Министерство образования и науки Российской Федерации

федеральное государственное бюджетное образовательное учреждение высшего профессионального образования

«Санкт-Петербургский государственный университет технологии и дизайна»

Кафедра прикладной информатики

Программная инженерия

Создание пользовательских проектов с применением объектно-ориентированного программирования

Методические указания и контрольные задания к изучению дисциплины «Программная инженерия»

для заочной формы обучения
по направлению подготовки 09.03.03
профиль – «Прикладная информатика в экономике»
(степень «бакалавр»)

Составители:

Ф. Л. Хватова

М. А. Ермина

Санкт-Петербург

2017

Введение

Методические указания предназначены для дальнейшего освоения современных компьютерных технологий в дисциплине «Программная инженерия».

Программная инженерия — это область компьютерной науки и технологии, которая занимается разработкой и проектированием больших и сложных программных систем (ПС), соответствующих сложности современных требований не только к инженерным, но и финансовым, экономическим задачам и задачам других направлений. Во второй половине прошлого столетия в программировании произошел переход от разработки относительно простых программ к разработке сложных программных комплексов. К числу таких сложных программ относятся: системы управления космическими объектами, управления оборонным комплексом, автоматизации крупного финансового учреждения и т.д.

Создание сложных программных систем требует систематизированного, научного и предсказуемого процесса проектирования, разработки и сопровождения программных средств и больших финансовых вложений, поэтому уже в конце прошлого столетия появилась необходимость создания новых технологий в программировании.

Регламентом программной инженерии являются: методология и стандарты современных процессов управления проектами сложных систем и программных средств, которые призваны обеспечить требуемые качество, надежность и безопасность функционирования программных продуктов, что должно привести к сокращению стоимости ПС. Для решения этих задач в программной инженерии формируется новая область знания и научная дисциплина — экономика жизненного цикла программных средств, как часть экономики промышленности и вычислительной техники в общей экономике государств и предприятий.

Программная инженерия относится к циклу базовых дисциплин, поэтому она должна опираться на знания дисциплин: «Информатика», «Вычислительные системы, сети и телекоммуникации», «Информационные системы и технологии», «Операционные системы», «Базы данных», «Разработка и стандартизация программных средств и информационных технологий». Известно, что Информатика занимается теорией и методами вычислительных и программных систем, в то время как программная инженерия занимается практическими проблемами создания ПС. Информатика составляет теоретические основы программной инженерии, поэтому инженер по программному обеспечению должен знать информатику. Круг проблем, стоящих перед программным инженером значительно шире просто написания программ, что требует фундаментальных знаний, выходящих за рамки информатики. Это управление финансами, организация работ в коллективе, взаимодействие с заказчиком и т.д.

Решением этих проблем стало использование подхода или метода, который стали называть объектно-ориентированным проектированием и программированием. Поддержка технологии объектно-ориентированного программирования (ООП) осуществляется средствами языков Паскаль, Интегрированных средств разработки (ИСР) Delphi и Си ++ и др. Далее изложение материала касается непосредственно ИСР Delphi7 .

1. Контрольные работы № 1, 2

Тема: «Создание пользовательских проектов на основе объектно-ориентированного программирования (ООП) в ИСР Delphi 7 – ОС Windows»

Цель работы - Изучение основных принципов создания надежного и качественного программного продукта на примерах пользовательского проекта, удовлетворяющего предъявляемым требованиям Программной инженерии.

1.1. Задания к выполнению контрольных работ № 1, 2

1.1.1. Задания к выполнению контрольной работы № 1

Теоретическая и практическая работа в среде Delphi7… по созданию проектов.

Задание 1. Краткий реферат (до 10 страниц). Развитие объектно-ориентированного проектирования и программирования. Язык Delphi – Версии и его развитие (по материалам Internet – среды, списка литературы).

Задание 2. Знакомство с внешним видом среды разработки. Основные окна Delphi (главное окно, окно дерева объектов, окно инспектора объектов, окно браузера формы, окно формы, окно кода программы). Внешний вид среды можно оформить на одном из заданий 3.1. – 3.10.

Задания 3. 1 – 3. 10. Создание пользовательских проектов на основе представленных макетов окон и алгоритмов поставленной задачи (раздел 3.1, дополнительно для всех заданий тексты программных кодов представлены в Приложении Б под соответствующим номером).

1.1.2. Задания к выполнению контрольной работы № 2

Работа с файлами.

Задания №1 – № 10. Текст заданий представлен ниже (раздел 3.2).

1.2. Требования к выполнению заданий:

Учебный план курса предусматривает две контрольные работы: контрольную работу № 1 и контрольную работу № 2.

Контрольные работы выполняется в период между сессиями в установленные учебным графиком сроки.

1.3. Обеспечение дисциплины методическими материалами

- Описание заданий к выполнению контрольной работы;
- методические материалы - краткое описание интегрированной среды разработки (ИСР), а также теории и практики проектирования в ООП;
- примеры заданий с иллюстрацией последовательности выполнения и конечного результата;
- порядок оформления контрольной работы;
- список рекомендуемой литературы.

1.4. Для каждого задания 3. 1 – 3. 10 выполнить:

- Запустить Delphi 7;
- Ознакомиться с внешним видом среды разработки Delphi, включая создание нового приложения, открытие и сохранение его.
- Изучить линейку компонентов и их поиск на вкладках линейки нужных компонентов для конкретного задания.
- Конструировать окно формы соответственно макету задания.
- Создать заготовки процедур обработки событий (по указанным в макете кнопкам для заданий 3.1-3.10).
- Разместить коды в заготовки процедур (из соответствующего задания в Приложении Б для заданий 3.1-3.10).
- Выполнить проект со своими данными и получить результат.
- Скопировать результат через Paint в отчет.

2. Методические указания к выполнению заданий

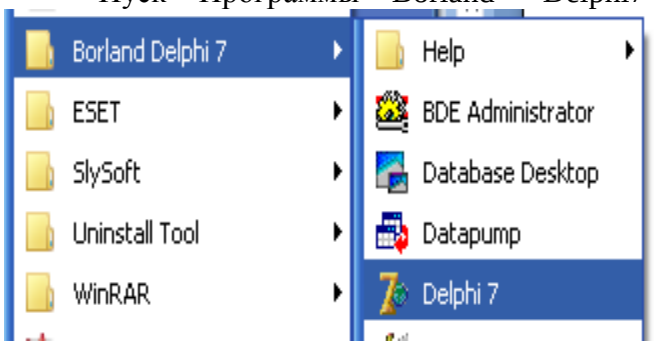
2.1. Практические основы работы в ИСР Delphi

Первое знакомство со средой Delphi.

Практические рекомендации:

- **Создание отдельной** папки для каждого приложения, так как в ней будет находиться порядка 12 файлов. Файлы Delphi должны быть изолированы от всех прочих файлов.
- **Загрузка Delphi**

  Пуск→Программы→Borland → Delphi7 (возможны другие версии)



с ярлычком 

Рис. 2.1.1. Обращение к среде **ИСР Delphi**

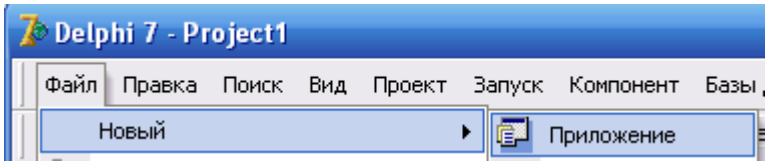


Рис. 2.1.2. Создание нового проекта

Для создания нового проекта выбрать Файл→Новый→Приложение (*рис. 2.1.2*):

- Первичная загрузка Delphi автоматически создает заготовку модуля для пустой формы, которая также открывается на экране (*рис. 2.1.3*).
- Модуль и проект, созданные в Delphi, по умолчанию с именами Unit1 и Project1, необходимо сохранить под другими именами (латинским шрифтом) по смыслу, например, Unlb1 и Prlb1 в созданной ранее папке с помощью команды Файл→Сохранить как.
- Настройка опций приложения (необязательна, но очень полезна): Главное меню, Инструменты, Опции редактора. В окне Свойства редактора выбрать вкладку Цвет, в окне вкладка «Элемент» выбрать «Занятые слова» и определить для них Главный цвет – Red. Цвет фона – Aqua, в верхнем списке – Ocean. Такая настройка позволит нагляднее ориентироваться в тексте программ.
- Установка положения формы в окне (с помощью Инспектора свойств), положив свойства, например Top - 80, Width – 200. Такое положение формы позволит следить за изменениями в модуле при внедрении того или иного компонента в форму. В дальнейшей работе можно выбирать другие установки.

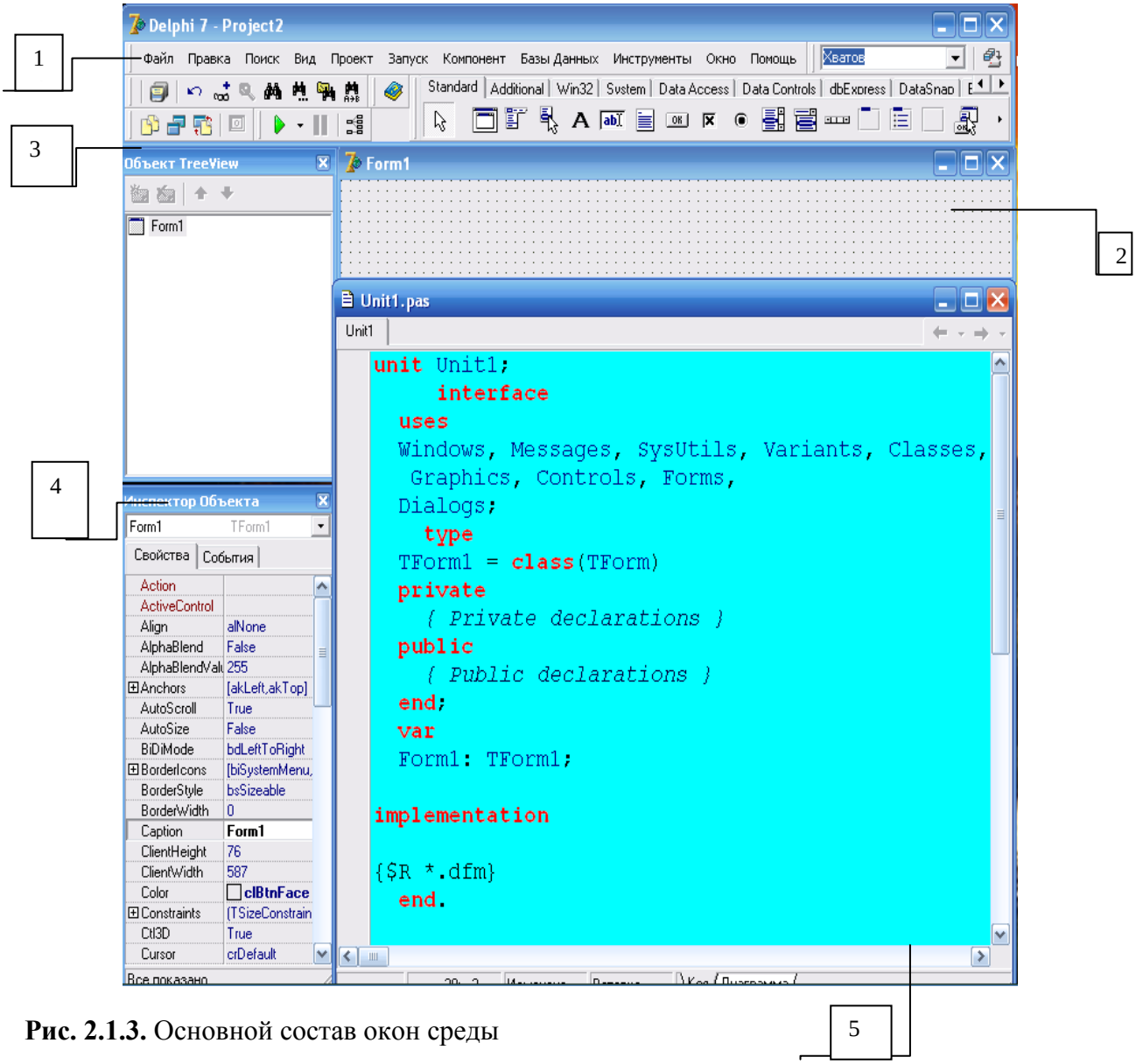


Рис. 2.1.3. Основной состав окон среды

Основной состав окон среды:

При запуске Delphi среда открывает ряд необходимых для работы окон:

1. Главное окно (**Delphi7 - Project1**);
2. Окно формы (**Form1**);
3. Окно дерева объектов (**Object TreeView**);
4. Окно инспектора объектов (**Object Inspector**);
5. Окно кода программы – редактора формы (**Unit1.pas**).

Кроме указанных окон часто используются:

- Обозреватель проекта (Browser);
- Менеджер проектов (Project Manager).

Главное окно содержит необходимый инструментарий для создания, конструирования, отладки и запуска приложения. Оно всегда должно присутствовать на экране. В состав главного окна входят:

- Заголовок.
- Главное меню.
- Панель инструментов.
- Линейка компонентов с набором страниц для каждого компонента.

Главное окно. Его назначение – управлять проектом создаваемого приложения. В нем располагаются: главное меню, набор инструментальных кнопок и палитра компонентов.

Команды главного меню представляют двухуровневые структуры, поэтому целесообразнее пользоваться кнопками панелей инструментов. **Инструментальные кнопки** осуществляют быстрый доступ к командам меню и разделены на ряд групп левой части окна под меню.

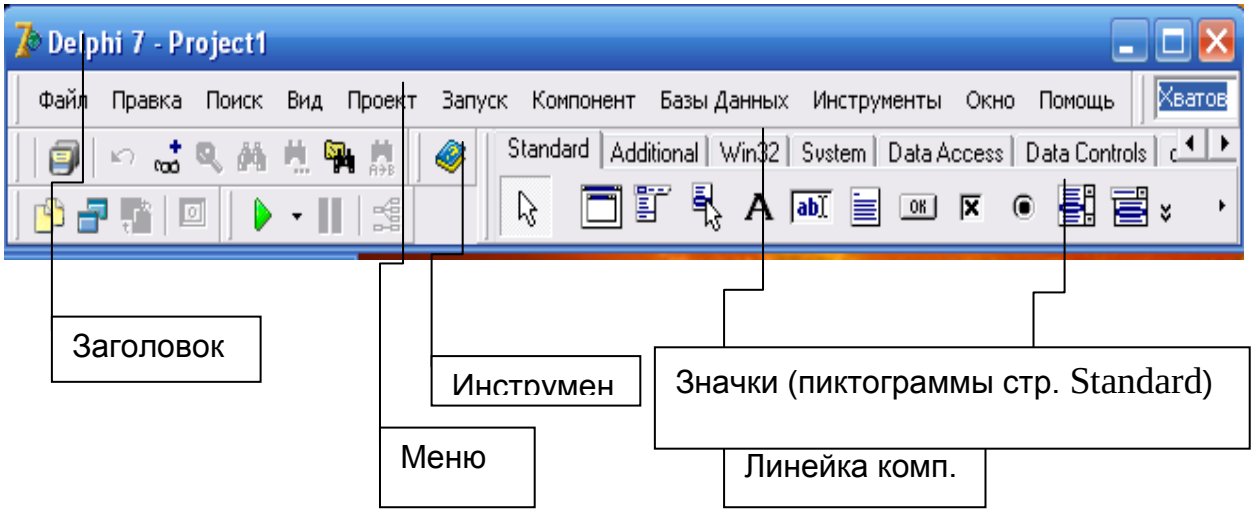


Рис. 2.1.4. Главное окно

Особое внимание рекомендуем обратить на кнопки, представленные ниже:

Наведя на них указатель мыши, Вы узнаете их назначение.

При выполнении запуска выполняется компиляция в том случае, если при компиляции не обнаружены неисправимые ошибки и загрузочный модуль создан, иначе – сообщение об ошибках (*Приложение В*).

Остальные окна, показанные на рисунке 2.1.3 будут рассмотрены ниже (*Пример 1*).

Краткий обзор вкладок (страниц) линейки компонентов

Компонент – функциональный элемент, размещаемый в главном окне – форме и имеющий определенный набор свойств. С внедрением компонентов создается внешний каркас программы для обеспечения заданного заказчиком алгоритма программы**.** Щелкая мышью по нужному компоненту, Вы вносите его в форму, а далее регулируете его место в форме. При конструировании форм Delphi готовит программные заготовки и файл ресурсов в зависимости от размещенных на форме компонентов (поля, списки, таблицы, кнопки, графики и т. д.). Окно формы с размещенными на нем компонентами представляет каркас программы пользовательского приложения.

Палитра внедряемых в форму объектов представлена широким набором, расположенных на страницах (вкладках) линейки компонентов:

Standard. На вкладке размещены стандартные для интерфейса элементы, без которых не обходится практически ни одна программа.

Additional. Содержит дополнительные компоненты (порядка 26), помогающие сделать диалоговые окна разнообразнее. Отметим некоторые. Например, кнопки с надписью и пиктограммой -BitBtn со значками, SpeedButton- с возможностью фиксироваться в утопленном состоянии, а также объединения в одну при нажатой только одной. Таблица DrawGrid – с ячейками, в которых могут содержаться строки и

изображения. Текстовая таблица - StringGrid с ячейками на пересечении строк и столбцов. Изображение – Image и т. д.

System. Компоненты, имеющие различное функциональное назначение. Например, компоненты, поддерживающие стандарты для технологии OLE, для управления средствами мультимедиа, Timer - для отметки интервалов реального времени и т. д.

Dialogs. Реализует стандарты для ОС диалоговых окон:

- открытия и сохранения файла.
- открытия и сохранения изображения.
- Выбора шрифта и выбора цвета.
- Настройка параметров печати и принтера.
- Поиска и замены.

Samples (выборка, образец**).** Компоненты **Samples** используются как образцы разработки нестандартных компонентов.

- Индикатор величин.
- Таблица цветов.
- Структура каталогов.
- Календарь.
- Спаренные кнопки – SpinButton (как верхняя и нижняя кнопки.).
- Поле со спаренными кнопками – SpinEdit для отображения и ввода целого числа.

ActiveX. Чужие для Delphi компоненты, созданные другими инструментальными средствами, внедренными с помощью OLE

Ниже раскрыты значки компонентов вкладки Standard (*рис. 2.1.5*).

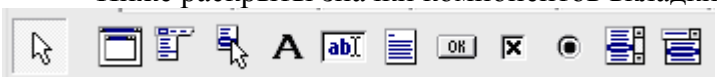


Рис. 2.1.5. Значки (Пиктограммы) страницы Standard

Пример 1. Внедрение компонентов в форму и определение им свойств.

