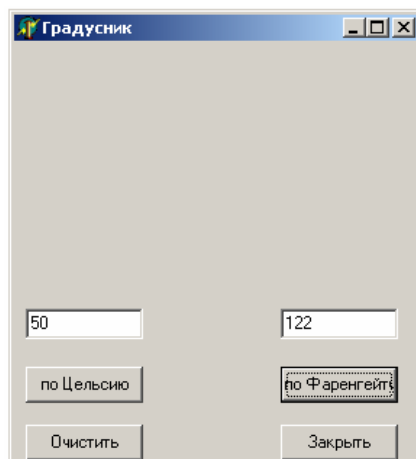


Рис. 59.

2. Ввести число в левое поле. По нажатию на кнопку «по Фаренгейту» в правом поле выводится преобразованное число. По нажатию кнопки «Очистить» очищаются поля ввода.
3. Ограничьте вводимые температуры диапазоном от 0°C до 100°C (если введено значение, превышающее 100°C или 212°F, то при нажатии кнопки в полях ввода должно отобразиться 100 и 212 соответственно).
4. Отобразить столбик термометра графически. Добавьте компоненты Image.
5. Сделать видимой только ту кнопку, которая необходима для преобразования.

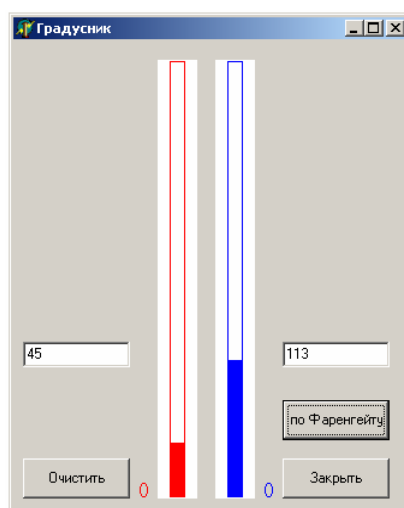


Рис. 60.

### Листинг программы

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

Dialogs, StdCtrls, ExtCtrls;

**type**

```
TForm1 = class(TForm)
  Edit1: TEdit;
  Edit2: TEdit;
  Button1: TButton;
  Button2: TButton;
  Button3: TButton;
  Button4: TButton;
  Image1: TImage;
  Image2: TImage;
  Label1: TLabel;
  Label2: TLabel;
  procedure FormCreate(Sender: TObject);
  procedure Button2Click(Sender: TObject);
  procedure Button1Click(Sender: TObject);
  procedure Button3Click(Sender: TObject);
  procedure Button4Click(Sender: TObject);
```

**private**

```
{ Private declarations }
```

**public**

```
{ Public declarations }
```

**end;**

**var**

```
Form1: TForm1;
```

**implementation**

```
{ $R *.dfm }
```

**procedure** TForm1.FormCreate(Sender: TObject);

**begin**

```
Edit1.Text := '';
Edit2.Text := '';
Image1.Canvas.Pen.Color := clRed;
Image1.Canvas.Brush.Color := clWhite;
Image1.Canvas.Rectangle(10,1,23,360);
Image2.Canvas.Pen.Color := clBlue;
Image2.Canvas.Brush.Color := clWhite;
Image2.Canvas.Rectangle(10,1,23,360);
```

**end;**

**procedure** TForm1.Button2Click(Sender: TObject);

**var** a, b, c : real;

**begin**

```
a := StrToFloat(Edit1.Text);
if (a > 0) or (a < 100) then b := 1.8 * a + 32;
if a > 100 then b := 212;
if a < 0 then b := 32;
Edit2.Text := FloatToStr(b);
```



```

Image1.Canvas.Pen.Color := clRed;
Image1.Canvas.Brush.Color := clWhite;
Image1.Canvas.Rectangle(10,1,23,360);
Image1.Canvas.Pen.Color := clRed;
Image1.Canvas.Brush.Color := clRed;
Image1.Canvas.Rectangle(10,round(360-a),23,360);
Image2.Canvas.Pen.Color := clBlue;
Image2.Canvas.Brush.Color := clWhite;
Image2.Canvas.Rectangle(10,1,23,360);
Image2.Canvas.Pen.Color := clBlue;
Image2.Canvas.Brush.Color := clBlue;
Image2.Canvas.Rectangle(10,round(360-b),23,360);
end;

```

```

procedure TForm1.Button1Click(Sender: TObject);
var a, c, d : real;
begin
  c := StrToFloat(Edit2.Text);
  if (c > 32) or (c < 212) then d := ( c - 32)/1.8;
  if c < 32 then d := 0;
  if c > 212 then d := 100;
  Edit1.Text := FloatToStr(d);
  Image2.Canvas.Pen.Color := clBlue;
  Image2.Canvas.Brush.Color := clWhite;
  Image2.Canvas.Rectangle(10,1,23,360);
  Image2.Canvas.Pen.Color := clBlue;
  Image2.Canvas.Brush.Color := clBlue;
  Image2.Canvas.Rectangle(10,round(360-c),23,360);
  Image1.Canvas.Pen.Color := clRed;
  Image1.Canvas.Brush.Color := clWhite;
  Image1.Canvas.Rectangle(10,1,23,360);
  Image1.Canvas.Pen.Color := clRed;
  Image1.Canvas.Brush.Color := clRed;
  Image1.Canvas.Rectangle(10,round(360-d),23,360);
end;

```

```

procedure TForm1.Button3Click(Sender: TObject);
begin
  Edit1.Text := "";
  Edit2.Text := "";
  Image1.Canvas.Pen.Color := clRed;
  Image1.Canvas.Brush.Color := clWhite;
  Image1.Canvas.Rectangle(10,1,23,360);
  Image2.Canvas.Pen.Color := clBlue;
  Image2.Canvas.Brush.Color := clWhite;
  Image2.Canvas.Rectangle(10,1,23,360);
end;

```

```

procedure TForm1.Button4Click(Sender: TObject);
begin
  Close;
end;

```

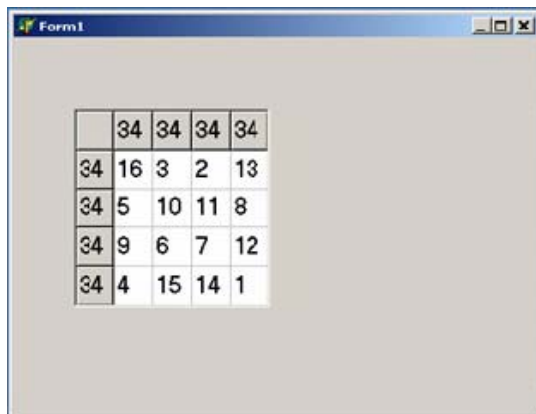
```
procedure TForm1.Edit1Enter(Sender: TObject);  
begin  
    Button2.Visible := true;  
    Button1.Visible := False;  
end;  
  
procedure TForm1.Edit2Enter(Sender: TObject);  
begin  
    Button1.Visible := True;  
    Button2.Visible := False;  
end;  
end.
```

### Практическая работа № 33

#### Вывод табличных данных

На одной из знаменитых гравюр Альбрехта Дюрера изображен магический квадрат. Одним из его свойств является то, что суммы по всем столбцам, строкам и диагоналям равны.

**Цель работы** - создать программу, рассчитывающую суммы по столбцам и строкам вводимых чисел.



	34	34	34	34
34	16	3	2	13
34	5	10	11	8
34	9	6	7	12
34	4	15	14	1

Рис. 61.

1. Добавить на форму сетку – компонент StringGrid. Этот компонент служит для ввода и вывода табличных данных.
2. Необходимо ввести данные и описать событие реагирования на их ввод. В свойстве Options установите пункт Editing, иначе сетка будет доступной только для чтения. Теперь при запуске программы можно вводить текст.
3. Щелкнув мышкой в Инспекторе Объектов, описать обработчик события OnSelectCell, которое возникает, когда пользователь выбирает какую-либо ячейку для редактирования.
4. Теперь при запуске программы и выборе ячейки в заголовке формы отображается информация о столбце и строке ввода.
5. Таким образом, в разделе uses возникла ссылка на модуль Grids, в котором описаны компоненты сеток. В описании класса формы добавилась переменная – ссылка на компонент сетки, а также описание метода – обработчика события. Сам обработчик описан уже в разделе implementation.
6. Обработать событие, возникающее, когда пользователь пытается выделить какую-нибудь ячейку.

```
procedure TForm1.StringGrid1SelectCell(Sender: TObject; Acol, Arow: Integer;  
                                     var CanSelect: Boolean);
```

```
begin
```

```
    Caption := 'Выделена клетка (' + IntToStr(Acol) + ':' + IntToStr(Arow) + ')';
```

```
end;
```

В качестве параметров обработчик получает (кроме Sender) номер столбца, номер строки и переменную CanSelect, которую можно изменить внутри обработчика, так как она передается как var.

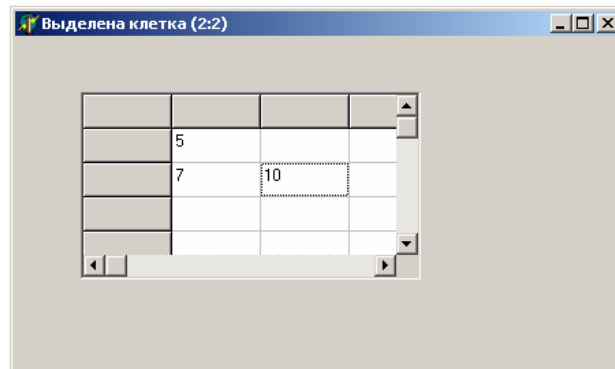


Рис. 62.

Можно, например, запретить выделение ячеек третьего столбца. Для этого вставьте в этот обработчик еще строчку: `CanSelect := (Acol <> 3);`  
В этом случае `CanSelect` будет равен `false`, если `Acol = 3`.

7. Настроить сетку в зависимости от значения констант, которые надо объявить в `interface`. Теперь при запуске программы сетка имеет размер, заданный в константе, и выглядит значительно аккуратнее. Если изменить значение константы, то при запуске и размер сетки будет соответствующим.
8. Установить размер сетки согласно значению констант `Num` и `cSize`, объявленных в `interface`:

```
const Num = 4;  
cSize = 30;
```

```
.....
```

```
procedure TForm1.FormCreate(Sender: TObject);  
begin  
  MyGrid.DefaultColWidth := cSize;  
  MyGrid.DefaultRowHeight := cSize;  
  MyGrid.ColCount := Num;  
  MyGrid.RowCount := Num;  
  MyGrid.Width := Num * (cSize + 1) + 3;  
  MyGrid.Height := Num * (cSize + 1) + 3;  
  MyGrid.Font.Size := cSize div 2;  
end;
```

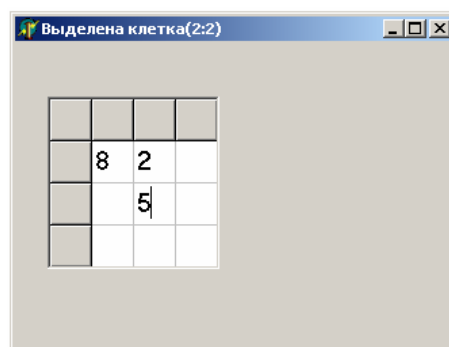


Рис. 63.

В приведенном участке кода изменяются ширина и высота сетки, установленные по умолчанию, а также количество строк и столбцов сетки. Добавление единицы к ширине каждой ячейки связано с наличием линий между ячейками, а добавление тройки ко всей сумме – наличием бордюра вокруг сетки. В последней строчке устанавливается соответствующий величине клетки размер шрифта.

9. Для расчета суммы по столбцам и строкам вводимых чисел написать собственную процедуру и две функции.

Функции будут рассчитывать сумму в строке и столбце, получая их номер в качестве параметра. Процедура будет в цикле вызывать эти функции и соответствующим образом заполнять клетки.

Теперь надо создать обработчик для нажатия на саму форму и описать в нем вызов расчета. Собственная процедура стоит в коде раньше, чем обработчик, из которого она вызывается. Когда описывается собственная процедура, нужно обращаться к компонентам через форму.

При запуске программы требуется ввести числа в сетку и кликнуть на саму форму.

**Function** ColSum(n: integer): integer;

**var**

i: integer;

**begin**

Result := 0;

**for** I := 1 **to** Num – 1 **do** Result := Result + StrToInt (Form1.MyGrid.Cells[n, i]);

**end;**

**function** RowSum(n: integer): integer;

**var**

i: integer;

**begin**

Result := 0;

**for** I := 1 **to** Num – 1 **do** Result := Result + StrToInt(Form1.MyGrid.Cells[I, n]);

**end;**

**procedure** Calculate;

**var** i: integer;

**begin**

**for** I := 1 **to** Num – 1 **do**

**begin**

Form1.MyGrid.Cells[I, 0] := IntToStr(ColSum(i));

Form1.MyGrid.Cells[0, i] := IntToStr(RowSum(i));

**end;**

**end;**

В данной программе используется свойство Cells, компонента сетки. Это свойство имеет тип двумерного массива строк. Счет в этом массиве начинается с нуля. Фиксированные (серые) клетки, с точки зрения индексирования массива, ничем не отличаются от прочих. В нашем случае фиксированы первая строка и первый столбец (в массиве они имеют соответствующие нулевые координаты). Поэтому при расчете суммы проходят от 1 (а не от 0) до Num – 1 (при индексировании с нуля последний столбец/строка имеют, понятно, номер Num – 1).

Однако, если произойдет ввод какой-нибудь буквы, сразу возникнет исключительная ситуация. Она, очевидно, возникает из-за невозможности перевода буквы в число в функции StrToInt. Если нежелательно, чтобы при возникновении исключительной ситуации программа в Delphi переходила в режим отладки, можно отключить флажок Menu => Tools => Debugger Options => Language Exceptions => Stop on Delphi Exceptions. Теперь программа будет выполняться так, как если бы она выполнялась из-под Windows. Теперь каждый раз, когда вводится нечисловой символ, программа не останавливается, а только появляется стандартное окно Windows с сообщением об ошибке.

С помощью обработки исключений можно избежать появления этого окна. Для этого надо добавить в функцию конструкцию `try..except`. Таким образом «опасная» команда (или целый блок) помещается внутрь конструкции `try..except..end` или `try..finally..end`:

```
function StrToVal(s: string): integer;  
begin  
  if S = '' then Result := 0 else  
    try  
      Result := StrToInt(s);  
    except  
      Result := 0;  
      Form1.Caption := 'Вводить надо числа!';  
    end;  
end;
```

Если возникает исключительная ситуация, т.е. введено нечисловое значение, считается, что введен ноль, а также выводится сообщение об этом пользователю. Еще нужно добавить вывод информации, что все в порядке. А также удалить код внутри обработчика Click формы. В результате, если введено неправильное значение, то окно ошибки не появляется, а в Caption формы выводится напоминание.

## Практическая работа № 34

### «Игра»

**Цель работы** - создать программу – игру. Игрок управляет пушкой зенитки, его боевое задание – справиться с нашествием воздушных шаров. Воздушные шары несут бомбы, которые они сбросят, как только окажутся над пушкой. Необходимо не допустить этого и уничтожить их все на подлете. Снаряды не ограничены, но следующий выстрел можно делать только после того, как выпущенный снаряд поразит цель, упадет на землю или уйдет из зоны видимости.

В игре участвуют:

- воздушные шары,
- зенитная пушка,
- пушечный снаряд,
- бомба,
- внешняя среда.

