МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
Рубцовский индустриальный институт (филиал)
федерального государственного бюджетного образовательного
учреждения высшего образования
«Алтайский государственный технический университет
им. И.И. Ползунова»
(РИИ АлтГТУ)

Е.А. Дудник

ПРОЕКТИРОВАНИЕ ПОЛЬЗОВАТЕЛЬСКИХ ИНТЕРФЕЙСОВ

(лабораторный практикум по программированию в среде Delphi)

Учебно-методическое пособие для студентов всех форм обучения

Рекомендовано Рубцовским индустриальным институтом (филиалом) ФГБОУ ВО «Алтайский государственный технический университет им. И.И. Ползунова» в качестве учебно-методического пособия для студентов всех форм обучения»

Рубцовск 2022

УДК 519.6

Дудник Е.А. Проектирование пользовательских интерфейсов: Учебно-методическое пособие для студентов всех форм обучения/ Рубцовский индустриальный институт. – Рубцовск, 2022. – 71 с.

В данном пособии представлен теоретический материал и лабораторный практикум по курсу программирование в среде Delphi. Предназначено для обучающихся изучающих дисциплину программирование в среде Delphi, на младших курсах.

Рассмотрено и одобрено
на заседании НМС РИИ.
Протокол № 5 от 2.06.22г.

Рецензент: начальник

Информационно-технического отдела РИИ Цыганков А.Н.

Оглавление

ВВЕДЕНИЕ

Данное пособие содержит изложение основных понятий объектно-ориентированного программирования, теоретические сведения по основам программирования на языке Delphi, методические указания и задания для выполнения лабораторных работ.

ЧАСТЬ I
ПРОЕКТИРОВАНИЕ ПОЛЬЗОВАТЕЛЬСКОГО ИНТЕРФЕЙСА

1.1. Понятие пользовательского интерфейса и требования к нему

Пользовательский интерфейс — это совокупность информационной модели проблемной области, средств и способов взаимодействия пользователя с информационной моделью, а также компонентов, обеспечивающих формирование информационной модели в процессе работы программной системы.

Разработчик приложения должен знать, что такое хороший интерфейс, и как его построить.

Основные принципы разработки пользовательского интерфейса:

- естественность интерфейса;
- согласованность интерфейса;
- дружественность интерфейса (принцип «прощения» пользователя;
- принцип «обратной связи»;
- простота интерфейса
- гибкость интерфейса
- эстетическая привлекательность

Основные правила эффективного пользовательского интерфейса

- Интерфейс пользователя необходимо проектировать и разрабатывать как отдельный компонент создаваемого приложения.
- Необходимо учитывать возможности и особенности аппаратно-программных средств, на базе которых реализуется интерфейс.
- Целесообразно учитывать особенности и традиции той предметной области, к которой относится создаваемое приложение

1.2. Стандартизация пользовательского интерфейса

В 1987 г., когда корпорация IBM объявила о намерении создать единую среду разработки приложений (Systems Application Architecture — SAA)

Целями проекта являются:

1. Повышение производительности труда программистов и конечных пользователей.

2. Облегчение эксплуатации и сопровождения программного обеспечения.
3. Повышение эффективности распределенной обработки информации.
4. Увеличение отдачи инвестиций в разработку информационных систем.

Проект SAA содержит 4 компонента:

• Соглашения по интерфейсу пользователя (Common User Access — CUA);

• Соглашения по программному интерфейсу (Common Programming Interface --CPI);

• Соглашения по разработке приложений (Common Applications — CA);

• Соглашения по коммуникациям (Common Communications Support — CCS).

По истечении почти двух десятков лет активного использования компьютерных технологий Министерство обороны США пришло к выводу, что основной причиной неудач крупных информационных проектов является отсутствие методология их реализации.

Первый стандарт был утвержден в 1985 году, а в 1994-1996 годах был разработан и принят новый, значительно более детальный и глубокий стандарт — MIL-STD-498.

Основные положения этого военного стандарта согласованы с международным стандартом ISO/IEC 12207:1995 («Процессы жизненного цикла программных средств»), но вместе с тем являются более конкретными и приближенными к практике.

В нашей стране попытка стандартизации процессов создания программных систем вылилась в создание комплекта документов под общим названием «Единая система программной документации», который увидел свет еще в 1977 году.

1.3. Проектирование пользовательского интерфейса

Под жизненным циклом программного продукта понимается последовательность процессов, действий и задач, которые осуществляются в ходе разработки, эксплуатации (использования) и сопровождения программного продукта в течение всей его жизни, от определения требований до завершения использования.

Проектирование

Начальный этап в разработке программного продукта является наиболее критичным, поскольку на этой фазе определяется общая концепция создаваемого продукта.

Эта часть процесса разработки включает не только определение цели и характеристик приложения, но и понимание того, кто является его потенциальными пользователями — их задачи, намерения, цели.

На этапе формирования требований к системе должны учитываться:

• область применения системы;

•требования пользователя (заказчика) к функциональным возможностям системы, к уровню ее безопасности и защищенности;

•эргономические требования и требования к уровню квалификации пользователей;

•степень документированности системы;

•организация сопровождения и т.д.

Прототипирование

После определения основных концепций проекта, разрабатывается прототип создаваемого приложения, отражающий некоторые основные аспекты его функционирования. В зависимости от уровня сложности приложения прототип может быть представлен либо в виде иллюстраций интерфейса, либо в виде специальных схем.

Прототип играет важную роль . Во-первых, он предоставляет хорошую возможность для обсуждения создаваемого приложения как внутри группы разработчиков, так и с потенциальными пользователями. Во-вторых, он может помочь определить характер потока заданий.

Форма представления прототипа зависит от цели разработки.

Действующие прототипы обычно наиболее полно позволяют оценить качество механизма взаимодействия пользователя с разрабатываемым приложением, т.е. качество интерфейса.

Испытание программного продукта и отладка

- Повторное выполнение этапов разработки
- Оценка потребительских свойств приложения в процессе разработки
- Техника проведения испытаний потребительских свойств приложения
- Альтернативный подход к проведению испытаний приложения

1.4. Этапы проектирования пользовательского интерфейса

При проектировании пользовательского интерфейса необходимо определить:

- структуру диалога;
- возможный сценарий развития диалога;
- содержание управляющих сообщений и данных, которыми могут обмениваться человек и приложение (семантику сообщений);
- визуальные атрибуты отображаемой информации (синтаксис сообщений).

Выбор структуры диалога

Диалог типа «вопрос - ответ»

Диалог на основе меню

список объектов, выбираемых прямым указанием, либо указанием номера (или мнемонического кода);

меню в виде блока данных;

меню в виде строки данных;

меню в виде пиктограмм.

Диалог на основе экранных форм

Как структура типа «вопрос — ответ», так и структура типа меню предполагают обработку на каждом шаге диалога единственного ответа.

Диалог на основе командного языка

Структура диалога на основе командного языка столь же распространена, что и структура типа меню. Она очень часто используется в операционных системах и располагается на другом конце спектра структур диалога по отношению к структуре типа меню.

Разработка сценария диалога

Целями разработки сценария диалога являются:

• выявление и устранение возможных тупиковых ситуаций в ходе развития диалога;

• выбор рациональных путей перехода из одного состояния диалога в другое (из текущего в требуемое);

• выявление неоднозначных ситуаций, требующих оказания дополнительной помощи пользователю.

Сценарий диалога позволяет описать процесс взаимодействия пользователя с приложением на уровне решаемой им прикладной задачи.

Однако для программной реализации интерфейса такое описание носит слишком общий характер. Поэтому на этапе реализации необходимо перейти на уровень описания соответствующих процессов с помощью используемых инструментальных средств разработки приложения

Визуальные атрибуты отображаемой информации

К визуальным атрибутам отображаемой информации относятся:

• взаимное расположение и размер отображаемых объектов;

• цветовая палитра;

• средства привлечения внимания пользователя.

Проектирование размещения данных на экране предполагает выполнение следующих действий:

1) Определение состава информации, которая должна появляться на экране;

2) Выбор формата представления этой информации;

3) Определение взаимного расположения данных (или объектов) на экране;

4) Выбор средств привлечения внимания пользователя;

5) Разработка макета размещения данных на экране;

6) Оценка эффективности размещения информации.

Общие принципы расположения информации на экране должны обеспечивать для пользователя:

возможность просмотра экрана в логической последовательности;

простоту выбора нужной информации;

возможность идентификации связанных групп информации;

различимость исключительных ситуаций (сообщений об ошибках или предупреждений);

возможность определить, какое действие со стороны пользователя требуется для выполнения задания.

Правила, регулирующие плотность расположения данных на экране (или в пределах окна):

оставлять пустым приблизительно половину экрана (окна); оставлять пустую строку после каждой пятой строки таблицы; оставлять четыре-пять пробелов между столбцами таблицы.

Другой набор рекомендаций определяется факторами, связанными с право-левой

асимметрией головного мозга человека. Известно, что левое и правое полушария по- разному участвуют в восприятии и переработке информации. При при запоминании слов ведущую роль играет левое полушарие, а при запоминании образов более активно правое.

Информация с правой части экрана поступает непосредственно в левое полушарие, а с левой части — в правое. В связи с этим можно рекомендовать текстовые сообщения группировать справа, а изображения — слева.

Рациональное размещение данных на экране является наиболее важным, но не единственным методом обеспечения удобства и естественности пользовательского интерфейса.

Выделение информации — это использование таких атрибутов: цвет символов, цвет фона, уровень яркости, мерцание и применение различных шрифтов для выводимых символов.

Один из возможных подходов к решению этой проблемы — отделить содержание от формы.

Для этого применяются два метода — прямоугольников и выделенных точек.

При использовании метода прямоугольников после разбиения экрана на поля каждое из них заполняется произвольным текстом и отделяется от других по всему периметру, по крайней мере, одним пробелом. Через центр экрана мысленно проводятся оси, позволяющие оценить сбалансированность размещения полей.

Метод выделенных точек позволяет определить число и размещение областей экрана, к которым будет привлечено внимание пользователя (из-за повышенной яркости, цвета или мерцания символов). Для этого каждая область, требующая повышенного внимания, моделируется группой символов, отличных от пробела.

Графический пользовательский интерфейс

Особенности графического пользовательского интерфейса:

Использование единой рабочей среды пользователя в виде так называемого Рабочего стола;

Объектно-ориентированный подход к описанию заданий пользователей;

Использование графических окон в качестве основной формы отображения данных;

Применение средств неклавиатурного ввода, основанного на выборе и указании с помощью манипулятора «мышь».

1.5. Объектный подход к проектированию интерфейса

Разработка, управляемая данными (сокращенно DCD — Data-centered Design) означает, что проектирование интерфейса поддерживает такую модель взаимодействия пользователя с системой, при которой первичными являются обрабатываемые данные, а не требуемые для этого программные средства.

Другими словами, при таком подходе основное внимание пользователя концентрируется на тех данных, с которыми он работает, а не на поиске и загрузке необходимого приложения.

При использовании DCD-технологии основным программным объектом является документ, который представляет собой некоторое абстрактное устройство хранения данных, используемых для выполнения заданий пользователей и для их взаимодействия. Документ должен быть доступен как различным приложениям, используемым для его обработки, так и всем взаимодействующим пользователям

Рассмотренные выше особенности графических интерфейсов, а также положенная в основу их реализации DCD-технология обуславливают необходимость применения для проектирования GUI объектно-ориентированного подхода.

Такой подход предполагает использование аналогий между программными объектами и объектами реального мира. С точки зрения пользовательского интерфейса, объектами являются не только файлы или пиктограммы, но и любое устройство для хранения и обработки информации, включая ячейки, параграфы, символы, и т.д.

Шаги проектирования эффективного пользовательского интерфейса

Первым шагом в объектно-ориентированном проектировании интерфейса должен быть анализ целей пользователей и особенностей выполняемых ими заданий. При проведении такого анализа следует определить основные компоненты или объекты, с которыми взаимодействует пользователь, а также характерные особенности объектов каждого типа. Необходимо также выявить перечень операций, выполняемых над объектами, их влияние на состояние и свойства объектов.

Описание взаимодействия пользователя с объектами различных типов. На этом шаге выбирается форма визуального представления объектов. При этом следует иметь в виду, что визуальный образ объекта в зависимости от ситуации может изменяться.

Компоновка и пространственное размещение на экране визуальных элементов интерфейса. На этом этапе должны быть решены такие проблемы, как выбор цвета, размера и других атрибутов этих элементов, а также выбор средств и методов привлечения внимания пользователя к наиболее важной информации, отображаемой на экране.

Предусмотреть возможность удобного доступа пользователя к средствам помощи, независимо от того, на каком шаге выполнения задания он находится, и какая именно информация представлена на экране

ЧАСТЬ II

1. ОСНОВНЫЕ ПОНЯТИЯ ОБЪЕКТНО-ОРИЕНТИРОВАННОГО ПРОГРАММИРОВАНИЯ

Объектно-ориентированное программирование (ООП) - это парадигма программирования, в которой основными концепциями которой являются понятия объектов и классов.

Класс – это абстрактный тип данных, описывающий устройство объектов. В классе описывается набор свойств (атрибутов) и операций над ними (методов). По сути, класс является чертежом, по которому создаётся объект.

Объекты - это структуры, которые содержат данные и процедуры. Объект – это экземпляр класса. Каждый объект обладает свойствами, методами и обработчиками событий.

Свойства (Properties) – это переменные, которые влияют на состояние объекта (например ширина и высота, настройки шрифта и т.п.);

Методы (Methods) – это процедуры и функции, принадлежащие классу или объекту;

События (Events) – это сообщения, которые возникают при выполнении определённых условий. Обработчики событий предназначены для описания реакции объекта на определённое событие (например, создание формы, нажатие клавиши и т.п.)

Современные среды разработки предоставляют пользователю широкий спектр заранее заготовленных классов (компонентов), предназначенных для выполнения различных целей.

ООП опирается на три основных понятия: инкапсуляция, наследование и полиморфизм.

Инкапсуляция – это свойство системы, позволяющее объединить данные и методы, работающие с ними, в классе, и скрыть детали реализации от пользователя.

Наследование – это свойство системы, позволяющее описать новый класс на основе уже существующего с частично или полностью заимствующейся функциональностью. Класс, от которого производится наследование, называется базовым, родительским или суперклассом. Новый класс – потомком, наследником, дочерним или производным классом. Если объект наследует свои свойства от одного родителя, то говорят об одиночном наследовании. Если объект наследует данные и методы от нескольких базовых классов, то говорят о множественном наследовании.

Полиморфизм – это свойство, которое позволяет один и тот же идентификатор (одно и то же имя) использовать для решения двух и более схожих, но технически разных задач (например, перемножения действительных или комплексных чисел, перемножение матриц).

Ещё одним важным понятием ООП является абстракция.

Абстрагирование – это способ выделить набор значимых характеристик объекта, исключая из рассмотрения незначимые. Соответственно, **абстракция** – это набор всех таких характеристик.

Delphi – императивный, структурированный, объектно-ориентированный язык программирования со строгой статической типизацией переменных. Основная область использования – написание прикладного программного обеспечения.

Первоначально носил название **Object Pascal** и исторически восходит к одноименному диалекту языка, разработанному в фирме Apple в 1986 году группой Ларри Теслера. Однако в настоящее время термин Object Pascal чаще всего употребляется в значении языка среды программирования Delphi. Начиная с Delphi 7, в официальных документах Borland стала использовать название **Delphi** для обозначения языка Object Pascal.

Синтаксис языка Delphi практически не отличается от синтаксиса языка Pascal. Различия в синтаксисе обусловлены ООП - важную роль в программе имеют обращения к объектам, а точнее, к их свойствам и методам, - и постоянным развитием языка.

Основной средой разработки ПО на языке Delphi является **Embarcadero Delphi**, первоначально созданная фирмой Borland и на данный момент принадлежащая и разрабатываемая Embarcadero Technologies. Эта среда является частью пакета **Embarcadero RAD Studio**.

2.1 Главное меню Embarcadero Delphi

Главное меню Embarcadero Delphi состоит из 11 пунктов:

1. Меню **File (Файл)** содержит команды для работы с файлами - создание нового проекта, формы, модуля или других файлов, открытие проекта и других файлов, сохранение текущего файла или всех открытых файлов и т.д.
2. Меню **Edit (Редактирование)** содержит основные команды редактирования (Копировать, Вырезать, Вставить и др.), а также специфичные команды работы с визуальными компонентами на форме.
3. Меню **Search (Поиск)** содержит стандартные для текстовых редакторов команды поиска и замены фрагментов текста.
4. Меню **View (Вид)** позволяет включить или отключить некоторые окна или панели среды разработки.
5. Меню **Refactor (Улучшение)** содержит команды для рефакторинга (реорганизации) кода. Это меню далее рассмотрено более подробно.
6. Меню **Project (Проект)** содержит основные команды управления проектом. Это меню далее рассмотрено более подробно.
7. В меню **Run (Запуск)** можно найти функции запуска приложения из среды разработки и отладки программ.
8. В меню **Component (Компонент)** находятся пункты меню, с помощью которых можно создать новый компонент или установить существующий пакет.
9. В меню **Tools (Инструменты)** находятся пункты меню, с помощью которых можно настроить среду разработки (например, Options (Опции)

покажет окно глобальных настроек), а также множество дополнительных утилит.

10. С помощью меню **Window (Окно)** можно переключаться между открытыми окнами.

11. В меню **Help (Справка)** содержится большой объём вспомогательной информации.

2.1.1 Меню Refactor

Рефакторинг или **реорганизация кода** – это процесс изменения внутренней структуры программы, не затрагивающий её внешнего поведения с целью сделать код программы легче для понимания.

Рассмотрим подробнее пункты этого меню:

- **Rename (Переименование)** - переименование переменной, объекта, процедуры или функции.
- **Declare variable (Объявить переменную)** - создаёт объявление необъявленной переменной. Тип переменной при этом подбирается автоматически, однако может быть изменён вручную. Есть возможность объявить массив и задать начальное значение.
- **Declare field (Объявить поле)** - этот пункт очень похож на Declare variable, но переменная при этом будет описана как поле заданного класса.
- **Extract Method (Извлечь метод)** - позволяет преобразовать выделенные строчки кода в отдельную процедуру.
- **Extract Resource String (Преобразовать строковую константу в ресурс)**. С помощью этой команды в программе создается раздел **resourcestring** (если он не существовал ранее), и выбранная строка записывается в него с заменой в исходном тексте на идентификатор, указанный в поле Name. Эта возможность используется при переводе приложений на другой язык.
- **Change Params (Изменить параметры)** - позволяет изменить набор аргументов выбранной процедуры или функции.
- **Find Unit (Найти модуль)** - позволяет по имени процедуры или функции найти модуль, содержащий её, и подключить его в программу.

