МБУ ДО ЦЕНТР ДЕТСКОГО (ЮНОШЕСКОГО) НАУЧНО-ТЕХНИЧЕСКОГО ТВОРЧЕСТВА

СОЗДАНИЕ WINDOWS-ПРИЛОЖЕНИЙ В СИСТЕМЕ ОБЪЕКТНО-ОРИЕНТИРОВАННОГО ПРОГРАММИРОВАНИЯ DELPHI

методическая разработка практических занятий для обучающихся по направлению «программирование»

Авторы:
педагоги дополнительного образования
Андреева Ирина Юрьевна
Горностаева Анна Владимировна

г. Армавир

Краткая аннотация

В предлагаемой методической разработке рассматриваются вопросы обучения технологии создания компьютерных программ на занятиях по информатике. Представленные материалы имеют практическую направленность – они посвящены разработке законченных Windows-приложений на уровне использования готовых компонентов системы программирования Delphi с написанием на их основе собственного программного кода.

Материал содержит серию практических занятий (практикумов). Каждое занятие включает необходимые теоретические сведения, обязательные задания с полным описанием технологии по их выполнению и задания для самостоятельной работы. Кроме того, методическое пособие содержит примеры творческих проектов учащихся, электронный сборник учебных проектов, выполненных в системе программирования Delphi.

Представленные материалы можно использовать на занятиях по программированию в рамках обучения детей информатике и компьютерным технологиям в системе дополнительного образования, на уроках информатики и элективных курсах.

Методическая разработка рекомендована в помощь педагогам дополнительного образования, учителям средних школ, преподавателям техникумов и вузов. Данное пособие может использоваться учащимися при заочной и дистанционной формах обучения.

Оглавление

Пояснительная записка

Профессия программиста является достаточно распространенной и престижной, а объектно-ориентированное программирование в настоящее время занимает ведущее место в разработке профессиональных программных средств. Ознакомление с его основами в системе дополнительного образования и в школьном курсе информатики представляется вполне возможным и полезным для тех учащихся, которые ориентируются на профессии, связанные с разработкой компьютерных программ. Обучение программированию развивает у учащихся операционный стиль мышления. Операционным компонентом мышления считается система мыслительных операций, состоящая из анализа, синтеза, сравнения, абстрагирования, обобщения, классификации, систематизации. Следует отметить и тот факт, что наибольшее количество задач итоговой аттестации по информатике в виде единого государственного экзамена представлено разделом «Алгоритмизация и программирование». В заданиях единого государственного экзамена остаются требования к высокому уровню программирования у выпускников, но стандарт среднего образования не позволяет достичь такого уровня.

Дополнительное образование является на сегодняшний день нереализованным резервом, который может дать хорошую подготовку школьникам в области программирования. Быстро растущая популярность визуального программирования в системах программирования типа Delphi, Visual Basic, C++Builder делает привлекательной идею положить одну из них в основу учебного курса. Однако методика изучения в школе и в дополнительном образовании любых видов объектного программирования разработана совершенно недостаточно и находится на начальной стадии. Имеется крайне мало литературы начального уровня, предназначенной для освоения Delphi применительно к использованию в учебном процессе.

Предлагаемая методическая разработка **«Создание Windows-приложений в системе объектно-ориентированного программирования Delphi»** представляет собой цикл практических занятий по визуальному программированию в системе объектно-ориентированного программирования Delphi 7.

Цель занятий: обучение технологии создания законченных Windows-приложений на уровне использования готовых компонентов системы программирования Delphi с написанием на их основе собственного программного кода.

Задачи:

- изучение приемов и методов создания Windows-приложений в среде программирования Delphi;
- освоение навыков работы с компонентами и их свойствами;
- формирование умения работы с обработчиками событий;

- формирование умений и навыков программирования с применением операторов и типов данных Object Pascal;
- развитие операционного стиля мышления и исследовательских умений.

Предварительные знания, умения и навыки:

- навыки работы в операционной системе Windows;
- умение программировать на языке Паскаль;
- знание основных понятий объектно-ориентированного программирования;
- знание идеологии программирования под Windows;
- знание основных категорий Delphi: свойства, события, методы;

Возраст учащихся: 14-18 лет.

Используемые методы обучения: проблемный метод и метод проектов.

Технологические особенности:

- *лаборатория* с мультимедийными компьютерами с процессором Intel 2000 МГц с объемом оперативной памяти 128 Мбайт и выше (не менее десяти рабочих мест), проектор, доска, столы, стулья;
- *программное обеспечение:* операционная система Windows XP; Borland Delphi 7;
- *раздаточный материал.*

Продолжительность одного занятия: 2 часа.

Методическая разработка содержит серию практических занятий (практикумов). Каждое занятие включает необходимые теоретические сведения, обязательные задания с полным описанием технологии по их выполнению и задания для самостоятельной работы. Теоретический материал предлагается обучающимся для записи в тетрадь. Задания для самостоятельной работы можно выполнить дома. Предлагаемые практические задания учащиеся могут выполнять самостоятельно, так и под руководством педагога, который в любой момент может дать консультацию по всем возникающим вопросам.

Кроме того, методическое пособие содержит примеры творческих проектов учащихся (приложение 1), электронный сборник учебных проектов, выполненных в системе программирования Delphi (приложение 2).

Представленные материалы можно использовать на занятиях по программированию в рамках обучения детей информатике и компьютерным технологиям в системе дополнительного образования, на уроках информатики и элективных курсах.

Методическая разработка рекомендована в помощь педагогам дополнительного образования, учителям средних школ, преподавателям техникумов и вузов. Данное пособие может использоваться учащимися при заочной и дистанционной формах обучения.

Методические рекомендации

Обучение программированию в системе дополнительного обучения информатике в основном строится по следующей схеме. На первом этапе рассматривается процедурный язык программирования или объектно-ориентированный язык (например, Turbo Pascal). В связи с распространением в настоящее время современных технологий программирования, во многих дополнительных образовательных программах на втором этапе предусмотрено изучение визуальной среды программирования и технологии создания Windows-приложений в ней (система программирования Delphi). Можно использовать и другой подход. Он состоит в том, чтобы учащиеся начинали составлять программы сразу в визуальной среде программирования, минуя объектно-ориентированный и процедурный языки.

Визуальная технология программирования является современным подходом к созданию программ. Визуальный язык программирования делает процесс создания программ наглядным и увлекательным и берет на себя большую часть рутинной работы. Но, вместе с этим, данная технология позволяет разрабатывать достаточно сложные и профессиональные приложения. Визуальное программирование обладает достоинством наглядного представления информации и гораздо лучше соответствует природе человеческого восприятия, чем методы традиционного программирования.

Обучение детей визуальному программированию целесообразно начать с описания особенностей и преимуществ составления программ в визуальной среде, например в Delphi. Среда программирования Delphi является профессиональной средой программирования и предназначена для разработки достаточно сложных и производительных программных комплексов, получивших название проектов. Для разработки грамотного и профессионального проекта требуется знание учащимися основных шагов в создании проектов.

Первоначальное рассмотрение среды программирования Delphi и основных понятий визуального программирования тесно связано с введением компонентов, их свойств и элементов программирования. Это одна из особенностей и трудностей обучения визуальному программированию, проявляющаяся в том, что изучаемый материал требует знаний далее рассматриваемых разделов.

На первом этапе (рис.1) разбирается конструирование графического интерфейса проекта, изменение свойств компонентов с помощью Инспектора объектов и через программный код. Здесь следует рассмотреть свойства компонентов, которые будут часто встречаться в дальнейшем. К этим компонентам в первую очередь относятся форма, текстовое поле, надпись, командная кнопка. Для того чтобы создаваемые проекты были красочными и интересными, при проектировании графического интерфейса следует предусмотреть возможность добавления графических объектов.

Таким образом, необходимо рассмотреть свойства объекта TImage, который позволяет добавлять графическое изображение в проект. Обычно данный компонент изучается не на первых этапах, а уже после введения основных операторов. Особенностью данного подхода является то, что учащиеся начинают работать в окне программного кода до введения элементов языка программирования.

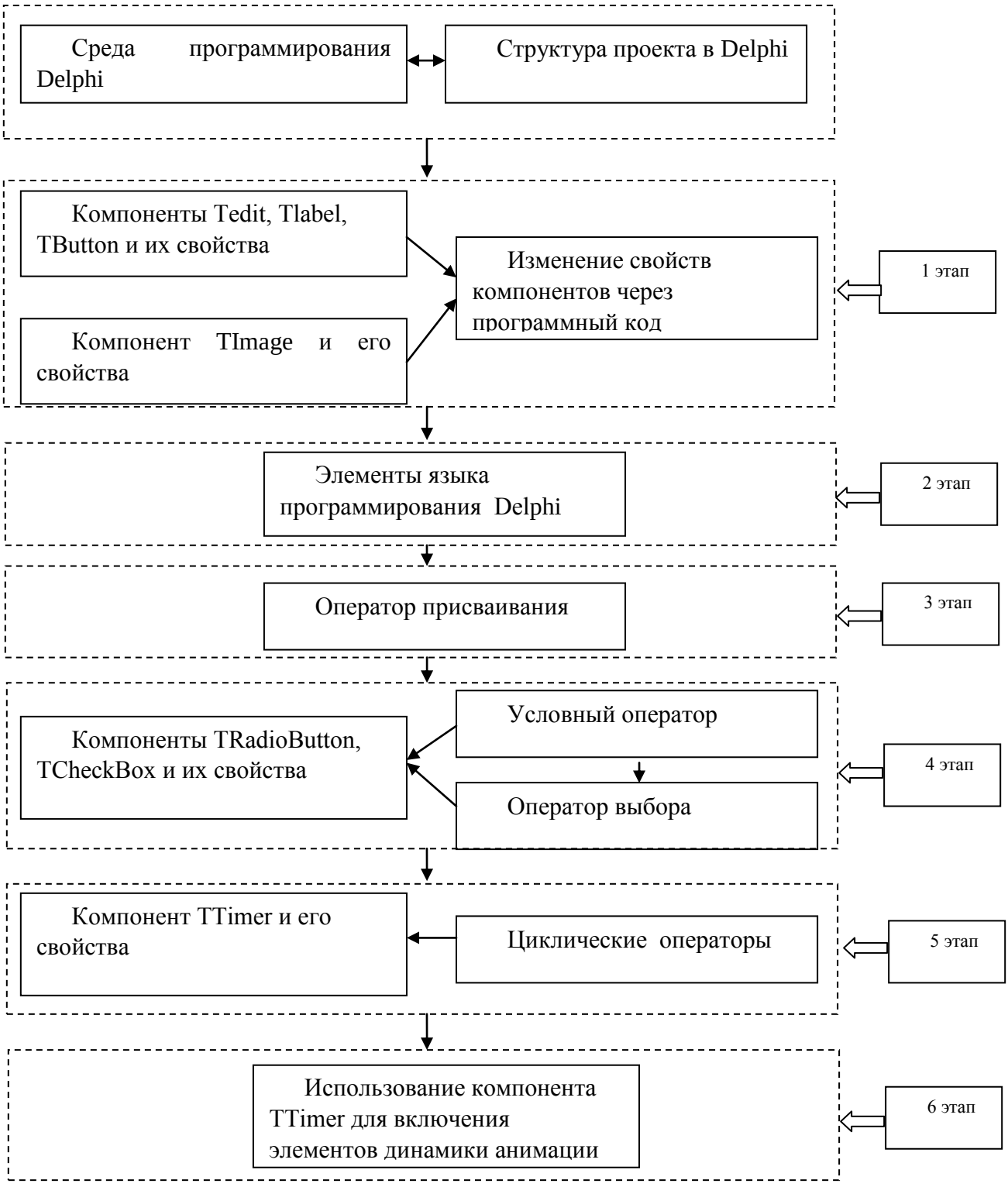


Рис.1 *Модель формирования структуры содержания обучения визуальному программированию*

Второй этап включает в себя изучение основных операторов, реализацию в визуальном языке программирования линейных, условных и циклических алгоритмических конструкций. На данном этапе вводятся элементы языка программирования (алфавит, зарезервированные слова, идентификаторы, типы данных, константы и т.п.). В основном в создаваемых вычислительных и логических проектах используется обработчик события OnClick.

На третьем этапе происходит возвращение к разработке графического интерфейса проекта, рассматривается работа компонента Таймер. Данный объект вводится как более удобный способ организации повторения действий через задаваемый интервал времени.

Далее на четвертом этапе таймер служит для включения элементов динамики анимации в проект (движение объектов, изменение форм, размеров и видов объектов). Здесь учащиеся работают с обработчиком события OnTimer, создавая красочные и интересные проекты с применением графических изображений.

При создании серьезного приложения следует предусмотреть различные варианты работы с программой. События, отвечающие за управление работой объектами с помощью мыши и клавиатуры, разбираются на пятом этапе. Именно здесь вводится символьный тип данных, хотя он использовался и на предыдущих этапах.

На шестом этапе рассматриваются составные типы данных, в частности массивы и строковые типы, показывается их реализация в визуальной среде программирования. Для изучения строкового типа данных целесообразно использовать такие компоненты, как Текстовая область, Список выбора и Выпадающий список. Практически все приложения Windows имеют меню, которое является распространенным элементом пользовательского интерфейса. Работа компонента Меню разбирается на седьмом этапе.

В структуре содержания обучения визуальному программированию можно выделить три направления, реализуемые в несколько этапов:

1. Работа с компонентами и их свойствами.
2. Работа с операторами и типами данных визуального языка программирования.
3. Работа с обработчиками событий.

Данную структуру можно расширить в зависимости от количества часов, отведенных в дополнительном образовании, и от запланированного результата обучения.

Основной задачей педагога является поддержание стабильного состава группы. Для решения этой проблемы необходимо в содержание учебного материла по алгоритмизации и программированию включать интересные, доступные и красивые задания. Это могут быть задачи с игровыми моментами, с динамикой и анимацией. В процессе обучения у ребят рождаются новые идеи, темы творческих проектов. Учащиеся защищают свои компьютерные программы на конференциях и конкурсах различного уровня и занимают призовые места.

Занятие 1.Элементы интерфейса Delphi 7

Цель: знакомство с интерфейсом среды программирования Delphi 7.

Задача: изучить структуру окон среды Delphi 7, технологию создания и сохранение проекта.

Теоретическая часть.

Delphi — система визуального объектно-ориентированного программирования, позволяющая на самом современном уровне создавать прикладные программы Windows. Возможности ООП в Delphi базируются на свойствах языка Object Pascal, программа, которую строит Delphi, основана на модульном принципе.

Интегрированная среда разработки Delphi дает возможность создавать программы в стиле визуального конструирования формы, разместив на ней какие-либо визуальные элементы.

В Delphi имеются 10 окон, после загрузки появляются пять окон (рис.1.1):

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

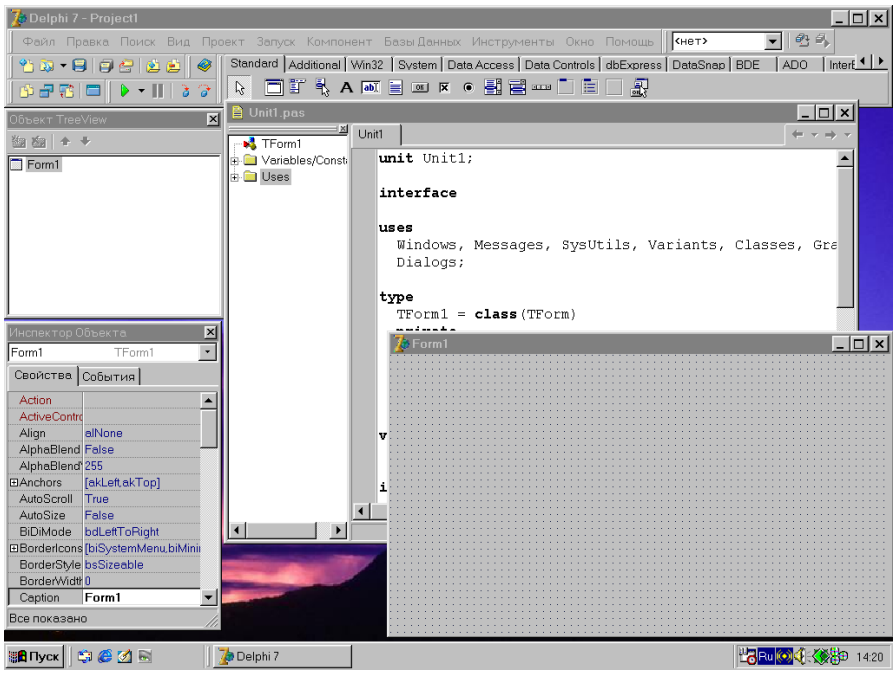


Рис. 1.1

В **главном окне реализуются основные** функции **управления проектом** создаваемой программы. **Это** окно содержит:

• строку заголовка;

• строку меню;

• панель инструментов;

• палитру компонентов.

Строка заголовка главного **окна отображает имя открытого в данный** момент проекта.

Строка меню содержит команды, необходимые для разработки и тестирования приложений, и используется так же, как любое стандартное меню **Windows.**

Панель инструментов предназначена для выполнения некоторых команд, реализуемых главным меню. На этой панели есть, в частности, кнопка сохранения проекта на диске, кнопка открытия проекта, кнопка запуска программы на выполнение.

Палитра компонентов устроена в виде наборов пиктограмм. **Совокупность наборов составляет** библиотеку визуальных компонентов **(Visual Component Library — VCL).** Имеется несколько категорий компонентов, каждая из которых расположена на своей вкладке. С помощью палитры компонентов мы будем создавать экземпляры компонентов **(объекты)** на форме.

Окно проектировщика формы — главное место, где происходит сборка программы из компонентов, содержащихся в палитре компонентов. Сама форма — это уже готовая к выполнению программа. В указанное место формы будет вставляться объект — экземпляр компонента выбранного типа.

Окно инспектора объектов отображает свойства активизированного щелчком мышью какого-либо компонента или самой формы. Имя активизированного компонента находится под заголовком окна.

Это окно имеет две вкладки — **Properties (Свойства) и Events (События).** Свойство определяет атрибут компонента, например размер кнопки или шрифт метки. Событие означает, например, такие действия, как щелчок мышью на кнопке или закрытие окна.

По ходу работы система формирует в **окне редактора кода** текст программы на языке Паскаль, связанной с формой. После загрузки Delphi это окно спрятано за окном формы, и его можно увидеть, щелкнув на кнопке **Toggle Form/Unit** на панели инструментов.

Пользователь может дополнять текст программы самостоятельно или по предложению системы в момент размещения объектов на форме.

Создание и сохранение проекта и файла модуля

Рассмотрим некоторые команды пункта меню **File.**

Команда **New** открывает окно New **Items,** при помощи которого можно создать новую форму, шаблон проекта или формы, которые затем могут быть использованы.

Команда **New Form** открывает окно новой формы и соответствующего ей модуля и добавляет их к активному проекту.

Команда New **Application** создает новый проект, состоящий из файла проекта projectl.dpr, файла модуля unitl.pas и файла формы unitl.dfm. При этом IDE отображает окно проектировщика формы и редактора кода. Окно проектировщика формы является активным. После этого рекомендуется

сохранить новый проект, присвоив ему имя.

Команда **Open** Project открывает диалоговое окно **Open Project,** в котором выбирается открываемый файл.

Команда **Save** сохраняет активные файлы под их именами. Если файл не был ранее сохранен и ему не было присвоено имя, Delphi открывает диалоговое окно Save As, в котором нужно указать имя файла.

Команда **Save Project As** позволяет сохранить проект под другим именем и в случае необходимости в другом каталоге. После выбора этой команды появляется диалоговое окно для сохранения файла кода программы. По умолчанию файлу дается расширение .pas, указанное в окне **Тип файла.** Далее следует ввести имя файла и щелкнуть на кнопке **ОК.** Появится следующее диалоговое окно для сохранения файла проекта (расширение .dpr — Delphi Project). Надо ввести имя проекта и щелкнуть на кнопке **ОК.**

Примечание.

- Для сохранения ранее созданного проекта достаточно выполнить команду Save.
- Отводите для каждого нового проекта новый подкаталог.
- После того, как вы создали приложение с пустой формой, сразу сохраните его в нужном каталоге. И в течение работы над проектом почаще выполняйте сохранение.
- Не задавайте одинаковые имена различным файлам, стремитесь не использовать в проекте повторяющиеся имена.
- Всегда при сохранении проектов и модулей задавайте осмысленные имена файлов.
- Создавайте новый проект командой **File | New |Application ,** сразу сохраните проект и файл модуля командой **File| Save Project As.**

Задание для самостоятельного выполнения

1. Создайте новый проект. Для этого выполните команду **File,** New **Application.**
2. Щелкните мышью на кнопке **Run** — выполните «пустую» программу, созданную в п.1. На экране исчезнут все вспомогательные окна, у формы пропадет координатная сетка.
3. Для возвращения в среду Delphi нажмите комбинацию клавиш Alt+F4.
4. Сохраните проект. Для этого выполните команду File, **Save** Project **as,** предварительно создав папку, в которую вы будете записывать свои проекты.

Примечание. Особое внимание обратите на п. 4: самая распространенная ошибка - выполнение команды **File, Save** вместо **File, Save Project as**.

5. Закройте Delphi.
6. Загрузите Delphi.
7. Загрузите свой проект, выполнив команду **File, Open Project** и указав правильно имя (см. п. 4).
8. Повторите п. 2 и 3.

Занятие 2. Понятие события. Реакция программы на событие. Изменение свойств объекта программным путем. Отображение текстовой информации

Цель: знакомство с компонентами классов **TForm, Tlabel и TButton.**

Постановка задачи: Создать приложение «Мое первое приложение», в котором при щелчке пользователя на кнопке появлялась бы какая-нибудь надпись.

Теоретические сведения

Основой программы-приложения является форма. Форма (объект Form) является основой, на которой размещаются другие компоненты. Форма имеет те же свойства, что и окна Windows. Во время проектирования форма покрыта сеткой из точек. В узлах сетки размещаются те компоненты, которые разработчик помещает на форму.

Обычно при нажатии кнопки выполняется некоторое действие. При каждом действии пользователя (нажатии на клавишу клавиатуры, кнопку мыши, перемещении мыши и т.д.) возникает **событие**, которое через Windows передается приложению. Например, при нажатии на кнопку мыши возникает событие OnMouseDown, при отпускании OnMouseUp. Можно рассматривать эти два действия как один щелчок, тогда это событие называется OnClick. Чтобы отреагировать на какое-либо событие, приложение должно вызывать соответствующую процедуру. Именно эту процедуру пишет программист.

План разработки программы

1. Запустите **Delphi с помощью меню Windows** Пуск | Программы. Если Delphi уже работает и вы уже делали какие-то эксперименты с формой, то откройте новое приложение. Для этого вам надо выполнить команду File | New и в открывшемся каскадном меню выбрать раздел Application. Ответьте «No» на вопрос Delphi, хотите ли вы сохранить изменения в вашем проекте.

2. Перенесите на пустую форму, которая открылась вам, кнопку типа **TButton** со страницы **Standard** палитры компонентов, Для этого выделите пиктограмму кнопки (она седьмая слева) и затем щелкните курсором мыши в нужном вам месте формы. На форме появится кнопка, которой Delphi присвоит имя по умолчанию — **Button1.**

3. Аналогичным образом перенесите на форму с той же страницы **Standard** палитры компонентов метку **Label** (она на странице четвертая слева). В этой метке в процессе выполнения приложения будет появляться текст при нажатии пользователем кнопки. Delphi присвоит ей имя **Label1.**

4. Разместите компоненты на форме примерно так, как показано на рис.2.1 При этом уменьшите до разумных размеров окно формы, так как в

вашем первом приложении никаких других компонентов не будет.

5. Выделите на форме компонент **Button1** — кнопку. Перейдите в Инспектор Объектов и измените ее свойство **Caption** (надпись), которое по умолчанию равно **Button1** (имя, которое по умолчанию присвоила этому компоненту Delphi) на «Пуск».

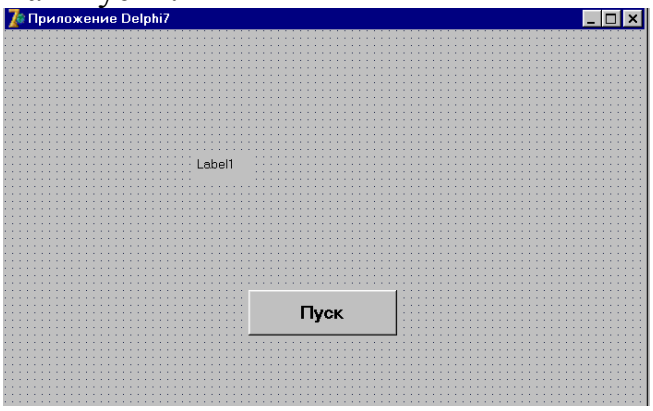


Рис.2.1

6. Укажите метке **Label1**, что надписи на ней надо делать жирным шрифтом. Для этого выделите метку, в окне **Инспектора Объектов** раскройте двойным щелчком свойство **Font** (шрифт), затем так же двойным щелчком раскройте подсвойство **Style** (стиль) и установите в **true** свойство **fsBold** (жирный).

7. Удалите текст в свойстве **Caption** метки **Label1**, чтобы он не высвечивался, пока пользователь не нажмет кнопку приложения.

8. В заголовке окна формы напишите какой-нибудь осмысленный текст. Для этого надо щелкнуть на форме, в окне **Инспектора Объектов** найти свойство **Caption** формы и написать в нем, например, «Приложение Delphi».

9. Теперь вам осталось только написать оператор, который заносил бы в свойство **Caption** метки **Label1** нужный вам текст в нужный момент. Этот момент определяется щелчком пользователя на кнопке. При щелчке в кнопке генерируется событие **OnClick.** Следовательно, обработчик этого события вы и должны написать.

10. Выделите кнопку **Button1** на форме, перейдите в **Инспектор Объектов**, откройте в нем страницу событий (Events), найдите событие кнопки OnClick (оно первое сверху) и сделайте двойной щелчок в окне справа от имени этого события. Это стандартный способ задания обработчиков любых событий**.** Но перейти в обработчик события **OnClick** (только этого события) кнопки можно иначе: достаточно сделать двойной щелчок на компоненте **Button1** на форме. В обоих случаях вы окажетесь в окне **Редактора Кода** и увидите там текст**:**

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

```
end;
```

Заголовок этой процедуры складывается из имени класса вашей формы**(Tform1),** имени компонента **(Button1)** и имени события без префикса **On (Click).** Можете закрыть окно **Исследователя Кода**, встроенное в окно Редактора Код», так как оно пока вам не нужно и будет только мешать. Закрыть это дополнительное окно можно, щелкнув на кнопке в его правом верхнем углу.

11. Напишите в обработчике оператор задания надписи метки **Label1.** Этот оператор может иметь вид:

```
Label1.Caption:='Это мое первое приложение!';
```

Таким образом, полностью ваш обработчик события должен иметь вид:

```
procedure Tform1.Button1Click(Sender: TObject) ;
begin
 Label1.Caption:= 'Это мое первое приложение!';
end;
```

Оператор, который вы написали, означает следующее. Символы «:=» обозначают в языке Object Pascal операцию присваивания, в которой тому, что написано перед этими символами, присваивается значение того, что написано после символов присваивания. Слева вы написали: **Label1.Caption.** Это значит, что вы присваиваете значение свойству **Caption** компонента **Label1.** Все указания свойств и методов производятся аналогичным образом: пишется имя компонента, затем ставится точка, а затем без пробела пишется имя свойства или метода. В данном случае свойству **Caption** вы присваиваете строку текста ‘Это мое первое приложение!’.

Если вы написали первый идентификатор оператора — **Labei1,** поставили точку, то вам всплывет подсказка содержащая список всех свойств и методов метки. Это начал работать Знаток Кода, который стремится подсказать вам свойства и методы компонентов, аргументы функции и их типы, конструкции операторов. Вы можете выбрать из списка нужное ключевое слово, нажать клавишу Enter и выбранное слово (свойство, метод) окажется вписанным в текст. Можете поступить иначе: начать писать нужное свойство. Тогда Знаток Кода сам найдет по первым введенным символам нужное свойство. Когда вы увидели, что нужное слово найдено, можете его не дописывать, а нажать Enter, и Знаток Кода допишет его за вас.

Подсказки Знаток Кода по умолчанию упорядочены по областям видимости и категориям, что не очень удобно. Вы можете изменить характер

упорядочивания, щелкнув в окне Знатока Кода правой кнопкой мыши и включив во всплывшем меню раздел Sort by Name — сортировка по алфавиту.

Подсказки Знатока Кода очень удобны в тех случаях, когда вы не очень точно помните последовательность перечисления параметров какой-нибудь функции, или когда не уверены в имени какого-то свойства. При настройке ИСР вы можете временно отключить Знатока Кода или увеличить задержку, с которой он срабатывает.

Итак, ваше приложение готово. Можете откомпилировать и выполнить его. Для этого выполните команду Run|Run, или нажмите соответствующую быструю кнопку или нажмите «горячую» клавишу F9. Если вы ничего не напутали, то после недолгой компиляции перед вами появится окно вашего первого приложения. Нажав в нем кнопку Пуск, вы увидите указанную вами строку текста (рис.2.2).

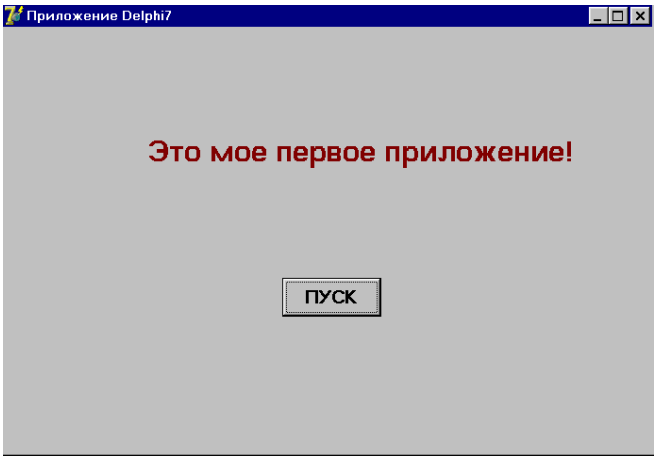


Рис.2.2

Можете попробовать различные манипуляции с окном: перемещение его, изменение размеров его рамки курсором мыши, свертывание и развертывание. В заключение закройте приложение, щелкнув на кнопке в его правом углу.

Занятие 3. Поле ввода текстовой информации (класс TLabeledEdit). Преобразование текста в число и обратно в текст. Вывод результата на форму

Цель: знакомство со стандартными компонентами **Panel** и полем ввода **LabeledEdit .**

Постановка задачи: Создайте приложение «Умножение», которое при нажатии кнопки перемножало бы два числа, введенных пользователем, и показывало бы результат умножения.

При построении этого приложения мы используем новые типы компонентов — окна редактирования. Кроме того, для разнообразия, будем выводить результат не в метку **Label,** а в панель **Panel.**

План разработки программы

1. Откройте новое приложение. Перенесите на него со страницы библиотеки **Additional** два окна редактирования с присоединенными к ним метками **LabeledEdit,** а со страницы библиотеки **Standard** — одну панель **Panel,** одну кнопку **Button** и одну метку **Label** для надписи. Разместите все это примерно так, как показано на рис.3.1

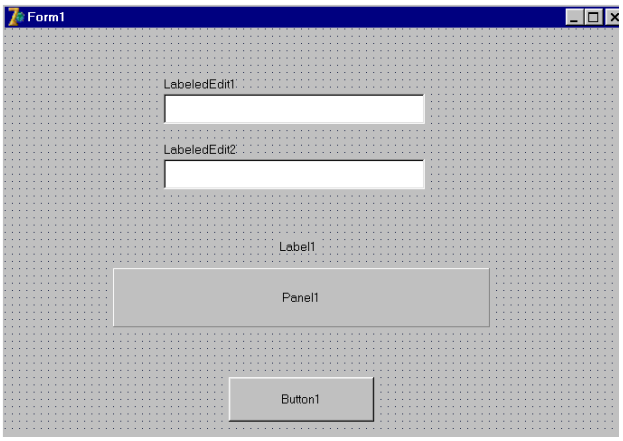


Рис.3.1

2. Измените надписи в метках компонентов LabeledEdit на что-то осмысленное, например, на «Число 1», «Число 2». Для этого щелкните на символе «+» в свойстве EditLabel этих компонентов и измените надпись в свойстве Caption раскрывшихся списков свойств меток. Полезно задать для меток жирный шрифт.
3. Замените свойство Caption вашей кнопки, например, на «Расчет». Очистите свойство Caption у панели. В метке над панелью напишите «Результат». В свойстве Text (текст) окон редактирования задайте «1» — начальное значение текста.

