КАЗАНСКИЙ (ПРИВОЛЖСКИЙ) ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ

ДОПОЛНИТЕЛЬНЫЕ ГЛАВЫ ПРОГРАММИРОВАНИЯ В DELPHI

Учебно-методическое пособие для студентов
отделения информационных технологий в гуманитарной сфере
Института Вычислительной математики и информационных
технологий

КАЗАНЬ
2012

УДК 519.682

Печатается по решению
заседания учебно-методической комиссии
Института Вычислительной математики и информационных технологий
Протокол №6 от 16 февраля 2012 года

заседания кафедры информатики и вычислительных технологий
протокол №6 от 19 января 2012 г.

Автор-составитель:
ст. преп. Р.Ш. Гайнанова

Научный редактор
кандидат физ. мат. наук, доцент А.Ф. Галимянов

Рецензенты:
асс. Ф.А. Галимянов (КНИТУ)
кандидат физ. мат. наук, доц. О.А. Широкова (КФУ)

Дополнительные главы программирования в Delphi: учебно-методическое пособие / Р. Ш. Гайнанова – Казань: КФУ, 2012. – 74с.

Данное пособие предназначено для проведения аудиторных занятий и самостоятельных работ студентов дневного и заочного обучения. Пособие может быть использовано для проведения курсов по выбору, предназначенных для дальнейшего изучения программирования в Delphi. В каждом разделе учебно-методического пособия приведены теоретический материал, примеры решения задач с разъяснениями и задания для самостоятельного выполнения.

Содержание

Введение

С появлением первых компьютеров в начале 50-х годов прошлого века появились и языки программирования первого поколения. Это был язык ассемблер, созданный по принципу «одна инструкция – одна строка». В конце 50-х начале 60-х годов был разработан символический ассемблер, в котором появилось понятие переменной. Он стал первым полноценным языком программирования. Благодаря его возникновению заметно возросли скорость разработки и надежность программ. В 60-е годы появились универсальные языки высокого уровня.

Языки программирования можно разделить на две большие группы: процедурные и непроцедурные. Процедурные языки программирования делятся на языки низкого уровня и высокого уровня. Языки низкого уровня – это машинно-ориентированные языки. Драйверы внешних устройств, системы программирования были разработаны с помощью языков низкого уровня. К таким языкам относятся ассемблер, макроассемблер, автокоды. Наиболее популярные языки высокого уровня – это Фортран, Кобол, Алгол, Паскаль, Бейсик, Си и др. Основные понятия процедурных языков - оператор и данные. Операторы объединяются в группы – процедуры.

Принципиально иное направление в программировании связано с методологиями (иногда говорят «парадигмами») непроцедурного программирования. В их число входят объектно-ориентированное и декларативное программирование.

Объектно-ориентированная программа совокупность множества независимых объектов. Каждый объект можно использовать для решения задачи, не вникая во внутренние механизмы его функционирования. Наиболее популярные языки объектного программирования C++, Delphi, Visual Basic.

Delphi - система программирования очень высокого уровня. Фирма Borland (в начале 1990-х сменила название на Inprise), разработавшая эту систему, вложила в Delphi максимально комфортный для разработчиков и пользователей интерфейс в стиле Windows.

Многие компоненты Delphi имеют свое визуальное изображение. Размещение компонентов на экране, задание их начальных свойств Delphi позволяет осуществлять на этапе конструирования формы (окна будущей программы). Такой способ работы с объектами, имеющими графическое представление, принято называть *визуальным программированием*.

Кроме этого Delphi является системой *объектно-ориентированного программирования*. Она позволяет использовать уже имеющиеся объекты для создания новой программы.

Объектом называется специальным образом оформленный фрагмент программы, содержащий в себе данные и подпрограммы для их обработки. Данные называются *полями* объекта, а подпрограммы его *методами*. Объект предназначен для решения какой-либо конкретной задачи и воспринимается в

программе как единое целое (т.е. нельзя из объекта «выдернуть» отдельное поле или метод). Объекты придуманы, чтобы увеличить производительность труда программиста и повысить качество программ. Разработчики Delphi придумали сотни объектов, которые можно рассматривать как кирпичики, из которых строится программа. Такой принцип построения программ называется *объектно-ориентированным программированием*.

1. Классы и объекты

Классами в Delphi называются функционально законченные фрагменты программ, служащие для создания подобных экземпляров. Однажды создав класс можно включать его экземпляры (копии) в разные программы или в разные места одной и той же программы. В состав Delphi входят несколько сотен стандартных классов, созданных программистами Inprise.

В основе классов лежат три фундаментальных принципа: инкапсуляция, наследование и полиморфизм.