2.1.2 Меню Project

Приложение собирается из многих элементов: форм, программных модулей, внешних библиотек, картинок, пиктограмм и др. Каждый элемент размещается в отдельном файле и имеет строго определенное назначение. Набор всех файлов, необходимых для создания приложения, называется проектом. Компилятор последовательно обрабатывает файлы проекта и строит из них выполняемый файл. Основные файлы проекта можно разделить на несколько типов:

- Файлы описания форм - текстовые файлы с расширением DFM, описывающие формы с компонентами. В этих файлах запоминаются начальные значения свойств, установленные вами в окне свойств.

• Файлы программных модулей - текстовые файлы с расширением PAS, содержащие исходные программные коды на языке Delphi. В этих файлах вы пишите методы обработки событий, генерируемых формами и компонентами.

• Главный файл проекта - текстовый файл с расширением DPR, содержащий главный программный блок. Файл проекта подключает все используемые программные модули и содержит операторы для запуска приложения. Этот файл среда Delphi создает и контролирует сама.

Меню Project (Проект) содержит основные команды управления проектом и входящими в него файлами:

• **Add to project** – добавить в проект существующий файл;

• **Remove from project** – удалить из проекта модуль;

• **Add to repository** – добавить в хранилище. Модуль будет добавлен в качестве шаблона, и на его основе можно будет создавать окна;

• **View source** – просмотреть исходный код проекта. Это не код какого-то модуля, это код именно проекта, где Delphi автоматически генерирует код инициализации автоматически создаваемых форм;

• **Compile XXXX** – компилировать XXXX проект. Вместо XXXX вы будете ви¬деть имя текущего проекта;

• **Build XXXX** – построить проект;

• **Compile All Projects** – компилировать все открытые проекты;

• **Build All Projects** – построить все открытые проекты;

• **Options** – свойства проекта.

Чем отличается компиляция от построения? Когда происходит компиляция программы, то Delphi создает промежуточные файлы, которые в дальнейшем используются при сборке проекта в исполняемый файл. При следующей компиляции неизмененные модули компилироваться не будут, а будут использоваться уже созданные ранее промежуточные файлы. При построении проекта компилируется все. Это необходимо, когда вы внесли какие-то изменения в настройках проекта, а Delphi не перекомпилирует модули.

Чтобы быстро скомпилировать проект, можно использовать сочетание клавиш **Ctrl+F9**. Компиляция очень удобна, чтобы проверить код на наличие ошибок.

2.2 Стандартные файлы проекта Delphi

Вся информация о любом Delphi проекте размещается в файлах, большинство из которых создается автоматически в процессе проектирования приложения.

Рассмотрим более подробно основные из этих файлов:

Тип файла	**Описание**
Файл проекта (.dpr)	Содержит операторы инициализации и запуска программы.
Файл проекта (.dproj)	Содержит служебную информацию о проекте, его версии.
Файл группы проектов (.groupproj)	Файл группы проектов (.groupproj)

Тип файла	Описание
Файл модуля (.pas)	Содержит программный код. Для каждой создаваемой формы автоматически создается файл модуля, однако в некоторых случаях модуль может быть не связан с какой-либо формой.
Файл формы (.dfm)	Содержит информацию о ваших формах и фреймах. Каждому файлу (.dfm) соответствуем модуль (.pas).
Файл ресурсов (.res)	Содержит используемую проектом пиктограмму и др. ресурсы.
Файл конфигурации (.cfg)	Сдержит установки проекта, директивы компилятора.
Объектный файл модуля (.dcu)	Откомпилированный файл модуля (.pas), который компонуется в исполняемый файл.
Файлы резервных копий (.~pa, .~dp, .~df)	Резервные копии файлов проекта.

В ходе написания программы программисту в основном приходится иметь дело с формами и модулями.

2.3 Запуск среды разработки

После запуска среды Embarcadero Delphi откроется стартовое окно среды разработки (Welcome Page):

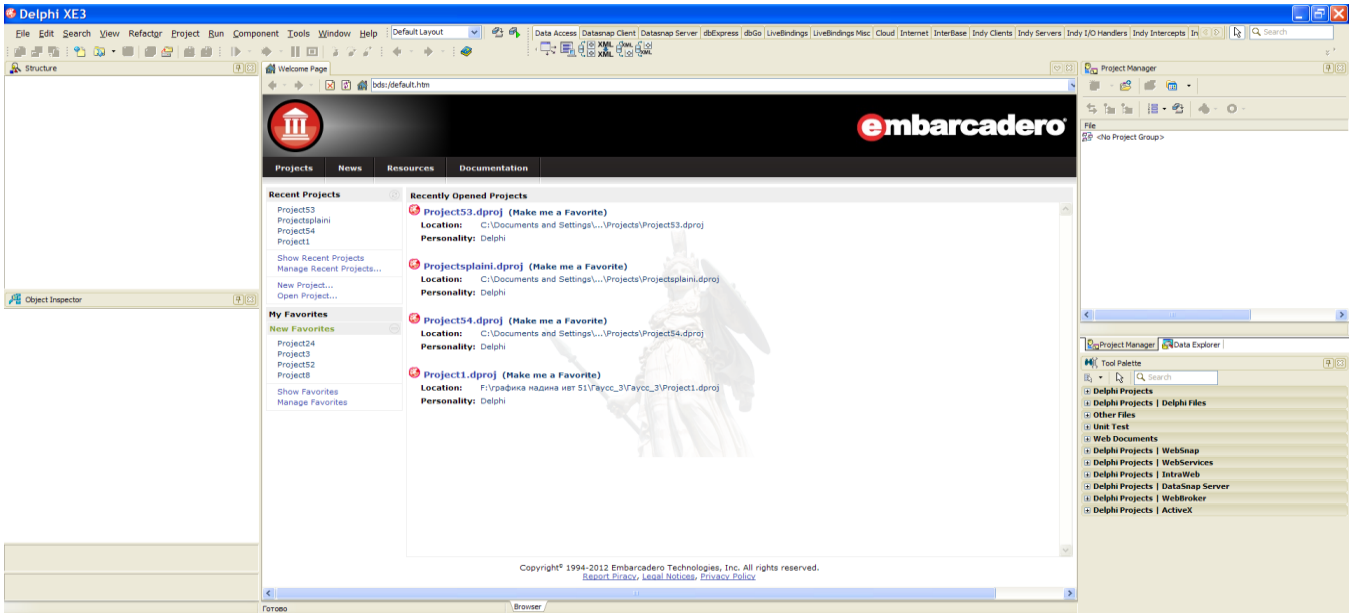


Рисунок 1 – Стартовое окно среды Embarcadero Delphi

В этом окне содержится список недавно открытых проектов и файлов, и предоставляется возможность создать новый проект (**New Project...**) или открыть уже существующий (**Open Project...**). Создать новый проект можно также через главное меню: **File | New | VCL Forms Application - Delphi**.

После создания или открытия проекта откроется главное окно среды разработки.

2.4 Элементы главного окна

Главное окно среды разработки состоит из нескольких элементов:

1. Главное меню программы.

- **Панель инструментов** содержит кнопки для вызова наиболее часто используемых команд главного меню
- **Дерево компонентов (Structure)** - отображает иерархическую структуру компонентов на форме (когда открыто окно Конструктора формы) или структуру кода программы (когда открыто окно Редактора кода).
- **Инспектор объектов (Object Inspector)** - предназначен для управления объектами. Он состоит из двух вкладок - **Properties (Свойства)** и **Events (События)**. Вкладка Properties служит для установки нужных свойств компонента, вкладка Events позволяет определить реакцию компонента на то или иное событие.
- В **Окне конструктора формы (Form Desinger)** выполняется проектирование формы, для чего на форму из Палитры компонентов помещаются необходимые компоненты.
- **Окно редактора кода (Code Editor)** представляет собой обычный текстовый редактор, с помощью которого можно редактировать текст моду¬ля и другие текстовые файлы приложения, например, файл проекта. Окно формы и окно редактора кода находятся в центре экрана и не могут быть открыты одновременно. Переключение между ними можно выполнять с помощью вкладок **Code (Код)** и **Design (Дизайн)** внизу главного окна или с помощью клавиши **F12**.
- **Менеджер проекта (Project Manager)** - в этом окне отображены сведения о конфигурации сборки (Build Configurations), платформе приложения (Target Platforms) и все файлы, входящие в текущий проект. Можно добавлять, удалять и открывать модули.
- **Палитра инструментов (Tool Palette)** содержит все визуальные и невизуальные компоненты Delphi. Для удобства компоненты разбиты на функциональные группы, для каждой из которых имеется отдельная вкладка.

2. Формы

Форма является основным визуальным компонентом Delphi и представляет собой основу, на которую помещаются все остальные компоненты приложения.

При создании нового проекта Delphi автоматически создаёт в нём форму Form1.

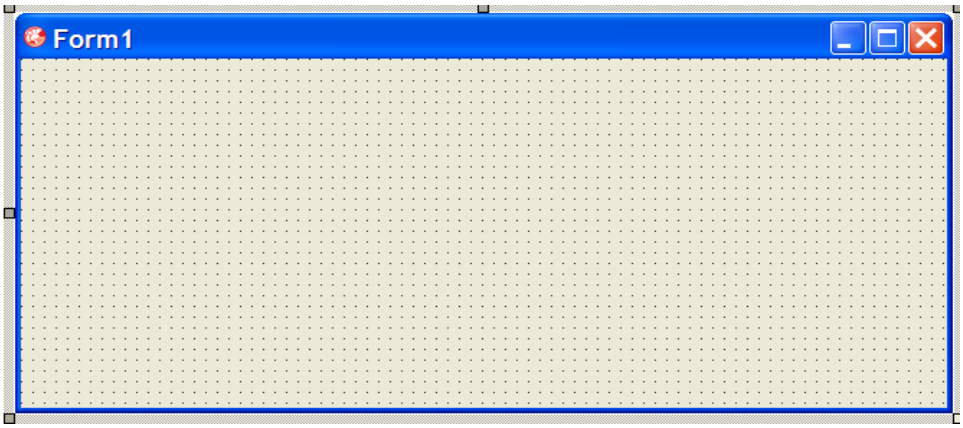


Рисунок 2 – Форма

Размещение компонентов на форме происходит очень просто: следует выбрать нужный компонент из палитры, щелкнув по нему мышкой, а затем щелкнуть по тому месту на форме, где этот компонент должен находиться по вашему замыслу (также можно перетащить компонент из палитры в нужное место).

При необходимости можно добавить в проект другие формы (через меню **File | New | VCL Form - Delphi**). Добавленная таким образом форма будет автоматически создана при запуске программы. Управлять процессом автоматического создания форм можно, непосредственно редактируя файл проекта или выполняя настройки в окне параметров проекта (меню **Project | Options**, список **Auto-create forms** на странице **Form**). Если форма переведена из этого списка в список доступных форм (**Available forms**) проекта, то инструкция ее создания исключается из файла проекта, и программист в ходе выполнения приложения должен динамически создать экземпляр этой формы.

Для создания экземпляров форм служит метод (конструктор) **Create.** Сам класс формы обычно предварительно описывается при конструировании приложения, и для формы уже существуют файлы формы (.dfm) и программного модуля (.pas).

Пример создания формы (Form 2):

Form2 := TForm2.Create(Application); {Форма создается, но не отображается на экране};

Form2.Caption := 'Новая форма'; //новый заголовок формы. Form2.Show; //Вывод формы на экран.

3. Модули

Файлы модулей содержат программный код, оформленный по обычным правилам языка Delphi. Для подключения модуля его имя указывается в разделе uses того модуля или проекта, который использует средства этого модуля.

Delphi автоматически создает новый модуль всякий раз, когда создается новый проект. В процессе создания среда Delphi устанавливает необходимые связи для ассоциирования форм с модулями, что освобождает программиста от

выполнения рутинных задач. Рассмотрим структуру стандартного модуля, создаваемого вместе с формой:

```
unit Unit1;
interface
uses
Winapi.Windows, Winapi.Messages, System.SysUtils, System.Variants, System.Classes, Vcl.Graphics,Vcl.Controls, Vcl.Forms, Vcl.Dialogs;
type
TForm1 = class(TForm)
private
{ Private declarations }
public
{ Public declarations }
end;
var
Form1: TForm1;
implementation
{$R *.dfm}
end.
```

Первая строка модуля по умолчанию объявляет имя модуля: *unit Unit1;*

После нее следует раздел *interface*, который уведомляет Delphi какие объекты, переменные, процедуры и т.д. доступны для других модулей. Если не определять переменных в этом разделе, то они не будут доступны для других модулей. После зарезервированного слова interface записывается необязательный раздел *uses*, в котором содержится список подключаемых модулей.

Далее следует зарезервированное слово *type* - оно создает ассоциацию, необходимую для того, чтобы форма существовала внутри модуля. Разделы *public* и *private* предназначены для определения, будут ли свойства формы доступными (public) или недоступными (private) для других модулей.

Затем следует раздел описания глобальных переменных (переменных, доступных всем процедурам и функциям в данном модуле) *var*. Здесь же форме присваивается имя, для того чтобы дать ей адрес в памяти.

В завершении описывается раздел *implementation* (реализация). Именно в этом разделе записывается код, который вводится для обработчиков событий. Самая последняя строка каждого модуля должна содержать точку после зарезервированного слова *end.*

Модуль может быть не связан с формой. При создании такого модуля его структура отличается от рассмотренной выше отсутствием некоторых разделов:

```
unit Unit3;
interface
implementation
end
```

2.5 Понятие компонентов

Среда разработки Delphi ориентирована, прежде всего, на создание программ для Windows. При этом большое внимание уделяется возможности визуальной разработки приложений с помощью большого набора готовых компонентов, позволяющих избежать ручного кодирования. Компоненты Delphi охватывают практически все аспекты применения современных информационных технологий.

Компоненты можно рассматривать с двух сторон:

- С точки зрения визуальной среды разработки, компоненты – это самодостаточные элементы приложения, располагающиеся на форме.
- С точки зрения Delphi, компоненты – это классы, порожденные прямо или косвенно от класса TComponent и имеющие опубликованные (published) свойства. Экземпляры компонентов – это объекты этих классов, существующие в качестве полей формы.

Эти подходы дают цельное представление о том, что такое компоненты.

Все множество компонентов подразделяется на две группы: визуальные и невизуальные компоненты.

Визуальные компоненты (Visual components) - это компоненты, которые видны на экране как во время проектирования формы, так и во время работы приложения. К таким компонентам относятся управляющие элементы пользовательского интерфейса (controls), т.е. кнопки, метки, блоки списков и др.

Невизуальные компоненты (Nonvisual components) - это компоненты, которые видны во время проектирования формы, но не видны во время работы приложения. В некоторых случаях пользователь видит сгенерированные такими компонентами меню, диалоговые окна и т.п. К невизуальным компонентам относятся главное и всплывающее меню, различные диалоговые окна, таймер, компоненты доступа к базам данным и др.

Форма, содержащая визуальные и невизуальные компоненты во время проектирования и во время работы программы.

Для удобства в **Палитре компонентов** компоненты разделены на группы по функциональным свойствам и частоте использования. Рассмотрим некоторые страницы.

Страница **Standard** содержит ряд часто используемых компонентов общего назначения. На этой странице расположены стандартные для Windows интерфейсные элементы, такие как главное и всплывающее меню, кнопка, однострочный и многострочный редакторы, переключатели, метки, списки, и некоторые другие компоненты, применяющиеся наиболее часто.

Страница **Additional**, по сути, является дополнением страницы Standard. На эту страницу помещены дополнительные компоненты, без некоторых из которых сегодня трудно представить программу для Windows: кнопки с дополнительными свойствами, таблицы, компоненты для размещения изображений и др.

Страница **Win32** содержит компоненты, представляющие собой интерфейсные элементы для 32-разрядных операционных систем Windows 95/98/NT. Использующие эти компоненты программы выглядят в стилистике последних версий операционных систем Windows.

На странице **System** представлены компоненты, которые имеют различное функциональное назначение (например, Timer - очень важный в любой программе компонент), в том числе компоненты, поддерживающие стандартные для Windows технологии межпрограммного обмена данными OLE и DDE.

Также есть страницы, содержащие более специализированные компоненты, например компоненты для работы с БД.

2.6 Общие свойства и события компонентов

Невизуальные компоненты практически не имеют общих свойств и событий, единственные общие для них свойства – это **Name (Имя)** и **Tag** (целочисленное значение, не несущее смысловой нагрузки (вы можете использовать его по своему усмотрению). А вот управляющие элементы пользовательского интерфейса имеют много общих свойств и событий.

Общие свойства управляющих элементов.

Свойство	Описание
Enabled	Определяет, активен (доступен пользователю) ли компонент.
Height	Высота компонента.
Hint	Всплывающая подсказка.
Left	Положение компонента относительно левого края формы.
Name	Имя компонента.
Top	Отступ от верха формы.
Visible	Определяет видимость компонента.
Width	Ширина компонента.
Caption	Заголовок компонента.
Color	Определяет цвет компонента.
Ctl3D	Определяет внешний вид компонента: рельефный или плоский.
Cursor	Определяет, какое изображение принимает курсор мыши, когда пользователь переводит его на компонент.
Font	Определяет шрифт надписи на компоненте. Параметры шрифта задаются с помощью вложенных свойств Color (Цвет), Name (Имя), Size (Размер), Style (Стиль), Height (Высота).
ShowHint	Определяет, включена ли всплывающая подсказка для компонента.
PopupMenu	Используется для привязки к компоненту локального всплывающего меню.

TabOrder	Содержит порядковый номер компонента в пределах своего владельца. Это номер очереди, в которой компонент получает фокус ввода при нажатии клавиши Tab.
TabStop	Определяет, может ли управляющий элемент получать фокус ввода.

