Рекомендовано к печати
учебно-методическим советом
НОУ ВПО "Академия управления "ТИСБИ"

Программирование и программное обеспечение информационных технологий. Часть I. Учебно-методическое пособие / Л.Б. Таренко. – Казань: НОУ ВПО "Академия управления "ТИСБИ", 2009.

Автор:
старший преподаватель кафедры информационных технологий **Л.Б. Таренко**
Рецензенты:
доцент кафедры информационных технологий НОУ ВПО "Академия управления "ТИСБИ", к.т.н. **А.Н. Козин;**
доцент кафедры информационных технологий управления Академии государственного и муниципального управления при президенте РТ, к.т.н. **Е.А. Печеный.**

Учебно-методическое пособие содержит материал для проведения занятий по программированию в среде Delphi. Изложены основы теории программирования, на базе которой изучаются основные возможности Delphi и отрабатываются технологии разработки различных приложений. Пособие содержит большое число заданий с рекомендациями по программной реализации, а также задания для самостоятельного практического решения.

© Таренко Л.Б. 2009
© НОУ ВПО "Академия управления "ТИСБИ", 2009

Оглавление

Введение 4
Выписка из Госстандарта 6
1. Основы создания приложений в среде визуального программирования 7
1.1. Классификация языков программирования 7
1.2. История развития среды разработки Delphi 9
1.3. Общая характеристика системы разработки приложений Delphi 14
1.4. Файлы, составляющие приложения Delphi 17
1.5. Обзор основных компонентов 19
2. Практические задания 23
2.1. Создание простейшей программы 23
2.2. Развитие простейшей программы 26
2.3. Реализация программы-калькулятора 31
2.4. Реализация основных операций со списком 35
2.5. Работа с файлами 37
2.6. Кодирование текстовых файлов 39
2.7. Приложение для решения квадратного уравнения 43
2.8. Программная реализация текстового редактора 44
2.9. Основы работы с табличными компонентами 49
2.10. Использование табличных компонентов для поиска в массиве 51
2.11. Использование табличных компонентов для обработки текстовых записей 56
2.12. Приложение с несколькими формами 61
2.13. Использование форм для ввода и редактирования информации 66
Самостоятельная работа 72
Контроль 76
Глоссарий 77
Литература 80

Введение

"...Разработка программного обеспечения – весьма трудоемкий и дорогостоящий процесс. Это многогранная деятельность, связанная не только с работой на ЭВМ "

Джон Фокс.

Современное общество ставит перед высшей школой задачу подготовки специалиста, который сможет гибко адаптироваться к меняющимся жизненным ситуациям, самостоятельно приобретать необходимые знания, успешно применять их на практике для решения проблем высокотехнологичного производства. Данное пособие отвечает требованиям подготовки самостоятельного, активного, компетентного специалиста, соответствующего современным потребностям единого информационного общества.

Среди современных информационных технологий программирование занимает особое место как в Computer Scince, так и в образовательной информатике. Обучение программированию — сложнейший процесс. В узком смысле программирование рассматривается как кодирование. В более широком смысле программирование — процесс создания программ, то есть разработка программного обеспечения. Для разработки программного обеспечения характерны научная и логическая сложность, абстрактность, наличие социальных факторов, использование самых передовых технологий обработки информации.

Пособие направлено на обучение современным информационным технологиям на примере создания Windows - приложений в среде Delphi. Delphi - сложная современная система программирования, поддерживающая парадигмы визуального и объектно-ориентированного программирования. Она содержит множество различных инструментов и средств, образующих в целом богатую и удобную интегрированную среду программирования, в которой можно разрабатывать программы самого различного назначения.

Существенной особенностью обучения программированию является практическая направленность, связанная с написанием большого количества небольших программ на лабораторных занятиях. Поэтому важную роль имеет разработанный комплекс заданий, главной задачей которого является обучение студентов в процессе их самостоятельной работы на компьютерах, получение навыков применения современных информационных технологий для решения профессиональных задач.

Выполнение развивающих заданий является абсолютно необходимым, поскольку именно оно прививает навыки самостоятельного написания, выполнения и отладки программ и способствует формированию умений у будущих специалистов. Задания имеют рекомендации по программной реализации, облегчающие самостоятельное написание программ.

Использование пособия существенно повышает скорость и эффективность обучения, содействует формированию мировоззрения и системного мышления студентов, получению глубоких фундаментальных знаний. С помощью практических работ достигается не только приобретение умений и навыков, но также приобретение новых знаний и выработка умения применять ранее полученные знания.

Данное пособие предназначено для студентов специальностей «Информационный менеджмент», "Прикладная информатика в экономике", "Программное обеспечение вычислительной техники и автоматизированных систем", а также студентов среднего профессионального образования по специальности "Программное обеспечение вычислительной техники и автоматизированных систем", начинающих освоение вопросов программирования.

Выписка из Госстандарта

Общая характеристика и классификация современных программных средств. Системные программные средства. Тестовые и диагностические программы. Антивирусные программы.

Системное программное обеспечение. Операционные системы. Однопользовательские однозадачные системы (MS-DOS). Однопользовательские многозадачные системы. Многопользовательские системы. Операционная система с открытым исходным кодом.

Прикладное программное обеспечение. Командно-файловые процессоры. Общая характеристика прикладных программных средств.

Языки программирования. Основные современные концепции. Среды визуального программирования. Объектно-ориентированное программирование. Гипертекстовый язык разметки (HTML) и XML.

Прикладные программные пакеты. Почтовые программы. Web-браузеры. Программы групповой работы.

Программное обеспечение корпоративной интеграции: корпоративное, промежуточное программное обеспечение и программное обеспечение корпоративной интеграции. Стандарты в области программного обеспечения.

1. Основы создания приложений в среде визуального программирования

1.1. Классификация языков программирования.

В современной информатике существуют два основных направления развития языков программирования: **процедурное** и **непроцедурное**.

Процедурное программирование возникло на заре вычислительной техники и получило широкое распространение. В процедурных языках программа явно описывает действия, которые необходимо выполнить, а результат задается только способом получения его при помощи некоторой процедуры, которая представляет собой определенную последовательность действий.

Среди процедурных языков выделяют в свою очередь **структурные** и **операционные** языки. В структурных языках одним оператором записываются целые алгоритмические структуры: ветвления, циклы и т.д. В операционных языках для этого используются несколько операций. Широко распространены следующие структурные языки: Паскаль, Си, Ада, ПЛ/1. Среди операционных известны Фортран, Бейсик, Фокал.

Непроцедурное (декларативное) программирование появилось в начале 70-х годов XX века, но стремительное его развитие началось в 80-е годы, когда был разработан японский проект создания ЭВМ пятого поколения, целью которого явилась подготовка почвы для создания интеллектуальных машин. К непроцедурному программированию относятся **функциональные** и **логические** языки.

В **функциональных** языках программа описывает вычисление некоторой функции. Обычно эта функция задается как композиция других, более простых, те в свою очередь разлагаются на еще более простые задачи и т.д. Один из основных элементов функциональных языков - рекурсия. Присваивания и циклов в классических функциональных языках нет.

В **логических** языках программа вообще не описывает действий. Она задает данные и соотношения между ними. После этого системе можно задавать вопросы. Машина перебирает известные и заданные в программе данные и находит ответ на вопрос. Порядок перебора не описывается в программе, а неявно задается самим языком. Классическим языком логического программирования считается Пролог. Построение логической программы вообще не требует алгоритмического мышления, программа описывает статические отношения объектов, а динамика находится в механизме перебора и скрыта от программиста.

Можно выделить еще один класс языков программирования - **объектно-ориентированные** языки высокого уровня. На таких языках не описывают подробной последовательности действий для решения задачи, хотя они содержат элементы процедурного программирования. Объектно-ориентированные языки, благодаря богатому пользовательскому интерфейсу, предлагают человеку решить задачу в удобной для него форме. Объектные языки программирования ориентированы на манипуляции с объектами. К ним относятся, например, C++, Java, C#, современный вариант языка Delphi. Программирование в объектной модели происходит не столько кодированием мелких шагов, сколько организацией взаимодействия объектов.

Языки описания сценариев, такие как Perl, Python, Rexx, Tcl и языки оболочек UNIX, предполагают стиль программирования, весьма отличный от характерного для языков системного уровня. Они предназначаются не для написания приложения с нуля, а для комбинирования компонентов, набор которых создается заранее при помощи других языков. Развитие и рост популярности Internet также способствовали распространению языков описания сценариев. Так, для написания сценариев широко употребляется язык Perl, а среди разработчиков Web-страниц популярен JavaScript.

1.2. История развития среды разработки Delphi

Развитие системы **Delphi** уходит своими корнями в язык программирования Pascal, созданный в 1971 году Никлаусом Виртом в качестве учебного языка компьютерного программирования. Язык Pascal сразу получил широкое распространение и стал использоваться как основа курса программирования во многих университетах. Благодаря легкости программирования и богатым функциональным возможностям, Pascal стал широко использоваться также в коммерческих разработках.

В свое время весьма популярной версией языка Pascal был Turbo Pascal, разработанный компанией Borland. С развитием операционной системы Windows и технологий объектно-ориентированного программирования компания Borland естественным образом расширила Turbo Pascal, создав версии Turbo Pascal for Windows и Object Pascal for Windows. Следующим шагом компании Borland было создание Delphi - визуальной среды разработки программ на Object Pascal под управлением операционной системы Windows. Главное назначение Delphi - быстрая разработка приложений Windows.

Работа над интегрированной графической средой программирования **Delphi** была начата корпорацией **Borland** в середине 1993 года. Новая среда коренным образом отличалась от всех предыдущих систем разработки. В ней была впервые воплощена полноценная концепция визуального создания программ.

Система **Delphi 1** появилась на рынке в 1995 году. Ее отличали мощные средства проектирования и программирования работы экранных форм. В Delphi 1 входили средства поддержки популярных СУБД, в том числе клиент-серверных. Сам язык на первых порах получил официальное название Object Pascal (объектный Паскаль). С тех пор версии Delphi выходят почти каждый год.

Версия **Delphi 2** (1996 год) обзавелась быстрым 32-разрядным компилятором. Она была настроена на программные интерфейсы появившейся тогда же операционной системы Windows 95. Delphi 2 также пополнилась улучшенными наборами компонентов для работы с базами данных и для построения визуальных табличных элементов, связанных с СУБД. Ее девизом стала фраза «легкость Visual Basic и мощь Си++».

В версии **Delphi 3** (1997 год) появились компоненты, позволившие использовать веб-технологии на корпоративном уровне. Возникла возможность отладки библиотек DLL, создания шаблонов и пакетов компонентов. Были добавлены компоненты для построения отчетов и анализа данных, средства оптимизации работы в редакторе.

Возможность создания распределенных приложений стала отличительной чертой версии **Delphi 4** (1998 год). В нее были включены средства работы с компонентной архитектурой CORBA, высокопроизводительные драйверы СУБД, технологии стыковки с модулями Java, интерфейсы доступа к настольным и серверным приложениям Microsoft и поддержка интерфейсов Windows 98. В языке появились динамические массивы и средства перегрузки методов.

Стабильная, производительная и многофункциональная версия **Delphi 5** (1999 год) стала продуктом, который завоевал массовое признание программистов во всем мире. В нее был добавлен большой набор компонентов для создания приложений для Интернета и поддержки технологии XML. Система Delphi 5 позволила настраивать и хранить несколько вариантов рабочих столов среды, в ней появились средства локализации. Доступ к СУБД из программ на Delphi мог организовываться по новой технологии Microsoft ADO. Название языка (объектный Паскаль) окончательно сменилось на сегодняшний вариант язык Delphi, совпадающий с названием среды.

Версия **Delphi 6** (2000 год) отличалась развитой поддержкой веб-сервисов. Был реализован набор «тонких» компонентов dbExpress для быстрого доступа к СУБД. Наряду с Delphi 6 корпорация Borland выпустила новый пакет **Kylix**, ориентированный на создание программ на языке Delphi для платформы Linux. В результате линейка продуктов Delphi стала кросс-платформной. Один и тот же исходный код приложения на языке Delphi мог компилироваться и в среде Delphi, и в среде Kylix для разных операционных систем.

Версия **Delphi 7** (2001 год) стала последней и наиболее развитой версией среды, реализованной для платформы Win32. В ней, в частности, появился компилятор для .NET, пока в консольном варианте. Добавилось множество компонентов и Мастеров, автоматизирующих создание сетевых приложений. Была продолжена поддержка кросс-платформной технологии создания программ на базе Delphi и Kylix.

Версия **Delphi 8** (2003 год) считается первой версией Delphi для платформы .NET. Переход был кардинальным: из Delphi 8 был полностью исключен компилятор для платформы Win32. В то же время допускалась создание приложений как с помощью стандартного набора компонентов .NET, так и с помощью новой версии библиотеки Borland VCL.NET.