Для шаров введен специальный тип `TBalloon`., в котором содержатся данные о координатах шара, его скорости, состоянии и цвете. Массив переменных `Balloons` типа `TBalloon` будет содержать полную информацию обо всех шарах. Индексирование идет, начиная с нуля. Общее число шаров в массиве задается соответствующей константой.

Кроме того, к шарам относятся константы, определяющие:

- количество шаров,
- их возможные цвета,
- возможную высоту над землей (всего предполагается четыре уровня),
- интервал между шарами и их радиус.

**type**

`TBalloon = record`

`x, y, v, Explosion: integer;`

`Color: TColor;`

`end;`

**const**

`BallCount = 10;`

`BallColors: array [0 .. 9] of TColor = (clRed, clGreen, clNavy, clMaroon, clPurple, clOlive, clLime, clYellow, clFuchsia, clSilver);`

`BallAltitude: array [0 .. 3] of integer = (240, 160, 200, 120);`

`BallInterval = 40;`

`BallRadius = 15;`

`V = 150;`

`g = -9.8 * 3;`

`dt = 0.1;`

**var**

`Form1: TForm1;`

`x, y, Vx, Vy, BombY, BombV: double;`

`Angle, GunPosition, GunExplosion: integer;`

`Balloons: array [0 .. BallCount - 1] of TBalloon;`

Зенитная пушка описывается переменными:

- угол наклона пушки,
- ее положение
- ее состояние.

Пушечный снаряд описывается переменными:

- координатами,
- текущей скоростью,

- начальной скоростью при выстреле.
- Бомба, сбрасываемая с воздушного шара, имеет:
- координату,
  - скорость.

Для расчета положений объектов нужно знать ускорение падения и шаг по времени.

В качестве состояния шара и пушки мы используем переменные `TBalloon`. В определенный момент времени каждый элемент должен находиться в каком-то состоянии, определяющем то, как элемент выглядит на экране. Шары и пушка могут быть боеспособными, взрывающимися и уничтоженными.

Если `Explosion = 0` - шар боеспособен, если `Explosion = 10`, то он уничтожен, а значения от 1 до 9 обозначают фазы взрыва. Чтобы инициировать взрыв боеспособного шара, нужно присвоить `Explosion := 1`. То же самое касается и пушки.

Состояние, в котором скорость снаряда равна нулю:  $V_x = 0$  и  $V_y = 0$ . В этом случае считают, что можно сделать новый выстрел. Пока же снаряд летит (то есть скорость его не нулевая), новый выстрел сделать нельзя.

Бомба будет сбрасываться шаром, когда он поравняется с пушкой. Таким образом, X - координата бомбы всегда равна X - координате пушки. Следовательно, необходимо хранить только Y-координату бомбы. Бомба не сброшена, если `BombY = 1000`. Если `BombY < 1000`, то она считается сброшенной и начинает падать.

Поместите на форму `TrackBar`, `Button`, `Image` и `Timer` и настройте их параметры.

`Timer: Interval = 100; Image: Width = 440, Height = 260; Button: Caption = 'Новая игра';`  
`TrackBar: Min = -90, Max = 90, Frequency = 10.`

`Image` - сражение в реальном времени. `Timer` - реальное время. `Button` - запуск новой игры. `TrackBar` - управление зенитной пушкой.

В обработчике нажатия на кнопку "Новая игра" необходимо обнулить параметры, расставить пушку и шары. Для каждого шара устанавливается случайное направление движения:  $v = 1$  или  $-1$ . В зависимости от направления, определяется высота, на которой находится шар (на одном и том же уровне шары летят в одну сторону). По x-координате шары расставляются так, чтобы они шли с примерно равным интервалом в направлении пушки из-за границ экрана.

Каждый шар `Balloons[i]`, элемент массива `Balloons`, имеет тип записи `TBalloon`, и для того, чтобы получить доступ к его элементам, нужно писать: `Balloons[i].Explosion := 0`, `Color := BallColors[random(10)]` и т.д. Конструкция **with** `Balloons[i]` **do begin .. end** позволяет избежать повторения.

```
procedure TForm1.Button1Click(Sender: TObject);
var
  i: integer;
begin
  Caption := 'Нашествие';
  TrackBar1.Position := 0;
  GunPosition := 170 + random(100);
  Vx := 0;
  Vy := 0;
  GunExplosion := 0;
  for i := 0 to BallCount - 1 do
    with Balloons[i] do
      begin
        if random < 0.5 then v := -1 else v := 1;
        y := BallAltitude[1 + v + random(2)] + random(10);
        x := 220 - v * (220 + i * BallInterval + random(20));
        Explosion := 0;
```



```

    Color := BallColors[random(10)];
end;
BombY := 1000;
end;

```

В обработчик события формы OnCreate вводится датчик случайных чисел, устанавливается клиентский размер формы (ClientWidth := 455; ClientHeight := 315;). Размер клиентской области формы в приложениях определяется автоматически как Width и Height формы минус ширина заголовка и бордюров.

```

procedure TForm1.FormCreate(Sender: TObject);
begin
    Randomize;
    ClientWidth := 455;
    ClientHeight := 315;
    Button1.Click;
end;

```

Пушка танка будет управляться с помощью компонента TrackBar. Выстрел производится нажатием пробела на клавиатуре. При этом фокус ввода должен быть у TrackBar. В обработчике OnKeyPress реализуется выстрел, задавая начальную скорость и положение снаряда, а также издавая звук выстрела.

Процедура, проигрывающая этот звук из wav-файла, описана в модуле, который необходимо подключить в разделе uses. В модуле MMSystem имеется функция PlaySound, с помощью которой можно проигрывать файлы wav. Для асинхронного воспроизведения (приложение не приостанавливает работу на время воспроизведения, а проигрывает его в фоновом режиме) вызов выглядит так:

```

    PlaySound(PChar('имяфайла.wav'), 0, SND_ASYNC).

```

В стандартных звуках Microsoft Office можно подобрать звуки для выстрела и взрыва. В зависимости от версии программы, эти файлы хранятся в C:\Windows\Media\ Microsoft Office 2000, C:\Program Files\Microsoft Office\Office\Media или в какой-то подобной папке. Найдите звуки для выстрела (условно назовем GunShot.wav) и взрыва (условно назовем Explode.wav) и скопируйте их в папку, в которой сохранен проект.

```

procedure TForm1.TrackBar1KeyPress(Sender: TObject; var Key: Char);
var
    i: integer;
begin
    if (Key = ' ') and (Vx = 0) and (Vy = 0) and (GunExplosion = 0) then
        begin
            try
                PlaySound(PChar('GunShot.wav'), 0, SND_ASYNC);
            except
            end;
        end;
    Vx := V * Sin(Angle * pi / 180);
    Vy := V * Cos(Angle * pi / 180);
    x := GunPosition + 15 * Sin(Angle * pi / 180);
    y := 20 + 15 * Cos(Angle * pi / 180);
end;

    if (Key = 'Q') then
        begin

```

```

try
  PlaySound(PChar('Explode.wav'), 0, SND_ASYNC);
except
end;
for i := 0 to BallCount - 1 do
  if Balloons[i].Explosion = 0 then
    Balloons[i].Explosion := 1;
    Caption := 'Уничтожен';
  end;
end;
end;

```

В обработчике изменения положения бегунка TrackBar меняем наклон пушки.

```

procedure TForm1.TrackBar1Change(Sender: TObject);
begin
  Angle := TrackBar1.Position;
end;

```

Выстрел происходит, если:

- 1) нажат пробел;
- 2) снаряд готов ( $V_x = 0$ ) и ( $V_y = 0$ );
- 3) пушка боеспособна ( $GunExplosion = 0$ ).

Процедура MoveAll, вызываемая каждый такт таймера, отвечает за то, чтобы все процессы шли своим чередом. Вставьте ее сразу после implementation.

```

procedure MoveAll;
var i: integer;
begin
  for i := 0 to BallCount - 1 do
    with Balloons[i] do
      if Explosion = 0 then
        x := x + v
      else begin
        if Explosion < 10 then
          inc(Explosion);
        end;
      end;
    end;
  if (Vx <> 0) or (Vy <> 0) then
    begin
      x := x + Vx * dt;
      y := y + Vy * dt;
      Vy := Vy + g * dt;
    end;
  if BombY < 1000 then
    begin
      BombY := BombY + BombV * dt;
      BombV := BombV + g * dt;
    end;
  if (GunExplosion > 0) and (GunExplosion < 10) then
    inc(GunExplosion);
  end;
end;

```

Процедура пододвигает все шары, которые живы (у них Explosion = 0), в направлении вектора их скорости. Остальные шары, если они находятся в фазе взрыва (0 < Explosion < 10), переходят в следующую фазу.

Если снаряд выпущен ( $V_x \neq 0$ ) или ( $V_y \neq 0$ ), то его координата и скорость изменяются согласно законам природы.

Если бомба сброшена (BombY < 1000), то она также подчиняется закону всемирного тяготения.

Если танк находится в фазе взрыва (0 < GunExplosion < 10), он переходит в следующую фазу.

Ошибки, возникающие в обработчике таймера или процедурах, которые вызываются из обработчика, не останавливают таймер и, значит, через секунду возникают опять. Чтобы их остановить, приходится использовать Ctrl-Alt-Del...

В этой процедуре мы работаем с массивом. Одной из самых частых ошибок, которые не так-то просто найти, является выход за границы массива. Для предотвращения подобных ситуаций включите проверку на выход из диапазона допустимых значений (Project => Options => Compiler => Runtime errors => Range checking), которая по умолчанию отключена.

Перемещения сами по себе могли бы продолжаться бесконечно. Необходимо проконтролировать:

- не попал ли снаряд в шар, следовательно, взорвать его;
- не уничтожены ли все шары, следовательно, выдать надпись о победе;
- не поравнялся ли шар с пушкой, следовательно, сбросить бомбу;
- не упал ли снаряд и не вышел ли он из зоны контроля, следовательно, перезарядить пушку;
- не упала ли бомба на пушку, следовательно, закончить игру.

Для контроля за передвижениями предусмотрена процедура:

```
procedure CheckCollisions;  
var  
i, j: integer;  
HappyEnd: boolean;  
begin  
  for i := 0 to BallCount - 1 do  
    with Balloons[i] do  
      begin  
        if (Explosion = 0) then  
          begin  
            if (sqr(x - Unit1.x) + sqr(y - Unit1.y) < sqr(BallRadius)) then  
              begin  
                try  
                  PlaySound(PChar('Explode.wav'), 0, SND_ASYNC);  
                except end;  
                Explosion := 1;  
                Unit1.x := 0;  
                Unit1.y := 0;  
                Unit1.Vx := 0;  
                Unit1.Vy := 0;  
                HappyEnd := (GunExplosion = 0);  
                for j := 0 to BallCount - 1 do  
                  HappyEnd := HappyEnd and (Balloons[j].Explosion > 0);  
                if HappyEnd then  
                  Form1.Caption := 'Победа!';  
                end;  
              end;  
            end;  
          end;  
        end;  
      end;  
    end;  
  end;  
end;
```

```

if (x = GunPosition) and (BombY = 1000) and (GunExplosion = 0) then
  begin
    BombY := y - BallRadius - 5;
    BombV := 0;
  end;
end;
end;
if (y < 0) or (x < 0) or (x > 440) then
  begin
    x := 0;
    y := 0;
    Vx := 0;
    Vy := 0;
  end;
  if BombY < 10 then
    begin
      BombY := 1000;
      GunExplosion := 1;
      try
        PlaySound(PChar('Explode.wav'), 0, SND_ASYNC);
      except end;
      Form1.Caption := 'Увы...';
    end;
  end;

```

В этом блоке возможная ошибка связана с использованием **with ... do begin ... end**.

X-координата снаряда хранится в глобальной переменной x, X-координата воздушного шара – в переменной Balloons[i].x. Однако когда пишут код внутри **with Balloons[i] do begin ... end**, то именно к координате шара обращаются просто как к x. Если нужно добраться до глобальной переменной, то теперь уже перед ней нужно ставить уточнение: Unit1.x, где Unit1 – имя модуля.

Рисование отдельных элементов батальи также разумно поместить в специализированные для этого процедуры.

Художественные образы шаров, пушки и взрыва передаются с помощью процедур:

```

procedure DrawBalloon(x, y: integer; Color: TColor);
begin
  with Form1.Image1.Canvas do
    begin
      Pen.Color := clBlack;
      Brush.Color := Color;
      Ellipse(x - BallRadius, y - BallRadius, x + BallRadius, y + BallRadius);
      Pen.Color := clWhite;
      Brush.Color := clWhite;
      Ellipse(x - BallRadius div 2 - 3, y - BallRadius div 2 - 3, x - BallRadius div 2 + 3, y - BallRadius
div 2 + 3);
      Pen.Color := clBlack;
      Brush.Color := clOlive;
      Rectangle(x - 5, y + BallRadius + 5, x + 5, y + BallRadius + 10);
      MoveTo(x - 5, y + BallRadius + 5);
      LineTo(x - BallRadius, y);
      MoveTo(x, y + BallRadius + 5);
      LineTo(x, y);
    end
  end

```

```

    MoveTo(x + 5, y + BallRadius + 5);
    LineTo(x + BallRadius, y);
end;
end;

procedure DrawGun;
begin
    with Form1.Image1.Canvas do
        begin
            if (Vx = 0) and (Vy = 0) then
                Pen.Color := RGB(0, 70, 0) else Pen.Color := clBlack;
                Brush.Color := Pen.Color;
                Pen.Width := 5;
                MoveTo(GunPosition, 240);
                LineTo(GunPosition + round(15 * sin(Angle * pi / 180)), 240 - round(15 * cos(Angle * pi / 180)));
                Pen.Width := 1;
                Ellipse(GunPosition - 8, 232, GunPosition + 8, 248);
                Rectangle(GunPosition - 10, 240, GunPosition + 10, 260);
            end;
        end;

```

Процедура DrawExplosion обеспечивает доступ к координатам снаряда.

```

procedure DrawExplosion(x, y, Phase: integer);
var
    i, xx, yy, Size: integer;
    a, b: double;
begin
    with Form1.Image1.Canvas do
        for i := 0 to Phase * 10 do
            begin
                a := random * 2 * pi;
                b := random * sqr(Phase) / 3 + 5;
                xx := x + round(1 * sin(a));
                yy := y + round(1 * cos(a));
                Size := round(sqr(10 - Phase) / 8 + 2);
                Pen.Color := RGB(random(100), 0, 0);
                Brush.Color := Pen.Color;
                Ellipse(xx - random(Size) - 1, yy - random(Size) - 1, xx + random(Size), yy + random(Size));
            end;
        end;

```

Кроме того, пушка меняет цвет, когда готов очередной снаряд. В довершение, нужно соединить все написанное воедино.

Процедуры рисования объединяются в процедуру, рисующей все поле боя:

- небо,
- шары,
- пушка,
- ядро,
- бомба.