Общие события управляющих элементов.

Событие	Описание
OnClick	Возникает в результате щелчка по управляющему элементу. Это означает, что пользователь перевел курсор на управляющий элемент и щелкнул левой кнопкой мыши.
OnDblClick	Возникает в результате двойного щелчка кнопки мыши, когда курсор находится над управляющим элементом.
OnEnter OnExit	Происходит при получении управляющим элементом фокуса ввода. При потере фокуса ввода в управляющем элементе генерируется событие OnExit. События OnEnter и OnExit не возникают при переключении между формами и приложениями.
OnKeyDown On Key Up OnKeyPress	Происходит при нажатии пользователем любой клавиши, когда управляющий элемент находится в фокусе ввода. При отпускании нажатой клавиши возникает событие On Key Up. Если пользователь нажал символьную клавишу, то вслед за событием OnKeyDown и до события OnKeyUp возникает событие OnKeyPress.
OnMouseDown OnMouseUp OnMouseMove	Происходит при нажатии пользователем кнопки мыши, когда курсор находится над управляющим элементом. При отпускании кнопки в управляющем элементе происходит событие OnMouseUp. При перемещении курсора мыши над управляющим элементом, в последнем периодически возникает событие OnMouseMove, что позволяет реагировать на простое изменение положения курсора.

2.7 Обращение к свойствам и методам

Ссылка на определённое свойство определённого объекта осуществляется в следующем формате:

<имя объекта>.<имя свойства>

После имени объекта пишется без пробела символ точки, а затем так же без пробела пишется имя свойства. Например, ссылка на свойство *Text* поля *Edit1* осуществляется записью *Edit1.Text.*

Иногда свойство объекта является в свою очередь объектом. Тогда в обращении к этому свойству указывается через точки вся цепочка предшествующих объектов. В качестве примера можно привести свойство,

доступное многим объектам - *Font* (шрифт), которое в свою очередь является объектом со своим набором свойств:

Memo1.Font.Color

Аналогичная нотация с точкой используется и для доступа к методам объекта:

<имя объекта>.<имя метода>

Теперь посмотрим, чем различаются константы, переменные, функции и процедуры, включенные и не включенные в описание класса.

Если в приложении создается только один объект данного класса, то различие в основном чисто внешнее. Для процедур, объявленных в классе, в их описании в разделе implementation к имени процедуры должна добавляться ссылка на класс. Для процедур, объявленных вне класса, такой замены не требуется.

Обращение к переменным и процедурам, описанным внутри и вне класса, из процедур, описанных вне класса, различается. К переменным и процедурам, описанным вне класса, обращение происходит просто по их именам, а к переменным и процедурам, описанным в классе, через имя объекта класса.

Например, если вы вне класса хотите описать какую-то процедуру, изменяющую надпись метки *Label1*, вы должны обратиться к ее свойству *Caption* следующим образом: *Form1.Label1.Caption*. Только через ссылку на объект *Form1* внешние по отношению к классу процедуры могут получить доступ ко всему, объявленному в классе. Однако эти ссылки на объект не требуются в процедурах, объявленных в классе. Поэтому в процедуре *TForm1.Button1Click* ссылка на объект *Label1* не содержит дополнительной ссылки на объект формы.

Если в приложении создается несколько объектов одного класса, например, несколько форм класса *TForm1*, то проявляются более принципиальные различия между переменными, описанными внутри и вне класса. Переменные вне класса так и остаются в одном экземпляре. А переменные, описанные в классе, тиражируются столько раз, сколько объектов данного класса создано. Т.е. в каждом объекте класса TForm1 будут свои переменные, описанные в классе, и их значения никак не будут связаны друг с другом.

Таким образом, в переменную, описанную внутри класса, можно заносить какую-то информацию, индивидуальную для каждого объекта данного класса. А переменная, описанная в модуле вне описания класса, может хранить только одно значение.

Оператор With

Оператор **with** используется для сокращения записи при обращении к полям записи или к свойствам и методам объекта. В этих случаях применение with позволяет избежать повторных ссылок на объект в последующих операторах. Например, группу операторов

```
Form1.label1.left:=Form1.label1.left+10;
Form1.label1.caption:=label2.caption;
Form1.label1.font.color:=clRed;
```

с помощью with можно записать намного короче:

```
with Form1.label1 do begin
left:=Form1.label1.left+10;
caption:=label2.caption;
font.color:=clRed;
end;
```

Однако использование этого оператора может, к удивлению, сделать код более трудным для восприятия. А также это может создать такие проблемы, где изменение кода может подразумевать неправильный адресат для 'дочернего' поля, которое упоминают.

2.8 Типы данных

Delphi является языком со статической типизацией переменных. Это значит, что переменная, параметр подпрограммы, возвращаемое значение функции связывается с типом в момент объявления и тип не может быть изменён позже (переменная или параметр будут принимать, а функция – возвращать значения только этого типа).

В Delphi предусмотрено большое число типов данных, которые подразделяются на следующие категории:

- Простые типы для хранения информации в форме чисел и других "упорядоченных" значений;
- Строковые типы для хранения последовательностей символов;
- Структурные типы для одновременного хранения информации разных типов;
- Указательные типы для косвенного обращения к переменным заданных типов;
- Процедурные типы для обращения к процедурам и функциям, рассматриваемым как переменные;
- Вариантные типы для хранения в одной переменной данных различных типов.

2.8.1 Простые типы данных

Простые типы данных можно разделить на следующие подгруппы:

1. Порядковые типы:
   - Целые типы
   - Символьные типы
   - Булевы типы
   - Перечислимые типы
   - Диапазонные типы
2. Действительные типы

Порядковые типы.

Порядковые типы - самые простые. В этих типах информация представляется в виде отдельных элементов, образующих упорядоченную последовательность. Значение переменной порядкового типа - это её место в этой последовательности. В большинстве простых типов, нумерация элементов начинается с 0. Исключение - целые типы, в которых порядковый номер значения равен самому значению и может принимать как положительные, так и отрицательные значения.

В Delphi определены три группы порядковых типов (целые, символьные и булевы типы) и два типа, определяемых пользователем (перечисления и диапазоны).

Для работы с порядковыми типами определены следующие операции:

Операция	Описание
Low (type or variable): Ordinal type;	Возвращает минимальное позволенное значение типа или переменной этого типа.
High (type or variable): Ordinal type;	Возвращает максимальное позволенное значение типа или переменной этого типа.
Ord (Arg: Ordinal Type): Integer;	Возвращает целочисленное значение для любого перечислимого типа Arg.
Pred (Ordinal Value): Ordinal type;	Возвращает предыдущее по порядку значение.
Succ (Ordinal Value): Ordinal type;	Возвращает следующее по порядку значение.
Dec (Variable: Ordinal variable);	Уменьшает значение переменной на 1.
Dec (Variable: Ordinal variable; Count: Integer);	Уменьшает значение переменной на числоCount.
Inc (Variable: Ordinal variable);	Увеличивает значение переменной на 1.
Inc (Variable: Ordinal variable; Count: Integer);	Увеличивает значение переменной на числоCount.
T(X)	Преобразование целого значения X в значение порядкового типа T с порядковым номером X.

Целые типы

В переменных целых типов информация представляется в виде целых чисел, т.е. чисел не имеющих дробной части.

В Delphi определены следующие целые типы:

Тип	Описание	Объём памяти (байт)	Принимаемые значения
Byte	Беззнаковое целое	1 байт	0..255
Shortint	Знаковое целое	1 байт	-128..127
Word	Беззнаковое целое	2 байта	0..65535
Smallint	Знаковое целое	2 байта	-32768..32767
CardinalLongword	Беззнаковое целое	4 байта	0..4294967295 или $0..2^{32}-1$
Integer Longint	Знаковое целое	4 байта	-2147483648..2147483647 или $-2^{31}..2^{31}-1$
Uint64	Беззнаковое целое	8 байт	0..18446744073709551615 или $0..2^{64}-1$
Int64	Знаковое целое	8 байт	-9223372036854775808…9223372036854775807 или $-2^{63}..2^{63}-1$

Почти все целые типы данные являются физическими (фундаментальными) - они занимают определённое количество байтов памяти и не меняются в зависимости от версии Delphi.

Типы integer и cardinal являются логическими (общими). Их диапазон значений зависит от микропроцессора, ОС и версии Delphi. На данный момент эти типы могут содержать 4-байтные значения.

Операции, предназначенные для работы с порядковыми типами, могут применены и для работы с целыми данными, но с ними удобней работать как с числами. Над целыми числами можно произвести арифметические действия - сложение (+), вычитание (-), умножение (*) и деление (/), а также и ряд других операций:

Операция	Результат
Abs(X)	Возвращает абсолютное целое значение (модуль) X
X DivY	Возвращает целую часть частного деления X на Y
X ModY	Возвращает остаток частного деления X на Y
Odd(X)	Возвращает булево True (истина), если X – нечетное целое, и False(ложь) – в противном случае
Sqr(X)	Возвращает квадрат X (т.е. X*X)

Символьные типы

Как понятно из названия, символьные типы данных предназначены для работы с символьной информацией. Значению переменной символьного типа ставится в соответствие символ из таблицы ANSI (1 байт) или UNICODE (2 байта).

В Delphi определены два физических и один логический тип символьных данных. Физические типы приведены в таблице:

Тип	Описание
AnsiChar	Однобайтовые символы, упорядоченные в соответствии с расширенным набором символов ANSI
WideChar	Символы объемом в слово, упорядоченные в соответствии с международным набором символов UNICODE. Первые 256 символов совпадают с символами ANSI

Логический символьный тип - char - зависит от ОС и языковых настроек. Значения символьным переменным могут быть присвоены несколькими способами. Обычные символы (буквы, цифры и некоторые другие знаки, включая пробел) перед присваиванием заключаются в одинарные кавычки. Специальные управляющие символы, например, возврат каретки (Enter) назначают при помощи их номера в таблице ANSI, и выделяют знаком решетки (этот способ применим ко всем символам):

```
var a,b: char;
...
a:='A';
b:=#13;
```

Для работы с символами в Delphi доступны некоторые специальные команды:

Операция	Результат
Chr(X)	Преобразует целую переменную в переменную типа char с тем же порядковым номером. В Delphi это эквивалентно заданию типа Char (X).
UpCase(C)	Преобразует строчную букву в прописную.

Булевы типы

Булевы типы данных названы в честь Георга Буля, одного из авторов формальной логики.

Язык Delphi поддерживает четыре булевых типа данных, которые приведены в таблице:

Тип данных	Диапазон значений	Объем памяти (байт)
Boolean	False, True	1
ByteBool	False (0), True (не равно 0)	1 (диапазон Byte)
WordBool	False (0), True (не равно 0)	2 (диапазон Word)
LongBool	False (0), True (не равно 0)	4 (диапазон LongWord)

Переменным типа Boolean можно присваивать только значения **True (Истина)** и **False (Ложь)**.

Переменные ByteBool, WordBool и LongBool могут принимать и другие порядковые значения, интерпретируемые обычно как False в случае нуля и True– при любом ненулевом значении.

Основным булевым типом, применяющимся в Delphi, является тип Boolean. Остальные типы введены для обеспечения совместимости с другими средами программирования.

2.8.2 Строковые типы

Строковые типы данных предназначены для работы со строками.

Строка –это последовательность символов. В Delphi существует несколько строковых типов, отличающихся допустимой длиной строки и кодировкой символов:

Тип	Максимальная длина (символы)	Используемая память	Кодировка символов
ShortString	255	от 2 до 256 байт	ANSI
AnsiString	Около 2^{31}	от 4 байт до 2 Гб	ANSI
WideString	Около 2^{30}	от 4 байт до 2 Гб	Unicode

Логический строковый тип называется String. В зависимости от настроек компилятора, тип String може рассматриваться как ShortString или AnsiString.

Работа со строками похожа на работу с массивами – можно так же обращаться к строке как единому объекту, так и к отдельному символу. При этом в Delphi предусмотрен ряд операций для работы со строками:

Функция	Описание
Concat (sl,s2,...,sN: String): String;	Конкатенация (объединение) строк. Эквивалентна оператору $s_1+s_2+...+s_N$
Copy (Source: String; StartChar, Count : Integer): String;	Возвращает подстроку длиной Count символов, начинающуюся в позиции StartChar строки Source.
Delete (Source: String; StartChar: Integer; Count: Integer) ;	Удаляет Count символов из строки Source, начиная с позиции StartChar.
Insert (InsertStr: string; TargetStr: string; Position: Integer) ;	Вставляет строку InsertStr в строковую переменную TargetStr, начиная с позиции Position.
Length (S: String): Integer;	Возвращает длину строки.
Pos (Needle, HayStack: string) : Integer;	Возвращает место первого вхождения подстроки Needle в строку HayStack.
SetLength (StringToChange: string; NewLength: Integer) ;	Задает новую динамическую длину NewLength строковой переменной StringToChange.
SetString (TargetString: string; BufferPointer: PChar; Length: Integer) ;	Копирует Length символов из буфера BufferPointer в строку TargetString.
Str (X, S);	Преобразует численное значение X в строковую переменную S.

StringOfChar (RepeatCharacter: Char; RepeatCount: Integer): String;	Создает строку из одного символа RepeatCharacter, повторенного RepeatCount раз.
Val (String: string; Var: Number Type; ErrorCode : Integer) ;	Преобразует строку String в соответствующее численное представление Var. Если преобразование успешно, то ErrorCode устанавливается в 0. Иначе, он устанавливается на первый символ в String, который привел к ошибке преобразования.

В Delphi предусмотрены несколько функций для преобразования чисел в строки и наоборот:

Функция	**Описание**
IntToStr (Number: Integer): string;	Преобразует целое число Integer или Int64 в строку
FloatToStr (Value: Extended): string;	Преобразует значение Value плавающей запятой в визуализуемую строку.
FloatToStrF (Value: Extended; Format: TFloatFormat; Precision, Digits: Integer): string;	Конвертирует значение Value с плавающей запятой в визуализуемую строку, с большим управлением по форматированию через значения Format, Precision, и Digits.
StrToFloat (FloatString: string): Extended;	Конвертирует числовую строку FloatString в значение с плавающей точкой с типом Extended.
StrToInt (IntegerString: string): Integer;	Конвертирует строку с целым значением - IntegerString в целое Integer.
StrToInt64 (IntegerString: string): Int64;	Конвертирует строку с целым значением - IntegerString, в целое Int64.

2.8.3 Структурные типы данных

Массивы

Работа с массивами в Delphi полностью аналогична работе с ними в языке Pascal.

Массив – это именованная группа однотипных данных, хранящихся в последовательных ячейках памяти. Каждая ячейка содержит элемент массива. Элементы нумеруются по порядку, но необязательно начиная с единицы. Порядковый номер элемента массива называется индексом этого элемента.

Массивы могут быть одномерными или многомерными. Число элементов массива в каждом измерении задается порядковым типом (ordinal_type). Для этого можно воспользоваться идентификатором некоторого типа (например, Byte или AnsiChar), однако на практике обычно явно задается диапазон целых значений.

array [ordinal_type] of type_definition;
array [ordinal typel, ordinal type2] of type definition;

Массивы могут быть описаны как в разделе описания типов, так и непосредственно при описании переменных:

```
type
array1d = array[1..10] of integer;
array2d = array[1..10, 1..5] of integer;
...
var
massiv1: array1d;
massiv2: array2d;
massiv3: array[1..25] of real;
...
```

Количество элементов массива равно произведению количеств элементов во всех измерениях.

Для обращения к конкретному элементу массива необходимо указать имя этого массива и индекс элемента в квадратных скобках.

Множества

Зарезервированное слово set (множество) определяет множество не более чем из 256 порядковых значений. Не стоит путать множества с перечислениями – они могут принимать только одно значение. Переменная множества всегда держит все значения набора - некоторые установлены, некоторые нет.

Минимальный и максимальный порядковые номера исходного типа (на основе которого определяется множественный тип) должны быть в пределах между 0 и 255. Каждое значение из заданного диапазона может принадлежать или не принадлежать конкретному множеству.

Для описания множеств используется синтаксическая конструкция вида

```
type Name = Set of Ordinal type;
type Name = Set of Value range;
```

Например

```
type
TCharset = set of ANSIChar;
TLetters = set of 'A'..'z';
```

Множественный тип может быть описан как в разделе описания типов, так и непосредственно при описании переменной:

```
var
myChar: TCharset;
myNum: set of '0'..'9';
```

Переменная типа множество может содержать как все элементы множества, так и только некоторые из них, или не содержать ни одного. При

присвоении значения переменной множественного типа элементы множества указываются в квадратных скобках:

myChar := ['A', 'B', 'F', 'K']; // Произвольный набор букв.
myNum := ['0'..'9']; // Полный набор цифр.

Пустые квадратные скобки задают пустое множество, не содержащее ни одного элемента. Это относится ко всем множественными типам. Для работы с множествами в Delphi предусмотрены следующие команды:

Функция	Описание
Exclude (SetVariable: set of SetValues; OneOfSet: SetValues);	Исключает значение множества OneOfSet из переменной множества SetVariable.
In	Ключевое слово In проверяет, является ли значение одним из членов множества. Если да, возвращается true, если нет - false.
Include (SetVariable: set of SetValues; OneOfSet: SetValues);	Включает значение множества OneOfSet в переменную множества SetVariable.

Файловый тип

Файловый тип предназначен для доступа к линейной последовательности элементов, которые могут представлять данные любого типа, кроме содержащих типы file и class. Объявление файлового типа подобно объявлению массива, только без указания числа элементов.

В Delphi предусмотрены три типа файловых переменных:

Тип	Описание
file of Type	Файл определенного типа, содержащий записи фиксированной длины.
file	Файл без типа или "блочный".
textfile	Файл с записями переменной длины, разделенными символами CR и LF ("возврат каретки" и "новая строка").

2.8.4 Указательные типы

Переменная указательного типа содержит значение, указывающее на переменную обычного типа – адрес этой переменной.

В Delphi существует два типа указателей: типизированные и нетипизированные (общие). Типизированные указатели можно использовать только с переменными конкретного типа, в то время как нетипизированные указатели могут указывать на любые данные.