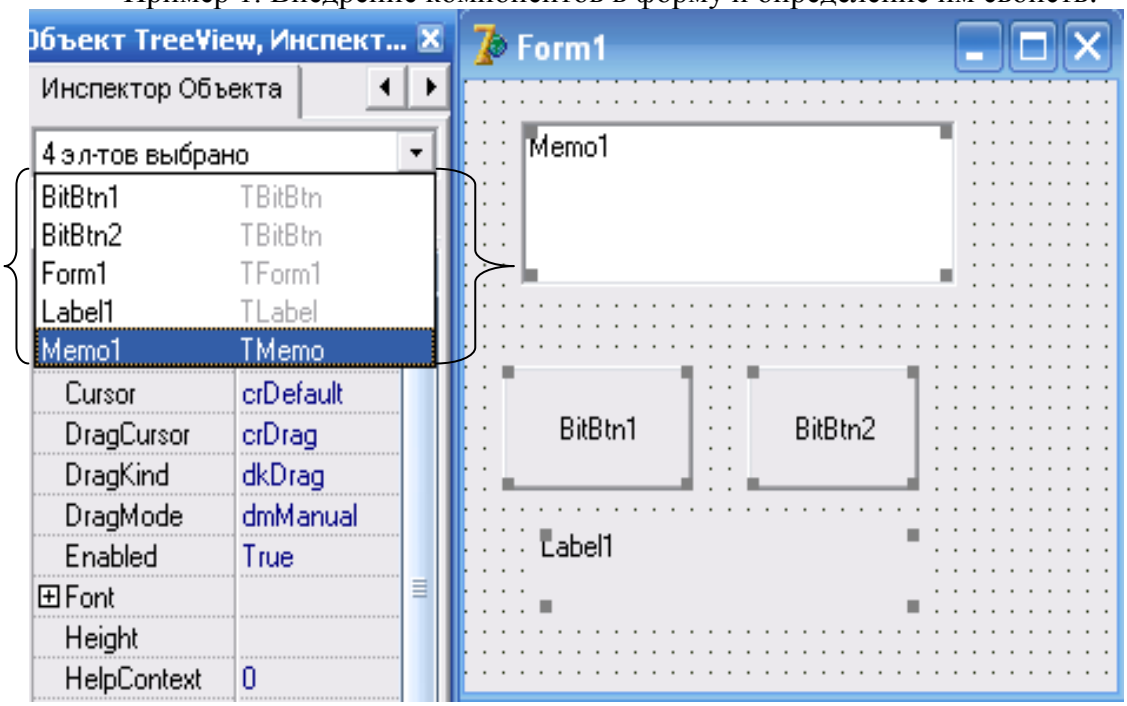


Рис. 2.1.6. Object TreeView. Инспектор объектов. Окно формы

Рисунок 2.1.6 содержит окна 2, 3, 4 среды:

- Окно формы – Form1. (2)
- Окно дерева объектов – Object TreeView. (3)
- Окно Инспектора объектов – Properties Object. (4)

На рисунке показаны окна для примера с внедренными компонентами в форму: Memo1 (многострочный редактор), BitBtn1 и BitBtn2 (две кнопки), Label1 (метка). К выбранным компонентам могут быть применены определенные свойства в инспекторе объектов. Все компоненты перечислены в дереве объектов. Окно дерева объектов предназначено для наглядного указания связей между отдельными компонентами, размещенными в активном окне формы. Щелчок на любом компоненте активизирует его и отображает его свойства в окне инспектора объектов. Двойной щелчок на компоненте приводит к срабатыванию механизма Code Insight, который вставляет в окно кода заготовку для обработчика события OnClick. Окно Инспектора объектов содержит две вкладки - Properties (свойства) и Events (события) (*рис. 2.1.7*). Первая вкладка служит для установки свойств компонентам, вторая - определяет реакцию компонента на то или иное событие. Каждая вкладка окна состоит из двух колонок. Левая колонка – название свойства или события, а правая – конкретное назначение свойства или имя подпрограммы, обрабатывающей событие.

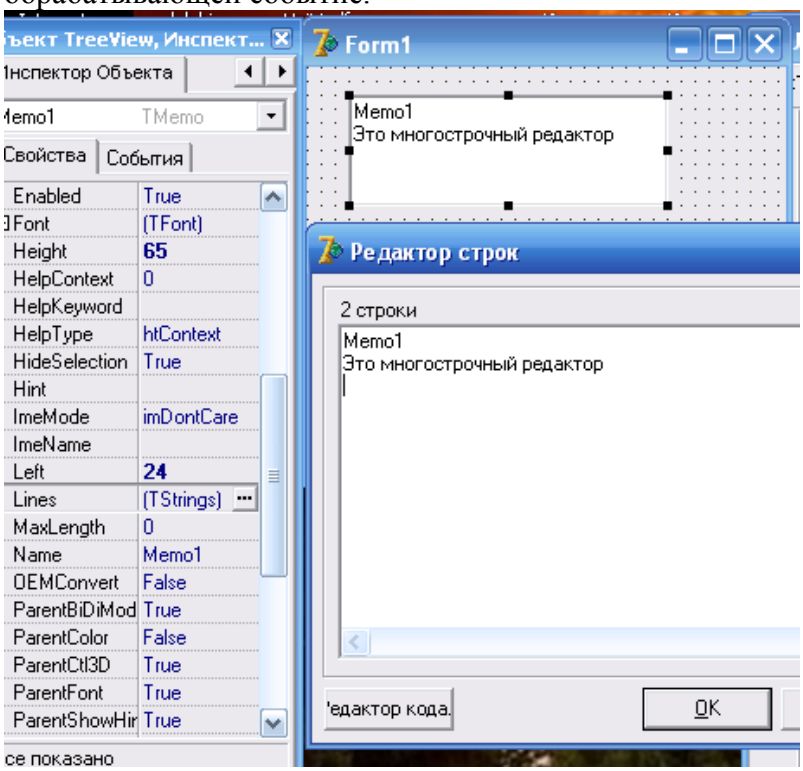


Рис. 2.1.7. Применение свойства Lines с вызовом TStrings (процедуры) для занесения текста в Memo1 с помощью редактора строк

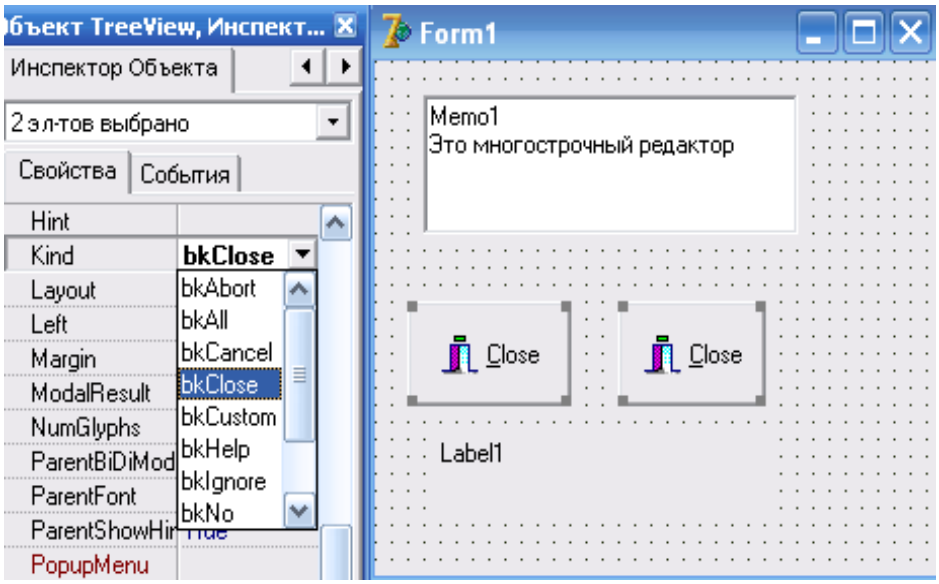


Рис. 2.1.8. Применение свойства bkClose к кнопкам BitBtn (значки)

Окно редактора кода программы - 5 (*рис. 2.1.3*) служит для создания кода программы, а также для редактирования его, если будет в этом необходимость. Первоначально, при создании нового приложения (*рис. 2.1.9*) в окне редактора кода содержится только модуль Unit1, включающий описание класса TForm1, в котором данными (полями) представлены экземпляры классов Memo1, BitBtn1, BitBtn2, Label1, расположенные в окне формы.

Unit1

```
uses
Windows, Messages, SysUtils, Variants,
Classes,   Graphics, Controls, Forms,
  Dialogs, StdCtrls, Buttons;
 type
  TForm1 = class(TForm)
    Memo1: TMemo;
    BitBtn1: TBitBtn;
    BitBtn2: TBitBtn;
    Label1: TLabel;
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

end.
```

Рис. 2.1.9. Содержимое модуля Unit1 на первом этапе – конструировании формы

2.2. Основы Объектно-ориентированного проектирования и программирования

2.2.1. Класс

Суть подхода к объектно-ориентированному проектированию и программированию состоит в том, что вводится понятие класса, как развитие понятия модуля с определенными свойствами и поведением, характеризующимися обязанностями класса. **Delphi** предусматривает только инструментарий создания классов, сами же классы создаются разработчиками **ПО. В Delphi7**…имеется свыше 400 стандартных классов – компонентов (библиотека), разработанных программистами фирмы Borland. Однако пользователь может создавать свои новые классы, используя важное свойство классов – наследование. Новый класс может наследовать свойства, методы и события класса, на основе которого он создан (родительского класса). Таким образом, **классами** называются функционально законченные фрагменты программ, служащие образцами для создания себе подобных экземпляров. Будучи однажды созданными, они могут быть включены в другие программы или в разные места одной и той же программы, что обеспечивает высокую производительность при программировании.

Каждый компонент библиотеки принадлежит к строго определенному классу. Класс является ядром объектно-ориентированного программирования и относится к структурированному типу, содержащему описание атрибутов и функциональности некоторой категории объектов. Каждый класс может порождать объекты – экземпляры данного класса.

Структура описания класса

Type<Имя класса>**= class(<Имя** класса-родителя>**)**
Private
<Частные описания>**;**
protected
<Защищенные описания>**;**
public
<Общедоступные описания>**;**
published
<Опубликованные описания>**;**
end;

Разделы **private** и **protected** содержат защищенные описания, доступные внутри модуля, в котором они находятся, а **Public** содержит общедоступные описания, видимые в любом месте программы, где доступен сам класс. Раздел **published** содержит опубликованные описания в дополнение к общедоступным описаниям, и порождают информацию о типе времени выполнения (Run-Time Type information-RTTI). Одно из назначений раздела **published –** обеспечение доступа к свойствам объектов при конструировании приложений. В **Инспекторе объектов** (это одно из окон интегрированной среды) видны те свойства, которые являются опубликованными. Если **published** не указан, то по умолчанию он является опубликованным.

Как было сказано ранее, разработчиками среды создано множество классов, включенных в библиотеку, используя важное свойство классов – наследование, однако пользователь может создавать свои новые классы.

Пример фрагмента описания пользовательского класса.

```
Type
TMyClass=class
aIntField:intger;
aStrField:String;
                        } Данные – Поля
……..
End;
```

Запись подобна описанию записей в **Pascal**, где поля - уникальные для каждого экземпляра данные, обладающие своим типом, например, имя пользовательского класса – **TMyClass** от предка **class,** а данные - это поля **aIntField** (целого типа) и **aStrField** (строкового типа)

Новый класс может наследовать свойства, методы и события класса, на основе которого он создан (родительского класса). Основой ООП являются три основных принципа:

- Инкапсуляция.
- Наследование.
- Полиморфизм.

Инкапсуляцией называется объединение данных и логики их обработки в одно целое, название которому объект. Так, например класс **TForm** инкапсулирует все необходимое для создания Windows - окна, а **TString** - обеспечивает работу с таблицей и т. д.

Наследование. Любой класс может быть порожден от другого класса, при этом он (порожденный класс) наследует поля, методы и свойства своего родителя, а также может создавать свои. Например, наследование: **TChildClass=class(ParentClass)** указывает на то, что порожденный новый класс наследует поля, свойства и методы класса – родителя. Таким образом, появляется возможность создавать сложные структуры классов.

Полиморфизм – это свойство классов решать схожие по смыслу проблемы различными способами. Например, при изменении алгоритма того или иного метода в потомках класса, потомки получают новые свойства, отсутствующие у родителя путем перекрытия метода в потомке**.** Все классы порождены от единственного родителя – класса **Tobject**.

Класс представляет собой единство трех сущностей – полей, методов и свойств. **Назначение полей класса** – хранение информации об объекте. **Методы** - это процедуры и функции для обработки полей. **Свойства** занимают промежуточное положение между полями и методами. С одной стороны, свойства можно использовать как поля, присваивая им различные значения с помощью оператора присваивания, с другой стороны - внутри класса доступ к значениям свойств выполняется методами класса. Объединение этих сущностей в единое называется инкапсуляцией – одним из основных принципов ООП. Задача, решаемая с использованием методики ООП, описывается в терминах объектов и операций над ними, а программа при таком подходе представляет собой набор объектов и связей между ними.

Все объекты компонентов размещаются в объектах – формах. Для каждой формы создается свой модуль, а в модулях осуществляется программирование задачи, в которой

идет обращение к свойствам и методам используемых объектов. Таким образом, объект – это:

- Совокупность данных и способов работы с ними.
- Совокупность свойств и методов, а также событий, на которые он реагирует.

Данные называются полями объекта, а подпрограммы его методами. Главные свойства объекта – функциональность и неделимость. Как уже указывалось выше, в Delphi разработаны сотни объектов, позволяющих строить уникальные структуры, позволяющие повысить производительность и качество программ, что соответствует принципам программной инженерии.

Программирование в современных средах строится на тесном взаимодействии двух процессов:

- Конструирование визуального проявления программы, т.е. ее окна. Таким окном в указанной технологии является форма, в которую внедряются другие объекты – компоненты (визуальные, невизуальные, диалоги). В ИСР Delphi7 библиотека визуальных компонентов представлена линейкой с набором страниц, каждая из которых содержит определенный набор пиктограмм компонентов (кнопки, списки, поля, таблицы и т. д.).
- Написание кода, придающего элементам этого окна и программы функциональность.

2.2.2. Конструирование проекта

Форма

Форма служит основой всех разрабатываемых приложений. Ее окно – контейнер для всех видов компонентов. Форме в Delphi соответствует класс **TForm**. При добавлении формы в проект создаются автоматически модуль формы, в котором она описывается как наследник класса **TForm**, и файл содержащий информацию о параметрах самой формы и внедренных в нее компонентах.

Ее окно имеет свойства, присущие всем окнам Windows.

Окно формы

Окно формы это проект Windows – окна будущей программы. Вся область на этапе конструирования заполнена точками координатной сетки для упорядоченного размещения компонентов. На начальной стадии это окно пусто (*рис 2.2.1*).

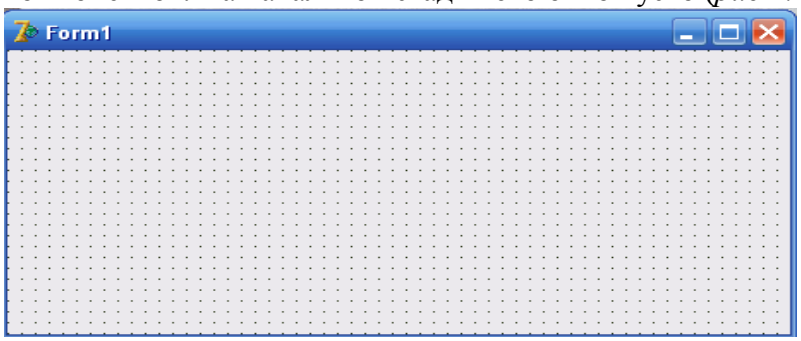


Рис 2.2.1. Окно пустой формы.

Форма имеет определенный набор свойств. Конструирование окна (внедрение в него компонентов) является главным достоинством визуального программирования.

Иерархия компонентов Delphi.

Все компоненты, которые могут быть внедрены в форму, представляют палитру компонентов. Каждый компонент, как класс, является наследником класса **TComponent** (базовый для компонентов), управляющего применением к ним возможностей визуального программирования:

- Возможность переноса компонента в форму – создание экземпляра данного класса, управление свойствами компонента с помощью визуального построителя и Инспектора объектов.

- Принадлежность. При создании экземпляра ему определяется его владелец, например, форма.
- Сохраняемость и восстанавливаемость. Определяется классом **TPersistent** (постоянный).
- Поддержка технологии COM, позволяющей преобразовывать компоненты Delphi в объекты **ActiveX** и импорт из **ActiveX** в Delphi.

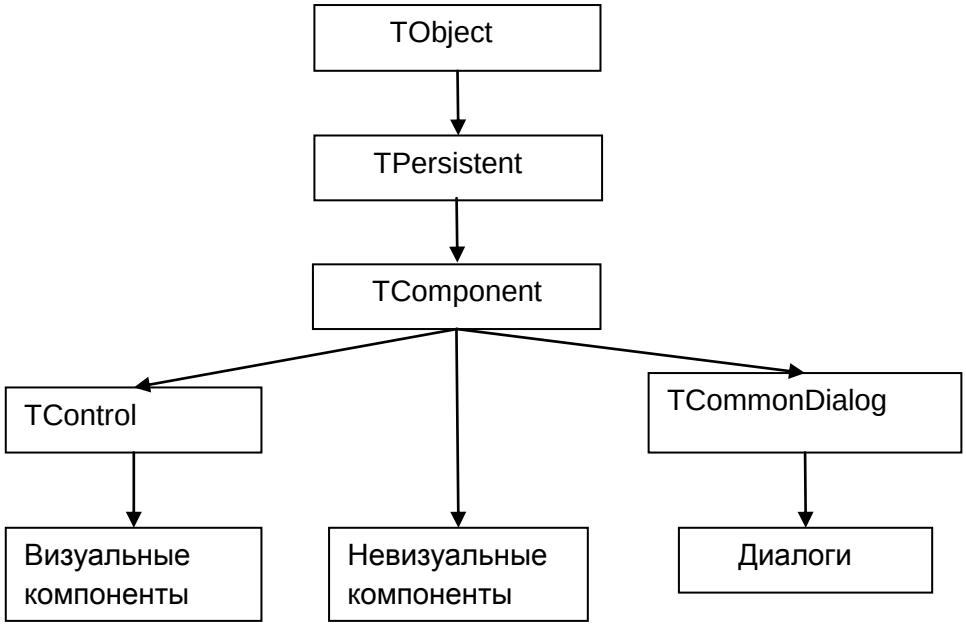


Рис. 2.2.2. Иерархия компонентов Delphi.