Версия **Delphi 2005** (2004 год) вновь включила средства разработки для Win32 — в дополнение к поддержке .NET. В нее также вошел компилятор языка C# — стандартного языка разработки для платформы .NET.

Версия **Delphi 2006** позволяет писать приложения для .NET, используя стандартную библиотеку классов .NET, VCL для .NET. Среда также позволяет создавать .NET-приложения на C# и Win32-приложения на C++. Delphi 2006 содержит функции для написания обычных приложений с использованием библиотек VCL и CLX. Delphi 2006 поддерживает технологию MDA с помощью ECO (Enterprise Core Objects) версии 3.0.

В марте 2006 года компания **Borland** приняла решение о прекращении дальнейшего совершенствования интегрированных сред разработки JBuilder, Delphi и C++ Builder по причине убыточности этого направления. Планировалась продажа IDE-сектора компании. Однако в ноябре того же года было принято решение отказаться от продажи IDE бизнеса. Разработкой IDE продуктов теперь занимается новая компания — **CodeGear**, которая будет финансово полностью подконтрольна Borland.

В марте 2007 года CodeGear порадовала пользователей обновленной линейкой продуктов **Delphi 2007 for Win32** и выходом совершенно нового продукта **Delphi 2007 for PHP**.

Delphi 2007 for Win32 является названием, как новой среды разработки, так и новой версии языка. Продукт распространяется в двух редакциях - **Professional** и **Enterprise**. Enterprise версия помимо всего набора возможностей Professional версии, включает в себя расширенные возможности для создания Web-приложений и поддержку таких СУБД как Oracle 10g, MS SQL Server 2000/2005, Informix 9x, IBM DB2 8.x, Sybase 12.5.

25 августа 2008 года компания **Embarcadero**, новый хозяин CodeGear, опубликовала пресс-релиз на **Delphi for Win32 2009**. Версия принесла множество нововведений в язык: по умолчанию полная поддержка Unicode во всех частях языка, VCL и RTL; замена обращений ко всем функциям Windows API на юникодные аналоги (то есть MessageBox вызывает MessageBoxW, а не MessageBoxA); обобщённые типы, они же **generics**; анонимные методы.

Итак, все версии среды развиваются не только наращиванием функциональных возможностей, вводом новых наборов компонентов или включением новых информационных технологий. Практически все современные технологии уже вошли в Delphi в той или иной форме — ведь жизненный цикл технологии программирования составляет не менее трех лет, а версии Delphi выпускаются каждый год. Среда Delphi теперь активно расширяется внешними продуктами, охватывающими все большее число этапов создания программного продукта. Автоматизируется работа по анализу требований, модельному проектированию, организации дистанционного доступа к версиям кода, автоматическому тестированию и развертыванию. Таким образом, система Delphi становится законченным комплексом, предлагающим все необходимые инструменты для организации жизненного цикла программного обеспечения.

1. 3. Общая характеристика системы разработки приложений Delphi

Пакет Delphi – это средство быстрой разработки Windows-приложений с минимальным написанием программного кода. Пакет основан на использовании объектной технологии, позволяющей для всех основных механизмов Windows создать легкие в использовании рабочие объекты. Особенно эффективно данная технология используется для автоматизации процесса создания пользовательского оконного интерфейса.

Delphi - это комбинация нескольких важнейших технологий:

- Высокопроизводительный компилятор в машинный код.
- Объектно-ориентированная модель компонент.
- Визуальное (а, следовательно, и скоростное) построение приложений из программных прототипов.
- Масштабируемые средства для построения баз данных.

Визуальные средства Delphi построены на концепции двойного инструментария, позволяющей изменять свойства объектов как в процессе визуального конструирования на этапе "сборки", так и программно, в процессе работы приложения. В Delphi эта концепция реализуется с помощью компонентов, представляющих собой заготовки программного кода. Размещение компонентов на форме создаваемого приложения приводит к автоматической генерации необходимого программного кода.

Компоненты Delphi объединены в библиотеку VCL – Visual component Library (библиотека визуальных компонентов). Все компоненты можно классифицировать следующим образом:

Невидимые (невизуальные) — компоненты, с помощью которых осуществляется доступ к системным ресурсам. Они отображаются во время конструирования интерфейса, но не видны во время работы приложения.

Элементы управления дают возможность пользователю во время работы программ получать информацию и/или управлять работой программы.

Среда Delphi включает полный набор визуальных инструментов для быстрой разработки приложений (Rapid Application Development — RAD), поддерживающий разработку пользовательского интерфейса.

Все компоненты Delphi разбиты на группы, называемые страницами, а набор страниц образует один из важнейших элементов системы Delphi – палитру компонентов. Палитра компонентов позволяет выбрать нужные объекты для размещения их на форме. Практически все компоненты в том или ином виде имеют три важнейшие составляющие:

- набор свойств компонента, описывающих основные характеристики компонента
- набор специальных подпрограмм (так называемых методов), определяющих основные действия, выполняемые компонентом для реализации своей функциональности
- набор событий, на которые может реагировать компонент

В результате, общая схема создания приложения выглядит следующим образом:

- размещение на форме необходимых компонентов из палитры
- настройка некоторых свойств компонентов с помощью специального инструмента – так называемого Инспектора Объектов (Object Inspector), что позволяет установить эти свойства в начальные значения с возможностью последующего изменения программным путем
- создание программного кода как реакцию на необходимые события, связанные с данным компонентом (написание обработчиков собы-

тий); внутри этого кода изменяются значения свойств и вызываются необходимые методы

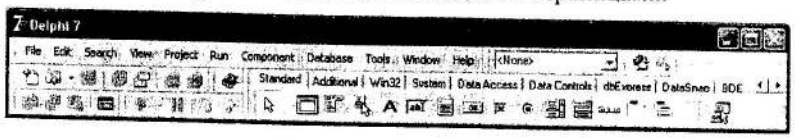
Язык программирования Delphi базируется на Borland Object Pascal.

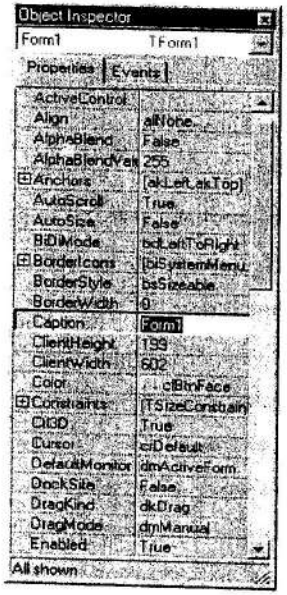
Delphi позволяет разработчикам настроить среду для максимального удобства. Вы можете легко изменить палитру компонент, инструментальную линейку, а также настраивать выделение синтаксиса цветом.

Запуск системы Delphi

После запуска системы Delphi появляются следующие основные составляющие, представляющие собой обычные окна:

1. В верхней части – набор основных управляющих элементов: строка меню, инструментальная панель с кнопками-заменителями пунктов меню, палитра компонентов с закладками-страницами.

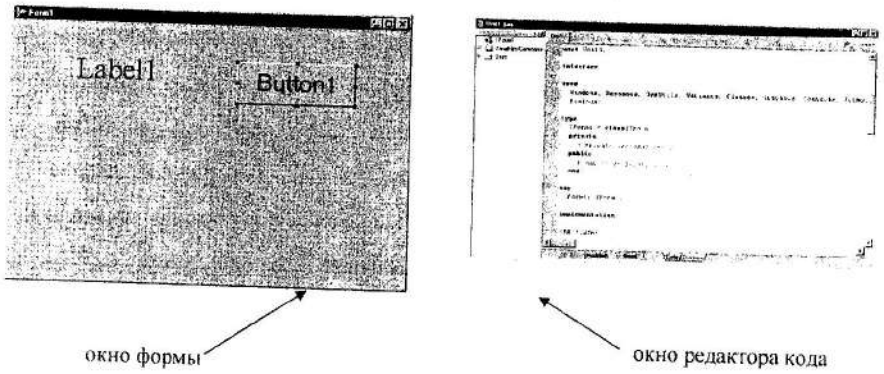




2. В левой части – Инспектор Объектов (ИО), в котором на закладках Properties и Events отображаются списки свойств и событий текущего активного компонента; для изменения свойства надо выбрать его мышью и ввести новое значение; некоторые свойства можно выбирать из списков; некоторые свойства (со знаком + слева) являются комбинированными (составными); если свойство является визуальным, то изменение его значения в ИО сразу приводит к изменению внешнего вида компонента на форме; отметим, что сама форма тоже является компонентом и ее свойства и события отображаются в ИО.

3. В центре экрана размещается форма-заготовка, на которой располагаются выбираемые из палитры компоненты; для удобства на форму нанесена точечная сетка, позволяющая размещать и выравнивать компоненты; положение и размеры формы можно менять обычным образом; выбор любого компонента на форме выполняется щелчком мыши, после чего вокруг компонента появляется рамка с размерными маркерами для изменения размера и перемещения компонента на форме.
4. Также в центре экрана находится окно Редактора Кода (РК), представляющего из себя обычный текстовый редактор; с его помощью вручную создается необходимый программный код, дополняющий тот код, который создается системой автоматически.

Окна формы и редактора кода чаще всего перекрывают друг друга.



1.4. Файлы, составляющие приложения Delphi

Создаваемое с помощью Delphi приложение принято называть проектом. Приложение Delphi хранится в нескольких различных файлах. Каждое имя файла заканчивается расширением, которое определяет содержимое файла. Настоятельно рекомендуется **для каждого проекта создавать отдельный каталог.**

Любое приложение рассматривается как набор программных моду-

лей (Unit) двух типов: интерфейсных и прикладных. Модули – это программные единицы, предназначенные для размещения фрагментов программ. Каждый интерфейсный модуль связан с соответствующей формой и имеет специальную структуру, скелет которой создается автоматически. Одна из форм объявляется главной. Обычно это форма, созданная раньше других, но при необходимости главной может быть назначена любая форма. Прикладные модули не имеют связанных форм и необходимы для реализации прикладного программного кода (чаще всего – различные вычисления).

С помощью содержащегося в модулях программного кода реализуется вся поведенческая сторона программы. Исходный код каждого модуля сохраняется в файлах ***.PAS**. После компиляции модуля или (чаще всего) проекта в целом, создаются файлы с объектным кодом ***.DCU** (Delphi Compiled Unit). По умолчанию все модули получают имена Unit1, Unit2, Unit3 и т.д., но рекомендуется заменять их более осмысленными именами. С каждым интерфейсным модулем-формой связывается еще один специальный файл – файл описания формы (расширение ***.DFM**), в котором сохраняется информация об использованных в этой форме компонентах.

Существует еще один специальный файл – файл проекта, который изначально называется **Project1.DPR**, в котором сохраняется информация о всех модулях проекта. Его стандартное имя также рекомендуется заменять более осмысленным. Каждое приложение имеет единственный файл проекта. Проект Delphi является Pascal-программой. Проект имеет, по крайней мере, два назначения — объявить модули приложения и запустить приложение. Delphi автоматически создает исходный код файла проекта и управляет им.

Кроме перечисленных, с проектом связывается еще несколько второстепенных файлов:

- файл **Project1.RES** — содержит иконку для проекта, создается автоматически;
- файл **Project1.DOF** — файл для сохранения установок, связанных с данным проектом, таких как параметры компилятора, командной строки, директории и другие.
- файл **Project1.DSK** содержит информацию о состоянии рабочего пространства;
- файл **Project1.EXE** — исполняемый файл.

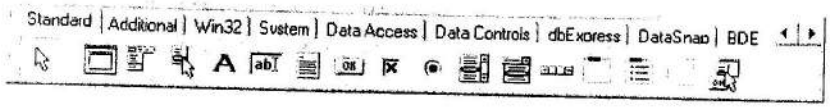
1.5. Обзор основных компонентов

Для начального освоения технологии визуального создания приложений наиболее важными являются три набора компонентов: стандартные (страница **Standard**), дополнительные (страница **Additional**) и диалоговые (страница **Dialogs**). По умолчанию, страница **Dialogs** находится в палитре компонентов довольно далеко в конце, поэтому удобно переместить ее закладку ближе к началу. Для этого можно войти в меню **Component / Configure Palette**, найти в списке **Pages** строку **Dialogs** и с помощью кнопки **Move Up** переместить ее в начало списка страниц.

Важность перечисленных наборов компонент определяется тем, что в большинстве случаев именно они используются для реализации пользовательского интерфейса. Эти компоненты являются объектными оболочками стандартных управляющих элементов системы Windows.

Стандартные компоненты

Рассмотрим состав первой страницы Палитры Компонент — **Stan-**



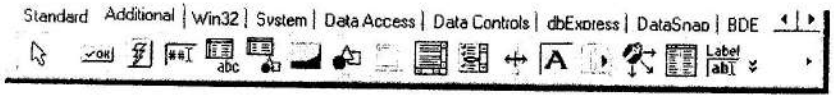
- **MainMenu** позволяет поместить главное меню в программу. При помещении TMainMenu на форму это выглядит, как просто иконка. Иконки данного типа называют "невидимыми компонентом", поскольку они не-

видимы во время выполнения программы. Создание меню включает три шага: (1) помещение TMainMenu на форму, (2) вызов Дизайнера Меню через свойство Items в Инспекторе Объектов, (3) определение пунктов меню в Дизайнере Меню.