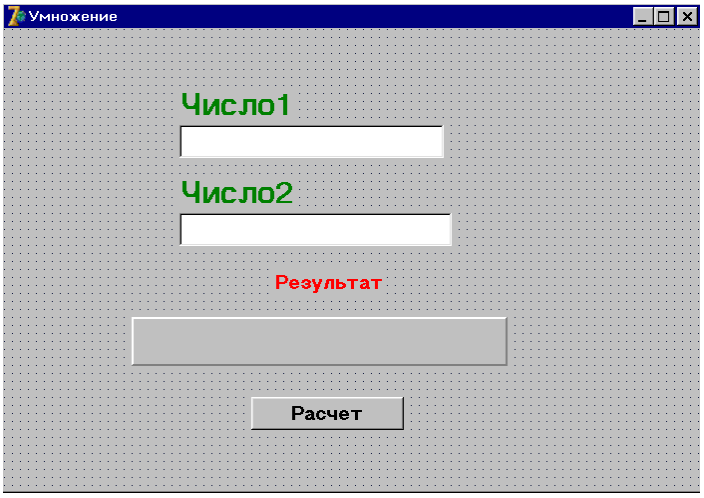


Рис.3.2

4. Попробуйте поварьировать такими свойствами панели, как **BevelInner и BevelOuter,** которые определяют вид (утопленный — **bvLowcred** или выпуклый — **bvRaised**) основного поля и рамки панели. Например, можете установить **BevelInner = bvLowered и BevelOuter = bvRaised.**

В итоге ваша форма приобретет вид, показанный на рис.3.2.

5. Напишите обработчик щелчка кнопки. Единственный оператор этого обработчика может иметь вид:

```
Panel1.Caption := LabeledEdit1.Text +  '*' +  LabeledEdit2.Text  + '=' +
FloatToStr (StrToFloat (LabeledEditl.Text)* StrToFloat(LabeledEdit2.Text) );
```

. Это выражение должно иметь тип строки текста. Начинается строка с текста, введенного пользователем в окно редактирования **LabeledEdit1** — этот текст хранится в свойстве **Text.** Затем вы прибавляете к этому тексту символы ’*’. Знак «+» в выражениях для строк означает конкатенацию — сцепление двух строк символов. Затем аналогичным образом к строке добавляется текст второго окна редактирования и символы « = ». После этого мы хотим вставить в строку результат перемножения двух целых чисел. Этот результат будет числом и, чтобы вставить его в текст, надо сначала преобразовать это число в строку. Эту операцию выполняет функция **FloatToStr,** которая преобразует заданный ей параметр типа действительного числа в строку символов. Осталось получить само произведение двух чисел. Но числа заданы пользователем в виде текстов — строк символов в окнах редактирования. Прежде, чем перемножать, эти строки надо перевести в числа. Эту операцию выполняет функция **StrToFIoat**, преобразующая символьное изображение числа в его значение типа действительного числа. Знак '*', указанный между двумя вызовами функции **StrToFIoat**, обозначает операцию умножения.

6. Ваше приложение готово. Можете его сохранить. Для этого лучше

всего создать отдельный подкаталог (папку Windows) и выполнить команду File | Save All.

7. Откомпилируйте свое приложение и выполните его. Убедитесь, что оно нормально работает.

Задание для самостоятельного выполнения.

1. Создайте приложение, реализующее ввод двух целых чисел, по щелчку на кнопке с символом «=» вычисляющее результат операции вещественного деления и выводящее значение результата на экран. Предусмотреть в программе проверку делителя на равенство нулю.

2. Разработайте проект, в котором при щелчке на кнопках происходит увеличение или уменьшение размера шрифта надписи (рис. 3.3). Исходный размер принять равным 12 пт, шаг изменения — 8 пт. При достижении размера шрифта 52 пт кнопка с надписью Увеличить должна быть отключена, а при уменьшении размера шрифта до 8 пт должна быта отключена вторая кнопка, при этом активизирована первая.

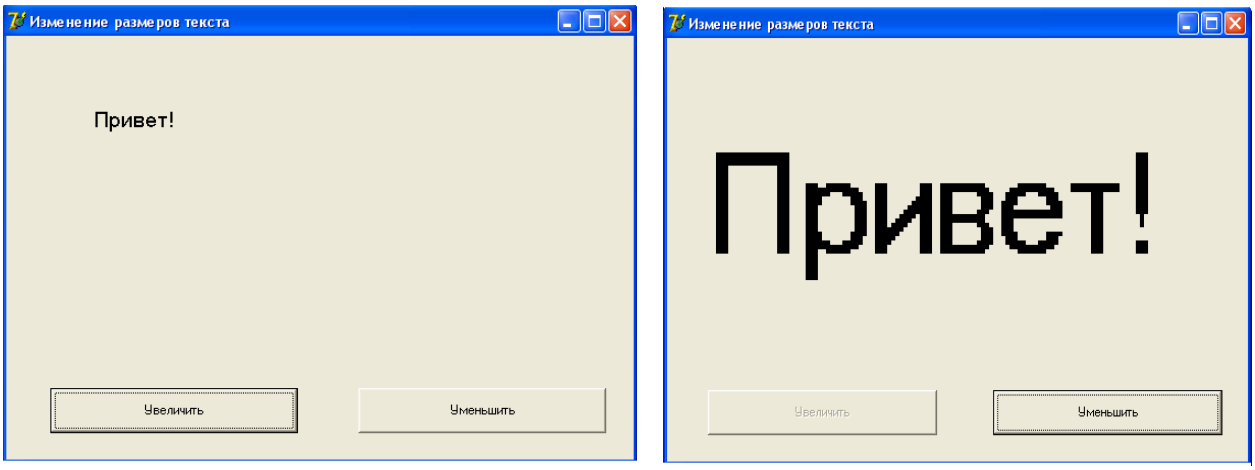


Рис. 3.3

Занятие 4. Поле ввода текстовой информации (класс TEdit). Обработка события OnKeyPress.

Цель: изучить назначение и порядок изменения свойств компонента **Edit,** создать обработчик события нажатия на клавишу в строке ввода компонента **Edit**.

Постановка задачи: Создать программу «Диалог», выполняющую следующие действия. После запуска программы пользователь вводит свое имя, например Артем, в прямоугольник с мигающим текстовым курсором и нажимает клавишу Enter (см. рис.4.1).

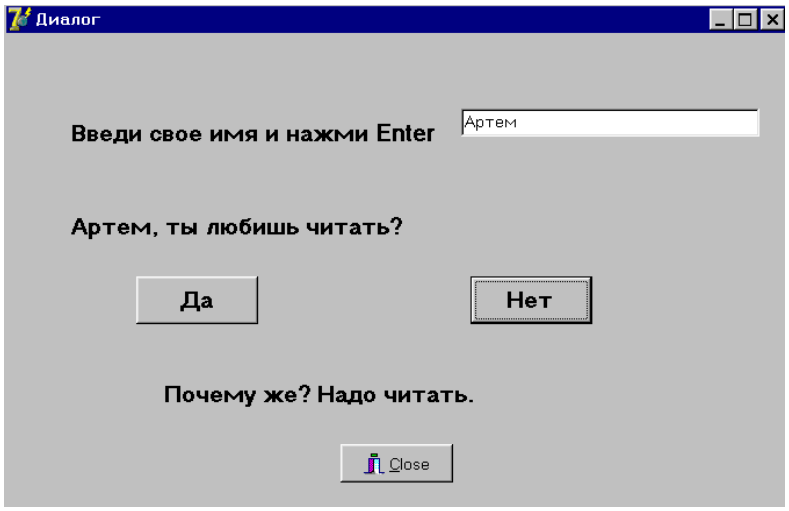


Рис.4.1

Появляется вопрос: «Артем, ты любишь читать?». Если пользователь щелкает на кнопке «Да», то появляется реплика «Молодец!», если на кнопке «Нет», то реплика «Почему же? Надо читать». Для выхода из программы необходимо щелкнуть на кнопке «Выход».

Пояснение. В этой работе использовать строку ввода **Edit** (вкладка палитры компонентов **Standard)** и обработку события **OnKeyPress** - нажатия клавиши.

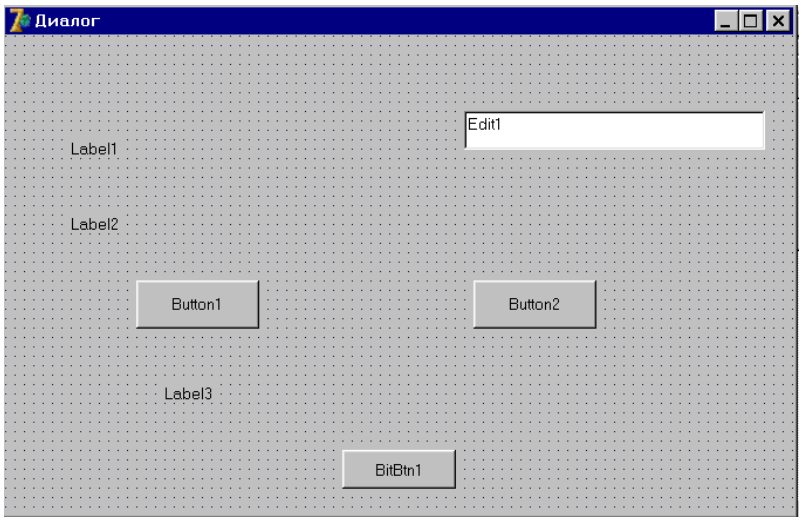


Рис.4.1

План разработки программы

1. Открыть новый проект. Дать форме название «Диалог».
2. Разместить на форме экземпляры компонентов в соответствии с рис.4.2

3. Выполнить следующие действия:

Выделенный объект	Вкладка окна Object Inspector	Имя свойства/ имя события	Действие
BitBtn1	Properties	Caption	Установка имени кнопки «Выход»
		Kind	bk Close
Label1	Properties	Caption	Ввод надписи «Введи свое имя и нажми Enter»

Edit1	Events	OnKeyPress	If key=#l3 then
			begin
			Label2.Caption:=Edit1.Text + ', ты любишь читать?';
			end;

Button1	Properties	Caption	Установка имени кнопки: «Да»
	Events	OnCIick	Label3 .Caption :==' Молодец!'
Button2	Properties	Caption	Установка имени кнопки: «Нет»
	Events	OnCIick	Label3.Caption:='Почему же? Надо читать.';

4. Сохранить код программы и проект под именами, например, un-4.pas и pr-4.dpr.

5. Запустить программу, затем закрыть окно проекта, щелкнув на кнопке «Выход».

6. Сохранить проект, запустить и протестировать его.

Дополнительное задание

1. Сделать кнопки «Да» и «Нет» доступными только после ввода имени и нажатия клавиши **Enter.**

Подсказка. Значение свойства **Enabled** кнопок «Да» и «Нет» установить равным False,

а в процедуру **Edit1KeyPress** включить код

```
Button1.Enabled := true ;
Button2.Enabled := true;
```

2. Удалить имена объектов **Edit1, Label2, Label3** для обеспечения возможности повторить диалог.

Подсказка, Разместить на форме еще одну кнопку **BitBtn.** Установить значение **bkRetry**

свойства **Kind** и значение «Повторить» свойства **Caption**. Ввести программно по нажатию кнопки **«Повторить»** пустые строки вместо надписей и имен кнопок:

```
LabeI2.Caption:=' ' ;
LabeI3.Caption:=' ' ;
Edit1.Text :=’ ’;
```

3. Сделать так, чтобы при повторении диалога строка ввода была снова активной.

Подсказка. Ввести команду

```
Form1.ActiveControl: =Edit1
```

Листинг программы

```
unit Unit2;
interface
uses
   Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms,Dialogs,
   Buttons, StdCtrls;
type
  Tform1 = class(TForm) Edit1: TEdit;
   Button1: TButton;
   Label1: Tlabe1;
   Label2: TLabel;
   Button2: TButton;
   Label3: TLabel;
   BitBtn1: TBitBtn;
   BitBtn2: TBitBtn;
procedure Edit1KeyPress(Sender: Tobject; var Key: Char);
procedure Button1Click(Sender: TObject);
procedure Button2Click(Sender: TObject);
procedure BitBtn2Click(Sender: TObject);
 private
  { Private declarations } public
   { Public declarations }
   end;
var
   Form1: TFom1 ;
implementation
($R *.DFM}
     procedure Tform1.Edit1KeyPress(Sender: TObject; var Key: Char);
  If  key=#13 then begin
```

```
      Label2 .Caption:=Edit1 .Text+' , ты любишь читать?';
      Buttonl. Enabled: ==true;
      Button2.Enabled:=true ;
    end ;
  end;
  procedure Tform1.Button1Click(Sender: TObject);
  begin
    Label3.Caption:='Молодец !';
  end;
  procedure Tfomi1.Button2Click(Sender: TObject);
  begin
    Label3.Caption:='Почему же? Надо читать.';
  end;
  procedure Tform1.BitBtn2Click(Sender: TObject);
  begin
     Edit1.Text:=' ' ;
     Label2. Caption :=' ' ;
     Label3.Caption:=' ' ;
     Button1.Enabled:=false ;
     Button2.Enabled:=false;
    Form1.ActiveControl:=Edit1 ;
  end;
  end.
```

Задание для самостоятельного выполнения.

Разработайте проект для расчета:

1. среднего арифметического и среднего геометрического двух заданных чисел (среднее геометрическое двух чисел равно квадратному корню из их произведения);
2. площади круга и длины окружности по заданному радиусу;
3. длины гипотенузы и площади прямоугольного треугольника по известным катетам;
4. площади поверхности и объема цилиндра по заданным размерам радиуса и высоты цилиндра;
5. пройденного пути, по заданным значениям начальной скорости, ускорения и продолжительности движения;
6. объема и полной поверхности куба с заданной длиной его ребра.

Ввод исходных данных и вывод результатов должны проводиться с помощью компонентов Edit.

Занятие 5. Работа со списком, компонента ListBox

Цель: сформировать навыки применения компонента ListBox для проектирования диалогового окна с элементом управления Список.

Постановка задачи: Создать программу «Справочник», выполняющую следующие действия. После запуска программы пользователь выбирает с помощью мыши или стрелок название цвета и нажимает клавишу Enter. На экране появляется название цвета на русском языке и код цвета в формате **RGB** (рис.5.1). Программа заканчивает свою работу по нажатию клавиши «Выход».

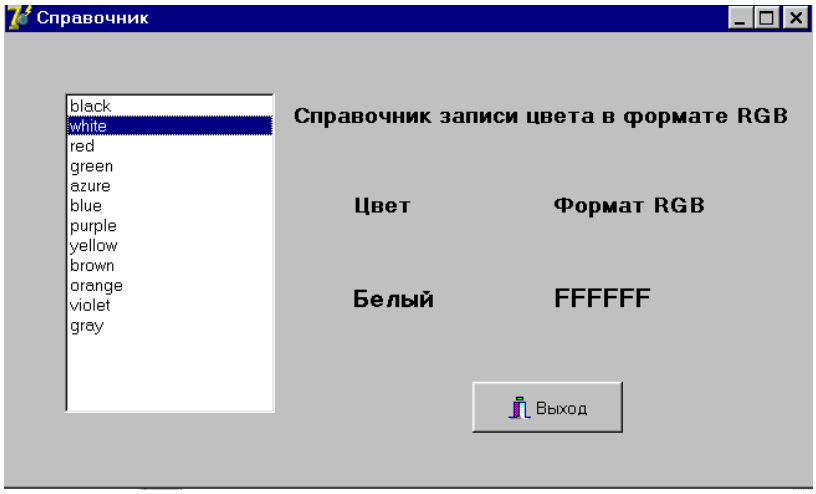


Рис.5.1

Пояснение. Новым в этой работе является использование компоненты **ListBox (список)** (вкладка палитры компонентов **Standard,** работа со встроенным редактором для ввода информации и алгоритм выбора (оператор Case).

План разработки программы

1. Открыть новый проект. Дать форме название «Справочник».
2. Разместить на форме экземпляры компонентов в соответствии с рис. 5.2.

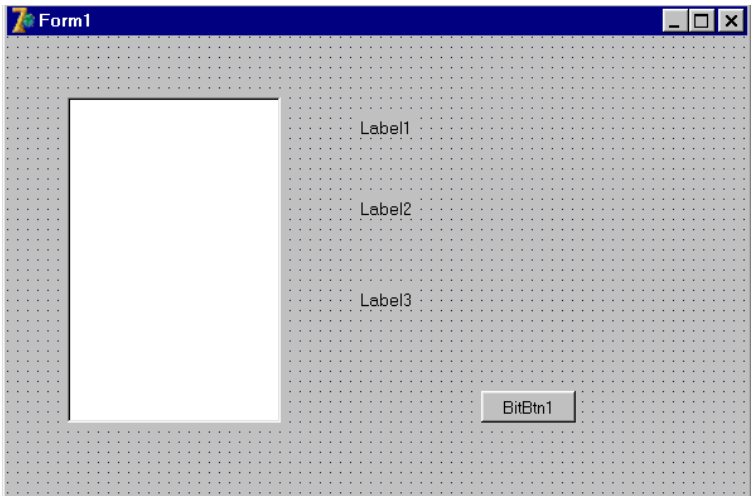


Рис.5.2

3. Выполните следующие действия:

Выделенный объект	Вкладка окна Object Inspector	Имя свойства/ имя события	Действие
BitBtn1	Properties	Caption	Установка имени кнопки: «Выход»
		Kind	bk Close
Label1	Properties	Caption	Ввод надписи «Справочник записи цвета в формате RGB»)
Label2	Properties	Caption	Ввод надписи «Цвет Формат RGB»
Label3	Properties	Caption	Удаление названия и кода цвета

4. Сохранить код программы и проект под именами, например, unit5.pas и pr-5.dpr.

5. Запустить программу, затем закрыть окно проекта, щелкнув на кнопке «Выход».

6. Выделить объект **ListBox1,** найти свойство **Items,** щелкнуть на кнопке с тремя точками, расположенной справа от него. В появившемся окне встроенного редактора ввести названия цветов, каждый на новой строке.

7. Сохранить набранный текст под именем color.txt. Для этого нажать правую кнопку мыши и выбрать режим Save. Для выхода из встроенного редактора щелкнуть на кнопке ОК.

8. Выполнить следующие действия:

Выделенный объект	Вкладка окна Object Inspector	Имя свойства/ имя события	Действие

ListBox1	Events	OnKey Press	If key=#13 then
			case Listbox1.Itemlndex of
			0: Label3.Caption:='черный 000000';
			1: Label3.Caption:='белый FFFFFF';
			2: Label3.Caption :='красный FF0000';
			3:Label3. Caption :='зеленый 00FF00' ;
			4: Label3. Caption:='бирюзовый 00FFFF';
			5: Label3.Caption:='синий 0000FF';
			6: Label3.Caption:='фиолетовый' FF00FF';
			7: Label3.Caption:='желтый FFFF00';
			8: Label3.Caption:='коричневый 996633';
			9: Label3.Caption:='оранжевый FF8000' ;
			10: Labe13.Caption:='лиловый 8000FF' ;
			11: Label3. Caption:=' серый A0A0A0' ;
			end;

9. Сохранить проект окончательно, запустить и протестировать его.

Задание для самостоятельного выполнения

1. Изменить шрифты, цвет экрана и букв.

Подсказка. Возможно, придется в коде программы подкорректировать количество пробелов между названием цвета и его кодом,

2. Сделать так, чтобы при установке курсора мыши в поле **ListBox1** появлялась подсказка о том, что надо сделать.

Подсказка. Воспользуйтесь свойствами **Hint** (текст сообщения), **Showhint** (показывать ли сообщение) объекта. **ListBox1.**

3. Внести изменения в программу, чтобы для надписей «Цвет» и «формат RGB» использовались два отдельных объекта Label.

Листинг программы

```
unit Unit5;
interface
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls/ Forms, Dialogs,
  StdCtrls, Buttons;
Type
   TFormI = class(TForm)
      ListBox1: TListBox;
      Label1: TLabel;
      Label2: TLabel;
      Label3: TLabel;
      Label4: TLabel ;
      BitBtn1: TBitBtn;
   procedure ListBox1KeyPress(Sender: TObject; var Key: Char);
private
  { Private declarations }
public
  { public declarations }
end;
var
  Form1: Tform1 ;
  Implementation
   {$R *.DFM}

  procedure Tform1 .ListBox1KeyPress (Sender: TObject; varKey: Char) ;
   const L3: array[0..11] of string[10]= ('черный', 'белый', 'красный',
             'зеленый', 'бирюзовый', 'синий', 'фиолетовый', 'желтый',
             'коричневый', 'оранжевый', 'лиловый', 'серый') ;
         L4: array[0..11] of string[10]= ('000000',
             'FFFFFF', 'FF0000', '00FF00', '00FFFF',
             '0000FF', 'FF00FF', 'FFFF00', '996633',
             'FF8000', '8000FF', 'A0A0A0') ;
begin
```

```
    if Key=#13 then
    begin
      Label3.Caption:= L3 [ListBox1. ItemIndex] ;
       Label4.Caption: = L4 [ListBox1. ItemIndex]
end;
end;
end.
```

4. Разработайте проект для перерасчета массы в граммах в массу в фунтах, пудах, унциях, драхмах или гранах (1 фунт = 454 г, 1 пуд = 16 380 г, 1 унция =28,35 г, 1 драхма = 1,772 г, 1 гран = 0,0648 г). Выбор новой единицы измерения должен проводиться с помощью компонента ListBox, ввод исходных данных и вывод результата — с помощью компонента Edit (рис.5.3).

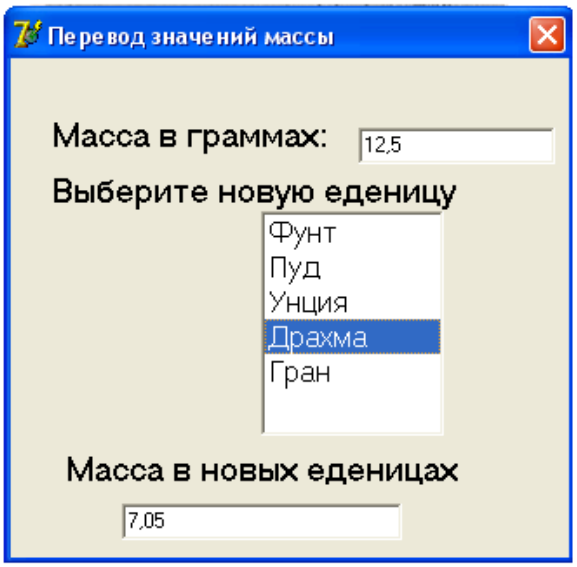


Рис 5.3

Занятие 6. Комбинированный список ComboBox и его свойства, выключатель CheckBox, функция MessageDlg

Цель: сформировать навыки применения компонентов **ComboBox** и **CheckBox,** функции **MessageDlg** для проектирования диалогового окна с элементами управления Комбинированный список и Выключатель.

Постановка задачи:

Разработать программу **«Словарь»** для составления словаря новых терминов. Должна иметься возможность внесения в словарь изменений, дополнения или сокращения его. При выборе термина на контрольной панели должны появляться его номер и общее количество слов в словаре (рис.6.1)

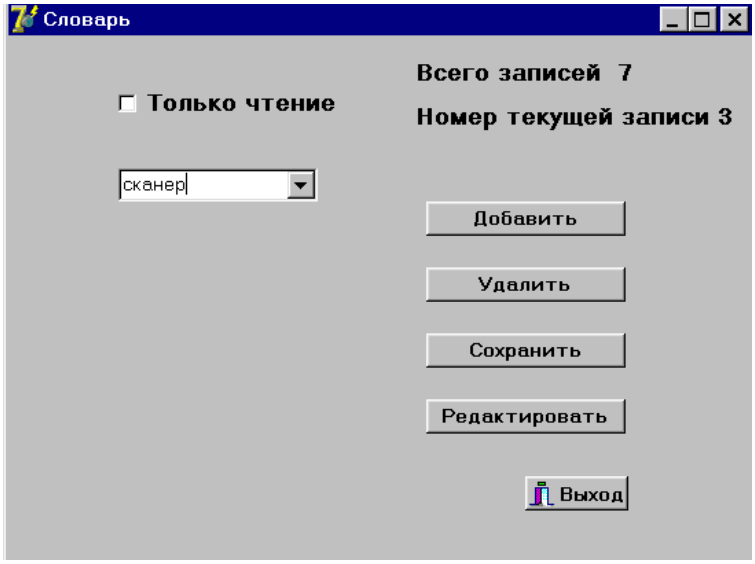


Рис.6.1

Пояснение. Новым в этой работе является использование комбинированного списка **ComboBox** (представляющего собой объединение строки ввода и компонента **ListBox),** компонента - выключателя **CheckBox,** а также функции **MessageDlg.**

План разработки программы

1. Открыть новый проект.

2. Разместить на форме экземпляры компонентов в соответствии с рис. 6.2

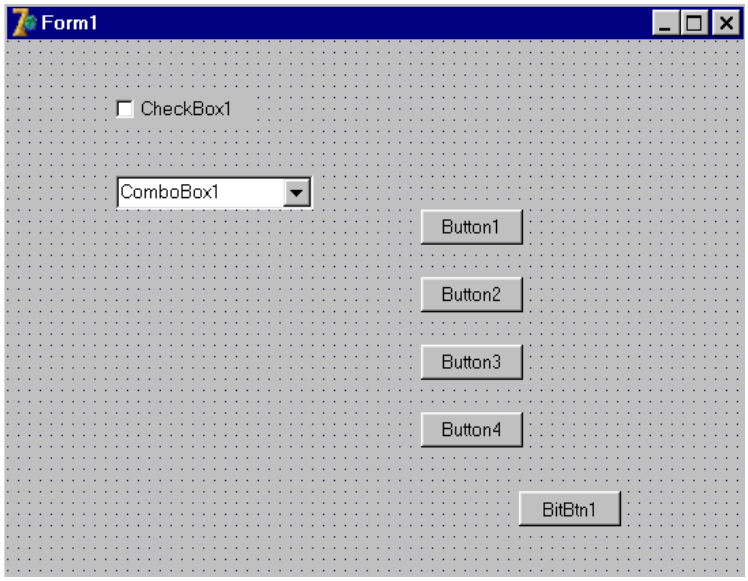


Рис.6.2

3. Выполнить следующие действия:

Выделенный объект	Вкладка окна Object Inspector	Имя свойства/ имя события	Действие
BitBtnl	Properties	Caption	Установка заголовка кнопки: «Выход»
		Kind	bkClose
Combo Boxl	Properties	Text	Вставка пробела
		Items	Открытие двойным щелчком списка **String list editor**. Ввод нескольких терминов, сохранение их в файле glostext.txt, предварительно убедившись, что выбрана нужная директория: (см. занятие 5)

4. Сохранить код программы и проект под именами, например, unit6.pas и pr6.dpr.

5. Запустить программу, затем закрыть окно проекта, щелкнув на кнопке «Выход».

6. Запустить программу, щелкнуть на стрелке объекта **ComboBoxl**, выбрать из открывшегося списка любой термин, а затем убедиться, что текст**,** набираемый в строке ввода, в список не вносится. После этого закрыть проект, щелкнув на кнопке «Выход».

7. Выполнить следующие действия:

Выделенный объект	Вкладка окна Object Inspector	Имя свойства/ имя события	Действие
Button1	Properties	Caption	Установка заголовка кнопки: «Добавить»
	Events	OnClick	ComboBoxl.Items.Add (ComboBoxl.Text); if ComboBoxl.ItemIndex = -1 then ComboBoxl .Text := ' '; *Примечание.* Последний оператор очищает строку ввода после того, как текст из нее попадает в список

8. Запустить программу и убедиться, что набранные в строке ввода слова после нажатия кнопки «Добавить» добавляются в список **ComboBox1.**

9. Выполнить следующие действия:

Выделенный объект	Вкладка окна Object Inspector	Имя свойства/ имя события	Действие
Button2	Properties	Caption	Установка заголовка кнопки: «Удалить»
	Events	OnClick	if MessageDlg('Вы действительно тите удалить запись?', mtWarning, [mbYes, mbNo], 0) = mrYes then ComboBox1.Items .Delete (ComboBoxl. ItemIndex) ;

Пояснение. Условие if MessageDlg ... означает, что при нажатии кнопки «Удалить» появится диалоговое окно с вопросом «Вы действительно хотите удалить запись?» с двумя кнопками «Yes» и «NO». Предварительно выбранный термин будет удален из списка только при нажатии кнопки «Yes».

MessageDlg является функцией с четырьмя аргументами:

1) сообщение, которое нужно отобразить, набирается в одинарных кавычках;
2) константы **mtWarning, mtError, mtlnformation, mtConformation** определяют специальную пиктограмму и заголовок диалогового окна;
3) список констант в квадратных скобках определяют присутствующие в окне кнопки. Возможные значения **— mbYes, mbNo, mbCancel,**

mbHelp;

4) число, с которым ассоциирована тема справки, отлично от нуля, если в окне присутствует кнопка **mbHelp,** а программа имеет связанный с ней файл справки Windows.

Функция **MessageDlg** возвращает константу, определяющую нажатую пользователем кнопку.

10. Запустить программу, добавить в список и (или) удалить несколько строк, закрыть ее, а затем снова запустить. Заметим, что внесенные изменения не сохранились. Это связано с тем, что в **ComboBox** загружается первоначальный список.

11. Выполнить следующие действия:

Выделенный объект	Вкладка окна Object Inspector	Имя свойства/ имя события	Действие
Button4	Properties	Caption	Установка заголовка кнопки: «Сохранить»
	Events	OnClick	ComboBox1.Items.SaveToFile ('glostext.txt') ;
Form1	Events	OnCreate	ComboBoxl.Items.LoadFromFile ('glostext.txt');

Пояснение. Теперь при создании формы в список **ComboBox1** будет загружаться содержимое файла glostext.txt, а измененный список запомнится в нем при нажатии кнопки «Сохранить».

12. Выполнить следующие действия:

Выделенный объект	Вкладка окна Object Inspector	Имя свойства/ имя события	Действие
Button3	Properties	Caption	Установка заголовка кнопки: «Редактировать»
ComboBox1	Events	OnClick	num:= ComboBoxl.ItemIndex; Предварительно надо описать в разделе Var целочисленную
Button3	Events	OnClick	ComboBox1.Items.Delete (num); ComboBox1.Items.Add (CortboBox1. Text) ;

Пояснение.

Для исправления ошибок, замеченных в строках списка, выделенную строку надо удалить из списка, а исправленную строку (т.е. содержимое

строки ввода) добавить в список.

Новая переменная *num* необходима для сохранения номера выбранной строки. При внесении изменений выбранной строкой становится строка ввода, для которой **ItemIndex = —1.**

13**.** Для кнопок «Добавить», «Редактировать», «Удалить» для сохранения внесенных изменений необходимо выполнить следующие действия:

Вкладка окна Object Inspector	Имя свойства/ имя события	Действие
Events	OnClick	ComboBox1.Items.SaveToFile ('glostext.txt');

14. Запустить программу и убедиться, что если выбрать строку, исправив ее, а затем щелкнуть на кнопке «Редактировать», то измененная строка попадет в конец списка.

15. Предусмотреть режим работы со списком, допускающий только чтение. Для этого выполнить следующие действия:

Выделенный объект	Вкладка окна Object Inspector	Имя свойства/ имя события	Действие
CheckBox1	Propeties	Caption	Установка заголовка кнопки: «Только чтение»
	Events	OnClick	if CheckBox1.Checked = False then ComboBox1. Style:= csDropDown else ComboBox1.Style: = csDropDownList; *Примечание.* Свойство **Checked** у выбранного выключателя имеет значение **True**, а в исходном состоянии — значение **False**

Пояснение. Комбинированные списки бывают трех типов:

• простые (simple) — список всегда открыт;

• раскрывающиеся (drop down) — список свернут;

• неизменяемые (drop down list) — список свернут и невозможна коррекция списка.

16. В режиме «Только чтение» сделать недоступной кнопку «Добавить»:

Выделенный объект	Вкладка окна Object	Имя свойства/ имя события	Действие
CheckBox1	Events	OnClick	if CheckBox1.Checked = false then Button1.Enabled := true else Button1.Enabled := false;

Аналогично сделать недоступными кнопки «Редактировать», «Удалить», «Сохранить».

17. Сохранить проект, запустить и протестировать его.

Задание для самостоятельного выполнения

1. Предусмотреть возможность добавления терминов в список не только при щелчке на кнопке «Добавить», но и при нажатии клавиши Enter.