Визуальные компоненты (ЭУ – элементы управления) имеют взаимодействие с пользователем, визуальное представление и управляются **TControl (наследником TComponent)** с целью определения им свойств, методов и событий, такими как расположение на экране, выравнивание друг относительно друга, применяемые шрифты и др.

Невизуальные компоненты не показаны на экране во время выполнения программы , а отображаются в окне формы в виде пиктограмм, добавляемых в поддержку некоторых технологий, например **BDE(), ADO(),** а также при взаимодействии программ через сети. Они не взаимодействуют с пользователем.

Диалоговые компоненты. Это не визуальные компоненты. Вызываются методом **Execute**, описанным в классе **TCommonDialog**, отображают диалоговое окно взаимодействия с пользователем.

Из очень большого состава библиотеки представим необходимые для учебного конструирования проектов компоненты, внедряемые в форму, для ввода и отображения текстовой информации.

Организация ввода/вывода данных.

Организация ввода/вывода данных включает использование ряда групп компонентов:

- Ввода – вывода (**Edit, MaskEdit, SpinEdit, Label, MontCalendar, DateTimePicker**), редактирования простых данных (ввод строк и чисел), редактирование логических значений;
- Работа со списками **ListBox, ComboBox, ColorBox, CheckListBox** для ввода, хранения, выбора и передачи данных;
- Работа с многострочными редакторами (**Memo, RichEdit**) для ввода, выбора и передачи многострочных данных с возможностью форматирования;
- Работа с компонентами (**StringGrid, DrawGrid**, **ValueListEditor**) для ввода, редактирования, обработки и передачи данных, представленных в табличной форме;
- Редактирование логических значений с помощью переключателей **(RadioButton, CheckBox)**.
- Изменение числового значения в заданном диапазоне (**ScrollBar –** полоса скроллинга**, TrackBar –** выбор числового значения с помощью бегунка).
- Работа с файлами (запись в файл, чтение из файла).

Компоненты **Edit, MaskEdit, Label, LabelEdit** относятся к одному классу, вследствие чего имеют множество схожих свойств, а также и свои функциональные особенности. **UpDown** используют для настройки пределов редактирования значения и шага его изменения (**Min, Max, Increment**). **MaskEdit (**строка маскированного ввода**)** используют для ввода информации по заданному шаблону (маске).

Остановимся на основных свойствах компонентов ввода/вывода: **Edit, Label, Memo, ListBox, StringGrid.** Остальные компоненты будем пояснять по мере рассмотрения лабораторных работ.

Основные свойства рассматриваемых компонентов:

Edit, Label –Caption (подпись).

ListBox – Items (элемент)**, Add** (добавить)**, MultiSelect** (множественный выбор)**, Selected** (выделение)**, SelCount – SelectedCount (**количество выбранных элементов**).**

Memo – Lines (строки)**, Add.**

StringGrid, как и все указанные компоненты для работы с данными в табличной форме, имеют следующие возможности**:**

- Отображение таблиц с возможностью изменения видимой области, если вся информация не может быть сразу выведена на экран.
- Редактирование текстовых значений в ячейках.
- Изменение высоты строк и ширины столбцов.
- Организацию фиксированных строк и столбцов, не перемещаемых во время прокрутки видимой области.

Компонент **StringGrid** представляет таблицу, содержащую строки и столбцы. Назначение компонента – отображение в таблице текстовой информации. В **StringGrid** возможно отображение и графической информации. Таблица может иметь полосы прокрутки, при этом заданное число первых столбцов и строк может не прокручиваться. Свойства **FixedCols**, **FixedRows**, **FixedColor** определяют число фиксированных столбцов и строк, а также цвет фона фиксированных ячеек таблицы.

Вертикальная стрелка указывает на фиксированный 0- й столбец, а левая – на фиксированную нулевую строку (*рис. 2.2.3*).

Свойства **ColCount и RowCount** определяют соответственно количество столбцов, строк в таблице. **ColWidths** – ширина для каждого столбца (по умолчанию в пикселях), **RowHeight** –высота строки, применяемая при инициализации таблицы и добавлении ячеек. В данной схеме размещения на рисунке 2.2.3 присутствует три столбца (0, 1, 2) и 12 (0 – 11) строк.

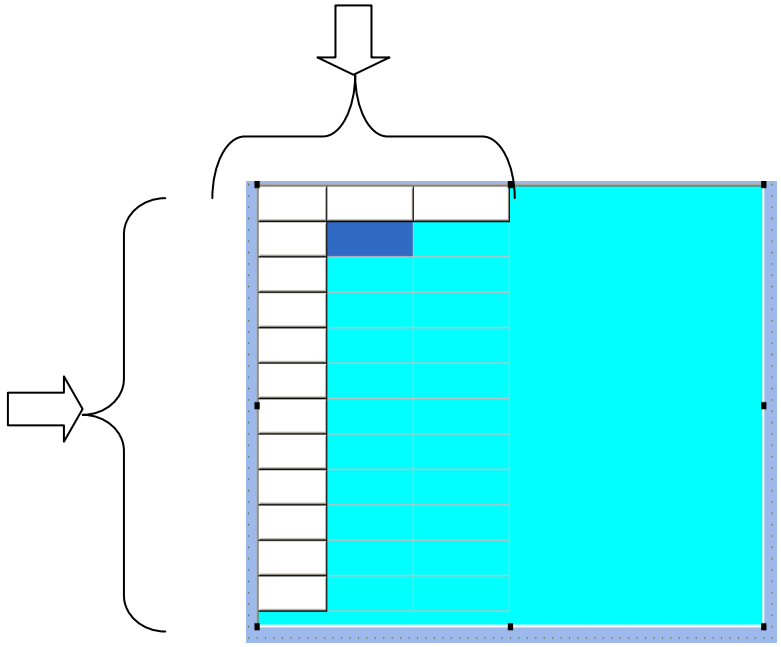


Рис. 2.2.3. Представление фиксированных строк и столбцов в таблице

Свойство **Options** представляет собой множество свойств таблицы, определяющих поведение строк и столбцов таблицы, такие как: разделительные линии в фиксированных и не фиксированных ячейках, перемещение столбцов и строк и т. д. В **Options** важным свойством является **goEditing** со значением = **True/False**, позволяющим редактировать/не редактировать значения в ячейках таблицы. В основном компонент **StringGrid** используется для обработки значений, размещенных в ячейках. Для этого используются свойства **Col и Row,** определяющие индексы столбцов и строк.

Свойство **Cells[ACol,ARow:integer]:string** – двумерный массив элементов таблицы размером **ColCount * RowCount,** нумерующийся с 0.

Например, **NewTabl.Cells[0,0]** указывает на ячейку в нулевом столбце и нулевой строке таблицы с именем **NewTabl.** А **Cells[i,j]** обозначает ячейку в i й колонке (столбце) и j –й строке.

Рассмотрев основные компоненты ввода/вывода, следует обратить внимание на классификацию типов и на функции преобразования типов (*Приложение А*).

Основные компоненты ввода/вывода текстовой и графической информации

Основные компоненты ввода/вывода текстовой и графической информации с пояснениями представлены ниже в таблицах (*табл. 2.2.1 – 2.2.4*).

Таблица 2.2.1

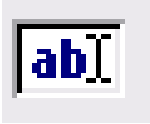
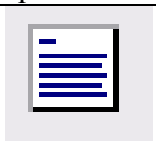
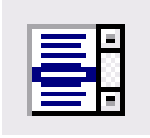
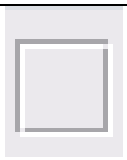
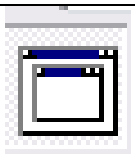
Страница Standard		
Компонент	**Тип**	**Описание**
 Метка	**Label**	Основное свойство – Caption. Доступно изменение цвета метки и шрифта. Используется для размещения текста на формах и других контейнерах.
 Поле редактирования.	**Edit**	Основное свойство – Text. Используется для отображения однострочных текстовтовых данных.
 Окно многострочного редактора текста.	**Memo**	Основное свойство – Lines Используется для отображения многострочных текстовых данных.
 Окно списка.	**ListBox**	Основное свойство – Items. Позволяется выбор элементов списка.
 Панель	**Panel**	Используется как контейнер для группирования органов управления и других малых контейнеров.
 Фрейм	**Frame**	Используется как контейнер других компонентов.
 Линейки прокрутки (скроллинга)	**ScrollBar**	Используется для создания зон отображения с прокруткой.
 Окно списка с полем редактирования.	**ComboBox**	Основное свойство – Items. Объединяет функции ListBox и Edit.
 Группа радиокнопок.	**RadioGroup**	Используется для создания групп радиокнопок.

Таблица 2.2.2

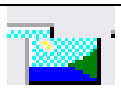
Страница Additional		
Компонент	**Тип**	**Описание**
 Таблица	**DrawGrid**	Используется для отображения изображений в строках и столбцах таблицы.
 Таблица	**StringGrid**	Таблица с массивом строк и столбцов. Основное свойство – **Cells.**
 Кнопки	**BitBtn SpeedButton**	Кнопка с надписью, значком (BitBtn), SpeedButton с зависимой, независимой фиксацией. Могут объединяться в группу при одной нажатой кнопке.
 Изображение	**Image**	Для рисования и вывода изображений из графических файлов (**BMP,WMF,JPG**).
 Диаграммы и графики	**Chart**	Для создания диаграмм и графиков.

Таблица 2.2.3

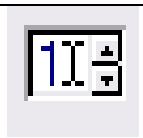
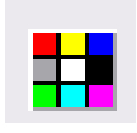
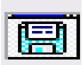
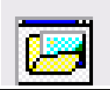
Страница Samples		
Компонент	**Тип**	**Описание**
 Окно редактирования со счетчиком.	**SpinEdit**	Окно редактирования в комбинации с кнопкой – счетчиком.
 Календарь	**Calendar**	Отображение календаря на указанный месяц в стандартном формате.
 Таблица цветов	**ColorGrid**	Для создания таблицы цветов, из которой можно выбрать требуемый цвет.

Таблица 2.2.4

Страница Dialogs		
Компонент	**Тип**	**Описание**
	**OpenDialog**	Диалог для выбора имени файла.
	**SaveDialog**	Диалог для выбора имени файла при записи.
	**OpenPictureDialog**	Диалог для выбора имени графического файла.
	**SavePictureDialog**	Диалог для выбора имени графического файла при записи.
	**ColorDialog**	Диалог для выбора цвета.

Между содержимым окон (компонентов) формы и кода существует неразрывная связь, которая строго отслеживается средой.

Отдельные экземпляры компонентов, вставляемые в форму, получают имя класса с порядковым числовым индексом. По используемому в Delphi соглашению все имена классов начинаются с буквы **T**. Например, **TForm1** от стандартного класса **TForm**, **Button1** –экземпляр стандартного класса **Tbutton** и т. д.

Пример описания класса

```
type
 TForm1 = class(TForm)
   SpeedButton1: TSpeedButton;
   BitBtn1: TBitBtn;
   Button1: TButton;
   Button2: TButton;
   Memo1: TMemo;
   Chart1: TChart;

Procedure TForm1.SpeedButton1Click(Sender:TObject);
Procedure TForm1.BitBtn1Click(Sender:TObject);
Procedure TForm1.Button1Click(Sender:TObject);
Procedure TForm1.Button2Click(Sender:TObject);
,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
End;
```

ПОЛЯ

МЕТОДЫ

Для связи подпрограмм с классом, методами которого они являются, название класса указывается перед именем самой процедуры и отделяется от него точкой, **TForm1**.

В описательной части класса могут располагаться заголовки обычных процедур и функций. В этом случае процедуры и функции (в отличие от методов) могут обращаться к свойствам класса без указания объекта.

Пояснение к предыдущему примеру описания класса:

Строка **TForm1 = class(TForm)** определяет новый класс **TForm1,** созданный по образцу стандартного класса **TForm,** который описывает пустое Windows – окно, а

TForm1 описывает окно с уже вставленными в него компонентами (поля). При этом, левая часть показывает компонент, а после двоеточия - принадлежность к его стандартному классу. Как видно из фрагмента, в форму внедрены 4 кнопки (экземпляры), имеющие свой стандартный класс.

Компонент **SpeedButton1** (кнопка) - экземпляр класса **TSpeedButton.**

Компонент **BitBtn1**(кнопка) - экземпляр класса **TBitBtn.**

Класс **TButton** в данном примере имеет два экземпляра - **Button1**, **Button2**.

Компонент **Memo1** (многострочный редактор) – экземпляр класса **TMemo**.

Компонент **Chart1** (диаграмма) - экземпляр класса **TChart.**

Каждый из выше описанных методов (заголовки процедур обработки событий), автоматически попадает в раздел описания типов, как только будет произведен двойной щелчок по объекту (в данном фрагменте – по кнопкам), либо **OnClick - в Events** (Инспектора объектов среды).

Рассмотрим один из заголовков методов описания класса -

Procedure TForm1.Button2Click(Sender:TObject);

Вслед за именем процедуры **TForm1.Button2Click** располагается в скобках параметр вызова **Sender:TObject**, передающийся во все обработки событий и имеющий тип **Tobject**. **Tobject,** является ключевым классом (*рис. 2.2.2*), наследниками которого являются все классы в Delphi – программе, в том числе и компоненты. **Tobject** обеспечивает фундаментальные основы поведения всех объектов, инкапсулируя методы, которые:

- Создают и разрушают экземпляры класса, реагируя на создание и разрушение.
- Возвращают информацию об объектах и классах и времени выполнения.
- Поддерживают обработку событий объектами.
- Поддерживают интерфейсы, реализуемые классами.

Данный пример наглядно свидетельствует об объединении данных с методами их обработки, что и является инкапсуляцией.

Глобальные объекты.

С любой запускаемой программой связываются от 2 до 5 глобальных объектов:

- **Application** (Программа)
- **Screen** (Экран)
- **Printer** (Принтер)
- **Session** (Сеанс)
- **Clipboard** (Буфер обмена).

Все указанные объекты объявляются глобальными переменными в наиболее часто используемых модулях:

Application и **Screen** - в модуле **Forms**,

Printer – в **Printers**,

Session – в **DBTables**,

Clipboard – в **Clipbrd**.

Основы создания приложений

При проектировании любого приложения Delphi строит программу, основываясь на модульном принципе. Конструирование компонентов осуществляется в окне дизайнера, связанном с формой (**Form1** – Окно формы), называемой главной и принадлежащей конкретной программе. Однако, программа может использовать и другие формы для решения локальных в ней задач. Окно формы, как и другие окна Windows, имеет заголовок. Оно является контейнером для внедрения в него компонентов, для их удобного размещения окно покрыто сеткой.

2.2.3. Базовые понятия программирования в Delphi

Программирование в **Delphi** состоит в разработке процедур (подпрограмм) обработки событий при создании проектов, при этом **Delphi** организовывает взаимодействие подпрограмм.

Проект и его состав.

Проект (программа в Delphi) – это разрабатываемое приложение. Проект состоит из форм, модулей, установок параметров проекта, ресурсов и т. д. Вся эта информация размещается в файлах (*рис. 2.2.3*), часть из которых автоматически создается при разработке приложения**.**

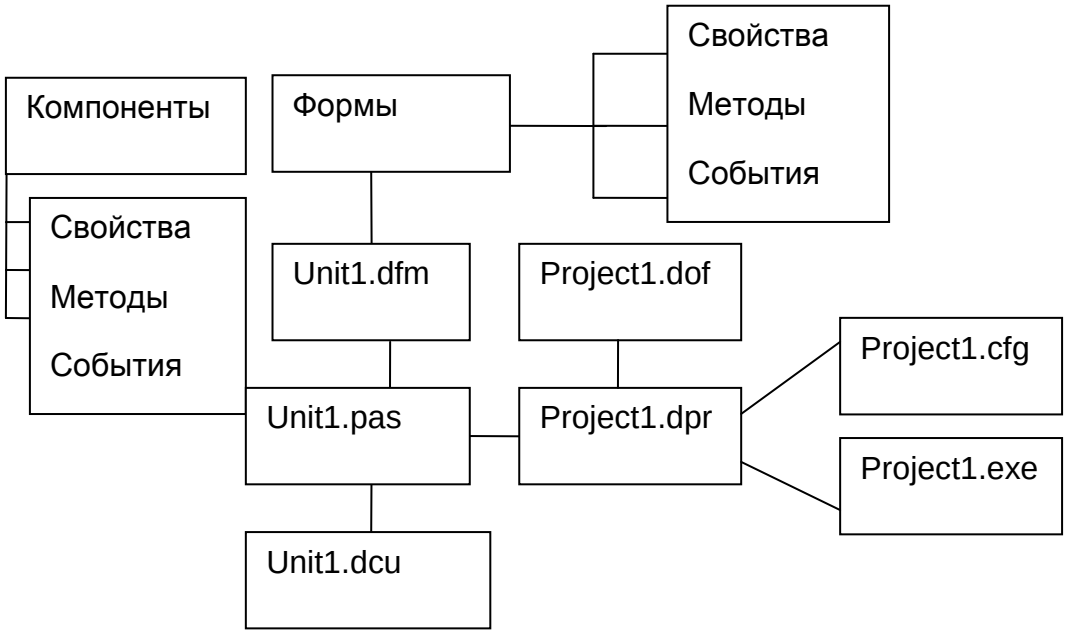


Рис. 2.2.3. Структура проекта Delphi

Программные файлы

Delphi создает множество файлов для одного приложения с различными расширениями. Часть из них имеет имя проекта, другая – имя модуля. При этом, первоначально файлам присваиваются имена **Project1** и **Unit1** с указанными в схеме расширениями, При сохранении их следует изменить на имена по смыслу разрабатываемого проекта. Файл проекта имеет расширение **dpr** и хранит код головной программы. С его помощью создается объект главной формы программы и обеспечивается связь программы с ядром Windows. Файлы с расширениями **exe**, **res**, **cfg**, **dsk**, **dof** имеют имя проекта. Пусть первоначальное имя проекта **Project1** сохранено под именем **ProjecUrok1**, тогда файлы с указанными расширениями, имеет следующие значки файлов (*рис. 2.2.4*):

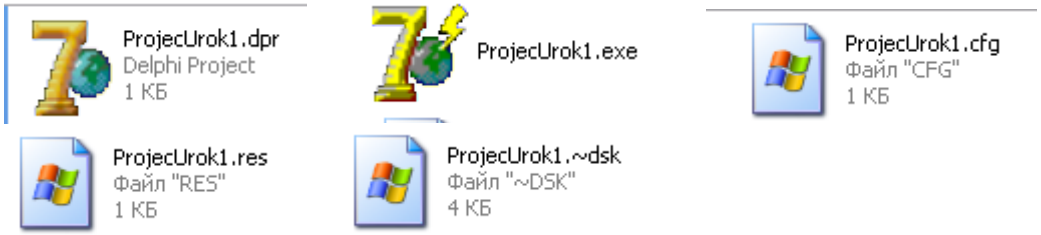


Рис. 2.2.4. Пиктограммы файлов проекта

Файлы с расширениями:

- **dpr** – читаемый файл,
- **exe** - загрузочный (исполняемый),
- **res** – создается автоматически для каждого проекта, содержит значок программы, ее версию и прочие данные. Этот файл подсоединяется к исполняемому файлу на этапе компоновки с помощью директивы **{$R *.res}**, которая содержится в головной программе.