- **PopupMenu** позволяет создавать всплывающие меню. Этот тип меню появляется по щелчку правой кнопки мыши.
- **Label** служит для отображения текста на экране.
- **Edit** стандартный управляющий элемент Windows для ввода. Он может быть использован для отображения короткого фрагмента текста и позволяет пользователю вводить текст во время выполнения программы.
- **Memo** иная форма Edit. Подразумевает работу с большими текстами. TMemo может переносить слова, сохранять в Clipboard фрагменты текста и восстанавливать их, и другие основные функции редактора.
- **Button** позволяет выполнить какие-либо действия при нажатии кнопки во время выполнения программы.
- **CheckBox** (переключатель) отображает строку текста с маленьким окошком рядом. В окошке можно поставить отметку, которая означает, что что-то выбрано.
- **RadioButton** (переключатель) позволяет выбрать только одну опцию из нескольких.
- **ListBox** нужен для показа прокручиваемого списка.
- **ComboBox** во многом напоминает ListBox, за исключением того, что позволяет водить информацию в маленьком поле ввода сверху ListBox. Есть несколько типов ComboBox, но наиболее популярен выпадающий вниз (drop-down combo box).
- **Scrollbar** - полоса прокрутки, появляется автоматически в объектах редактирования, ListBox'ах при необходимости прокрутки текста для просмотра.

- **GroupBox** используется для визуальных целей и для указания Windows, каков порядок перемещения по компонентам на форме (при нажатии клавиши TAB).
- **RadioGroup** используется для быстрой организации группы взаимоисключающих переключателей.
- **Panel** - управляющий элемент, используется в декоративных целях.

Компоненты страницы Additional.

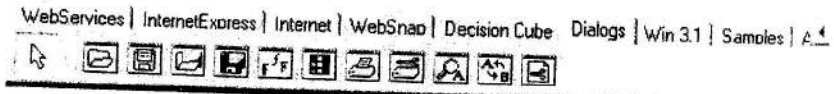


Рассмотрим некоторые компоненты страницы **Additional:**

- **BitBtn** – кнопка, на которой можно разместить картинку.
- **SpeedButton** - кнопка для создания панели быстрого доступа к командам.
- **MaskEdit** - аналог Edit, но с возможностью форматированного ввода. Формат определяется в свойстве EditMask. В редакторе свойств для EditMask есть заготовки некоторых форматов: даты, валюты и т.п.
- **StringGrid** - служит для представления текстовых данных в виде таблицы. Доступ к каждому элементу таблицы происходит через свойство Cell.
- **DrawGrid** - служит для представления данных любого типа в виде таблицы. Доступ к каждому элементу таблицы происходит через свойство CellRect.
- **Image** - отображает графическое изображение на форме.
- **Shape** - служит для отображения простейших графических объектов на форме: окружность, квадрат и т.п.
- **Bevel** - элемент для рельефного оформления интерфейса.

- **ScrollBox** - позволяет создать на форме прокручиваемую область с размерами большими, нежели экран. На этой области можно разместить свои объекты.

Компоненты страницы Dialogs



На странице **Dialogs** представлены компоненты для вызова стандартных диалогов Windows. Объекты, представленные на данной странице невидимы во время выполнения. Рассмотрим основные компоненты страницы **Dialogs:**

- **OpenDialog** — выбрать файл;
- **SaveDialog** — сохранить файл;
- **FontDialog**— настроить шрифт;
- **ColorDialog** — выбор цвета;
- **PrintDialog** — печать;
- **PrinterSetupDialog** — настройка принтера;
- **FindDialog** — поиск строки;
- **ReplaceDialog** — поиск с заменой;

Для получения справки по основным компонентам необходимо сделать следующее:

- запустить Delphi, в меню выбрать пункт Help, в нем -- команду Delphi Help
- в окне справки выбрать закладку **Указатель** и окне редактора ввести слова **Component palette**
- в списке страниц выбрать нужную и открыть ее описание

2. ПРАКТИЧЕСКИЕ ЗАДАНИЯ

2.1. Создание простейшей программы

Постановка задачи: Создать простейшую оконную программу, содержащую небольшой текст и кнопку закрытия окна.

Порядок работы.

1. Запустить систему **Delphi** с автоматическим созданием нового проекта, содержащего только одну форму
2. В Инспекторе Объектов изначально показаны только свойства пустой формы, которые надо изучить и установить следующие значения:
   - **Caption** (текст заголовка формы) – в название проекта (можно по-русски, например – Первый проект),
   - **Name** (имя объекта-формы) – в осмысленное значение (например – **FSimple**, причем можно использовать только латинские буквы и цифры, т.к. это внутреннее имя-идентификатор объектной переменной!)
3. Изменить размеры или положение формы как обычного окна с отслеживанием мгновенного изменения значений соответствующих свойств в ИО (свойства **Top, Left, Width, Height**); этот механизм работает и в обратную сторону

4. Разместить на форме компонент **Panel** (щелчок на компоненте в па-

литре компонентов, потом щелчок в любом месте формы)

5. С помощью Инспектора Объектов для компонента **Panel1** установить:
   - свойство **Caption** в пустую строку (заголовок не нужен)
   - свойство **Align** – в значение **alBottom** (выравнивание по нижней части формы)
   - свойство **Name** – в любое осмысленное значение (например – **PanBottom**, т.е. объект-Нижняя_Панель)
6. С помощью верхнего размерного маркера на контуре компонента **Panel1** немного увеличить высоту компонента (примерно ¼ высоты формы) для последующего размещения в этой области управляющих кнопок
7. Разместить на форме второй компонент **Panel**, убрать его заголовок и установить:
   - свойство **Align** в значение **alClient** (вторая панель автоматически захватит всю оставшуюся свободную область формы);
   - свойство **Name** – в имя объекта (например – **PanClient**
8. Разместить в любом месте верхней панели компонент **Label** для вывода небольшого текста и с помощью ИО установить следующие его свойства (изменение значения свойства сразу приводит к изменению внешнего вида компонента на форме):
   - **Caption** – в небольшой текст (например – "Привет Всем!", только без кавычек)
   - подсвойство **Size** свойства **Font** (размер шрифта текста) – в какое-нибудь большое значение (30-40 пунктов)
   - подсвойство **Name** свойства **Font** (имя шрифта) – в любое значение (например – Times New Roman)
   - подсвойство **Color** свойства **Font** (цвет шрифта) – в любое значение (например – красный)

9. Разместить в любом месте нижней панели компонент **Button** для реализации кнопки закрытия окна и с помощью ИО установить следующие свойства кнопки:
   - **Caption** – в текст ЗАКРЫТЬ ОКНО (если текст не помещается внутри кнопки – изменить ее размеры с помощью маркеров)
   - **Name** (имя объекта-кнопки) – в любое осмысленное значение (например – **ButtonClose**)
   - при желании – изменить размер и тип шрифта для текста кнопки, как описано выше
10. Создать обработчик события нажатия кнопки, выполняющий лишь закрытие окна приложения. Этот обработчик можно создать двумя способами:
    - выполнить двойной щелчок на компоненте-кнопке на создаваемой форме
    - в ИО перейти на закладку **Events** и дважды щелкнуть напротив одного из нужных событий (для кнопки это событие **OnClick**)
11. В обоих случаях происходит автоматический переход в окно Редактора Кода, в котором уже создан программный скелет модуля, более того – заготовка подпрограммы-обработчика необходимого события. Эта подпрограмма имеет имя, составленное из имени кнопки (например **ButtonClose**) и имени события без префикса **On**, например – **ButtonCloseClick**. В тело обработчика необходимо вручную внести код для закрытия окна, который в простейшем случае сводится к вызову метода **Close**:

```
procedure TForm1.ButtonCloseClick(Sender: TObject);
begin
   Close;     //только это надо ввести вручную!!!
end;
```

12. Сохранить готовое приложение по команде **File / Save All** в отдельном каталоге, причем при сохранении задать имя файла как для единственного модуля, так и для проекта в целом. После сохранения можно зайти в этот каталог и просмотреть, какие файлы были в него записаны.

13. Проверить работу созданного приложения, выбрав команду **Run / Run**, соответствующую кнопку инструментальной панели или клавишу **F9**. Запущенное приложение является полноценным окном, которое можно перемещать, изменять его размеры и закрывать разными способами!

2.2. Развитие простейшей программы

Постановка задачи: Расширить функциональность созданного простейшего приложения, постепенно добавляя новые возможности.

Порядок работы:

1. Открыть проект по команде **File / Open Project**.
2. Добавить возможность вывода диалогового окна при закрытии приложения. Для этого в обработчик события кнопки закрытия вместо простого вызова Close ввести следующий код:

if MessageDlg('Закрыть?', mtConfirmation, [mbYes, mbNo], 0) = mrYes then Close;

Здесь вызов **MessageDlg** имеет следующие параметры:

- текст сообщения
- тип сообщения (можно использовать константы **mtCustom, mtInformation, mtWarning, mtError, mtConfirmation**)
- комбинация кнопок в сообщении (можно использовать константы **mbOk, mbCancel, mbYes, mbNo, mbAbort, mbRetry, mbIgnore**)
- указатель на изображение, используемое в окне сообщения

(или ноль при отсутствии оного)

Вызов возвращает код выбранной в окне кнопки, который можно распознать с помощью одной из констант **mrOk, mrCancel, mrYes, mrNo, mrAbort, mrRetry, mrIgnore**.

1. Добавить кнопки для уменьшения/увеличения размера шрифта текста. Для этого:

- в нижней части верхней панели разместить еще одну панель, убрать ее заголовок и выровнять вниз; новую панель можно использовать для размещения кнопок, управляющих размером текста
- на новой панели разместить две кнопки, у которых в свойстве **Caption** ввести необходимые надписи (например – **Увеличить Шрифт**), а в свойстве **Name** – имена объектных переменных (например – **ButtonFontSizeUp**)
- создать 2 обработчика события выбора каждой из кнопок, содержащий код увеличения/уменьшения значения подсвойства **Size** свойства **Font** компонента **Label1**:

if Label1.Font.Size > 50 then ShowMessage('Больше увеличивать нельзя!')

else Label1.Font.Size := Label1.Font.Size + 2;

При вводе кода в окне Редактора Кода можно использовать оперативные подсказки: после набора имени компонента Label1 и ввода точки сразу появляется список возможных свойств и методов этого компонента, из которого можно выбрать необходимое свойство (например, Font) не вводя его с клавиатуры. Точно также, после ввода имени Font и последующей точки появляется список свойств этого объекта с возможностью выбора необходимого (например – свойства Size)

- сохранить проект (можно в том же каталоге) и проверить работу приложения

2. Добавить компонент **Edit** для ввода размера шрифта. Для этого:
   - на панели с кнопками разместим компонент-редактор **Edit**, очистим его свойство **Text** и изменим имя (свойство **Name**) на более осмысленное (например – **EditFontSize**)
   - уменьшим ширину редактора (свойство **Width**) до значения 40
   - слева от редактора разместить компонент-текст **Label2** с краткой поясняющей надписью (Caption - Размер шрифта)
   - справа от редактора разместить кнопку для изменения размера шрифта текста в соответствии с введенным в редакторе значением
   - создать обработчик нажатия кнопки изменения размера шрифта, который должен:
   - проверить текущее значение редактора (свойство Edit1.Text) на пустоту
   - если строка в редакторе пустая, вывести ShowMessage
   - если строка не пустая, то преобразовать текст в числовое значение с помощью стандартной функции StrToInt, проверить полученное число на попадание в допустимый диапазон (например – от 8 до 50) и либо вывести предупреждение, либо установить размер шрифта у компонента Label1 в соответствующее значение
   - сохранить проект и проверить его работу.
3. Добавить возможности изменения цвета и типа шрифта с помощью