Подсказка. Для объекта **ComboBox1** в обработчик события **onKeyPress** вставить код:

if Key = #13 then ... ;

2. Добавить две метки и вывести на них общее количество терминов в списке и номер выбранного термина.

Подсказка. Для объекта **ComboBox1** в обработчик события **Change** вставить код, присваивающий заголовку панели значение **ComboBox1.Items.Count,** а в обработчик события **Click** — код, присваивающий заголовку другой панели значение **ComboBox1.ItemIndex+1,** имея в виду, что первый термин имеет индекс 0, второй - 1 и т. д.

3. Обеспечить сохранение всех изменений словаря при закрытии формы.

Подсказка. Для объекта **Form** в обработчик события **CloseQuerry** вставить код, сохраняющий измененный список в файле glostext.txt.

4. Отсортировать список.

Подсказка. Свойству **Sorted** объекта **ComboBox1** присвоить значение **True.**

Листинг программы

```
unit Unit6;
interface
  private
    {Private declarations }
  public
      {Public declarations }
  end;
  var
   Form1: TForm1;
  Num: Integer;
implementation
{$R *.DFM}
procedure TForm1.BitBtn5Click(Sender: TObject) ;
begin
  Close ;
end;
procedure TForml.BitBtnlClick(Sender: TObject) ;
begin
   If ComboBox1.Text<>' ' then
 ComboBox1.Items.Add(ComboBox1.Text);
   If Combobox1.ltemIndex = -1 then ComboBox1.Text:=' ';
end;
procedure TFormI.BitBtn2Click(Sender:TObject);
begin
    If  MessageDlg('Вы действительно хотите удалить запись ?',
    MtWarning,[mbYes,mbNo] ,   0) = mrYes  Then
ComboBox1.Items. Delete (ComboBox1.ItemIndex) ;
end;
procedure Tform1.BitBtn4Click(Sender: TObject) ;
begin
  ComboBox1.Items.SaveToFile ('glostext. txt') ;
 end;
procedure Tform1. FormCreate (Sender: TObject) ;
 begin
  ComboBox1.Items.LoadFromFile(‘glostext.txt' ) ;
 Label1.Caption:='Всего записей :’ + intToStr(ComboBox1.Items.Count);
end;
procedure TForm1.BitBtn3Click(;(Sender: TObject) ;
begin
  ComboBox1.Items.Delete(Num) ;
   ComboBox1. Items. Add (ComboBox1.Text) ;
   If ComboBoxl .ItemIndex = -l  Then ComboBoxl.Text:=' ';
end;
procedure TForm1.ComboBox1Click (Sender: TObject) ;
```

```
begin
  Num:=ComboBoxl.ItemIndex;
  Label2.Caption:= 'Номер выбранного:' +
  IntToStr(ComboBox1.ItemIndex+1) ;
end;
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  If CheckBox1.Checked=Falae  then
       begin
         ComboBox1.Style:=csDropDown;
          BitBtn1.Enabled:=True;
          BitBtn2.Enabled:=True;
          BitBtn3.Enabled:=True ;
          BitBtn4.Enabled:=True;
      end
  Else
    begin
         ComboBox1.Style:=csDropDownList;
          BitBtn1.Enabled:=False;
          BitBtn2.Enabled:=False;
         BitBtn3.Enabled:=False;
         BitBtn4.Enabled:=False;
    end; end;
procedure TForm1.ComboBox1KeyPress(Sender: TObject; var Key: Char);
   begin
     If Key=#13 Then BitBtn1Click(Sender) ;
  end;
procedure TForm1.ComboBox1Change(Sender: TObject);
  begin
    Label1.Caption:= 'Всего записей : ' + IntToStr(ComboBox1.Items.Count);
  end;
procedure TForm1. FormCloseQuery (Sender:TObject; var CanClose: Boolean) ;
  begin
    ComboBox1.Items.SaveToFile('glostext.txt') ;
end;
end.
```

Занятие 7. Формирование цвета из отдельных компонент. Класс TColor, функции преобразования значений цветовых составляющих *TColorRef* и *RGB*. Компонент для ввода данных - полоса прокрутки ScrollBar.

Цель: сформировать навыки применения компонента ***полоса прокрутки*** **ScrollBar.,** функций преобразования значений цветовых составляющих ***TColorRef*** и ***RGB.***

Постановка задачи:

Создать программу «Цвета в формате RGB», с помощью которой пользователь мог бы увидеть в зависимости от значений насыщенности красного, зеленого и синего цветов результирующий цвет (рис.7.1).

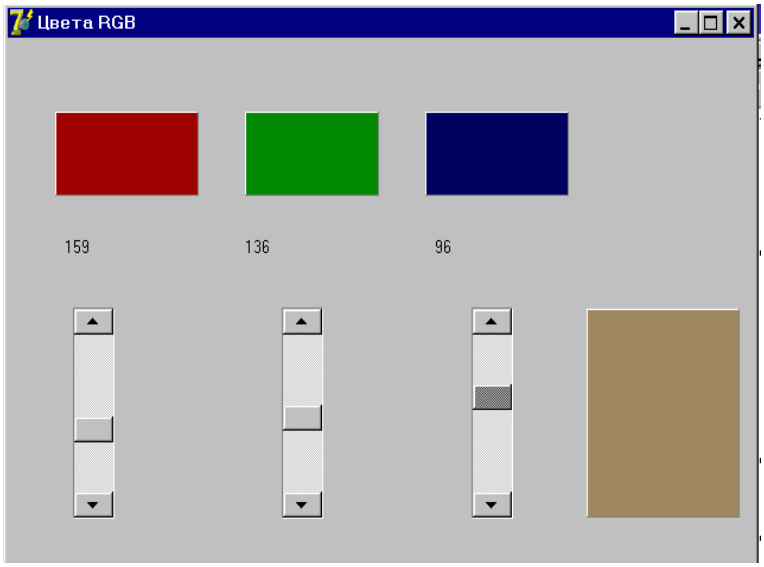


Рис.7.1

Пояснение. Новым в этой работе будет использование:
- полос прокрутки **ScrollBar** для ввода данных,
- функции преобразования значений цветовых составляющих ***TColorRef.***

Цвета объектов образуются смешением трех компонент — красной (red), зеленой (green) и синей (blue). Интенсивность каждой составляющей цвета может изменяться от 0 до 255. Комбинация (0, 0, 0) соответствует черному, а (255, 255, 255) — белому цвету. Почти у каждого визуального компонента есть свойство Color. До сих пор мы выбирали его значение из списка стандартных цветов, но можно создать цвет из отдельных компонент. Для этого воспользуемся функцией **RGB**: Color:= RGB (red, green, blue). Для подбора цвета разработаем проект, позволяющий легко изменять цвет панели с помощью полос прокрутки — объектов класса TScrollBar. Каждая полоса прокрутки будет отвечать за интенсивность одной из трех цветовых компонент. Крайнее левое положение ползунка должно соответствовать

минимальному, а крайнее правое — максимальному значению интенсивности.

План разработки программы

1. Открыть новый проект.

2. Разместить на форме экземпляры компонентов в соответствии с рис.7.2.

3.Полоса прокрутки **ScrollBar** может быть горизонтальной (по умолчанию) или вертикальной. Это определяется свойством **Kind**. В нашем случае используется вертикальная полоса прокрутки. Свойства **LargeChange**—шаг перемещения ползунка при щелчке на самой полосе; **SmallChange** — шаг перемещения ползунка при щелчке на стрелке; **Position** — числовой эквивалент положения ползунка на полосе скроллинга. Основное событие для полосы прокрутки — перемещение ползунка (событие **OnChange).**

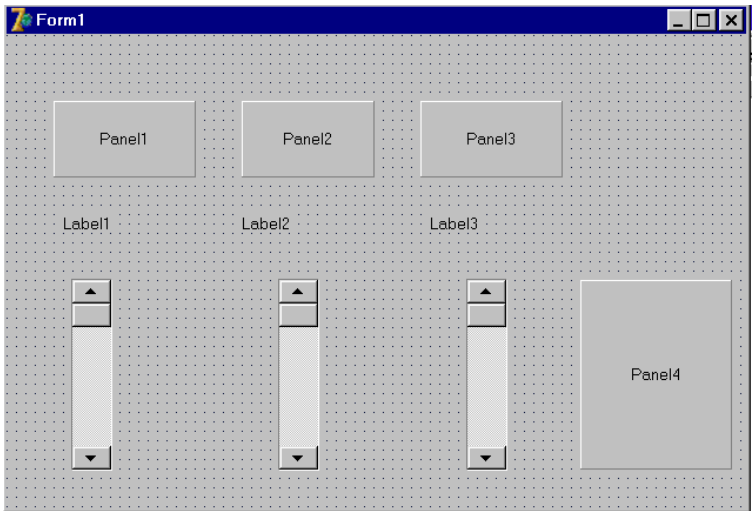


Рис.7.2

4. Сохранить код программы и проект под именами, например, unit7.pas и pr7.dpr.

5. Выполнить следующие действия:

Выделенный объект	Вкладка окна Object Inspector	Имя свойства	Действие
ScrollBar1	Properties	Name	Установка имени полосы прокрутки «RedBar», под которым компонент будет известен программе.
		Max	Установка максимального количества градаций компонент RGB: 255
		Position	Установка начального значения: 122

Аналогично задать значения для **ScrollBar2 и ScrollBar3,** присвоив им имена «GreenBar» и «BlueBar».

6. Для всех компонентов формы установите значение свойства Caption ' '.

7. Выполнить следующие действия:

Выделенный объект	Вкладка окна Object Inspector	Имя свойства/ события	Действие
RedBar	Events	OnChange	Panel1.Color:=TColorRef(RGB(RedBar.Position,0,0)) ; Label.Caption:=IntToStr(RedBar.Position) ; Panel4.Color:= TcolorRef(RGB(RedBar.Position, GreenBar.Position, BlueBar.Position));

Аналогично задать значения для **ScroIlBar2 и ScrolIBar3,** проследить за правильностью записи параметров функций *RGB* и *IntToStr*.

Пояснение. В зависимости от передвижения ползунка **ScrollBar1** будет меняться цвет **Panel1,** выводиться числовое значение кода на месте Label1 и меняться цвет **Panel4.**

8. Сохранить проект, запустить и протестировать его.

Задание для самостоятельного выполнения

1. Поместить на форму кнопку выхода из программы.

2. Предусмотреть, чтобы после запуска программы устанавливались начальные цвета панелей в зависимости от исходных значений ползунков.

Подсказка. Предусмотреть в реакции на событие OnCreate для формы Form1 обработку значений ScrollBar1, ScrollBar2 и ScrollBar3.

3. Внести в программу изменения, чтобы значение кода цвета выводилось не только в десятичной, но и в шестнадцатеричной системе счисления.

Листинг программы

```
unit Unit7;
interface
private
   {Private declarations)
public
   {Public declarations}
end;
var
Form1: TForm1;
implementation
{$R *.DFM}
   procedure TForm1 .RedBarChange(Sender: TObject);
begin
Panel1.Color:=RGB(RedBar.Position,0,0);
 Label1.Caption:=Int1oStr(RedBar.Position);
 Label4.Caption:=Format('0x%x', [RedBar.Position]) ;
```

```
 Panel4.Color:=RGB(RedBar.Position,GreenBar.Position,BlueBar.Position);
end;
 procedure TForm1.GreenBarChange(Sender: TObject) ;
begin
Panel2.Color:=RGB(0,GreenBar.Position,0);
Label2.Caption:=IntToStr(GreenBar.Position);
Label5. Caption: format ( 0x%x' , [GreenBar .Position] ) ;
Panel4.Color:=RGB(RedBar.Position,GreenBar.Position,BlueBar.Position) ;
end;
procedure TForm1. BlueBarChange (Sender: TObject) ;
begin
Panel3.Color:=RGB(0,0,BlueBar.Position);
Label3.Caption:=IntToStr(BlueBar.Position);
Label6.Caption:=Format('0x%x',[BlueBar.Position]) ;
Panel4.Color:=RGB(RedBar.Position,GreenBar.Position, BlueBar.Position);
end;
procedure TForm1.FormCreate(Sender: TObject);
 begin
Label4.Caption:=Fomat('0x%x',[RedBar.Position]) ;
Label5.Caption:=Format('0x%x',[GreenBar.Position]) ;
Label6.Caption:=Format('0x%x',[BlueBar.Position]) ;
Panel1.Color:=RGB(RedBar.Position,0,0) ;
Labell.Caption:=IntToStr(RedBar.Position) ;
Panel2.Color:=RGB(0,GreenBar.Position,0) ;
Label2.Caption:=IntToStr(GreenBar.Position) ;
Panel3.Color:=RGB(0,0,BlueBar.Position);
Label3.Caption: =IntToStr (BlueBar. Position) ;
Panel4.Color:=RGB(RedBar.Position,GreenBar.Position,BlueBar.Position) ;
end;
   end.
```

4.Разработайте проект, в котором для полосы прокрутки с помощью контекстного меню можно узнать максимальное и минимальное значения, а также значение, соответствующее текущему положению ползунка (рис. 7.3).

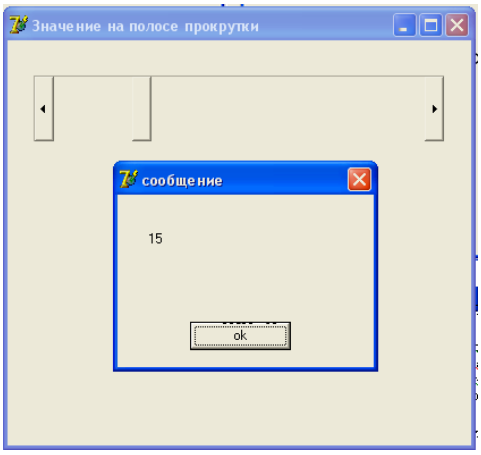


Рис.7.3

Занятие 8. Компонент для выбора варианта. Класс TRadioGroup. - переключатели. Оператор вывода сообщений ShowMessage

Цель: изучить назначение компонента **RadioGroup** и способ определения выбранного переключателя.

Постановка задачи. Разработать программу «Оптимист-пессимист», с помощью которой пользователь мог бы выполнить следующее. После запуска программы появляется изображение, аналогичное рис.8.1. Пользователь по своему усмотрению выбирает один переключатель в группе. Каждому переключателю соответствует определенный балл. В зависимости от суммы набранных баллов появляется одно из сообщений «Вы пессимист», «Вы реалист» или «Вы оптимист».

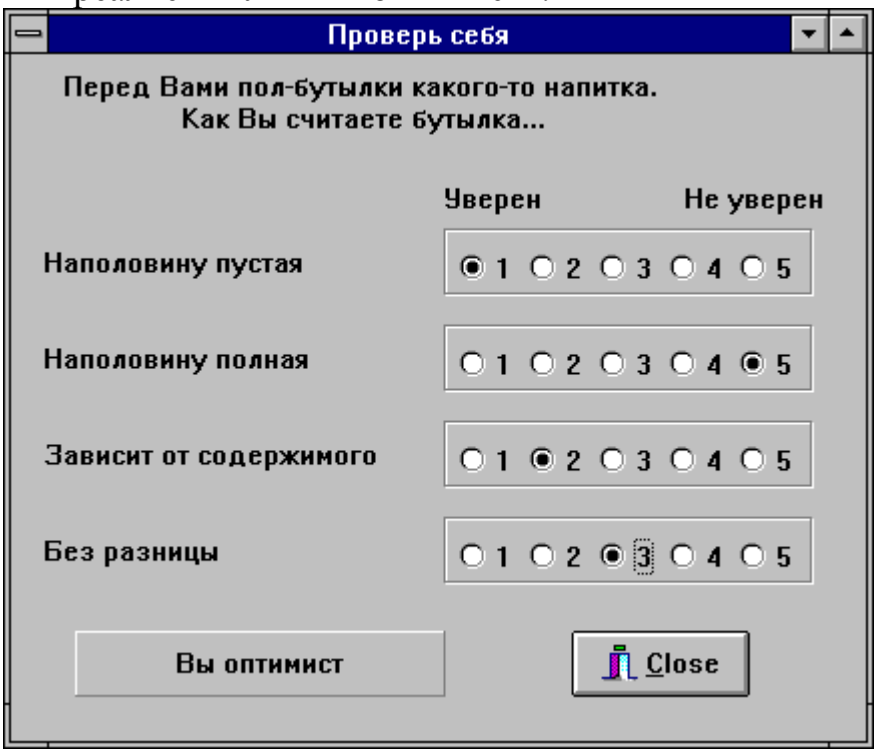


Рис.8.1

Пояснение. Новыми в этом проекте являются группа переключателей RadioGroup и оператор вывода сообщений ShowMessage.

План разработки программы:

1. Разместить на форме компоненты согласно рис.8.2 и присвоить заголовки меткам и панелям.

2. Выделить компонент RadioGroup1, найти в Инспекторе объектов свойство Caption и удалить заголовок. Свойству Columns, определяющему количество колонок, в которые будут отображаться переключатели, присвоить значение 5. Вызвать String List Editor, дважды щелкнув мышкой

рядом со свойством Items, ввести 5 строк со значениями 1, 2, 3, 4 и 5 соответственно. Сохранить эти строки под именем, например, t1.txt .

3. Аналогичные действия проделать с остальными компонентами RadioGroup.

4. Чтобы суммировать набираемые пользователем баллы, в обработчик события RadioGroup1.OnClick вставить код:

```
sum:=0;
with RadioGroup1 do
  if ItemIndex>=0 then sum:=sum+ItemIndex+1;
```

Единицу необходимо прибавлять, так как индекс первого переключателя равен 0, но соответствует 1 баллу. Целочисленную переменную sum необходимо описать в разделе var (выше слова Implementation).

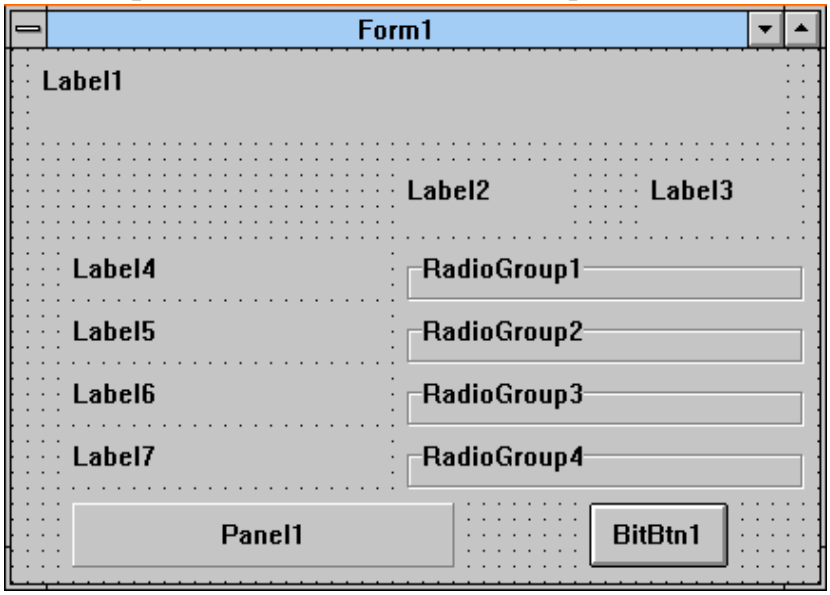


Рис 8.2

5.Вставить в обработчики событий RadioGroup2.OnClick, RadioGroup3.OnClick и RadioGroup4.OnClick аналогичные коды, но без обнуления переменной sum, так как оно необходимо лишь один раз перед началом суммирования.

6. Выведем на контрольную панель итоговое сообщение в зависимости от набранной суммы баллов. Для этого в обработчик события RadioGroup4.OnClick добавим код

```
case sum of
   4..9:    Panel1.Caption:='Вы пессимист';
   10..15: Panel1.Caption:='Вы реалист';
   16..20: Panel1.Caption:='Вы оптимист';
  end;
```

7. Вывести сообщение об окончании тестирования, добавив в обработчик события RadioGroup4Click код

```
ShowMessage('Конец теста');
```

В результате выполнения этого оператора появится информационное окно со словами ‘Конец теста’ и единственной кнопкой Ok .

Задание для самостоятельного выполнения.

1. После нажатия кнопки Ok в информационном окне сделать недоступными все RadioGroup, а все переключатели в них - невыбранными.

Подсказка. Свойствам Enabled всех RadioGroup присвоить значение False, а свойствам ItemIndex - значение -1.

2. Для контроля правильности работы программы вывести на панель набранную пользователем сумму баллов.

Подсказка. Заголовку соответствующей панели присвоить значение IntToStr(sum).

3. Запустить программу и убедиться, что верная сумма баллов получается лишь при последовательном выборе переключателей сначала из RadioGroup1, затем из RadioGroup2 и т.д. Если порядок был нарушен, или пользователь, изменив решение, выбрал другой переключатель в одной и той же группе, то результат будет неверным. Чтобы этого не случалось, сделать доступной только ту группу переключателей, в которой необходимо сделать выбор.

Подсказка. Первоначально свойству Enabled всех RadioGroup, кроме RadioGroup1, присвоить значение False. В обработчик события RadioGroup1Click добавить код, присваивающий свойству Enabled RadioGroup2 значение True и т.д.

4. Сделать возможным повторный запуск программы.

Подсказка. Разместить на форме кнопку Button1, свойству Caption которой присвоить значение Retry, а в обработчик события Button1Click вставить код, делающий доступной RadioGroup1.

.

5. Разработайте проект, в котором вид фигуры в компоненте **Shape** (окружность, квадрат, прямоугольник или эллипс) выбирается с помощью компонентов **RadioButton** (рис. 8.3).

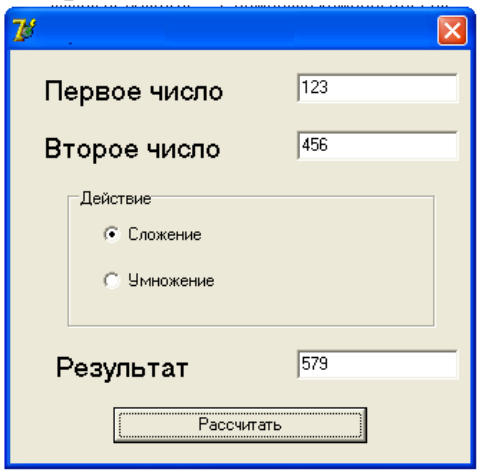


Рис. 8.3

Занятие 9. Класс TMainMenu. Реакция программы на выбор пункта меню.

Цель: научиться проектировать приложения с главным и всплывающим (локальным) меню.

Постановка задачи:

Разработать программу «Меню» выполнения арифметической операции над двумя числами, в которой действие над числами пользователь указывает через соответствующий пункт меню. В программе предусмотреть сообщение об ошибке при вводе делителя равного нулю.

Теоретические сведения:

Меню как элемент интерфейса используется практически во всех серьезных приложениях. В Windows поддерживаются два типа меню — строчное, которому соответствует компонент MainMenu, и всплывающее (или локальное) — ему соответствует компонент PopupMenu. Обычно эти типы используются в комбинации.

MainMenu позволяет поместить главное меню в программу. При размещении на форме этот компонент выглядит как иконка. Создание меню включает три шага:

—помещение MainMenu на Форму;

—вызов Конструктора меню двойным щелчком по значку (или через свойство items в Инспекторе объектов);

—определение пунктов меню.

Каждый пункт меню имеет имя (по умолчанию N1, N2 и т.д.) и название (свойство Caption). Но Конструктор меню так устроен, что сначала нужно ввести название пункта, только после этого пункт меню создается реально.

После создания первого пункта есть две возможности: двигаться вниз, создавая раскрывающийся список подпунктов, или двигаться вправо, создавая следующий пункт главного меню.

PopupMenu позволяет создавать всплывающее меню, которое появляется по щелчку правой кнопки мыши на объекте, к которому оно привязано. У всех видимых объектов имеется свойство PopupMenu, где и указывается нужное меню. Создается PopupMenu аналогично главному меню.

Пункты главного меню располагаются под заголовком Формы

План разработки программы:

1. Разместить на форме компоненты согласно рис.9.1 и присвоить заголовки меткам. Маркерами выделен объект MainMenu1. Этот объект может размещаться в любом месте формы, так как он невидим при работе программы. Такие объекты называются «невизуальными» компонентами.

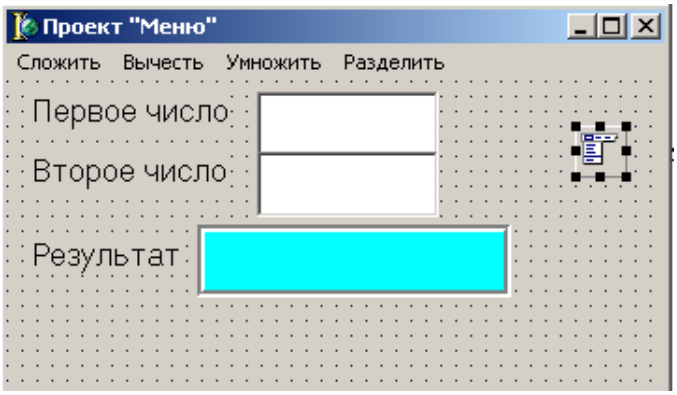


Рис.9.1

2. Чтобы программа реагировала на выбор пункта меню, следует написать для каждого "конечного" пункта меню процедуру обработки данного события. Таким образом, придется написать четыре почти одинаковые процедуры, так как каждая должна получить числовое значение полей ввода, произвести операцию над этими значениями и вывести результат. Отличаться эти процедуры будут только выполняемыми операциями.

3. Напишем отдельную процедуру, которую будем вызывать из всех четырех пунктов меню. В качестве параметра будем передавать ей символ операции: "+", "-", "*" или "/".

```
procedure Exec(ch:  char);
 var   A,   B,   C:   real;
       code:  integer;
       S:  string;
Begin
  Val(Form1Edit1.Text,  A,  code);
  Val(Forml.Edit2.Text,  B,  code);
 Case  ch of
 '+': C:=A+B;
 '-': C:=A-B;
  '*': C:=A*B;
  '/': C:=A/B;
 end;
 Str(C:8:2, S);
Form1.Panel1.Caption := S
end;
```

4.Тогда "обработчик" пункта меню "Сложить" будет выглядеть так:

```
procedure Tform1.N1Click(Sender:  TObject);
```

```
  begin
Exec('+')
end;
```

Аналогично пишутся обработчики других пунктов меню. Запустите программу и поработайте с ней:

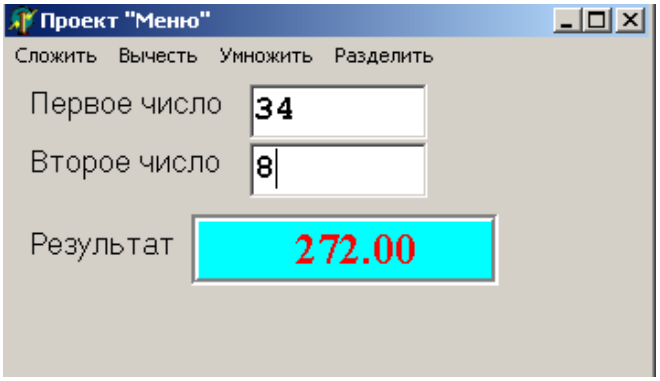


Рис. 9.2

Все идет хорошо, пока мы не решим поделить на ноль. Конечно, все знают, что этого нельзя делать, но интересно: что получится? Появится сообщение о делении на ноль

(Проект *Menu.exe* вызвал исключение "Деление на ноль". Процесс остановлен. Используйте Step или Run для продолжения.)

Чтобы избежать возникновения исключительной ситуации, добавим в оператор выбора проверку второго числа при делении. Если оно равно нулю, то на панель поместим слово "Ошибка". Окончательный текст процедуры станет таким:

```
procedure Exec(ch:  char);
label m99;
 var   A,   B,   C:   real;
       code:  integer;
       S:  string;
Begin
  Val(Form1Edit1.Text,  A,  code);
  Val(Forml.Edit2.Text,  B,  code);
 Case  ch of
 '+': C:=A+B;
 '-': C:=A-B;
  '*': C:=A*B;
   '/': if  B <>  0 then C:=A/B else begin
                                      S := 'Ошибка!' ; goto m99
                                     end;
```

```
      end;
      Str(C:8:2, S);
m99: Form1.Panel1.Caption := S
 end;
```

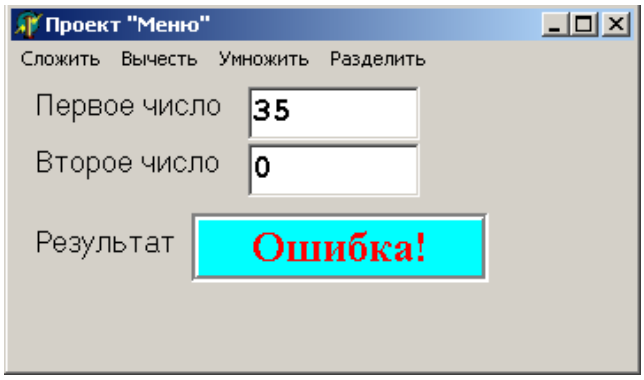


Рис.9.3

Теперь программа будет правильно работать даже при попытке деления на ноль (рис.9.3).

Задание для самостоятельного выполнения.

Разработайте проект, в котором в главном меню отражены названия времен года, а в выпадающих меню — названия соответствующих месяцев каждого времени года (*рис 9.4)*

При выборе того или иного месяца должно выводиться количество дней в этом месяце (для февраля — в невисокосном году)

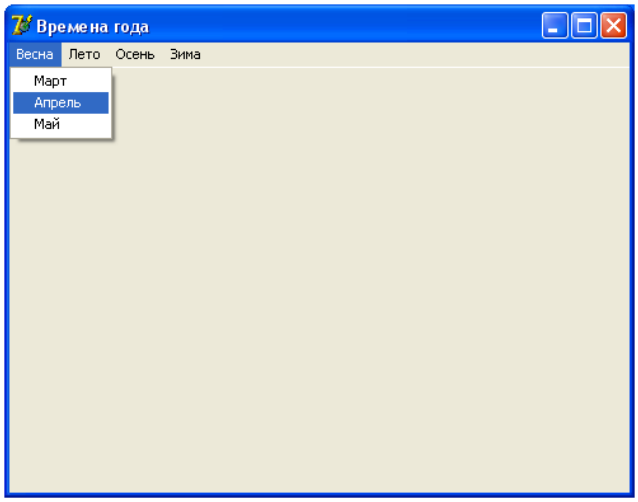


Рис.9.4

Занятие 10. Класс TMemo, стандартные диалоги для работы с текстовыми файлами: TOpenDialog, TSaveDialog. Установка параметров шрифта с использованием диалога TfontDialog

Цель: изучить назначение и порядок изменения свойств визуальных компонентов **TMemo, TOpenDialog, TSaveDialog.**

Постановка задачи: построить простейший текстовый редактор, воспроизводящий некоторые функции настоящих редакторов.

Теоретические сведения:

Компонент **Memo**, в отличие от строки ввода Edit, может содержать множество строк, то есть, предназначен для работы с большими текстами. В Memo можно переносить слова, сохранять в буфере обмена фрагменты текста и восстанавливать их, выполнять другие базовые функции текстового редактора. При этом объем текста не должен превышать 32 Кб. Свойства Border Style, Readonly, HideSelection, MaxLenght — те же, что и у строки ввода. Специфические свойства компонента Memo:

Lines — содержимое Memo как набор строк;
Align — заполнение пространства формы по краям;
AlignMent — выравнивание текста внутри формы;
Scrollbars — наличие полос прокрутки содержимого поля;
Wordwrap автоперенос текста от правого края.

Для удобства работы с полем примечаний как с текстовым редактором имеются свойства WantTabs и WantReturns, которые "восстанавливают в правах" функции клавиш Enter и Tab применительно к работе с текстом.

Строки текста (свойство Lines) вводятся в редакторе строк, который вызывается при нажатии на кнопку с тремя точками. Но их можно загружать из файла и выводить в файл. Для этого Lines использует методы LoadFromFile *(имя файла)* - загрузить из файла и SaveToFile *(имя файла)* - записать в файл.

Стандартные диалоги - это диалоги открытия и сохранения файла для текстовых и графических файлов, установки цвета, параметров шрифта и т.д. Все они размещены на вкладке Dialogs Палитры компонент.

Все диалоги относятся к невизуальным компонентам, поэтому могут быть размещены в любом месте формы, в том числе и на других объектах.

Порядок создания проекта «Редактор текста»:

1. Начните новое приложение.