Файлы (текстовые) с расширениями **cfg, dsk**, **dof** сохраняют параметры настройки:

- **cfg** (компиляции),
- **dsk** (проекта),
- **dof** (среды).

При внесении в проект изменений создаются файлы резервных копий, начинающихся со значка "тильда" **–** "~"**?**.

Например, значок **ProjecUrok1.~dsk** (*рис. 2.2.4*).

Файл проекта.

Файл проекта – это главная программа, автоматически создаваемая Delphi и написанная на языке Delphi. Она содержит всегда несколько строк и не допускает редактирования. Именно она обрабатывается компилятором. В Delphi (как и в любом языке программирования) началом программной единицы выступает заголовок, а концом ее записи слово **end**.

Структура главной программы проекта с именем Project1

```
program Project1;
uses
Forms,
Unit1 in 'Unit1.pas' {Form1};
{$R *.res}
begin
Application.Initialize;
Application.CreateForm(TForm1, Form1);
Application.Run;
end.
```

Выделенные зарезервированные (служебные) слова **Program, uses, in, begin, End** служат информацией компилятору о начале и конце обработки программы сверху вниз. При этом **end.** (т. н. терминатор) указывает на то, что все, расположенное за **end** с точкой, игнорируется. Информация, расположенная в фигурных скобках, представляет комментарий, например - **{Form1}**. В качестве комментария могут выступать следующие знаки: {фрагмент текста от скобки и до скобки }, (* фрагмент текста *), // фрагмент текста до конца строки.

Как видим, каждое предложение заканчивается знаком точка с запятой.

После заголовка программы следуют строки,

```
uses
Forms,
Unit1 in 'Unit1.pas' {Form1};
```

Они указывают на то, что в данной программе будут использованы модули **Forms** (известный компилятору) и Unit1 (новый), поэтому указывается файл с текстом модуля **in** 'Unit1.pas' и имя связанного с модулем файла описания формы **Form1**.

Директива {$R *.res} не является комментарием. Она указывает компилятору о необходимости подключения файла ресурсов. Директивы начинаются со знака $, стоящего вслед за фигурной скобкой.

Далее следует тело процедуры, заключенное в операторные скобки (слова **begin** – начало и **end.** - конец):

```
begin
```

Application.Initialize;
Application.CreateForm(TForm1, Form1);
Application.Run;
end.

Тело процедуры – исполняемые операторы с обращением к трем методам объекта **Application**. Delphi автоматически создает объект – программу **Application** для каждого нового проекта. Первый оператор выполняет переход к некоторому фрагменту подпрограммы Delphi, позволяющий осуществить переход к строке

Application.CreateForm(TForm1, Form1);

Метод **CreateForm** показывает на экране окно главной формы, метод **Application.Run** выдает сообщения Windows о действиях пользователя.

Модуль и его структура

Модуль (модули) - автономно компилируемая программная(ые) единица(ы) с включением различных компонентов интерфейсного раздела. Основная работа программы в соответствии с заданным алгоритмом управляется кодом в модуле (модулях).

Состав модуля:

- Заголовок;
- Интерфейсные объявления;
- Реализация;
- Окончание

Заголовок. Определяется зарезервированным словом **Unit** и следующим за ним именем модуля. Первоначально имя – **Unit1**, при сохранении модуля имя следует определить по смыслу. Если предусмотрено несколько модулей, то заголовок имеет вид - **Unit** <список модулей>, в котором имена разделяются запятыми. На первом этапе программирования в Delphi мы будем использовать один модуль простейшей структуры:

Unit <имя>;
Interface –
<интерфейсный раздел>
implementation
<исполняемая часть>
end.

Раздел **Interface.** Сюда помещаются списки подключаемых модулей, объявления типов, констант, переменных, процедур и функций, к которым будет доступ из других модулей. Подключение модулей отображается в предложении **Uses**. Если алгоритм разрабатываемого проекта предусматривает обращение к статистическим, математическим функциям или, например даты/времени, тогда в предложение **Uses** необходимо подключить соответственно модуль **Math**, а для дат – **DateUtils** и др.

В разделе **Implementation** располагается код программы, он может включать в себя процедуры обработки событий (одну или несколько), процедуры общего вида, а также процедуры – функции.

end. - зарезервированное слово (признак конца модуля) с точкой – терминатор.

Модули

Форма – модуль, интерфейсная часть которого включает объявление нового класса, объявление объекта для соответствующего оконного класса.

Файлы модулей имеют расширения: **dfm, pas, ddp, dcu.** Файлы с текстами модулей имеют расширения **pas**. Откомпилированный файл машинного кода получает расширение **dcu.** Файл с расширением **dfm** создается на диске после сохранения модуля. В нем сохраняются все свойства компонентов окна. Файл с расширением **ddp** сохраняет сведения обо всех диаграммах окна. Среди указанных файлов модулей файлы с расширениями **dfm и pas** являются для проекта наиболее важными.

Ниже представлены значки файлов модуля с именем **UnUrok1** и представленными выше расширениями (*рис. 2.2.5*).

Рис. 2.2.5. Пиктограммы файлов модуля

2.2.4. Структура программ

В **Delphi,** как и в **Object Pascal,** основной программной единицей является подпрограмма. Различают два вида подпрограмм: процедуры и функции. Отличие состоит в том, что с именем функции связано значение, поэтому имя функции можно использовать в выражениях. В соответствии с заданным алгоритмом могут использоваться как процедуры общего вида, так и процедуры обработки событий.

Программа – последовательность строк, где строка может располагаться с любой позиции экрана. Структурно программа состоит из заголовка и блока. Заголовок располагается в начале программы и имеет вид:

Program <Имя программы>

Блок содержит описательную и исполнительную части. В описательной части содержатся описание элементов программы, а в исполнительной части указываются действия над элементами программы, выполнение которых должно привести к нужному результату. Общий случай описательной части содержит следующие разделы:

- Подключение модулей.
- Объявление меток.
- Объявление констант.
- Описание типов данных.
- Объявление переменных.
- Описание процедур и функций.

Каждый из указанных разделов заканчивается точкой с запятой.

Структура процедуры

Procedure Имя(СписокПараметров);
const
объявление констант);
type
(объявление типов);
var
(объявление переменных);
begin
(последовательность исполняемых инструкций);
end;

При создании приложений и записи программ значение имеет соблюдение основных правил.

2.2.5. Правила оформления приложений и программ

Под каждое приложение должна быть выделена отдельная папка, в которой размещаются все файлы приложения, при этом, имена модуля и проекта должны быть осмысленными словами английского языка (как и все другие элементы процедур).

Правила записи программ

- Каждая исполняемая инструкция (оператор) может быть записан в одной или нескольких строках без повторения знаков при переносе. Концом оператора является знак точки с запятой «;». Целесообразнее же располагать в строке только один оператор.
- В операторе присваивания левая часть от правой части отделяется знаком «:=» (присваивания**).**

- Весьма важным инструментом выступает составной оператор, который представляет последовательность операторов, заключенных в операторные скобки – **begin** … **end**, внутри которых могут использоваться вложенные операторные скобки, например в циклах, в условных переходах, множественном выборе. Использование составных операторов позволяет структурно строить программу.
- При записи текста процедур должна быть соблюдена ступенчатая структура текста.

begin	repeat	while…do	for…to…do	if…then	case…of
……..	……	begin	begin	begin…….	……
end;	until…;	…….	……	end	……
		end;	end;	else	end;
				begin…….	
				end;	

- Повторяющийся код программы в большинстве случаев должен быть оформлен в виде отдельной процедуры или функции.
- Текст программы должен быть снабжен подробными комментариями и начинаться с них. Комментариями считается:
  - любой текст, заключенный символами // и концом строки.
  - любой текст, заключенный между символами {..} в одной строке или в - нескольких строках

{
….
}.

  - любой текст между открывающими и закрывающими символами языка: (*..(..)..*), ('..').

Комментарии не влияют на смысл программы, а служат для пояснения того или иного действия в программе. Комментарии могут использоваться для включения инструкций во время отладки программы без удаления их из текста. Впоследствии для восстановления выключенного фрагмента достаточно снять фигурные скобки.

2.2.6. Элементы программы

Элементы программы – ее неделимые части, распознаваемые компилятором (как и в прочих языках программирования).

Зарезервированные (служебные) английские слова

Слова определяют начало какого либо действия (оператора, команды). Они дают информацию компилятору о назначении той или иной синтаксической конструкции. Их можно представить следующими группами:

- Заголовки – **program, unit, procedure, function.**
- Блоки описания –**const, var, label, type.**
- Создание новых типов – **array, string, record…end, file, file…of.**
- Операторные скобки **– begin…end.**
- Операторы **– if…then…else…; case…of…end; for…to…do…; for…downto…do…; repeat…until…; while...do…; with…do…**
- Директивы **– export, Public, published, virtual, protected, index…**Их список достаточно широк.
- Слова со специальным назначением – **at**, **on**.

Имена (идентификаторы), как известно, имена в программе используются для обозначения переменных, констант, процедур, функций, типов и самих программ. Это последовательность, начинающаяся с латинских символов с возможностью включения цифр и знака подчеркивания «_», при этом длина имени не ограничена (одна из возможностей современных ОС), но распознается по первым 255 символам. Перед использованием имени в программе оно должно быть объявлено (описано).

Внимание. Не использовать имена, совпадающие с зарезервированными (служебными) словами.

Константы. Могут быть с использованы в выражениях, операторах программы в виде констант (именованные) и константных выражений.

Именованные константы объявляются с помощью ключевого слова **const** и имеют следующий синтаксис:

Const <идентификатор константы>=< константное выражение >;

Например. **Const** S=’ Моя группа’+’ 3-МД-10’;

Переменные. Переменная является идентификатором, обозначающим некоторую область памяти, в которой хранится ее значение. Переменная может менять значение в ходе выполнения программы. Объявление переменной имеет вид:

Var

<список идентификаторов переменных, разделенных запятыми**>:**<тип>;

Например**.**

Var

i, j, n: integer;

c,f :real;

Выражения – строка, состоящая из последовательности операндов.

Операнды – это константы, переменные, указатели различных дозволенных в языке функций. Тип выражения определяется типом входящих в него операндов.

Основные сведения:

- операции над операндами в выражении выполняются в строго определенном порядке;
- значение каждого операнда имеет определенный тип;
- значение каждого выражения имеет определенный тип;

Выражения составляют неотъемлемую часть операторов.

Delphi отличается большим разнообразием различного рода функций (математических, статистических, даты и времени, строковых), используемых в выражениях, часть из которых обеспечивается модулем **Match**. Для подключения этих функций необходимо сослаться на их имя со списком параметров в выражении инструкции, а в предложении **Uses** разрабатываемого модуля указать его имя.

Стандартные процедуры и функции для выполнения операций над значениями порядковых типов

Inc(X[,N]) – увеличивает X на N или на 1, если N опущено.

Dec(X[, N]) –уменьшает X на N или на 1, если N не указано.

ODD(x) – результат True / False (число нечетное / четное).

Примечание. Аргумент данных функций должен иметь целый тип.

Основные стандартные арифметические функции:

ABS(аргумент) – абсолютное значение аргумента,

Sqrt(аргумент) – квадратный корень аргумента,

Sqr(аргумент) – квадрат аргумента,

Sin(аргумент) – синус аргумента,

Cos(аргумент) – косинус аргумента,

Arctan(аргумент) – арктангенс аргумента,

Exp(аргумент) – экспонента аргумента,

Ln(аргумент) – натуральный логарифм аргумента.

Через указанные функции можно выразить такие функции как: **Tan(**аргумент**), ArcSin(**аргумент**), ArCos(**аргумент**), Loga(**аргумент**).**

Функции даты и времени

YearsBetween, **YearSpan** возвращают число лет между двумя значениями даты и времени.

Функции преобразования

Кроме стандартных функций **Delphi** обеспечивает множеством иных функций, таких как функции преобразования:

Trunc(n) – целая часть от n,
Frac(n) – дробная часть вещественного n**,**
Int(n) – целая часть вещественного n **,**
Chr(n) – символ ASCII с номером n,
Round(n) – генерируется случайная величина от 0 до целого n.
Random() – случайная величина из диапазона [0,1].

Основные функции преобразования строки в целое и вещественное и наоборот

FloatToStr(Value) - преобразует вещественное число в строку.
FloatToStrF(Value) - преобразует вещественное число в строку с форматированием.
IntToStr(Value) - преобразует целое число в строку.
StrToInt(value) - -преобразует строку в целое число.
StrToFloat(value) - преобразует строку в вещественное число.

Примечание. В качестве **Value** могут выступать имена переменных, массивов, ячейки таблиц, значения строк списка, значения ячеек таблиц, значения редакторов.

Операции

Арифметические -), *, /,div, mod, +,-, sh1, shr **.** Тип **Integer** или **Rea**l.

Логические – not, or, xor, and **.** Тип **Boolean.**

Операторы отношения - , =, >, >=, <, <=, <>. Тип **Boolean**.

Операторы строк. Для строк применимы также операторы отношений. Две строки могут быть соединены знаком « +». Тип результата **String**.

Операторы. Классификация

Виды операторов:

1. Операторы присваивания;
2. Оператор условного перехода - **if**;
3. Оператор выбора - **Case**;
4. Операторы цикла**: While; Repeat; For.**

Оператор присваивания

Общий вид записи:

<имя >:=<выражение>;

В качестве имени может выступать имя переменной, элемент массива, имя процедуры-функции.

Управляющие операторы

Оператор условного перехода If

Оператор служит в первую очередь для организации условных переходов.

Оператор **if** имеет две формы**:**

a) **If …Then**
b) **If …Then… Else**

Логика работы всех перечисленных операторов известна из ранее изучаемых языков программирования, остановимся на разнице в синтаксисе.

Общий вид записи:

```
If  <условие>
   Then
        Begin
            Операторы для выполнения, если условие истинно
         End
   Else
        Begin
```

Операторы для выполнения, если условие ложно

End;

Необходимо помнить, что выражение в условии имеет всегда логический тип. **Begin и End** выполняют роль операторных скобок, заключающих в себя операторы той или другой ветви, если операторов больше одного. Перед **Else знак «;»** не ставится.

Оператор выбора Case

Оператор **Case** позволяет выполнить множественный выбор.

Общий вид записи:

Case <выражение> of
<Список 1>: **begin**
< Операторы 1>
end;
<Список 2>: begin
< Операторы 2**>**
end
<Список N>: begin
< Операторы N>
end
……
else
begin
< Операторы >
end;
end;

Выражение – любое выражение порядкового типа. Строковый тип не разрешен.

Список значений может включать:

1. число;
2. объявленная константа;
3. другое выражение;
4. диапазон значений, состоящий из начального и конечного значений, например, 1..7;
5. список вида: элемент_1, …, элемент_N, в котором каждый элемент удовлетворяет всем предыдущим условиям.
6. Наличие операторных скобок, как и в If зависит от одного или нескольких операторов в конкретном списке.

Операторы цикла

Операторы цикла представлены:

1. Оператором с предусловием **While,** который имеет общий вид записи:

While <условие>**do** <оператор>**;**

Если в цикле выполняется несколько операторов, они заключаются в операторные скобки – **Begin … End.**

2. Оператором с постусловием **repeat**, который имеет общий вид записи:

repeat
<оператор_1>;
<оператор_2>;
…….
<оператор_n>**;**
Until <Условие>;

3. Оператором с заданным числом повторений – f**or,** который имеет следующие формы:

а) цикл на возрастание

for <**счт**>:=<**нзн**> **to** <**кзн**> **do** < оператор >;

б) цикл на убывание

for < **счт** >:=< **нзн** >**downto** < **кзн** > **do** <оператор >;

где

счт (счетчик**)** – переменная порядкового типа.

нзн (нач. значение), кзн (кон. значение) – выражения того же типа, что и **счт**.

В форме записи – **а):**

нзн > кзн.

В форме записи – **б):**

нзн < кзн.

Как и в операторе **While**, используется составной оператор, если операторов внутри цикла несколько.

Примеры программных кодов с использованием разветвленных и циклических структур для вычисления суммы (summa) для разной организации циклов

Пример 1.

Используются операторы циклов **for**.

```
procedure TForm1.Button1Click(Sender: TObject);
 var
summa,n,i:integer;
 begin
  n:=strToInt(Edit1.Text);
 summa:=0;
 if n>=0 then
 for i:=1 to n do
  summa:=summa+i
  else
  for i:=-1 downto n do
    summa:=summa+i;
    Edit2.Text:=IntToStr(summa);
  end;

end.
```

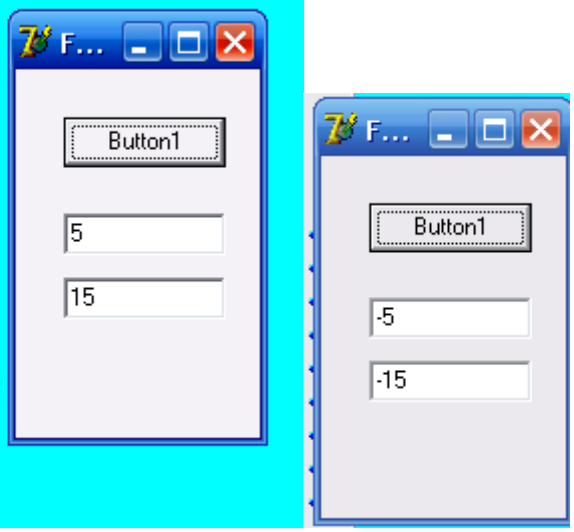


Рис. 2.2.6. Фрагмент модуля, содержащего процедуру обработки события нажатия кнопки для вычисления суммы

Далее в примерах пользуются операторы циклов **repeat** (с постусловием) **и While (**с предусловием).