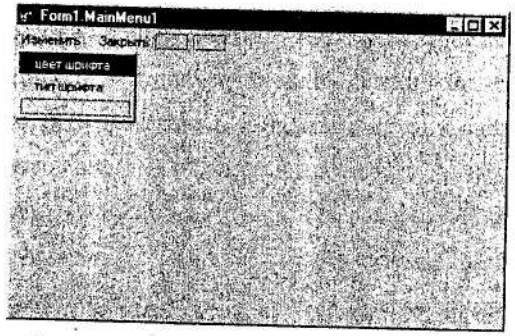
стандартных диалогов. Для этого:
   - разместить на форме (в любом месте) компонент **ColorDialog** со страницы **Dialogs**; в отличие от всех ранее использованных компонентов этот компонент при выполнении программы не виден и поэтому его расположение на форме не имеет значения
   - добавить на форму еще одну кнопку для изменения цвета шрифта и установим ее необходимые свойства (**Caption** и **Name**)
   - в обработчике этой кнопки ввести следующий код, запускающий стандартный диалог выбора цвета и в случае его успешного закрытия – установку цвета текста в компоненте **Label1** в выбранный цвет:

     if ColorDialog1.Execute then Label1.Font.Color := ColorDialog1.Color;
   - аналогично, для управления всеми характеристиками шрифта разместить на форме (в любом месте) стандартный компонент **FontDialog**, добавить кнопку для изменения шрифта и создать ее обработчик следующего вида:

     if FontDialog1.Execute then Label1.Font := FontDialog1.Font ;
   - после сохранения проекта выполнить его и проверить работоспособность.
4. Добавить в приложение меню, дублирующее некоторые ранее введенные управляющие кнопки: **Изменить Цвет**, **Изменить Шрифт**, **Закрыть Окно**. Для этого:
   - разместить в любом месте формы компонент **MainMenu**
   - запустить визуальный генератор меню двойным щелчком на этом компоненте
   - ввести в свойстве **Caption** текст первого пункта основного меню

(например – **Изменить**)

- перейти к соседнему пункту основного меню и ввести в его свойство **Caption** текст Закрыть
- вернуться к предыдущему пункту и оформить последовательно пункты его раскрывающегося меню, введя тексты **Цвет Шрифта** и **Тип Шрифта**



- двойным щелчком на пункте **Цвет Шрифта** создать обработчик данного пункта, код которого может просто содержать вызов подпрограммы-обработчика соответствующей кнопки
- аналогично создать обработчик для второго пункта **Тип Шрифта**
- двойным щелчком на пункте **Закрыть** создать обработчик, содержащий вызов подпрограммы, созданной для кнопки **Закрыть**:

  ButtonCloseClick(Sender);
- такой подход позволяет выполнять абсолютно одинаковые действия как с помощью кнопок, так и с помощью команд меню
- дополнительно для каждого пункта меню можно ввести горячие клавиши, выбрав в списке для свойства **ShortCut** необходимую клавишу или их комбинацию
- сохранить и выполнить проект.

2.3. Реализация программы-калькулятора

Постановка задачи: Создать проект, выполняющий арифметические действия над целыми числами (калькулятор).

Порядок работы:

1. Создать новый проект.
2. Задать свойство **Caption** формы – Калькулятор, **Name** — FormCalculator.
3. Разместить компонент **Edit** и установить следующие свойства: **Enabled** – False; **Name** — Number.
4. Разместить на форме компонент **Panel** для логического объединения компонентов в группу. Установить следующие свойства:

BevelInner —bvraised;

BevelOuter — bvraised;

BevelWidth — 1;

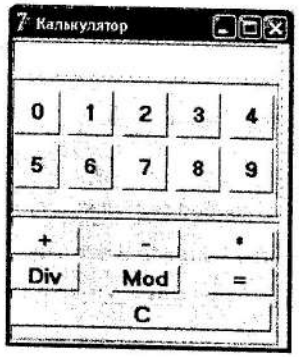
BorderStyle — bsSingle;

Caption — '';

5. Размесите на панели 10 компонентов **Button**. для этого, удерживая клавишу Shift, щелкните на компоненте Button в палитре компонент — компонент стал выделенным. Щелкните на панели 10 раз — на панели появилось 10 кнопок. Для всех кнопок установить **Width** — 40, **Height** — 40. Выровнять кнопки относительно друг друга.
6. Для каждой кнопки задать соответствующие **Caption** (0, 1, 2…) и **Name** (например, But0, But1 и т.д.)

7. Разместить в нижнюю часть формы еще один компонент **Panel** и выполнить для него настройки, как и для Panel1.
8. Разместить на нем 7 компонентов **Button** и изменить их свойства **Name** и **Caption**:

Caption	Name
+	ButAdd
-	ButSub
*	ButMult
Div	ButDiv
Mod	ButMod
=	ButEq
C	ButCancel



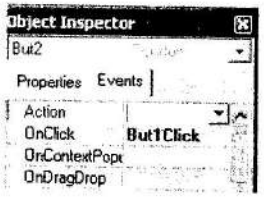
9. При нажатии на клавиши "0", "1", "2", ….., "9" должно формироваться число и к содержимому окна редактирования справа добавляться цифра, отображенная на кнопке. Для этого надо создать обработчик события **OnClick** кнопки But1:

```
procedure TFormCalculator.But1Click(Sender: TObject);
begin
number.Text := number.Text +But1.Caption ;
end;
```


10. При щелчке на все оставшиеся кнопки должно происходить то же самое действие. Поэтому обобщим написанный выше код, чтобы не прописывать его для всех кнопок:

```
procedure TFormCalculator.But1Click(Sender: TObject);
begin
number.Text := number.Text +(Sender as TButton).Caption ;
end;
```

11. Чтобы связать этот обработчик события с обработкой событий OnClick оставшихся кнопок, в Инспекторе объектов выбрать для них на закладке **Events** для события **OnClick — But1Click**. (настроить для всех кнопок)



12. Ввести переменную **operand1**: Integer для хранения значения первого операнда вычисляемого выражения. Для этого необходимо описать ее в модуле основной формы в разделе **Private**.
13. Для запоминания знака математической операции воспользуемся свойством **Tag** компонентов – кнопок. Для этого установить свойства Tag компонентов в соответствии с таблицей:

Name	tag
ButAdd	1
ButSub	2
ButMult	3
ButDiv	4
ButMod	5

14. Обработать нажатие на кнопку "+":

```
procedure TFormCalculator.ButAddClick(Sender: TObject);
begin
operand1:=StrToInt(number.text);
number.text:='';
Tag:=(Sender as TButton).Tag;
end;
```

15. Соединить данный обработчик события с обработчиками событий On-Click оставшихся четырех знаков действий, выбрав для них на закладке **Events** для события **OnClick — ButAddClick**.
16. Обработать нажатие на кнопку "=", при котором вычисляется результат арифметической операции.

```
procedure TFormCalculator.ButEqClick(Sender: TObject);
var operand2, Result: integer;
begin
Operand2:= StrToInt(number.text);
case Tag of
1: Result:= operand1 + Operand2 ;
2: Result:= operand1 - Operand2 ;
3: Result:= operand1 * Operand2 ;
4: Result:= operand1 div Operand2 ;
5: Result:= operand1 mod Operand2 ;
end;
number.Text :=inttostr(result);
end;
```

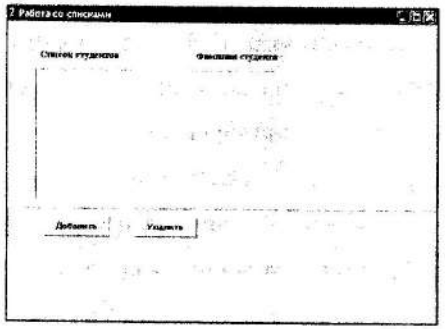
17. Обработать нажатие на кнопку "C" – должно очиститься окно редактора ввода: **Number.Text := '';**
18. Запустить приложение и проверить его работоспособность.

2.4. Реализация основных операций со списками

Постановка задачи: Создать приложение, выполняющее основные операции с текстовыми списками: добавление нового элемента, удаление выбранного элемента, очистка списка, сортировка по алфавиту. Задание выполняется по шагам с постепенным наращиванием возможностей.

Порядок работы:

1. Создать новый проект, установить для формы свойства **Caption** и **Name**, разместить на форме нижнюю панель, разместить верхнюю панель на всю оставшуюся клиентскую область и в ее нижней части добавить еще одну небольшую панель для размещения управляющих кнопок **Добавить** и **Удалить**
2. Разместить компонент **ListBox** на верхней панели; ширину окна для списка сделать достаточной для отображения фамилий студентов (100-120 пикселов); в простейшем случае никакие свойства списка можно не менять
3. Разместить над списком текстовую надпись **Label**. Задать **Caption** — Список студентов
4. Правее списка разместить компонент-редактор **Edit** для ввода фамилий очистить свойство **Text** у редактора и сделать ширину окна редактора достаточной для ввода фамилий
5. Разместить над редактором соответствующую текстовую надпись **Label** (Фамилия студента)



6. Создать обработчик кнопки **Добавить**, код которого должен проверить свойство **Text** редактора **Edit** на пустоту и либо вывести предупреждающее сообщение (**ShowMessage**), либо добавить текст из редактора в список:

   Listbox1.Items.Add (Edit1.Text);

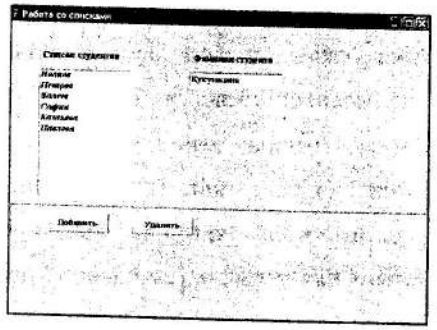
   Здесь используется основное свойство компонента-списка – свойство **Items**, представляющее собой внутреннее хранилище элементов списка и имеющее много полезных подсвойств и методов для удаления, добавления, сортировки элементов, работе с файлами. Например, метод **Add** добавляет в список новую строку либо в конец списка, либо (для упорядоченных списков) в нужное место по алфавиту

7. Создать обработчик кнопки **Удалить** для удаления из списка выделенного элемента:

   Listbox1.Items.Delete (Listbox1.ItemIndex);

   Здесь свойство **ItemIndex** хранит номер текущего выбранного в списке элемента, а метод **Delete** удаляет его из списка

8. Сохранить проект в своем каталоге и проверить правильность его



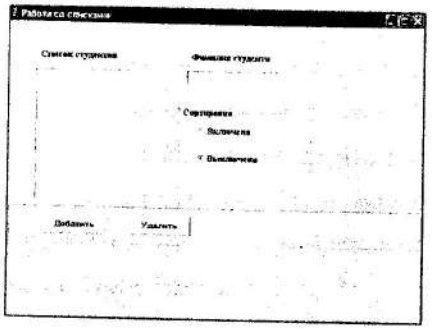
9. Для возможности работы с отсортированными списками добавить рядом со списком и редактором компонент **GroupBox**, в свойство **Caption** ввести название группы (например – **Сортировка**), внутри группы разместить два зависимых переключателя **RadioButton** с

надписями **Включена** и **Выключена**, один из них установить во включенное состояние (свойство **Checked = true**) и соответственно установить список в отсортированное или неотсортированное по умолчанию состояние (свойство списка **Sorted**)

10. Вместо отдельных компонентов **GroupBox** и **RadioButton** можно использовать их комбинацию – компонент **RadioGroup**

11. Двойным щелчком на каждой из радиокнопок создать их обработчики для перевода списка в соответствующее состояние:

    Listbox1.Sorted := true; // или **false**



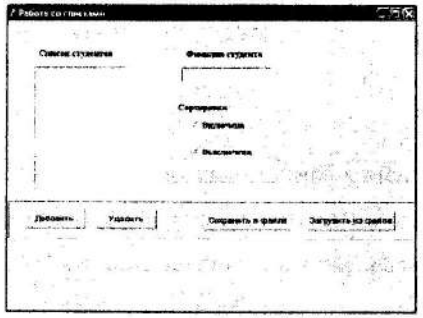
12. Сохранить проект и проверить его работу

2.5. Работа с файлами

Постановка задачи: Добавить в предыдущий проект возможность работы с файлами: сохранение в файле, загрузка из файла. Задание выполняется по шагам с постепенным наращиванием возможностей.

Порядок работы:

1. Открыть проект "Работа со списками".
2. Разместить две кнопки - **Сохранить в файле** и **Загрузить из файла**



3. Обработать сохранение

   ListBox1.Items.SaveToFile ('test.txt');

4. Обработать загрузку из файла

   ListBox1.Items. LoadFromFile ('test.txt');

5. Сохранить проект и проверить его новые возможности: ввести в список несколько фамилий, сохранить их в текстовом файле, проверить с помощью Проводника появление файла в текущем каталоге проекта, загрузить список из файла.
6. Добавить возможность выбора имени и расположения текстового файла со списком строк. Для этого надо воспользоваться стандартными диалогами открытия и сохранения файла (компоненты **OpenDialog** и **SaveDialog** страницы **Dialogs**). Эти компоненты не видны при выполнении приложения, поэтому их можно разместить в любом месте формы.
7. Для этих компонентов установить фильтры отбора файлов. Фильтр для компонента **OpenDialog** установим так, чтобы можно было выбирать показ либо только текстовых файлов, либо всех файлов. Для установки фильтров используется специальный Редактор Фильтров: выбрать на форме компонент **OpenDialog**, в Инспекторе Объектов выбрать свойство **Filter** и щелкнуть на кнопке с тремя точками. В левой колонке ввести текст описания фильтров (строку "Текстовые файлы" и строку "Все файлы"), а в правой колонке – соответствующие шаблоны **(*.txt и *.*)**

8. Аналогично, для компонента **SaveDialog** установить один фильтр только для текстовых файлов, причем свойство **DefaultExt** компонента **SaveDialog** надо установить в значение **txt**
9. Изменить обработчики кнопок сохранения в файле и загрузки из файла так, чтобы можно было работать с файлами, имена которых задаются с помощью соответствующих диалогов. Имена выбранных в диалогах файлов возвращаются в свойстве **FileName.** Код обработчиков может выглядеть так:

   if SaveDialog1.Execute

   then ListBox1.Items.SaveToFile(SaveDialog1.FileName);

10. Сохранить проект и проверить его работу. Теперь можно сохранять файл из списка в любом месте и под любым именем, а также – загружать в список строки из любых текстовых файлов.
11. Добавить в приложение возможность использования меню. Меню должно содержать 4 позиции: **Файл** (с командами **Сохранить** и **Загрузить**), **Список** (с командами **Добавить** и **Удалить**), **Сортировка** (с командами **Включить** и **Выключить**) и **Выход**. Предусмотреть использование горячих клавиш для быстрого выполнения команд.
12. Сохранить проект и проверить его работу

2.6. Кодирование текстовых файлов.