Объединять все вместе обработчик таймера.

```

procedure DrawBattleField;
var
i: integer;
begin
  with Form1.Image1.Canvas do
    for i := 0 to 259 do
      begin
        Pen.Color := RGB(i div 2, i div 2, 255);
        MoveTo(0, i);
        LineTo(440, i);
      end;
      for i := 0 to BallCount - 1 do
        with Balloons[i] do
          if Explosion = 0 then DrawBalloon(x, 260 - y, Color) else if Explosion < 10
then DrawExplosion(x, 260 - y, Explosion);
          if GunExplosion = 0 then DrawGun else if GunExplosion < 10 then DrawExplosion(GunPosition,
240, GunExplosion);
          with Form1.Image1.Canvas do
            begin
              Pen.Color := clMaroon;
              Brush.Color := clRed;
              if (Vx <> 0) or (Vy <> 0) then Ellipse(round(x) - 2, 260 - round(y) - 2, round(x) + 3, 260 -
round(y) + 3);
              if (BombY <> 1000) then Ellipse(GunPosition - 2, 260 - round(BombY) - 2, GunPosition + 3,
260 - round(BombY) + 3);
            end;
          end;
end;

procedure TForm1.Timer1Timer(Sender: TObject);
begin
  MoveAll;
  CheckCollisions;
  DrawBattleField;
end;

```

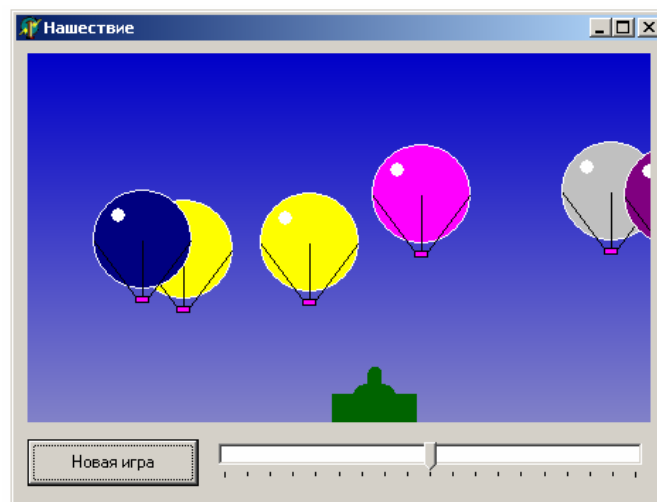


Рис. 64.

## Практическая работа № 35

### «Тест»

**Цель работы** - создать программу, которая тестирует учащегося по информатике и математике.

Проект должен содержать последовательность форм, реализующих диалог с тестируемым учащимся.

На первой форме происходит регистрация учащегося.

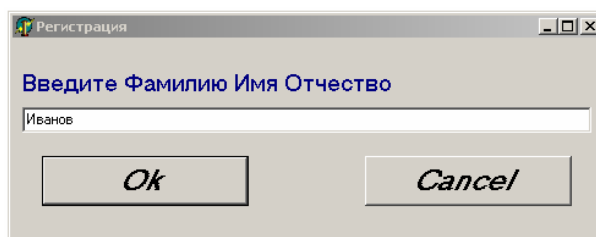


Рис. 65.

Фрагмент программы (unit1):

```
uses Unit2;  
{ $R *.dfm }
```

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

```
procedure TForm1.Button1Click(Sender: TObject);  
begin  
  Form2.Label3.Caption:=Form1.Edit1.Text;  
  Form2.ShowModal;  
end;
```

На второй форме предлагается выбрать один из тестов.

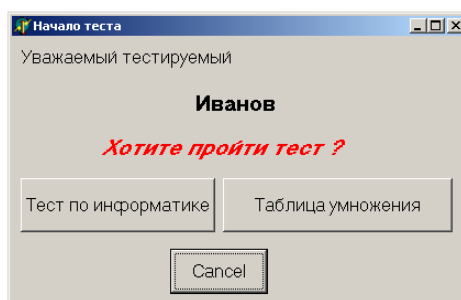


Рис. 66.

Фрагмент программы (unit2):

```
uses Unit3, Unit6;  
{ $R *.dfm }
```

```
procedure TForm2.Button1Click(Sender: TObject);  
begin  
  Form3.ShowModal;  
end;
```

```

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

```

```

procedure TForm2.Button3Click(Sender: TObject);
begin
  Form6.Edit2.Text:="";
  Form6.ShowModal;
end;

```

На третьей форме предлагается проути тест по информатике.

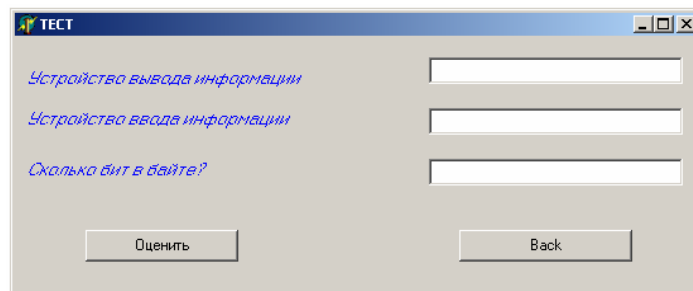


Рис. 67.

Фрагмент программы (unit3):

```

uses Unit4, Unit2;
{$R *.dfm}

```

```

procedure TForm3.Button1Click(Sender: TObject);
begin
  k:=0;
  if (Form3.Edit1.Text='монитор') or (Form3.Edit1.Text='Монитор') or
    (Form3.Edit1.Text='МОНИТОР') then k:=k+1;
  if (Form3.Edit2.Text='Клавиатура') or (Form3.Edit2.Text='клавиатура')
    or (Form3.Edit2.Text='КЛАВИАТУРА') then k:=k+1;
  if Form3.Edit3.Text='8' then k:=k+1;
  Form4.Label2.Caption:=IntToStr(k);
  if k=0 then Form4.Label1.Caption:='Очень плохо' else
    if k=1 then Form4.Label1.Caption:='Плохо' else
      if k=2 then Form4.Label1.Caption:='Хорошо' else
        if k=3 then Form4.Label1.Caption:='Очень хорошо';

  Form3.Edit1.Text:="";
  Form3.Edit2.Text:="";
  Form3.Edit3.Text:="";
  Form4.ShowModal;
end;

```

```

procedure TForm3.Button2Click(Sender: TObject);
begin
  Form3.Close;
end;

```

На следующей форме отображается результат тестирования и предложение о просмотре ответа.



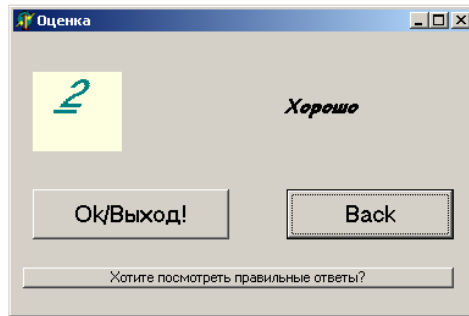


Рис. 68.

Фрагмент программы (unit4):

```
uses Unit1, Unit5, Unit3, Unit2;
{$R *.dfm}
```

```
procedure TForm4.Button2Click(Sender: TObject);
begin
  Form4.Close;
end;
```

```
procedure TForm4.Button1Click(Sender: TObject);
begin
  Form4.Close;
  Form3.Close;
end;
```

```
procedure TForm4.Button3Click(Sender: TObject);
begin
  Form5.ShowModal;
end;
```

На следующей форме отображаются правильные ответы.

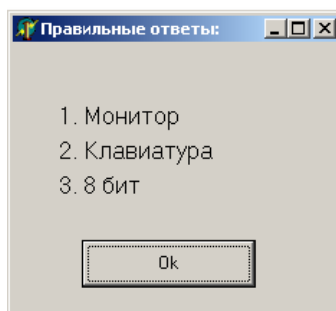


Рис. 69.

Фрагмент программы (unit5):

```
uses Unit4, Unit3, Unit2, Unit1;
{$R *.dfm}
```

```
procedure TForm5.Button1Click(Sender: TObject);
begin
  Form4.Close;
  Form3.Close;
  Form2.Close;
  Form1.Close;
```

```
Form5.Close;
end;
```

Если учащийся выбрал тест по математике, то ему предлагается проверить свои знания таблицы умножения. На следующей форме случайным образом выбираются числа. Учащийся должен ввести значение произведения в текстовое поле. С помощью кнопки «проверка» выясняется правильность введенного ответа. Если ответ правильный, то можно сгенерировать следующий пример. После нескольких примеров можно проверить свой рейтинг.

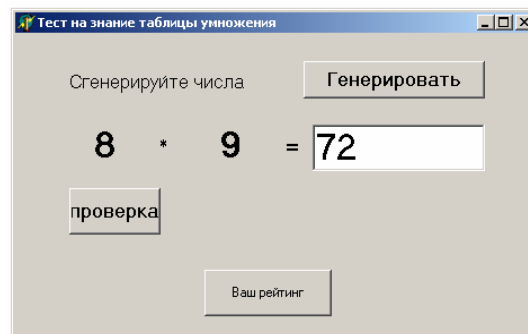


Рис. 70.

Фрагмент программы (unit6):

```
uses Unit7, Unit1;
{$R *.dfm}

procedure TForm6.Button1Click(Sender: TObject);
var n,i:integer;
begin
  randomize;
  a:=random(10)-0;
  b:=random(10)-0;
  Form6.Label1.Caption:=IntToStr(a);
  Form6.Label2.Caption:=IntToStr(b);
  Form6.Edit2.Text:='';
  Form6.Label6.Caption:='';
end;

procedure TForm6.Button2Click(Sender: TObject);
begin
  if (a*b=StrToInt(Form6.Edit2.Text)) then
  begin
    Form6.Label6.Caption:='Правильно';
    m:=m+1;
    r:=r+1;
    q:=q+1;
  end
  else
  begin
    Form6.Label6.Caption:='Не правильно';
    r:=r-1;
    q:=q+1;
  end;
  Form6.Label1.Caption:='';
  Form6.Label2.Caption:='';
```

**end;**

**procedure** TForm6.Button3Click(Sender: TObject);

**var**

c:real;

**begin**

Form7.Label2.Caption:=Form1.Edit1.Text;

Form7.Label1.Caption:='Вы ответили на '+IntToStr(q)+' вопросов, из них правильно '+IntToStr(m);

Form7.Label4.Caption:='Ваш рейтинг = '+IntToStr(r);

c:=m/q;

**if** c=0 **then** Form7.Label3.Caption:='Очень плохо' **else**

**if** (c>0)and(c<0.5) **then** Form7.Label3.Caption:='Плохо' **else**

**if** c=0.5 **then** Form7.Label3.Caption:='Надо доучить' **else**

**if** (c>0.5) **and** (c<1) **then** Form7.Label3.Caption:='Хорошо' **else**

**if** c=1 **then** Form7.Label3.Caption:='Молодец!';

Form7.ShowModal;

**end;**

При нажатии на кнопке «Ваш рейтинг» на следующей форме появляется результаты рейтинга. Тестирование можно завершить.

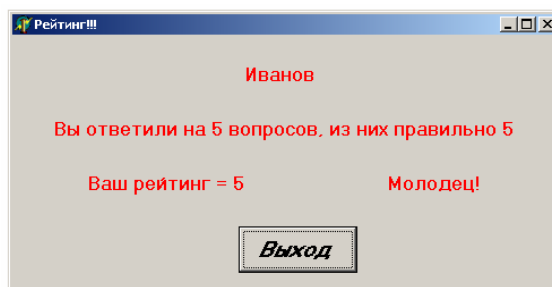


Рис. 71.

Фрагмент программы (unit7):

**uses** Unit6;

{SR \*.dfm}

**procedure** TForm7.Button1Click(Sender: TObject);

**begin**

Form7.Close;

Form6.Close;

**end;**

Обратите внимание на подключение модулей в строке Uses. Таким образом, происходит обращение к соответствующей форме. Для отображения формы используется функция

**function** ShowModal: Integer;

Данная функция позволяет показывать форму в работе режима диалога.

## Практическая работа № 36 «Проигрыватель»

**Цель работы** - создать программу - проигрыватель файлов мультимедиа, снабженный системой визуализации. Для начала многооконность будет реализована посредством стандартных диалогов. Проигрыватель файлов мультимедиа, проигрывающий различные форматы, а также сопровождающий музыкальные файлы визуализацией.

1. Создать на диске папку (например C:\MyDelphi\MyMPlayer), в которой будет создан проект, скопировать в эту папку несколько музыкальных файлов и клипов. Для демонстрации полноценной работы проигрывателя понадобятся мультимедийные файлы wav, mid, wma, mp3, avi, которые можно найти в соответствующих папках.

Видеофайлы:

C:\Program Files\Borland\Delphi5\Demos\Coolstuff\

Аудиофайлы:

C:\Windows\Media\

C:\Program Files\Microsoft Office\Office10\Media\

C:\Мои документы\Мои музыкальные записи\

2. Создать простейшую программу проигрывания аудиофайлов (см. практическая работа № 10). Для этого поместить на форму кнопку, медиапроигрыватель, диалог загрузки.

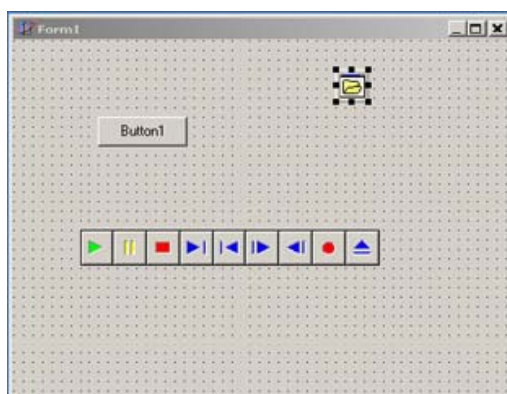


Рис. 72.

3. Настроить фильтр диалога на файлы мультимедиа, соответственно заполнив таблицу. Убрать ненужные нам кнопки проигрывателя, отключив их в Инспекторе Объектов (VisibleButtons). В обработчике нажатия на кнопку описать загрузку файла.

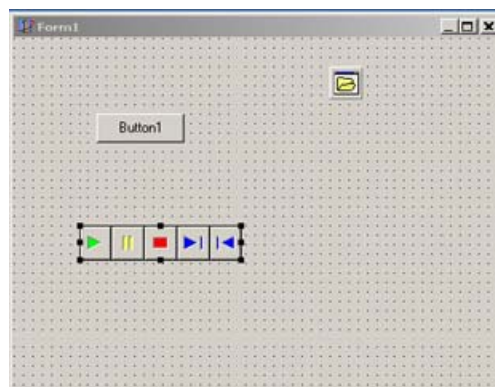
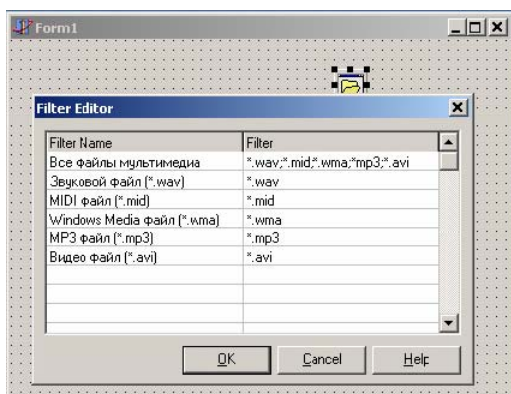


Рис. 73.

**OpenDialog** – диалог загрузки

Основное свойство – **FileName**: string, основной метод – **Execute**. При вызове из программы метода Execute происходит вывод на экран диалога. Параметры вывода

определяются свойствами компонента: Title, Options, Filter и др. Если пользователь выберет файл для открытия и нажмет "Открыть", то в свойство FileName компонента будет записано имя этого файла в формате string, а результатом метода Execute будет значение true. Иначе, если пользователь нажмет в диалоге кнопку "Отмена", результат Execute будет false, т.е. Execute является булевой функцией.

**MediaPlayer** – медиапроигрыватель.

Основное свойство – **FileName**: string, основные методы – **Play, Pause, Stop**. После того как в свойство FileName помещено имя файла мультимедиа, этот файл нужно открыть с помощью метода Open (если установлено свойство AutoOpen, то открытие происходит автоматически, но это не всегда хорошо, поскольку, если FileName не задано или задано неверно, то при автооткрытии произойдет ошибка).