Пример 2.

Заполнение каждого третьего элемента списка случайными величинами в цикле типа **repeat.**

Примечание. В процессе конструирования кнопка **« ОК»** переименована в **«Выполнить».**

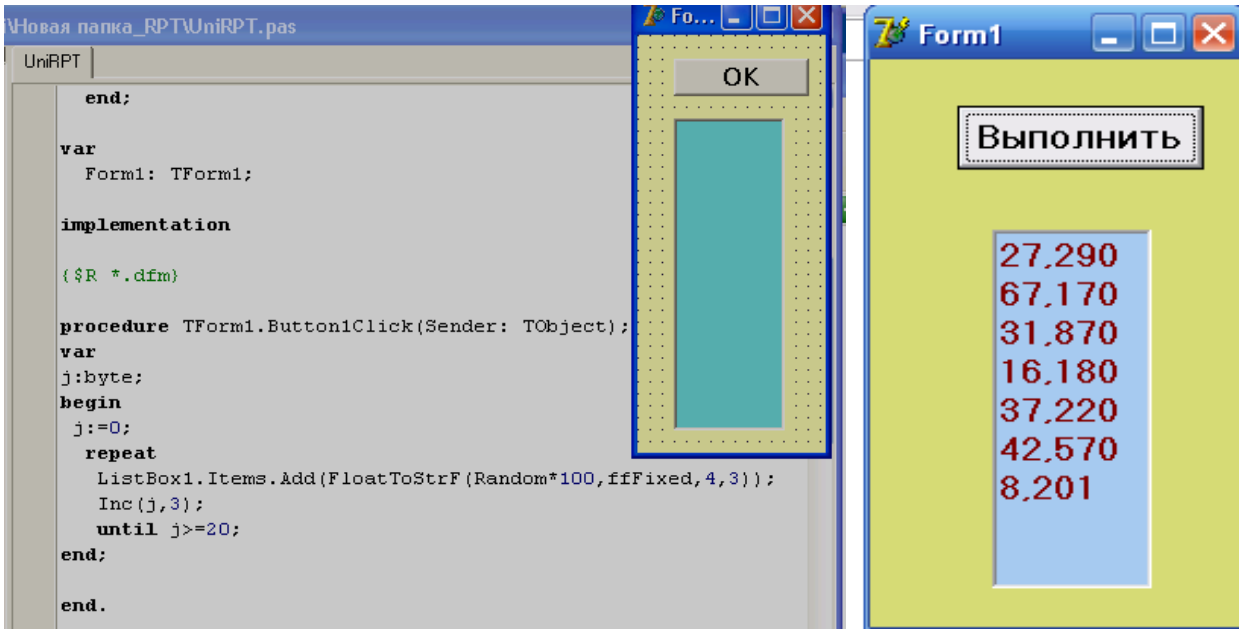


Рис. 2.2.7. Фрагмент модуля, содержащего процедуру обработки события нажатия кнопки

Пример 3. Заполнение в цикле типа **While** каждой второй строки **Memo** случайными числами**.**

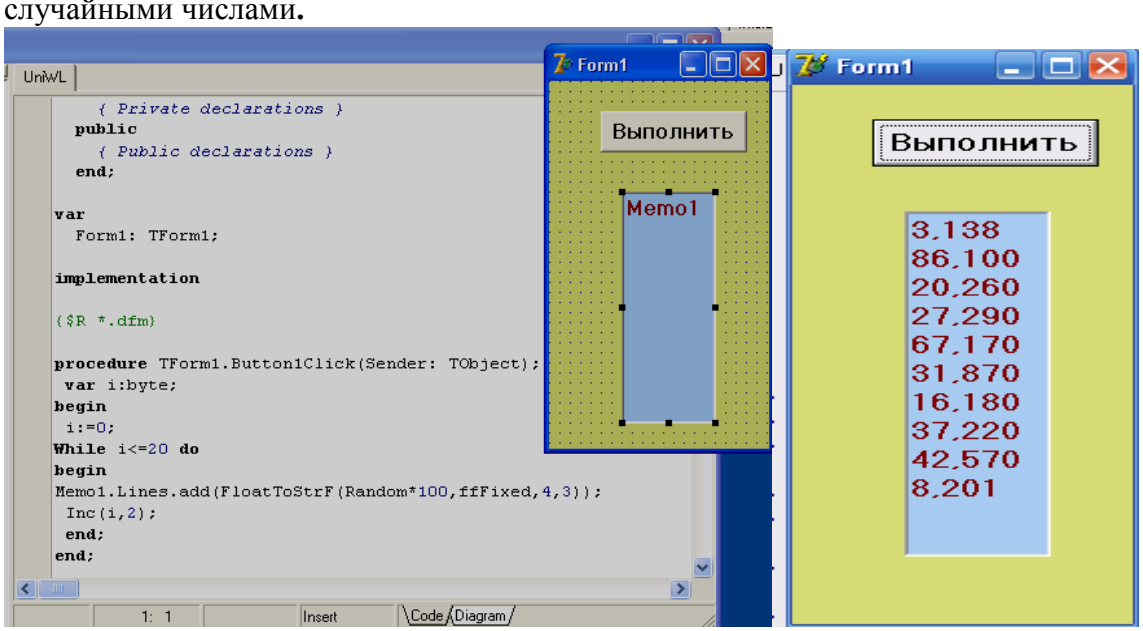


Рис. 2.2.8. Фрагмент модуля, содержащего процедуру обработки события нажатия кнопки

Демонстрационный пример

Решение задачи от начала проектирования до получения результатов

Постановка задачи. Пусть необходимо создать проект, построенный на форме, для определения результата вычитания значения из первого числа (уменьшаемого) значения второго числа (вычитаемого). Числа целого типа.

Реализация поставленной задачи в среде Delphi 7

Для реализации данной постановки задачи выполняем:

- Загружаем **Delphi 7** (*рис. 2.1.1*);
- Выбираем **Файл, Новый, Приложение** (*рис. 2.1.2*);

- Открывается окно пустой формы (*рис. 2.1.3*), в которое внедряем следующие компоненты по прилагаемому макету (*рис. 2.2.9*).
  - ✓ 3 метки - **Label1, Label2, Label3**, которым в свойстве **Caption** даем подпись («Уменьшаемое», «Вычитаемое», «Разность»).
  - ✓ 3 поля - **Edit1, Edit2, Edit3**, которым в свойстве меняем имена свойством **Name** на EdWYCH1, EdWYCH2, EdRes. В программном коде уже используются новые имена.
  - ✓ 2 кнопки - **Button1, BitBtn1**. Для **Button1** даем подпись в свойстве **Caption** «Вычислить разность» с переносом слов, для этого свойство **Wordwrap** в окне **Инспектор объектов** устанавливаем = **True**. Для кнопки **BitBtn1** выбираем свойство **Kind** и из списка выбираем **bkclose**, что имеет значок закрытия (*рис 2.2.10*).

После конструирования формы первая часть модуля должна иметь вид:

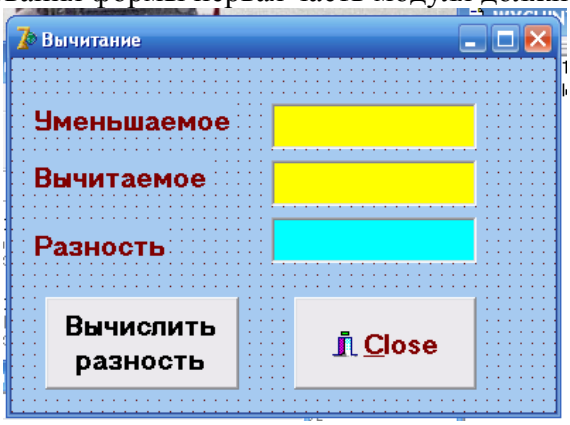


Рис. 2.2.9. Макет окна формы разрабатываемого проекта

```
WYCHINT
unit WYCHINT;
  interface
 uses
  Windows, Messages, SysUtils, Variants, Classes,
   Graphics, Controls, Forms,
  Dialogs, Buttons, StdCtrls;
 type
  TForm1 = class(TForm)
    EdWYCH1: TEdit;
    EdWYCH2: TEdit;
    EdRes: TEdit;
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    Button1: TButton;
    BitBtn1: TBitBtn;
    procedure Button1Click(Sender: TObject);
    procedure BitBtn1Click(Sender: TObject);
     private
    { Private declarations }
  public
    { Public declarations }
  end;
 var
    Form1: TForm1;
```

Рис. 2.2.10. Первый раздел модуля, включающий описание данных при внедрении

компонентов

Как отображено на рисунке 2.2.10, одновременно с конструированием формы, компоненты (как экземпляры классов) попадают в раздел описания данных в модуль с именем. После завершения конструирования формы проект желательно сохранить. При сохранении проекта модулю дадим имя WYCHINIT. В соответствии с указанной структурой программы при проектировании пользователем приложения **Delphi** создает автоматически код головной программы (*рис. 2.2.11*) и отдельных модулей.

Примечание. Код головной программы не допускает в него вмешательства.

DEISTVIJ_WYCH

```
program DEISTVIJ_WYCH;

uses
  Forms,
  WYCHINT in 'WYCHINT.pas' {Form1};

{$R *.res}

begin
  Application.Initialize;
  Application.CreateForm(TForm1, Form1);
  Application.Run;
end.
```

Рис. 2.2.11. Код головной программы

Создание процедур обработки событий

1) Двойным щелчком по кнопкам создаем заготовки процедур обработки событий – одну для вычисления разности, другую – для закрытия формы. После двойного щелчка по кнопкам их заготовки для кодов появятся в разделе кода, т. е после **implementation и {$R *.dfm}**.

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

begin
.....
end;
procedure TForm1.BitBtn1Click(Sender: TObject);
begin
.....
end;
```

2) Записываем в заготовку процедуры ее тело, начиная со слова **var** и заканчивая **end** (*рис. 2.2.12*).

Аналогично записываем тело второй процедуры, состоящее всего из одного оператора – **Form1.Close**, что закрывает форму при нажатии кнопки со значком.

3) Выполнить код программы, щелкнув по кнопке  панели инструментов.

Результат вычисления отображен на рисунке 2.2.13.

После конструирования окна формы и выполнения будут созданы файлы (*рис. 2.2.14*).

При проверке работы проекта с другими данными можно запускать его с помощью файла 

с расширением exe.

```
  implementation

{$R *.dfm}

procedure TForm1.Button1Click(Sender: TObject);
  var
C,D,Res:integer;
begin
 C:= StrToInt(EdWYCH1.Text);
 D:= StrToInt(EdWYCH2.Text);
 Res:=C-D;
 EdRes.Text:= IntToStr(Res);
end;