Постановка задачи: Создать приложение для кодирования и декодирования текстовых файлов с отображением исходного и преобразованного текстов.

Основные требования:

- управление программой через меню и функциональные клавиши
- возможность загрузки файла с исходным текстом, который может быть как в нормальном, так и закодированном виде
- кодирование нормального файла за счет уменьшения кода каждого

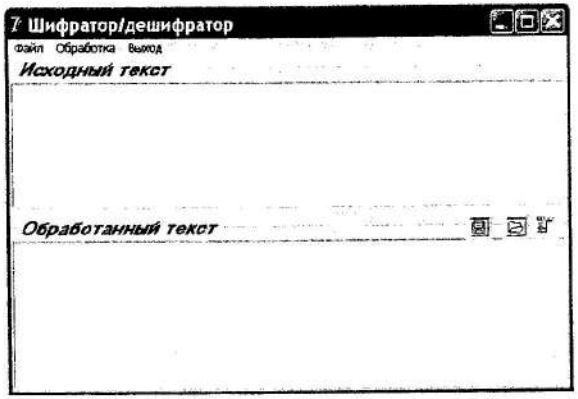
символа на 1

- сохранение закодированного текста в файле
- декодирование ранее созданного файла за счет увеличения кода каждого символа на 1

Порядок работы:

1. Создать новый пустой проект (меню **File / New / Application**) и сохранить его вместе с единственным модулем в отдельном каталоге (меню **File / Save All**)
2. С помощью компонента **MainMenu** создать меню с командами **Файл** (пункты **Открыть, Сохранить**), **Обработка** (пункты **Кодировать, Декодировать, Очистить**) и **Выход**. Для каждого пункта меню в Инспекторе Объектов установить осмысленные имена объектов через свойство **Name** (например, вместо стандартного имени **N1** для пункта **Открыть** лучше использовать имя **mOpen**). Задать "горячие клавиши" с помощью свойства **ShortCut**.
3. Разместить на форме 2 компонента **GroupBox**, выровнять один по нижней части окна, второй – по клиентской области, сделать их размеры примерно одинаковыми по середине окна; ввести их надписи
4. Внутри каждого **GroupBox** разместить по одному компоненту **Memo**, представляющего собой простейший текстовый редактор; выровнять эти компоненты на всю клиентскую область и установить для каждого режим вертикальной прокрутки (свойство **ScrollBars**)
5. Для каждого компонента **Memo** активизировать свойство **Lines** и щелчком на кнопке с тремя точками открыть встроенный текстовый редактор **String List Editor**, с помощью которого удалить текст с именем компонента (счетчик строк должен показывать 0)
6. С помощью компонентов **OpenDialog** и **SaveDialog** организовать работу с текстовыми файлами так, как описано в предыдущих заданиях. Обработчик команды **Открыть** должен с помощью метода

LoadFromFile установить свойство **Lines** верхнего компонента **Memo** в набор строк открываемого файла, а кроме того – очистить поле нижнего компонента **Memo** (метод **Clear**). Обработчик команды **Сохранить** должен с помощью метода **SaveToFile** записать строки из свойства **Lines** нижнего компонента **Memo** в заданный в диалоге файл. Оба этих действия аналогичны работе со списком, только вместо свойства **Items** надо использовать свойство **Lines**.



7. Создать основной обработчик пункта меню **Кодировать**, который должен:
   - очистить поле нижнего компонента **Memo**
   - организовать цикл обработки строк текста в верхнем компоненте **Memo** от 0 до значения

Memo1.Lines.Count – 1

   - внутри цикла прочитать очередную строку из свойства-массива **Lines** компонента **Memo1** во вспомогательную строковую переменную, найти ее длину и создать внутренний цикл обработки отдельных символов этой строки (найти внутренний код очередного символа, уменьшить его на 1 и присвоить этот код тому же символу, что в целом реализует замену символа на со-

седний символ в кодировочной таблице)

- после завершения внутреннего цикла (но внутри внешнего цикла!) добавить измененную строку в набор строк нижнего компонента **Memo**: **Memo2.Lines.Add(строка)**

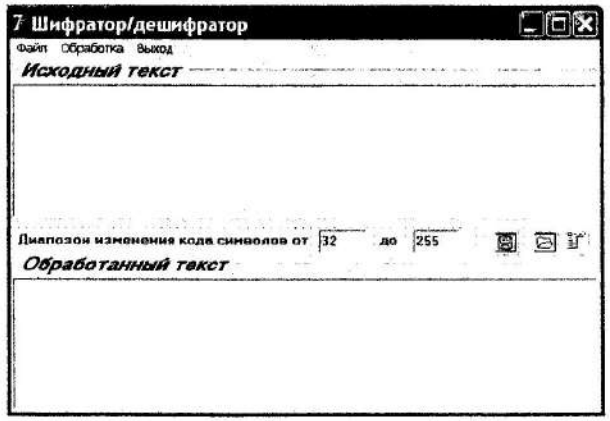
8. Создать обработчик команды меню **Декодировать**, который должен преобразовать закодированный текст в окне верхнего редактора в нормальное исходное состояние и отобразить результат в окне нижнего редактора. Код этого обработчика полностью совпадает с описанным выше, за исключением того, что внутренний код символа должен не уменьшаться, а увеличиваться на 1 для восстановления его начального значения.
9. Создать обработчик команды меню **Очистить** для удаления текста из рабочих окон обоих редакторов (метод **Clear**)
10. Сохранить проект и выполнить его. Проверить работоспособность для разных ситуаций (ввод исходного текста вручную в окне верхнего редактора, кодирование его и сохранение в файле, загрузка закодированного файла и возврат (декодирование) в исходное состояние).

Дополнительное задание

Постановка задачи: Реализовать в предыдущем проекте возможность задания шага изменения внутренних кодов символов текста.

Порядок работы:

1. Использовать случайное преобразование кодов символов: для каждого кода в диапазоне от 32 до 255 сгенерировать другое случайное число в том же диапазоне (без повторений!).



2. Сохранить соответствие в таблице и выполнять оба преобразования с помощью этой таблицы;
3. Для хранения таблицы между разными сеансами запуска программы использовать файл;
4. Загрузка таблицы из файла может выполняться в обработчике события формы OnCreate, а выгрузка – в обработчике события закрытия формы. Работа с файлами выполняется обычным образом с помощью подпрограмм AssignFile, Reset/Rewrite, Read/Write и CloseFile.

2.7. Приложение для решения квадратного уравнения

Постановка задачи: Приложение должно давать возможность в наглядном виде вводить коэффициенты квадратного уравнения и в зависимости от их значений выводить на форму либо соответствующее сообщение, либо найденные корни.

Порядок работы:

1. Коэффициенты вводить с помощью трех текстовых редакторов, причем пустую строку можно интерпретировать как ноль (выполняя соответствующую проверку и присвоение), а непустую строку преобразовывать в вещественное число с помощью стандартной функции **StrToFloat**.
2. Для показа на форме результата вычислений можно разместить на форме компонент **Label** с пустым текстом (свойство **Caption**), а после решения уравнения по кнопке **Решить** программно формировать значение свойства **Caption** в зависимости от результата решения. Преобразование значений корней в текстовую строку можно выполнить с помощью стандартной функции **FloatToStr**

2.8. Программная реализация текстового редактора

Постановка задачи: Создать текстовый редактор, обеспечивающий основные стандартные функции:

- сохранение текста в файле и загрузка из файла
- выбор шрифта и стиля отображения текста
- выбор фона текста

- выполнение операций с текстовым буфером (копирование, вырезание и вставка текстовых фрагментов)

Инициализация всех перечисленных действий должна выполняться как через элементы меню, так и через кнопки инструментальной панели. Поставленная задача решается в несколько этапов с постепенным наращиванием возможностей. Основой редактора является стандартный компонент **Memo**, который обеспечивает все основные действия по вводу и удалению текста.

Порядок работы:

1. Создать новый проект.
2. Разместить на форме компонент для реализации панели инструментов **ToolBar** со страницы **Win32,** который автоматически выравнивается по верхней части клиентской области формы. Компонент будет использоваться позже, а пока он нужен только для разметки рабочего пространства формы
3. Разместить на форме компонент **Memo**:
   a. установить его свойство **Align** в значение **alClient** для захвата редактором всей оставшейся клиентской области окна-формы,
   b. установить свойство **ScrollBars** в значение **vertical** для инициализации вертикального скроллера,
   c. активизировать свойство **Lines**, которое хранит строки текста и удалить из него имя объекта
4. Разместить на форме компоненты **OpenDialog** и **SaveDialog** со страницы **Dialogs**, установить фильтр для имен файлов с помощью **Filter Editor**, причем для компонента **OpenDialog** установить фильтры для показа текстовых и любых файлов, а для компонента **SaveDialog** установить фильтр только для текстовых файлов, дополнительно установив свойство **DefaultExt** в значение **txt**
5. Разместить на форме компонент **MainMenu** и двойным щелчком на

нем активизировать визуальный конструктор меню, с помощью которого создать меню, содержащее 2 элемента - **Файл** (с командами **Новый, Открыть, Сохранить**) и **Выход**

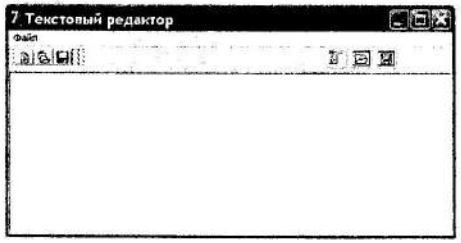
6. Закрыть конструктор меню и щелкнуть в форме поочередно на каждом пункте меню для автоматической генерации обработчиков событий

7. Код обработчика команды **Новый** минимально должен содержать лишь вызов метода **Clear** объектного свойства **Lines** компонента **Memo1** для удаления всего текста: **Memo1.Lines.Clear**
8. Код обработчика команды **Открыть** минимально должен содержать очистку свойства **Lines** и вызов стандартного диалога открытия файла:

```
if OpenDialog1.Execute then
  Memo1.Lines.LoadFromFile(OpenDialog1.FileName)
```

9. Аналогичный код имеет обработчик команды **Сохранить**, только вместо метода **LoadFromFile** надо вызвать метод **SaveToFile**
10. Сохранить проект, провести компиляцию и проверку его работоспособности.
11. Добавить в приложение наполнение панели инструментов, для чего на компоненте **ToolBar** разместить 3 компонента-кнопки **SpeedButton**.(новый, открыть, сохранить). Для каждой из них:

- установить свойство **ShowHint** в **true** для активизации всплывающих ярлыков-подсказок
- в свойство **Hint** ввести текст подсказки (например - **Открыть файл**)
- активизировать свойство **Glyph** для установки рисунка на кнопке с помощью специального редактора **Picture Editor** (стандартные рисунки в виде **bmp**-файлов обычно находятся в каталоге C:\Program Files\Common Files\Borland Shared\Images\Buttons)
- создать обработчик, который должен выполнять те же самые действия, что и обработчик соответствующей команды меню; для этого код обработчика события от кнопки может вызвать соответствующую процедуру обработки события от меню

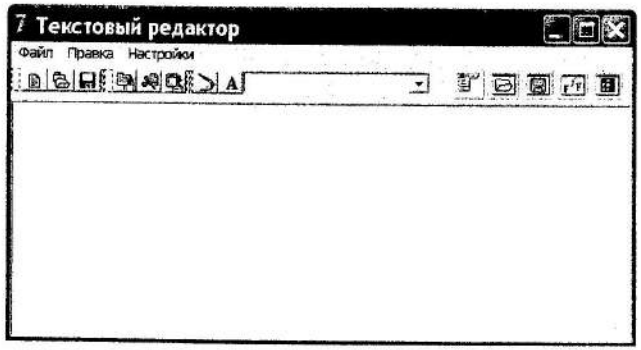


12. Сохранить проект и проверить работу кнопок инструментальной панели
13. Разместить на форме компоненты-диалоги **Font** и **Color**. Расширить меню новыми элементами **Правка** (с командами **Вырезать, Копировать** и **Вставить**) и **Настройка** (с командами **Шрифт** и **Цвет фона**). Добавить на панель инструментов соответствующие кнопки, объединив их в группы с помощью компонента-разделителя **Splitter** (страница **Additional** палитры компонентов).
14. Создать обработчики событий от кнопок и меню, используя следующие рекомендации:
    - для работы с буфером компонент **Memo** имеет 3 метода:

CopyToClipboard, **CutToClipboard** и **PasteFromClipboard**.