Когда файл открыт, он может быть воспроизведен. Это можно сделать с помощью кнопок самого компонента, либо вызвав метод Play. То же касается остановки и паузы – методов Stop и Pause. Для настройки внешнего вида компонента используется множество свойств, к примеру, можно скрыть часть кнопок, настраивая свойство VisibleButtons.

4. В обработчике нажатия на Button1 вызывается метод диалога Execute, и если он выдает результат true (пользователь выбрал файл и нажал "Открыть"), то медиапроигрывателю передается имя этого файла и файл открывается. Теперь пользователь может воспроизвести его, нажав кнопки проигрывателя.

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  if OpenFileDialog1.Execute then
  begin
    MediaPlayer1.FileName := OpenFileDialog1.FileName;
    MediaPlayer1.Open;
  end;
end;
```

5. При запуске программы пользователь видит кнопку и проигрыватель. Проигрыватель не является активным, поскольку файл, который он должен воспроизводить, не открыт.

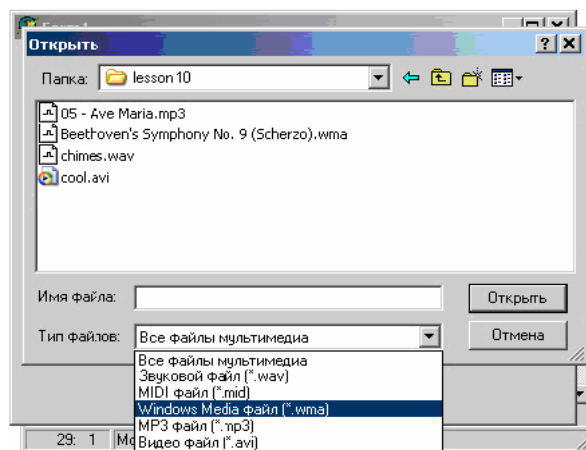
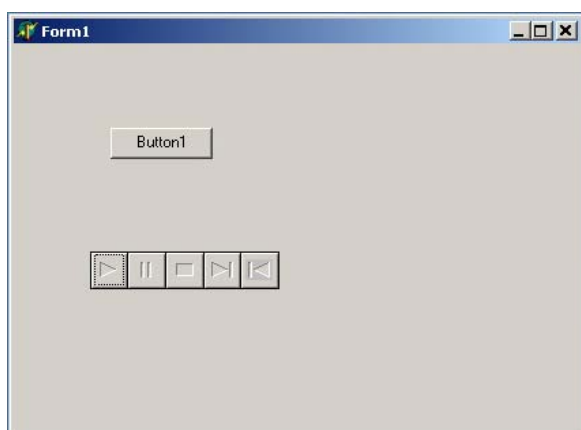


Рис. 74.

При нажатии на кнопку активизируется диалог. Параметры диалога соответствуют значениям свойств, заданных предварительно в Инспекторе Объектов. К примеру, выбранный фильтр звуковых файлов скрывает все файлы с отличающимся расширением.

После выбора в музыкальной папке файла при нажатии кнопки "Открыть" проигрыватель стал доступным. Файл открыт, и можно воспроизвести его.

Если необходимо воспроизвести видеофайл, то при запуске автоматически будет создано отдельное окно, в котором отобразится изображение.

6. Проигрыватель готов. Необходимо оформить пользовательский интерфейс. Поместите на форму панель, а на ней расположите компонент Image, растянутый на всю панель. Добавьте таймер. Сохраните модуль как MainUnit и проект как MyMPlayer в подготовленную папку.

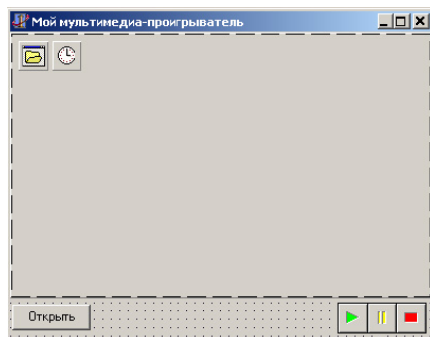


Рис. 75.

Дайте каждому компоненту подходящее название. Обработчик кнопки поменяйте согласно последним изменениям и добавьте растяжение видеоэкрана. Для того чтобы видеофайлы воспроизводились не в отдельном окне, а в заданном месте, например, на Panel1, необходимо соответствующим образом установить свойство медиапроигрывателя Display. Для того чтобы установить и растянуть нужным образом изображение, нужно установить свойство DisplayRect:

```
MPlayer.DisplayRect := Rect(0, 0, ScreenPanel.Width, ScreenPanel.Height);
```

где Rect(x1, y1, x2, y2: integer):TRect – функция, преобразующая четыре числа к формату TRect – прямоугольник, а свойство DisplayRect имеет именно этот формат.

Для того чтобы переименовать компоненты измените свойство Name. Настройте параметры компонентов:

- 1) MainForm: TmainForm  
Caption = 'Мультимедиа-проигрыватель'  
ClientWidth = 402  
ClientHeight = 290
- 2) LifeTimer: Ttimer  
Enabled = false  
Interval = 100
- 3) MediaOpenDlg: TOpenDialog  
Options: ofFileExist = true  
Title = 'Загрузить файл мультимедиа'
- 4) LifeImage: TImage  
Align = alClient
- 5) ScreenPanel: TPanel  
Align = alTop  
Caption = ''  
Width = 402  
Height = 252
- 6) MediaOpenBtn: TButton  
Caption = 'Открыть'
- 7) MPlayer: TMediaPlayer  
Display = ScreenPanel  
VisibleButtons = [btPlay, btPause, btStop]

7. Необходимо создать единую форму для воспроизведения звуковых и видеофайлов. Причем воспроизведение звуковых файлов тоже будет сопровождаться анимацией. В качестве этого сопровождения используется игра "Жизнь".

Для этого добавим соответствующий "рассчитывающий" и "рисующий" код.

Опишем константы, определяющие размер поля, введем новый тип – массив клеток и объявим глобальную переменную этого типа.

```
const
  XSize = 40;
  YSize = 25;
type
  TLifeCells = array [0 .. XSize - 1, 0 .. YSize - 1] of boolean;
var
  MainForm: TMainForm;
  A: TLifeCells;
```

Создадим процедуру, случайным образом заполняющую массив.

```
procedure RandomCells;
var
  i, j: integer;
begin
  for i := 0 to XSize - 1 do
    for j := 0 to YSize - 1 do
      A[i, j] := random < 0.5;
end;
```

Это процедура случайного заполнения поля. Булева переменная присваивается логически, а не с помощью условного оператора `if random < 0.5 then A[i,j] := true else A[i,j] := false`.

Создадим функцию, определяющую число живых клеток-соседей у данной клетки.

```
function NumOfCells(x, y: integer): integer;
var
  i, j, xx, yy: integer;
begin
  Result := 0;
  for i := -1 to 1 do
    for j := -1 to 1 do
      begin
        xx := x + i; yy := y + j;
        if xx = -1 then xx := XSize - 1;
        if yy = -1 then yy := YSize - 1;
        if xx = XSize then xx := 0;
        if yy = YSize then yy := 0;
        if A[xx, yy] then inc(Result);
      end;
      if A[x, y] then dec(Result);
end;
```

Это функция подсчета живых соседей. Обратите внимание на переменные `xx` и `yy`. Подобным образом реализуется "сшивки" верхнего и нижнего, левого и правого краев. В конце исключается сама клетка из числа соседей: `if A[x, y] then dec(Result)`;

Создадим процедуру, рисующую поле клеток.

```
procedure DrawCells;
var
  i, j: integer;
```

```

begin
  with MainForm.LifeImage.Canvas do
    begin
      Brush.Color := clBlue;
      Pen.Color := Brush.Color;
      Rectangle(0, 0, XSize * 10, YSize * 10);
      Brush.Color := clRed;
      for i := 0 to XSize - 1 do
        for j := 0 to YSize - 1 do
          if A[i, j] then Rectangle(i * 10, j * 10, i * 10 + 10, j * 10 + 10);
        end;
      end;
    end;
  end;

```

Опишем обработчик таймера; каждый такт у нас будет происходить перерасчет живых клеток по правилам игры.

```

procedure TMainForm.LifeTimerTimer(Sender: TObject);
var
  i, j: integer;
  B: TLifeCells;
begin
  for i := 0 to XSize - 1 do
    for j := 0 to YSize - 1 do
      case NumOfCells(i, j) of
        2: B[i, j] := A[i, j];
        3: B[i, j] := true;
      else B[i, j] := false;
    end;
  A := B;
  DrawCells;
end;

```

Каждый такт таймера пересчитываются живые клетки. Алгоритм игры, как известно, состоит в следующем. Если у живой клетки два или три соседа, то она остается живой, если меньше двух, – погибает от одиночества, если больше трех, – гибнет от тесноты. Рождается клетка, если у нее ровно три соседа. С помощью оператора case реализуются правила. Использование локального массива B необходимо. Если записывать сразу в A, то клетки нового поколения будут также считаться при учете соседей, что недопустимо: каждому – свое. В конце, однако, нужно скопировать новое поколение, содержащееся в B, в массив A. Простые массивы можно копировать присвоением: A := B.

Модифицируется обработчик нажатия кнопки под новые задачи. Проанализируйте расширение загружаемого файла, и, если загружен видеофайл, Image становится невидимым, освобождая экран для ролика, в противном случае, наоборот, выводится картинку, на которой идет игра, заполняется поле клеток и запускается таймер.

```

procedure TMainForm.MediaOpenBtnClick(Sender: TObject);
begin
  if MediaOpenDlg.Execute then
    begin
      MPlayer.FileName := MediaOpenDlg.FileName;
      MPlayer.Open;
      MPlayer.DisplayRect := Rect(0, 0, ScreenPanel.Width, ScreenPanel.Height);
      MPlayer.Play;
      if ExtractFileExt(MPlayer.FileName) = '.avi' then LifeImage.Visible := false
      else begin
        LifeImage.Visible := true;
      end;
    end;

```



```

RandomCells;
LifeTimer.Enabled := true;
end;
end;
end;

```

Экран вывода "растягивается" на всю панель. Добавлен автоматический запуск воспроизведения. В зависимости от расширения воспроизводимого файла, LifeImage, на котором происходит "Жизнь", либо прячется (в случае '.avi'), либо выводится (в других случаях).

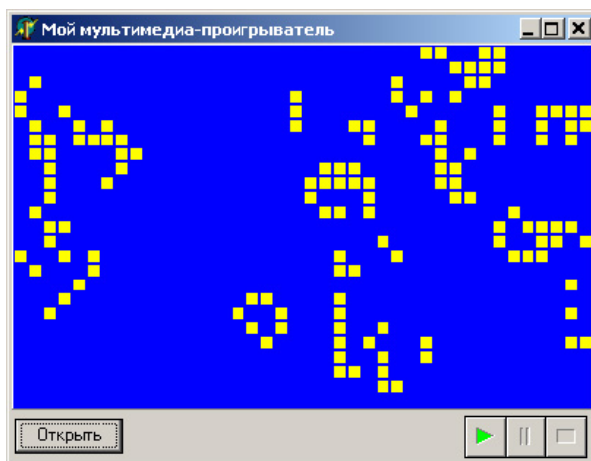


Рис. 76.

8. Необходимо предоставить пользователю возможность настройки цветов. На форму добавляем три компонента – два диалога выбора цвета FieldColorDlg: TColorDialog (Color = clNavy), CellColorDlg: TColorDialog (Color = clYellow) и одну кнопку SetColorBtn: TButton (Caption = 'Цвета').

В обработчике нажатия на эту кнопку нужно описать вызов диалогов.

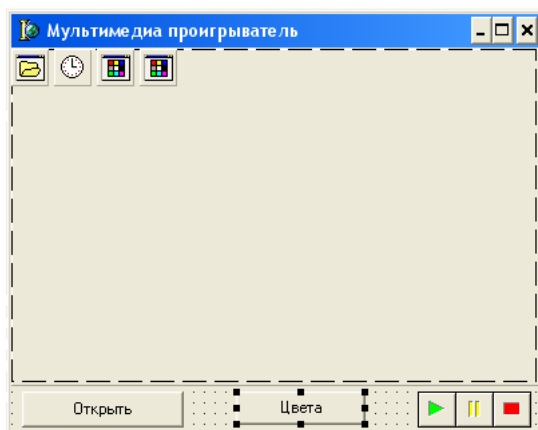


Рис. 77.

**ColorDialog** (основное свойство **Color**: TColor, основной метод – **Execute**) - диалог выбора цвета. Если пользователь выбрал цвет и нажал "ОК", результат этого метода – true, если пользователь нажал "Отмена", то результат false.

**procedure** TMainForm.SetColorBtnClick(Sender: TObject);

**begin**

if MessageDlg('Изменить цвета?', mtConfirmation, [mbYes, mbNo], 0) = mrNo **then** exit;

if MainForm.FieldColorDlg.Execute **then** ScreenPanel.Color := MainForm.FieldColorDlg.Color;

MessageDlg('Теперь измените цвет клеток', mtInformation, [mbOk], 0);

CellColorDlg.Execute;

```
DrawCells;
end;
```

**MessageDlg**(const Msg: string; DlgType: TMsgDlgType; Buttons: TMsgDlgButtons; HelpCtx: Longint): Word; - вызов окна сообщения. Это служебное окно.

Msg – текст сообщения.

DlgType – тип диалога. Возможные значения: mtWarning, mtError, mtInformation, mtConfirmation, mtCustom.

Buttons – используемые кнопки. Переменная типа "множество", возможные элементы: mbYes, mbNo, mbOK, mbCancel, mbAbort, mbRetry, mbIgnore, mbAll, mnNoToAll, mbYesToAll, mbHelp.

HelpCtx – индекс контекстной справки. В данном примере индекс равен нулю. При написании проектов с развернутой справочной системой указывается реальный индекс.

MessageDlg возвращает число, равное одной из следующих констант: mrNone, mrAbort, mrYes, mrOk, mrRetry, mrNo, mrCancel, mrIgnore, mrAll. Эти константы – так называемый модальный результат (modal result) нажатия на соответствующую кнопку диалога.

В данной программе проверяется, равен ли модальный результат mrNo, т.е. нажата ли кнопка "Нет"; если нажата, то вызывается exit, и программа немедленно выходит из текущей процедуры-обработчика.

Функции MessageDlg и Execute Delphi позволяет вызывать как процедуры:

MediaOpenDlg.Execute вместо if MediaOpenDlg.Execute then ...

При этом невозможно проконтролировать, какие кнопки были нажаты: "ОК" или "Отмена".

Далее необходимо откорректировать процедуру рисования, затем запустить программу, загрузить музыкальный файл и настройте цвета. Наконец, необходимо использовать цвета диалогов при рисовании.

Внесите изменения в DrawCells:

```
...
with MainForm.LifeImage.Canvas do
begin
Brush.Color := MainForm.FieldColorDlg.Color;
Pen.Color := Brush.Color;
Rectangle(0, 0, XSize * 10, YSize * 10);
Brush.Color := CellColorDlg.Color;
Brush.Color := MainForm.CellColorDlg.Color;
.....
```

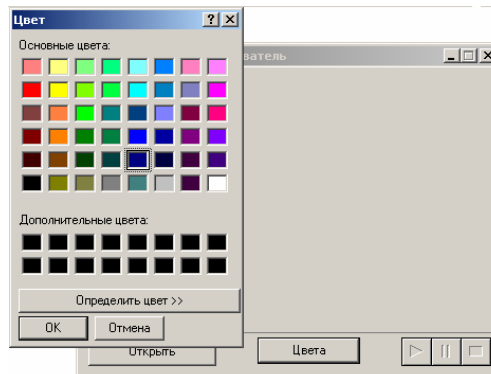
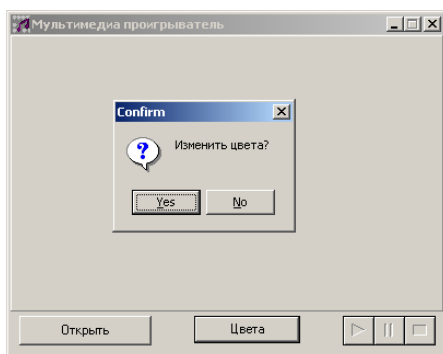


Рис. 78.

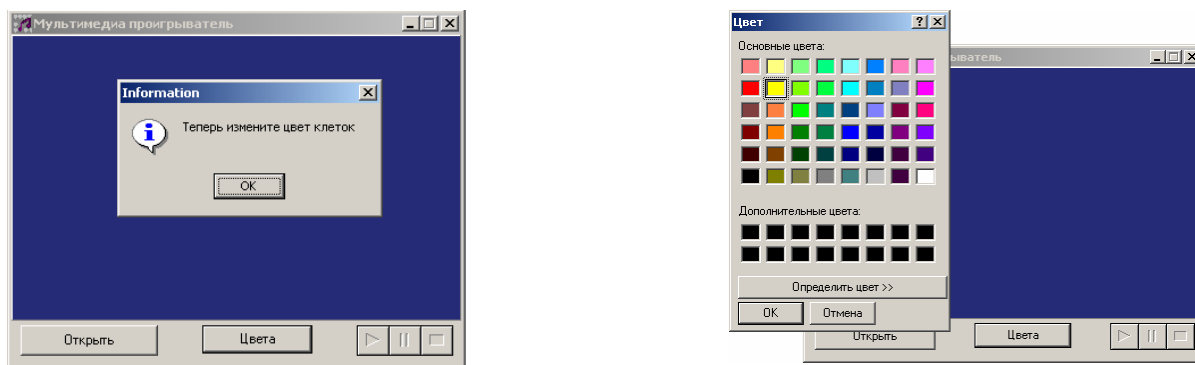


Рис. 79.

9. На данный момент форма может быть масштабирована во время работы приложения. Для того чтобы зафиксировать размер формы, необходимо установить соответствующее свойство `BorderStyle`. Кроме того, нужно запретить разворачивание формы на весь экран (это также нежелательно).

Перечислим основные свойства и события формы.

### **Свойства**

**BorderIcons:** `TBorderIcons` – иконки заглавной панели формы, тип свойства – множество, возможные элементы: `biSystemMenu`, `biMinimize`, `biMaximize`, `biHelp` (в данном случае нужно отключить `biMaximize`, чтобы форму нельзя было развернуть).

**BorderStyle:** `TBorderStyle` – стиль бордюра `bsDialog`, `bsSingle`, `bsNone`, `bsSizeable`, `bsToolWindow`, `bsSizeToolWin`. Если установить `bsNone`, то форма будет без заглавной панели, а значит и без кнопки закрытия программы. Следовательно, выходить из программы возможно либо с помощью `Alt-F4`, либо предусмотреть дополнительную кнопку с методом `Close`.

**ClientWidth, ClientHeight:** `integer` – ширина и высота клиентской области формы.

**FormStyle:** `TFormStyle` – стиль формы. Наиболее важным значением является `fsStayOnTop`. Если установлено это значение, то форма будет находиться над остальными формами, даже если она не активна, и пересекается с активной на данный момент формой.

**Position:** `TPosition` – расположение формы. Здесь задаются различные начальные положения формы, например, задать положение по центру экрана (`poScreenCenter`). При запуске программы она окажется в центре.

**WindowState:** `TWindowState` – состояние окна. Возможные значения: `wsNormal`, `wsMinimized`, `wsMaximized`. С помощью этого свойства можно разворачивать и сворачивать форму.

### **События**

**OnActivate, OnDeactivate(Sender: TObject)** – события, возникающие, когда форма активируется и деактивируется (при этом в форму переходит или ее покидает фокус ввода).

**OnCloseQuery(Sender: TObject; var CanClose: Boolean)** – событие, возникающее при попытке закрыть форму. Например, некоторая программа спрашивает: "Вы хотите завершить программу?". Это – результат действия подобного события. Делается это в обработчике подобным образом: `CanClose := MessageDlg('Exit now?', mtConfirmation, [mbYes, mbNo], 0) = mrYes`.

**OnCreate, OnDestroy(Sender: TObject)** – событие, необходимое для описания действий, требуемых в начале или в конце работы приложения (открытие и закрытие файлов, чтение и сохранение настроек, инициализация переменных и т.п.).

**OnHide, OnShow(Sender: TObject)** – вызываются, когда форма становится видимой или невидимой (при присвоении соответствующего значения свойства `Visible`).

**OnResize(Sender: TObject)** – происходит, когда форма растягивается или сжимается. Здесь описывают изменение размеров и расположение компонентов на форме.

10. Продолжить оформление медиапроигрывателя.

1) Для того чтобы случайное заполнение было более случайным, в событии формы OnCreate добавить Randomize.

```
procedure TMainForm.FormCreate(Sender: TObject);  
begin  
    Randomize;  
    RandomCells;  
end;
```

2) Также желательно зациклить воспроизведение медиафайла, описать обработчик события медиапроигрывателя OnNotify. У MediaPlayer есть два свойства – Position: LongInt и Length: LongInt. Это, соответственно, текущая позиция в записи и длина записи. Событие проигрывателя OnNotify возникает при завершении различных управляющих методов MediaPlayer. Состояние проигрывателя можно узнать из свойства Mode.

Когда запись будет доигрываться до конца, будет вызываться метод Play, запускающий ее с начала. При остановке проигрывателя «перематываем» запись на начало.

if Position = Length then Play

```
procedure TMainForm.MPlayerNotify(Sender: TObject);  
begin  
    with MPlayer do case Mode of mpPlaying: if Position = Length then Play;  
    mpStopped: Rewind;  
    end;  
end;
```

3) Установить название приложения и выбрать ему иконку. В главном меню открыть Project => Options. На вкладке Application, задать название приложения (это название, в частности, будет отображаться снизу на панели задач Windows), а также загрузить подходящую иконку. Стандартную иконку можно найти в папке C:\Program Files\Common Files\Borland Shared\Images\Icons, создать свою иконку – в редакторе Delphi (меню Tools => Image Editor) или в любом другом редакторе ресурсов.



Рис. 80.

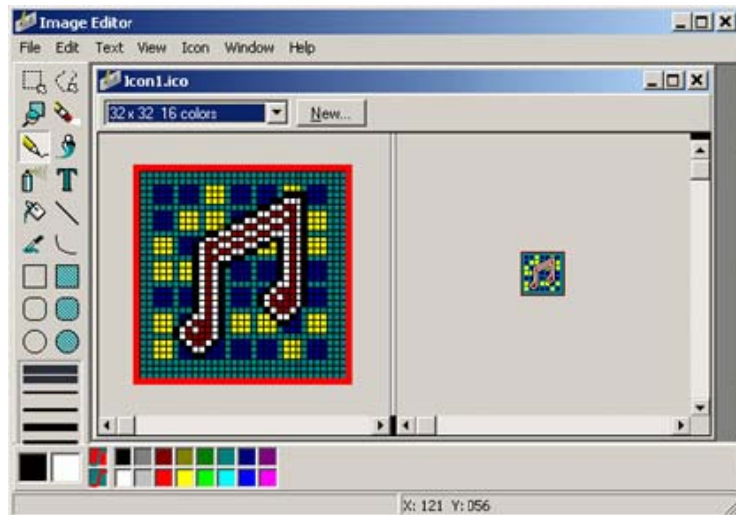


Рис. 81.

11. Проанализируем недостатки: не выводится название проигрываемого файла, компоненты на форме не очень-то гармонируют друг с другом, неизвестно, сколько процентов записи проиграно на данный момент, нельзя устанавливать скорость визуализации, нет возможности редактирования и сохранения позиций игры "Жизнь".

Создайте копию проекта. Добавить в проект новую форму для управления воспроизведением медиафайла. Для создания новой формы щелкните на кнопку меню. Есть возможность сделать это и в пункте меню File => New... Там же можно создать форму с помощью мастера.

Добавьте на форму пять кнопок **SpeedButton**, **TrackBar** и таймер.

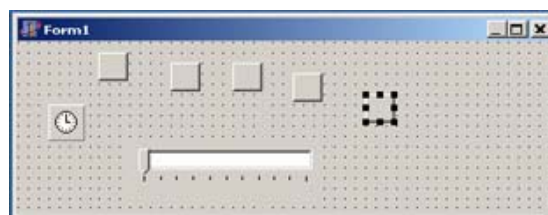


Рис. 82.

Кнопка **SpeedButton** отличается от обычной кнопки, во-первых, тем, что может содержать картинку, а во-вторых, тем, что может оставаться нажатой.



Рис. 83.

Основные свойства – **Glyph:TPicture** (картинка), **Down:boolean** (нажата), основное событие – **OnClick**. У этой кнопки есть еще несколько важных свойств. Для того чтобы кнопка могла быть нажата, ее свойство **Group: integer** должно быть больше нуля. В этом случае все кнопки с одинаковым значением группы становятся переключателями – если нажать одну, остальные отщелкиваются. Это может быть нежелательно в случае одной кнопки с ненулевой группой – один раз нажав, ее нельзя отключить. В этом случае поможет свойство **AllowAllUp: boolean**. Современный вид кнопкам придает свойство **Flat: boolean**.

В программе откажемся от стандартного интерфейса **MediaPlayer**, и, сделав его невидимым, будем обращаться к нему через созданные кнопки. Коллекция иконок для кнопок находится в папке C:\Program Files\Common Files\Borland Shared\Images\Buttons\.

Настройка **TrackBar** проводится с помощью свойств **ThumbLength**: integer (длина бегунка), **TickMarks**: TTickMark (положение рисок линейки), **TickStyle**: TTickStyle (стиль рисок).

Настройте форму и компоненты. Измените имена компонент.

1) ControlForm: TControlForm

Caption = 'медиафайл не загружен'

BorderStyle = bsToolWindow

ClientHeight = 24

ClientWidth = 402

2) NewSpBtn: TSpeedButton

AllowAllUp = true

Flat = true

Glyph: rety.bmp

GroupIndex = 1

3) PosTrk: TTrackBar

ThumbLength = 15

TickMarks = tmBoth

TickStyle = lsNone

4) PosTimer = TTimer

Enabled = false

Interval = 500

5) MediaOpenSpBtn: TSpeedButton

Flat = true

Glyph: MdOpen.bmp

6) PlaySpBtn: TSpeedButton

AllowAllUp = true

Flat = true

Glyph: vcrplay.bmp

GroupIndex = 1

7) PlaySpBtn: TSpeedButton

AllowAllUp = true

Flat = true

Glyph: vcrpause.bmp

GroupIndex = 1

8) StopSpBtn: TSpeedButton

AllowAllUp = true

Flat = true

Glyph: vcrstop.bmp

GroupIndex = 1

12. Когда требуется обратиться к каким-то переменным, функциям, типам, объектам, описанным в другом модуле, необходимо описать ссылку на этот модуль в разделе uses. Однако тонкость состоит в том, что uses может располагаться и в разделе interface, и в разделе implementation. При запуске приложения новое окно должно находиться под главным окном. Но если описать ссылки двух модулей друг на друга в разделах interface, возникнет так называемая ошибка круговой ссылки. В рамках Pascal два модуля не могут использовать интерфейс друг друга. Ссылка в интерфейсе нужна для того, чтобы при описании типов, классов, переменных интерфейса использовать типы и константы, описанные в других модулях. В данном интерфейсе используется, к примеру, класс TForm, TSpeedButton, TTrackBar, TTimer, описанные, соответственно, в модулях Forms, Buttons, ComCtrls, ExtCtrls. То есть использование других модулей в интерфейсе дает возможность, прежде всего, описать новые типы с помощью уже описанных типов. Именно поэтому

запрещена круговая ссылка. Тогда потенциально возможно было бы описать: TypeA = array[0..4] of TypeB – в одном модуле и TypeB = array [0..99] of TypeA – в другом. То есть *ignotum per ignotius* – неизвестное через еще более неизвестное (лат.).

Что же касается использования не типов, но конкретных переменных и объектов, фигурирующих в других формах, то это использование происходит как раз в `implementation`, и тут круговые ссылки вполне допустимы. При работе с несколькими формами важно также учитывать порядок их создания. Это касается обработки событий `OnCreate` – обращаться можно только к созданным формам.

```
Var ControlFom: TControlFom;
```

**implementation**

```
{ $R *.DFM }
```

```
procedure TControlFom.FormCreate (Sender: TObject);
```

```
begin
```

```
    Top := MainForm.Top + MainForm.Height + 4;
```

```
    Left := MainForm.Left;
```

```
end;
```

```
end.
```

Свои свойства `Left` и `Top` вторая форма подстраивает под те же свойства первой формы. При попытке запуска программы, однако, выводится сообщение о том, что не указан модуль, в котором описана первая форма.



Рис. 84.

Если нажать кнопку `Yes`, то в раздел `uses` автоматически добавится ссылка на нужный модуль. Однако, система не всегда автоматически работает. Лучше добавлять ссылки самостоятельно. В данном случае – в раздел `implementation`.

**implementation**

```
    uses MainUnit;
```

Здесь идет обращение к `MainForm`, и данная форма должна быть к этому моменту (то есть моменту создания `ControlForm`) уже созданной.

По умолчанию при создании приложения полагается, чтобы при запуске выводилась только главная форма. Однако в данном случае в программе должны быть видны обе формы. Обязательно, у `ControlForm` установить свойство `Visible := true`. Иначе форма будет не видна при запуске программы.

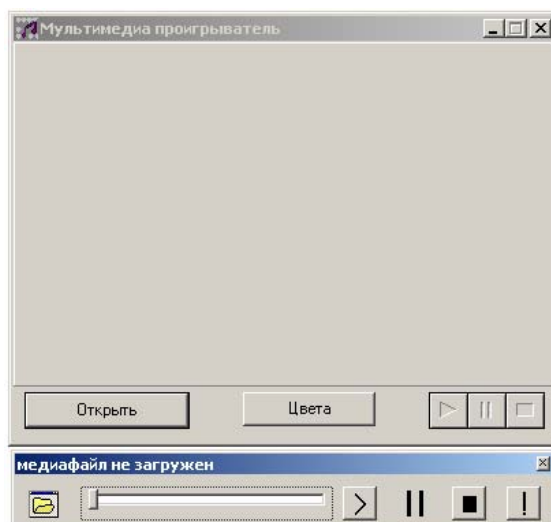


Рис. 85.

Можно описать тот же код, с точностью до перемены ссылок, в обработчике MainForm.OnCreate главной формы. Для того чтобы выбрать нужный модуль, нажмите на кнопку панели инструментов (выбрать модуль для редактирования). Соседняя кнопка служит для выбора форм (выбрать форму для редактирования). Следующая кнопка – переключатель модуль/форма.

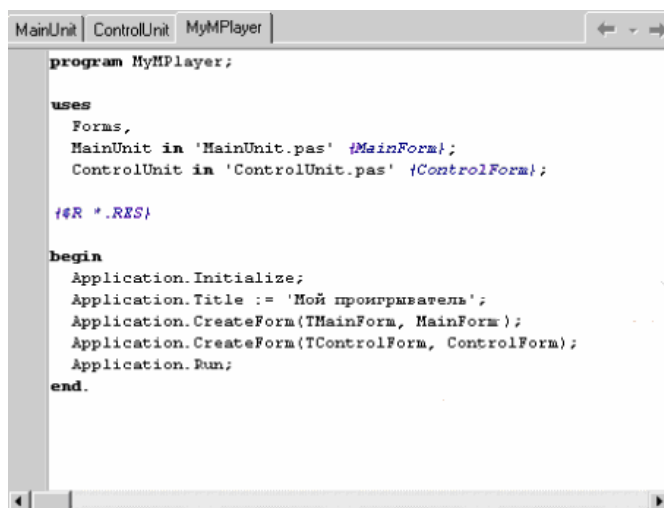


Рис. 86.

Первая форма создается раньше второй. Следовательно, и событие OnCreate у нее возникает еще до того, как создана вторая форма, и значит попытка доступа к ней (попытка установить ее положение) вызовет ошибку.

13. Оставим главной форме только предназначение экрана. Удалите код внутри обработчиков кнопок, а также сами кнопки.