procedure TForm1.BitBtn1Click(Sender: TObject);
begin
 Form1.Close;
end;
   end.
```

Рис. 2.2.12. Коды процедур обработки событий по кнопкам Button1 и BitBtn1

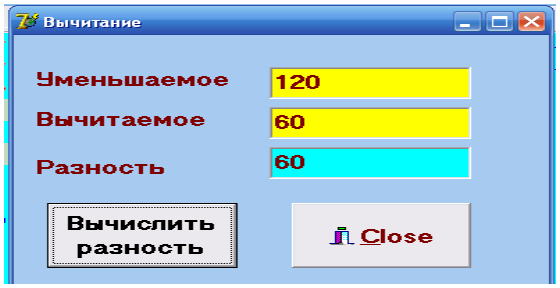


Рис. 2.2.13. Результат вычисления

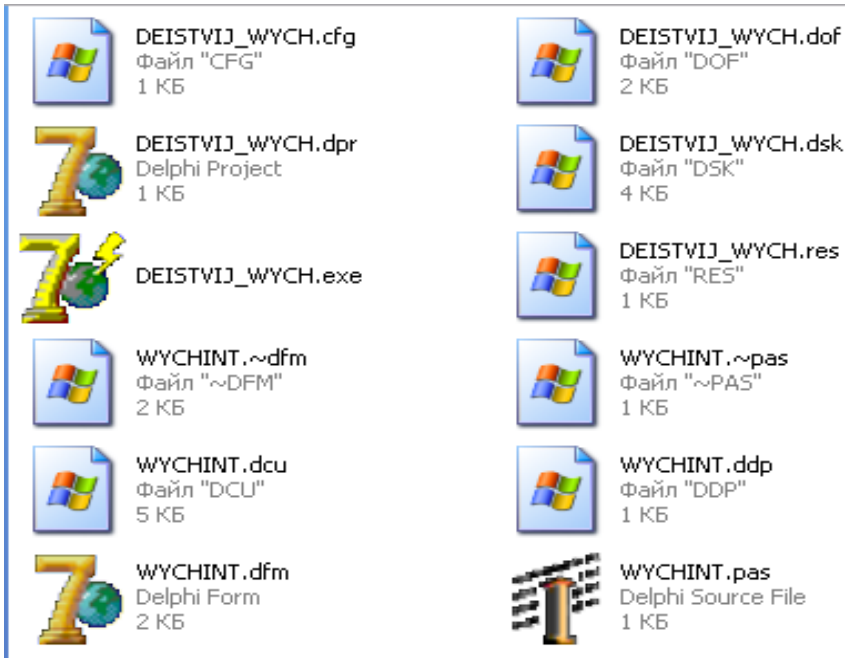


Рис. 2.2.14. Пиктограммы модулей файлов проекта

3. Создание пользовательских проектов

Общий порядок работы:

1) Создать новое приложение, соответственно представленного макета.
2) В открывшееся окно формы внедрить компоненты. Проверить правильность внедрения выделенных компонентов в окне Инспектора объектов.
3) Определить свойства каждому выделенному компоненту.
   Например.
   Кнопка **BitBtn1** имеет подпись 

   Кнопка **BitBtn2** имеет подпись - 

4) В режиме конструктора выделить последовательно каждую кнопку и щелкнуть дважды. Будут созданы заготовки для процедур обработки событий.
   Например.

procedure TForm1.BitBtn1Click(Sender: TObject);
begin
Тело процедуры

end;
procedure TForm1.BitBtn2Click(Sender: TObject);
begin
Тело процедуры

end;

5) Сохранить проект:
   1. Создать отдельную папку с именем соответственно заданию.
   2. Выполнить - Файл, Сохранить Проект как ...Unit1 под именем, например, UnDate, Project1 – ProDate.dpr.
   3. Выполнить проект, нажав на панели инструментов значок 

Если будет декларация ошибок (*Приложение В*) в нижней части окна «Редактор кода», записать текст сообщения и определить ошибку. В таких несложных проектах возникают, как правило, синтаксические ошибки по вине пользователя.

3. 1. Задания к Контрольной работе № 1

Задание 3. 1.

Создание проекта с использованием алгоритма линейной структуры для определения собственного возраста – «Возраст». Макет формы (*рис. 3.1*).

Состав компонентов в форме:

- **Memo** (1 многострочный редактор).
- **BitBtn** (2 кнопки)
- **Label** (1 метка)

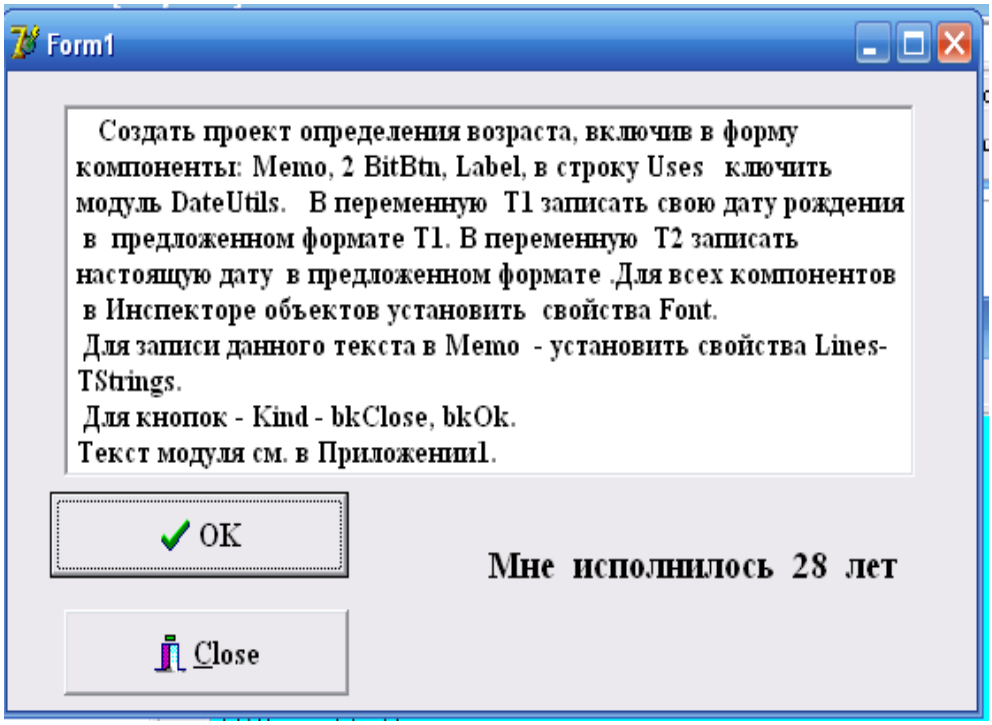


Рис. 3.1. Макет главного окна проекта «Возраст»

Порядок работы:

1) Создать новое приложение (*рис. 2.1.1, рис.2.1.2*) соответственно представленного макета;
2) В открывшееся окно формы внедрить компоненты: **Memo** (многострочный редактор), две **BitBtn** (кнопка), **Label** (метка). Для этого активировать вкладку **Additional** линейки компонентов, найти компонент **Memo** и компонент **Label**, а во вкладке **Standard** – компонент **BitBtn**.
3) Проверить правильность внедрения выделенных компонентов в окне Инспектора объектов.
4) Определить свойства каждому выделенному компоненту.
5) В компонент **Memo** занести текст с макета (в Инспекторе объектов выполнить свойство **Items, TStrings**), представленного на рисунке 3.1.
6) В режиме конструктора выделить последовательно каждую кнопку и щелкнуть дважды. Будут созданы заготовки для процедур обработки событий.

procedure TForm1.BitBtn1Click(Sender: TObject);
begin
Тело процедуры, Приложение Б, Задание 3.1

end;

procedure TForm1.BitBtn2Click(Sender: TObject);
begin
Тело процедуры, Приложение Б, Задание 3.1

end;

Кнопке **BitBtn1** определим свойство (*рис.2.1.8*) и она получит вид 

, что подразумевает «Выполнить».

Аналогично выполнить действия для кнопки **BitBtn2** (*рис.2.1.8*) и она получит вид 

что подразумевает «Закрыть форму».

7) Сохранить проект.

1. Создать отдельную папку с именем Возраст на флеш – носителе;
2. Выполнить - Файл, Сохранить проект как ...Unit1 под именем UnDate, Project1 – ProDate.dpr.
3. Выполнить проект, нажав на панели инструментов значок  .

Примечание. Если будет декларация ошибок (в нижней части окна - «Редактор кода»), записать текст сообщения и определить ошибку. В таких несложных проектах возникают, как правило, синтаксические ошибки по вине пользователя.

Задание 3. 2.

Создание проекта на основе примера «**Светофор»** (алгоритм ветвящейся структуры). Макет формы (*рис. 3.2*). Постановка задачи – в тексте компонента **Memo.**

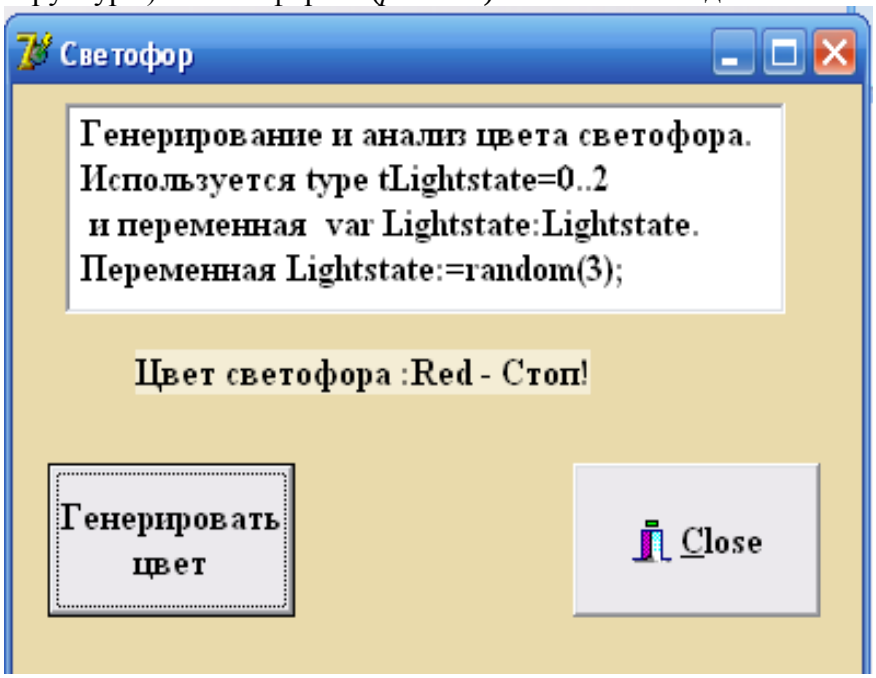


Рис. 3.2. Макет окна проекта «Светофор»

Состав компонентов в форме:

- **Memo** (1 многострочный редактор),
- **Label** (1 метка),
- **Button** (1 кнопка),
- **BitBtn** (1 кнопка со значком).

Задание выполняется аналогично предыдущему примеру.

Задание 3. 3.

Создание проекта «Скорость движения» на основе алгоритма циклической структуры. Макет формы (*рис. 3.3*).

Задание выполняется, как описано ранее.

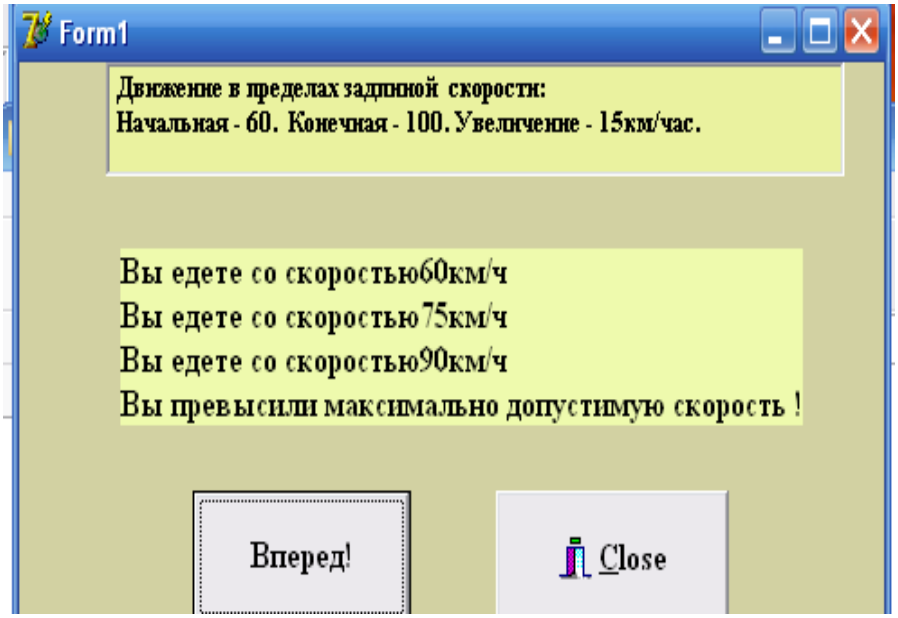


Рис. 3.3. Макет окна проекта «Скорость движения»

Задание 3. 4.

Создание проекта «Расчет суммы элементов главной диагонали матрицы» на основе алгоритма циклической структуры с использованием таблиц. Макет формы (*рис. 3.4*).

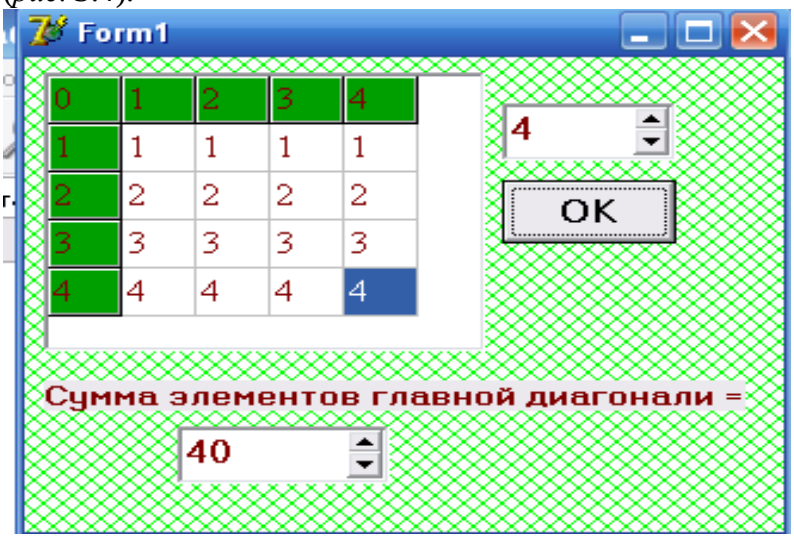


Рис. 3.4. Макет окна проекта «Расчет суммы элементов главной диагонали матрицы»

Состав компонентов в форме:

- **StringGrid** (1 таблица),
- **SpinEdit** (2 счетчика с полем),
- **BitBtn** (1 кнопка со значком),
- **Label** (1 метка).

Первым счетчиком задается значение, на которое умножается каждый элемент главной диагонали, во второй счетчик выводится сумма.

Задание 3. 5.

Создание проекта «Реализация товаров» на основе алгоритма циклической структуры с использованием компонентов **StringGrid** и **Chart** для графического отображения результата. Макет формы (*рис. 3.5*).

Состав компонентов в форме:

- **StringGrid** (1 таблица),
- **Button** (1 кнопка),
- **Chart** (1 диаграмма),
- **BitBtn** (1 кнопка со значком).

Задание выполняется, как описано ранее.

Рис. 3.5. Макет окна проекта «Реализация товаров»

Задание 3.6.

Создание проекта с графическим отображением данных в форме - «Графическое отображение данных». Макет формы (*рис. 3.6*).

Состав компонентов в форме:

- **SpeedButton** (4 кнопки),
- **Panel** (4 панели),
- **Chart** (4 диаграммы).

Рис. 3.6. Макет окна проекта «Графическое отображение данных»

Задание выполняется, как описано ранее.

Задание 3.7.

Создание проекта «Сортировка массива из случайных величин». Макет формы (*рис. 3.7*).

Состав компонентов в форме:

- **TrackBar** (1 линейка скроллинга),
- **Button** (1 кнопка),
- **Label** (4 метки),
- **Panel** (2 панели),
- **BitBtn** (1 кнопка со значком).

Примечание. В задании предусмотрены следующие процедуры обработки событий для компонентов: **BitBtn1**, **Button1**, **FormCreate**, **TrackBarChange**.

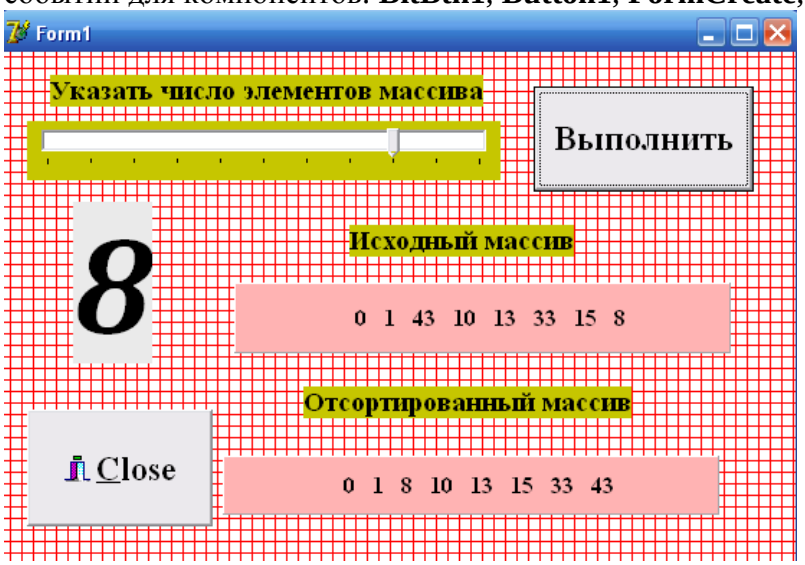


Рис. 3.7. Макет окна проекта «Сортировка массива из случайных величин»

Задание 3.8.

Создание проекта «Перевод чисел из 2 – ой системы счисления в 10 - ю и обратно». Макет формы (*рис. 3.8*).

Состав компонентов в форме:

- **Radiogroup** (1 группа радиокнопок),
- **BitBtn** (1 кнопка со значком),
- **Edit** (1 поле редактирования),
- **Label** (1 метка).

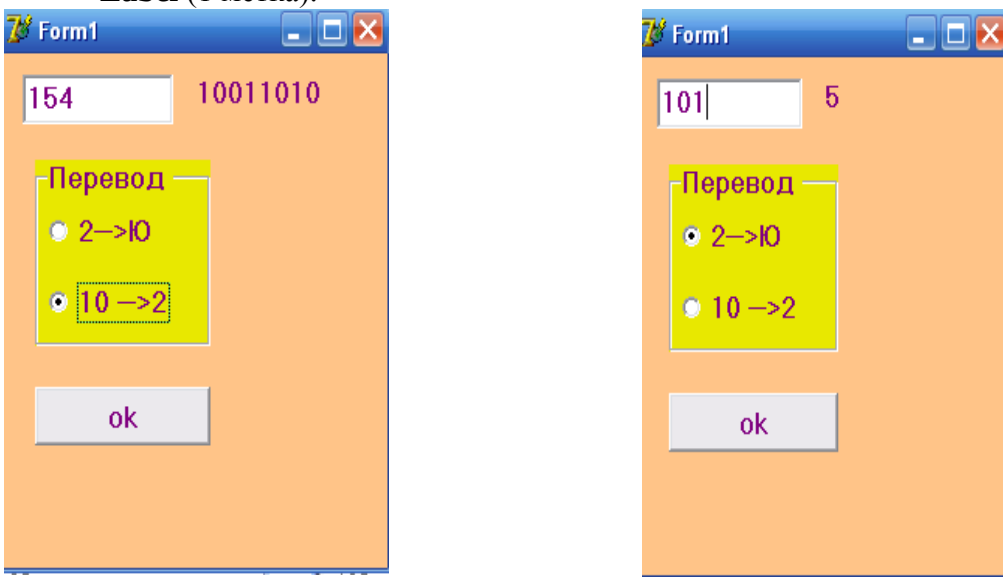


Рис. 3.8. Макет окна проекта «Перевод чисел из 2 – ой системы счисления в 10 - ю и обратно

Задание 3.9. Создание проекта «Вычисление среднего арифметического массива». Макет формы (*рис. 3.9*).

Вычислить среднее арифметическое массива, созданного на основе элементов списка **ListBox2** с именем **LstBox2**, используя статистическую функцию **Mean**.

Состав компонентов в форме:

- **ListBox1** (1 список – имя LstBox),
- **ListBox2** (1 список – имя LstBox2),
- **Label**(1 метка),
- **Edit** (1 поле редактирования),
- BitBtn (2 кнопки со значком),.

В задании создаются 3 процедуры обработки событий для кнопок **BitBtn**, формы - **FormCreate**. В предложение **Uses** добавить модуль **Math**.

Примечание. Строго отследить содержимое процедур при копировании кода.

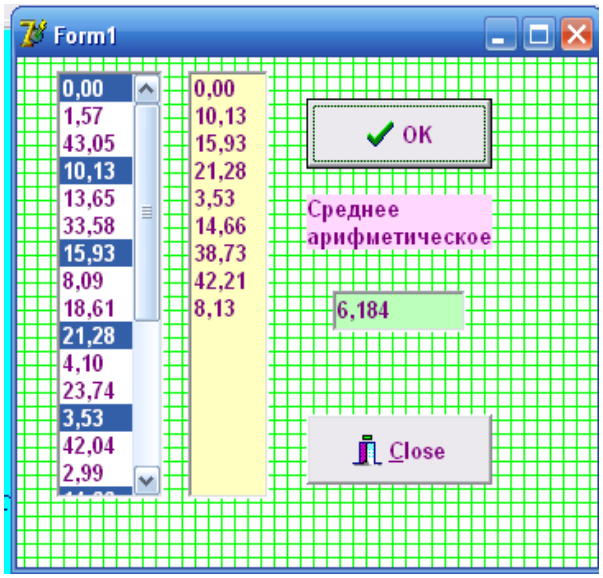


Рис. 3.9. Макет окна проекта «Вычисление среднего арифметического массива»

Задание 3.10.

Создание проекта «Работа с текстовым файлом в режиме **OpenDialog**». Макет формы (*рис. 3.10*).

Работа с файлами с использованием компонента **OpenDialog**. Выбрать содержимое текстового файла с флеш – носителя в **Memo** в режиме диалога.

Состав компонентов в форме:

- **OpenDialog** (1 страница **Dialogs**),
- **Button** (1 кнопка),
- **Memo** (1 многострочный редактор).

```
unit UnDial;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes,
   Graphics, Controls, Forms,
  Dialogs, StdCtrls;
 type
  TForm1 = class(TForm)
    OpenDialog1: TOpenDialog;
    Button1: TButton;
    Memo1: TMemo;
    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation
```

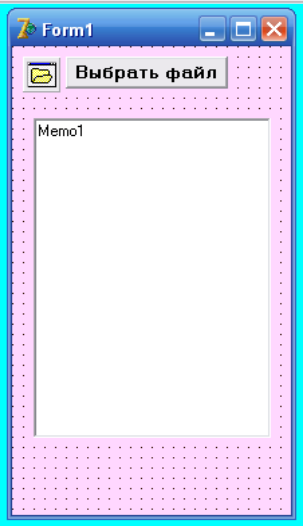


Рис. 3.10. Макет окна проекта «Работа с текстовым файлом в режиме OpenDialog» и окно кода программы