Инкапсуляция. Класс представляет собой единство трех сущностей – полей, методов и свойств. Объединение этих сущностей в единое целое и называется инкапсуляция. Инкапсуляция позволяет изолировать класс от остальных частей программы, сделать его "само достаточным" для решения конкретной задачи. В результате класс всегда несет в себе функциональность. Например, класс TForm содержит все необходимое для создания Windows – окна, TTimer обеспечивает работу программы с таймером и т.д. Инкапсуляция представляет собой мощное средство для обмена готовыми к работе программными заготовками.

Наследование. Любой класс может быть порожден от другого класса. При объявлении класса указывается имя класса – родителя.

```
type
   TForm1 = Class(TForm) ;
```

Порожденный класс автоматически наследует поля, методы и свойства своего родителя и может добавлять их новыми. Принцип наследования обеспечивает поэтапное создание сложных классов и разработку собственных библиотек классов.

Все классы Delphi порождены от единственного родительского класса TObject. Этот класс не имеет полей и свойств, но включает в себя методы самого общего назначения, обеспечивающие весь жизненный цикл любых объектов – от их создания до уничтожения.

Полиморфизм. Это свойство классов решать схожие по смыслу проблемы разными способами. Поведенческие свойства класса определяются набором входящих в него методов. Изменяя алгоритм того или иного метода в потомках класса, можно придавать этим потомкам отсутствующие у родителя специфические свойства. Для изменения метода необходимо *перекрыть* его в потомке, т.е. объявить в потомке одноименный метод и

реализовать в нем нужные действия. В результате в объекте-родителе и в объекте-потомке будут действовать два *одноименных* метода, имеющих разную алгоритмическую основу и, следовательно, придающих объектам разные свойства. Это и называется *полиморфизмом* объектов.

Составляющие класса

Поля. Полями называются инкапсулированные в класс данные. Поля могут быть любого типа, в том числе – классами. Например,

```
type
    TClass1 = Class
      a : integer;
      b : string;
      Obj1 : TObject;
    end;
```

Каждый объект получает уникальный набор полей, но для всех объектов данного класса набор методов и свойств общий. Принцип инкапсуляции требует обращаться к полям только с помощью методов и свойств класса. Однако в Delphi разрешается обращаться к полям и напрямую :

```
var
    Obj1 : TClass1;
begin
    . . . . .
    Obj1.a: = 0;
    Obj1.b: = 'Строка символов';
    . . . . .
end;
```

Методы. Инкапсулированные в класс процедуры и функции называются методами. Они объявляются так же, как и обычные подпрограммы:

```
type
    TClass2 = Class
    Function Func1 (par1: integer ) : real;
    Procedure Proc1 (par2 : real );
end;
var
    Obj2 : TClass2;
    a1 : integer;  a2 : real ;
begin
    a1:=1;
    a2:= Obj2.Func1 (a1);
    . . . . .
    Obj2.Proc1 (a2);
    . . . . .
end;
```

Свойства. Свойства – это специальный механизм классов, регулирующий доступ к полям. Свойства объявляются с помощью зарезервированных слов ***property, read, write*** (слова read, write считаются

зарезервированными только при объявлении свойств). Обычно свойство связано с некоторым полем и указывает те методы класса, которые должны использоваться при записи в это поле или при чтении из него. Например,

```
type
    TClass1 = Class
    Function GetField : integer;
    Procedure SetField (v : integer );
    Property IntegerValue : integer read GetField  write
 SetField;
end;
```

Имя свойства IntegerValue, тип – integer, для записи данных используется процедура SetField, для чтения – GetField.

Директивы Protected и Private

Помимо объявления элементов класса (полей, методов и свойств) описание класса, как правило, содержит директивы **Protected** (защищенный) и **Private** (личный), которые устанавливают степень видимости элементов класса в программе.

Элементы класса, объявленные в секции *Protected*, доступны только в порожденных от него классах. Область видимости элементов класса этой секции не ограничивается модулем, в котором находится описание класса. Обычно в секцию *Protected* помещают описание методов класса.

Элементы класса, объявленные в секции *Private*, видимы только внутри модуля. Эти элементы не доступны за пределами модуля, даже в производных классах. Обычно в секцию *Private* помещают описание полей класса, а методы, обеспечивающие доступ к этим полям - в секцию *Protected*.

2. Общие свойства компонентов

Все компоненты Delphi являются объектами от класса *TComponent*, в котором инкапсулированы самые общие свойства и методы компонентов. Предком TComponent является класс *TPersistent*, который произошел непосредственно от базового класса *TObject*. Класс TComponent служит базой для создания как видимых, так и невидимых компонентов. Большинство видимых компонентов происходит от класса TControl. Два наследника этого класса *TWinControl* и *TGraphicControl* определяют две группы компонентов: имеющие оконный ресурс (компоненты класса TWinControl и его потомки) и не имеющие этого ресурса (компоненты класса TGraphicControl и его потомков). Оконный ресурс – это специальный ресурс Windows, предназначенный для создания и обслуживания окон. Только оконные компоненты способны получать и обрабатывать сообщения Windows.