2. Разместите на форме объект Memo1. Установите свойства ScrollBars = ssVertical, Align = alClient (полное заполнение формы).

3. Добавьте на форму диалоги OpenDialog1, SaveDialog1 и FontDialog1. Настроим диалоги открытия и сохранения файла.

Настройка сводится к заданию свойства Filter. Это свойство определяет, какие файлы "увидит" диалог при запуске. Для настройки свойства выделим его поле в Инспекторе объектов и щелчком на трех точках развернем Редактор фильтра.

Его внешний вид показан на рисунке 10.1.

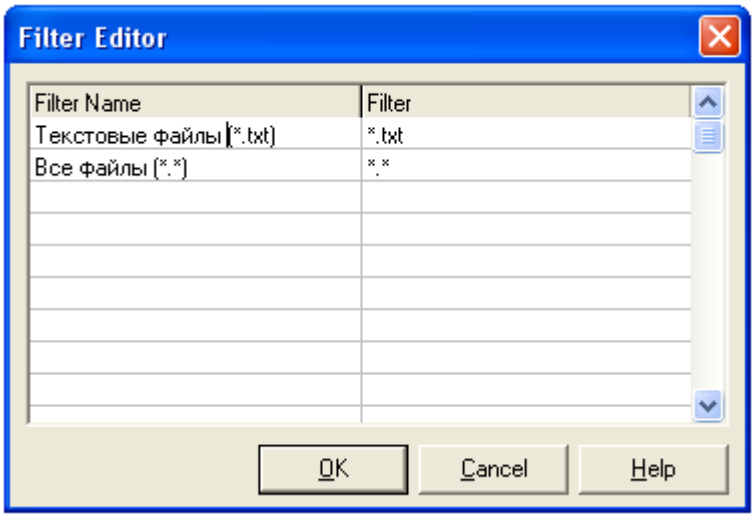


Рис. 10.1

В левый столбец заносится поясняющая строка, а в правый — шаблон имени файла. В нашем примере таких строк две. В первой строке (она будет видна в поле Тип файлов при открытии диалога.) заданы только текстовые файлы, во второй — все файлы.

4. Аналогично настройте диалог SaveDialog1. В нем только дополнительно зададим свойство DefaultExt = txt, чтобы расширение *txt* автоматически добавлялось к имени при записи файла.

5. Для удобства управления проектом создадим меню, при выборе пунктов которого будут вызываться диалоги. На рисунке 10.2 показан момент выбора пункта «Сохранить».

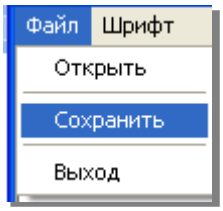


Рис.10.2

Разделитель вставляется в меню, если в поле Caption в Инспекторе объектов ввести дефис (знак минус).

Общий вид проекта на этапе разработки показан на рисунке 10.3.

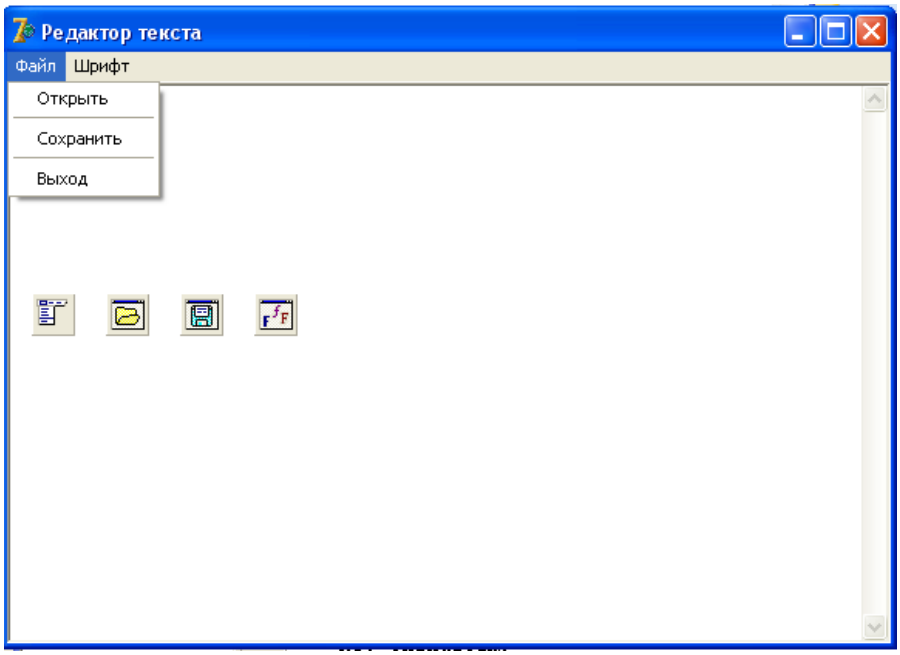


Рис. 10.3

6. Осталось только написать реакцию программы на выбор пунктов меню.

Основное назначение диалога открытия (закрытия) файла — выбрать файл. Для этого нужно запустить выполнение диалога. Это делает метод Execute, принадлежащий каждому диалогу. Метод возвращает True, если пользователь завершил диалог нажатием кнопки "Ok", и False, если пользователь нажал кнопку "Cancel".

Полное имя выбранного файла (то есть вместе с путем) помещается в свойство FileName диалога и может быть использовано при чтении (записи) текста. Для удобства работы имя файла можно поместить и в заголовок формы. Все описанное реализует следующий код:

```
With OpenDialog1 do
  begin
    if Execute then
            begin
       Memo1.Lines.LoadFromFile (Fi leName) ;
       Form1.Caption:=’Проект "Редактор текстов" ’+ FileName;
             end;
  end;
```

7. Напишите самостоятельно код процедуры сохранения файла.

Диалог для выбора параметров шрифта программируется аналогично, только вместо имени файла необходимо использовать свойство Font:

```
 With FontDialog1 do begin
if Execute then Memo1.Font:= Font;
 end;
```

8. Окно работающего проекта показано на рисунке 10.4.

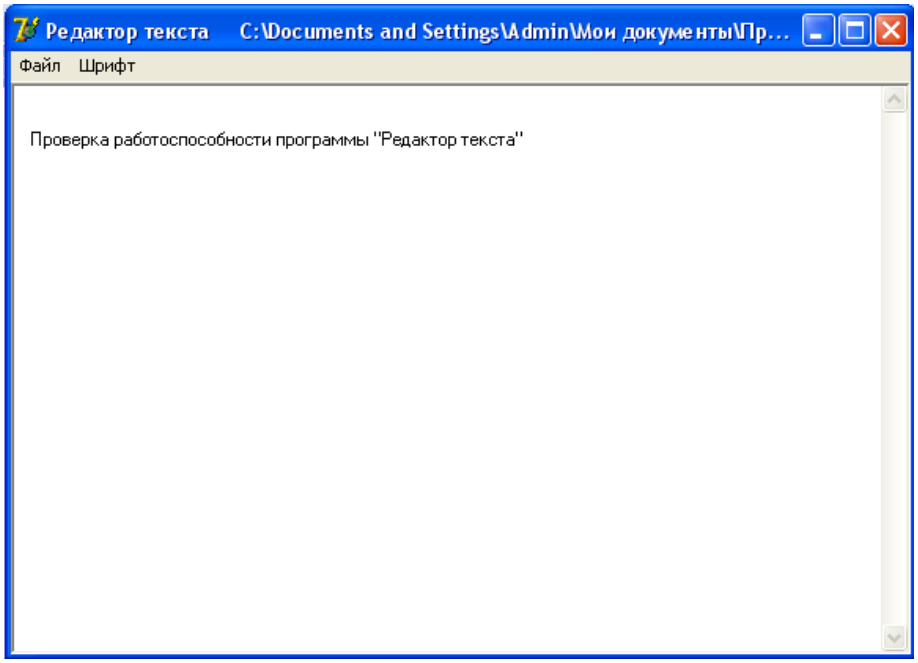


Рис. 10.4

При изменении размеров окна поле Memo1 также изменяет свои размеры благодаря тому, что свойство Align имеет значение alClient. При этом текст также переформатируется.

Листинг программы

```
unit Unit10;
interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, Menus, StdCtrls;
type
  TForm1 = class(TForm)
    Memo1: TMemo;
    OpenDialog1: TOpenDialog;
    SaveDialog1: TSaveDialog;
    FontDialog1: TFontDialog;
    MainMenu1: TMainMenu;
    N1: TMenuItem;
    N2: TMenuItem;
    N3: TMenuItem;
    N4: TMenuItem;
    N5: TMenuItem;
    N6: TMenuItem;
    N7: TMenuItem;
    procedure N3Click(Sender: TObject);
    procedure N5Click(Sender: TObject);
    procedure N2Click(Sender: TObject);
private
```

```
    { Private declarations }
  public
    { Public declarations }
  end;
var
  Form1: TForm1;
implementation
{$R *.dfm}

procedure TForm1.N3Click(Sender: TObject);
begin
with OpenDialog1 do begin
if execute then begin
memo1.Lines.LoadFromFile(filename);
form1.Caption:='Редактор текста     '+filename;
                end;
                   end;
end;
procedure TForm1.N5Click(Sender: TObject);
begin
with SaveDialog1 do begin
if execute then begin
memo1.Lines.SaveToFile(filename);
form1.Caption:='Редактор текста     '+filename;
                end;
                   end;
end;
procedure TForm1.N2Click(Sender: TObject);
begin
with FontDialog1 do begin
if Execute then Memo1.Font:=Font;
                  end;
end;
end.
```

Задание для самостоятельной работы:

Создайте приложение, которое выполняет следующие операции с текстовым файлом: открывает файл с расширением *.txt, указанный пользователем в диалоговом окне, затем изменяет текст, удаляя из него последовательность символов, заданную пользователем, и сохраняет файл на диске с новым именем и расширением, которое задает пользователь в диалоговом окне.

Занятие 11. Класс TImage, стандартные диалоги для работы с картинками. Класс Canvas (холст), инструменты рисования TBrush и TPen, графические примитивы

Цель: изучить назначение, возможности, свойства и методы классов **TImage**, **TCanvas.**

Постановка задачи: разработать приложение «Рисование картинки» для простейших манипуляций с изображениями: рисования простых изображений, сохранения их в файле и загрузки из файла.

Теоретические сведения:

Основой проекта является класс **TImage** — набор данных и методов для работы с изображениями в формате *bmp.*

Основными являются свойства Picture и Canvas. Свойство **Picture** и есть картинка. Ее можно задать в Инспекторе объектов с помощью загрузчика картинок, который, хотя и называется Picture Editor (Редактор картинок), никаких операций редактирования не выполняет.

Для загрузки и сохранения картинок в процессе работы приложения у свойства Picture есть методы LoadFromFile и SaveToFile. Для выбора имени файла на вкладке Диалоги имеются специализированные диалоги **OpenPictureDialog** и **SavePictureDialog**.

Для вывода графических примитивов объект Image содержит свойство **canvas** — холст. Заметим тут же, что это свойство есть у многих визуальных объектов, в том числе и у Формы, следовательно, можно рисовать и непосредственно на форме.

Для рисования Canvas содержит два инструмента и множество графических примитивов. Инструмент Pen (Ручка) определяет цвет, толщину и стиль линий и границ областей, а инструмент Brush (Кисть) - цвет и стиль заливки области. Сами же графические примитивы рисуются методами, принадлежащими холсту. Перечислим некоторые из них:

Arc - дуга окружности или эллипса;
Ellipse - закрашенный эллипс или окружность;
FillRect - закрашенный прямоугольник;
LineTo - провести линию в заданную точку;
MoveTo - перейти в заданную точку;
Rectangle — прямоугольная рамка;
TextOut - вывод текста.

План разработки программы:

1. Разместим на форме Панель. Она будет играть роль рамки для картины, поэтому установим BevelInner = bvNone, BevelOuter = bvLovered. Поместим внутрь объект Image (он находится на вкладке Дополнительная). Добавим диалоги для загрузки и сохранения графических файлов OpenPictureDialog и SavePictureDialog. Добавим меню, чтобы обеспечить выполнение диалогов, рис.11.1.

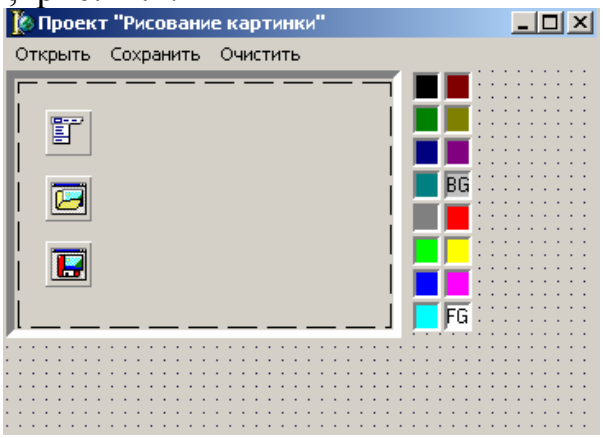


Рис. 11.1

2.Настроим диалоги.

Свойство Filter уже настроено. Оно содержит несколько вариантов:

Filter Name	Filter
All (*.ico; *.emf; *.bmp;*.wmf)	*.bmp; *.ico; *.emf; *.wmf
Bitmaps(*.bmp)	*.bmp
Icons(*.ico)	*.ico
...	...

Настроим еще два полезных свойства. Свойство InitialDir (Начальный каталог) указывает, в какой каталог мы попадаем при вызове диалога. Свойство DefaultExt позволяет не вводить расширение при записи файла — расширение, заданное в свойстве, будет добавлено автоматически.

3.Проект уже наполовину готов. Но у нас пока нет возможности нарисовать свою картинку. Для рисования картинки выберем метод

Ellipse(X-R, Y-R, X+R, Y+R) — окружность радиуса R с центром в точке X,Y.

Предварительно зададим параметры инструментов:

Pen. Color : = ...; - цвет линии контура;

Pen. width :=...; - толщина линии контура;

Brush. Color : = ...; - цвет заливки.

По умолчанию линия является сплошной, но можно задать и другой стиль линии Pen . style:

psSolid - сплошная линия;

psClear - линия не рисуется;

psDash - линия из тире;

psDot - линия из точек и т.д.

К сожалению, все пунктирные и штриховые линии могут быть толщиной только в один пиксель.

По умолчанию заливка является сплошной, однако можно задать и другой стиль заливки Brush. Style:

bsSolid - сплошная заливка;
bsClear - нет заливки;
bsHorizontal - горизонтальная штриховка;
bsVertical - вертикальная штриховка;
bsFDiagonal - диагональная штриховка;
bsBDiagonal - диагональная штриховка;
bsCross - клетки;
bsDiagCross - диагональные клетки.

4. Запрограммируем собственно рисование. Для этого используем событие OnMouseDown, которое возникает, когда пользователь нажимает кнопку мыши. Если в этот момент курсор мыши находится над рисунком, вызывается процедура Tform1.image1MouseDown с параметрами:

Sender - отправитель сообщения о событии OnMouseDown;
Button - номер нажатой кнопки (mbLef t, mbRight, mbMiddle);
shift - нажата ли вспомогательная клавиша Shift, Alt, Ctrl;
X, Y - координаты базовой точки курсора мыши.

5. Завершите проект и поработайте с ним. Нарисуйте картинку из окружностей и сохраните ее на диске. Откройте ее снова, дополните рисунок и запишите его под другим именем.

6. Улучшим полученную программу. Во-первых, хотелось бы иметь возможность стирать неудачный рисунок. Вызовем Редактор меню и добавим пункт "Очистить". В процедуре обработки события закрасим всю рабочую область рисунка — ClientRect — белым цветом:

```
procedure Tform1.N3Click(Sender: TObject);
begin
  with Image1.Canvas do
  begin
    Brush.Color := clWhite;
    FillRect(ClientRect);
   end;
end;
```

7. Эту же процедуру можно использовать для закраски фона рисунка при запуске программы. Чтобы не повторять тот же текст в процедуре обработки события OnCreate, возникающего при создании Формы, назначим для этого события уже написанный обработчик.

Для этого в Инспекторе объектов в списке событий Формы щелкнем мышкой на строке OnCreate. Справа появится раскрывающийся список с

перечнем всех имеющихся в программе обработчиков. Выберем из списка N3Click — и получим требуемое.

8. Во-вторых, у нас нет возможности при работе программы изменять параметры окружности. Добавим в проект возможность изменения цвета с помощью компонента **ColorGrid** (Таблица цветов). Расположен он на вкладке **Samples** (Примеры).

Щелчок левой кнопки мыши по клетке с некоторым цветом выбирает его как цвет переднего плана (ForegroundColor). Щелчок правой кнопки выбирает цвет заднего плана (BackgroundColor). Для настройки объекта используем свойство GridOrdering — размеры таблицы (go2x8, go16x1, go4x4 и т.д.). В любом случае нам доступны 16 цветов.

Начальные установки выбранных клеток таблицы:

Foregroundindex— индекс цвета переднего плана (линии);

Backgroundlndex — индекс цвета заднего плана (заливка).

В выбранных клетках появляются обозначения "FG" и "BG" соответственно (при совпадении клеток — буквы "FB").

Иллюстрация работы программы "Рисование картинки" рис.11.2 и полный текст процедуры обработки нажатия кнопки мыши:

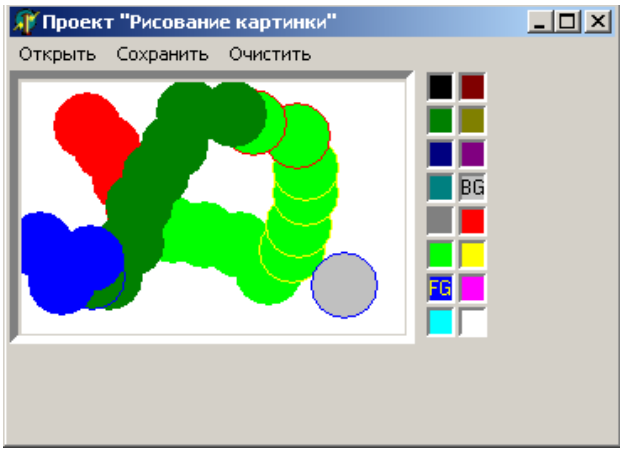


Рис.11.2

```
procedure TForml.ImagelMouseDown(Sender: TObject; Button: TMouseButton;
Shift: TShiftState;X, Y: Integer);
  begin
     with Imagel.Canvas do
        begin
      Pen.Color :=ColorGrid1ForegroundColor;
     Brush.Color := ColorGrid1BackgroundColor;
     Ellipse(X - 20, y - 20, X + 20, Y + 20);
          end; end;
```

Задание для самостоятельной работы:

Создайте приложение, которое при щелчке на кнопке Нарисовать рисует в окне приложения цветок ромашки.

Занятие 12. Графические возможности Delphi. Построение графиков функций

Цель: изучение приемов и методов построения графиков функций в системе программирования Delphi.

Постановка задачи: Создать приложение, позволяющее построить графики функции вида:

$$y=ax^2+bx+c;$$

где $10 \le x \le 10$, коэффициенты a, b, c – задаются пользователем.

Для перехода от декартовых координат к экранным использовать формулы:

$x_l=ox+mx*x;$

$y_l=oy-my*y;$

где x,y -декартовые координаты, x_l,y_l - экранные координаты, (ox, oy) — экранная точка, выбранная в качестве начала координат; mx, my — масштаб, т.е. длина единичных отрезков в пикселях.

План разработки программы:

1. Открыть новый проект. Дать форме название «Построение параболы».
2. Разместить на форме компоненты Label1, LaLabel2, LabeledEdit1, LabeledEdit2, LabeledEdit3, Editl, Edit2, Buttonl, Button2, PaintBoxl и настроить их свойства в соответствии с рис.12.1.

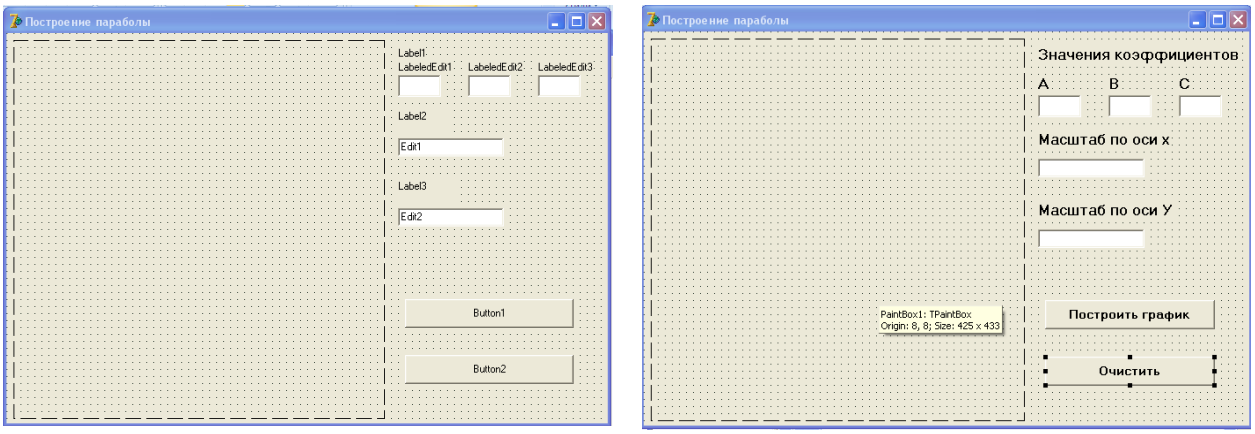


Рис. 12.1.

3.Для кнопки «Построить график» написать программный код - процедуру построения графика функции:

```
procedure TForm1.Button1Click(Sender: TObject);
var
x1,y1,ox,oy,mx,my,i: integer;
a,b,c,x,y: real;
begin
```

```
{устанавливаем черный цвет пера для построения осей}
paintbox1.Canvas.pen.Color:=clblack;
ox:=paintbox1.Width div 2;
oy:=paintbox1.Height div 2;
mx:=strtoint(edit1.text);
my:=strtoint(edit2.text);
a:=strtofloat(labelededit1.text);
b:=strtofloat(labelededit1.text);
c:=strtofloat(labelededit1.text);
paintbox1.Canvas.MoveTo(0,oy);
paintbox1.Canvas.LineTo(paintbox1.Width,oy); {ось x}
paintbox1.Canvas.MoveTo(ox,0);
paintbox1.Canvas.LineTo(ox,paintbox1.Height); {осьy}
{устанавливаем синий цвет пера для прорисовки графика}
paintbox1.Canvas.pen.Color:=clblue;
 x:=-10;y:=a*sqr(x)+b*x+c;     {вычисление координаты начальной точки графика}
{перевод в экранные координаты}
x1:=trunc(ox+mx*x); y1:=trunc(oy-my*y);
paintbox1.Canvas.MoveTo(x1,y1); {перемещение пера в начало прорисовки графика}
while x<=10 do {цикл построения графика}
begin y:=a*sqr(x)+b*x+c;
x1:=trunc(ox+mx*x);
y1:=trunc(oy-my*y);
paintbox1.Canvas.lineTo(x1,y1); x:=x+0.1
end;
end;
```

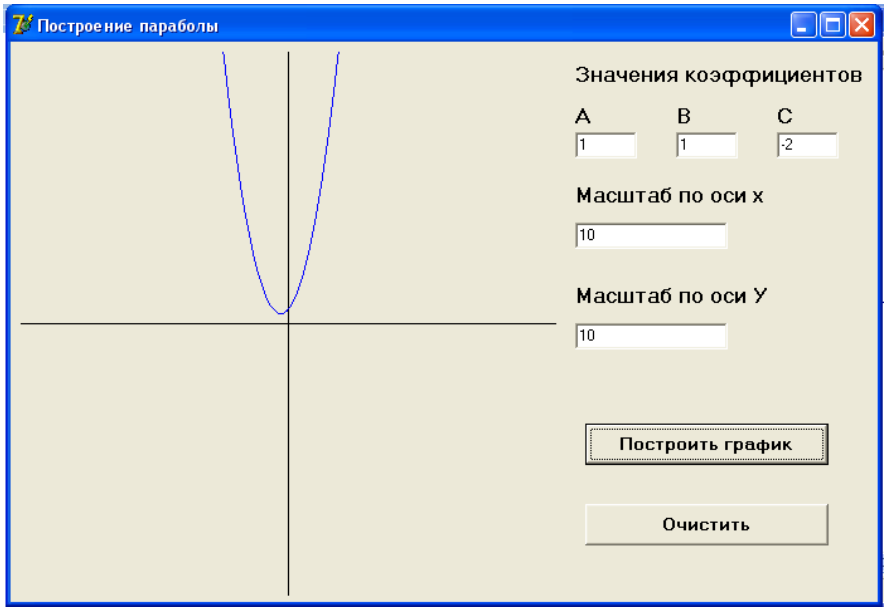


Рис. 12.2

3. Создать процедуру для очистки изображения графика (рис.12.2).

```
procedure TForm1.Button2Click(Sender: TObject);
begin
paintboxl.Canvas.Pen.Color:=clblack;
paintboxl. Canvas.Brush.Color:=clwhite;
paintboxl .Canvas.Rectangle(0,0,paintbox1.Width, paintbox1.Height);
end;
```

Задание для самостоятельной работы:

1. Запрограммировать разметку осей координат.
2. Разместить на форме радиокнопки выбора функций для построения графиков y=a*sin(x); y=l/x; y =ax+b.

Занятие 13. Графические возможности Delphi. Построение диаграмм. Компонент Chart

Цель: изучение приемов и методов построения диаграмм в системе программирования Delphi.

Постановка задачи: разработать проекты «Круговая диаграмма», «График функции» с помощью компонента Chart.

Теоретические сведения

Компонент Chart позволяет строить графики, диаграммы различных видов, расположен на палитре Additional.

Компонент Chart является контейнером объектов Series — наследников класса TChartSeries. Каждый такой объект представляет серию данных, характеризующихся определенным стилем отображения: тем или иным графиком или диаграммой. Каждый компонент Chart может включать несколько серий. Если вы хотите отображать график, то каждая серия будет соответствовать одной кривой на графике. Бели вы хотите отображать диаграммы, то для некоторых видов диаграмм можно наложить друг на друга несколько различных серий. Однако, и в этом случае вы можете задать для одного компонента Chart несколько серий одинаковых данных с разным типом диаграммы. Тогда, делая в каждый момент времени активной одну из них, вы можете предоставить пользователю выбор типа диаграммы, отображающей интересующие его данные.

Разместите один или два компонента Chart на форме и посмотрите открывшиеся в Инспекторе Объектов свойства. Пояснения к некоторым из них:

AllowPanning	Определяет возможность пользователя прокручивать наблюдаемую часть графика во время выполнения, нажимая правую кнопку мыши. Возможные значения: pmNone — прокрутка запрещена, pmHorizontal, pin Vertical или pmBoth — разрешена соответственно прокрутка только в горизонтальном направлении, только в вертикальном или в обоих направлениях
AllowZoom	Позволяет пользователю изменять во время выполнения масштаб изображения, вырезая фрагменты диаграммы или графика курсором мыши. Если рамка фрагмента рисуется вправо и вниз, то этот фрагмент растягивается на все поле графика. А если рамка рисуется вверх и влево, то восстанавливается исходный масштаб
Title	Определяет заголовок диаграммы

Foot	Определяет подпись под диаграммой. По умолчанию отсутствует. Текст подписи определяется подсвойством Text
Frame	Определяет рамку вокруг диаграммы
Legend	Легенда диаграммы — список обозначений
MarginLeft, MarginRight, MarginTop, MarginBottom	Значения левого, правого, верхнего и нижнего полей
BottomAxis, LeftAxis, RightAxis	Эти свойства определяют характеристики соответственно нижней, левой и правой осей. Задание этих свойств имеет смысл для графиков и некоторых типов диаграмм
LeftWall, BottomWall, BackWall	Эти свойства определяют характеристики соответственно левой, нижней и задней граней области трехмерного отображения графика
SeriesList	Список серий данных, отображаемых в компоненте
View3d	Разрешает или запрещает трехмерное отображение
View3DOptions	Характеристики трехмерного отображения
Chart3DPer-cent	Масштаб трехмерное - это толщина диаграммы и ширина лент графика

План разработки программы «Круговая диаграмма»:

1. Создайте форму следующего вида (рис.13.1)

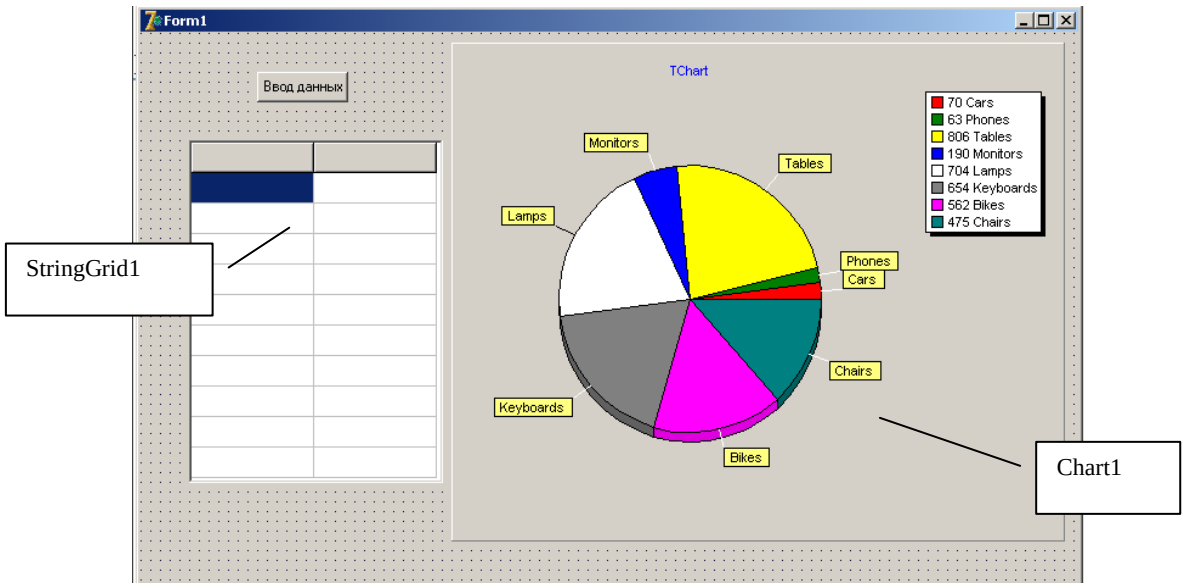


Рис.13.1

2. Присвойте компонентам следующие значения

Компонент	Свойство	Значение
Button1	Caption	Ввод
SrtingGrid1	ColCount RowCount FixedCols	2 11 0

3.Дважды щелкните по объекту Chart и в появившемся окне добавьте элемент Series (Add) и выберите тип Pie (рис.13.2)

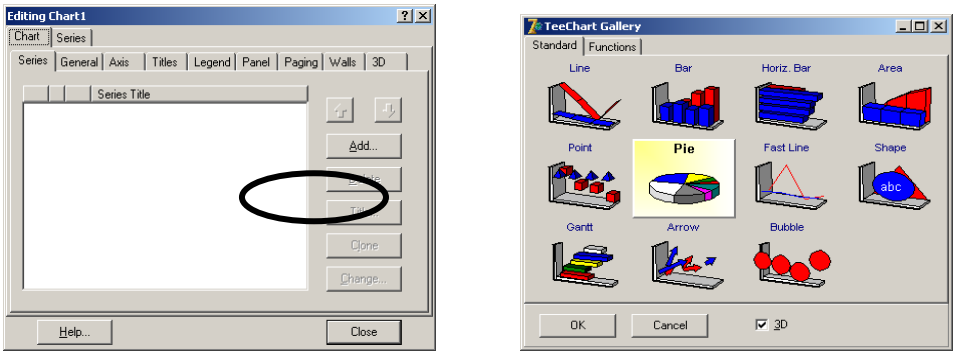
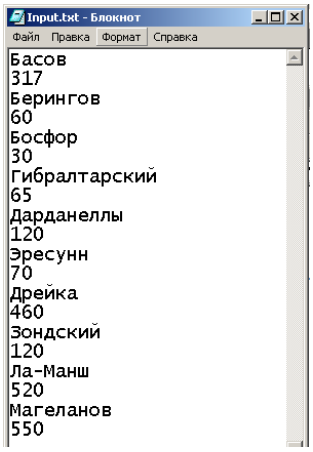


рис 13.2

4. Сохраните проект и в папке (где сохранили проект) создайте текстовый документ Input.txt со следующим содержанием:



5. Выделите объект Form1 в диспетчере объектов и перейдите на закладку Events в Object Inspector и дважды щелкните в поле справа от события OnCreate

6. Запрограммируйте событие создания формы.

```
procedure TForm1.FormCreate(Sender: TObject);
begin
StringGrid1.Cells[0,0]:='Пролив';
StringGrid1.Cells[1,0]:='Длина';
Chart1.Title.Text.Clear;
Chart1.Title.Text.Add('Диаграмма');
Series1.Clear;
Series1.Marks.Visible:=false;

end;
```

7. Запрограммируйте событие onClick для button1

```
procedure TForm1.Button1Click(Sender: TObject);
  Var
     f:textfile;
     proliv:string;
     dlina,i:integer;
begin
  AssignFile(f,'Input.txt'); {связь файловой переменной с файлом Input.txt}
  reset(f); {открываем файл Input.txt для чтения}
  i:=1;
  while not Eof(f) do
    begin
     readln(f,proliv);{считываем из файла строку содержащую текст}
     readln(f,dlina); {считываем из файла строку содержащую число}
     StringGrid1.Cells[0,i]:=proliv;
     StringGrid1.Cells[1,i]:=inttostr(dlina);
     Series1.Add(dlina,proliv, clTeeColor); {передаем в Chart данные: число, текст и цвет}
     inc(i); {тоже самое что и i:=i+1}
    end;
end;
```

8.Запустите программу на исполнение.