```
procedure TMainForm.MediaOpenBtnClick (Sender : TObject);
begin

end;
procedure TMainForm.SetColorBtnClick (Sender : TObject);
begin

end;
```



Установите медиапроигрывателю Visible := false, переместите его поверх панели. Если панель в форму добавлена позже проигрывателя, то проигрыватель окажется под панелью. Поэтому предварительно поменяйте порядок – щелкните правой кнопкой мыши на проигрыватель и выберите "Bring To Front".

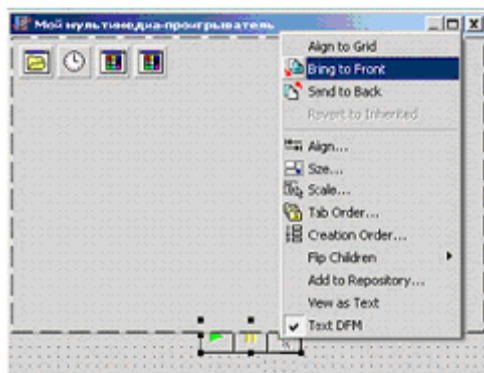


Рис. 87.

Теперь измените клиентскую высоту формы. Экран готов.

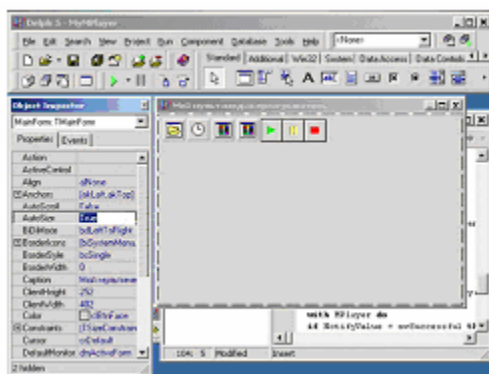


Рис. 88.

Теперь предстоит описать обработчики компонентов второй формы. Таймер будет передвигать бегунок, отображая процент проигранной записи.

```
procedure TControlForm.PosTimerTimer(Sender: TObject);
begin
  PosTrkBar.Position := MainForm.MPlayer.Position;
end;
```

```
procedure TControlForm.PosTrkBarChange(Sender: TObject);
begin
  if not PosTimer.Enabled then
    MainForm.MPlayer.Position := PosTrkBar.Position;
end;
```

В обработчике кнопки загрузки помещается код, похожий на тот, что уже писали. Код относится к компонентам главной формы. Из-за этого необходимо указывать вторую форму, когда обращаются к ее компонентам, хотя и так находятся в обработчике второй формы.

Обратите внимание на два таймера – один запускает визуализацию, а другой двигает бегунок. Передвижения бегунка действенны, только когда воспроизведение остановлено – PosTimer.Enabled := false. Иначе происходит конфликт – таймер меняет позицию бегунка, а

бегунок в ответ меняет позицию проигрывателя. Если необходимо менять позицию во время воспроизведения, придется вводить дополнительные проверки.

Когда запись остановлена, можно поменять ее позицию тем же бегунком. Кнопки управления программно запускают и останавливают проигрыватель. Пока файл не загружен, их следует отключить, иначе при нажатии на них произойдет ошибка. Последняя кнопка определит, проигрывать ли запись в цикле.

```
procedure TControlForm.PlaySpBtnClick(Sender: TObject);  
begin  
    MainForm.MPlayer.Play;  
    PosTimer.Enabled := true;  
end;
```

```
procedure TControlForm.PauseSpBtnClick(Sender: TObject);  
begin  
    MainForm.MPlayer.Pause;  
    PosTimer.Enabled := false;  
end;
```

```
procedure TControlForm.StopSpBtnClick(Sender: TObject);  
begin  
    MainForm.MPlayer.Stop;  
    PosTimer.Enabled := false;  
end;
```

Запуская и останавливая проигрыватель, мы запускаем и останавливаем таймер. Необходимо внести изменения в код в *модуле MainUnit*, событие OnNotify проигрывателя:

```
procedure TMainForm.MPlayerNotify(Sender: TObject);  
begin  
    if ControlForm.RewSpBtn.Down then  
        with MPlayer do  
            if NotifyValue = nvSuccessful then  
                begin  
                    Notify := true;  
                    Play;  
                end;  
    end;  
end;
```

Здесь происходит проверка, зажата ли кнопка "циклического воспроизведения" во второй форме. При открытии файла необходимо включить все кнопки и бегунок (изначально Enabled := false), запустить таймеры, настроить PosTrkBar на длину записи и вывести название воспроизводимого файла в Caption формы:

```
procedure TControlForm.MediaOpenSpBtnClick(Sender: TObject);  
begin  
    with MainForm do  
        if MediaOpenDlg.Execute then  
            begin  
                MPlayer.FileName := MediaOpenDlg.FileName;  
                MPlayer.Open;  
                MPlayer.Display := ScreenPanel;  
            end;
```

```

MPlayer.DisplayRect := Rect(0, 0, ScreenPanel.Width, ScreenPanel.Height);
MPlayer.Play;
LifeImage.Visible := not (ExtractFileExt(MPlayer.FileName) = '.avi');
LifeTimer.Enabled := true;
ControlForm.PosTimer.Enabled := true;
ControlForm.PosTrkBar.Max := MPlayer.Length;
ControlForm.PosTrkBar.Enabled := true;
ControlForm.PlaySpBtn.Enabled := true;
ControlForm.PauseSpBtn.Enabled := true;
ControlForm.StopSpBtn.Enabled := true;
ControlForm.Caption := MediaOpenDlg.FileName;
end;
end;

```

14. Добавить к медиапроигрывателю игру «жизнь». Для управления визуализацией потребуется отдельная форма. С помощью этой формы можно очищать, заполнять и редактировать конфигурации клеток игры «Жизнь». Более того, появляется возможность сохранять интересные комбинации в специальные файлы с введенным нами расширением .lif. Кроме того, можно будет управлять скоростью «жизни» и, что уже реализовывалось, задавать цвет поля и клеток.

Создайте новую форму, сохраните ее модуль, поместите на нее пять кнопок Button, одну SpeedButton, один TrackBar, две надписи Label и диалоги загрузки и сохранения. Измените параметры компонент:

```

LifeSaveDlg: TSaveDialog
DefaultExt = 'lif'
Filter = 'файлы игры «Жизнь» (*.lif)|*.lif'
Title = 'Загрузить позицию игры'
LifeOpenDlg: TOpenDialog
DefaultExt = 'lif'
Filter = 'файлы игры «Жизнь» (*.lif)|*.lif'
Options.ofFileMustExist = true
Title = 'Сохранить текущую позицию игры'

```

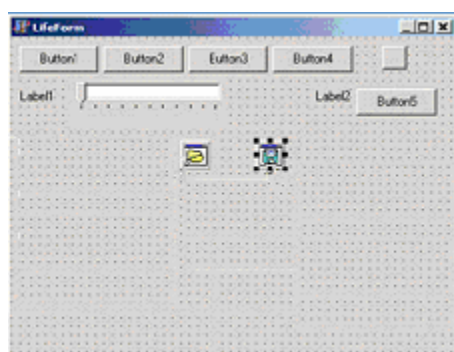


Рис. 89.

В модуле второй формы добавьте строки, при открытии файла выводящие или скрывающие третью форму и отпускающие кнопку.

```

procedure TControlForm.MediaOpenSpBtnClick(Sender: TObject);
begin
  with MainForm do
    if MediaOpenDlg.Execute then

```

**begin**

```
MPlayer.FileName := MediaOpenDlg.FileName;  
MPlayer.Open;  
MPlayer.Display := ScreenPanel;  
MPlayer.DisplayRect := Rect(0, 0, ScreenPanel.Width, ScreenPanel.Height);  
MPlayer.Play;  
LifeImage.Visible := not (ExtractFileExt(MPlayer.FileName) = '.avi');  
LifeForm.Visible := LifeImg.Visible;  
LifeForm.EditSpBtn.Down := false;  
LifeTimer.Enabled := true;  
ControlForm.PosTimer.Enabled := true;  
ControlForm.PosTrkBar.Max := MPlayer.Length;  
ControlForm.PosTrkBar.Enabled := true;  
ControlForm.PlaySpBtn.Enabled := true;  
ControlForm.PauseSpBtn.Enabled := true;  
ControlForm.StopSpBtn.Enabled := true;  
ControlForm.Caption := MediaOpenDlg.FileName;
```

**end;**

**end;**

На новой форме LifeForm располагаются кнопки LifeOpenBtn, LifeSaveBtn, FillBtn, ClearBtn, SetColorBtn, EditSpBtn, SpeedTrkBar. Настройте компоненты новой формы:

```
LifeForm: TLifeForm  
BorderStyle = bsToolWindow  
Caption = 'Визуализация'  
ClientWidth = 402  
ClientHeight = 56  
EditSpBtn: TSpeedButton  
AllowAllUp = true  
Caption = 'редактировать'  
GroupIndex = 1  
SpeedTrkBar: TTrackBar  
Frequency = 50  
Max = 1000  
Min = 1  
Position = 100  
ThumbLength = 15  
TickMarks = tmTopLeft
```



Рис. 90.

В обработчике загрузки медиафайла необходимо добавить следующие строки.

**LifeForm.Visible := LifeImg.Visible;**

Если LifeImg видимый, т.е. нужна визуализация для звукового файла, то форма управления визуализацией тоже выводится, если же LifeImg скрыт (воспроизводится видеофайл), то и присутствие формы управления визуализацией излишне.

**LifeForm.EditSpBtn.Down := false;**

Кнопка EditSpBtn будет отвечать за редактирование клеток (если она нажата, включен режим редактирования). Во время редактирования таймер LifeTimer должен быть отключен.

Поскольку здесь, при открытии, мы включаем этот таймер, то режим редактирования должен быть отключен и кнопка опущена.

```
...
MPlayer.Play;
LifeImg.Visible := not (ExtractFileExt(MPlayer.FileName) = '.avi');
LifeForm.Visible := LifeImage.Visible;
LifeForm.EditSpBtn.Down := false; LifeTimer.Enabled := true;
ControlForm.PosTimer.Enabled := true;
....
```

15. Необходимо описать обработчики событий для новых компонентов. Определим при создании формы ее местоположение.

В модуле LifeUnit:

```
var LifeForm: TLifeForm;
implementation
uses MainUnit, ControlUnit;
procedure TLifeForm.FormCreate(Sender: TObject);
begin
  Top := ControlForm.Top + ControlForm.Height + 4;
  Left := ControlForm.Left;
end;
```

При нажатии на кнопки очистки и заполнения мы используем процедуры, описанные в модуле главной формы.

В модуле LifeUnit:

```
procedure TLifeForm.ClearBtnClick(Sender: TObject);
var
  i, j: integer;
begin
  for i := 0 to XSize - 1 do
    for j := 0 to YSize - 1 do
      A[i, j] := false;
    DrawCells;
end;

procedure TLifeForm.FillBtnClick(Sender: TObject);
begin
  RandomCells;
  DrawCells;
end;
```

Однако для того, чтобы подобный вызов был возможен, необходимо в главном модуле объявить эти процедуры в интерфейсе. Извне доступно только то, что объявлено в интерфейсе. В MainUnit необходимо "вывести" в интерфейс объявление процедур:

```
var
  MainForm: TMainForm;
  A: TLifeCells;
procedure RandomCells;
procedure DrawCells;
```

## implementation

```
uses ControlUnit;  
{&r *.DFM}
```

TrackBar устанавливает интервал для таймера, задавая тем самым скорость визуализации. А кнопка редактирования переключает режим редактирования, останавливая и включая таймер.

```
procedure TLifeForm.FillBtnClick(Sender: TObject);  
begin  
  RandomCells;  
  DrawCells;  
end;
```

```
procedure TLifeForm.SpeedTrkBarChange(Sender: TObject); {Меняет скорость «жизни»}  
begin  
  MainForm.LifeTimer.Interval := SpeedTrkBar.Position;  
end;
```

```
procedure TLifeForm.EditSpBtnClick(Sender: TObject); {включает/выключает режим редактирования}  
begin  
  MainForm.LifeTimer.Enabled := not EditSpBtn.Down;  
end;
```

Само редактирование клеток происходит в главном модуле, в обработчике OnMouseDown компонента LifeImg.

```
procedure TMainForm.LifeImgMouseDown(Sender: TObject; Button: TMouseButton;  
Shift: TShiftState; X, Y: Integer);  
begin  
  if LifeForm.EditSpBtn.Down then  
    A[X div 10, Y div 10] := not A[X div 10, Y div 10];  
    DrawCells;  
end;
```

В uses нужно описать ссылку на LifeUnit.

При запуске программы получается примерно такая картинка.

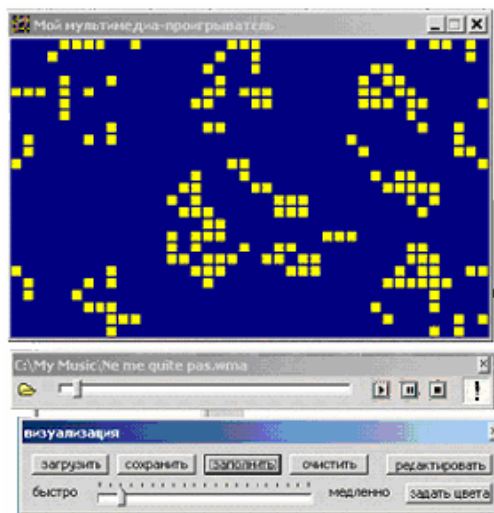


Рис. 91.

Теперь можно редактировать позиции.

16. Вначале при загрузке и сохранении картинок и прочих файлов использовались специальные методы объектов, в которые происходила загрузка. Теперь произведем эти операции вручную, пользуясь процедурами работы с типизированными файлами.

Типизированные файлы содержат записи определенного типа. К примеру, `file of integer` может хранить произвольное количество переменных `integer`; `file of TLifeCells` – произвольное количество переменных типа `TLifeCells`. Однако для наших целей достаточно только одной записи. При работе с типизированными файлами используются следующие процедуры.

**AssignFile**(var F; FileName: string) – связывает файловую переменную F с текстовым именем файла. Это действие необходимо сделать перед всеми последующими операциями.

**Reset**(F) – открывает существующий файл, связанный с переменной F, и устанавливает позицию чтения-записи в начало.

**Rewrite**(F) – создает файл, соответствующий файловой переменной F.

**Read**(F, V) – читает из файла, связанного с F, данные в типизированную переменную V. Позиция файла увеличивается на один файловый компонент, байтовый размер которого равен размеру типа переменной (для `integer` это 4 байта).

**Write**(F, V) – записывает в файл, связанный с F, типизированную переменную V. Позиция файла увеличивается на один файловый компонент.

**Seek**(F, N: LongInt) – перемещает позицию файла к номеру N, начальная позиция при N = 0.

**CloseFile**(var F) – закрывает файл F.

В главном модуле описан тип `TFileCells`:

```
{Private declaration}
public
{Public declaration}
end;

const
    XSize = 40;
    YSize = 25;
type
    TFileCells = array [0 .. XSize - 1, 0 .. YSize - 1] of boolean;
var
    MainForm: TMainForm;
    A: TFileCells;
```

В обработчике `OnClick` кнопки `LifeOpen` определяем файл такого же типа. Если файл выбран в диалоге, ассоциируем имя файла с переменной, устанавливаем позицию для чтения в начало и считываем из файла данные в переменную *A*, описанную в `MainUnit` и хранящую положения клеток.