- для установки шрифта и цвета фона надо использовать свойства **Font** и **Color** компонента **Memo**, устанавливая их в те значения, которые возвращаются соответствующими диалогами.

15. Добавить на панель инструментов новый управляющий элемент - комбинированный список для выбора шрифта **без использования** стандартного диалога. Для этого разместить на панели компонент **ComboBox** (отделить от соседней кнопки разделителем!), установить его свойство **Style** в **DropDownList**, включить показ ярлыка-подсказки



16. Создать обработчик события формы **OnCreate**, код которого должен в момент создания формы загружать элементы списка (свойство **Items**) именами установленных в системе шрифтов; для этого можно использовать специальный **невизуальный** компонент **Screen**, у которого свойство **Fonts** (тоже типа **TStrings**) устанавливается автоматически
17. Создать обработчик события **OnClick** для комбинированного списка; этот код должен устанавливать подсвойство **Name** свойства **Font** компонента **Memo** в значение, выбранное пользователем в комбинированном списке; индекс выбранного в списке элемента возвращается в свойстве **ItemIndex**

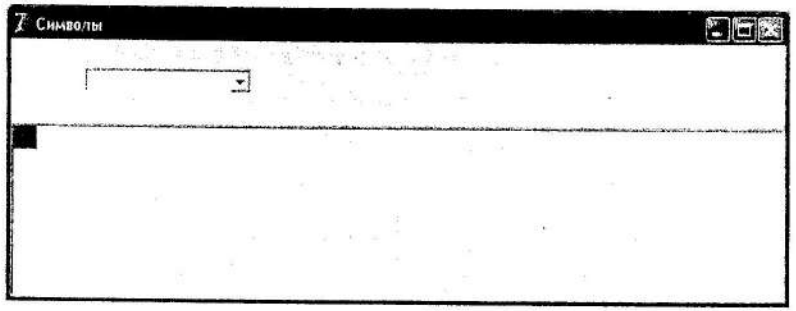
18. Сохранить проект и проверить его работу

2.9. Основы работы с табличными компонентами

Постановка задачи: Создать приложение, позволяющее просматривать символы системных шрифтов.

Порядок работы:

1. Создать новый проект.
2. Настроить для формы свойства: Name – Fonts; Caption — Символы.
3. Разместить на форме компонент Panel со следующими свойствами: Align — alTop.
4. Разместить на панели компонент ComboBox (Name — FontCB).
5. Расположить на форме компонент DrawGrid со следующими свойствами: Name — FontDG; Align — alClient; RowCount — 7; ColCownt — 32; FixedCols —0; FixedRows —0; DefaultColWidth —20; DefaultRowHeight —20.
6. Изменить размер формы так, чтобы сетка не имела полос прокрутки, а вокруг ячеек не было пустого пространства.



7. Создать обработчик события OnDrawCell для компонента FontDG, чтобы содержимое каждой ячейки перерисовывалось. Для изображения символов шрифта необходимо воспользоваться свойством Canvas компонента, а именно методом TextRect для вывода текстовой

информации в определенной ячейке.

```
procedure TFonts.FontDGDrawCell(Sender: TObject; ACol, ARow: Integer;
 Rect: TRect; State: TGridDrawState);
begin
with fontDG.Canvas do
 TextRect (Rect, Rect.Left , Rect.Top ,Char ((ARow+1)*32+ACol));
end;
```

Параметры ячейки для вывода символов шрифта берутся из параметров обработчика события, а символ шрифта для отображения в ячейке определяется в зависимости от строки и столбца.

8. Сохранить и проверить проект.
9. Обработать событие OnCreate формы, чтобы все шрифты при запуске приложения отображались в компоненте ComboBox.

Названия всех экранных шрифтов можно узнать с помощью глобальной переменной Screen типа TScreen, которая автоматически добавляется во все приложения Delphi. Информация о доступных приложению шрифтах содержится в свойстве Font.

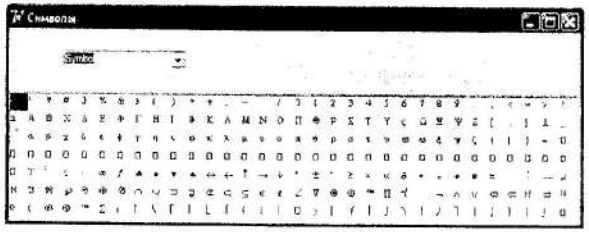
```
procedure TFonts.FormCreate(Sender: TObject);
begin
with FontCB do
 begin
  Items:=Screen.Fonts ;
  ItemIndex:=Items.IndexOf(Font.Name);
  end;
end;
```

10. Создать обработчик события, связывающий значения имени шрифта у FontDG и FontCB.

```
procedure TFonts.FontCBClick(Sender: TObject);
begin
```



```
FontDG.Font.Name := FontCB.Text ;
end;
```



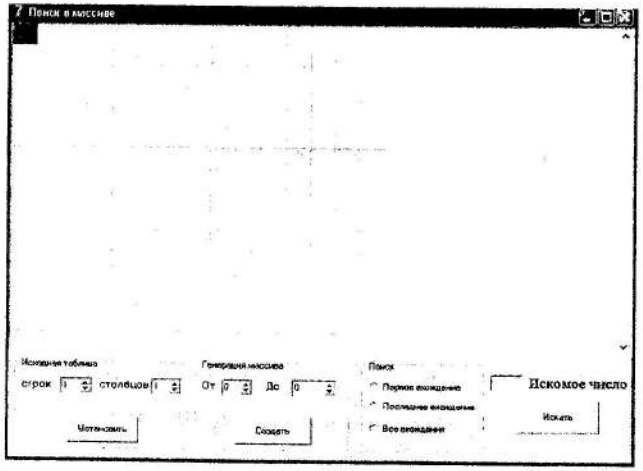
11. Сохранить и проверить проект, выбрав различные шрифты.

2.10. Использование табличных компонентов для поиска в массиве

Постановка задачи: Реализовать поиск в случайном неупорядоченном массиве целых чисел. Основные возможности приложения:

- задание числа строк и столбцов в неупорядоченном массиве
- задание границ диапазона генерируемых чисел
- задание искомого числа
- установка режима поиска (первое появление искомого числа, последнее появление искомого числа, все вхождения искомого числа)
- наглядное отображение в таблице найденных чисел

Для табличного представления данных использовать компонент **StringGrid** (страница **Additional**).



Порядок работы:

12. Создать новый проект.
13. Установить свойства **Caption** и **Name** формы, а также – свойство **WindowState** в значение **wsMaximized** для автоматического распахивания окна во весь экран при запуске приложения.
14. Разместить на форме управляющую панель с выравниванием в нижней части формы, а затем разместить в верхней части формы компонент **StringGrid** с выравниванием по клиентской области.
15. Для этого компонента установить свойства **FixedCols** и **FixedRows** в 0 (нет фиксированных строк и столбцов для размещения заголовков), свойства **ColCount** и **RowCount** в любые подходящие значения (например – 20 и 10) для задания начальной размерности таблицы, свойства **DefaultColWidth** и **DefaultRowHeight** в значения близкие к 30 и 24 (это подходящие значения для отображения небольших числовых данных).
16. Разместить на нижней (управляющей) панели компонент **GroupBox** для задания параметров таблицы при выполнении программы; в этом компоненте разместить кнопку с текстом **Установить**, два компонента **SpinEdit** со страницы **Samples** для удобного ввода числа строк и столбцов и две надписи.

17. Для обоих компонентов **SpinEdit** установить следующие свойства: **MinValue** =1, **MaxValue** = 50 (число строк и столбцов таблицы будет меняться от 1 до 50), **Increment** = 1 (шаг изменения значения при одном щелчке).
18. С помощью ИО переключиться на форму, выбрать закладку **Events** и создать обработчик события **OnCreate** (это событие возникает при первоначальном создании формы при запуске приложения), в котором ввести код для установки начальных значений компонентов **SpinEdit** в начальные значения числа строк и столбцов:

    SpinEdit1.Value := StringGrid1.RowCount ;

    SpinEdit2.Value := StringGrid1.ColCount ;

8. Создать обработчик кнопки **Установить** для формирования таблицы в соответствии с введенными значениями числа строк и столбцов; код этого обработчика обратен только что приведенному коду.
9. Разместить на нижней (управляющей) панели еще один компонент **GroupBox** для задания параметров генерации массива случайных чисел при выполнении программы; в этом компоненте разместить кнопку с текстом **Создать**, два компонента **SpinEdit** для удобного ввода границ генерируемого массива и две надписи.
10. Установить свойства компонента для задания нижней границы так, чтобы минимальное значение было –100, максимальное +100, начальное значение – 0, а шаг изменения = 10.
11. Установить свойства компонента для задания верхней границы так, чтобы минимальное значение было 0, максимальное 1000, начальное значение – 100, а шаг изменения = 10.
12. Создать обработчик кнопки **Создать** для генерации случайной таблицы заданной размерности в заданном диапазоне чисел; обработчик

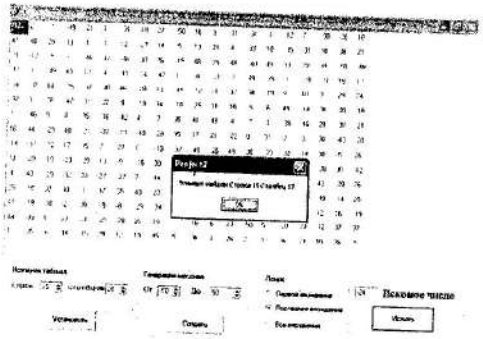
должен прежде всего сравнить нижнюю и верхнюю границы диапазона генерации и выдать сообщение при невозможности генерации; сама генерация выполняется двойным циклом по числу строк и столбцов (использовать соответствующие свойства компонента **StringGrid**) и состоит в заполнении внутреннего свойства-массива ячеек **Cells** случайными значениями из заданного диапазона (использовать свойства **Value** соответствующих компонентов **SpinEdit**); при оформлении цикла учесть следующие особенности:

- в массиве **Cells** сначала идет номер столбца, а потом – номер строки
- нумерация строк и столбцов начинается с НУЛЯ
- каждое случайное число надо преобразовать в строку с помощью функции **IntToStr**

13. Разместить на нижней (управляющей) панели компонент **RadioGroup** для задания режима поиска при выполнении программы; в этом компоненте разместить три радиокнопки с необходимыми надписями; установить во включенное состояние кнопку для режима "Искать первое вхождение заданного числа".
14. В правой части управляющей панели разместить окно-редактор для ввода искомого значения и соответствующую надпись, а также кнопку **Искать** для запуска процесса поиска.
15. Создать код обработчика кнопки **Искать**, который должен:
    - проверить на пустоту свойство **Text** компонента **Edit** и либо вывести соответствующее сообщение (например – "Введите искомое число"), либо преобразовать этот текст в число с помощью функции **StrToInt**
    - проверить, какая радиокнопка режима поиска включена для вызова одного из трех вариантов алгоритма поиска
    - выполнить сам поиск по одному из трех вариантов

- вывести результат поиска (номер строки и номер столбца) в окно сообщений с помощью **ShowMessage**

16. При организации поиска можно следовать следующим рекомендациям:
    - поиск во всех трех случаях выполнять двойным циклом **For – To – Do** по строкам и столбцам (не забыть, что в массиве **Cells** сначала задается номер столбца, а потом – номер строки)
    - при обнаружении первого совпадения вывести сообщение и прекратить поиск с помощью стандартной функции **Exit**
    - для поиска последнего вхождения надо запоминать координаты всех найденных совпадений, установив перед поиском эти координаты в фиктивные значения (-1); после завершения двойного цикла либо вывести координаты последнего вхождения, либо сообщение об отсутствии искомого элемента
    - аналогично можно найти все вхождения, просто выводя координаты при обнаружении совпадения



17. Сохранить и выполнить проект, проверить его работоспособность.

2.11. Использование табличных компонентов для обработки текстовых записей

Постановка задачи: Разработать приложение, позволяющее в наглядном и удобном виде обрабатывать наборы текстовых записей, например – для книг библиотеки. Каждая запись содержит порядковый номер книги, фамилию автора, название книги, количество имеющихся экземпляров.