```
{$R *.dfm}

procedure TForm1.Button1Click(Sender: TObject);
var
S:string;
f:TextFile;
FileName:string[15];
begin
// Настраиваем диалог на отбор текстовых файлов
OpenDialog1.Filter:='Текстовые файлы(*.txt)' +
'Файлы Паскаля(*.pas)';
 //выполняем диалог и анализируем результат
if OpenDialog1.Execute and
FileExists(OpenDialog1.FileName)then
 begin
  //Если результат успешный- выбран нужный файл
 // открываем найденный файл
 AssignFile(F,OpenDialog1.FileName);
 reset(F);
 memo1.Lines.Clear;
 While not EOF(F) do
 begin
 Readln(F,S);
  //заполняем memo1
  memo1.Lines.Add(S);
  end;
  CloseFile(F);
  End
  else
  ShowMessage('not file');
```

Рис. 3.11. Код процедуры обработки события для операции копирования содержимого файла

3.2. Задания к Контрольной работе № 2

Задание № 1. Заполнить список (компонент ListBox) каждым вторым числом, генерируемым Random в цикле, при помощи оператора while для i<=20. Посчитать и вывести сумму элементов списка в однострочный редактор.

Задание № 2. Заполнить компонент Memo каждым третьим числом, генерируемым Random в цикле , при помощи оператора repeat с условием until j>=30.

Задание № 3. Заполнить компонент ListBox1 случайными величинами и записать в файл. Читать из файла в список - компонент ListBox2. Использовать две кнопки (для записи в файл и чтения из файла).

Задание № 4. Найти произведение матрицы (4x4) на столбец (4). Результат поместить в другой столбец (4). Использовать три таблицы (компонент StringGrid).

Задание № 5. Записать в файл случайные величины. Прочитать в компоненте ListBox.

Задание № 6. Записать в файл случайные величины. Прочитать в компоненте Memo.

Задание № 7. Заполнить один столбец компонента StringGrid целыми числами, задав их количество в компоненте SpinEdit, создать массив, отобразив его в компоненте Label, и отправить в файл, прочитать из файла во второй столбец таблицы. Столбцы именовать «В файл» и «Из файла».

Задание № 8. Создать массив целых величин, отобразив его в в компоненте Panel. Найти его среднее значение, используя функцию Mean. Результат среднего вывести в однострочный редактор.

Задание № 9. Найти совпадающие элементы компонента StringGrid, с образцом, вводимым в компонент Edit. В первый столбец занести порядковые номера артикулов, а во второй - названия артикулов. Вывести в компонент Label число совпадений и их порядковые номера.

Задание № 10. Найти совпадающие элементы компонента StringGrid, с образцом, вводимым в компонент Edit. В первый столбец занести порядковые номера фамилий, во второй – фамилии. Вывести в компонент Label число совпадений и их порядковые номера.

4. Список литературы

1. Санников Е.В. Курс практического программирования в Delphi. Объектно – ориентированное программирование [Электронный ресурс]/ Санников Е.В.— Электрон. текстовые данные.— М.: СОЛОН-ПРЕСС, 2013.— 188 с.— Режим доступа: http://www.iprbookshop.ru/26921.— ЭБС «IPRbooks»
2. Федотова С.В. Создание Windows-приложений в среде Delphi [Электронный ресурс]/ Федотова С.В.— Электрон. текстовые данные.— М.: СОЛОН-ПРЕСС, 2010.— 220 с.— Режим доступа: http://www.iprbookshop.ru/8664.— ЭБС «IPRbooks».
3. Ремнев А.А. Курс Delphi для начинающих. Полигон нестандартных задач [Электронный ресурс]/ Ремнев А.А., Федотова С.В.— Электрон. текстовые данные.— М.: СОЛОН-ПРЕСС, 2010.— 360 с.— Режим доступа: http://www.iprbookshop.ru/8680.— ЭБС «IPRbooks»

Типы данных

Тип данных:

- множество значений, которое может принимать переменная объявленного типа в допустимом для него (типе) диапазоне;
- множество операций разрешенных для данного типа.

Классификация типов данных

- **Простые:**
  - ✓ **Порядковые:**
    - Целые;
    - Логические (**Boolean**);
    - Символьные **(Char**);
    - Перечисляемые (**val1,…,valn**);
    - Интервальные (**диапазоны - min..max**);
  - ✓ **Вещественные (действительные);**
- **Строковые (string, string[n])**
- **Структурные:**
  - ✓ Массив **(array…of…);**
  - ✓ Множество **(set…of);**
  - ✓ Запись **(record…end);**
  - ✓ Файл:
    - Текстовый (**TextFile**);
    - Типизированный (**File of**…);
    - Нетипизированный (**file**);
- **Классы (class…end):**
  - ✓ Ссылки на класс;
  - ✓ Интерфейсы;
- **Указатели:**
  - ✓ Типизированные (**^,@**);
  - ✓ Нетипизированные (**pointer**);
- **Процедурные:**
  - ✓ Процедура **(procedure);**
  - ✓ Функция **(function);**
- **Тип Variant.**

Первоначально рассмотрим простые типы, а затем - строковые. Как видим из перечня типов, простые типы представляются порядковыми и вещественными. Каждому значению порядкового типа соответствует целое число, характеризующее его порядковый номер в заданном для этого типа диапазоне. Для целого – само его значение служит порядковым номером. Значения для типа **Boolean – 1 или 0,** т. е**. (**True, False**).** Значение для символов – соответствие коду ASCII (от 0..255).

Значения перечисляемых и интервальных типов определяются порядком описания допустимых значений – 0,1,2,…. Перечисляемые и интервальные типы позволяют ограничить количество возможных значений, принимаемых переменной в виде интервала или списка перечислений.

Перечисляемый тип (**val1,…,valn**), где **val1,…,valn –** упорядоченное множество имен.

Диапазоны. Тип диапазон представляет подмножество величин базового (порядкового типа). Использует ограничения путем задания минимального и максимального значений. Если для определения диапазона используются числовые или символьные константы, то их можно объявлять в разделе описания переменных.

Символьный тип **Char**(). Его значением являются символы из множества ASCII, Unicod.

Из множества целых типов в вычислениях лучше использовать (по возможности) Integer и Cardinal (*табл. А.1*).

Таблица А.1

Тип	Диапазон	Размер в байтах
Integer	-2147483648..2147483647	4
Cardinal	0..429467295	4
ShortInt	-128..127	1
Smallint	-32768..32767	2
LongInt	-2147483648..2147483648	4
Int64	$-2^{63}..2^{63}-1$	8
Byte	0..255	1
Word	0..65535	2
LongWord	0..429467295	4
Boolean	Логический тип: True, False	1

Определения (объявления) указанных типов могут быть выполнены в разделе объявления переменных var, либо описанием типа в разделе типов.

Например.

Для целого типа:

Var i,j,k:integer; m:byte;

Для перечисляемого типа:

Var Col: (Black,White,Green);

Для интервального типа (диапазона):

Var date: 1..31;

Var mont: 1..12;

Var Lch: ’a’..’z’;

Примечание. Для интервальных типов необходимо соблюдать корректную запись интервалов (нижний предел не должен быть больше верхнего).

Стандартные функции для значений и идентификаторов порядковых типов

Таблица А.2

Функция	Параметр	Действие
Ord(x)	Выражение порядкового типа	Возвращает порядковый номер значения x
Pred(x)	Выражение порядкового типа	Возвращает порядковый номер, предшествующий порядковому номеру значения x
Succ(x)	Выражение порядкового типа	Возвращает порядковый номер, следующий за порядковым номером значения x
High(x)	Идентификатор порядкового типа, выражение порядкового типа	Возвращает максимальное значение типа, к которому принадлежит переменная x. Вместо переменной возможна передача идентификатора интервального или целочисленного типов.

Основные вещественные типы

Таблица А.3

Тип	Диапазон	Значащие цифры	Байты
Real	$5{,}0*10^{-45}$ ..$1{,}7*10^{38}$	15-16	8
Currency	-922337203685477,5808.. 922337203685477,5807	19-20	8
Extended	$3{,}6*10^{-4951}$..$1{,}1*10^{4932}$	19-20	10
Double	$5{,}0*10^{-324}$..$1{,}7*10^{308}$	15-16	8
Single	$1{,}5*10^{-45}$..$3{,}4*10^{38}$	7-8	4

Операции, используемые в выражениях

- Унарные **– not, @;**
- Мультипликативные **-*,/,div,mod,and,sh1,shr;**
- Аддитивные **- +,-,or,xor;**
- Отношения **- =,<>,<, >,<=,>=,in;**

Приоритет выполнения в обозначенном порядке и устанавливается компилятором исходя из условий оптимизации программного кода.

Таблица А.4

Операция	Действие	Тип операндов	Тип результата
*	Умножение	Любой целый, вещественный	Наим. Целый Extended
/	Деление	Любой вещественный	Extended
div	Целочисленное деление	Любой целый	Наим. целый
mod	Остаток от деления	Любой целый	Наим. целый
and	Логическое И	Любой целый	Логический
sh1	Левый сдвиг	Любой целый	Наим. целый
shr	Правый сдвиг	Любой целый	Наим. целый
+	Сложение/Сцепление строк	Любой целый, вещественный, строковый	Наим. Целый Extended Строковый
-	Вычитание	Любой целый, вещественный	Наим. Целый Extended
or	Логическое ИЛИ	Логический, любой целый	Логический Наим. целый
=	Равно	Любой простой или строковый	Логический
<>	Не равно	Любой простой или строковый	Логический
<	Меньше	Логический	Логический
>	Больше	Логический	Логический
<=	Меньше или равно	Логический	Логический
>=	Больше или равно	Логический	Логический

Логические операции

- **Not (НЕ),**
- **Or (ИЛИ),**
- **And (И),**
- **Xor (**исключительное **ИЛИ).**

Операции применяются как к операндам логического типа, так и целого. Таблица логических операций известна из начальной информатики. Результат логической операции к операндам целого типа - целое число, биты которого формируются по правилам двоичной арифметики. Также к логическим операциям относят сдвиговые операции **sh1,shr** над целыми числами.

I shi j – сдвигает содержимое **I** на **j** разрядов влево, заполняя освободившиеся младшие разряды нулями.

I shr j– сдвигает содержимое **I** на **j** разрядов вправо, заполняя освободившиеся старшие разряды нулями.

I и **j –** выражения любого целого типа.

Строки – текстовые строки, состоящие из символов какого - либо языка, т.е. любого алфавита.

Базовые операции над строками:

- Определение длины строки.
- Сцепление (конкатенация) – сцепление строк.
- Сравнения.
- Выделения подстроки.

Для выполнения этих операций используются функции и процедуры.

Синтаксис этих функций и процедур:

Function Length(s:string):integer; - возвращает длину строки s.
Function Pos(s1:string;s:string):Byte; - ищет подстроку s1 в строке s.
Function ConCat(s1[,s2,…,sn]:string):string; - объединение строк от s1 до s.
Procedure Val(s:string;var x; var code:integer); -преобразует строку s в вещ. или целое.
Function StrToInt(s:string):integer; - преобразует строку s в целое число.
Function FloatToStr(value:Extended):string;
Function FloatToStrF(value:Extended;Format:TFloatFormat;Precision,Digits:Integer):string; - преобразует вещ. число в строку с учетом формата.
Function IntToStr(Value:Integer):string; - преобразует целое число в строку.
Function StrToFloat(const s: string):Extended; - преобразует строку s в вещ. число.

Структурные (структурированные) типы

- Массивы в Delphi и их характеристики.
- Допустимые размерности.
- Распределение массивов в памяти.
- Объявление статических массивов.

В отличие от переменных простых типов, переменные структурных типов имеют более одного значения. К переменным структурного типа относят **множества, массивы, записи, файлы**.

Массивы

Массив представляет собой структуру данных, состоящую из совокупности данных под одним именем и одного типа. При этом каждая совокупность может иметь свой (допустимый в Delphi) тип.

Допустимые размерности:

- Одномерные.
- Многомерные.

Характеристики массивов:

- Имя. Формируется (как и для простых переменных) по правилам Delphi.
- Тип хранимых элементов.
- Размер (число хранимых элементов).
- Нумерация элементов (порядковый номер элемента в массиве).
- Размерность (число измерений).
- Распределение массивов в памяти (статическое и динамическое).

Объявление статических массивов

Объявление одномерного массива:

Var <имя массива>:array[<ограниченный тип>] of <тип элементов>;

Примеры:

```
Var Массив: array[1..15] of  Integer; (интервальный)
Var Символ: array[1..25] of  chr;
Var Строка: array[1..25] of  String[15];
```

При работе со строковыми элементами необходимо указывать допустимую длину элементов.

Массив можно определить следующим образом:

```
Type color= ( red, yellow, green);
```

В качестве индекса выбран перечисляемый тип.

```
Var Acol: array[color] of integer;
```

Тогда обращение к элементам - к первому – Acol [red], ко второму -Acol [yellow] и к третьему соответственно - Acol[green].

Объявление массива можно совмещать с заданием элементам их начальных значений в том случае, если массив не является локальным, т. е. объявлен не в процедуре обработки события, а в разделе переменных модуля. В этом случае одновременно с объявлением массива можно выполнить его инициализацию. Объявление массива в общем виде выглядит так:

Имя:array [нижний_индекс..верхний_индекс] of <тип> = (список);

где список — разделенные запятыми значения элементов массива.

Например:

```
a: array[10] of integer = (0,0,0,0,0,0,0,0,0,0); (можно очистить одновременно)
b: array[1..5] of String[10]=('Зенит','Динамо','Спартак','Ротор','СКА');
Var Массив: array[0..10] of  Integer=(1,2,3,4,5,6,7,8,9,10,11);
```

Аналогично объявляется и типизированная константа:

```
const A: array[1..5] of  integer=(1,2,3,4,5);
```

Примечание. Если статический массив создан, но заданы не все значения его элементам, то неиспользованные элементы принимают произвольные значения в занимаемой ими памяти, поэтому перед заполнением массива его следует обнулить.

Для обработки статических массивов можно использовать в циклах следующие функции:

- **Length(**<идентификатор >**)** – возвращает количество элементов в массиве,
- **High(**<идентификатор >**)** – возвращает самое большое значение индекса массива (равное **Length -1**),
- **Low(**<идентификатор >**)** - возвращает 0.

Например:

for i:=0 to High(идентификатор) do <оператор>;

Способы обнуления массива.

Например, для массивов с именами **MemoMas** и **ms**:

1. **For i:=1 to 25 do MemoMas[i]:=0;**
2. **for i:=0 to High(ms) do ms[i]:=0;**

Многомерные массивы

Компактное объявление двумерного массива:

Var BMассив: array[1.. 5,1..3] of Integer;

Массив с элементами целого типа, расположенными в 5 строках и 3 столбцах (таблица). Данный двумерный массив может быть также объявлен следующим образом:

Var BMассив: array[1.. 5] of array 1..3] of Integer;

Динамическое распределение массива в памяти

- Объявление и переопределение динамических массивов.
- Действия над массивами.
- Использование функций, определенных в модуле math.
- Некоторые алгоритмы обработки динамических массивов.

Объявление и переопределение динамических массивов

Динамический массив в разделе объявлений содержит только ссылку на тип без указания размерности.

Например:

var ms:array of integer;

Далее, при выполнении процедуры, происходит переопределение массива с заданием его размерности с помощью процедуры: **SetLength.**

Например:

SetLength(A,25)

Выделяется место в памяти для 25 элементов массива **A**. В последующем возможно повторное применение указанной процедуры с изменением размера в сторону его увеличения или уменьшения.

```
Var mas:array of integer;
K,i:integer;
Begin
K:=5;
SetLength(ms,k),  // массив ms(0,0,0,0,0)
For i:=0 to k do ms[i]:=i+1;  // массив ms(1,2,3,4,5)
K:=7;
SetLength(ms,k),  // массив ms(1,2,3,4,5,0,0)
K:=3;
SetLength(ms,k),  // массив ms(1,2,3)
K:=7;
SetLength(ms,k),  // массив ms(1,2,3,0,0,0,0)
End;
```

Для обработки статических как и динамических массивов можно использовать в циклах следующие функции:

- **Length(**<идентификатор >**)** – возвращает количество элементов в массиве,
- **High(**<идентификатор >**)** – возвращает самое большое значение индекса массива (равное **Length -1**),
- **Low(**<идентификатор >**)** - возвращает 0.

Например:

for i:=0 to High(идентификатор) do <оператор>;

если в разделе объявлений указан динамический массив

DMas : of array of integer;

то в исполняемой части кода можно записать перераспределение памяти

SetLength(DMas, 15**);**

For i:= low(DMas**) to High(**DMas**) do ….** далее выполнить соответственно алгоритму код.

Действия над массивами:

- Ввод массива;
- Вывод массива;
- Суммирование элементов массива. Поиск MAX и MIN и прочих статистических характеристик совокупностей среди элементов в массиве
- Поиск заданного элемента и его порядкового номера в массиве
- Сортировка по возрастанию и убыванию.
- Присваивание значений в однотипных элементах и обработка многих других алгоритмов.

Использование функций, определенных в модуле math

Для числовых массивов допустимо использование следующих функций, определенных в модуле **math**:

Таблица А.5

Функция	Тип аргумента	Тип результата	Описание
MaxIntValue	Array of integer	integer	Возвращает максимальное значение элемента в массиве целых чисел
MinIntValue	Array of integer	integer	Возвращает минимальное значение элемента в массиве целых чисел
MaxValue	Array of double	double	Возвращает максимальное значение элемента в числовом массиве
MinValue	Array of double	double	Возвращает минимальное значение элемента в числовом массиве
Sum	Array of double	extended	Возвращает сумму элементов массива
SumInt	Array of integer	integer	Возвращает сумму элементов целочисленного типа
Mean	Array of double	extended	Возвращает среднее арифметическое значение элементов массива
StdDev	Array of double	extended	Возвращает среднее квадратическое отклонение элементов массива
PopnVariance	Array of double	extended	Возвращает дисперсию элементов массива

Обращения к функциям имеют общий вид:

Имя переменной:=Имя функции (Имя массива);

Например:

S:=SumInt(Massiv);

Переменная S в данном примере должна быть объявлена целочисленной, как и имя массива – Massiv. Во всех случаях обращения к функциям должно выполняться соответствие между типами аргументов и результатов.

Обработка массивов производится с помощью операторов цикла любой структуры (предпочтительнее - **For**).

Ввод / вывод массива

Значения элементов в массив могут быть введены / выведены различными способами, например из файла (в файл), используя в процессе работы с массивом различные компоненты, такие как:

- **StringGrid,**
- **Memo,**
- **ListBox**,

Кроме указанных выше компонентов для обработки массивов, можно использовать и однострочные редакторы **Edit,** а также **Label, Panel** для отображения массивов.

Поиск в массиве заданного элемента

При решении многих задач возникает необходимость определить, содержит ли массив определенную информацию или нет. Например, проверить, есть ли элемент массива, содержащий, определенный артикул товара. Задачи такого типа называются поиском в массиве. Для организации поиска в массиве могут быть использованы различные алгоритмы. Наиболее простой — это алгоритм простого перебора. Поиск осуществляется последовательным сравнением элементов массива с образцом до тех пор, пока не найдется элемент, равный образцу, или не будут проверены все элементы. Алгоритм простого перебора применяется, если элементы массива не упорядочены.

Сортировка по возрастанию и убыванию

Для сортировки массивов могут быть использованы различные алгоритмы**:**

- Простым перебором;
- Включениями (простыми, бинарными);
- Обменом (простым, шейкер - сортировка);
- Сортировка Шелла;
- Сортировка подсчетом.

Тексты кодов модулей проектов для выполнения заданий 3.1 -3.10 контрольной работы № 1

Задание 3.1

Текст кода модуля проекта «Возраст»

```
procedure TForm1.BitBtn1Click(Sender: TObject);
var T1,T2:TDateTime;
i:integer;
begin
  T1:=EnCodeDateTime(1984,10,15,0,0,0,300);
  T2:=EnCodeDateTime(2012,12,15,0,0,0,300);
  i:=YearsBetween(T2,T1);
  Label1.Caption:='Мне  исполнилось'+'  '+
  IntToStr(i)+'  лет';
end;

procedure TForm1.BitBtn2Click(Sender: TObject);
begin
form1.Hide
end;

end.
```

Задание 3.2

Текст кода модуля проекта «Светофор»

```
procedure TForm1.Button1Click(Sender: TObject);
type tLightState=0..2;
var LightState:tLightState;
begin
 label1.Caption:='';
 LightState:=random(3);
 if LightState=0 then
 label1.Caption:= label1.Caption+ 'Цвет светофора :Red - Стоп!';
 if LightState=1 then
 label1.Caption:= label1.Caption+ 'Цвет светофора:Yellov – Приготовься! ';
 if LightState=2 then
 label1.Caption:= label1.Caption+ 'Цвет светофора: Green-Поехали !';
end;

procedure TForm1.BitBtn1Click(Sender: TObject);
begin
 form1.Hide;
end;

end.
```

Задание 3.3

Текст кода модуля проекта «Скорость движения»