План разработки программы «Построение графика функции»

1. Создайте форму следующего вида

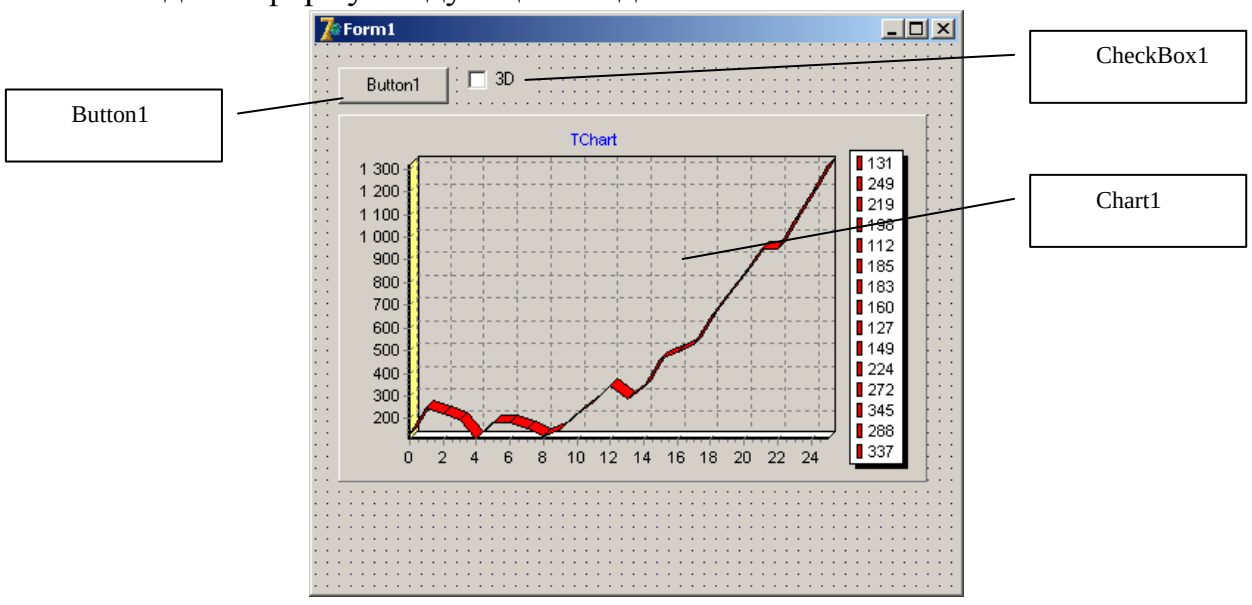


Рис.13.3

2. Присвойте компонентам следующие значения

Компонент	Свойство	Значение
CheckBox1	Caption	3D

3. Дважды щелкните по объекту Chart и в появившемся окне добавьте элемент Series (Add) и выберите тип Line (рис.13.4)

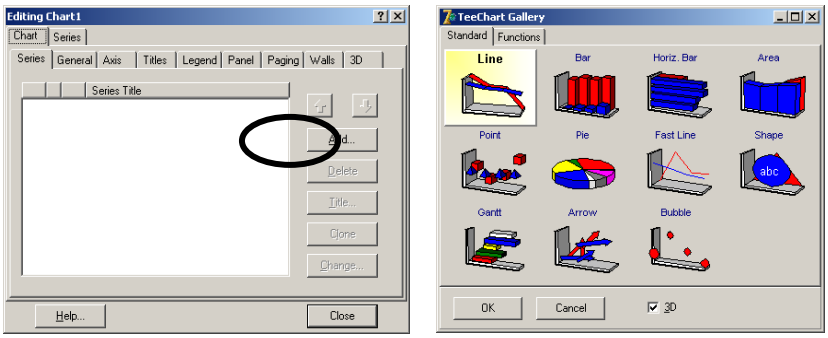


Рис.13.4

4. Запрограммируйте событие onClick для Button1

```
procedure TForm1.Button1Click(Sender: TObject);
var i:integer;
begin
Series1.Clear;
Chart1.View3D:=false;
Series1.Title:='График функции ';
for i := 0 to 100 do
begin
  Series1.AddXY(0.02 * Pi * i, sin(0.02 * Pi * i), 'sin(x)', clRed);
end;
end;
```

Задания для самостоятельной работы

Создайте проект который бы выводил график функции y1= $\sin(x^2)+\cos(x^2)$ и y2=sin(x)*cos(x) на интервале [-Pi, Pi] с шагом h=-Pi/10.

Занятие 14. Создание анимации в Delphi. Классы TTimer(Таймер) и TShape (Фигура)

Цель: изучение приемов и методов создания анимированных изображений в системе программирования Delphi.

Постановка задачи: разработать проект «Светофор», переключение цвета должно выполняться автоматически через установленный интервал времени, включение и выключение светофора выполняется соответствующими кнопками (рис.14.1).

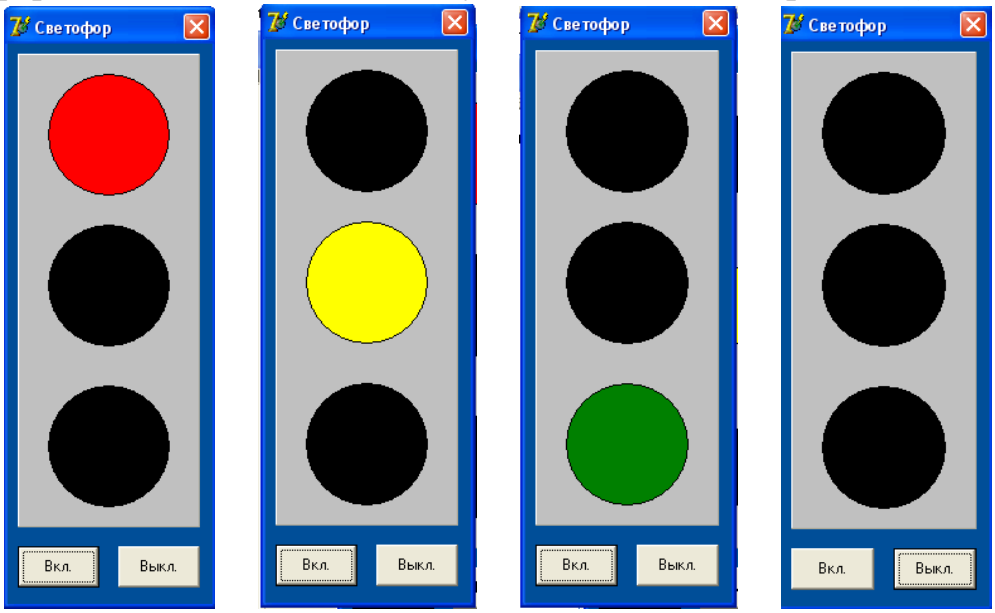


Рис.14.1

Теоретические сведения

Таймер(**Timer)** - невизуальный компонент, который позволяет создавать в приложении интервалы времени, он не создает элемента управления, но как и все компоненты имеет свои свойства и события. Он может взаимодействовать с другими объектами программы. Для таймера главное событие – истечение заданного интервала времени.

Таймер применяется для синхронизации мультипликации, закрытия каких-то окон, с которыми пользователь долгое время не работает, для задания времени на ответ в обучающих программах и т.д.

Таймер может размещаться в любом месте формы. Имеет два свойства, позволяющие им управлять:

- **Interval** – интервал времени в миллисекундах, задает период срабатывания таймера. Через заданный интервал времени после предыдущего срабатывания , или после программной установки свойства **Interval**, или после запуска приложения, если значение **Interval** установлено во время проектирования, таймер срабатывает,

вызывая событие **OnTimer**. В обработчике этого события записываются необходимые операции.

- **Enabled** – доступность (значение **True** или **False)**.

Если задать **Interval=0** или **Enabled = False**, то таймер перестанет работать. Чтобы запустить отсчет времени, надо или задать **Enabled =True** (если установлено положительное значение **Interval**) или задать положительное значение **Interval**, если **Enabled = False**.

Пример 1: если требуется, чтобы через 5 секунд после запуска приложения закрылась форма – заставка. На ней надо разместить таймер, задать в нем **Interval=5000**, а в обработчик события **OnTimer** вставить оператор **Close**, закрывающий окно формы.

Пример 2: необходимо в некоторой процедуре запустить таймер, который отсчитал бы заданный интервал времени, например 5 секунд, после чего надо выполнить некоторые операции и отключить таймер. Это можно сделать следующим образом:

- Способ 1. при проектировании таймер делается доступным (**Enabled =True),** но свойство **Interval =0.** Таймер работать не будет, пока в момент, когда его нужно запустить, не выполниться оператор

  Timer1.Interval:=5000;

Через 5 секунд после этого наступит событие **OnTimer**. В его обработчике надо задать оператор

Timer1.Interval:=0;

который отключит таймер, после чего можно выполнять требуемые операции.

- Способ 2. – использование свойства **Enabled.** Во время проектирования задается значение **Interval =5000** и значение **Enabled = False.** В момент, когда надо запустить таймер выполняется оператор

  Timer1.Enabled:=true;

В обработчик события **OnTimer**, которое наступит через 5 секунд после запуска таймера, можно вставить оператор

Timer1.Enabled:=false;

который отключит таймер.

План разработки программы

1. Открыть новый проект.
2. Разместить на форме экземпляры компонентов в соответствии с рис.14.2

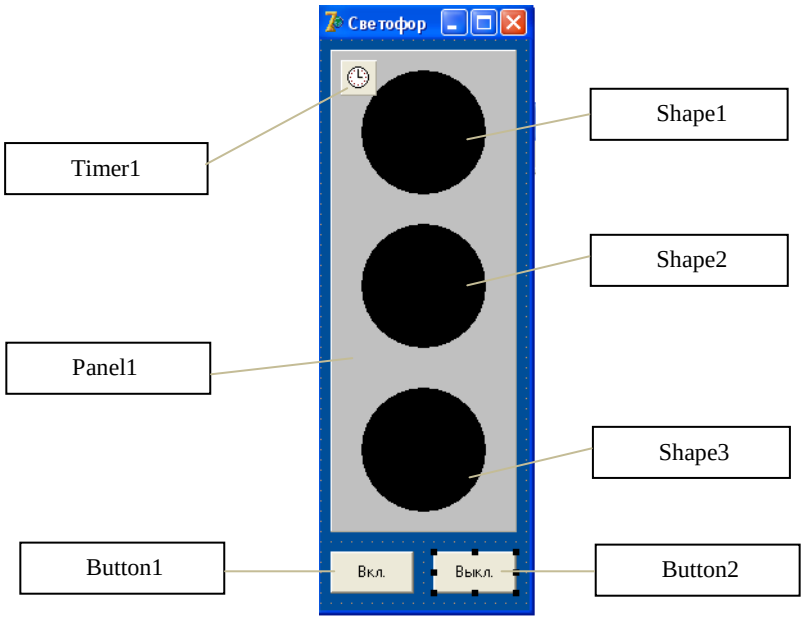


Рис.14.2

3. Выполнить следующие действия:

Выделенный объект	Вкладка окна Object Inspector	Имя свойства/ имя события	Действие
Form1	Properties	Caption	Установка имени формы «Светофор»
		Position	poScreenCenter
		BorderIcons	biMinimize=false biMaximize=false
		Color	clBackground
		BorderStyle	bsSingle
Timer1	Properties	Enabled	False
		Interval	50000
Shape1 Shape2 Shape3	Properties	Shape	stCircle
Panel1	Properties	Color	clSilver

Button1	Properties	Caption	Установка имени кнопки:«Вкл.»

	Events	OnCIick	shape1.Brush.Color:=clred;
			c:='r';
			timer1.Interval:=5000;
			timer1.Enabled:=true;

Button2	Properties	Caption	Установка имени кнопки: «Выкл.»

	Events	OnCIick	timer1.Enabled:=false;
			shape1.Brush.Color:=clblack;
			shape2.Brush.Color:=clblack;
			shape3.Brush.Color:=clblack;
			timer1.Interval:=5000;

Добавить в раздел ***Var*** глобальную переменную ***c:char;***, которую будем использовать в качестве индикатора состояния светофора (‘r’ – горел красный;‘g’- горел зеленый).

4. Напишите процедуру обработки события **OnTimer** для таймера. Для этого двойным щелчком левой кнопки мыши выделите **Таймер** на форме и создайте программный код, реализующий автоматическое переключение цветов светофора через заданный интервал времени. Задать время работы для красного и зеленого цвета – 5 сек., время работы желтого цвета – 1 сек.

```
procedure TForm1.Timer1Timer(Sender: TObject);
label m;
begin
timer1.Enabled:=false;
if (shape1.Brush.Color=clred)or (shape3.Brush.Color=clgreen) then
begin
shape1.Brush.Color:=clblack;
shape2.Brush.Color:=clyellow;
shape3.Brush.Color:=clblack;
timer1.Interval:=1000;  goto m
end;

if (shape2.Brush.Color=clyellow)and (c='r') then
begin
shape1.Brush.Color:=clblack;
shape2.Brush.Color:=clblack;
shape3.Brush.Color:=clgreen; c:='g';
timer1.Interval:=5000;  goto m
end;
```

```
if (shape2.Brush.Color=clyellow)and (c='g') then
begin
shape1.Brush.Color:=clred; c:='r';
shape2.Brush.Color:=clblack;
shape3.Brush.Color:=clblack;
timer1.Interval:=5000;
end;
m:timer1.Enabled:=true;
end;
```

5.Запустите проект на выполнение и протестируйте его.

6.Сохраните проект в папке *Светофор*.

Задание для самостоятельной работы:

1. Разработайте модель работающего светофора с секундомером.

2. Добавьте режим настройки светофора, позволяющий устанавливать время (в секундах) работы каждого из цветов.

3. Разработайте проект, в котором периодически происходит смена окружности на квадрат, прямоугольник, эллипс и прямоугольник со скругленными краями, затем опять на квадрат и т.д. При смене фигур должен также меняться их цвет.

Занятие 15. Полярные координаты при создании анимации в Delphi. Класс TtrackBar

Цель: изучение приемов и методов создания анимированных изображений в системе программирования Delphi.

Постановка задачи: разработать проект «Движение по орбите», в котором при нажатии на кнопку «Пуск» по орбите радиусом r выполнялось движение шарика. Кнопка «Стоп» должна останавливать перемещение объекта (рис.15.1).

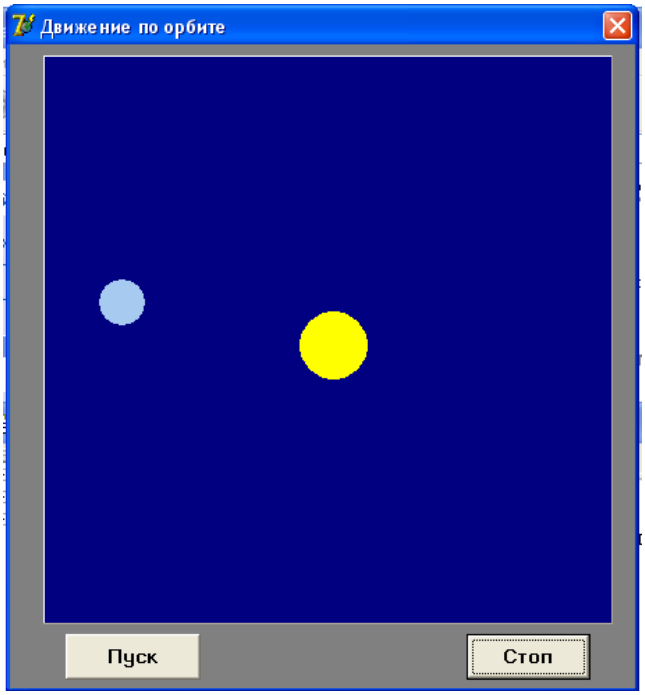


Рис.15.1

Теоретические сведения

Для расчета траектории движения шарика по орбите будем использовать уравнение окружности с центром в начале координат (рис.15.2).

Координаты точки, лежащей на окружности, можно вычислить по формулам:

$$\begin{cases} x = r \cdot cos\varphi \\ y = r \cdot sin\varphi \end{cases}$$

где r – радиус окружности (орбиты), φ – угол поворота в радианах.

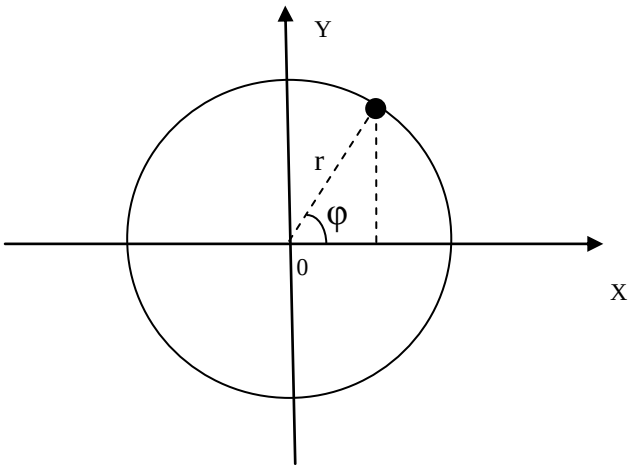


Рис.15.2

Начало координат расположено в левом верхнем углу панели, на которой будем строить движение, поэтому сместим его в центр:

$$\begin{cases} x = x0 + r \cdot cos\varphi \\ y = y0 - r \cdot sin\varphi \end{cases}$$

где (x0,y0) – координата точки центра панели (компонент Panel1).

План разработки программы

1. Открыть новый проект.
2. Разместить на форме компоненты в соответствии с рис.15.3

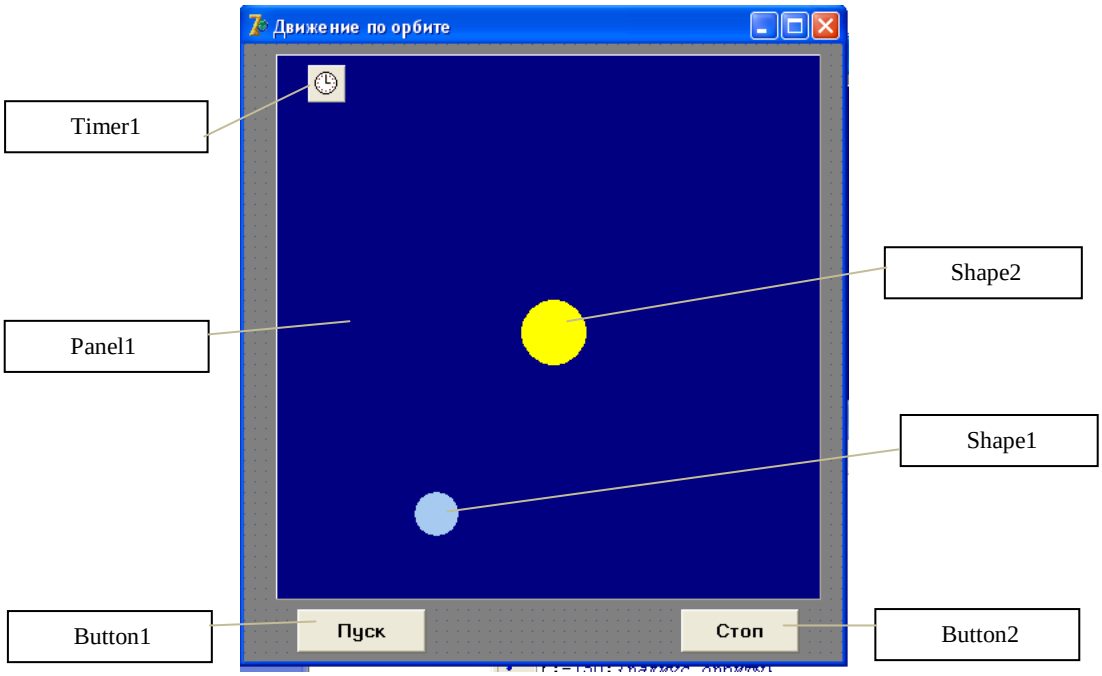


Рис.15.3

3. Выполнить следующие действия:

Выделенный объект	Вкладка окна Object Inspector	Имя свойства/ имя события	Действие
Form1	Properties	Caption	Установка имени формы «Движение по орбите»
		Position	poScreenCenter
		BorderIcons	biMinimize=false biMaximize=false
		BorderStyle	bsSingle

	Events	OnCreate	x0:=panel1.Width div 2;
			y0:=panel1.Height div 2;
			f:=0; {угол поворота}
			r:=150;{радиус орбиты}
			x:=trunc(x0+r*cos(f));
			y:=trunc(y0-r*sin(f));
			shape1.Left:=x-shape1.Width div 2;
			shape1.top:=y-shape1.Height div 2;

Timer1	Properties	Enabled	False
		Interval	100
Shape1 Shape2	Properties	Shape	stCircle
Panel1	Properties	Color	clNavy
Button1	Properties	Caption	Установка имени кнопки: «Пуск»
	Events	OnCIick	timer1.Enabled:=true;
Button2	Properties	Caption	Установка имени кнопки: «Стоп»
	Events	OnCIick	timer1.Enabled:=false;

Для задания цвета и способа заливки компонента Shape используется свойство Brush (Кисть). Выполним в Инспекторе объектов двойной щелчок на плюсе слева от слова Brush, тогда появятся составляющие этого свойства, которые можно задать обычным образом.

Чтобы изменить положение (координаты) фигуры на панели необходимо изменить ее свойства Left и Top.

4.Добавить в раздел ***Var*** глобальные переменные:

```
x0,y0,r,x,y:integer;    f:real;
```

5.Напишите процедуру обработки события **OnTimer** для таймера. Для этого двойным щелчком левой кнопки мыши выделите **Таймер** на форме и создайте программный код, реализующий движение шарика по орбите

```
procedure TForm1.Timer1Timer(Sender: TObject);
begin
f:=f+0.1;
x:=trunc(x0+r*cos(f));
y:=trunc(y0-r*sin(f));
shape1.Left:=x-shape1.Width div 2;
shape1.top:=y-shape1.Height div 2;
end;
```

6.Запустите проект на выполнение и протестируйте его.

7.Сохраните проект в папке *Движение по орбите*.

Задание для самостоятельной работы:

1. Разработать проект «Солнечная система», моделирующий перемещение планет вокруг Солнца.

2. Разработать проект «Управление движением объекта с помощью клавиатуры»

Занятие 16. Создании анимации в Delphi с применением функции Random() – генератора случайных чисел.

Цель: изучение приемов и методов создания анимированных изображений в системе программирования Delphi.

Постановка задачи: Создать приложение «Капли», в котором на поверхности формы случайным образом будет возникать окружность и, меняя цвет от черного до белого, увеличиваться в размере, имитируя капли дождя на поверхности воды (рис.16.1).

Цвет образуется смешением трех компонент — красной, зеленой и синей. Интенсивность каждой может изменяться от 0 до 255. Воспользуемся функцией RGB, для того чтобы создать цвет из отдельных компонент, заданных случайным образом:

RGB(random(256), random(256), random(256))

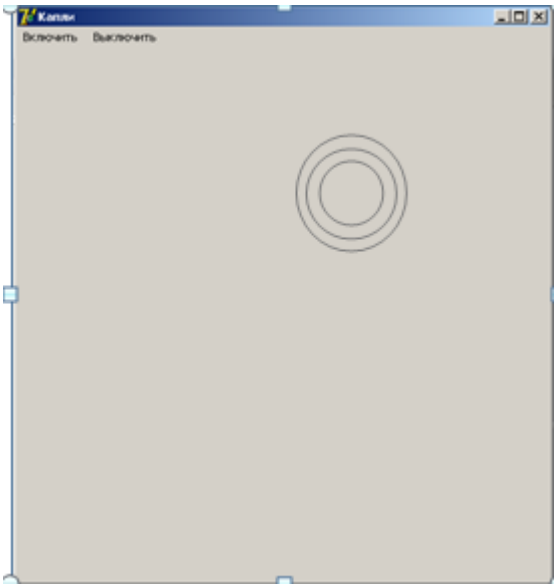


Рис. I6.1

Переменная, которой будет присвоено значение функции **RGB,** относится к классу **TColor.** Присвоим значение полученной переменной свойству **Canvas.Brash.Color** непосредственно в процедуре обработки события **OnTimer.**

План разработки программы:

1. Запустите среду программирования Delphi.

2. Задайте свойства формы:

Caption — Капли;
ClientHeight - 500;
ClientWidth - 500;
Color — clMoneyGreen.

3. Разместите на форме объект **Главное меню (MainMenu),** задав названия пунктов меню: **Включить, Выключить.**

4. Разместите на форме объект **Таймер (Timerl).** Задайте свойство **Enabled,** равное **False.**

5. Напишите процедуры обработки для пункта меню **Включить:**

```
procedure TForml.NlClickfSender:   TObject);
begin
Timerl.Enabled:=True; end;
```

и пункта меню **Выключить:**

```
procedure TForml.N2Click(Sender: TObject);
begin
Timerl.Enabled:=False;
end;
```

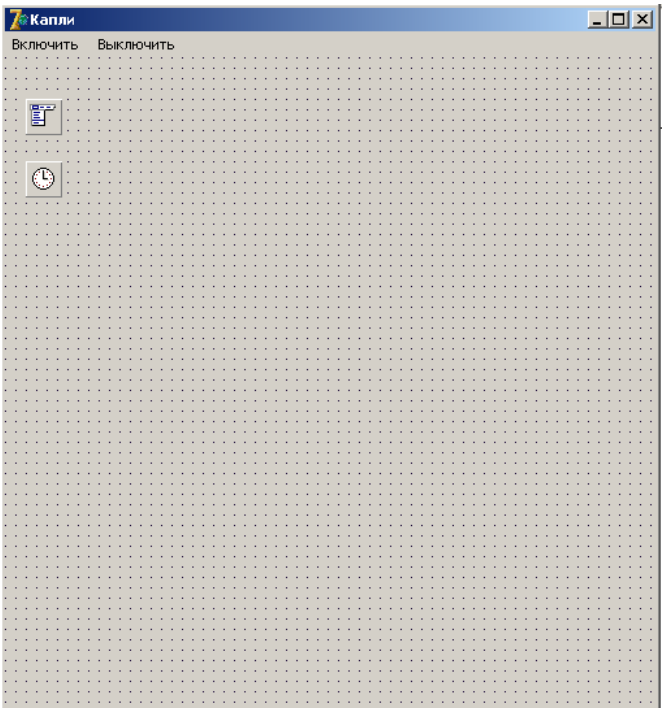


Рис. 16.2. Форма проекта «Капли»

6. Напишите процедуру обработки события **OnTimer** для таймера. Для этого двойным щелчком левой кнопки мыши выделите **Таймер** на форме и создайте программный код:

```
procedure TForml.TimerlTimer(Sender: TObject);
var
x, y, r, i: integer;
c: TColor;
 begin
   randomize;
    r:=1;
   x:=5+random(Forml.ClientWidth-10);
   y:=5+random(Forml.ClientHeight-10);
    Canvas.Pen.Width:=2;
    for i:=0 to 55 do
        begin
           c:=RGB(i*4,i*4,i*4);
          Canvas.Pen.Color:=c;
          Canvas.Ellipse(x-r, y-r, x+r, y+r);
         if i mod 2=0 then Sleep(25);
           r:=r+2;
      end;
    Canvas.Pen.Color:=clBtnFace;
Canvas.Rectangle(0,  0,  Forml.Clientwidth,  Form1.ClientHeight)
 end;
```

7. Запустите проект на выполнение и протестируйте его.
8. Сохраните проект в папке *Капли.*

Задание для самостоятельной работы:

1. Создать приложение «Мозаика», в котором на поверхности формы случайным образом будут возникать разноцветные шарики, радиус и цвет которых задается случайным образом.

2. Создать приложение «Калейдоскоп», в котором при выборе пункта меню «Пуск» вылетали восемь шариков симметрично относительно четырех линий симметрии: вертикальной, горизонтальной и двух диагоналей рабочей области формы. Координаты и радиус первого шарика задаются случайным образом.

Занятие 17. Воспроизведение звука в Delphi. Процедуры и функции воспроизведения звуков

Цель: изучение приемов и методов использования аудио и видео в приложениях, созданных в среде Delphi.

Постановка задачи: создать приложение «Демонстрация звука», которое позволит прослушать стандартные звуки Windows. Для выбора воспроизводимого звука использовать переключатели панели RadioGroup (рис.17.1).

Теоретические сведения

Процедуры и функции воспроизведения звуков:

Beep- наиболее простая процедура, управляющая звуком. Она не имеет параметров и воспроизводит стандартный звуковой сигнал, установленный в Windows;

MessageBeep - функция, она определена как

Function MessageBeep(uType:word):Boolean;

Параметр uType определяет воспроизводимый звук как идентификатор раздела sounds реестра, в котором записаны звуки, сопровождающие те или иные события Windows. С помощью приложения Звук в Панели управления Windows пользователь может удалять или устанавливать соответствующие звуки.

Для параметра uType определены следующие константы:

значение	Звук
MB_ICONASTERISK	звездочка
MB_ICONEXCLAMATION	восклицание
MB_ICONHAND	критическая ошибка
MB_ICONQUESTION	вопрос
MB_OK	стандартный звук

Если функция **MessageBeep** не находит указанный тип звука, она пытается воспроизвести стандартный звук. Если он не установлен или в компьютере отсутствует звуковая карта, то звук воспроизводится через динамик компьютера.

Примечание. Для работы приложения необходимо, чтобы на компьютере были установлены звуки соответствующие различным типам сигналов Windows. Для их установки выберите команду Звук на Панели управления. Откройте диалоговое окно Свойства: Звук.

План разработки программы «Демонстрация звука»

1. Открыть новый проект.

2. Разместить на форме компоненты в соответствии с рис.17.1

3. Выполнить следующие действия:

Выделенный объект	Вкладка окна Object Inspector	Имя свойства/ имя события	Действие
Form1	Properties	Caption	Установка имени формы «Демонстрация звука»
RadioGroup1	Properties	Caption	«Звуки»
		Items	Звездочка, восклицание, критическая ошибка, стандартный звук, вопрос, Beep
Button1	Properties	Caption	Установка имени кнопки: «Воспроизвести»

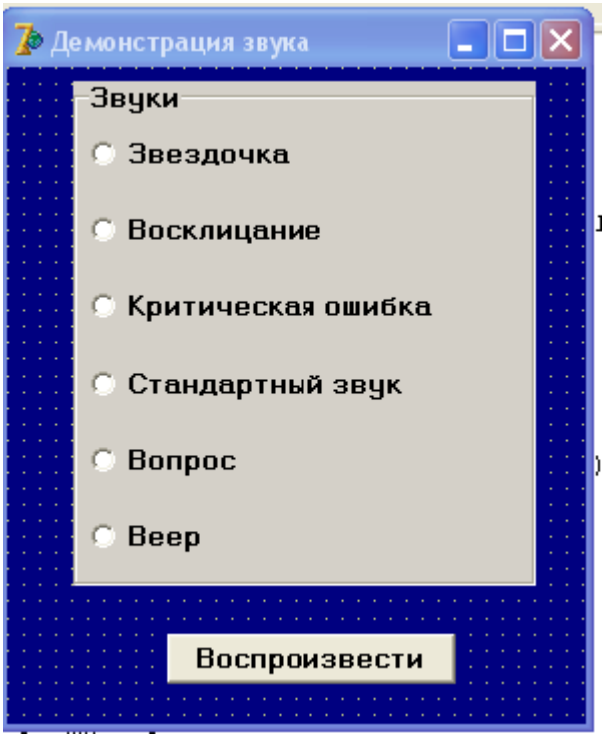


Рис. 17.1

4. Для события OnClick копки «Воспроизвести» напишите программный код обработки события. Так как в процедуре обработки нажатия кнопки Button1 должно быть шесть вариантов реализации для разных звуков, то рационально записать выбор варианта звука с помощью оператора **case of**.