```
procedure TLifeForm.LifeOpenBtnClick(Sender: TObject);
var
    F: file of TLifeCells;
begin
    if LifeOpenDlg.Execute then
        if FileExists(LifeOpenDlg.FileName) then
            begin
                AssignFile(F, LifeOpenDlg.FileName);
                Reset(F);
                Read(F, A);
```

```

CloseFile(F);
DrawCells;
end;
end;

```

Подобное чтение корректно, поскольку типы файла и переменной *A* совпадают. При записи в файл ситуация похожая, только вдобавок проводится проверка, существует ли файл. Если нет, то он создается; если да, то должны спросить, действительно ли пользователь хочет его перезаписать. Если он просто ошибся, то закрываем файл и выходим из обработчика; если нужно переписать, открываем его и устанавливаем позицию записи в начало. Затем производится запись переменной и закрытие файла.

```

procedure TLifeForm.LifeSaveBtnClick(Sender: TObject);
var
  F: file of TLifeCells;
begin
  if LifeSaveDlg.Execute then
    begin
      AssignFile(F, LifeSaveDlg.FileName);
      if not FileExists(LifeSaveDlg.FileName) then
        Rewrite(F)
      else if MessageDlg('Перезаписать?', mtWarning, [mbYes, mbNo], 0) = mrYes then
        Reset(F)
      else begin
        CloseFile(F);
        exit;
      end;
      Write(F, A);
      CloseFile(F);
    end;
  end;

```

Работа с нетипизированными файлами похожа на работу с типизированными. Отличия в том, что объявляются такие файлы просто `var F: file;` (без `of ...`), при открытии – `Reset(F, 1)` – нужно указать размер блока записи (который по умолчанию равен 128), чтение и запись происходят с помощью процедур **BlockRead**, **BlockWrite** (`var F: File; var Buf; Count: Integer`), где `Buf` – переменная, в которую (или из которой) идет запись, `Count` – число блоков записи (размер блока был определен в `Reset`). Если надо писать произвольные данные (к примеру, `a: integer; b: boolean; c: TLifeCells; d: string[20]`), сделайте следующее.

```

При записи:
Rewrite(F, 1);
BlockWrite(F, a, SizeOf(a));
BlockWrite(F, b, SizeOf(b));
BlockWrite(F, c, SizeOf(c));
BlockWrite(F, d, SizeOf(d));
CloseFile(F);
При чтении, соответственно:
Reset(F, 1);
BlockRead(F, a, SizeOf(a));
BlockRead(F, b, SizeOf(b));
BlockRead(F, c, SizeOf(c));

```



```
BlockRead(F, d, SizeOf(d));
CloseFile(F);
```

17. Чтобы решить проблему цвета, необходимо сделать специальное собственное диалоговое окно, позволяющее выбрать цвет. Раньше мы вызывали стандартные диалоги.

**Диалог** – это обычная форма, которая, однако, вызывается специальным методом ShowModal и возвращает модальный результат. Модальный вызов приводит к тому, что при выведенном диалоге нельзя переключиться ни к одной другой форме приложения, пока не будет нажата одна из модальных кнопок диалога, после чего диалог закрывается и управление вернется в приложение. Модальным кнопкам даже не надо описывать обработчики. Просто надо установить свойство ModalResult не равным mrNone.

Создайте новую форму, сохраните модуль как ColorUnit. Добавьте на нее две фигуры Shape (CellShape: TShape, FieldShape: TShape), кнопку «Утвердить» (ModalResult = mrOk), кнопку «Отмена» (ModalResult = mrCancel).

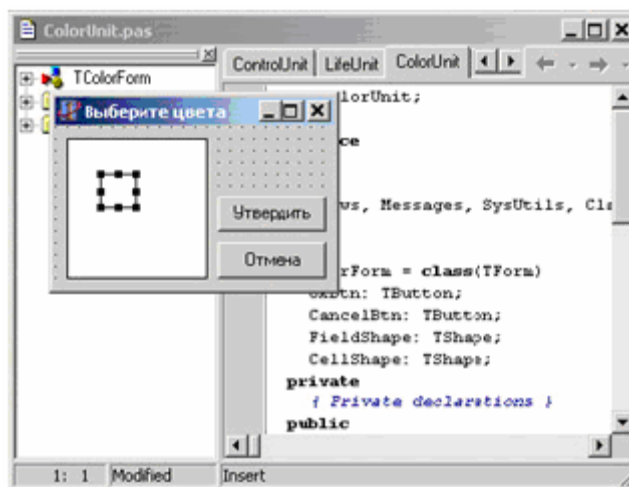


Рис. 92.

Настройте форму. В частности, задайте ее позицию и определите модальный результат для кнопок.

```
ColorForm: TColorForm
BorderStyle = bsToolWindow
Caption = выберите цвета
Position = poScreenCenter
```

В обработчике нажатия на фигуры Shape опишите вызов диалогов. Обработчики для кнопок можете не описывать.

В модуле ColorUnit – обработчики нажатия на FieldShape и CellShape:

```
procedure TColorForm.FieldShapeMouseDown(Sender: TObject; Button: TMouseButton;
Shift: TShiftState; X, Y: Integer);
begin
  MainForm.FieldColorDlg.Execute;
  FieldShape.Brush.Color := MainForm.FieldColorDlg.Color;
end;
```

```
procedure TColorForm.CellShapeMouseDown(Sender: TObject; Button: TMouseButton;
Shift: TShiftState; X, Y: Integer);
begin
  MainForm.CellColorDlg.Execute;
```

```

CellShape.Brush.Color := MainForm.CellColorDlg.Color;
end;

```

В обработчике кнопки "задать цвета" на форме визуализации опишите модальный вызов этого нового диалога.

В модуле LifeUnit описываем обработчик SetColorBtn:

```

procedure TLifeForm.ColorBtnClick(Sender: TObject);
var
    C1, C2: TColor;
begin
    C1 := MainForm.FieldColorDlg.Color;
    C2 := MainForm.CellColorDlg.Color;
    ColorForm.FieldShape.Brush.Color := C1;
    ColorForm.CellShape.Brush.Color := C2;
    if ColorForm.ShowModal = mrCancel then
        begin
            MainForm.FieldColorDlg.Color := C1;
            MainForm.CellColorDlg.Color := C2;
        end;
end;

```

В данном случае вызывается форма ColorForm – диалог установки цвета. Если в нем нажата кнопка "Отмена", то значения цветов восстанавливаются. Они специально для этого перед вызовом сохраняются в C1 и C2.

18. Пользователь проигрывателя может случайно закрыть окно формы управления. Следовательно, придется перезапускать программу.

Можно решить эту проблему следующим образом. При попытке закрытия спросить пользователя, хочет ли он совсем выйти из программы. И если хочет – закрыть главную форму.

```

procedure TControlForm.FormCloseQuery(Sender: TObject; var CanClose: Boolean);
begin
    if MessageDlg('Выйти из программы?', mtWarning,
        [mbYes, mbNo], 0) = mrYes then MainForm.Close
    else CanClose := false;
end;

```

Этой конструкцией закрывается главная форма из формы управления. То же событие можно обработать и в главной форме, каждый раз спрашивая о желании выйти.

Запустите программу.

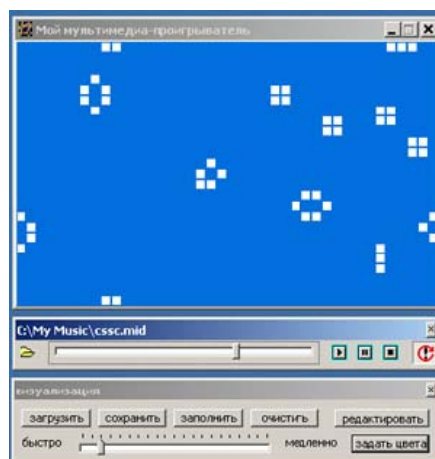


Рис. 93.

19. Подведем итоги. Для создания многооконности необходимо:

1. Создать очередную форму.
2. Модули, предназначенные пользователю, поместить в `interface`, а модули, предназначенные программисту, поместить в `uses`.
3. Необходимо помнить о круговых ссылках и избегать *ignotum per ignotius*. В интерфейс поместить только необходимое.
4. В `OnCreate` делать описание, не забывая о порядке создания форм в файле проекта.
5. Программа заканчивается, когда закрывается главная форма. Остальные формы приложения могут закрываться сколько угодно.
6. Диалоги: вызов `ShowModal`, результат – `ModalResult` нажатой кнопки.
7. Выбрать нужную форму или модуль для редактирования можно с помощью кнопок панели инструментов или клавишами `Ctrl-F12` и `Shift-F12`. Переключение модуль-форма – просто клавишей `F12`.

### 1.1. Часто встречающиеся свойства элементов управления, используемые в Delphi

В зависимости от своего предназначения свойства могут быть представлены в окне свойств в самых разных видах. Например, свойства, диапазон значений которых невелик, нередко представляются в виде выпадающего списка, в котором перечислены все эти значения, и из них достаточно выбрать наиболее подходящее.

К таким свойствам относится в частности Color (цвет элемента). Его можно задать как для любого элемента управления на форме, так и для самой формы. При этом в списке возможных значений в дополнение к названию цвета также показывается небольшой прямоугольник, закрашенный соответствующим цветом - в правой части выпадающего списка значений окна свойств.

Свойство BorderStyle (стиль границ) позволяет задать внешний вид границ элемента на форме (или самой формы). Такие границы могут отличаться от элемента к элементу, а у некоторых элементов (например, кнопки) просто отсутствовать.

Свойство PopupMenu (контекстное меню) позволяет привязать составленное разработчиком локальное меню к некоторому элементу - оно будет вызываться при щелчке на нем правой кнопкой мыши.

Свойство Cursor (курсор) позволяет выбрать вид курсора, который может изменяться при наведении его на соответствующий элемент. Это изменение будет происходить только в работающей программе, а не в проектируемой форме в дизайнере. Доступные формы курсора наглядно показываются в выпадающем списке.

Свойство Enabled (включено) позволяет делать различные элементы временно недоступными для пользователя. Это свойство имеет логический тип Boolean и соответственно может принимать одно из двух значений True или False. Когда выбрано значение False, пользователь не сможет взаимодействовать с этим элементом управления - не сможет нажимать на кнопку, переключать переключатель, выбирать пункт меню и так далее. Но сам элемент на форме будет виден, только возможно в некотором "пассивном" виде.

Свойство Visible определяет видимость элемента на экране. Если его значение равно True, то, хотя такой элемент будет виден в дизайнере, в рабочей скомпилированной программе он виден не будет. Но это свойство надо отличать от свойства Enabled, которое, будучи переведенным в состояние False, хотя и делает элемент недоступным пользователю для воздействия, тем не менее сохраняет его видимым в окне. А свойство Visible может просто сделать элемент невидимым.

С помощью свойства AutoSize можно задать автоматическую подстройку размеров элемента под длину содержащегося в нем текста. Данное свойство востребовано прежде всего в элементах вроде поля-надписи и имеется не у всех элементов, а преимущественно у тех, которые применяются для обработки и отображения текста.

### 1.2. Вложенные свойства

Некоторые свойства могут иметь вложенную структуру и сами быть вложенными. Такие свойства выделяются в окне свойств маленьким плюсиком с левой стороны списка. При щелчке этот плюсики раскрывается во вложенный список дополнительных свойств, относящихся к данному свойству. Только они будут несколько сдвинуты вправо, чтобы разработчик не путал их с собственными свойствами текущего элемента.

Вложенные свойства представляют собой другие объекты и элементы управления. Когда обращаются к вложенному объекту, то в окне свойств на дополнительном уровне показываются свойства этого объекта. Типичный пример такого свойства, часто вкладываемого в другие объекты - свойство Font (шрифт). Если выделить в дизайнере поле-надпись и раскрыть данное свойство, то можно увидеть более подробно, из каких свойств оно на самом деле состоит.

В поле Name, относящемся к шрифту, можно подбирать подходящий профиль шрифта. С помощью свойства Size - задавать его размер, с помощью свойств Bold и Italic (они в свою очередь вложены в свойство Style) - делать шрифт жирным или наклонным и так далее. При этом, как видно, названия различных свойств обычно точно отражают их предназначение и совпадают где возможно с названиями, принятыми в различных популярных прикладных программах в качестве стандартных обозначений.

Свойство Color (цвет шрифта) определяет, каким цветом будут показываться надписи, сделанные на элементе управления. Свойства Left и Top определяют местонахождение элемента на форме (координаты на форме левого - верхнего угла элемента). Свойства вложенных Width (ширина) и Height (высота) задают размер элемента. Свойство Position реализовано для формы. Оно позволяет определить, в каком месте экрана будет показана соответствующая форма при ее открытии (после запуска скомпилированной программы). Position может принимать одно из следующих значений:

- poDesigned. Форма показывается на экране в точности в той позиции, в какой она находилась на экране при ее подготовке в дизайнере.
- poDesktopCenter. Форма центрируется по отношению к краям рабочего стола.
- poScreenCenter. Форма центрируется по отношению к краям экрана.
- poDefaultPosOnly. Форма показывается в месте, установленном в Windows по умолчанию.
- poDefault. Форма показывается в месте, установленном в Windows по умолчанию, а размеры окна также принимаются равными с установленными в Windows по умолчанию.
- poOwnerFormCenter. Форма центрируется по отношению к родительскому окну.

### 1.3. Списки

Очень полезный элемент управления - список (набор строк) ListBox на панели Standard. Свойство ItemIndex - номер текущей выделенной строки. Если ничего не выделено, то принимает значение -1, иначе - номер выделенной строки в списке. Нумерация строк начинается с нуля. Фактически, список - это массив строк (на самом деле - объект, включающий массив строк), только он дополнен разными удобными свойствами и методами для работы с этим массивом.

Число элементов можно узнать через свойство Count, а флажок Sorted (тип boolean), когда принимает значение true, приводит к автоматической сортировке значений массива. С помощью Sorted удобно выполнять сортировку разных значений. Сами строки хранятся в свойстве Items. Доступ к содержимому Items происходит через подсвойство Strings.

Например, в s записываем седьмую строку списка строк ListBox1:

```
s := ListBox1.Items.Strings[ 7 ];
```

Содержимое списка можно очистить, обратившись к методу Clear свойства Items:

```
ListBox1.Items.Clear;
```

Исходно список пустой, а добавление новой строки в список происходит таким образом:

```
s := ' добавляемая строка ' ;  
ListBox1.Items.Add( s );
```

### 1.4. Элементы управления

Для списка можно отслеживать выбор какого-то элемента мышкой. Для этого в инспекторе объектов на закладке Events, где задаются события текущего объекта, которые можно обрабатывать, дважды щелкнуть на строчке OnClick (событие, возникающее при щелчке мышкой на списке), и создастся обработчик такого щелчка (сначала надо выделить конечно сам список на форме). В этом обработчике можно с помощью свойства ItemIndex можно узнать, какой элемент был выделен. Это удобно, например, когда у нас есть список заданий, и мы хотим, когда выбрали в списке упражнений элемент-название, динамически

показывать, например, в поле-надписи или многострочном поле подробное описание соответствующего упражнения.

### 1.5. Многострочное поле Мемо

Многострочное поле Мемо позволяет редактировать несколько строчек. Основное свойство - Lines (массив строк).

Очистка содержимого: Memo1.Lines.Clear;

Memo1 - это имя компонента Мемо по умолчанию.

Обратиться к любой строке можно по номеру (нумерация начинается с нуля:

Memo1.Lines[0] := ' Это первая строка! ' ;

Добавление строки осуществляется методом Add:

Memo1.Lines.Add( ' эту строку добавляем в конец... ' );

Можно сохранить все содержимое Мемо в текстовый файл (указываем полный/относительный к нему путь):

Memo1.Lines.SaveToFile( 'c:\txt\memotekst.txt' );

Число строк - метод Count:

n := Memo1.Lines.Count; // текущее число строк в Мемо

Свойство WordWrap задает, будут ли слова переноситься на новую строчку, если строка не уместилась в длину в окне Мемо. При этом, даже если перенос будет автоматическим, считается строка все равно одной целой, при автопереносе новые строки в Lines не появляются.

Свойство ReadOnly задает, доступно ли содержимое поля для редактирования, или же только для просмотра и выделения/копирования.

### 1.6. Компоненты, используемые в графике

Компонент Shape (панель Additional) позволяет на форме создавать разноцветные круги, квадраты, эллипсы. Конкретная форма задается свойством Shape, которое может принимать значения, из списка stRectangle, stSquare, stRoundRect, stRoundSquare, stEllipse, stCircle. Следовательно, форму расположенного на форме объекта можно менять динамически, по нажатию на кнопку:

**procedure** TForm1.Button1Click(Sender: TObject);

**begin**

Shape1.Shape := stCircle;

**end;**

И цвет соответственно, свойство Color, само вложено в свойство Brush (кисть), поэтому к нему надо обращаться по цепочке: Shape1.Brush.Color := clRed;

У кисти есть подсвойство Style, которое определяет способ "заливки" внутренности объекта - сплошное bsSolid, в решеточку bsCross и т.д. Конкретные значения некоторого свойства можно посмотреть в инспекторе объектов, в выпадающем списке.

Shape1.Brush.Style := bsCross;

Компонент Bevel позволяет создавать выпуклые/невыпуклые панели, рамки и линии. Его главные свойства - Shape и Style.

Компонент Panel (раздел Standard) предназначен для создания декоративных панелей, на которых удобно группировать элементы управления, выделяя их визуально. Панель оформляется с помощью свойств, связанных с внешним видом ее каемки - BorderStyle, BorderWidth, BevelWidth, BevelInner, BevelOuter.

Кроме того, можно использовать кнопки с картинками BitBtn или SpeedButton на панели Additional. Через свойство Glyph можно задать картинку, которая будет показана на самой кнопке.

## ПРИЛОЖЕНИЕ 2

Таблица 1.1. Цвет компонента или объекта (свойство Color)

Значение	Цвет	Значение	Цвет
clBlack	Черный	clGray	Серый
clMaroon	Темно-красный	clSilver	Серебряный
clGreen	Зеленый	clRed	Красный
clOlive	Оливковый	clLime	Ярко-зеленый
clNavy	Темно-синий	clBlue	Голубой
clPurple	Фиолетовый	clFuchsia	Сиреневый
clTeal	Сине-зеленый	clAqua	Ярко-голубой
		clWhite	Белый

Таблица 1.2. Системные цвета Windows, определяемые цветовой схемой

Значение	Цвет для элемента
clBackground	фон окна
clActiveCaption	заголовок активного окна
clInactiveCaption	заголовок неактивного окна
clMenu	фона меню
clWindow	фон Windows
clWindowFrame	рамка окна
clMenuText	текст элемента меню
clWindowText	текст внутри окна
clCaptionText	заголовок активного окна
clActiveBorder	рамка активного окна
clInactiveBorder	рамка неактивного окна
clAppWorkSpace	рабочая область окна
clHighlight	фон выделенного текста
clHightlightText	выделенный текст
clBtnFace	Кнопка
clBtnShadow	фон кнопки
clGrayText	недоступный элемент меню
clBtnText	текст кнопки

Таблица 1.3. Базовые свойства Color как шестнадцатеричные константы

Цвет	Значение	Цвет	Значение
Черный	\$000000	Синий	\$000080
Светло-синий	\$0000FF	Зеленый	\$008000
Светло-зеленый	\$00FF00	Сине-зеленый	\$008080
Голубой	\$00FFFF	Коричневый	\$800000
Светло-красный	\$FF0000	Темно-сиреневый	\$800080
Сиреневый	\$FF00FF	Оливковый	\$808000
Светло-желтый	\$FFFF00	Темно-серый	\$808080
Белый	\$FFFFFF	Светло-серый	\$C0C0C0

Таблица 1.4. Выравнивание компонента внутри формы (свойство Align)

Значение	Расположение компонента
alNone	Без выравнивания на месте размещения при создании программы (значение по умолчанию)
alTop	Перемещение в верхнюю часть формы, ширина компонента становится равной ширине формы (высота не меняется)
alBottom	Перемещение в нижнюю часть формы, ширина становится равной ширине формы (высота не изменяется)
alLeft	Перемещение в левую часть формы, высота компонента становится равной высоте формы (ширина не изменяется)
alRight	Перемещение в правую часть формы, высота становится равной высоте формы (ширина не изменяется)
alClient	Компонент полностью занимает всю рабочую область формы



### ПРИЛОЖЕНИЕ 3

Таблица 2.1. Функции ввода вывода

Функция	Описание
InputBox (Заголовок, Подсказка, Значение)	В результате выполнения функции на экране появляется диалоговое окно, в поле которого пользователь может ввести строку символов. Значением функции является введенная строка. Параметр <i>Значение</i> задает значение функции «по умолчанию», т.е. строку, которая будет в поле редактирования в момент появления окна.
ShowMessage (s)	Процедура выводит окно, в котором находится сообщение <i>s</i> и командная кнопка <i>Ok</i> .
MessageDlg (s, t, b, h)	Выводит на экран диалоговое окно с сообщением <i>s</i> и возвращает код кнопки, щелчком на которой пользователь закрыл окно. Параметр <i>t</i> определяет тип окна: <i>mtWarning</i> – Внимание; <i>mtError</i> – ошибка; <i>myInformation</i> – информация; <i>mtConfirmation</i> – запрос; <i>mtCustom</i> – пользовательское (без значка). Параметр <i>b</i> (множество – заключенный в квадратные скобки список констант) задает командные кнопки диалогового окна ( <i>mbYes</i> , <i>mbNo</i> , <i>mbOk</i> , <i>MbCancel</i> , <i>mbHelp</i> , <i>mbAbort</i> , <i>mbRetry</i> , <i>mbIgnore</i> , <i>mbAll</i> ). Параметр <i>h</i> задает раздел справочной системы программы, который появится в результате нажатия кнопки <i>Help</i> или клавиши <F1>. Если справочная система не используется, значение параметра должно быть 0. Значением функции может быть одна из констант: <i>mbYes</i> , <i>mbNo</i> , <i>mbOk</i> , <i>MbCancel</i> , <i>mbHelp</i> , <i>mbAbort</i> , <i>mbRetry</i> , <i>mbIgnore</i> , <i>mbAll</i> , обозначающая соответствующую команду.

Таблица 2.2. Математические функции

Функция	Описание
Abs (n)	Абсолютное значение n
Sqrt (n)	Квадратный корень из n
Sqr (n)	Квадрат n
Exp (n)	Экспонента n
Ln (n)	Натуральный логарифм n
Random (n)	Случайное целое число в диапазоне от 0 до n-1 (перед первым обращением к функции необходимо вызвать функцию <i>Randomize</i> , которая выполнит инициализацию программного генератора случайных чисел)
Sin (α)	Синус выраженного в радианах угла α
Cos (α)	Косинус выраженного в радианах угла α
Arctan (α)	Арктангенс выраженного в радианах угла α

Таблица 2.3. Функции преобразования

Функция	Описание
Chr (n)	Символ, код которого равен n
IntToStr (k)	Строка, являющаяся изображением целого k
FloatToStr (n)	Строка, являющаяся изображением вещественного n
FloatToStr (n, f, k,m)	Строка, являющаяся изображением вещественного n. При вызове функции указывают: f - формат; k - точность; m - количество цифр после десятичной точки. Формат определяет способ изображения числа: ffGeneral - универсальный; ffExponent - научный; ffFixed - с фиксированной точкой; ffNumber - с разделителями групп разрядов; ffCurrency - финансовый. Точность – нужное общее количество цифр: 7 или меньше для значений типа <i>Single</i> , 15 или меньше для значения типа <i>Double</i> и 18 или меньше для значения типа <i>Extended</i>
Format (s, [n1, n2, ...])	Строка, являющаяся изображением значений <i>n1</i> , <i>n2</i> и т.д. Способ преобразования значений в строку символов определяют управляющие символы, которые находятся в строке форматирования <i>s</i>
StrToInt (s)	Целое, изображением которого является строка <i>s</i>
StrToFloat (s)	Вещественное, изображением которого является строка <i>s</i>
Round (n)	Целое, полученное путем округления <i>n</i> по известным правилам
Trunc (n)	Целое, полученное путем отбрасывания дробно части <i>n</i>
Frac (n)	Дробное, представляющее собой дробную часть вещественного <i>n</i>
Int (n)	Дробное, представляющее собой целую часть вещественного <i>n</i>

## ПРИЛОЖЕНИЕ 4

Таблица 3.1. События

Событие	Происходит
OnClick	При щелчке мыши
OnDbClick	При двойном щелчке кнопкой мыши
OnMouseDown	При нажатии кнопки мыши
OnMouseUp	При отпускании кнопки мыши
OnMouseMove	При перемещении мыши
OnKeyPress	При нажатии клавиши клавиатуры
OnKeyDown	При нажатии клавиши клавиатуры.
События <i>OnKeyDown</i> и <i>OnKeyPress</i> – это чередующиеся, повторяющиеся события. Которые должны происходить до тех пор, пока не будет отпущена удерживаемая клавиша (в этот момент происходит событие <i>OnKeyUp</i> )	
OnKeyUp	При отпускании нажатой клавиши клавиатуры
OnCreate	При создании объекта (формы, элемента управления). Процедура обработки этого события обычно используется для инициализации переменных, выполнения подготовительных действий
OnPaint	При появлении окна на экране в начале работы программы, после появления части окна, которая, например, была закрыта другим окном и в других случаях. Событие сообщает о необходимости обновить (перерисовать) окно
OnEnter	При получении элементом управления фокуса
OnExit	При потере элементом управления фокуса

## Литература

1. Симонович С.В., Евсеев Г.А. Занимательное программирование: Delphi. – М.: АСТ – ПРЕСС КНИГА: Инфорком – Пресс, 2001.
2. Культин Н.Б. Delphi в задачах и примерах. – СПб.: БХВ – Петербург, 2005.
3. Фаронов В.В. Программирование на языке высокого уровня: Учебник для вузов. – СПб.: Питер, 2003.
4. Бобровский С.И. Delphi 7. Учебный курс. – СПб.: Питер, 2005.
5. Кетков Ю.Л., Кетков А.Ю. Практика программирования: Visual Basic, C++ Builder, Delphi. - СПб.: БХВ – Петербург, 2005.

## Содержание

Введение .....	2
1. Идеология объектно-ориентированного программирования .....	3
1.1. Элементы интерфейса программы Компоненты .....	3
1.2. Инспектор объектов .....	7
2. Три основных принципа ООП: наследование, инкапсуляция, полиморфизм .....	8
3. Использование справки .....	10
4. Структура программы .....	11
5. Методы отладки и борьбы с ошибками .....	13
Практическая работа № 1 «Моя первая программа» .....	16
Практическая работа № 2 Создание консольного приложения .....	17
Практическая работа № 3 «Приветствие» .....	19
Практическая работа № 4 «Случайный выбор» .....	21
Практическая работа № 5 Изменение заголовка формы .....	23
Практическая работа № 6 «Двигающаяся кнопка» .....	25
Практическая работа № 7 «Альбом» .....	26
Практическая работа № 8 Работа с меню .....	28
Практическая работа № 9 Случайный выбор из списка .....	31
Практическая работа № 10 Простейший плеер .....	33
Практическая работа № 11 «Прыгающая кнопка» .....	35
Практическая работа № 12 «Таблица умножения» .....	37
Практическая работа № 13 Применение полос прокрутки .....	40
Практическая работа № 14 «Светофор» .....	43
Практическая работа № 15 «Ханойские башни» .....	46
Практическая работа № 16 «Электронный альбом» .....	49
Практическая работа № 17 «Вычисление процентов» .....	52
Практическая работа № 18 «Головоломка № 1» .....	54
Практическая работа № 19 «Головоломка № 2» .....	56
Практическая работа № 20 «Обычный калькулятор» .....	58
Практическая работа № 21 «Строковый калькулятор» .....	61
Практическая работа № 22 Нахождение индекса в массиве случайных чисел .....	63
Практическая работа № 23 Нахождение минимального и максимального числа в массиве .....	65
Практическая работа № 24 «Текущее время и текущая дата» .....	67
Практическая работа № 25 «Электронные часы» .....	68
Практическая работа № 26 Графика .....	70
Практическая работа № 27 «Олимпийский флаг» .....	75
Практическая работа № 28 «Узоры» .....	77
Практическая работа № 29 Перемещение рисунка .....	80
Практическая работа № 30 Рисунок .....	83
Практическая работа № 31 Построение графика .....	86
Практическая работа № 32 «Градусник» .....	88
Практическая работа № 33 Вывод табличных данных .....	92
Практическая работа № 34 «Игра» .....	96
Практическая работа № 35 «Тест» .....	104
Практическая работа № 36 «Проигрыватель» .....	109
ПРИЛОЖЕНИЕ 1 .....	
1.1. Часто встречающиеся свойства элементов управления, используемые в Delphi .....	133
1.2. Вложенные свойства .....	133
1.3. Списки .....	134
1.4. Элементы управления .....	134
1.5. Многострочное поле Метод .....	135

1.6. Компоненты, используемые в графике .....	135
ПРИЛОЖЕНИЕ 2 .....	137
Таблица 1.1. Цвет компонента или объекта (свойство Color) .....	137
Таблица 1.2. Системные цвета Windows, определяемые цветовой схемой .....	137
Таблица 1.3. Базовые свойства Color как шестнадцатеричные константы .....	138
Таблица 1.4. Выравнивание компонента внутри формы (свойство Align) .....	137
ПРИЛОЖЕНИЕ 3 .....	139
Таблица 2.1. Функции ввода вывода .....	139
Таблица 2.2. Математические функции .....	139
Таблица 2.3. Функции преобразования .....	140
ПРИЛОЖЕНИЕ 4 .....	141
Таблица 3.1. События .....	141
Литература .....	142