Тип	Описание
Pointer	Указатель без типа
^typename	Указатель с типом

Существуют и предопределённые типизированные указатели: **PAnsiChar, PAnsiString, PChar, PCurrency, PDateTime, PExtended, PInt64, PShortString, PString, PVariant, PWideChar, PWideString**.

Если исходный тип (тип переменной, на которую должен ссылаться указатель) еще не объявлен, его надо объявить в том же разделе объявления типов, что и тип указателя.

Для работы с указателями в Delphi введены следующие средства:

Средство	Описание
New (VariablePointer: Pointer-Type);	Распределяет новый участок динамической памяти и записывает его адрес в переменную указательного типа
Оператор @	Направляет переменную-указатель на область памяти, содержащую любую существующую переменную, процедуру или функцию, включая переменные, имеющие идентификаторы Пример: p :=@n;
GetMem (StoragePointer: Pointer; StorageSize: Integer);	Процедура GetMem пытается получить указанные в StorageSize байт памяти, сохраняя указатель на память в StoragePointer.

Указатели и адресные функции

Информация, содержащаяся в переменной указательного типа – это адрес некоторого участка в машинной памяти. Эти значения задаются во время работы программы и могут меняться от одного запуска к другому.

Следующие функции обеспечивают доступ к адресной информации в программе и тестирование переменных-указателей:

Функция	Описание
Addr (Variable name): Pointer; Addr (Function name): Pointer; Addr (Procedure name): Pointer;	Возвращает адрес переменной, функции или процедуры.
Assigned (PointerName: Pointer): Boolean	Проверяет является ли указатель nil. Если не nil, то возвращает True; если nil, то False.
Ptr	Преобразует адрес в указатель

Зарезервированное слово **Nil** указывает значение указателя, который ни на что не указывает. Такие указатели называют неопределенными.

Определенный в Delphi тип **Pointer** -- это указатель без типа. Обратиться к переменной через такой указатель невозможно (к переменной типа Pointer нельзя дописывать символ "^"). Однако можно задать ей другой указательный тип.

По значениям переменных тип Pointer совместим с остальными указательными типами.

2.8.5 Процедурные типы

Процедурные типы позволяют трактовать процедуры и функции как значения, которые можно присваивать переменным и передавать в качестве параметров.

Объявление процедурного типа подобно объявлению заголовка процедуры или функции. Единственная разница состоит в том, что опускается имя, следующее обычно после ключевых слов procedure и function:

type typename = function(...): type;
type typename = procedure(...): type;

Вне типа Class в Delphi разрешены только глобальные процедурные переменные. Это означает, что процедурной переменной не может быть присвоена процедура или функция, объявленная внутри другой процедуры или функции.

Кроме того, в Delphi можно применять данные процедурных типов, представляющих собой методы. При выполнении программы можно обращаться к определенным методам определенных переменных-объектов. Использование методов в качестве данных процедурных типов позволяет модифицировать поведение переменной-объекта некоторого класса, не объявляя нового класса с новыми методами.

В Delphi указатели процедурного типа на методы применяются для сопоставления событий с образцами текста программы. С точки зрения синтаксиса, единственное отличие процедурного типа для метода от обычного процедурного типа состоит в фразе *of object*, следующей за прототипом процедуры или функции в случае метода. Особым образом применяется в процедурных методах указательное значение *Nil*. Это единственное указательное значение, которое можно присвоить процедурной переменной. После присвоения такого значения процедурная переменная становится неопределенной. Состояние определенности можно проверить с помощью функции *Assigned.*

Глобальные процедурные типы и процедурные типы для методов взаимно несовместимы. Нельзя присваивать значение одного типа переменной другого.

Физически процедурные типы в Delphi совпадают с указательными, однако они различаются синтаксически, поэтому нельзя обращаться к функции или процедуре через указатель. Тем не менее при обращении к процедурной переменной задействуется именно значение указательного типа.

2.8.6 Вариантные типы

Тип **Variant** предназначен для представления значений, которые могут динамически изменять свой тип. Если любой другой тип переменной зафиксирован, то в переменные типа Variant можно вносить переменные

разных типов. Шире всего тип Variant применяется в случаях, когда фактический тип данных изменяется или неизвестен в момент компиляции.

Переменным типа Variant можно присваивать любые значения любых целых, действительных, строковых и булевых типов. Для совместимости с другими языками программирования предусмотрена также возможность присвоения этим переменным значений даты/времени и объектов OLE Automation. Кроме того, вариантные переменные могут содержать массивы переменной длины и размерности с элементами указанных типов.

Все целые, действительные, строковые, символьные и булевы типы совместимы с типом Variant в отношении операции присваивания. Вариантные переменные можно сочетать в выражениях с целыми, действительными, строковыми, символьными и булевыми; при этом все необходимые преобразования Delphi выполняет автоматически. Можно произвольно задавать для выражении тип Variant в форме *Variant (X).*

В Delphi определены два особых значения Variant. Значение **Unassigned** применяется для указания, что вариантной переменной пока не присвоено значение какого бы то ни было типа. Значение **Null** указывает на наличие в переменной данных неизвестного типа или потерю данных. Разницу между этими двумя значениями трудно уловить. Значение Unassigned присваивается вариантным переменным автоматически при их создании, независимо от того, локальная это переменная или глобальная, и является ли она частью другой, структурной, переменной, такой как запись или массив. Unassigned означает, что к данной вариантной переменной еще не обращались. Null же означает, что к вариантной переменной обращались, но не ввели в нее никакой информации. Таким образом, Null указывает, что значение вариантной переменной недействительно или отсутствует.

Вариантные переменные предоставляют широкие возможности формирования выражений с переменными разных типов. Однако за это приходится платить большим, по сравнению с жестко задаваемыми типами, расходом памяти. К тому же на выполнение операций с вариантными переменными требуется больше времени.

2.9 Основные компоненты для работы с текстом

Работа с текстовыми данными, а точнее со строками и символами, в Delphi играет важную роль. Чаще всего, значения переменных вводятся и выводятся именно в текстовом виде с соответствующими преобразованиями типов.

Для обработки текстовых данных в Delphi предусмотрен ряд компонентов, расположенных на разных страницах Палитры инструментов. В модуле будут рассмотрены следующие компоненты:

- Label
- Edit
- Memo
- RichEdit
- StringGrid

Существуют и другие компоненты, но большинство из них базируется на перечисленных выше. Например, компонент **LabeledEdit** сочетает в себе компоненты Label и Edit, а компонент **StaticText**, по сути, является меткой Label с бордюром вокруг текста.

2.9.1 Компонент Label

Компонент **Label** (Метка) расположен в Палитре инструментов на странице **Standard** и предназначен для показа статического текста, содержащего различные пояснения, названия, заголовки и т.п.

Чаще всего текст метки (свойство **Caption**) задаётся при проектировании программы, но он может быть изменён в ходе выполнения программы (только программным методом):

Label1.Caption := 'Текст';

Свойство Caption содержит строковые данные. Для отображения числовой информации необходимо использовать функции конвертирования чисел в строки.

Разбиение текста на строки обеспечивает символ «новая строка» (его код 10 в таблице ASCII):

label1.Caption := 'первая строка'+#10+'вторая строка';

Основные свойства компонента Label приведены в таблице:

Свойство	Описание
Color	Цвет фона компонента.
Font	Свойства шрифта компонента.
Visible	Видимость компонента.
WordWrap	Перенос текста на новую строку.
Caption	Отображаемый текст.
Alignment	Выравнивания текста: taLeftJustify - по левому краю компонента; taCenter - по центру; taRightJustify - выравнивание по правому краю.
Transparent	Прозрачность фона компонента.
Name	Имя компонента.

2.9.2 Компонент Edit

Компонент **Edit** расположен на странице **Standard** в Палитре инструментов и представляет собой однострочное текстовое поле, служащее для ввода данных пользователем, однако может быть использован и для вывода теста.

Основным свойством компонента Edit, передающим введённую информацию, является свойство **Text** типа **String**.

Edit1.Text := 'Текст';

Вводимый в компонент Edit текст никак не влияет на длину (свойство **Width**) этого компонента. Не помещающаяся в установленную длину

часть текста сдвигается за границы компонента. Перенос строк в этом компоненте невозможен.

Основные свойства компонента Edit приведены в таблице:

Свойство	Описание
Text	Содержит отображаемую компонентом строку.
MaxLegth	Максимальная длина текста (0 - количество символов неограниченно).
Font	Свойства шрифта компонента.
ReadOnly	Если значение этого свойство равно True, то текст в поле ввода пользователь изменить не сможет
AutoSize	Автоматическое изменение высоты поля в соответствии с размером шрифта.
BevelEdges BevelInner BevelKind BevelOuter	Эти свойства определяют эффекты объемности поля ввода
BorderStyle	Стиль рамки поля ввода
PasswordChar	Свойство, позволяющее сделать поле для ввода пароля.
AutoSelect	Если значение равно True то при получении фокуса компонентом весь текст будет выделен
CharCase	Позволяет вводить текст определенного регистра. это свойство может принимать следуюие значения: ecUpperCase - текст преобразуется в верхний регистр; ecLowerCase - текст пишется в нижнем регистре; ecNormal (по умолчанию) - размер символом не меняется.
HideSelection	Если False, то выделенный текст сохраняется выделенным при потере компонента фокуса ввода

Методы Edit перечислены ниже в таблице:

Метод	Описание
Clear	Очищает поле ввода.
ClearSelection	Удаляет выделенный текст.
ClearUndo	Очищает буфер метода Undo.
CopyToClipboard	Копирует выделенный текст в буфер обмена.
CutToClipboard	Вырезает выделенный текст в буфер обмена.
PasteFromClipboard	Вставляет текст содержащийся в буфера обмена в позицию курсора.
SelectAll	Выделяет весь текст.
Undo	Восстанавливает текст в той форме, в которой он был перед последним получением компонентом фокуса

Основные события (Events) для Edit'a:

Событие	**Описание**
OnChange	Наступает при изменении текста, причём не имеет значения, каким способом было произведено это изменение - вручную с клавиатуры или присвоением в программе.
OnKeyDown	Наступает при нажатии любой клавиши пользователем.
OnKeyPress	Наступает при нажатии клавиши символа.
OnKeyUp	Наступает при отпускании какой-либо клавиши пользователем.

2.9.3 Компонент Memo

Компонент **Memo** расположен на странице **Standard** в Палитре инструментов. Это простой многострочный текстовый редактор, который позволяет вводить текст с клавиатуры, загружать его из файла, редактировать и сохранять в файл.

Простота текстового редактора компонента Memo заключается в том, что он не обладает возможностями форматирования содержащегося в нём текста - формат текста (шрифт, выравнивание, цвет и т.д.) одинаков для всего текста и определён свойством **Font.**

Имеется доступ как ко всему тексту (свойство **Text**), так и к каждой строке текста отдельно. Строки в Memo являются объектами **Lines[i]** типа **TStrings**, где i - номер строки, отсчитываемый от нуля. Объект Lines[i] доступен и для чтения, и для записи. Учтите, что если окно редактирования изменяется в размерах при работе с приложением и свойство **WordWrap = True**, то индексы строк будут изменяться при переносах строк, так что в этих случаях индекс мало о чем говорит.

Текст в компоненте Memo можно редактировать не только с клавиатуры, но и программно:

Memo1.Lines[2] := 'Третья строка по счёту';

Для успешного присвоения текста определённой строке необходимо, чтобы эта строка физически существовала. То есть, данным способом можно только редактировать текст - для создания новых строк служат методы *Add* и *Insert.*

Memo1.Lines.Add('Последняя строка'); //Добавить строку 'Последняя строка' в конец текста

Memo1.Lines.Insert(3, 'Четвёртая строка'); //Вставить четвёртую строку 'Четвёртая строка'

Удалить строку можно с помощью метода *Delete*.

Memo1.Lines.Delete(3); //Удалить четвёртую строку

Основные свойства компонента Memo приведены в таблице:

Свойство	Описание
Text	Текст, находящийся в поле Memo в виде одной строки. Необходимо учитывать, что эта строка также будет включать в себя и непечатаемые символы конца строки #13 и символы переноса строки #10.
Lines	Массив строк, доступ к которым осуществляется по номеру. Нумерация начинается с нуля.
Lines.Count	Количество строк в поле.
Font	Шрифт, применяемый для отображения выводимого текста.
ParentFont	Признак наследования свойств шрифта от родительской формы.

Основные методы объектов Lines (объектов типа TStrings) приведены в таблице:

Метод	Описание
Clear;	Очистка поля.
Add(S: String);	Добавить в конец строку S.
Insert(Index: Integer; S: String);	Вставить строку S перед строкой с индексом Index.
Delete(Index: Integer);	Удалить строку с индексом Index.
LoadFromFile('filename);	Загрузить строки из файла filename.
SaveToFile('filename);	Сохранить строки в файл filename.

2.9.4 Компонент RichEdit

Компонент **RichEdit** расположен на странице **Win32** в Палитре инструментов. Это - многострочное окно редактирования текстов. В отличии от Memo, RichEdit поддерживает работу с текстом в формате RTF, позволяющем задавать цвет текста, гарнитуру и размер шрифта отдельному участку текста.

В окне редактирования имеется множество функций, которые свойственны для большинства редакторов. и предусмотрены стандартные сочетания горячих клавиш для работы с буфером обмена. Большинство свойств компонента RichEdit совпадают со свойствами компонента Memo.

При желании изменить атрибуты вновь вводимого фрагмента текста вы можете задать свойство **SelAttributes**. Это свойство типа **TTextAttributes**, которое в свою очередь имеет подсвойства: **Color (цвет)**, **Name (имя шрифта)**, **Size (размер)**, **Style (стиль)** и ряд других (удобно задавать эти атрибуты с помощью компонента **FontDialog**). Устанавливаемые атрибуты влияют на выделенный текст или, если ничего не выделено, то на атрибуты нового текста, вводимого начиная с текущей позиции курсора (позиция курсора определяется свойством **SelStart**).

В компоненте имеется также свойство **DefAttributes**, содержащее атрибуты по умолчанию. Эти атрибуты действуют до того момента, когда

изменяются атрибуты в свойстве SelAttributes. Но значения атрибутов в DefAttributes сохраняются и в любой момент эти значения могут быть методом *Assign* присвоены атрибутам свойства SelAttributes, чтобы вернуться к прежнему стилю. Свойство DefAttributes доступно только во время выполнения. Поэтому его атрибуты при необходимости можно задавать, например, в обработчике события **OnCreate.**

За выравнивание, отступы и т.д. в пределах текущего абзаца отвечает свойство Paragraph типа **TParaAttributes**. Этот тип в свою очередь имеет ряд свойств:

Свойство	Описание
Alignment	Определяет выравнивание текста. Может принимать значения taLeftJustify (влево), taCenter (по центру) или taRightJustify (вправо).
FirstIndent	Число пикселей отступа красной строки.
Numbering	Управляет вставкой маркеров, как в списках. Может принимать значения nsNone -- отсутствие маркеров, nsBullet -- маркеры ставятся.
LeftIndent	Отступ в пикселях от левого поля.
RightIndent	Отступ в пикселях от правого поля.
TabCount	Количество позиций табуляции.
Tab	Значения позиций табуляции в пикселях.

Значения подсвойств свойства Paragraph можно задавать только в процессе выполнения приложения, например, в событии создания формы или при нажатии какой-нибудь кнопки. Значения подсвойств свойства Paragraph относятся к тому абзацу, в котором находится курсор.

Свойства **TabCount** и **Tab** имеют смысл при вводе текста только при значении свойства компонента **WantTabs = True**. Это свойство разрешает пользователю вводить в текст символ табуляции. Если **WantTabs = False**, то нажатие пользователем клавиши табуляции просто переключит фокус на очередной компонент и символ табуляции в текст не введется.

Компонент StringGrid

Компонент **StringGrid** расположен на странице **Additional** в Палитре инструментов и предназначен для отображения данных в табличной форме.

В ячейках компонента StringGrid содержатся данные типа **String**. Таблица StringGrid состоит из зафиксированных ячеек-заголовков(их количество определено свойствами **FixedCols** и **FixedRows**), недоступных для редактирования вручную (их содержимое изменяется только программно), и обычных ячеек. Компонент StringGrid имеет возможность адресации каждой отдельной ячейки по номеру столбца и строки:

StringGrid.Cells[i, j], где i - номер столбца, j - номер строки, отсчитываются от 0.

Содержимое ячейки доступно как для чтения, так и для записи.

Выделенная ячейка таблицы имеет
номер столбца: *StringGrid.Col*
номер строки: *StringGrid.Row*
поэтому содержимое выделенной ячейки будет адресоваться как *StringGrid.Cells[StringGrid.Col, StringGrid.Row]*.

Основные свойства компонента StringGrid приведены в таблице:

Свойство	**Описание**
ColCount	Количество колонок в таблице.
RowCount	Количество строк в таблице.
DefaultColWidth	Ширина колонок в таблице.
DefaultRowHeight	Высота строк в таблице.
FixedCols	Количество зафиксированных колонок слева таблицы.
FixedRows	Количество зафиксированных колонок сверху таблицы.
Cells[i,j]	Двумерный массив ячеек таблицы, находящихся на пересечении столбца (col) и строки (row) соответствует элементу cells[col, row].
GridLineWidth	Ширина линии ограничивающиая ячейки таблицы.
Font	Шрифт отображения содержимого ячеек.

За многие свойства компонента StringGrid отвечает свойство **Options**. В Инспекторе Объектов Options - это раскрывающийся список, представляющий собой элементы данного множества. Если значение элемента равно True, то он присутствует в множестве, если False - то нет.