```
procedure TForm1.Button1Click(Sender: TObject);
const MaxSpeed=100;
var CurrentSpeed:integer;
begin
 Label1.caption:='';
 CurrentSpeed:=60; //Начальная скорость
 While CurrentSpeed < MaxSpeed do
  begin
  Label1.caption:=Label1.caption+'Вы едете со скоростью'+
  IntToStr(CurrentSpeed)+'км/ч'+#13;
  inc( CurrentSpeed,15); //Увеличиваем скорость
  end;
 Label1.caption:=Label1.caption+'Вы превысили максимально допустимую скорость!';
end;

end.
```

Задание 3.4

Текст кода модуля проекта «Суммирование элементов главной диагонали матрицы в таблице»

```
procedure TForm1.FormCreate(Sender: TObject);
var i:byte;
begin
   Brush.Style:=bsSolid;
   Brush.Style:=bsDiagCross;
   Brush.Color:=cllime;
   for i:=0 to StringGrid1.ColCount-1 do
      begin
      StringGrid1.ColWidths[i]:=30;
      StringGrid1.Cells[0,i]:=IntToStr(i);
       StringGrid1.Cells[i,0]:=IntToStr(i);
       end;
end;
procedure TForm1.Button1Click(Sender: TObject);
var  sum,i,j:byte;
begin
 sum:=0;
 for i:=1 to StringGrid1.ColCount-1 do
 for j:=1 to StringGrid1.ColCount-1 do
 begin
  if i=j then sum:=sum+StrToInt(StringGrid1.Cells[i,j])*SpinEdit1.value;
 end;
 Label1.Caption:=''+'Сумма элементов главной диагонали =';
 SpinEdit2.Value:=sum;
end;
end.
```

Задание 3.5

Текст кода модуля проекта «Реализация товаров» на примере работы с таблицами

```
procedure TForm1.FormCreate(Sender: TObject);
var i:byte;
begin
  for i:=1 to 4 do
  begin
   StringGrid1.ColWidths[i]:=40;
   StringGrid1.Cells[i,0]:=inttostr(i);
   StringGrid1.Cells[0,i]:='Tovar'+inttostr(i);
  end;
  StringGrid1.Cells[0,5]:='SumKol';
end;

procedure TForm1.BitBtn1Click(Sender: TObject);
var SumKol,i,j,f:Integer;
begin
 for j:=1 to 4 do
 begin
 SumKol:=0;
 for i:=1 to 4 do
   begin
   SumKol:= SumKol+strToInt(StringGrid1.Cells[j,i]);
   StringGrid1.Cells[j,5]:=IntToStr(SumKol);
   end;
 end;
 for i:=1 to 4 do
    begin
    f:=StrToInt(StringGrid1.Cells[i,5]);
    series1.AddXY(i,f,'',clred);
    end;
end;

procedure TForm1.BitBtn2Click(Sender: TObject);
begin
  form1.Hide;
end;

end.
```

Задание 3.6

Текст кода модуля проекта «Графическое отображение данных»

```
procedure TForm1.SpeedButton1Click(Sender: TObject);
begin
 With Series1 do
 begin
  Add(500,'Noutbook1',clRed);
  Add(1000,'Noutbook2',clRed);
  Add(1500,'Noutbook3',clRed);
```

```
  Add(700,'Noutbook4',clRed);
  Add(1800,'Noutbook5',clRed);
 end;
end;

procedure TForm1.SpeedButton2Click(Sender: TObject);
begin
 With Series2 do
 begin
  Add(19500,'Noutbook1',clRed);
  Add(39000,'Noutbook2',clLime);
  Add(21900,'Noutbook3',clGreen);
  Add(37600,'Noutbook4',clBlack);
  Add(41000,'Noutbook5',clOlive);
 end;
end;

procedure TForm1.SpeedButton4Click(Sender: TObject);
var
i:byte;
begin
 Series4.Clear;
 Series5.Clear;
 for i:=0 to 100 do
 begin
  Series4.AddXY(0.02*Pi*i,sin(0.02*Pi*i),'',clRed);
  Series5.AddXY(0.02*Pi*i,Cos(0.02*Pi*i),'',clGreen);
 end;
end;

procedure TForm1.SpeedButton3Click(Sender: TObject);
begin
 With Series3 do
 begin
  Add(500,'Noutbook1',clRed);
  Add(1000,'Noutbook2',clRed);
  Add(1500,'Noutbook3',clRed);
  Add(700,'Noutbook4',clRed);
  Add(1800,'Noutbook5',clRed);
 end;
end;

end.
```

Задание 3.7

Текст кода модуля проекта «Сортировка массива из случайных величин»

```
procedure TForm1.BitBtn1Click(Sender: TObject);
begin
 Form1.Hide
end;

procedure TForm1.Button1Click(Sender: TObject);
Var mas1:array of integer;
K,i,j,kk:integer;
begin
 panel1.Caption:='';
 panel2.Caption:='';
 K:=TrackBar1.Position; //установка движка трека
 SetLength(mas1,K); //динамический массив
  {ввод массива}
 For i:=0 to k-1 do
 begin
  mas1[i]:=random(50); //заполнение массива случайными числам
  panel1.Caption:=panel1.Caption+'   '+inttostr(mas1[i]);
 end;
    {сортировка}
 For j:=0 to k-1 do
 begin
  kk:=0;
  For i:=0 to k-1 do
  begin
   if mas1[j]<mas1[i] then
   begin
    kk:=mas1[i];
    mas1[i]:=mas1[j];
    mas1[j]:=kk;
   end;
  end;
 end;
//  вывод отсортированного массива в панели
For i:=0 to k-1 do
panel2.Caption:=panel2.Caption+'   '+inttostr(mas1[i]);
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
 Brush.Style:=bsSolid;
 Brush.Style:=bsCross;
 Brush.Color:=clRed;
 Label1.Caption:=IntToStr(TrackBar1.Position);
end;
```

```
procedure TForm1.TrackBar1Change(Sender: TObject);
begin
 Label1.Caption:=IntToStr(TrackBar1.Position);
end;

end.
```

Задание3.8

Текст кода модуля проекта «Перевод чисел из 2 –ой системы счисления в 10 – ю и наоборот»

```
procedure TForm1.okClick(Sender: TObject);
begin
 If Edit1.Text<>'' then
 case RadioGroup1.ItemIndex of
 0:if Test2
  then Rewer2_10
  else ShowMessage('Это не двоичное число');
 1: if Test10
  then Rewer10_2
  else ShowMessage('Это не десятичое число');
 end
 else ShowMessage('Пустая строка');
end;

function TForm1.Test2:Boolean;
var
i:byte;
ok:Boolean;
s:string;
begin
 s:=Edit1.Text; ok:=True;i:=0;
 while (i<length(s)) and ok do
 begin
  i:=i+1; ok:=s[i] in ['0','1'];
 end;
 test2:=ok;
end;

function TForm1.Test10:Boolean;
var
i:byte;
ok:Boolean;
s:string;
begin
 s:=Edit1.Text; ok:=True;i:=0;
 while (i<length(s)) and ok do
 begin
 I :=i+1; ok:=s[i] in ['0'..'9'];
 end;
 test10:=ok;
end;
```

```
Procedure TForm1.Rewer10_2;
var i:byte;
l:integer;
s:string;
begin
l:=StrToInt(Edit1.Text);s:='';
While l <>0 do begin
s:=IntToStr(l mod 2)+s;l:=l div 2;
end;
label1.caption:=s;
end;

Procedure TForm1.Rewer2_10;
var i:byte;
l:integer;
s:string;
begin
 s:=Edit1.Text;  l:=0;
 for i:=1 to length(s) do l:=2*l+ ord(s[i])-ord('0');
 Label1.caption:=IntToStr(l);
end;

end.
```

Задание 3.9

Текст кода модуля проекта «Вычисление среднего арифметического массива»

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  Brush.Style:=bsCross;
  Brush.Color:=cllime;
end;

procedure TForm1.BitBtn1Click(Sender: TObject);
var
j,i:byte;
ms:array[1..25] of Double;
begin
  LstBox.Clear;
  LstBx2.Clear;
  j:=0;
  for i:=1 to 25 do
  ms[i]:=0;
  for i:=1 to 25 do
  begin
   LstBox.Items.Add(FloatToStrF(random*50,ffFixed,4,2));
  end;
  repeat
   LstBox.Selected[j]:=true;
   If  LstBox.Items[j]<>'' then
   begin
```

```
    LstBx2.Items.Add(LstBox.Items[j]);
    Inc(j,3);
   end;
  until j>25;
  for i:=1 to LstBx2.Count-1 do
  ms[i]:= StrToFloat(LstBx2.Items[i]);
  Edit1.Text:=FloatToStr(Mean(ms));
end;
procedure TForm1.BitBtn2Click(Sender: TObject);
begin
 close;
end;

end.
```

Задание 3.10

Текст кода модуля проекта «Работа с текстовым файлом в режиме Opendialog»

```
procedure TForm1.Button1Click(Sender: TObject);
var
S:string;
f:TextFile;
FileName:string[15];
begin
// Настраиваем диалог на отбор текстовых файлов
 OpenDialog1.Filter:='Текстовые файлы(*.txt)' +
 'Файлы Паскаля(*.pas)';
// выполняем диалог и анализируем результат
 if OpenDialog1.Execute and
FileExists(OpenDialog1.FileName) then
 begin
//Если результат успешный- выбран нужный файл
// открываем найденный файл
  AssignFile(F,OpenDialog1.FileName);
  reset(F);
  memo1.Lines.Clear;
  While not EOF(F) do
  begin
   Readln(F,S);
//заполняем memo1
   memo1.Lines.Add(S);
  end;
  CloseFile(F);
 end
else
  ShowMessage('not file');
end;

end.
```

Сообщение об ошибках

В результате компиляции могут быть сделаны замечания, предупреждения и сообщения об ошибках.

Сообщения компилятора располагаются в окне, расположенном ниже окна редактора кода.

Различают три разновидности сообщений: ошибки, подсказки, предупреждения.

Без устранения ошибки дальнейшая сборка проекта невозможна.

Подсказка подразумевает предложения компилятора по оптимизации кода.

Предупреждения не влияют на ход сборки проекта, но дают знать, что могут встретиться проблемы при выполнении программы.

Ниже приведены сообщения компилятора о пяти ошибках, связанных с отсутствием символа «:» и несоответствием типов.

В откомпилированной программе могут также появляться сообщения, возникающие по причине вводимых данных, отсутствии файла, на который есть ссылка и т. д.:

- ошибки выполнения программы (например, деление на 0),
- логические ошибки (получен неверный результат).

Кодовая часть проекта с декларацией ошибок показана на рисунке В.1.

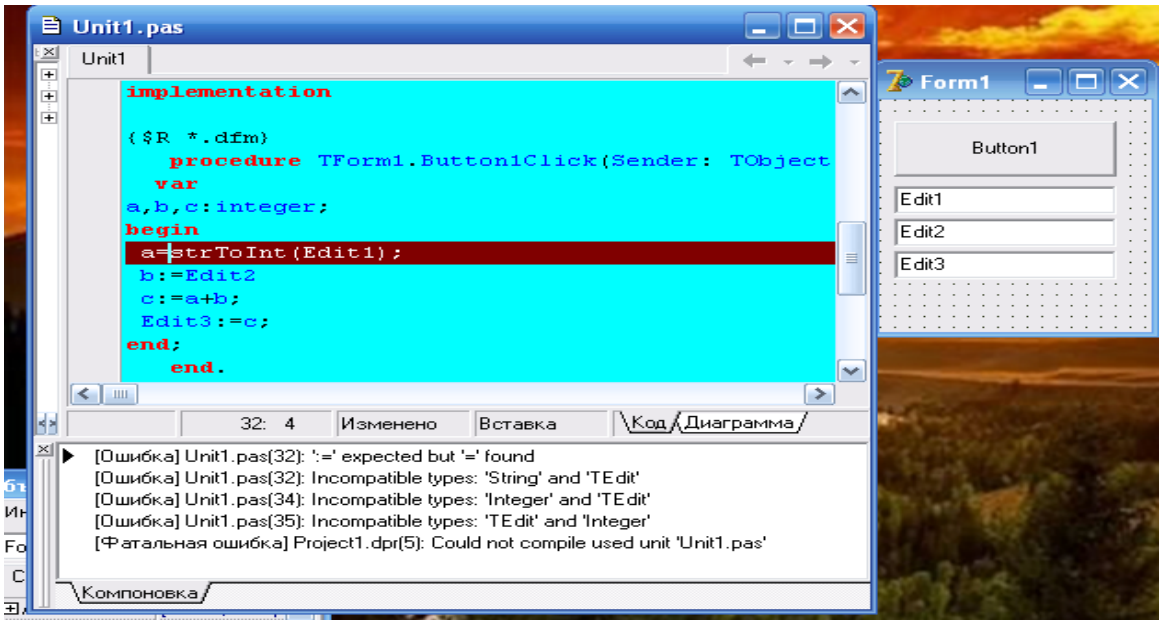


Рис. В.1. Сообщения об ошибках при обработке кода проекта

Работа с файлами

Демонстрационный пример к заданиям № 1 - № 10 контрольной работы № 2

Создание текстовых файлов

При работе с файлами необходимо знать:

- С какими типами файлов собираетесь работать.
- Цель открытия – запись, чтение, добавление.
- Действия при работе с файлами.

Доступ к информации в файлах при открытии:

- К текстовым.
- Нетипизированным.
- Типизированным.

Текстовые файлы могут содержать символы #13#10- и признак конца строки -#26.

В типизированных файлах чтение и запись допускается порциями одинаковой длины. При этом структура порции указывается при объявлении файлового указателя.

Нетипизированные файлы служат для быстрого доступа к ним независимо от их типа и структуры и объявляются простым файлом.

Например:

var MyFile:file;.

Общие действия при работе с любыми файлами:

Таблица Г.1

Номер действия	Действие	Текстовые	Типизированные	Нетипизированные
1	Описать файловый указатель (ф.у.)	F:TextFile	F:File of тип; X: тип;	F:File
2	Связать ф.у. с именем файла	AssignFile(f,name)	AssignFile (f,name)	AssignFile(f,name)
3	Объявить существующий файл или ***(новый)***	Reset(f) Append(f) ***Rewrite(f)***	Reset(f) ***Rewrite(f)***	Reset(f,1) ***Rewrite(f,1)***
4	Читать	Read(f, список) Readln(f,список)	Read(f,x)	BlockRead (f,buf,SizeOf(buf), NumWrite)
5	Записать	Write(f,список) Writeln(f, список)	Write(f,x)	BlockWrite (f,buf,SizeOf(buf), NumWrite)
6	Закрыть	CloseFile(f)	CloseFile(f)	CloseFile(f)

где

f- файловый указатель (любая переменная файлового типа – текстового, типизированного, нетипизированного). До момента использования ее, она должна быть связана с внешним файлом (файл на диске, а также на устройствах – дисплее, клавиатуре). Если связь установлена, то существующий файл может быть открыт с помощью процедуры **Reset(),** устанавливающая указатель в начало файла, а также процедурой **Append()** для добавления.

Новый файл создается и открывается с помощью процедуры **Rewrite().**

Текстовые файлы, открываемые **Reset(),** предназначены только для **чтения**, а по **Rewrite() и Append() –** только для записи.

Запись или чтение в типизированных и нетипизированных файлах не зависит от способа открытия.

Доступ к файлам – последовательный. При чтении файла с помощью Read() и записи - Write() файловый указатель перемещается на следующий компонент.

Типизированные и нетипизированные файлы доступны произвольно с помощью процедур **Seek(), FilePos(), FileSize().**

Seek() – перемещает файловый указатель на заданный компонент.

FilePos() – определяет текущую файловую позицию.

FileSize() – размер файла**.**

Закрытие файлов. Для всех типов файлов - это процедура **CloseFile().**

Примечание. Ввиду ограничения времени остановимся на работе с текстовыми файлами.

Работа с текстовыми файлами

Если текстовый файл имеет указатель типа **TextFile,** то он рассматривается как последовательность символов, сформированную в строки, заканчивающиеся признаком конца строки #13 – возврат каретки, либо в сопровождении #10 - перевод строки.

```
  var mass:array[1..10] of integer;
F:TextFile;
i:integer; //,n
```

В файле

```
AssignFile(f,'f.txt');Rewrite(f);
 for i:=1 to  10 do begin
 write(f,i:2,'',mass[i]:2,'');
  writeln(f);
  end;
 CloseFile(f);
```

```
 1 1
 2 2
 3 3
 4 4
 5 5
 6 6
 7 7
 8 8
 9 9
1010
```

Рис. Г.1. Запись в файл

Демонстрационный пример

Запись информации из строки таблицы в файл и чтение из файла в таблицу.

Значения 1,2,....,10, записанные в нулевую строку StringGrid в цикле в процедуре обработки события по FormCreate (двойной щелчок по форме) отправляются в файл, а при выполнении второй процедуры по Button1 читаются из файла в первую строку. Параметры StringGrid определяются в процедуре FormCreate (см. код модуля проекта).

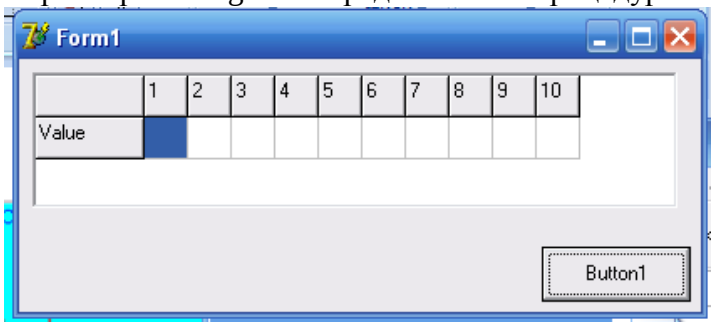


Рис. Г.2. Данные для записи в файл (обработка события FormCreate)

```
procedure TForm1.FormCreate(Sender: TObject);
  var i:integer;
   f:TextFile;
begin
   for i:=1 to 10 do begin
    StrGr.ColWidths[i]:=25;
    StrGr.cells[i,0]:=inttostr(i);
   end;
  StrGr.Cells[0,1]:='Value';
   AssignFile(f,'F:\MuF.txt'); Rewrite(f);
          for i:=1 to 10 do begin
           Write(f,StrGr.cells[i,0]);
       Writeln(f);
     end;
     closeFile(f);
end;
 procedure TForm1.Button1Click(Sender: TObject);
   var lst,i:integer;
 f:TextFile;
begin
 i:=1;
   AssignFile(f,'F:\MuF.txt');reset(f);
   while not eof(f) do begin
   read(f,lst);
 StrGr.cells[i,1]:= inttostr(lst);
 i:=i+1;
   end;
   closeFile(f);
end;
  end.
```

Рис. Г.3. Код модуля проекта

Результат выполнения проекта представлен ниже.

Рис. Г.4. Результат чтения из файла (обработка события Button1Click)