Основные возможности приложения:

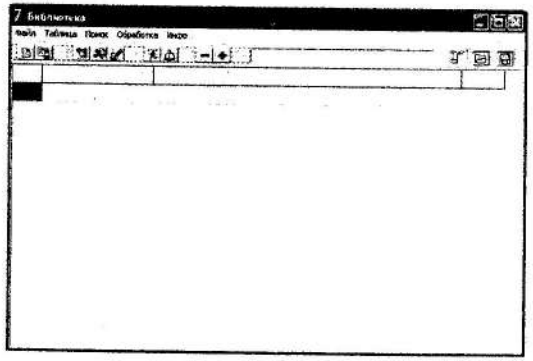
- отображение полей записей в клетках таблицы с возможностью их редактирования
- сохранение набора записей в текстовом файле и загрузка из файла
- добавление новых записей в таблицу
- удаление записей из таблицы
- очистка всех полей выбранной записи
- поиск записи по фамилии или названию
- выдача книги с проверкой ее наличия
- возврат книги

Работа с приложением должна выполняться с помощью команд меню, функциональных клавиш и кнопок инструментальной панели.

Порядок работы.

1. Создать новый проект и сохранить его в своем каталоге.
2. Создать основное меню с командами **Файл (Открыть, Сохранить), Таблица (Добавить строку, Удалить строку, Очистить строку), Поиск (По фамилии, По названию), Обработка (Выдать, Вернуть), Выход**.. Для каждой команды задать клавиши быстрого доступа.
3. Разместить на форме компонент для панели инструментов и наполнить ее необходимыми кнопками, добавить редактор для ввода фамилии или названия при поиске в таблице

4. Разместить на форме компонент **StringGrid** и задать для него:
   - выравнивание по всей клиентской области
   - число фиксированных столбцов = 0
   - число фиксированных строк = 1 (шапка таблицы)
   - число столбцов = 4
   - подобрать для столбцов подходящие размеры по ширине
   - с помощью свойства **Options** установить возможность редактирования текста непосредственно в клетках таблицы (**goEditing = true**), возможность изменения мышью ширины столбцов при выполнении программы (**goColSizing = true**), возможность перехода между клетками таблицы с помощью клавиши **Tab** (**goTabs = true**)
5. Разместить на форме стандартные диалоги открытия файла и сохранения в файле и настроить их стандартным образом на работу с текстовыми файлами.



6. Создать обработчик события формы **OnCreate**, в котором присвоить клеткам нулевой (фиксированной) строки таблицы текстовые значения – заголовки столбцов. Для этого используется свойство **Cells**, представляющее собой двухмерный массив клеток с нумерацией

строк и столбцов с нуля, причем первый индекс – номер столбца

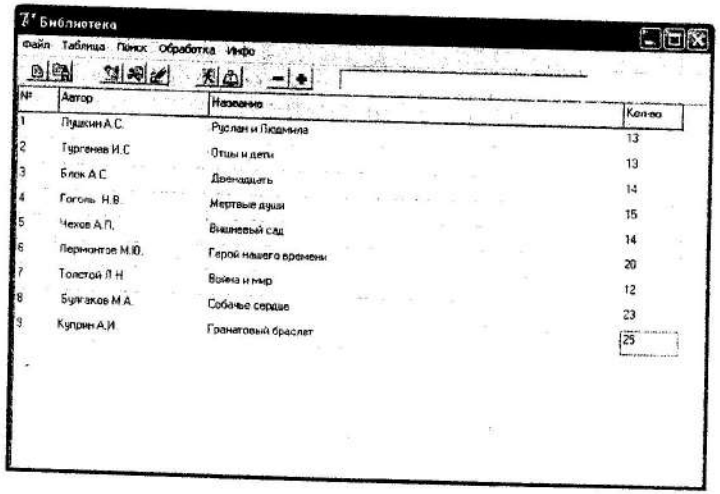
7. Создать обработчик для заполнения ячеек таблицы записями из текстового файла. В простейшем случае каждое поле таблицы можно сохранить как отдельную строку текстового файла. Для этого:
   - в разделе реализации модуля объявить файловую переменную с помощью встроенного типа **TextFile**, который используется вместо обычного типа **Text**
   - вызвать стандартный диалог открытия файла с помощью метода **Execute**
   - связать файловую переменную и выбранный в диалоге файл (свойство **FileName**) с помощью стандартной функции **AssignFile**, которая используется вместо обычной функции **Assign**
   - открыть файл для чтения с помощью функции **Reset**
   - обычным образом организовать цикл по условию достижения конца файла
   - внутри организовать вложенный цикл для чтения с помощью **ReadLn** очередных четырех строк текстового файла с присвоением прочитанных значений соответствующим клеткам таблицы, что в итоге сформирует одну строку таблицы
   - после завершения внешнего цикла установить свойство таблицы **RowCount** в число прочитанных строк
   - закрыть файл с помощью функции **CloseFile**
8. Создать обработчик для сохранения введенной в таблице информации в текстовом файле. Для этого:
   - вызвать стандартный диалог сохранения файла с помощью метода **Execute**
   - связать файловую переменную и выбранный в диалоге файл

(свойство **FileName**) с помощью стандартной функции **AssignFile**
   - открыть файл для записи с помощью функции **Rewrite**
   - вложенными циклами записать в текстовый файл строки из ячеек таблицы, используя для этого свойства **RowCount**, **ColCount** и свойство-массив **Cells** (нулевую строку с шапкой таблицы записывать в файл не надо)
   - закрыть файл
9. Создать обработчик для добавления в таблицу новой строки после текущей (выделенной, активной). Для этого:
   - увеличить на 1 число строк в таблице (свойство **RowCount**)
   - циклически сдвинуть все строки, расположенные ЗА текущей (номер текущей строки хранится в свойстве **Row**) на одну строку вниз, используя для этого свойство-массив строк **Rows** и цикл типа **For-Downto**
   - очистить строку **Row**+1 с помощью метода **Clear** свойства-массива **Rows**
10. Создать обработчик для удаления из таблицы текущей строки (ее номер хранится в свойстве **Row**). Для этого:
   - циклически все строки, начиная с **Row**+1 сдвигаются на 1 вверх
   - счетчик числа строк уменьшается на 1
11. Создать обработчик для очистки текущей строки
12. Создать обработчик для поиска в таблице записи с заданной фамилией. Для этого можно использовать метод **IndexOf** (строка) свойства-массива столбцов **Cols**[1], если фамилия задана во втором столбце. Метод возвращает номер первой записи, где найдена заданная строка или значение –1 при отсутствии этой строки. Основные шаги:
   - проверить значение в строке ввода и при ее пустоте вывести со-

общение

- сохранить во вспомогательной переменной значение, возвращенное функцией **IndexOf**
- если это значение равно -1, вывести предупреждающее сообщение
- иначе – присвоить это значение свойству **Row** таблицы для активизации соответствующей записи

13. Аналогично создать обработчик для поиска в таблице записи с заданным названием книги
14. Создать обработчик команды выдачи книги, который должен:
    - проверить текущее количество книг и при необходимости вывести сообщение о невозможности выдачи
    - уменьшить на 1 значение числа экземпляров выданной книги с изменением значения в соответствующей клетке таблицы (преобразовать текст в число, уменьшить на 1 и выполнить обратное преобразование)
15. Создать обработчик команды возврата книги, который должен лишь увеличить значение количества экземпляров в клетке таблицы.



16. Проверить работу приложения для достаточно большого числа книг (20-25 наименований)

2.12. Приложения с несколькими формами

Подавляющее большинство реальных приложений содержат несколько взаимодействующих форм. Одна из них считается главной, обычно это форма, созданная при разработке приложения первой. При необходимости главной можно назначить любую форму. Главная форма управляет работой всего приложения в целом, вызывая при необходимости подчиненные формы. Каждой форме соответствует свой модуль, сохраняемый в отдельном файле.

Постановка задачи: Разработать приложение, содержащее несколько форм:

- главную
- информационную форму **"О программе (About)"**
- форму для ввода пароля при запуске приложения

За основу можно взять любое ранее созданное приложение, например – таблицу текстовых строк.

Порядок работы.

1. Открыть базовый проект
2. Добавить в проект новую форму с информацией о программе и ее авторе:
    - выбрать в меню системы **Delphi** команду **File / New / Form**, после чего появляется новая форма и к проекту автоматически добавляется соответствующий модуль, что можно увидеть в окне **Редактора Кода** за счет появления второй закладки с именем модуля
    - установить основные свойства новой формы: **Caption, Name** (например – **FAbout**)

- установить требуемые размеры формы-заставки, установить свойство **BorderStyle = bsDialog** (окно не может изменять свои размеры), задать режим показа по центру экрана (свойство **Position = poScreenCenter**)
- оформить внешний вид формы с помощью панелей (в примере ниже использовано 7 панелей, некоторые из которых расположены друг на друге для красивого оформления окна), текста, графических изображений
- для текста (компонент **Label**) можно использовать свойства **WordWrap** (перенос длинного текста по словам на другие строки) и **Alignment** (для выравнивания текста по центру)
- для изображения (компонент **Image**) надо использовать свойства **Picture** (для загрузки необходимого изображения) и **Stretch** (для масштабирования изображения внутри области компонента)

- Создать обработчик кнопки закрытия формы стандартным образом (метод **Close**)
- в меню главной формы добавить пункт **"О программе"** и создать его обработчик для показа формы-заставки в модальном ре-

жиме:

FAbout.ShowModal;

- сохранить новый модуль в каталоге проекта командой **File / Save As** с изменением имени модуля со стандартного значения **Unit2** на более осмысленное, например - **UnitAbout**
- подключить новый модуль к основному (главному) модулю, для чего в разделе реализации главного модуля после директивы **implementation** записать строку

Uses UnitAbout;

- сохранить проект и проверить его работу

3. Добавить дополнительные возможности закрытия окна-заставки – по нажатию любой клавиши или по истечению заданного интервала времени. Для этого:
   - в **Инспекторе Объектов** перейти на форму-заставку, активизировав ее и выбрав в раскрывающемся списке ее имя
   - установить для формы-заставки свойство **KeyPreview** в **true** для того, чтобы форма перехватывала все сообщения от клавиатуры раньше расположенных на ней компонентов
   - перейти в **Инспекторе Объектов** на закладку **Events** и создать обработчик события формы **onKeyDown** с вызовом метода закрытия формы-заставки
   - разместить на форме-заставке компонент **Timer** со страницы **System**, установить его свойство **Interval** в необходимое значение (например – 10000 миллисекунд) и создать обработчик его единственного события **OnTimer** с вызовом метода **Close**
   - сохранить проект и проверить его работу
4. Добавить в проект еще одну форму – для ввода пароля. Ее особенности: она должна открываться только один раз при запуске приложе-

false (таблица НЕ редактируется), **Options / goRowSelect = true** (в таблице выделяется вся строка полностью)

3. Для главной формы установить режим показа при запуске во весь экран (в максимизированном состоянии)
4. Оформить вспомогательную форму: две панели, редакторы для ввода текстовых полей (автор, название), компоненты **SpinEdit** для ввода ограниченных числовых данных (количество экземпляров, год издания, стоимость), компонент **ComboBox** для выбора типа издания из заранее заданного списка, компонент **DateTimePicker** для ввода даты; каждый компонент ввода снабжается своей текстовой надписью. Две надписи на самом верху формы необходимы для показа номера текущей выбранной в таблице строки.
5. При создании подобных форм удобно использовать следующие приемы:
   - можно сгруппировать любое число компонентов вместе, выделяя их мышью при нажатой клавише **Shift**
   - сгруппированные компоненты можно целиком переместить в другое место формы, захватив один из них мышью (аналогично групповым операциям с файлами)
   - сгруппированные компоненты можно выровнять друг относительно друга, если щелкнуть на одном из них правой кнопкой мыши и в контекстном меню выбрать команду **Position / Align**, а далее – **Horizontal** или **Vertical** и один из режимов (в форме изменения данных все текстовые надписи у элементов ввода надо сгруппировать и выровнять по правому краю)
6. Для каждого компонента ввода подобрать подходящие размеры
7. Для компонентов **SpinEdit** задать подходящие величины для минимального и максимального значений
8. Для компонента **ComboBox** установить режим выбора из заранее

подготовленного списка, для чего:

   - свойство **Style** установить в значение **csDropDown**
   - щелкнуть на свойстве **Items** и в открывшемся текстовом редакторе ввести несколько (5-6) строк, определяющих тип издания
9. Настроить компонент ввода даты **DateTimePicker** следующим образом (обычно перечисленные ниже значения устанавливаются по умолчанию):
   - **DateFormat = dfShort** (короткий формат представления даты)
   - **DateMode = dmComboBox** (выпадающий календарь)
   - **Kind = dtkDate** (режим ввода дат, а не времени)
10. Установить порядок обхода элементов клавишей **Tab**: сначала обходятся все компоненты ввода, затем – обе кнопки, затем – опять компоненты ввода и т.д. Порядок обхода устанавливается для каждого компонента с помощью свойства **TabOrder** целого типа, начиная с 0. Для каждого компонента надо установить соответствующее значение по порядку. При этом надо учитывать, что для каждого контейнерного компонента (например, панель **TPanel**) порядок обхода СВОЙ, т.е. надо еще установить порядок обхода самих панелей. Если какой-либо компонент НЕ должен выбираться мышью, можно установить для него свойство **TabStop = false**.
11. Создать обработчик кнопки **Сохранить**, который должен лишь установить нормальный режим закрытия окна с последующим сохранением новых данных в таблице. Для этого можно использовать свойство **ModalResult**, установив его в значение **mrOK**
12. Создать обработчик кнопки **Отменить**, который должен свойство **ModalResult** установить в **mrCancel**
13. Подключить модуль **UChange** к главному модулю в разделе реализации
14. Создать обработчик команды **Изменить Строку**, который должен:

- присвоить всем компонентам формы изменения данных значения из соответствующих полей выбранной (текущей) строки таблицы, например – следующим образом:

FChange.LabelNomer.Caption := StringGrid1.Cells[0, StringGrid1.Row];

FChange.SpinEdit1.Value := StrToInt(StringGrid1.Cells[3, StringGrid1.Row]);

FChange.DateTimePicker1.Date := StrToDate(StringGrid1.Cells[5, StringGrid1.Row]);

(в последней строке используется функция преобразования строки в формат представления дат, принятый в Windows: дата и время внутри системы представляются как специальное вещественное число, и для перехода надо использовать стандартные функции **StrToDate, DateToStr, StrToTime, TimeToStr**)

- вызвать форму в модальном режиме
- проверить результат закрытия формы (свойство **ModalResult**), и если он равен **mrOK**, то присвоить полям текущей строки таблицы значения, введенные в соответствующих компонентах формы изменения данных (программный код аналогичен рассмотренному выше, только присваивания выполняются в противоположном порядке)
- если результат закрытия формы равен **mrCancel**, то вывести сообщение об отсутствии изменений в данной строке таблицы

15. Изменить обработчик команды **Удалить Строку** следующим образом:

- если таблица содержит только одну реальную строку (не считая заголовка), то удаление этой строки заменяется ее очисткой (метод **Clear**) без уменьшения числа строк
- в противном случае выполняется сдвиг строк на 1 вверх и соответствующее изменение их порядковых номеров (нулевой столбец таблицы)

16. Изменить обработчик команды **Добавить Строку** следующим образом:

если в таблице только одна пустая строка, то добавление новой строки должно выполняться именно на место этой пустой строки, для чего можно использовать подпрограмму

17. В связи с увеличением числа столбцов таблицы, внести необходимые изменения в ранее созданный программный код, в частности:

- в обработчик события создания главной формы OnCreate, где устанавливаются названия полей нулевой заголовочной строки таблицы
- в метод, выполняющий загрузку данных из текстового файла
- в метод, выполняющий сохранение данных из таблицы в текстовый файл

18. Сохранить и проверить проект.

САМОСТОЯТЕЛЬНАЯ РАБОТА

Самостоятельная работа является важным элементом освоения основной программы курса и нацелена на развитие индивидуальности студента, его творческих возможностей, создание педагогических условий образования и воспитания личности, формирование ведущих качеств компетентности специалиста технического профиля и гражданина, обладающего высоким уровнем информационной культуры. Специфичные для информационно - ориентированных специальностей цели направлены на обучение гибко адаптироваться в меняющихся информационных технологиях, самостоятельно приобретая необходимые знания, умело применяя их на практике для решения разнообразных производственных ситуаций, чтобы оставаться конкурентоспособным в процессе быстрой смены программного обеспечения и оборудования.

В соответствии со спецификой предмета, ведущие формы организации самостоятельной работы для информационно-ориентированных специальностей включают в себя:

- ***Самостоятельное изучение материала по заданным преподавателем темам;***

Данный вид самостоятельной работы предназначен для подготовки студентов к самообразованию и творческому труду, который предусматривает самостоятельную предварительную работу и обсуждение студентами вопросов, призванных обеспечить углубление, расширение и систематизацию знаний, выработку познавательных умений и формирование опыта творческой деятельности.

- ***Подготовку к практическим занятиям по темам.***

Самостоятельная подготовка к практическим занятиям выполняется студентом в свободное от занятий время и включает в себя:

- проработку лекционного материала по указанной теме;
- подготовку алгоритмов решения задач по теме;

- написание программ в соответствии с составленным алгоритмом;
- проверку работоспособности программ, как в домашних условиях, так и в компьютерных классах института;
- предоставление результатов самостоятельной работы преподавателю.

- ***Работу с учебно-методическими пособиями.***

Самостоятельная работа с учебно-методическими пособиями ставит обучающегося перед необходимостью выбора и принятия решения. В учебных пособиях создаются условия для самоконтроля и самокоррекции в процессе самостоятельного изучения программного материала. Они содержат вопросы к информационному тексту и выборочные ответы к ним. Именно эти ответы студент и должен проанализировать. Перед вопросом приводится информация, которая концентрирует внимание на определенной части ранее изученного материала и из которой последовательно вытекает заданный вопрос. При этом студент включается в активный познавательный процесс, сопровождающейся формированием эффективных приемов самостоятельной умственной деятельности.

Задания для самостоятельного выполнения

Задание 1.

Постановка задачи: Добавить в проект "Программная реализация текстового редактора" (задание 2.9) возможность работы с форматированным текстом.

Рекомендации: заменить компонент **Memo** на компонент **RichEdit**. Наиболее полезные его свойства – **SelAttributes** и **Paragraph**.

Первое свойство позволяет выполнить шрифтовое оформление текущего выделенного фрагмента, т.е. установить цвет (подсвойство **Color**), имя шрифта (**Name**), его размер (**Size**), стиль (**Style**). Значения этих подсвойств можно взять как значения одноименных подсвойств свойства **Font**

стандартного диалога **FontDialog**.

Второе свойство (**Paragraph**) управляет параметрами текущего абзаца и имеет следующие основные подсвойства:

- **Alignment** – выравнивание абзаца (значения **taLeftJustify**, **taCenter**, **taRightJustify**)
- **LeftIndent** – отступ в пикселах от левого края
- **RightIndent** – отступ в пикселах от правого края
- **FirstIndent** – отступ в пикселах для красной строки
- **Numbering** – оформление в виде списка с маркерами (**nsNone** – нет, **nsBullet** – есть)

Задание 2.

Постановка задачи: Для приложения, обрабатывающего наборы тестовых записей (задание 2.13), реализовать сортировку таблицы по фамилии, названию книги или числу экземпляров. Компонент **StringGrid** не содержит методов сортировки, поэтому ее необходимо реализовать вручную, воспользовавшись любым простейшим методом сортировки.

Задание 3.

Постановка задачи: Реализовать хранение списка паролей в файле с возможностью добавления новых паролей в список в главной программе.

Рекомендации: Для хранения паролей во время работы программы можно использовать массив, загружая его значениями при создании главной формы (событие формы **OnCreate**). Проверка пароля сводится к поиску в этом массиве. В меню добавить команды для включения нового пароля и удаления ненужного. Для этих операций можно создать небольшую форму с текстовым редактором и кнопками **Добавить** и **Удалить**. По кнопке **Добавить** проверяется уникальность введенного текста и добавление нового пароля в массив. По кнопке **Удалить** проверяется наличие в списке паро-

лей введенного текста и удаление этого пароля из списка. Для сохранения списка паролей в файле можно использовать событие **OnClose** главной формы, возникающее при закрытии главного окна и завершении приложения. Можно продумать механизм кодирования паролей в файле.

КОНТРОЛЬ

1. Какие файлы составляют проект Delphi?
2. Какие файлы проекта нельзя удалять ни в коем случае? Объясните.
3. Что такое форма?
4. Для чего предназначен файл формы?
5. Какая форма является главной?
6. Опишите этапы создания приложения в среде программирования Delphi.
7. Что такое компонент?
8. Для чего предназначены компоненты Edit и Label. В чем их принципиальное отличие?
9. Перечислите основные свойства компонента Edit.
10. Перечислите основные свойства компонента Label.
11. В чем отличие компонента Button от BitBtn?
12. Что такое событие?
13. В чем отличие создания свойства от события?
14. Для чего предназначен компонент StringGrid?
15. Перечислите основные свойства и события компонента StringGrid?
16. В чем отличие компонента StringGrid от компонента DrowGrid?
17. Какие компоненты называются невизуальными?
18. Для чего предназначены компоненты страницы Dialogs?
19. Что такое модуль?
20. Какие разделы содержит описание модуля?

ГЛОССАРИЙ

Алгоритм – четко определенная последовательность действий (шагов), приводящая к получению необходимого результата.

Глобальные переменные – переменные, объявленные в главной программе и доступные всюду внутри этой программы (кроме особых случаев совпадения имен глобальных и локальных переменных).

Заголовок программы – первая строка текста программы, содержащая служебное слово program и имя программы.

Заголовок процедуры – текстовая строка со служебным словом procedure, именем процедуры и, возможно, набором формальных параметров.

Заголовок функции – текстовая строка со служебным словом function, именем функции, типом возвращаемого результата и, возможно, набором формальных параметров.

Инструкция – формализованное описание некоторого элементарного, с точки зрения языка программирования, действия.

Компилятор (транслятор) – программа, выполняющая перевод текста программы на уровень машинных команд.

Компиляция (трансляция) – перевод текста программы на языке высокого уровня на уровень машинных команд.

Компонент — оформленный специальным образом класс.

Компоновка – объединение отдельных частей программы в готовую к выполнению машинную программу.

Константа (const) – объект программы, не изменяющий своего значения в процессе выполнения программы.

Локальные переменные – переменные, объявленные в подпрограмме и доступные для использования только внутри данной подпрограммы.

Модуль – структурная единица организации большой программы, которая может разрабатываться, компилироваться, отлаживаться и храниться отдельно от других модулей и главной программы.

Окно редактирования модуля — содержит текст модуля на языке Object Pascal, связанный с каждой формой приложения.

Окно Инспектора Объектов — отображает свойства (или события) одного или нескольких выбранных компонентов.

Ошибка синтаксическая – нарушение правил оформления исходного текста программы.

Ошибка семантическая – ошибка этапа выполнения корректной программы (попытка выполнить недопустимое действие).

Пакет Delphi –средство быстрой разработки Windows-приложений с минимальным написанием программного кода.

Переменная (var) – объект программы, который может изменять свое значение в процессе выполнения программы.

Подпрограмма – часть программы, выполняющая некоторое логически завершенное действие и оформленная как самостоятельная единица со своим уникальным именем.

Процедура (procedure) – разновидность подпрограммы.

Раздел выполняемых инструкций (тело) – часть программы или подпрограммы, содержащая формализованное описание набора действий, которые надо выполнить для решения поставленной задачи.

Раздел интерфейса модуля (interface) – открытый набор констант, типов данных, переменных и заголовков подпрограмм, объявленных в данном модуле и предназначенных для использования в других модулях или в главной программе.

Раздел реализации модуля (implementation) – закрытый для других модулей набор внутренних констант, типов данных, переменных и кодовых реализаций подпрограмм модуля.

Свойства объекта — влияют на то, как объект выглядит и на его поведение.

События — методы для организации взаимодействия между программой и пользователем.

Файл модуля (.pas) — содержит исходный код приложения на языке Object Pascal.

Файл проекта (.dpr)— автоматически создаваемая программа, написанная на языке Object Pascal.

Файл формы (.dfm) — используется для сохранения информации о внешнем виде формы.

Форма — визуальное изображения окна приложения.

Функция (function) – разновидность подпрограммы, возвращающая единственное значение, присваиваемое имени функции.

Язык программирования – набор формализованных правил оформления алгоритма решения задачи.

RAD — Rapid Application Development (быстрая разработка приложений)

VCL – Visual Component Library (библиотека визуальных компонентов)

ЛИТЕРАТУРА

1. Архангельский А.Я. Программирование в Delphi. Учебник по классическим версиям Delphi. — Бином, 2009.
2. Бабушкина И.А. Практикум по объектно-ориентированному программированию. — 2-е изд. — М.: БИНОМ. Лаборатория знаний, 2009.
3. Бобровский С. И. Технологии Delphi 2006. Новые возможности. — Питер, 2006.
4. Иванова Г.С. Основы программирования: Учебник для вузов. — 4-е изд. — М.: Изд-во МГТУ им. Баумана, 2007.
5. Культин Н.Б. Основы программирования в Delphi 7. Самоучитель. — BHV, 2007
6. Парижский С.М. Delphi. Только практика. — МК-Пресс, 2005.
7. Поган А. М. Программирование в Delphi. Просто как дважды два. — Эксмо-Пресс, 2007.
8. Попов В.Б. Паскаль и Дельфи. Учебный курс — СПб.: Питер, 2005.
9. Фаронов В.В. Delphi. Программирование на языке высокого уровня. Учебник для ВУЗов. — СПб.: Питер, 2007.
10. Хомоненко А.Д. Delphi 7 в подлиннике. — BHV, 2003.