Свойство	**Описание**
goFixedVertLine	Наличие вертикальных разделительных линий между "фиксированными" ячейками
goFixedHorzLine	Наличие горизонтальных разделительных линий между "фиксированными" ячейками
goVertLine	Наличие вертикальных разделительных линий между "обычными" ячейками
goHorzLine	Наличие горизонтальных разделительных линий между "обычными" ячейками
goRangeSelect	Возможность выделить диапазон ячеек
goDrawFocusSelected	Закрашивание ячейки с фокусом ввода
goRowSizing	Возможность менять высоту строк мышкой
goColSizing	Возможность менять ширину столбцов мышкой
goRowMoving	Возможность менять номер строки, то есть перемещать её, мышкой
goColMoving	Возможность менять номер столбца, то есть перемещать его, мышкой
goEditing	Возможность редактировать содержимое ячейки с клавиатуры
goTabs	При значении True фокус смещается на следующую

	ячейку в таблице, False - на следующий компонент
goRowSelect	Выделяется вся строка с "фокусированной" ячейкой
goAlwaysShowEditor	При значении True содержимое ячейки при получении фокуса сразу доступно редактированию, False - сначала необходимо щёлкнуть по ней мышкой, либо нажать Enter или F2 (прим.: не действует при goRowSelect=True)
goThumbTracking	При значении True перемещение "бегунка" прокрутки мышкой вызывает немедленное перемещение ячеек, False - ячейки перемещаются только при отпускании "бегунка"

КОМПЛЕКТ ЗАДАНИЙ ДЛЯ ВЫПОЛНЕНИЯ ЛАБОРАТОРНЫХ РАБОТ

по дисциплине «Проектирование пользовательских интерфейсов»

Лабораторная работа №1

Тема: Начало работы в среде Delphi. Первый проект. Запуск программы. Настройка приложения.

Цель работы: научиться создавать проекты, изменять отлаживать и настраивать приложения в среде Delphi

Задание:

Создайте новый проект. Потренируйтесь добавлять и удалять визуальных компонентов на форму. Поработайте со свойствами объектов с помощью инспектора объектов. Добавьте обработчики событий. Запустите программу из интегрированной среды. Выполните пошаговую отладку с инспектированием значений переменных. Потренируйтесь в создании, добавлении и удалении модулей и форм. Выполните настройку опций проекта, а также настройку опций среды программирования. Сохраните проект.

Методические указания по выполнению:

Откройте новый проект можно, выбрав пункт меню File | New Project. Для работы можно создать форму .

Поместите на форму объект TMemo, а затем TEdit так, чтобы он наполовину перекрывал TMemo. Теперь выберите пункт меню Edit | Send to Back, что приведет к перемещению TEdit вглубь формы, за объект TMemo. Это называется изменением Z-порядка компонент. Поместите кнопку TButton в нижнюю часть формы. Теперь растяните Инспектор Объектов так, чтобы свойства Name и Caption были видны одновременно на экране. Теперь измените имя кнопки на Terminate. Текст, который Вы видите на поверхности кнопки - это значение свойства Caption. Напишите обработчик нажатия на кнопку, который будет закрывать приложение.

Для работы со свойствами нажмите клавишу <Shift> и щелкните на TMemo и затем на TListBox. Теперь оба объекта имеют по краям маленькие квадратики, показывающие, что объекты выбраны. Выбрав два или более объектов одновременно, можно выполнить большое число операций над ними, например, передвигать по форме. Затем попробуйте выбрать пункт меню Edit | Size и установить оба поля Ширину(Width) и Высоту(Height) в Grow to Largest. Теперь оба объекта стали одинакового размера.

Затем выберите пункт меню Edit | Align и поставьте в выравнивании по горизонтали значение Center .

Поскольку выбрано два компонента, то содержимое Инспектора Объектов изменится - он будет показывать только те поля, которые являются общими для объектов. Это означает то, что изменения в свойствах повлияют не на один, а на все выбранные объекты.

Рассмотрим изменение свойств объектов на примере свойства Color. Есть три способа изменить его значение в Инспекторе Объектов. Первый - просто напечатать имя цвета (clRed) или номер цвета. Второй путь - нажать на маленькую стрелку справа и выбрать цвет из списка. Третий путь - дважды

щелкнуть на поле ввода свойства Color. При этом появится диалог выбора цвета.

Свойство Font работает аналогично свойству Color.

Дважды щелкните на свойство Items объекта ListBox. Появится диалог, в котором Вы можете ввести строки для отображения в ListBox.

Напечатайте несколько слов, по одному на каждой строке, и нажмите кнопку OK. Текст отобразится в ListBox.

Сохранение программы:

Создать поддиректорию для программы. Лучше всего создать директорию, где будут храниться все Ваши программы и в ней создать поддиректорию для данной конкретной программы.

После создания поддиректории для хранения программы нужно выбрать пункт меню File | Save Project. Сохранить нужно будет два файла. Первый - модуль (unit), над которым Вы работали, второй - главный файл проекта, который "владеет" Вашей программой. Сохраните модуль под именем MAIN.PAS и проект под именем TIPS1.DPR.

Для разработки обработчика нажатия на кнопку перейдите на форму и дважды щелкните мышкой на объект TButton. Вы попадете в окно Редактора, в котором будет фрагмент кода:

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

Данный код был создан автоматически и будет выполняться всякий раз, когда во время работы программы пользователь нажмет кнопку Terminate. Определение класса в начале файла теперь включает ссылку на метод TerminateClick:

```
TForm1 = class(TForm)
Edit1: TEdit;
Memo1: TMemo;
Terminate: TButton;
procedure TerminateClick(Sender: TObject);
private
{ Private declarations }
public
{ Public declarations }
end;
```

Для того, чтобы приложение закрывалось при нажатии на кнопку необходимо написать код:

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

Справка по простейшим операциям в Delphi

1. Открыть новый проект:
    a. Автоматически при запуски Delphi
    b. На панели быстрых кнопок New|Application / В главном меню File|New|Application
2. Открыть существующий проект выбрать файл проекта .dpr:
    a. На панели быстрых кнопок Open Project / В главном меню File| Open Project
    b. Ctrl+F11
3. Сохранить проект:
    a. На панели быстрых кнопок Save All /В главном меню File| Save All
4. Вернуть на экран Форму:
    a. На панели быстрых кнопок View| Form /В главном меню File| View| Forms
    b. Shift+F12
5. Запустить программу:
    a. На панели быстрых кнопок щелкнуть на кнопку в виде зеленого треугольника
    b. В главном меню Run| Run
    c. F9
6. Остановить программу:
    a. щелкнуть на кнопку х закрытие окна формы
7. Аварийная остановка программы:
    a. В главном меню Run| Program Reset
    b. Ctrl+F2
8. Всплывающая подсказка
    a. Подвести курсор мыши к элементу интерфейса выждать паузу
9. Переключение с окна на форму или с формы на окно
    a. F12
10. Отладка программы:
    a. F4 – выполнение программы до строки , в которой находится курсор
    b. F5 – задана точка останова
    c. F7 – пошаговое выполнение программы с заходом в подпрограммы
    d. F8 – пошаговое выполнение программы без захода в подпрограммы
11. Отменить текущие изменения:
    a. Ctrl+Z
    b. В главном меню Edit| Undo
12. Редактирование кода модуля программы:
    a. Ctrl+C – копирование /Ctrl+X –вырезать /Ctrl+V –восстановить / Ctrl+Y –удалить строку /Delete – удалить фрагмент

Лабораторная работа №2

Тема: Знакомство с основами визуального программирования.

Цель работы: Получить навыки работы в инструментальной среде Delphi. Размещение компонентов на форме и задания их свойств

Задание:

Создание графического интерфейса объекта. Создать приложение, в котором после запуска на форме печатается некоторый текст, например, «Первый проект на языке Delphi».

1. Создать заготовку проекта, для этого свойству *Caption* (заголовок) формы Form1 присвоим значение "Первый проект'. Для этого активизировать форму *Form1* щелчком мыши (чтобы вокруг компонента появились черные квадратики – маркеры для изменения размера). В окне *Object Inspector (Инспектор объектов) → Properties (Свойств объекта)* выбрать свойство *Caption* и присвоить ему значение «Первый проект» (написать в ячейке справа от *Caption* Первый проект).

2. Вывести на форму текстовое сообщение. Это можно сделать различными способами:

- С помощью элемента управления *Label* (Метка)
- С помощью элемента управления *Edit* (Одна строка)
- С помощью элемента управления *Memo* (Несколько строк)

3. Разместить на форме метку *Label1* и присвоить свойству *Caption* (Надпись) значение выводимого текстового сообщения:

- Выбрать на Панели инструментов класс управляющих элементов *TLabel* и разместить экземпляр метки *Label1* на форме *Form1*
- Активизировать метку *Label1* щелчком мыши. В окне *Object Inspector (Инспектор объектов) → Properties (Свойства объекта)* выбрать свойство *Caption* (заголовок) и присвоить ему значение 'Первый проект на языке Delphi'.

4. Разместить на форме текстовое поле *Edit1* и присвоить свойству *Text* значение выводимого сообщения (сделать самостоятельно).

Событийные процедуры. Любой объект можно связать с набором процедур, исполняемых при наступлении определенных событий, такие процедуры называют *событийными процедурами*

Двойной щелчок мышью по объекту вызывает окно Программного кода (*Unit*) с пустой заготовкой событийной процедуры (метода). Если осуществить двойной щелчок по метке *Memo1*, то появится заготовка событийной процедуры *TForm1.Memo1Change (...);* (**эта надпись появляется автоматически!**)

```
procedure TForm1.Memo1Change(Sender: TObject);
begin
  ... end;
```

В теле процедуры (между служебными словами *begin* и *end*) должен быть записан метод, т.е. последовательность операторов, которые будут выполняться при наступлении событий (нажатии на *Memo1*)

5. Выбрать на Панели инструментов класс управляющих элементов *TMemo* и разместить экземпляр многострочного редактора *Memo1* на форме *Form1*. В свойстве *Lines* удалить существующую надпись *Memo1*. (щелкнув по многострочному полю *Memo1*, кликнуть напротив свойства *Lines* и удалить там надпись *Memo1*).

6. Активизировать *Memo1* двойным щелчком мыши, тем самым  вызвать окно Программного кода с пустой процедурой *TForm1.Memo1.Change (...);* (**эта надпись появляется автоматически**)

7. В теле процедуры записать

Memo1.Text:='Второй проект на языке Delphi';

т.е. при запуске программы, щелкнув по *Memo1*, появится надпись «Второй проект на языке Delphi»

Использование кнопок. Обычно, при работе с событийными процедурами используют Кнопки (*TButton*). Логично, что при нажатии на кнопку с определенным именем происходит какое-либо действие. Теперь необходимо решить предыдущую задачу, но с помощью Кнопки. В дальнейшем большинство задач будет решаться при обработке события *OnClick* – двойной щелчок по кнопке.

8. Выбрать на Панели инструментов класс управляющих элементов Button и поместить экземпляр кнопки *Button1* на форме *Form1*

9. Выбрать на Панели инструментов класс управляющих элементов *TLabel* и разместить экземпляр метки *Label2* на форме *Form1*

10. Активизировать кнопку *Button1*. В окне Свойств объекта выбрать свойство Caption и присвоить ему значение Моя кнопка (щелкнув по кнопке Button1, написать в колонке Properties Моя Кнопка в ячейке справа от Caption).

11. Активизировать *Button1* двойным щелчком мыши, тем самым вызвать окно Программного кода с пустой процедурой *TForm1.Button1Click(Sender: TObject);* (**эта надпись появляется автоматически**)

12. В теле процедуры записать

Label2.Caption:='Третий проект на языке Delphi';

т.е. при запуске программы, щелкнув по Моей кнопке, появится надпись на метке *Label2* «Третий проект на языке Delphi»

Выход из программы легко осуществлять, записывая в событийной процедуре оператор *Close;*

Установка цвета формы и параметров шрифта. Сделаем внешний вид более привлекательным и для этого изменим свойства объектов, определяющих их внешний вид.

13. Активизировать форму *Form1*. В окне свойства объекта выбрать свойство Color (цвет фона) и двойным щелчком открыть диалоговое окно с цветовой палитрой. Выбрать цвет, например желтый.

14. Активизировать метку *Label1*. В окне свойства объекта установить значения *Color* – синий, *Font* – *Arial*, жирный, 26 размер, травяной цвет символов.

15. Поэкспериментировать с объектами *Memo1, Edit1, Label2*

Для того, чтобы менять параметры шрифта с помощью кнопки при запуске программы необходимо в событийной процедуре указать, например *Label2.Font.Size:=30; Label2.Font.Color:=clWhite*; Т.е. при нажатии на кнопку размер шрифта метки 2 станет равным 30, а цвет - белым.

Для изменения начертания шрифта пишем

- *Label2.Font.Style:=fsbold; {полужирный},*
- *Label2.Font.Style:=fsitalic; {курсив},*
- *Label2.Font.Style:=fsunderline; {полужирный}.*

Если необходимо поменять шрифт пишем *Label2.Font.Name:='Arial'; {Arial}* или *Label2.Font.Name:='Times New Roman'; {Times New Roman}*

Задание для самостоятельной работы:

1. Создать проект «Вывод сообщений», в котором на форму выводится текстовое сообщение «Первое задание выполнено!» с помощью метки, одной строки и многострочного редактора, а выход из программы осуществляется щелчком по кнопке Выход. Придумать графический интерфейс программе.

2. Создать проект «Вывод сообщений в строку», в котором два различных варианта текста выводятся в строку *Edit1* по щелчку по двум кнопкам. Предусмотреть возможность выхода из программы по третьей кнопке.

3. Создать проект «Печать на форме», в котором в нескольких метках при нажатии на кнопки тексты будут со следующими параметрами:

- «Times New Roman, 18, курсив, красный»
- «Courier New, 40, полужирный, зеленый»
- «Arial, 22, подчеркнутый, синий»
- «Webdings, 12, подчеркнутый курсив, желтый»
- «Monotype Corsiva, 32, полужирный»

Лабораторная работа №3

Тема: расположение объектов и управляющих элементов на форме.

Цель работы: приобрести навыки работы с компонентами форм.

Задание:

Расположение объектов на форме. Местоположение объекта, которое он будет занимать на форме после запуска приложения на выполнение, можно легко изменить.

Для этого достаточно перетащить объект с помощью мыши в любое место экрана. Если необходимо установить точные значения местоположения и размеров объектов, то это можно сделать, установив значения свойств *Left* (расстояние по горизонтали от левого верхнего угла монитора до верхнего левого угла формы), *Top* (расстояние по вертикали от левого верхнего угла монитора до верхнего левого угла формы), *Width* (ширина формы), *Height* (высота формы). Для этого необходимо активизировать, например Edit. В окне Свойства объекта последовательно присвоить свойствам *Left, Top, Width, Height* значения 32, 104, 225, 30.

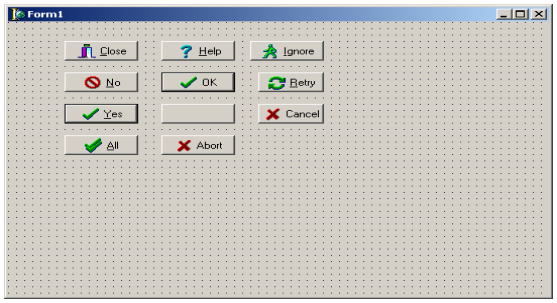
Расположение управляющих элементов на форме. Расположение на форме и размеры управляющих элементов можно легко изменять с помощью мыши. Для этого необходимо активизировать объект щелчком мыши (он будет выделен восемью квадратиками - маркерами) и перетащить его на новое место или изменить размер элемента.

Точное месторасположение и размеры выделенных элементов управления отображаются двумя парами чисел: *Left, Top, Width* и *Height.*

1. Активизировать метку *Label1*. В окне свойства объекта последовательно присвоить свойствам *Left, Top, Width* и *Height* значения 30, 0, 100, 100.
2. Активизировать метку *Label2*. В окне свойства объекта последовательно присвоить свойствам *Left, Top, Width* и *Height* значения 70, 0, 450, 100
3. Активизировать метку *Label3*. В окне свойства объекта последовательно присвоить свойствам *Left, Top, Width* и *Height* значения 0, 330, 100, 30.

Задание для самостоятельной работы:

1. Создать проект «Перемещение кнопки по форме», в котором, например, в центре, располагаются кнопки и по щелчку перемещаются в заданное место на форме, например по углам формы
2. Создать проект «Перемещение формы на экране», в котором будет изменяться местоположение формы на экране монитора с помощью четырех командных кнопок. Кнопки расположить в углах формы. После щелчка по кнопке, форма должна переместиться в соответствующую часть экрана. Перемещение формы должно сопровождаться комментариями, выводимыми с помощью надписи в центре формы.



Кнопка с изображением BitBtn

Одной из разновидностью кнопок служит *BitBtn* – кнопка с изображением. Она находится во вкладке *Additional*. Свойство *Kind* для такой кнопки определит одну из 11 стандартных разновидностей кнопки. Если у кнопки был изменен рисунок (свойство *Glyph*), *Delphi* автоматически присваивает свойству *Kind* кнопки значение *bkCustom*.

Лабораторная работа №4

Тема:Арифметические операции в ООП

Цель работы: овладеть навыками решениями математических задач.

Задание:

Для решения арифметических задач необходимо описывать все используемые переменные:

Var a, b, c, dude: integer; //целые числа

x, y, z, max: real; // вещественные числа

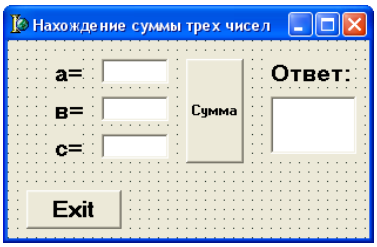
Для решения примеров мы будем использовать функции *IntToStr (<переменная>)* (перевод числа из целого в строчный тип) и *StrToInt (<переменная>)* (перевод из строкового в целый тип). Это необходимо в связи с тем, что при вводе числа в строку *Edit* оно становится строкового типа, а со строками нельзя выполнить арифметических операций. Для вычисления необходимо преобразовывать тип из строкового в целый, а для вывода результата нужно обратно переводить из целого в строчный тип.

Однако если ответ задачи лежит в переменной вещественного типа, то для вывода его в строчном поле классов *TEdit* или *TMemo* необходимо использовать функцию *FloatToStr(<переменная>).*

Решим простейший пример: найти сумму трех чисел.

1. Описываем после служебного слова *VAR* несколько переменных целого типа для удобства решения: *Var* a, b, c, s: integer;
2. На форму заводим 3 метки: *Label1*, *Label2*, *Label3*. У каждой из них в свойстве Caption написать соответственно a=, в=, c=.
3. Наносим однострочные элементы управления *Edit1*, *Edit2*, *Edit3*. В них будем вводить числа. Для этого в свойстве Text нужно удалить надписи.
4. Заводим кнопку, которую называем Сумма
5. Наносим элемент управления *Memo1* и удаляем все надписи в многострочном редакторе. Для этого в окне Свойства объекта выбираем свойство Lines и удаляем все надписи в нем.