Текст процедуры может быть записан следующим образом:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
case radiogroup1.ItemIndex of
0: messagebeep(mb_iconasterisk);
1: messagebeep(mb_iconexclamation);
2: messagebeep(mb_iconhand);
3: messagebeep(mb_ok);
4:  messagebeep(mb_iconquestion);
5: beep;
end;
end;
```

Задание для самостоятельной работы

Создать приложение «Тренажер таблицы умножения», в котором реакция программы на ввод верного и неверного ответа сопровождается соответствующим звуковым сигналом. Сомножители задаются случайным образом (рис.17.2).

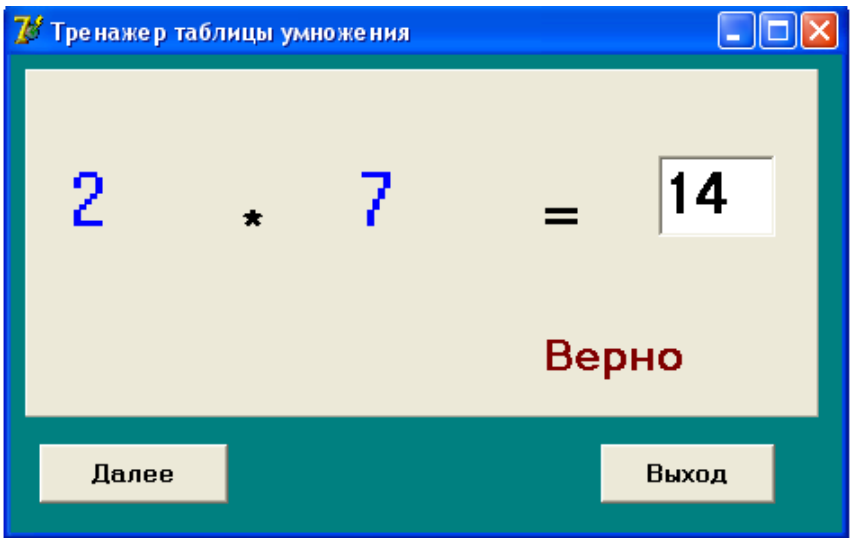


Рис.17.2

Занятие 18. Воспроизведение звука и видеоклипов в Delphi . Компонент TMediaPlayer

Цель: изучение приемов и методов использования аудио и видео в приложениях, созданных в среде Delphi.

Постановка задачи: создать приложение «Мультимедийный плеер», которое позволит прослушивать звуковые файлы формата *.wav, *.mid , *.wma,*.mp3 и воспроизводить видео формата *.avi. Приложение должно предоставлять возможность управления воспроизведением и отображать текущее состояние ролика (время, прошедшее с начала воспроизведения, длину ролика). В качестве экрана для показа ролика использовать панель. Для отображения текущего состояние объекта MediaPlayer и самого ролика использовать компонент ProgressBar; для выбора файла – компоненты TFileListBox, TDirectoryListBox, TDriveComboBox, TFilterComboBox.

Теоретические сведения

Компонент TMediaPlayer

Delphi позволяет легко и просто включать в программу такие мультимедийные объекты, как звуки, видео и музыку, сделать это можно используя встроенный в Delphi компонент TMediaPlayer (страница System Палитры Компонент Delphi). TMediaPlayer дает возможность проигрывать AVI, MIDI и WAVE файлы.

Для проигрывания файлов мультимедиа может потребоваться наличие некоторого оборудования и программного обеспечения.

Компонент TMediaPlayer (см. рис.18.1) оформлен, как панель управления устройством с кнопками: “воспроизведение”, “перемотка”, “запись” и др. Поместив компонент на форму, щелкните дважды на свойстве **FileName** и выберите имя файла с расширением AVI, WAV или MID. Установите свойство **AutoOpen** в True.

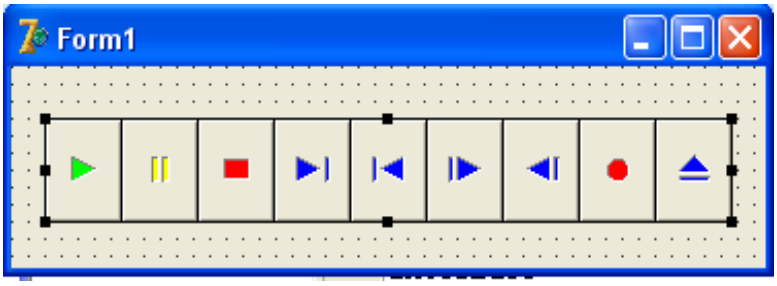


Рис.18.1: Компонент TMediaPlayer на форме.

После выполнения этих шагов программа готова к запуску. Запустив программу, нажмите зеленую кнопку “воспроизведение” (крайняя слева) и вы увидите видеоролик (если выбрали AVI) или услышите звук (если

выбрали WAV или MID). Если этого не произошло или появилось сообщение об ошибке, то возможны два варианта:

1. Вы ввели неправильное имя файла.
2. Вы не настроили правильным образом мультимедиа в Windows. Это означает, что либо у Вас нет соответствующего оборудования, либо не установлены нужные драйверы.

Еще одно важное свойство компонента TMediaPlayer - **Display.** Изначально оно не заполнено и видео воспроизводится в отдельном окошке. В качестве экрана для показа ролика можно использовать, например, панель. На форму нужно поместить компонент TPanel, убрать текст из свойствава Caption. Далее, для TMediaPlayer, в свойстве Display выбрать из списка Panel1. После этого надо запустить программу и нажать кнопку “воспроизведение” (рис.18.2)

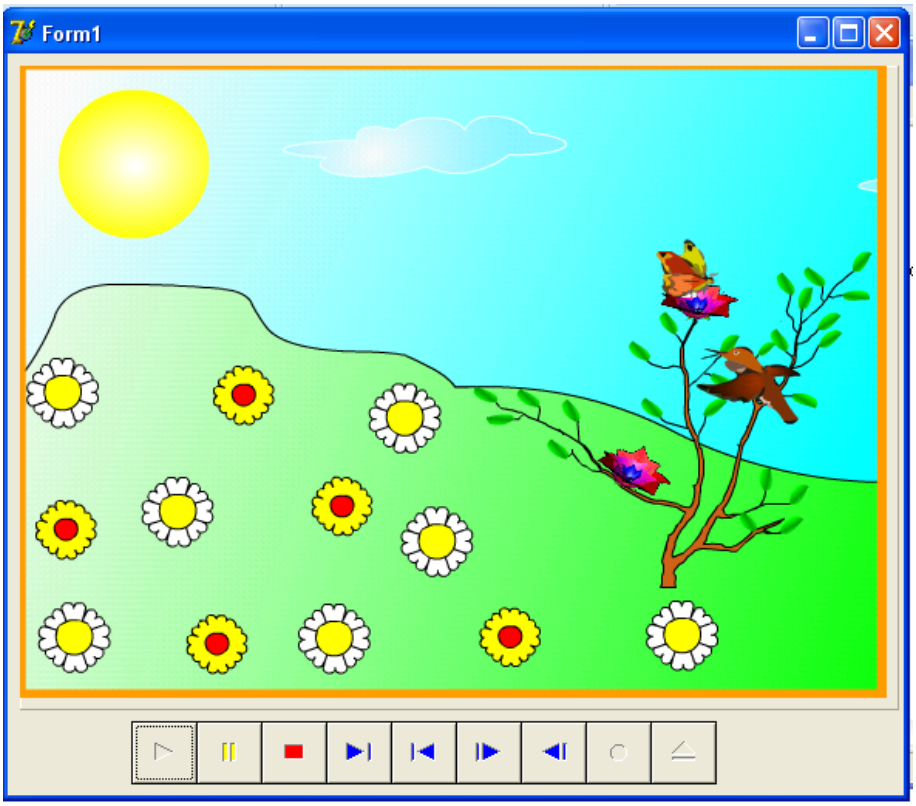


Рис.18.2: Воспроизведение AVI на панели.

Способы использования TMediaPlayer:

- предоставление возможности пользователям проигрывания максимально широкого круга файлов. В этом случае, на форме обычно располагается TMediaPlayer, предоставляющий возможность управления воспроизведением;

- программист может захотеть скрыть от пользователя существование компонента TMediaPlayer. То есть, воспроизвести звук или видео без того, чтобы пользователь заботился об их источнике. Для этого компонент делается невидимым (Visible = False) и управляется программно.

План разработки программы «Мультимедийный плеер»

1. Создайте новый проект (File | New Project).

2. Поместите TMediaPlayer на форму; поместите компоненты **TFileListBox, TDirectoryListBox, TDriveComboBox, TFilterComboBox** для выбора файла (страница Win 3.1). Внешний вид формы представлен на рис.18.4.

3. В свойстве FileList для DirectoryListBox1 и FilterComboBox1 поставьте FileListBox1.

4. В свойстве DirList для DriveComboBox1 поставьте DirectoryListBox1.

5. В свойстве Filter для FilterComboBox1 укажите требуемые расширения файлов:

avi file(*.avi)|*.avi

wave file(*.wav)|*.wav

midi file(*.mid)|*.mid

(*.wma)| *.wma

(*.mp3)|*.mp3

6. Пусть по двойному щелчку мышкой в FileListBox1 выбранный файл будет воспроизводиться. В обработчике события OnDblClick для FileListBox1 укажите

```
Procedure TForm1.FileListBox1DblClick(Sender:TObject);
begin
  with MediaPlayer1 do
  begin
    Close;
    FileName:=FileListBox1.FileName;
    Open;
    Play;
  end;
end;
```

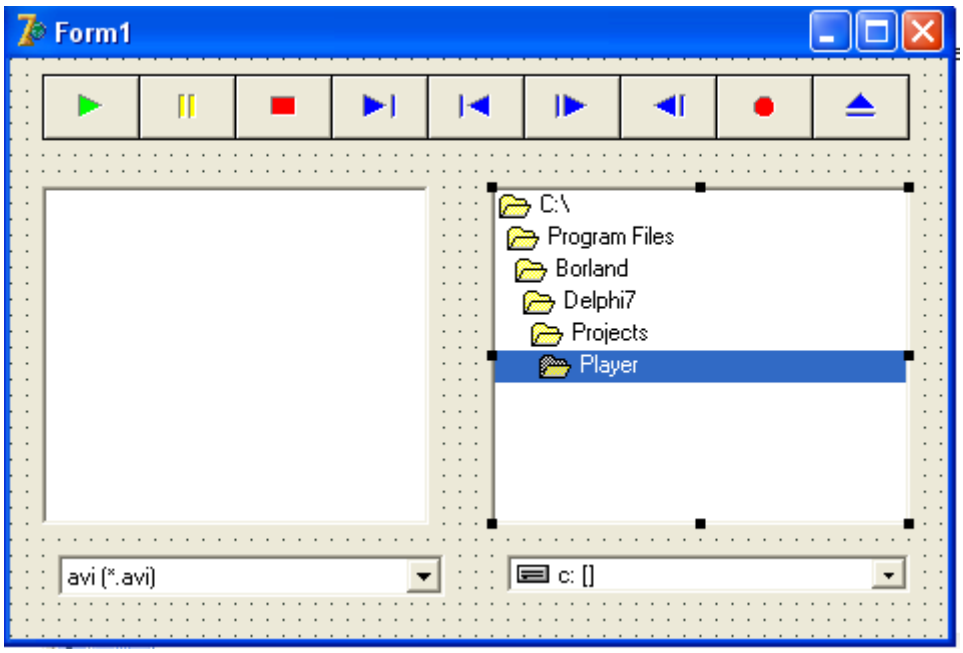


Рис.18.4: Начальный вид проекта

7. Сохраните проект, запустите его, выберите нужный файл и дважды щелкните на него мышкой. MediaPlayer должен воспроизвести этот файл в отдельном окне.

8. Видеоролик можно воспроизводить внутри формы, например, на панели. Модифицируем проект и добавим туда панель TPanel (см. рис.18.5). В свойстве Display для MediaPlayer1 укажите Panel1, уберите надпись с панели (Caption), свойство BevelOuter = bvNone. Чтобы переключаться при воспроизведении с окна на панель - поместите TCheckBox на форму и в обработчике события OnClick для него запишите:

```
procedure TForm1.CheckBox1Click(Sender: TObject);
var
  Start_From : Longint;
begin
  with MediaPlayer1 do begin
    if FileName='' then Exit;
    Start_From:=Position;
    Close;
    Panel1.Refresh;
    if CheckBox1.Checked then
      Display:=Panel1
    else
      Display:=NIL;
    Open;
    Position:=Start_From;
    Play;
  end; end;
```

Запустите проект и воспроизведите видеоролик. Пощелкайте мышкой на CheckBox.

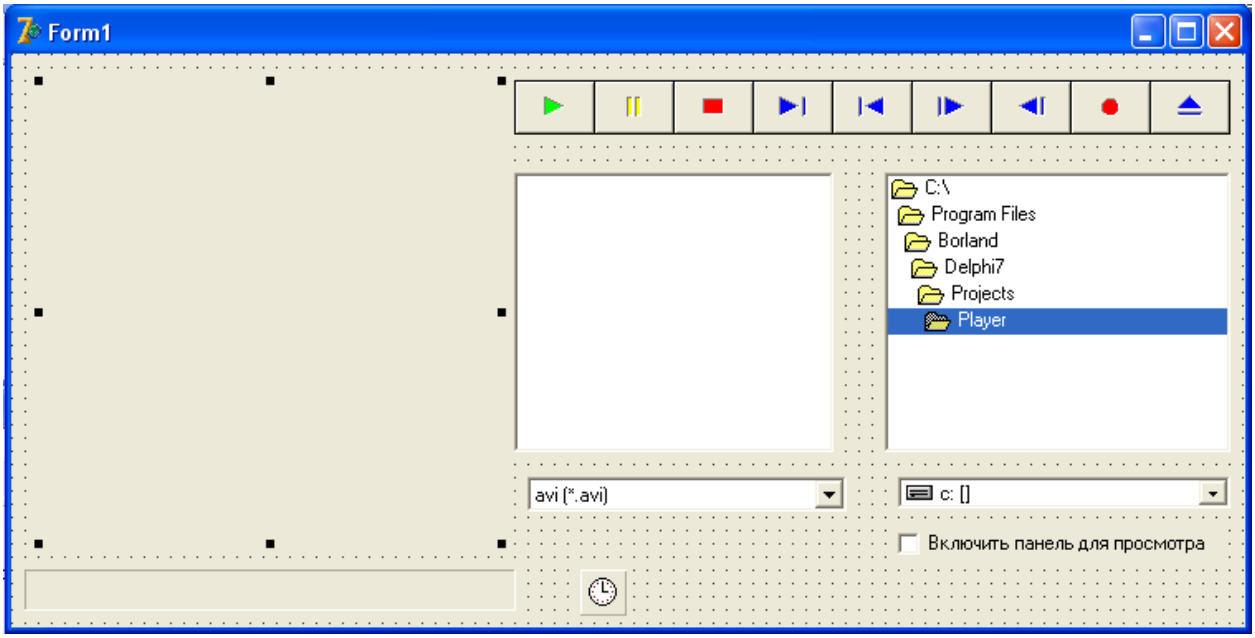


Рис.18.5. Добавлена панель для воспроизведения видео и переключатель окно/панель.

9.Во время выполнения программы может потребоваться отобразить текущее состояние объекта MediaPlayer и самого ролика (время, прошедшее с начала воспроизведения, длину ролика). Для этого у объекта TMediaPlayer есть соответствующие свойства и события: Length, Position, OnNotify и др. Добавим в проект прогресс-индикатор ProgressBar (страница Win 3.1), который отобразит в процентах, сколько прошло времени (рис.18.6).

10. Для обновления показаний индикатора можно воспользоваться таймером. Поместите на форму объект TTimer, установите для него Interval = 100 (100 миллисекунд). В обработчике события OnTimer нужно записать:

```
procedure TForm1.Timer1Timer(Sender: TObject);
begin
  with MediaPlayer1 do
    if FileName<>'' then
      ProgressBar1.Position:=Round(100*Position/Length);
end;
```

11.Запустите проект, выберите файл (AVI) и щелкните на нем два раза мышкой. При воспроизведении ролика прогресс-индикатор должен отображать процент, соответствующий прошедшему времени (рис.18.6).

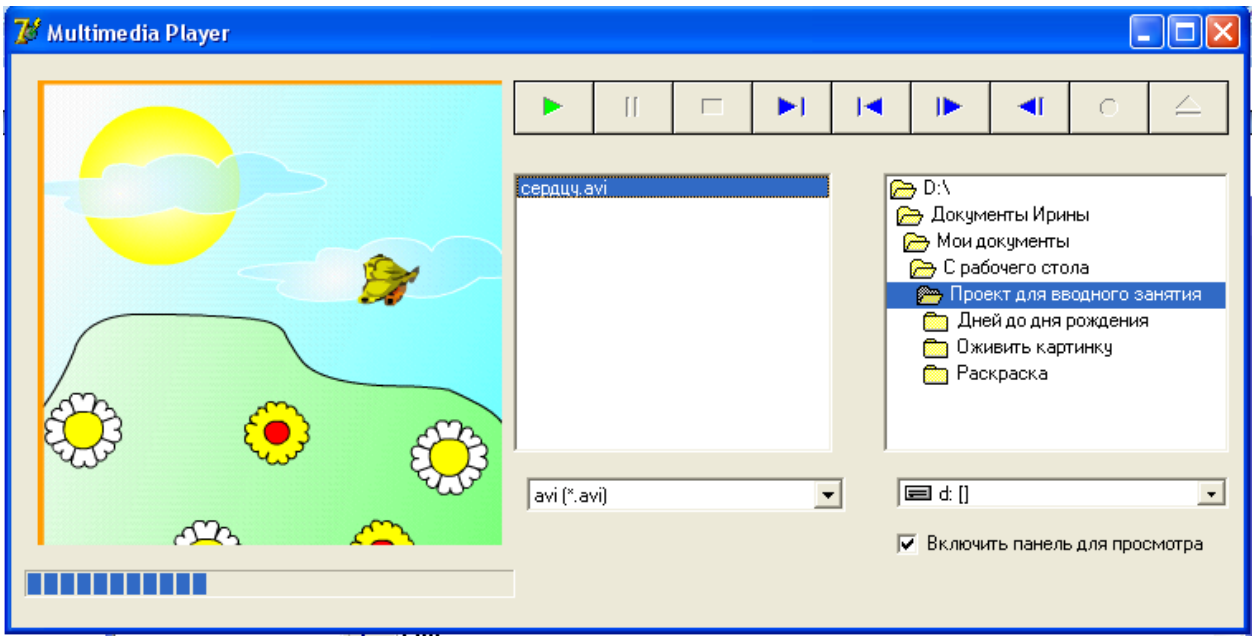


Рис.18.6. Законченное приложение

Задание для самостоятельной работы:

Создать приложение «Электронные часы - будильник», которое отображает системное время и подает звуковой сигнал в установленное пользователем время.

Примечание. Для получения системного времени использовать тип TDateTime и процедру DecodeTime, которая определена как:

```
procedure DecodeTime(Time: TDateTime; var Hour, Min, Sec, MSec: Word);
```

Процедура позволяет получить из значения переменной типа TDateTime (дата-время) часы, минуты, секунды, миллисекунды.

Пример:

```
var
  P:TDateTime;
  Hour, Min, Sec,  MSec: Word;

Begin
……
P:=now;
DecodeTime(P, Hour, Min, Sec, MSec);
……
```

Занятие 19. Создание и обработка одномерного массива. Динамические массивы

Цель: изучение приемов и методов обработки одномерных массивов при создании приложений в среде программирования Delphi 7.

Постановка задачи: Создайте приложение, которое предлагает пользователю задать одномерный массив, заполняет этот массив случайными целыми числами, выводит список элементов массива, а затем по выбору пользователя определяет минимальный и максимальный элементы массива, сумму всех элементов и количество четных элементов.

План разработки программы

1. Создайте форму по предложенному образцу. На форме расположите компоненты Edit1 и Edit2, кнопку Button1, для свойства Caption кнопки задайте значение «Создать массив». Разместите на форме панель GroupBox1, для свойства Caption которой задайте значение «Определить».
2. В панели GroupBox1 разместите компоненты CheckBox1, CheckBox2, CheckBox3, CheckBox4, для свойств Caption которых задайте значения (рис.19.1). Напротив них расположите компоненты Edit3, Edit4, Edit5, Edit6. В нижней части формы разместите кнопку «Вычислить».
3. Выровняйте компоненты на форме, зафиксируйте их положение, выбрав в меню Delphi команду Edit + Lock Controls.
4. Сохраните файл проекта и программного модуля.

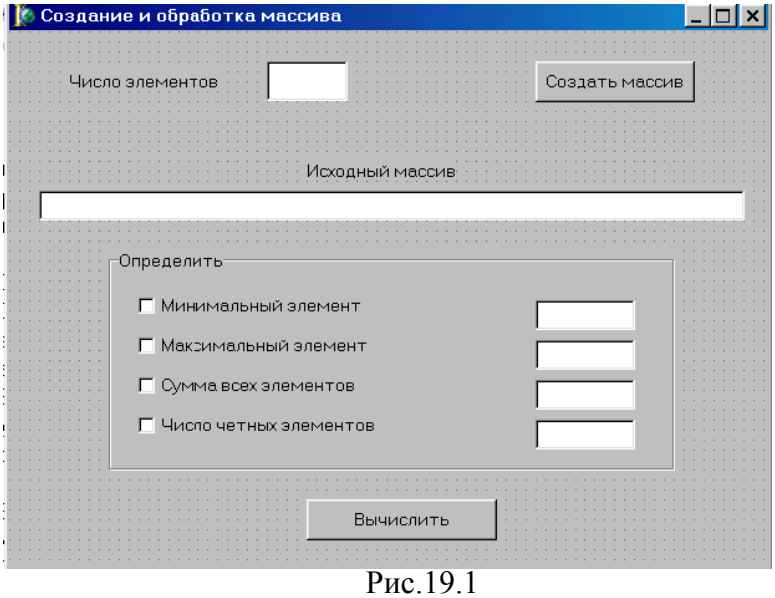


Рис.19.1

5. В разделе описания переменных модуля опишите переменные целого типа: N - размер массива, I - порядковый номер элемента массива, M– динамический массив целых чисел:

```
Var
   Form1:Tform1;
N, I: integer;
M: array of  integer;   {описание динамического массива целых чисел}
```

Примечание Динамические массивы отличаются от статических тем, что для них заранее не объявляется длина – число элементов. При объявлении динамического массива место под него не отводится. Нумерация элементов массива начинается с нуля. Прежде, чем использовать массив в программе, надо задать его размер процедурой **SetLength**, например SetLength(M,N) - задать массиву M длину N.

6. Для запрета ввода любых символов, кроме цифр от 0 до 9 в окне Edit1, для события OnKeyPress в процедуру Edit1KeyPress вставить оператор:

```
If not (key in [‘0’..’9’]) then key:=#0;
```

Действие этого оператора этого оператора: если символ нажатой клавиши не входит в множество от 0 до 9, то key присваивается значение нулевого символа (#0). В окне Edit1 будет отображаться текст, состоящий только из цифр (рис.19.2).

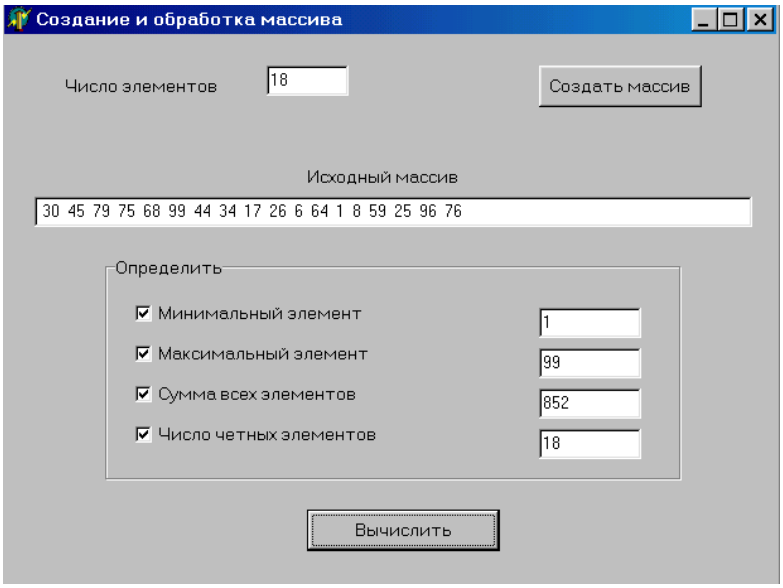


Рис.19.2

7. Создание массива целых чисел опишите в процедуре обработчика события OnClick кнопки Button1. Программный код может быть записан следующим образом:

```
procedure TForm1.Button1Click(Sender: TObject);
var i:integer;
begin
edit2.text:='';
n:=strtoint(edit1.Text);
setlength(m,n);
```

```
randomize;
for i:=1 to n do
 begin
  m[i]:=random(99)+1;
  edit2.text:=edit2.Text+inttostr(m[i])+' ';
 end;

end;
```

8. Обработку массива запрограммируйте в процедуре обработчика события OnClick кнопки Button2. Текст процедуры обработки массива может быть записан следующим образом:

```
procedure TForm1.Button2Click(Sender: TObject);
     var
        i:integer;
     begin
        min:=m[1];
        max:=m[1];
        for i:=1 to n do
         begin
         if checkbox1.Checked then
          begin
          if min>m[i] then min:=m[i];
          edit3.Text:=inttostr(min);
          end;
          if checkbox2.Checked then
           begin
           if max<m[i] then max:=m[i];
           edit4.Text:=inttostr(max);
           end;
          if checkbox3.Checked then
           sum:=sum+m[i];  edit5.Text:=inttostr(sum);
          if checkbox4.checked then
           if m[i] mod 2=0 then
            begin
            sum2:=sum2+1;
            edit6.Text:=inttostr(sum2);
            end;
          end
end;
```

9. Сохраните файлы проекта, запустите программу на выполнение. Задавая различные количества элементов массива и варианты его обработки, убедитесь в правильной работе программы.

Задание для самостоятельной работы:

1. Внесите изменения в проект таким образом, чтобы массив формировался из отрицательных и положительных целых чисел, подсчитывалось количество положительных и количество отрицательных элементов массива.

2. Создать приложение, формирующее целочисленный массив с количеством элементов n. Программа «сжимает» массив, выбрасывая из него каждый второй элемент.

3. оздать приложение, которое предлагает пользователю создать массив случайных целых чисел и указать направление сортировки массива (по возрастанию, по убыванию); выполняет сортировку и выводит результат.

Занятие 20. Создание и обработка двумерного массива. Использование компонента StringGrid для представления двумерных массивов

Цель: изучение приемов и методов обработки двумерных массивов при создании приложений в среде программирования Delphi 7.

Постановка задачи: Создайте приложение, которое выводит двумерный массив случайных целых чисел в объекте StringGrid и определяет минимальный и максимальный элементы, а также сумму элементов массива, расположенных на главной диагонали.

План разработки программы

1. Создайте форму, свойству Caption которой присвойте значение «Обработка двумерного массива».
2. Выберите в палитре компонентов страницы Additional компонент StringGrid и расположите его в левом верхнем углу формы. Задайте для свойств ColCount (Количество столбцов) и RowCount (Количество строк) значения 6. Задайте значения для свойств FixedCols и Fixed Rows (количество фиксированных, не прокручиваемых столбцов и строк, используемых для размещения номеров столбцов и строк).

✍Примечание. Компонент StringGrid представляет собой таблицу, содержащую строки. Таблица может иметь полосы прокрутки, причем заданное число первых строк и столбцов может быть фиксированным и не подвергаться прокрутке. Таким образом, можно задать заголовки столбцов и строк, постоянно присутствующие в окне компонента. Каждой ячейке таблицы может быть поставлен в соответствие некоторый объект.

3. Справа от объекта StringGrid разместите кнопку Buttonl и задайте для ее свойства Caption значение «Заполнить».
4. Ниже объекта StringGrid расположите панель GroupBoxl и присвойте ее свойству Caption значение «Определить».
5. На этой панели разместите компоненты CheckBoxl, CheckBox2, ChekBox3 и присвойте их свойствам Caption значения «Минимальный элемент», «Максимальный элемент», «Сумма элементов главной диагонали» соответственно.
6. Справа от компонентов CheckBoxl, CheckBox2, CheckBox3 разместите компоненты Editl, Edit2, Edit3 и удалите текст «Editl», «Edit2», «Edit3» из соответствующих компонентов.
7. Правее панели GroupBoxl расположите кнопку Button2 и задайте для ее свойства Caption значение «Вычислить». Выровняйте компоненты на форме, как показано на рис.20.1, и зафиксируйте их положение на форме.
8. Сохраните файл проекта и программного модуля.

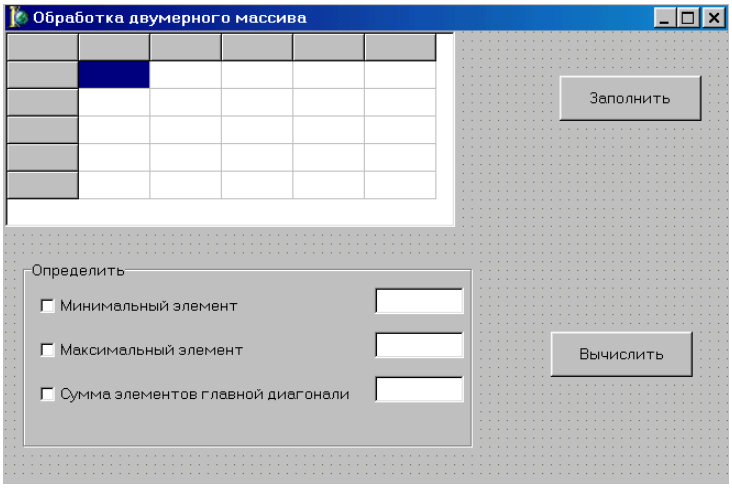


Рис.20.1

9. Прежде чем создавать обработчики событий щелчка мышью по кнопкам Buttonl и Button2, следует добавить в раздел описания переменных данного модуля целочисленные переменные I и J, предназначенные для хранения индексов массива (I — номер столбца, J — номер строки):

```
var
Forml: TForml;
I, J : integer;
```

10. Для получения подсказки Delphi по объекту StringGridl, указав объект, нажмите F1. В окне Delphi Help просмотрите общую информацию о назначении объекта. Щелкнув мышью по ссылке Properties (Свойства), откройте окно со списком свойств объекта и, выбирая нужные свойства, например, Cells, ColCount и т. п., просмотрите справочную информацию. Для возврата к предыдущему экрану справки воспользуйтесь кнопкой Назад в панели инструментов окна Delphi Help. Щелкая мышью по ссылкам Methods (Методы) и Events (События), просмотрите список методов и событий объекта. Для просмотра примеров следует щелкнуть мышью по ссылке Example.

Примечание. Используйте оператор with .. do для обращения к элементам объекта. Его применение позволяет сократить записи при обращении к свойствам и методам объекта. В операторе, следующем за ключевым словом do, можно не указывать ссылки на поля, свойства и методы объекта. При этом каждый идентификатор в операторе, совпадающий с именем поля, свойства, метода объекта, трактуется как относящийся к этому объекту, и к нему неявно добавляется ссылка на этот объект. Например, вместо того чтобы, обращаясь к ячейке объекта StringGridl, писать StringGridl.Cells[I,J], удобнее использовать оператор with StringGridl do, в теле которого можно неоднократно обращаться к Cells[I, J], не упоминая имени объекта StringGridl.

11. Создайте процедуру обработчика события щелчка мышью по кнопке Buttonl, к которой сначала будет выполнена операция вывода номеров строк и столбцов, а затем ячейки StringGridl будут заполнены случайными целыми

числами. Для этого выберите в окне Инспектора объектов объект Buttonl и на странице События сделайте двойной щелчок на пустом поле списка в событии OnClick. После этого отредактируйте заготовку процедуры обработчика этого события следующим образом:

```
procedure TForm1.Button1Click(Sender: TObject): {заполнение массива}
Begin
Randomize;
with StringGrid1  do    {вывести номера строк и столбцов}
begin
 I:=0;                        {столбец 0}
 for  J: = 1  to  RowCount - 1  do    {вывести номера строк}
Cells[I,J] := IntToStr(J);
J:=0;                                 {строка 0}
  for  I:= 1  to  ColCount - 1  do        {вывести номера столбцов}
Cells[I,J] := IntToStr(I);
end;
with  StringGrid1  do    {вывести в таблице элементы двумерного массива}
for I:= 1  to  ColCount - 1 do
   for J:= 1 to RowCount – 1  do
begin
     Cells[I,J] := IntToStr(Round(Sin(Random(100))*100)):
    End;
```

12. Для создания процедуры обработки массива выберите в окне Инспектора объектов объект Button2 и на странице События сделайте двойной щелчок на пустом поле списка в событии OnClick. После этого отредактируйте заготовку процедуры обработчика этого события следующим образом:

```
procedure TForm1.Button2Click(Sender: TObject);   {обработка массива}
var
Min, Max, Sum : Integer;     {локальные переменные}
begin
if Checkbox1.Checked  then{определение Min-элемента}
 with StringGrid1  do
begin
Min:=StrToInt(Cells[1,1]);          {пусть это Min-элемент}
 for I := 1  to  ColCount - 1 do
      for J:= 1 to RowCount - 1 do
          if StrToInt(Cells[I,J])<Min then
            Min:=StrToInt(Cells[I,J]);
 Edit1.Text:=IntToStr(Min);
 End
else Edit1.Text:='';
if Checkbox2.Checked   then      {определение Max-элемента}
```

```
with StringGrid1 do
   begin
    Max:=StrToInt(Cells[1,1l]);       {пусть это Max-элемент}
    for I := 1 to ColCount - 1 do
     for J: = 1 to RowCount - 1 do
         if StrToInt(Cells[I.J])>Max then
           Max:=StrToInt(Cells[I,J]);
    Edit2.Text:=IntToStr(Max);
 end
 else Edit2.Text:='';
 if Checkox3.Checked  then  {вычисление Sum}
    with StringGrid1 do
      begin
        Sum:=0;                {обнулить сумму перед подсчетом}
        for I:= 1 to ColCount - 1 do
        Sum:=Sum+StrToInt(Cells[I,I]);
Edit3.Text:=IntToStr(Sum);
 end
 else Edit3.Text:='';
 end;
```

Как видно из текста процедуры, в ней имеется три фрагмента, каждый из которых выполняет вычисления, если свойство Checked соответствующего флажка (Checkbox1, Checkbox2 или Checkbox3) имеет значение True. Операторы типа Editl.Text: = '';, Edit2.Text: = '';, Edit3.Text:=''; обеспечивают очистку соответствующего окна Edit, если вычисление не проводилось. Для обращения к элементу массива, расположенному на главной диагонали StringGrid, указываются одинаковые номера строки и столбца, Cells[I,I].

13. Сохраните файлы проекта и программного модуля, откомпилируйте и запустите программу на выполнение. Щелкнув мышью по кнопке Заполнить, проверьте заполнение надписей номеров строк и столбцов, а также заполнение объекта StringGrid значениями элементов двумерного массива. Убедитесь, что приложение правильно обрабатывает двумерный массив.

Задание для самостоятельной работы:

1. Создайте приложение, выводящее в окно двумерный массив – табель успеваемости с оценками по нескольким предметам за 1-4 четверти и за год. Годовая оценка должна вычисляться как среднее арифметическое всех четвертных оценок по данному предмету. Используйте закраску ячеек таблицы.
2. Создайте приложение с использованием компонента StringGri, обеспечивающее пользователю возможность ввода значений элементов двумерного массива в ячейки и вычисляющее сумму элементов массива.

Занятие 21. Создание текстового файла. Запись данных в текстовый файл. Чтение данных из текстового файла

Цель: изучение приемов и методов работы с текстовыми файлами (создание, запись, чтение) при создании приложений в среде программирования Delphi 7.

Постановка задачи 1: Создайте приложение, которое **создает текстовый файл** textl.txt и записывает в него текст, введенный пользователем в окно Edit, после чего закрывает файл.

План разработки программы

1. Открыть новый проект.
2. Создайте форму и задайте для ее свойства Caption значение «Создание файла и вывод в него текста из Edit».
3. Разместите на форме компоненты Editl, Labell и Buttonl (рис.21.1). Задайте значения для свойств Labell.Caption — «Введите текст», Buttonl.Caption — «Сохранить». Удалите текст «Editl» из соответствующего компонента. Выровняйте компоненты и зафиксируйте их положение на форме.

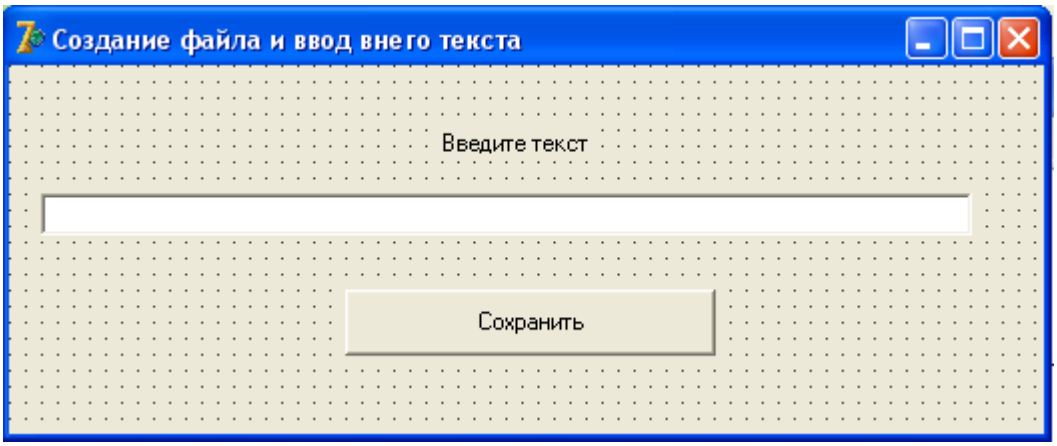


Рис. 21.1. Окно формы

4. Сохраните файлы модуля (под именем main) и проекта (под именем TextEditFile) в папке «Обработка текстовых файлов». Откомпилируйте приложение и запустите его на выполнение. Убедитесь в том, что приложение позволяет пользователю вводить текст в окно Edit1.
5. Для того чтобы на диске создавался текстовый файл и текст из окна Edit1 записывался в него, создайте процедуру обработчика события щелчка мышью по кнопке Button1 «Сохранить» и в ней реализуйте создание нового файла и запись в него текста.

6. Дополните текст процедуры следующим кодом:

```
var
```

```
f:TextFile;                    {описание файловой переменной}
begin
   AssignFile(f, 'textl.txt');  {связь файловой переменной с файлом}
   Rewrite(f);                  {создать новый файл для записи}
   Writeln(f, Editl.Text);      {записать в файл}
    CloseFile(f);               {закрыть файл}
end;
```

В начале процедуры файловая переменная f связывается с именем файла textl.txt, а затем на диске создается новый файл с этим именем. После этого записывается значение свойства Text объекта Editl, и файл закрывается. Сохраните файл модуля под именем Main, файл проекта — под именем TextEditFilel, откомпилируйте программу и запустите ее на выполнение.

В окне приложения введите текст и щелкните мышью по кнопке Сохранить. Закройте окно приложения и откройте в окне Проводника Windows папку «Обработка текстовых файлов», в которой сохранены файлы проекта. В списке файлов этой папки вы увидите и вновь созданный файл textl.txt.

Постановка задачи 2. Создайте приложение, **открывающее текстовый файл** для чтения и считывающее из него текст в окно Memo. Перед открытием файла следует проверить его наличие; в случае отсутствия файла должно выводиться соответствующее сообщение.

Примечание. Многострочное окно редактирования Memo используется для ввода, отображения и редактирования многострочных текстов. Свойство Lines, доступное как во время проектирования, так и во время выполнения, имеет большое количество свойств и методов типа Strings, которые обычно используются для формирования и редактирования текста. Весь текст содержится а свойстве Text.

План разработки программы:

1. Создайте форму и задайте для ее свойства Caption значение «Чтение текста из файла в окно Memo».
2. На форме Forml разместите компоненты Memol, Label1 и Button1 (рис.21.2), задайте значения для свойств Labell.Caption — «Текст из файла», Buttonl.Caption — «Прочитать текст из файла».

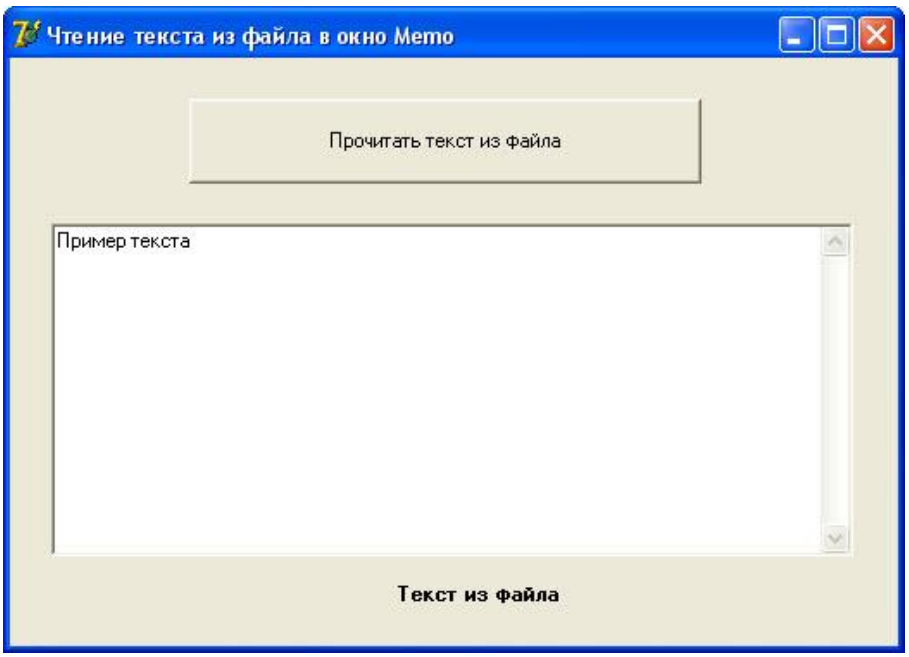


Рис. 21.2 Окно формы

3. Для удаления текста «Memol» из окна компонента Memol выберите в окне Инспектора объектов объект Memol, затем на странице Свойства дважды щелкните мышью на поле значения Strings свойства Lines для формирования и редактирования текста. После этого в окне String List Editor удалите текст «Memol» (рис.21.3) и щелкните мышью по кнопке ОК.

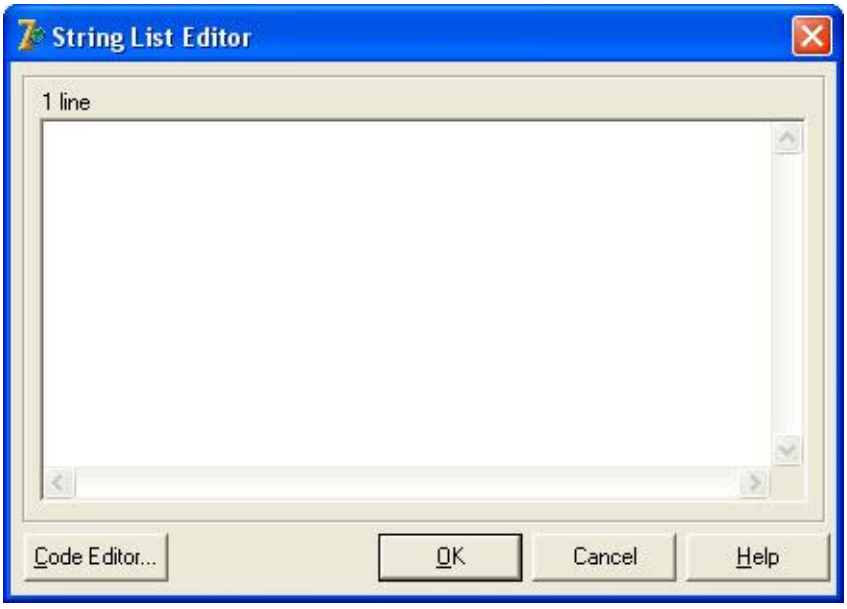


Рис. 21.3. Редактирование текста в окне Memol

4. Для обеспечения возможности просмотра в окне Memol длинных текстов с использованием вертикальной полосы прокрутки в окне Инспектора объектов выберите объект Memol и на странице Свойства установите для свойства ScrollBars значение ssVertical. Выровняйте

компоненты и зафиксируйте их положение на форме.

5. Выбрав в окне Инспектора объектов объект Buttonl, на странице Событий сделайте двойной щелчок на пустом поле списка в событии OnClick. После этого в окне Редактора кода будет сгенерирована заготовка процедуры обработчика события. Чтобы эта процедура выполняла открытие текстового файла и выводила текст в окно Memo1, отредактируйте текст процедуры следующим образом:

```
procedure TForml.ButtonlClick(Sender: TObject);
var
   F:TextFile;
   Ch : Char;
   begin
   AssignFile(F,'textl.txt');
{$I-}          {директива компилятора: отключить проверку ошибок ввода-
                    вывода}
   Reset (F);     {открыть файл для чтения}
   {$I+}
    If  IOResult = 0 then begin      (если операция «Открыть файл»
выполнена успешно}
      while not Eof(f) do    {пока не конец файла}
        begin
          Read(F,Ch);          {прочитать из файла символ}
       Memol.Text:=Memol.Text+Ch; {вывести символ в поле Memol}
       end:
       CloseFile(F);           {закрыть файл}
                    end
 else                          {IOResult <>0 - операция «Открыть файл» не
 выполнена}
   ShowMessage(‘Нет такого файла');
end;
```

Как видно из текста процедуры, после связывания файловой переменной с именем файла при помощи специальной директивы компилятора отключается контроль ошибок ввода-вывода для того, чтобы проверить функцией IOResult успешность операции. В случае успешности операции открытия файла на чтение, производится чтение файла по одному символу и вывод его в окно Memol. Условием прекращения чтения из файла является достижение конца файла (Eof(f)=True). После этого файл закрывается. Если открыть файл не удается, то IOResult возвращает значение, отличное от нуля, при этом в отдельном окне выводится соответствующее сообщение.

Примечание: Директивы компилятора $I позволяют включить или отключить автоматический контроль результата вызова процедур ввода-

вывода Object Pascal. Если используется директива {$I+}, то при возвращении процедурой ввода-вывода ненулевого значения генерируется исключение EInQutError и в его свойство errorcode заносится код ошибки. Таким образом, при использовании директивы {$I+} операции ваода-вывода располагаются в блоке try …except, имеющем обработчик исключения EInOutError. Если такого блока нет, то обработка производится методом TApplication.HandleException. Если используется директива {$I-}, то исключение не генерируется. В этом случае можно проверить наличие ошибки, обратившись к функции IOResult. Эта функция возвращает код ошибки, который можно проанализировать.

6. Сохраните файл модуля под именем main1, а файл проекта — под именем TextMemoFile1 в папке Обработка текстовых файлов. Откомпилируйте и запустите приложение. Щелкнув на кнопке Прочитать текст из файла, убедитесь, что текст из файла считывается в окно Memol.

Постановка задачи 3. Создайте приложение, **открывающее текстовый файл** для дополнения и затем **добавляющее** в него введенный текст**.**

План разработки программы:

1. Создайте форму и задайте для ее свойства Caption значение «Добавление текста в файл» (рис. 21.4).
2. На форме Form1 разместите компоненты Button1, Memo1, Label1 и, как показано на рис. 21.4, присвойте значения свойствам Label1.Caption — «Текст из файла», Buttonl.Caption — «Прочитать текст из файла». Под Label разместите Label2 и задайте для свойства Label2.Caption значение «Добавляемый текст». Ниже Label2 разместите на форме объект Edit1. Под объектом Edit1 расположите кнопку Button2 и задайте для свойства Buttonl.Caption значение «Добавить текст в файл». Удалите текст из окон компонентов Memol, Editl. Для обеспечения возможности просмотра в окне Memol длинных текстов с использованием вертикальной полосы прокрутки в окне Инспектора объектов выберите объект Memol и на странице Свойства установите для свойства ScrollBars значение ssVertical. Выровняйте компоненты и зафиксируйте их положение на форме.
3. Создайте обработчики нажатий кнопок Прочитать текст из файла и Добавить текст в файл. Для создания процедуры обработчика события нажатия кнопки Добавить текст в файл выберите в окне Инспектора объектов объект Button2, на странице События сделайте двойной щелчок на пустом поле списка в событии OnClck. После этого в окне Редактора кода будет сгенерирована заготовка процедуры обработчика события.

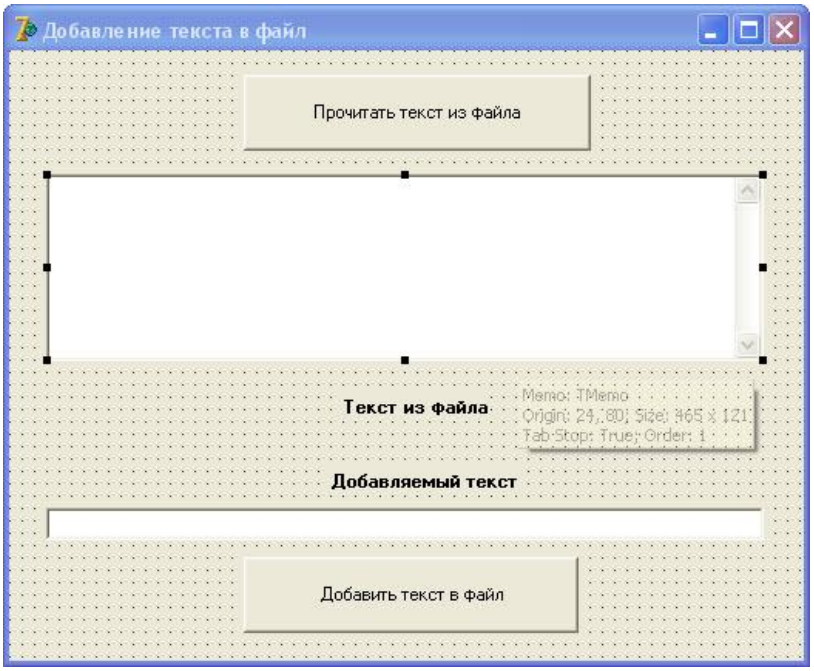


Рис. 21.4 Окно формы

4. Чтобы эта процедура открывала существующий текстовый файл и добавляла в него текст, являющийся значением свойства Text объекта Edit1, отредактируйте текст процедуры следующим образом:

```
procedure TForml.Button2Click(Sender: TObject);
var
f:TextFile;                    {описание файловой переменной}
begin
   AssignFile(f, 'textl.txt');   {связь файловой переменной с файлом}
   Append(f);                    {открыть файл для дозаписи}
   Writeln(f, Editl.Text);       {записать в файл}
    CloseFile(f);                {закрыть файл}
end;
```

5.Сохраните файл модуля под именем Main2, а файл проекта — под именем TextMemoFile2 в папке Обработка текстовых файлов. Откомпилируйте и запустите приложение. Щелкнув мышью по кнопке Прочитать текст из файла, убедитесь, что текст из файла на диске считывается в окно Memo1.

Введите текст в окно Editl и щелкните мышью по кнопке Добавить текст в файл. Щелкнув по кнопке Прочитать текст из файла, просмотрите текст, прочитанный из файла, и убедитесь, что при повторном считывании из файла считывается текст, включающий фрагмент, только что добавленный вами. Закройте окно приложения.

Задание для самостоятельной работы

Создайте приложение, которое открывает текстовый файл с использованием метода OpenDialog, считывает текст из него в объект Memo, сохраняет измененный текст в файле с использованием метода SaveDialog.

Занятие 22. Создание типизированного файла. Запись данных в типизированный файл

Цель: изучение приемов и методов работы с типизированными файлами (создание, запись, чтение) при создании приложений в среде программирования Delphi 7.

Постановка задачи. Создайте приложение, которое создает типизированный файл anketa.dan и записывает в него анкетные данные учащихся, после чего закрывает файл.

План разработки программы:

1. Открыть новый проект.
2. Спроектируйте форму (рис.22.1) и задайте для ее свойства Caption значение «Ввод данных в типизированный файл». Для ввода даты рождения используйте компонент MaskEdit, настройте его для ввода даты по шаблону чч.мм.гггг.

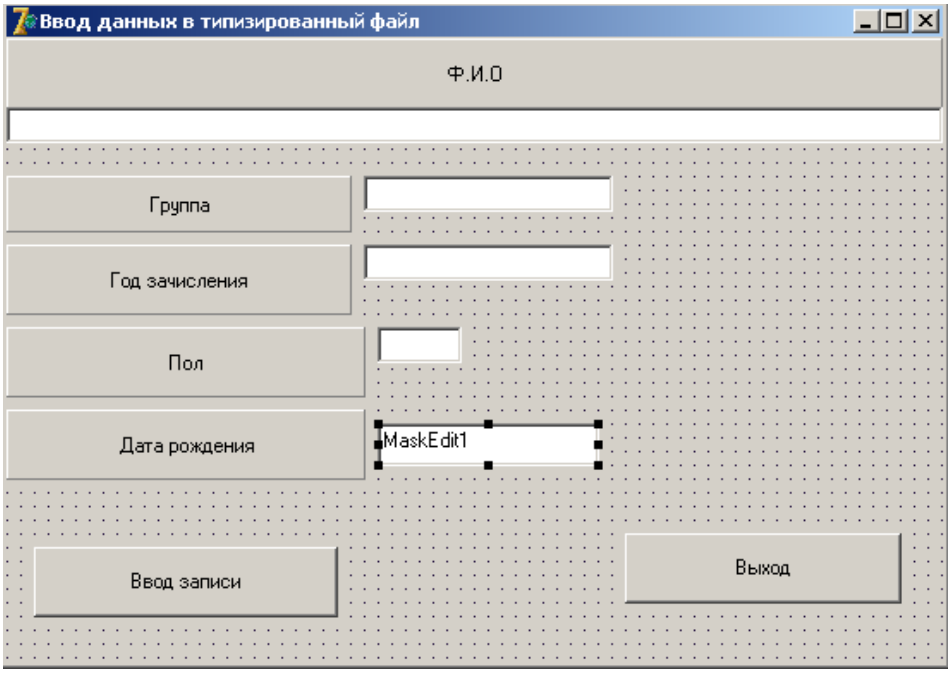


Рис. 22.1 Окно формы

3. Сохраните файлы модуля (под именем main) и проекта (под именем RecordFile) в папке «Обработка типизированных файлов». Откомпилируйте приложение и запустите его на выполнение. Убедитесь в том, что приложение позволяет пользователю вводить текст в окна Edit.

Указания к написанию программного кода:

4. Для ввода анкетных данных используйте тип запись uchenik состоящую из полей:

```
type
    uchenik=record
        fio:string[30];
        gruppa:string[5];
        god:integer;
        pol:char;
        data_r:string[8]
    end;
```

5. Объявите файловую переменную и переменную для обмена данными с файлом.

```
var
  Form1: TForm1;
   f:file of uchenik;
 pupil:uchenik;
implementation


{$R *.dfm}
```

6. Дополните текст процедуры
procedure(TForm1.FormCreate(Sender: TObject)
следующим кодом:

```
    begin
assignfile(f,'anketa.dan');
rewrite(f);
end;
```

7. Для кнопки Ввод записи напишите соответствующий обработчик.

Задание для самостоятельной работы

Создайте приложение, которое открывает типизированный файл anketa.dan, выполняет его чтение и подсчет количества мальчиков и девочек в учебной группе.

Заключение

Не так давно программирование, ориентированное на объекты и события, казалось непрофессионалам большой экзотикой. Учащиеся, умеющие работать в Windows и его приложениях, совершенно естественно воспринимают среду разработки Delphi. Конструирование форм с различными визуальными компонентами и работа с инспектором объектов вызывает у ребят неподдельный интерес. Ряд учащихся с эмпирическим стилем мышления успешно конструирует весьма интересные приложения, ограничиваясь минимальным применением базовых управляющих структур при разработке процедур. В то же время за этими структурами они видят теперь не средства для организации трудно воспринимаемых абстрактных вычислений, а средства решения очень интересной задачи. Ребята с развитым теоретическим стилем мышления, освободившись от рутины организации интерфейса в Турбо Паскале и жестких рамок его возможностей, выходили на разработку серьезных развитых проектов с использованием разнообразных компонентов Delphi и хорошей проработкой кода процедур. По моему мнению, современные технологии конструирования Windows-приложений на Delphi во многом более доступны учащимся, чем традиционное программирование, вызывают большой интерес, способствуют раскрытию творческого потенциала ученика и обладают замечательными дидактическими возможностями.

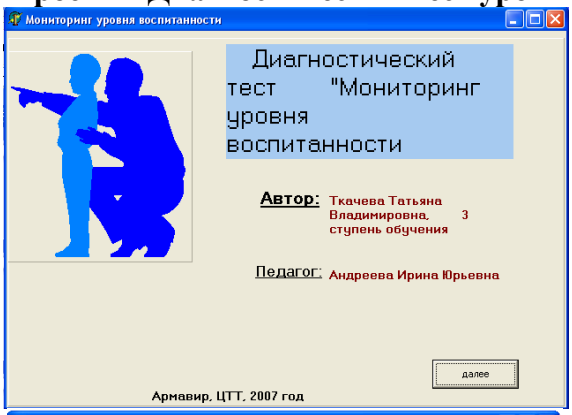
Литература

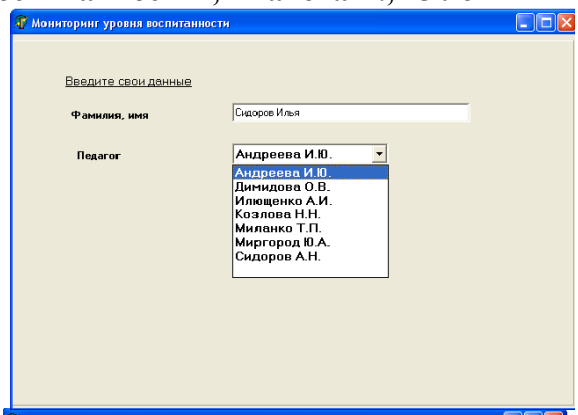
1. Архангельский А.Я. Программирование в Delphi 6, М. Бином, 2002
2. Блинова Е.Е., Спицина Л.Г. Практикум по объектно-ориентированному программированию в среде Delphi, Армавир, 2003
3. Буч Г. Объектно-ориентированное проектирование с примерами применения. – И.: Конкорд, 1992 .
4. Гутман Г.Н. Учебные мини-проекты на Delphi. М.:Чистые пруды, 2005.
5. Информатика в школе. №1, М.: Образование и информатика,2010.
6. Павлова И.М., Власова Н.Г. Delphi для учителя в примерах. Информатика в школе: Приложение к журналу «Информатика и образование».2007.№1
7. Попов В. Паскаль и Дельфи. Учебный курс. Питер, 2005.
8. Симонович С., Евсеев Г. Занимательное программирование в Delphi, М. АСТ-ПРЕСС-КНИГА, 2001
9. Стефан Моррис. Объектно-ориентированное программирование. Серия «Enter».Ростов-на-Дону:Феникс,1997.Керниган Б., Пайк Р. Практика программирования.- СПб.: Невский диалект, 2001

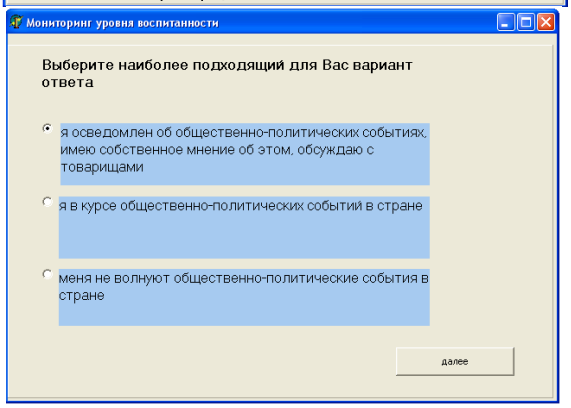
Приложение 1

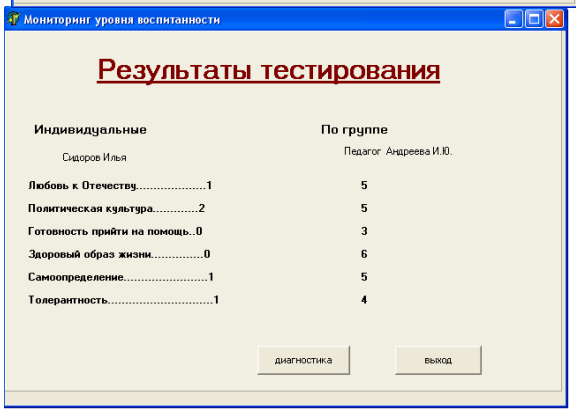
Примеры творческих проектов выполненных учащимися ЦНТТ

Проект «Диагностический тест уровня воспитанности», Ткачева Т., 15 лет

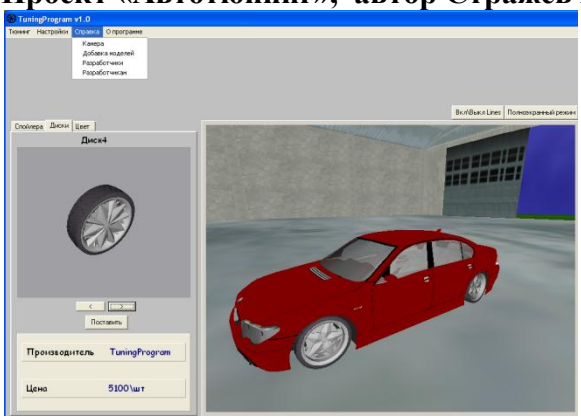


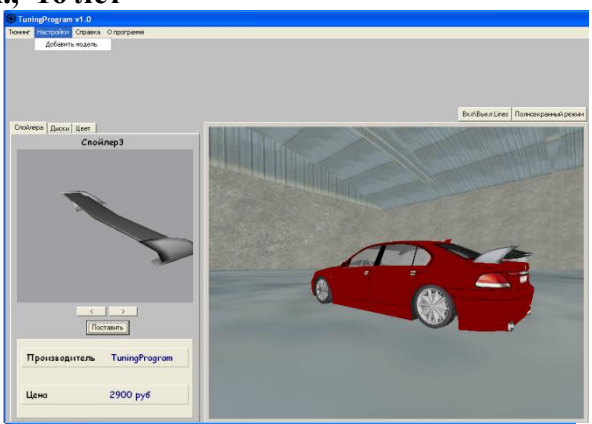


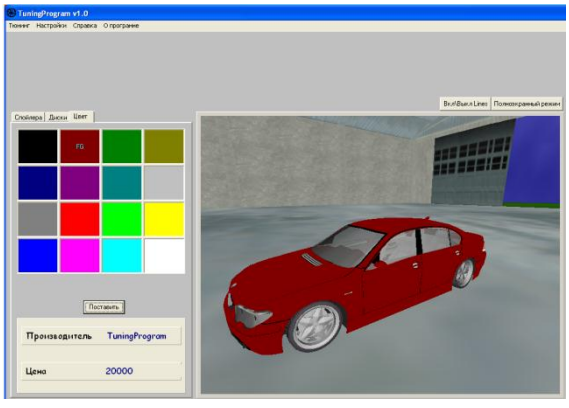


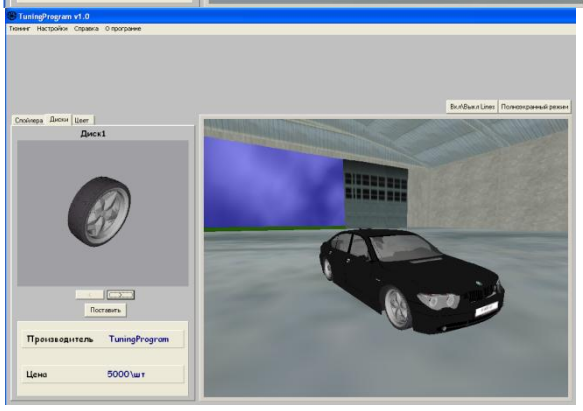


Проект «Автотюнинг», автор Стражев А., 16 лет

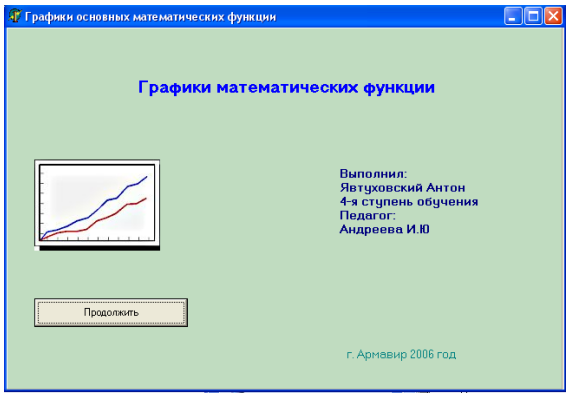


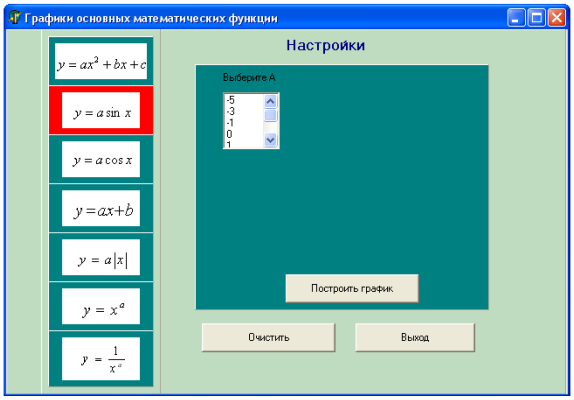


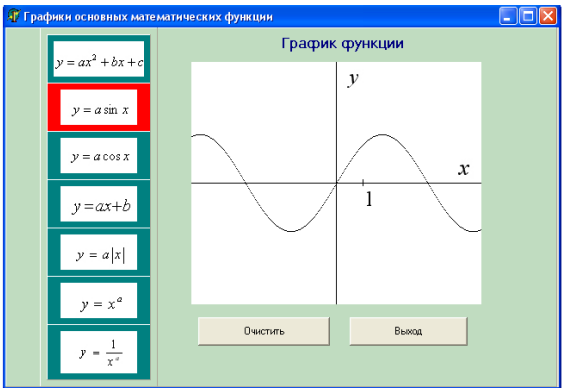


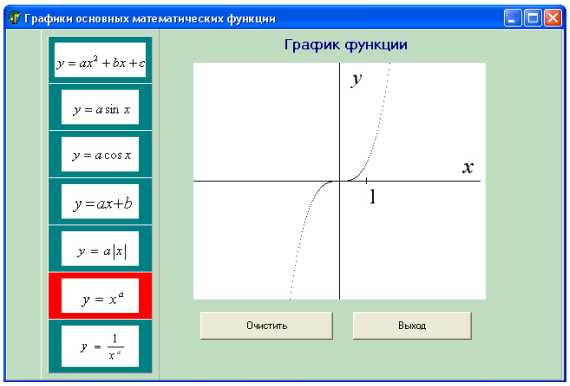


Проект «Графики функций», Явтуховский Антон, 15 лет

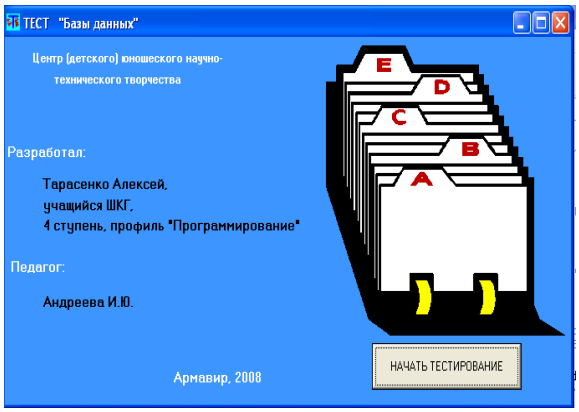


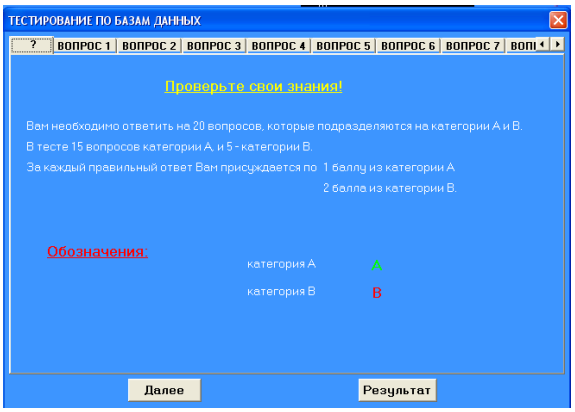


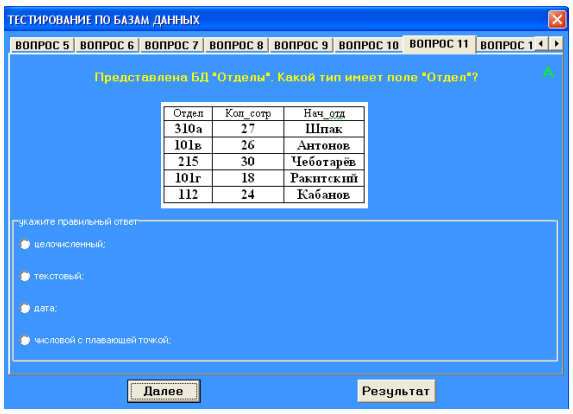


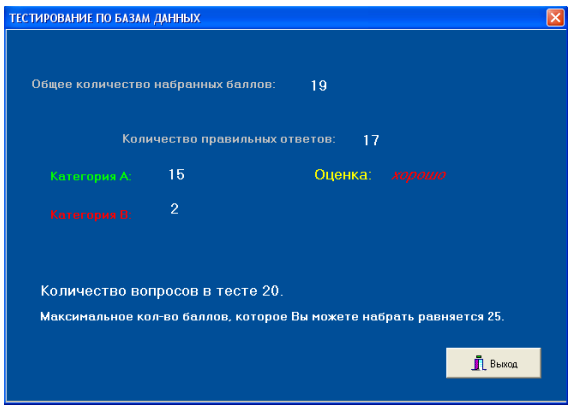


Тест «Базы данных», автор Тарасенко А., 16 лет

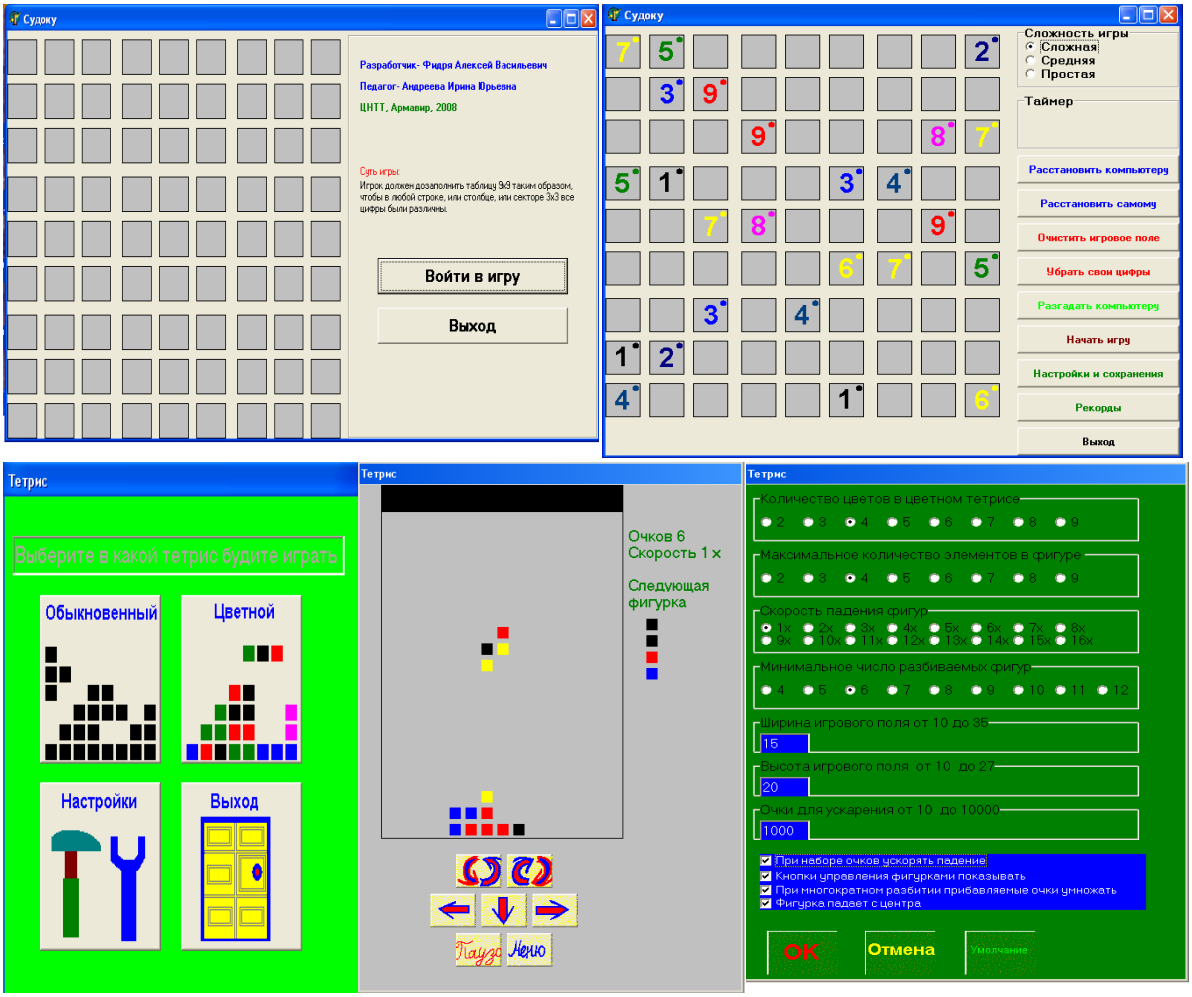






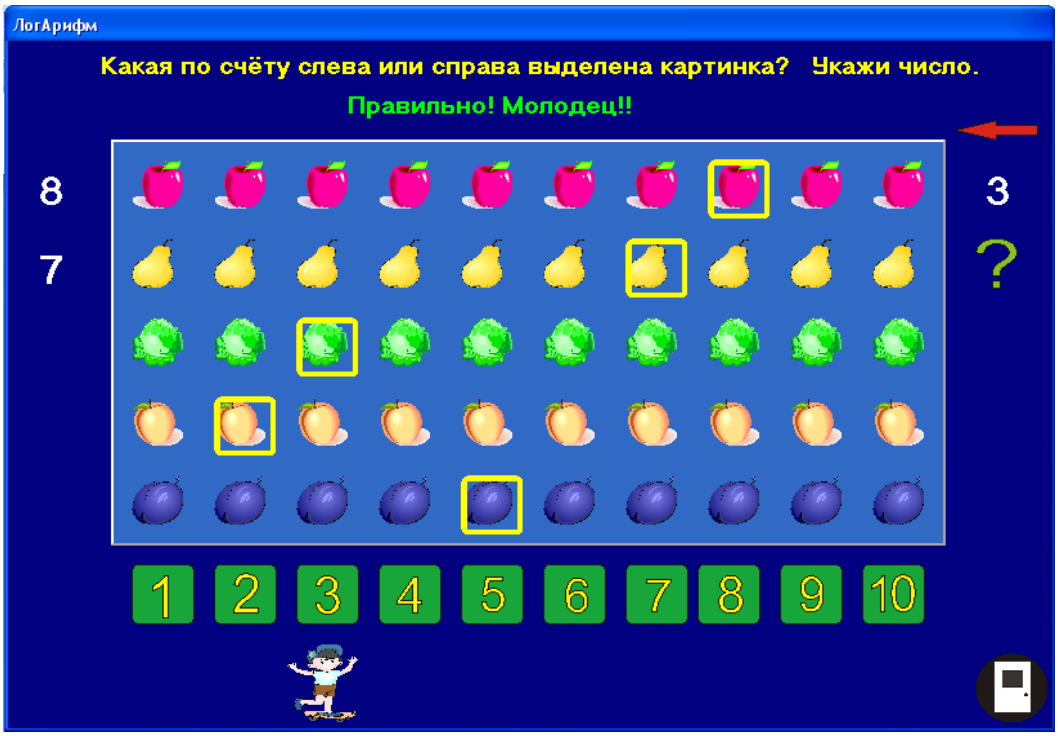


Игры «Судоку», «Тетрис», автор Фидря А., 17 лет

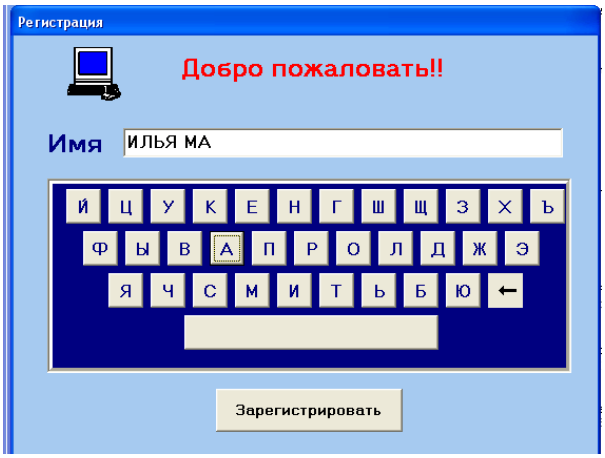


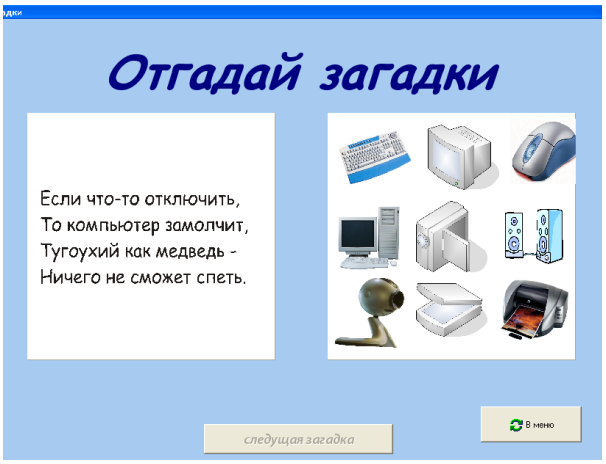
G

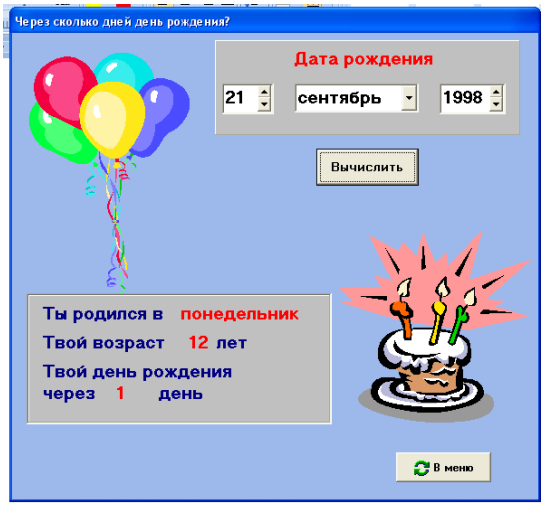
Проект «ЛогАрифм»- считалка для дошкольников, автор Глазков Д., 17 лет

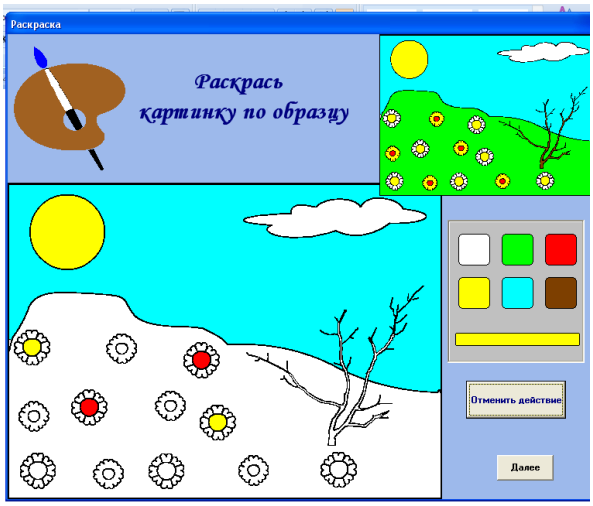


Проекты «Виртуальная клавиатура», «Загадки о компьютере»,»День рождения», «Раскраска», автор Малыхин Илья, 15 лет

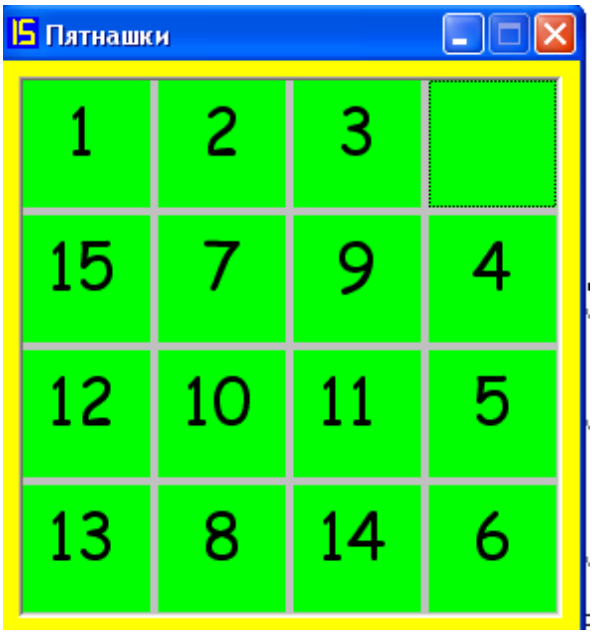




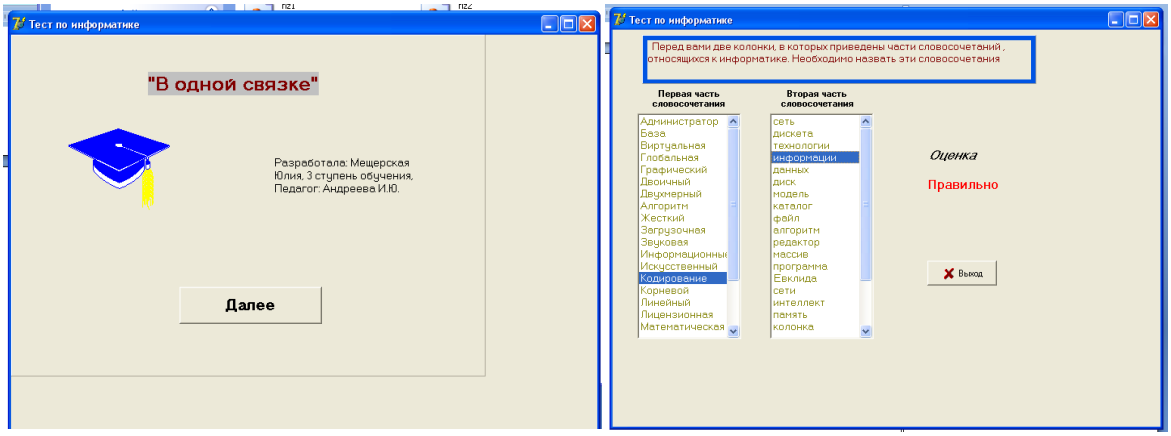




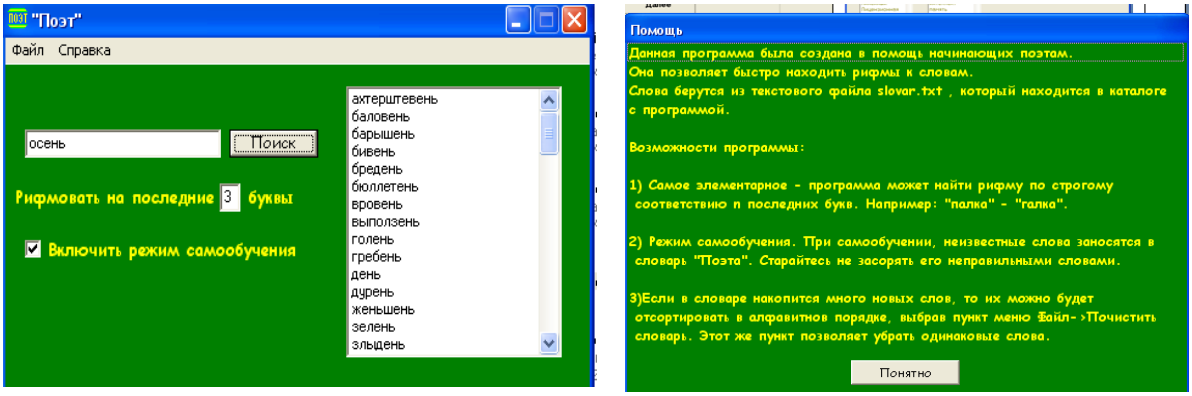
Игра «Пятнашки», автор Шаповалов Игорь, 15 лет



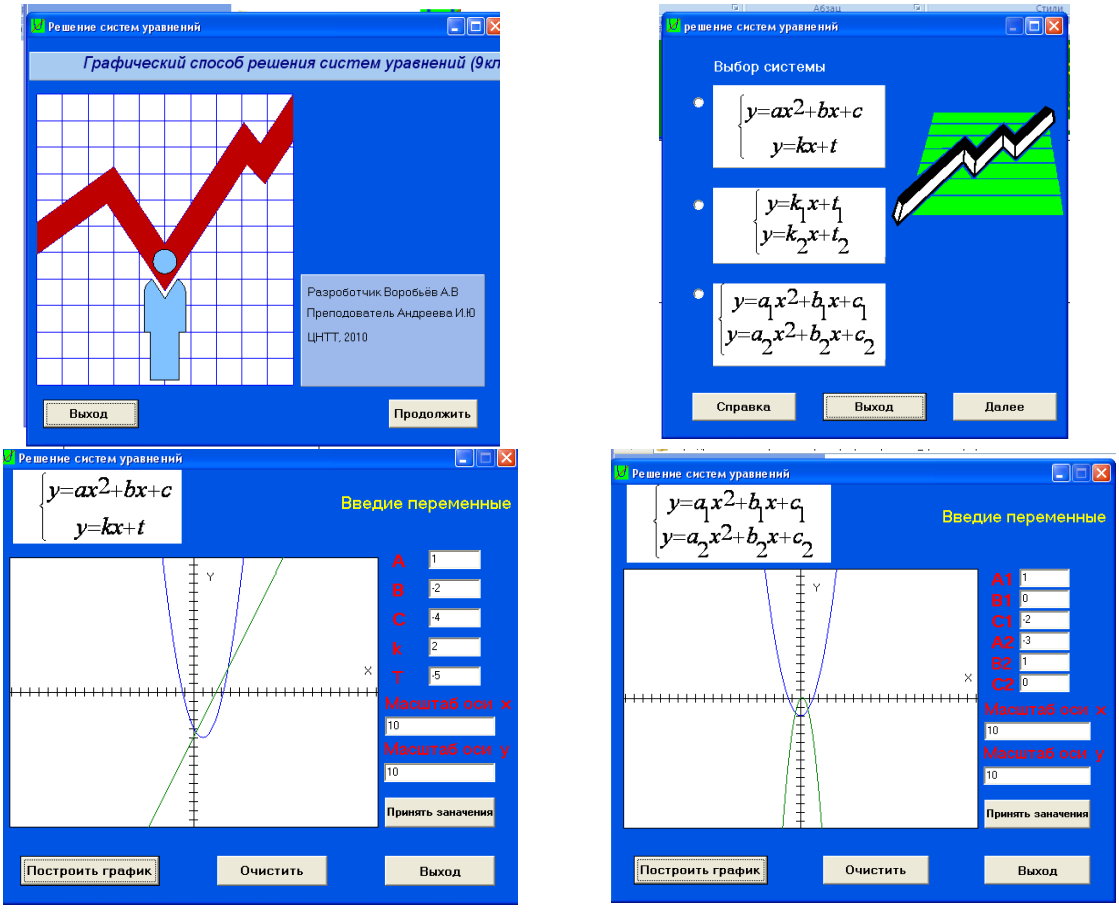
Тест « В одной связке», автор Мещерская Ю., 15 лет



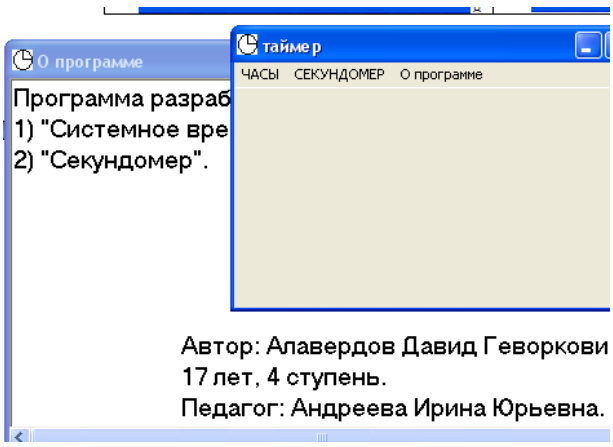
Проект «Поэт», автор Еремин В., 15 лет

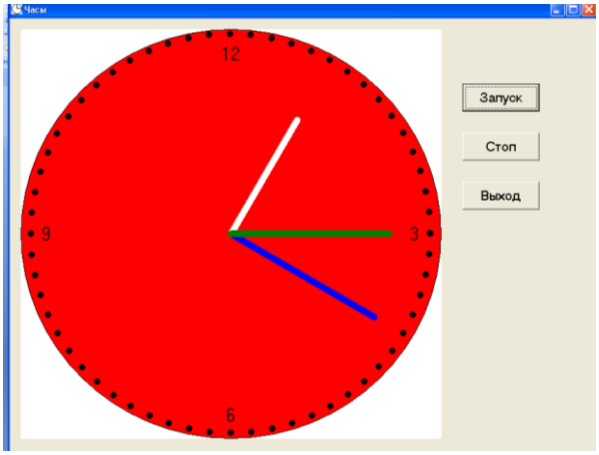


Проект «Решение систем уравнений в 9 классе», автор Воробьев А., 15 лет

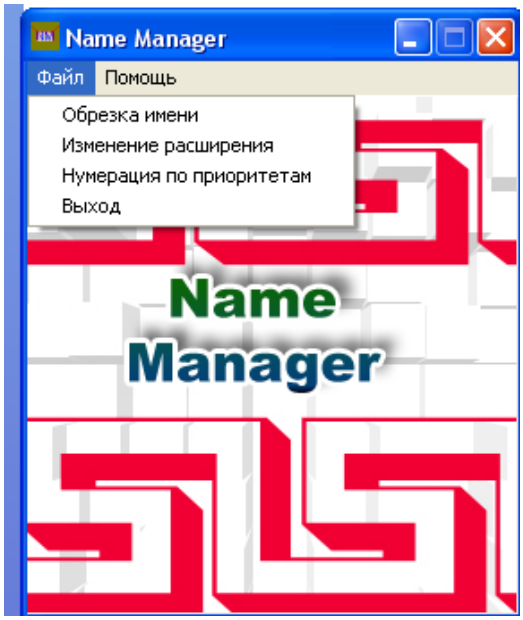


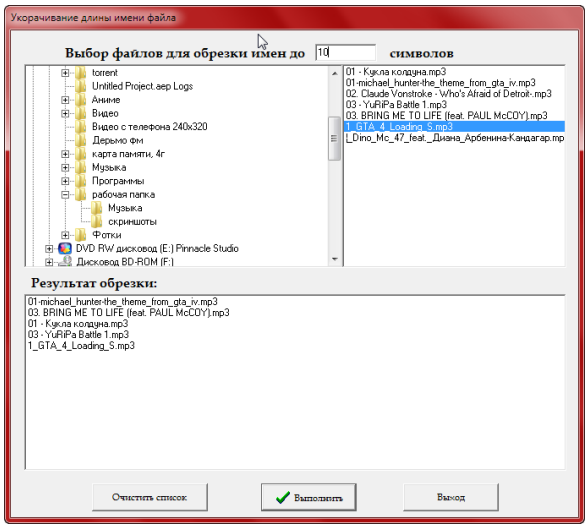
Проект «Таймер», автор Алавердов Д., 17 лет

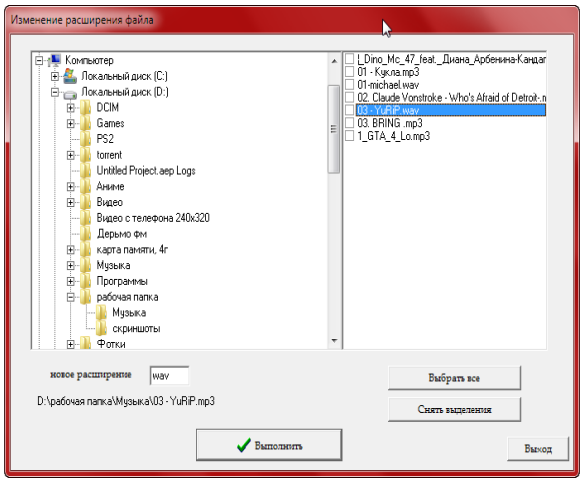


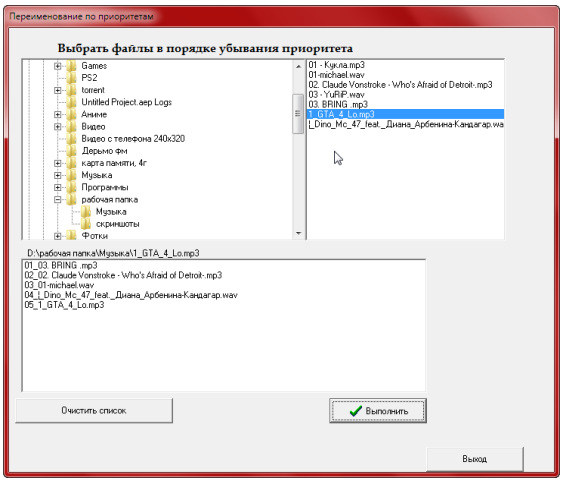


Проект «Менеджер имен файлов», автор Есионов А., 16 лет

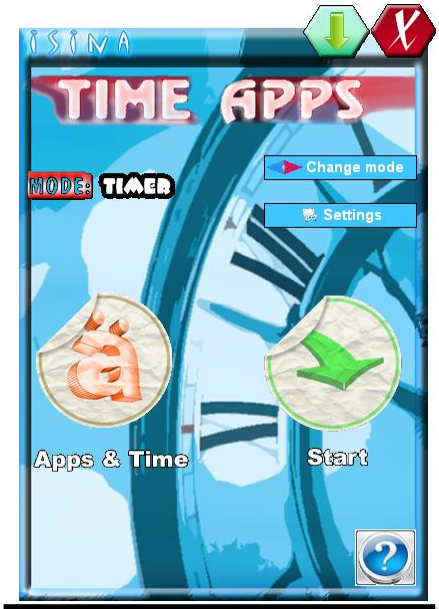


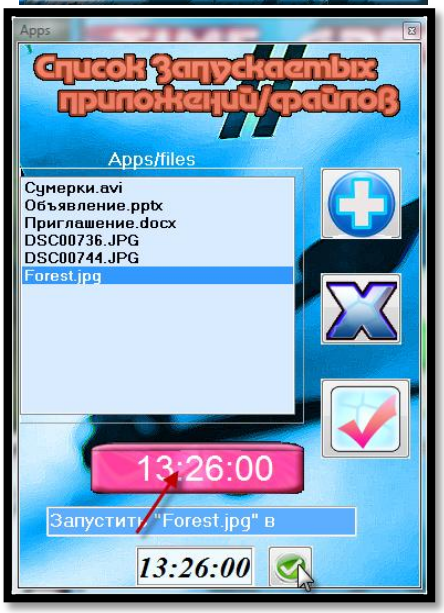






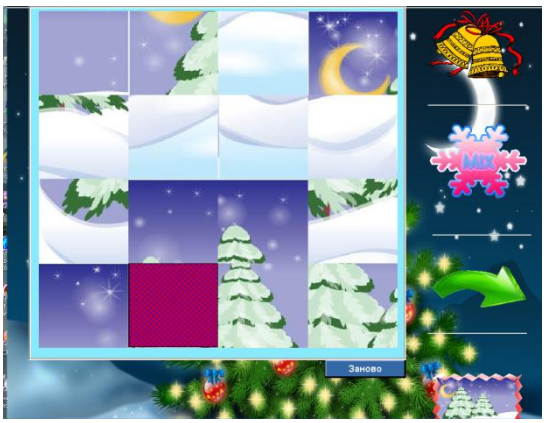
Сервисная программа «Планировщик задач Windows», Клименко С., 15 лет





Утилита «USB Blocker» . Защита ПК от несанкционированного подключения съемных носителей информации, Клименко С., 15 лет

Игра «Новогодний пазл», автор Клименко С., 15 лет



Проект «Сборник кроссвордов», автор Комардин А., 16 лет