Примерно ваша форма должна выглядеть так:



6. Активируем двойным щелчком мыши кнопку. Появляется процедура *TForm1.Button1Click(Sender: TObject);*
7. В теле процедуры присваиваем заведенным переменным значения строк следующим образом *a:=StrToInt(Edit1.text);* (аналогично для b и c)
8. Вычисляем сумму *s:=a+b+c;*
9. Выводим результат в *Memo1*. Для этого записываем *Memo1.Text:=IntToStr(s);*

Продумать свое графическое оформление этой задачи.

Задания для самостоятельной работы:

1. Вычислить произведение четырех чисел. Графическое оформление задачи самостоятельное.
2. Решить пример a=b+c/m*k. Графическое оформление задачи самостоятельное.

Алгоритм решения задачи имеет 3 основные части:

1. ввод данных

Например:

a:=StrToInt(Memo1.text); // занести в переменную a содержимое многострочного редактора Memo1, переведенное из строкового в числовую форму для выполнения математических действий.

b:=StrToInt(Edit1.text); // занести в переменную b содержимое однострочного редактора Edit1, переведенное из строкового в числовую форму для выполнения математических действий.

2. проведение определенных действий, требуемых в условии задачи.
3. вывод результата или сообщения о проведенном анализе данных.

Вывод результата: *c:=a+b;* // ответ c

Memo1.text:=IntToStr(c); // вывод результата решения примера в многострочный редактор Memo1 путем перевода ответа из числовой в строковую форму.

Вывод сообщения: Edit1.text:=’Задача решена успешно!’; // вывод сообщения о проведенном анализе данных в многострочный редактор Edit1. Перевод сообщения из числовой в строковую форму НЕ ОСУЩЕСТВЛЯЕТСЯ!

Стандартные арифметические функции

abs(x)	Модуль x
cos(x)	Косинус x
exp(x)	Экспонента
frac(x)	Дробная часть от x
int(x)	Целая часть от x
random	псевдослучайное число [0,1)
random(x)	псевдослучайное число [0,x)
sin(x)	Синус x
sqr(x)	квадрат x
sqrt(x)	корень квадратный

Задания для самостоятельной работы:

1.

$$y := \frac{a^2 + \pi}{\sqrt{\left(b \cdot \frac{d}{r}\right)}}$$

2.

$$x := \cos(a) + \frac{\sin(b)}{\left|\frac{c}{d}\right|}$$

3. Дано число. Вывести на экран его целую и дробную части.

4. Даны числа a, b, c. Возвести в третью степень a, в квадрат произведения b на c и найти модуль суммы этих чисел.

Лабораторная работа №5

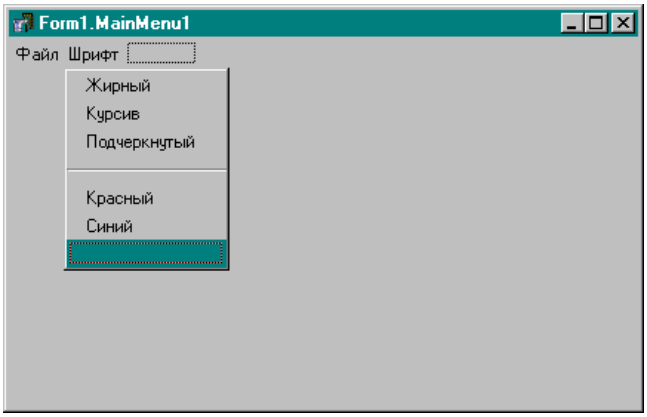
Тема: Организация меню в программе

Цель работы: получения навыков работы с компонентой *MainMenu*

Задание:

MainMenu - главное меню формы. Компонент класса *TMainMenu* определяет главное меню формы (программы).

1. Установите компонент на форму



2. Создайте пункты меню. Для этого следует дважды щелкнуть по компоненту мышью.

Меню обладает свойством *Caption*, в котором можно задать его имя. Каждый пункт меню может раскрываться в подменю или являться конечной командой. Для создания подпунктов:

3. Щелкните мышью ниже пункта меню и введите имя первого пункта подменю. В названиях пунктов можно указать символ амперсанда (&) перед тем символом, который определит клавишу быстрого вызова. Для вставки разделительной черты, определяющей пункты меню, нужно ввести в качестве имени очередного пункта меню дефис (-).

В Delphi имеется возможность связывать с пунктами меню небольшие изображения. Эти изображения можно задать либо свойством *BitMap*, либо свойством *ImageIndex*. Изображение (если оно есть) появляется слева от пункта меню.

Задание для самостоятельной работы: создать проект для работы с компонентом класса *TMainMenu* . Все пункты меню должны быть рабочими.

Лабораторная работа №6

Тема: использование визуальных компонентов Delphi

Цель работы*:* Изучение основных визуальных компонентов по страницам. Приобретение навыков использования визуальных компонентов из страниц: Standard, Additional, Win32, Dialogs, Samples.

Задание:

Разработать программу с интерфейсом в главной строке меню: РАЗДЕЛЫ, ВЫХОД

Пункт меню РАЗДЕЛЫ содержит четыре пункта:

- Работа с изображениями;
- Текстовый редактор;
- Настройки;
- Построение графиков.

При выборе пункта РАБОТА С ИЗОБРАЖЕНИЯМИ появляется диалоговое окно выбора графического файла стандартного вида. В случае

выбора файла появляется форма с выбранным изображением, в строке состояния отображается имя открытого файла и координаты мыши.

Рисунок 6.1 – Текстовый редактор

При выборе пункта ТЕКСТОВЫЙ РЕДАКТОР выводится форма, содержащая многострочное окно редактирования Memo и текстовый редактор RichEdit.

С визуальными компонентами Memo и RichEdit необходимо связать всплывающую строку подсказки и всплывающее меню для организации работы с текстом (копировать, перемещать, изменять шрифт). Примерный вид формы приведен на рисунке 2.3.

Форма НАСТРОЙКИ представляет собой многостраничное окно. Окно имен три страницы:

- Ввод чисел и дат (рисунок 6.2);

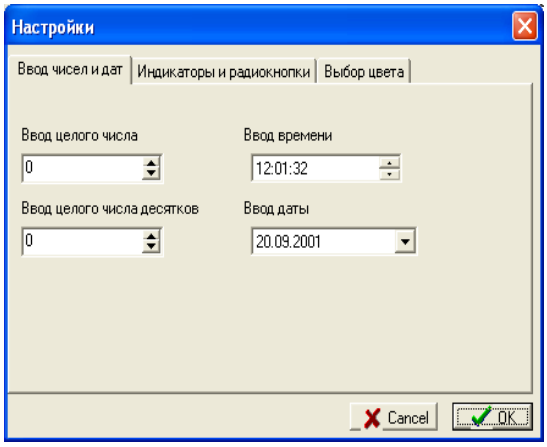


Рисунок 6.2 – Страница «Ввод чисел

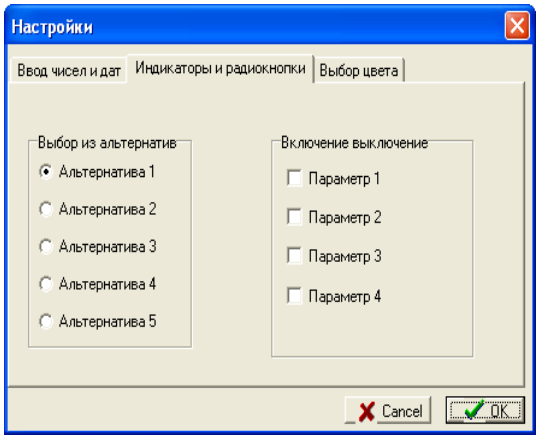


Рисунок 6.3 – Страница «Индикаторы

- Индикаторы и радиокнопки (рисунок 6.3);
- Выбор цвета (рисунок 6.4).

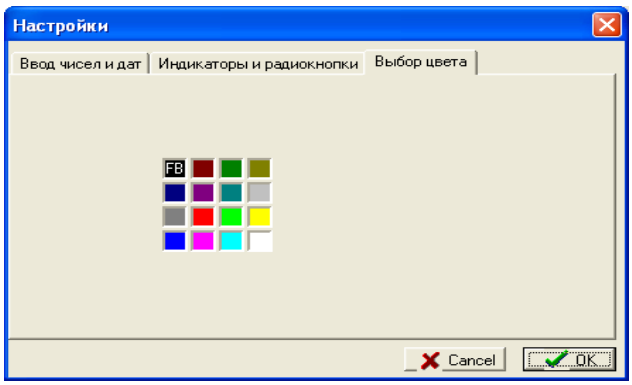


Рисунок 6.4 – Страница «Выбор цвета»

При нажатии кнопки ОК необходимо:

1. запросить, необходимо ли сохранить настройки (окно сообщения Windows)
2. организовать сохранение установленных настроек в текстовом файле. Для выбора имени файла вызывается диалоговое окно стандартного вида.

При выборе пункта ПОСТОЕНИЕ ГРАФИКОВ выводится форма с графиками функций y=sin(x) и y=cos(x) (рисунок 2.7). При изменении размеров окна график должен масштабироваться.

Лабораторная работа №7

Тема: Работа со строками в объектно-ориентированном программировании. Оператор цикла с параметрами. Оператор варианта

Цель работы: овладеть навыками работы со строковыми переменными, оператором выбора, операторами цикла

Оператор варианта

Оператор варианта *Case* является обобщением оператора *if* и позволяет сделать выбор из произвольного числа вариантов.

Case <выражение-селектор> of
<список 1>: <оператор 1>;
...
<список N>: <оператор N>
else <оператор>
end;

Селектор может иметь любой скалярный тип, кроме вещественного. Использование строкового типа в качестве селектора запрещено. Список констант выбора может состоять из произвольного количества значений или диапазонов, отделенных друг от друга запятыми.

Тип констант в любом случае должен совпадать с типом селектора.

1. селектор интервального типа.

Case i of
1..10: Memo1.Lines.Add ('число', I:4, 'в диапазоне 1-10');
11..20: Memo1.Lines.Add ('число', I:4, 'в диапазоне 11-20');

21..30: Memo1.Lines.Add (‘число’, I:4, ‘в диапазоне 21-30’)
else Memo1.Lines.Add (‘число’, I:4, ‘вне пределов контроля’) end;

2. селектор целочисленного типа.

```
Case I of
 1: z:=i+10;
 2: z:=i+100;
end;
```

Найдите ошибку в примере: Задача:Написать программу, которая по введенному числу 1,2,3 выводит его название для события по щелчку кнопки.

```
 Var a: integer;
Begin
 a:=(Edit1.text); // ввод числа
 case a of
 1: Memo1.text:=IntToStr(‘один’);
 2: Memo1.text:=IntToStr(‘два’);
 3: Memo1.text:=IntToStr(‘три’);
 else Memo1.text:=IntToStr(‘Введите число 1, 2 или 3’);
end;
End;
```

Задания для самостоятельной работы:

1. Составить программу, которая по заданному числу (1-12) выводит название соответствующего месяца.
2. Написать программу, которая по введенному номеру времени года выдавала соответствующие этому времени года месяцы и число дней в каждом месяце.
3. Составить программу, которая для любого натурального числа 1-1000 печатает количество цифр в записи этого числа.
4. Для целого числа k от 1 до 99 напечатать фразу «я прочел k книг», согласовав окончание слова «книга» с числом k.

Работа со строками в объектно-ориентированном программировании. Оператор цикла с параметрами

Для работы со строками необходимо использовать тип данных *string* (*var s: string*). Отличительной чертой этого типа данных является то, что мы можем обратиться как к строке целиком, так и посимвольно, то есть к каждой букве в отдельности. Для этого нам необходимо использовать оператор цикла

При работе со строкой запишем оператор цикла следующим образом: *for i:=1 to length(s) do (*обращение к символам от первого до последнего в строке s). Кроме того, используются следующие процедуры и функции:

1. Функция length (s) определяет длину строки. Результат - целое число 0..255
2. Процедура *Delete (s,pos,n)* - удаление части строки. Удаляет из строки s n-символов начиная с символа № pos. Пример: *s:='рогатка'; Delete(s,5,3)* => рога

3. Процедура *Insert (s1,s2,pos)*. Вставка части строки. S1-что, S2-куда, Pos-с какой позиции Пример: *S1:='свет'; Insert(s1,'о',2)* => 'совет'

4. Функция *Copy (s,pos,n)* - возвращает часть строки s длиной n, начиная с позиции pos. Пример*: S:='пароход'; t:=copy(s,1,3)* => 'пар'

5. Сцепление строк - функция *Concat (s1,s2,s3,…,sn)* Пример: *Concat ('к','о','т')* => 'кот'

6. Функция *Pos (s1,s2)* - поиск одной строки в другой. Возвращает номер символа, начиная с которого строка s1 является частью s2. Пример*: pos ('cd','abcdf')* => 3

Пример: подсчитать количество вхождений символа «*а*» в строку. Используем компоненты классов *TEdit* и *TMemo* для ввода строки и вывода результата.

```
Var s: string;
   i, k: integer;
Procedure TForm1.Button1Click(Sender: TObject);
Begin
   s:=edit1.text; //ввод строки
  For i:=1 to length(s) do //идем по строке
    If s[i]=’a’ then k:=k+1; {если i-ый символ a, тогда увеличивай счетчик k}
  Memo1.text:=IntToStr(k) //вывод результата
End;
```

Задания для самостоятельной работы:

1. Дана строка s: Найти количество вхождений букв a,c,d в строку.
2. Найти количество цифр в строке.
3. Сцепить несколько строк в одну.
4. Из данной строки выбрать цифры и сформировать из них новую строку.
5. Найти количество слов, начинающихся на букву с.
6. Подсчитать количество слов в строке.
7. Определить начинается и заканчивается ли слово одной буквой.
8. Удалить каждую четную букву в строке.
9. Проверить одинаковое ли число открытых и закрытых скобок в строке.
10. По введенным: фамилии, имени, отчеству выводить информацию о том, является ли пользователь автором программы, его теской или являются ли теской отец автора и пользователя программы.

Лабораторная работа №8

Тема: Работа с таблицами

Цель работы: овладеть навыками работы с таблицами

TStringGrid – текстовая таблица.

Компонент *TStringGrid* предназначен для создания таблиц, в ячейках которых располагаются произвольные текстовые строки. Таблица делится на две части – фиксированную и рабочую. Фиксированная часть служит для показа заголовков колонок и рядов, а так же для ручного управления их размерами. Обычно фиксированная часть занимает левую колонку и верхний

ряд таблицы, однако с помощью свойств *FixedCols* и *FixedRows* можно задать другое количество фиксированных колонок и рядов.

Рабочая часть – это остальная часть таблицы. Она может содержать произвольное количество колонок и рядов, более того, эти величины могут изменяться программно. Если рабочая часть таблицы не помещается целиком на экране, то автоматически появляется полоса прокрутки.

Свойства компонента:

Свойство	*Описание*
BorderStyle	Определяет рамку компонента: *bsNone* – нет рамки, *bsSingle* – рамка толщиной 1 пиксел
Cells[*col,row*]	Определяет содержимое ячейки с табличными координатами *(col,row)*
Col	Содержит номер колонки с ячейкой, имеющей фокус ввода
ColCount	Содержит количество колонок таблицы
ColWidths	Содержит ширину колонки с индексом *Index*
EditorMode	Разрешает/запрещает редактирование ячеек. Игнорируется, если свойство Options включает значение *goAlwayseShowEditor* или не включает значение *goEditing*
FixedColor	Определяет цвет фиксированной зоны
FixedCols	Определяет количество колонок фиксированной зоны
FixedRows	Определяет количество рядов фиксированной зоны
GridHeight	Содержит значение высоты таблицы
GridLineWidth	Определяет ширину линий, расчерчивающих таблицу
GridWidth	Содержит значение ширины таблицы
Row	Содержит номер ряда ячейки, имеющий фокус ввода
RowCount	Содержит количество рядов таблицы
RowHeights	Содержит значение высоты ряда с индексом *Index*
Rows	Содержит все текстовые строки ряда с индексом *Index*
ScrollBars	Определяет полосы прокрутки: *ssNone* – нет полос; *ssHorizontal* – в таблицу вставляется горизонтальная полоса; *ssVertical* – вставляется вертикальная полоса; *ssBoth* – вставляются обе полосы.

Для обращения к ячейке [0,0] нужно использовать свойство *Cells*:

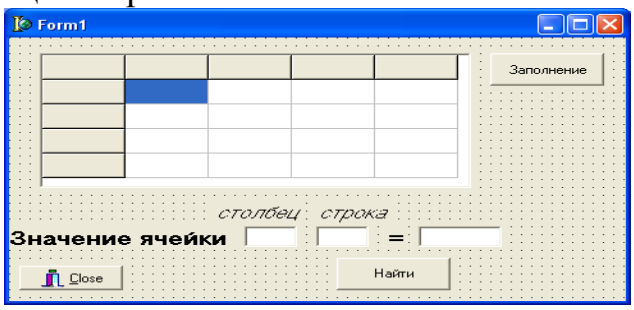
StringGrid1.Cells[0,0]:=IntToStr(5)

Задание:

1. Проиллюстрируйте возможности рассмотренного компонента, составив таблицу следующего вида:

	Дисциплина	ФИО педагога
1.	Математика	Иванов И.И.
2.	Русский язык	Петров П.П.
3.	…	…

2. Составить таблицу 4x4, состоящую из целых чисел. Программа должна выводить число, лежащее в ячейке с введенным с клавиатуры номером столбца и строки.



3. Составить таблицу, выводящую расписание занятий в школе.

Массивы

Массив – упорядоченный набор однотипных значений – компонент массива. Тип компонент называется базовым типом массива.

В *Delphi* массив рассматривается как переменная структурированного типа. Массиву присваивается имя, посредством которого можно ссылаться на него, как на единое целое, так и на любую из его компонент.

Описание массивов

VAR <имя переменной>:ARRAY [<type1, type2,…>] OF <type>;

[<type1, type2,…>] – типы индексов

<type> - базовый тип

Количество типов индексов определяет размерность массива. Тип индексов может быть любым, кроме *Real.* Базовым типом может быть любой тип данных.

Примеры описания массивов

Var a: array[1..10] of integer; - последовательность целых чисел

Var t: array[1..2,1..2] of float; - таблица 2x2.

Var a1: array ['A'..'Z'] of word;

Обращение к элементам массива

a[1] – первый элемент из массива *a*.

t[1,2] – элемент из первой строки второго столбца.

Индексы можно задавать либо конкретными значениями, либо выражениями.

i:=1; a[i] - первый элемент из массива *a*; *a[i+1]* – второй элемент массива *a*.

В динамических массивах не указывается размерность в описании:

Например, *var a: array of integer;*

Размерность устанавливается в теле программы: *SetLength(a, 10);* - длинна массива a – 10 элементов.

По массиву проход осуществляется от первого до последнего элемента:

for i := Low(a) to High(a) do где i: integer;

Задания для самостоятельной работы:

1. Заполнить линейный массив произвольными числами и вывести его на экран.

2. Заполнить линейный массив числами, введенными с клавиатуры, вывести этот массив, увеличить все элементы на 5 и снова вывести получившийся массив на экран.

3. В линейном массиве найти элементы массива равные 10 и вывести их индексы.

4. В линейном массиве вывести элементы с четными индексами.

5. В двумерном массиве найти количество четных элементов.

6. В двумерном массиве найти сумму элементов главной диагонали.

7. В двумерном массиве подсчитать произведение наибольших элементов каждой строки.

8. В динамическом массиве вывести на экран элементы равные своему индексу.

9. В динамическом массиве найти элементы, удовлетворяющие условию сумма индексов меньше значения элемента.

10. Вывести массив следующего вида. Размерность произвольная.

11	1	1		1	0	0		1	2	2	2
10	0	1		0	2	0		3	1	2	2
10	…	1	и	0	0	…	и	3	3	1	2
11	1	1		0	0	N		3	3	3	1

Лабораторная работа №9-10

Тема: Delphi: Создание графических примитивов в среде Delphi**.**

Цель работы: приобретение навыков создания приложения с использованием графических примитивов.

Теория. Многие компоненты в Delphi имеют свойство Canvas (канва, холст), представляющие собой область компонента, на которой можно рисовать или отображать готовые изображения. Свойство Canvas используется для рисования пером (свойство Pen) и (кистью (свойство Brush)) .

Для описания цвета используется **тип TColor**.

Карандаш используется для вычерчивания точек, линий, контуров геометрических фигур: прямоугольников, окружностей, эллипсов, дуг и др. Вид линии, которую оставляет карандаш на поверхности холста, определяют свойства объекта Tpen (Color-Цвет линии, Width- Толщина линии, Style-Вид линии, Mode-Режим отображения)

Например, Form1.Canvas.Pen.Width:=2; устанавливает толщину линии 2 пиксела.

Form1.**Canvas.Pen.Color:=clRed**; - устанавливает красный цвет линии.

Style определяет стиль линии, который можно задать именованной константой (psSolid-Сплошная линия, psDash-Пунктирная линия, длинные штрихи; psDot-Пунктирная линия, короткие штрихи; psDashDot-Пунктирная линия, чередование длинного и короткого штрихов; psDashDotDot- Пунктирная линия, чередование одного длинного и двух коротких штрихов; psClear-Линия не отображается. Например:Canvas.Pen.Style:=psdot;

Кисть используется методами, (свойство Brush)) обеспечивающими вычерчивание замкнутых областей и обладает двумя свойствами: Color- Цвет закрашивания замкнутой области, Style-Стиль заполнения области. Константы, позволяющие задать стиль заполнения области: (bsSolid-Сплошная заливка; bsClear-Область не закрашивается; bsHorizontal-Горизонтальная штриховка; bsVertical-Вертикальная штриховка; bsFdiagonal-Диагональная штриховка с наклоном линий вперед; bsBDiagonal-Диагональная штриховка с наклоном линий назад; bsCross-Горизонтально-вертикальная штриховка в клетку; bsDiagCross-Диагональная штриховка, в клетку. Например:Canvas.brush.style:=bsVertical;

Для вывода текста на поверхности графического объекта используется **метод TextOut**.

Например, Form1.**Canvas.TextOut(x,y,Текст),** где x,y - координаты точки графической поверхности, от которой выполняется вывод текста. Шрифт, который используется для вывода текста, определяется значением свойства Font.

Методы вычерчивания графических примитивов: **Линия**: Компонент **Canvas.LineTo(x,y);** Начальную точку линии можно задать, переместив карандаш в нужную точку графической поверхности при помощи метода **MoveTo(x,y).**

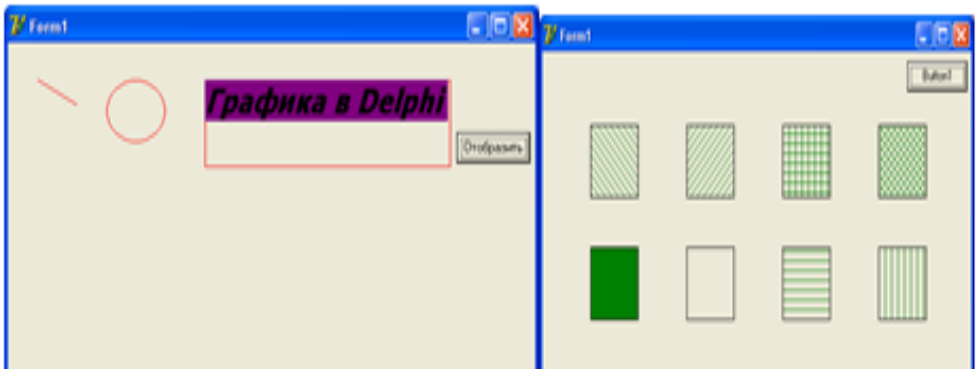
Окружность, эллипс можно начертить, используя **метод Ellipse**. Вызов метода выглядит: Компонент.Canvas.Ellipse(x1,y1, x2,y2), где x1,y1, x2,y2 – координаты прямоугольника, внутри которого вычерчивается эллипс или, если прямоугольник является квадратом, окружность.

Дуга . Вычерчивание дуги выполняет **метод Arc,** инструкция вызова которого имеет вид: Компонент.Canvas.Arc(x1,y1, x2,y2, x3,y3, x4,y4), где x1,y1, x2,y2, - параметры, определяющие эллипс, частью которого является вычерчиваемая дуга; x3,y3 – параметры, определяющие начальную точку дуги;x4,y4 - конечную точку дуги.

Прямоугольник вычерчивается методом **Rectangle,** вызов которого имеет вид: **Компонент Canvas. Rectangle(x1,y1,x2,y2), где x1,y1,x2,y2** – координаты левого верхнего и правого нижнего углов прямоугольника. Прямоугольник со скругленными углами вычерчивается методом**: RoungRec(x1,y1,x2,y2)** .

Задание:

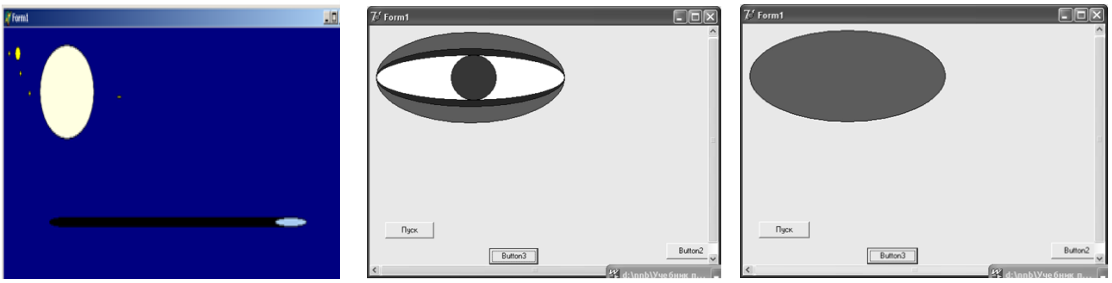
Задание 1. Начертить графические примитивы: линию, прямоугольник, эллипс



Задания 2. Создание приложений с мультипликацией в среде Delphi

Под мультипликацией понимается движущийся и меняющийся рисунок. Обеспечить перемещение рисунка можно следующим образом: вывести рисунок на экран, затем через некоторое время стереть его или нарисовать цветом фона, а затем снова вывести этот же рисунок, но уже на некотором расстоянии от его первоначального расположения. Впечатление равномерного движения достигается путем подбора времени между выводом и удалением рисунка. Для задания некоторого интервала времени можно использовать невизуальный компонент Timer (таймер) на вкладке System.

Вариант 1. Отобразить на фоне звездного неба летящую летающую тарелку.



Вариант 2. Изобразить на форме космический глаз, периодически открывающийся и закрывающийся.

Лабораторная работа № 11

Тема: Создание псевдонима база данных. Работа с полями набора данных.

Цель работы: Приобретение практических навыков создания программных приложений с использованием баз данных. Приобретение практических навыков создания псевдонима для базы данных. Создание простейшей базы данных и приложения для работы с таблицами этой базы данных.

Задание:

1. Разработать базу данных и заполнить данными.
2. Создать псевдоним для базы данных.
3. Разработать интерфейс приложения

4. Создать приложение для просмотра и редактирования таблиц базы данных.

5. Оформить отчет и сдать работу преподавателю.

Содержание отчета:

1. Номер, тема, цель работы, задание к работе, вариант задания.
2. Программа (текст рабочего модуля)
3. Результаты работы программы.
4. Выводы о проделанной работе

Постановка задачи

Требуется создать в среде Delphi приложение, предназначенное для учета поступающих на склад товаров. База данных состоит из двух таблиц: справочника «Товары» и операционной таблицы «Приход товара». Таблицы необходимо создать и хранить в формате Access. Отношение между таблицами «один-ко-многим», т.е. одному товару в таблице «Товары» может соответствовать более одной записи в таблице «Приход товара».

Рабочая структура таблиц:

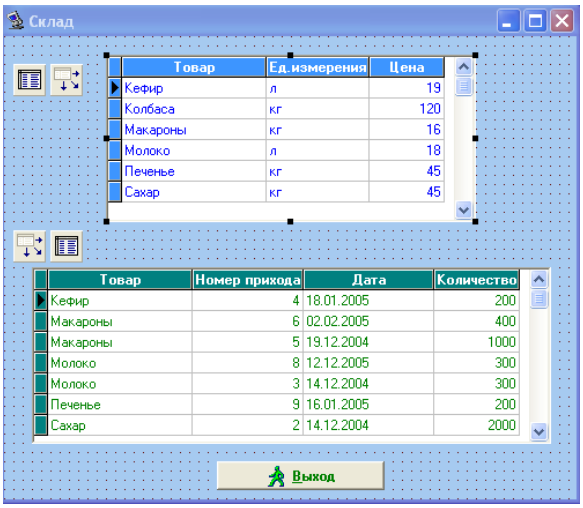
Таблица «Товары»

Смысл	Название	Тип	Длина
Название товара	Товар	строка	20
Единица измерения	ЕдИзм	строка	10
Цена за ед. измер.	Цена	целочисленный	

Таблица «Приход товара»

Смысл	Название	Тип	Длина
№ прихода	НомПрих	Автоинкремент	
Название товара	Товар	строка	20
Дата прихода	ДатаПрих	дата	
Количество	Колич	целочисленный	

Для таблицы «Товары» первичный ключ построен по полю Товар. Для таблицы «Приход товара» построен по полю НомПрих. Для реализации ссылочной целостности (с таблицей «Товары») необходимо построить внешний ключ по полю Товар.



Создать базу «Учет товаров» данных и занести в таблицы по несколько записей.

Создать псевдоним базы данных – Baza.

Создать новую форму приложения.

На форме разместить следующие компоненты:

-2 компонента Table;

-2 компонента DataSourse;

-2 компонента DBGrig;

Связать не визуальные компоненты с базой данных. Связать визуальные компоненты с не визуальными для отображения содержимого таблиц. Отформатировать таблицы (DBGrig) с помощью редактора Columns Editor…

Для компонентов TTable создать объекты TFields с помощью редактора Fields editor…

Запустить приложение и внести в таблицу «Товары» новые товары.

Insert – вставить новую запись Ctrl/Del – удалить запись

Для всех визуальных компонентов создать всплывающие подсказки.

Лабораторная работа № 12

Тема: Добавление, удаление, редактирование записей в базе данных.

Цель работы: Приобретение практических навыков создания программных приложений позволяющих выполнять различные действия с наборами данных: добавление, удаление, редактирование.

Задание:

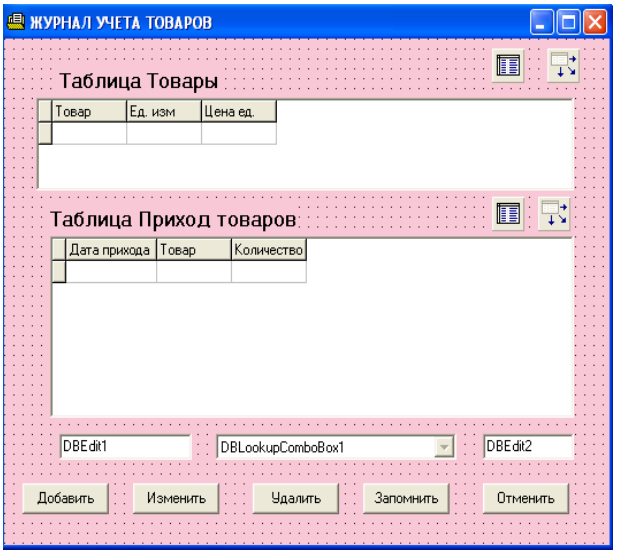
6. Разработать интерфейс приложения
7. Создать приложение для обработки базы данных.
8. Оформить отчет и сдать работу преподавателю.

Содержание отчета:

5. Номер, тема, цель работы, задание к работе, вариант задания.
6. Программа (текст рабочего модуля)
7. Результаты работы программы.
8. Выводы о проделанной работе

Постановка задачи

Требуется создать в среде Delphi приложение, предназначенное для выполнения следующих операций с наборами данных: добавление новых записей; удаление записей; изменение записей.



При запуске приложения на экран должно выводиться окно с таблицами «Товары» (TDBGrid1) и «Приход товаров» (TDBGrid2), поля ввода и редактирования для даты прихода (TDBEdit1) и количества товара (TDBEdit2), и поле выбора товара (TDBLookupComboBox1). Поле выбора товара должно обеспечивать выбор только тех товаров, которые имеются в таблице «Товары». Кроме этого на форме должны быть следующие кнопки: «Добавить», «Изменить», «Удалить», «Запомнить», «Отменить». При удалении записи на экран должно выдаваться окно диалога для подтверждения или отмены удаления записи.

Для компонента TDBLookupComboBox1 установить следующие значения свойств:

- DataSource > DataSourse2
- DataField >Tovar
- ListSource > DataSource1
- ListField >Tovar
- KeyField > Tovar

Для компонента TDBEdit1 установить следующие значения свойств:

- DataSource > DataSourse2
- DataField >DatPrix

Для компонента TDBEdit2 установить следующие значения свойств:

- DataSource > DataSourse2
- DataField >Kolvo

Для того чтобы набор данных нельзя было переводить в состояние добавления и изменения данных, а также удалять записи непосредственно из компонентов TDBGrid2 и TDBGrid1, установить свойство ReadOnly в значение True.

Для всех визуальных компонентов создать всплывающие подсказки.

```
procedure TForm1.Button1Click(Sender: TObject);
begin
 if Table2.State=dsBrowse then Table2.Insert;
end;
procedure TForm1.Button2Click(Sender: TObject);
begin
 if Table2.State=dsBrowse then Table2.Edit;
```

```
end;
procedure TForm1.Button3Click(Sender: TObject);
begin
 if Table2.State=dsBrowse then
   if MessageDlg('Удалить',mtConfirmation,[mbYes,mbNo],0)=mrYes then
Table2.Delete;
end;
procedure TForm1.Button4Click(Sender: TObject);
begin
 if Table2.State in [dsInsert,dsEdit] then Table2.Post;
end;
procedure TForm1.Button5Click(Sender: TObject);
begin
  if Table2.State in [dsInsert,dsEdit] then Table2.Cancel;
end;
```

Лабораторная работа № 13

Тема: Поиск записей в базе данных.

Цель работы: Приобретение практических навыков создания программных приложений позволяющих выполнять поиск записей в базе данных с помощью методов: FindKey, SetKey, Locate, Lookup. Создание вычисляемых полей и полей просмотра.

Задание:

1. Разработать интерфейс приложения
2. Создать приложение для обработки базы данных.
3. Оформить отчет и сдать работу преподавателю.

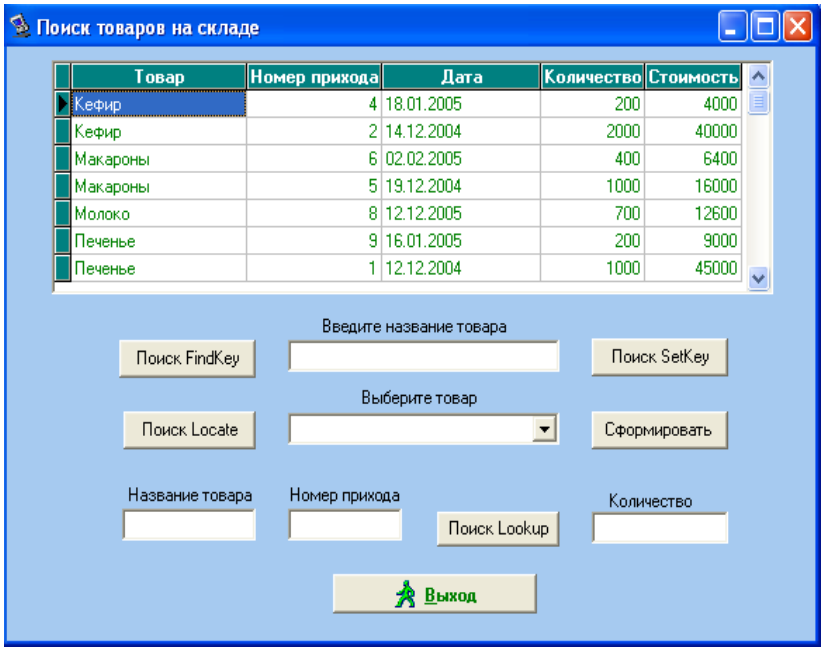
Содержание отчета:

1. Номер, тема, цель работы, задание к работе, вариант задания.
2. Программа (текст рабочего модуля)
3. Результаты работы программы.
4. Выводы о проделанной работе

Постановка задачи

Требуется создать в среде Delphi приложение, предназначенное для выполнения поиска записей в базе данных.

Поиск записей должен производиться в таблице «Приход товаров» по названию товара с помощью методов FindKey, SetKey. Ключ для поиска вводится в поле ввода TDBEdit1. Для поля ввода TDBEdit1 обеспечить возможность быстрого поиска данных. Также в приложении поиск должен производиться с помощью метода Locate. Ключ для поиска должен выбираться их списка товаров, который должен формироваться на основании таблицы «Товары» по полю товары.



Для поиска с помощью метода Lookup на форме расположить два компонента TDBEdit в одном из которых вводится название товара а в другом номер прихода. Результат поиска количество товара отобразить в компоненте TDBEdit4. Примерный вид формы приложения приведен на рисунке.

В таблице «Приход товара» создать два поля: поле просмотра – Cena и вычисляемое поле Stoim. В компоненте TDBGrig1 необходимо отобразить значения поля Stoim, а значения поля Cena отображать не надо. При создании поля поиска установить следующие параметры в окне «New Fields»:

- Name ⇒ Cena
- Type ⇒ Integer
- Lookup ⇒ Установить переключатель
- Key Fields ⇒ Tovar
- Dataset ⇒ Table1
- Lookup Key ⇒ Tovar
- Result Fields ⇒ Cena

При создании вычисляемого поля установить следующие параметры в окне «New Fields»:

- Name ⇒ Stoim
- Type ⇒ Integer
- Calculated ⇒ Установить переключатель

Для всех визуальных компонентов приложения создать всплывающие подсказки.

Лабораторная работа № 14

Тема: Поиск данных с помощью статических и динамических запросов.

Цель работы: Приобретение практических навыков создания программных приложений позволяющих выполнять поиск записей в базе данных с помощью компонента Tquery.

Задание:

1. Разработать интерфейс приложения
2. Создать приложение для обработки базы данных.
3. Оформить отчет и сдать работу преподавателю.

Содержание отчета:

1. Номер, тема, цель работы, задание к работе, вариант задания.
2. Программа (текст рабочего модуля)
3. Результаты работы программы.
4. Выводы о проделанной работе

Постановка задачи

Требуется создать в среде Delphi приложение, предназначенное для выполнения поиска записей в базе данных.

Поиск записей должен производиться в таблицах «Приход товаров» и «Товары». Создать следующие запросы на выборку:

- Статический запрос, записанный первоначально в свойство SQL компонента TQuery , осуществляющий отбор из таблиц «Приход товаров» и «Товары» полей Tovar, Data, Ed_izm. Условие отбора – отбирать только те записи таблицы «Приход товаров» для которых значение поля Tovar равно заданному значению (например, «Кефир»). Для данного товара вывести соответствующую единицу измерения, выбранную из таблицы «Товары».
- Статический запрос, который выполняется при нажатии на кнопку «Поиск». По данному запросу должен осуществляться отбор из таблицы «Приход товаров» полей Tovar, NomPrix, Kolvo . Условие отбора – отбирать только те записи, для которых значение поля Kolvo >300.
- Динамический запрос, который выполняется при нажатии кнопки «По названию». Для формирования запроса значение ключа отбора извлекается из поля редактирования TEdit. В данное поле вводится название товара. Условие отбора – отбирать только те записи, для которых значение поля Tovar равно значению, введенному в поле TEdit1. В результате выполнения запроса в TDBGrid должны отображаться поля Tovar, KolVo.
- Динамический запрос, который выполняется при нажатии кнопки «Из списка». Для формирования запроса значение ключа отбора извлекается из списка компонента TComboBox. В данный список вводятся названия товаров из таблицы «Товары». Условие отбора – отбирать только те записи, для которых значение поля Tovar равно значению, выбранному из списка компонента TComboBox . В результате выполнения запроса в TDBGrid должны отображаться поля Tovar, KolVo.
- Динамический запрос, который выполняется при нажатии кнопки «Два поля». Для формирования запроса значение ключа отбора извлекается из полей редактирования TEdit. В данные поля вводятся название товара и номер

прихода. Условие отбора – отбирать только те записи, для которых значение поля Tovar равно значению, введенному в поле TEdit2 и значение поля NomPrix равно значению, введенному в поле TEdit3 . В результате выполнения запроса в TDBGrid должны отображаться поля Tovar, KolVo.

В таблице TDBGrid названия всех полей должны быть расшифрованы. Все визуальные компоненты должны иметь всплывающие подсказки. Примерный вид приложения приведен на рисунке.

Лабораторная работа № 15

Тема: Создание отчетов.

Цель работы: Приобретение практических навыков создания программных приложений позволяющих выполнять просмотр и печать отчетов с помощью редактора Rave.

Задание:

1. Разработать интерфейс приложения
2. Создать приложение для обработки базы данных.
3. Оформить отчет и сдать работу преподавателю.

Содержание отчета:

1. Номер, тема, цель работы, задание к работе, вариант задания.
2. Программа (текст рабочего модуля)
3. Результаты работы программы.
4. Выводы о проделанной работе

Постановка задачи

Требуется создать в среде Delphi приложение, предназначенное для формирования и печати отчетных документов.

Создать отчет по таблице Товар.

Создать отчет по таблице Приход товара.

Для создания отчета выполнить следующую последовательность действий:

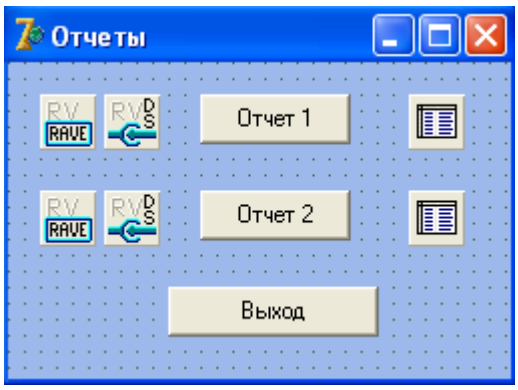
1. Создать новую форму
2. На форме разместить две кнопки: «Отчет по товарам» и «Отчет по приходу товаров»
3. На форме разместить два компонента RvDataSetConnection со страницы Rave политры компонентов и два компонента RvProject.
4. Связать компонент RvDataSetConnection1 с Table1, а компонент RvDataSetConnection2 с Table2 (свойство DataSet)
5. Далее, используя визуальный дизайнер Rave Visual Designer для разработки отчета, надо создать проектный файл отчета (RAV-файл). Это, в свою очередь, требует выполнения следующей последовательности действий:

- Выбрать в Delphi пункт меню Tools \ Rave Designer для вызова Rave Visual Designer
- Выбрать в Rave Visual Designer пункт меню File \ New Data Object для отображения диалогового окна Data Connections

- В списке Data Object Type выбрать Direct Data View и нажать кнопку Next
- В списке Active Data Connections выбрать RVDataSetConnectionl и нажать кнопку Finish, после чего в левой части окна дизайнера отчетов Rave Visual Designer будут показаны свойства компонента DataViewl.
- Выбрать пункт меню Tools | Report Wizards \ Simple Tableдля отображения мастера отчетов Simple Table
- Выбрать Data Viewl и нажать кнопку Next
- Выбрать несколько полей, которые нужно отобразить в отчете, и нажать кнопку Next
- Следуя указаниям на последующих страницах мастера, установить порядок отображения полей, границы, текст заголовка и шрифты, которые будут использоваться в отчете
- На последней странице мастера нажать кнопку Generate для завершения его работы с последующим отображением отчета в дизайнере страниц Page Designer, расположенном вправой части окна
- Выбрать пункт меню File | Save для вывода на экран диалогового окна Save As, после чего перейти в каталог, в котором будет расположено создаваемое приложение Delphi, и сохранить файл проекта Rave, как Projectl.rav.
- Свернуть окно дизайнера отчетов Rave Visual Designer и вернуться в Delphi.

6. После этого в Инспекторе объектов нужно установить в свойстве ProjectFile ссылку на файл проекта отчета (Projectl.rav), который был создан на предыдущем шаге.

В результате выполнения всех вышеперечисленных действий форма на этапе проектирования принимает такой вид, как показано на рисунке.

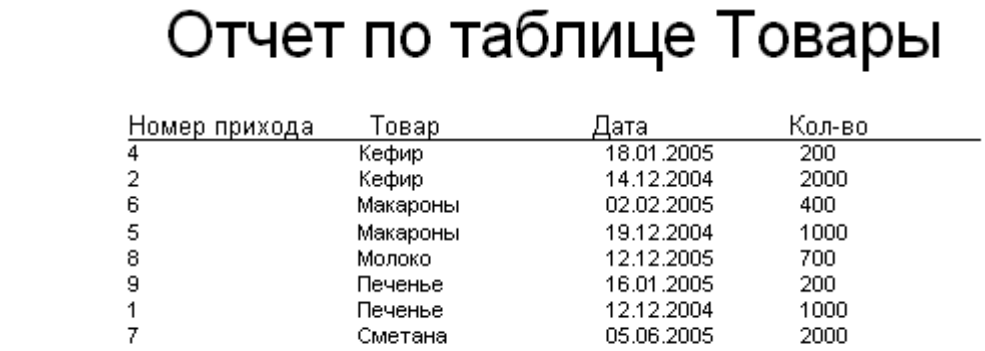


Написать обработчик события OnClick кнопки Button1 следующего вида:

```
procedure
TForml.ButtonlClick(Sender:TObject);
  begin
    RvProjectl.Execute;
  end;
```

Аналогично создать второй отчет.

Приблизительный вид отчета приведен на рисунке.



Лабораторная работа № 16

Тема: Создание справочной системы приложения.

Цель работы: Приобретение практических навыков разработки справочной системы

Задание:

1. Создать справочную систему приложения.
2. Проверить работоспособность программы.
3. Оформить отчет и сдать работу преподавателю.

Содержание отчета:

1. Номер, тема, цель работы, задание к работе.
2. Результаты работы программы.
3. Выводы о проделанной работе.

Справочная система приложения должна содержать следующий теоретический материал:

1. События мыши.
2. События клавиатуры.
3. Редактор символьной информации

3.1. Edit
3.2. MaskEdit
3.3. Memo
3.4. RichEdit

4. Списки

4.1. ComboBox
4.2. ListBox

5. Кнопки

5.1. Button
5.2. BitBtn

5.3. SpeedButton
6. Переключатели
6.1. CheckBox
6.2. RadioButton
7. Стандартные диалоги
7.1. OpenDialog
7.2. SaveDialog
7.3. FontDialog
7.4. ColorDialog
7.5. PrintDialog
7.6. PrintSetupDialog
8. Главное и локальное меню
8.1. MainMenu
8.2. PopupMenu
9. Обработка исключительных ситуаций
10. Вызов внешних приложений
11. Компоненты для работы с файлами и каталогами.
11.1. DriveComboBox
11.2. FileListBox
11.3. FolerComboBox
12. Панели инструментов
12.1. ToolBar
12.2. CoolBar
12.3. ControlBar
13. Строка состояния
14. Индикаторы
14.1. ProgressBar
14.2. Gauge
15. Графические компоненты
15.1. Image
15.2. Shape
15.3. Bevel
15.4. PaintBox
15.5. Chart
16. Мультимедиа
16.1. Animate
16.2. MediaPlayer
17. Базы данных
17.1. Table
17.2. Query
17.3. DataSource
17.4. DBGrid
17.5. DBNavigator
17.6. DBText
17.7. DBEdit

17.8. DBMemo
17.9. DBRichEdit
17.10. DBImage
17.11. Поиск данных
17.12. Навигация по набору данных
17.13. Модификация набора данных
17.14. Отчёты

По каждому разделу должна быть кратко изложена информация о свойствах и методах.

Последовательность создания справки:

1. Создать текстовые файлы, содержащие разделы справочной системы и необходимые перекрёстные ссылки.
2. Разработать проектный файл (*.hpj)
3. Создать справочный файл с помощью компилятора (*.hlp)
4. Разработать файл содержания (*.cnt)
5. Подключить справочный файл к приложению.

Лабораторная работа № 17

Тема: Построение инфологической модели.

Цель работы: Приобретение практических навыков построения инфологической модели.

Задание:

Требуется, основываясь на описании предметной области, спроектировать базу данных, все отношения в которой нормализованы до нормальной формы Бойса-Кодда.

Существуют различные программные инструменты работы с ER-диаграммами. Например, это ERD PLUS DiagrammingTool, работающий через web-интерфейс.

https://erdplus.com

Зайдите по приведенной выше ссылке, зарегистрируйтесь на сайте и создайте ER-диаграмму. В конструкторе там используются несколько иные правила изображения связей (сходные с нотацией Мартина), в остальном – все нам уже знакомо. В качестве дополнительного материала можно предложить обучающий ролик, доступный по приведенной ниже ссылке. Он на английском, но даже если вы не очень хорошо владеете английским, там все достаточно понятно из видеоряда.

https://www.youtube.com/watch?v=A5-yy51eNGY

Теперь для той же задачи постройте ER-диаграмму в среде ERD PLUS, указав на диаграмме все атрибуты.

Кроме ER-диаграмм ERD PLUS позволяет создавать «реляционные схемы» (relationalschemas) и генерировать код на SQL, выполнение которого в среде СУБД позволяет создать таблицы и связывающие их внешние ключи.

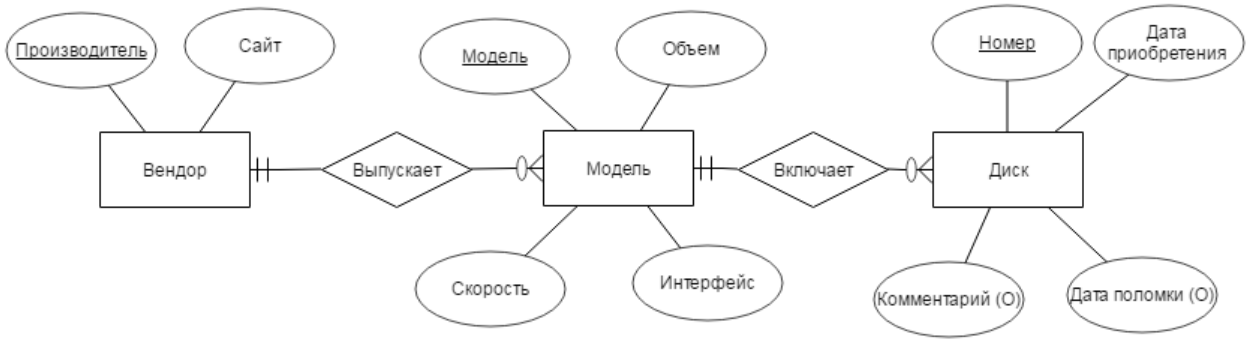
Демонстрационный ролик от разработчиков доступен по приведенной ниже ссылке.

https://www.youtube.com/watch?v=o5L2HUb_mqQ

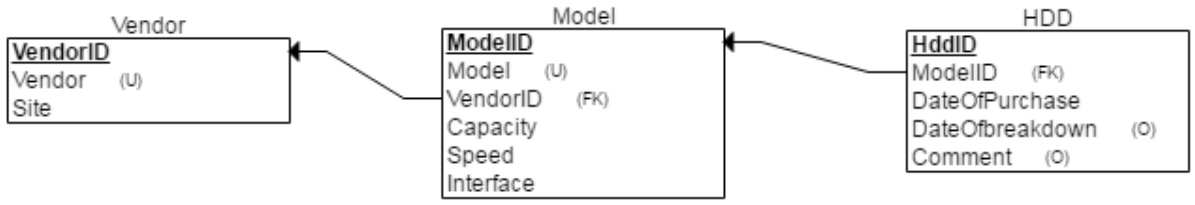
Для той же задачи постройте реляционную схему в ERD Plus. Обратите внимание, что создавать в таблице атрибуты, являющиеся внешними ключами, не надо: они автоматически будут добавлены, когда вы определите связи между таблицами.

Выполните генерацию скрипта на SQL и скопируйте его через буфер обмена в текстовый редактор. ERD Plus можно рассматривать как пример простого средства автоматизированного проектирования баз данных. Для простых по структуре небольших баз подобных средств может оказаться вполне достаточно. В более сложных случаях используется профессиональный инструментарий, например ERwinDataModeler.

Пример ER-диаграмм на рисунке 17.1.



а)



б)

Рисунок 17.1 Модели базы данных: а) ER-диаграмма; б) Даталогическая модель

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Культин, Н.Б. Программирование в Turbo Pascal 7.0 и Delphi/ Н.Б. Культин. - СПб.: БХВ-Петербург, 2009. - 400с.: ил. + CD –ROM. Экз.10

2. Культин, Н. Б. Основы программирования в Delhi XE: [текст]/ Н. Б. Культин. - СПб.: БХВ, 2011. - 416 с.

3. Осипов, Д.Л. Базы данных и Delhi: Теория и практика [текст]/ Д.Л. Осипов. - СПб.: БХВ-Петербург, 2011.

4. Хорев, П.Б Объектно - ориентированное программирование: [текст]: Учеб. пособие для бакалавриата/ П.Б Хорев. - М.: Академия, 2012. - 447 с

Дудник Евгения Александровна

ПРОЕКТИРОВАНИЕ ПОЛЬЗОВАТЕЛЬСКИХ ИНТЕРФЕЙСОВ
(лабораторный практикум по программированию в среде Delphi)
Учебно-методическое пособие для студентов всех форм обучения

Подписано к печати 03.06.22. Формат 60x84/16.
Усл. печ. л. 4,44. Тираж 25 экз. Зак. 221810 Рег. № 11.

Отпечатано в ИТО Рубцовского индустриального института
658207, Рубцовск, ул. Тракторная, 2/6.