Имя компонента строится по тем же правилам, что и имена любых других объектов программирования – констант, переменных, подпрограмм и

т.д.: оно представляет собой правильный идентификатор. Компоненты помещаются на форму средой Delphi, каждый компонент автоматически получает создаваемое средой имя, совпадающее с именем своего класса (без начальной буквы T) и дополненное числовым суффиксом: *Form1*, *Label2*, *Edit3* и т.д.

Положение и размеры компонента определяются четырьмя его свойствами (в пикселях):

```
Property Height : integer;     // Высота
property Left :  integer;      //  Положение левой кромки
property Top :  integer;       //   Положение верхней кромки
property  Width :  integer;    //  Ширина
```

Важную роль играет свойство *Align*, определяющее выравнивание положения компонента относительно границ своего родителя:

```
type  TAlign = (Alnone, Altop, Albottom,  AlLeft, AlRight, AlClient) ;
    property  Align : TAlign;
```

Если это свойство не равно Alnone, компонент прижимается к верхней (Altop), нижней (Albottom), левой (AlLeft) или правой (AlRight) границе своего родителя. При этом размеры компонента по соседним с границей измерениям игнорируются и компонент «растекается» по границе. Например, если Align = Altop, значения свойств компонента Left, Width игнорируются, и его прямоугольник будет занимать всю верхнюю часть родителя высотой Height пикселей. Если Align = AlLeft, свойства Top и Height игнорируются и прямоугольник занимает левую часть родителя шириной Width пикселей и т.д. Если несколько компонентов имеют одинаковое выравнивание, они последовательно прижимаются друг к другу. Вся не заполненная другими компонентами область родителя заполняется компонентами со свойствами Align = AlClient, которые в данном случае накладываются друг на друга.

Свойство

```
property Enabled : Boolean;
```

определяет возможность активизации компонента. Если оно имеет значение Fulse, компонент запрещен для выбора. Такие компоненты обычно отображаются серым цветом.

Любой видимый компонент можно спрятать или показать с помощью свойства Visible или методами Hide и Show:

```
property Visible : Boolean ;
procedure Hide ;
procedure Show ;
```

Спрятанный компонент не реагирует на события от мыши или клавиатуры, ему нельзя передать фокус ввода.

Некоторые компоненты имеют плоское изображение (например, метка Label), другие – всегда объемные (например, кнопка Button). Объемность изображения элементов регулируется свойством

```
property Ct13D : Boolean ;
```

С каждым управляющим компонентом связывается текстовая строка, которая становится доступно либо через свойство Caption, либо через свойство Text. Свойство

type TAlignment =(TaLeftJustify, TaCenter, TaRightJustify) ;

property Alignment : TAlignment;

регулирует положение текста относительно границ компонента:

TaLeftJustify - прижать к левой границе;

TaCenter – расположить по центру;

TaRightJustify - прижать к правой границе.

Наиболее часто используемые компоненты

Компонент Label - метка для отображения текста, расположена на странице Standard, предназначена для размещения на форме не очень длинных надписей. С помощью свойства Caption метка может отображать длинную текстовую строку в несколько строк. Для этого свойство Autosize установить false, задать большие размеры метки (Width – ширина, Height – высота), Wordwrap установить true. Свойства компонента:

Align – способ выравнивания компонента внутри родительского компонента;

Alignment – способ выравнивания текста внутри компонента Label;

Height – высота компонента;

Width – ширина компонента;

Font – шрифт. Можно изменить вид надписи (размер, стиль шрифта, цвет);

Autosize (тип Boolean) - указывает будет ли метка изменять свои размеры в зависимости от помещенного в Caption текста: true – будет;

Wordwrap (тип Boolean) - разрешает / запрещает разрыв строки на границе слова.

Компонент Edit представляет собой однострочный редактор текста. С его помощью можно вводить и отображать достаточно длинные текстовые строки. Свойства:

Text (mun string) – содержит отображаемый компонентом текст. Это центральное свойство компонента;

MaxLength (mun integer) – определяет максимальную длину строки, если 0, длина строки не ограничена;

Modified (mun Boolean) – содержит true, если текст был изменен;

SelStart (mun integer) – содержит номер первого символа выделенной части текста;

SelLength (mun intrger) – содержит длину выделенной части текста;

SelText (mun string)–содержит выделенную часть текста.

Методы компонента:

Procedure Clear – удаляет весь текст;

Procedure ClearSelection – удаляет выделенный текст;

Procedure SelectAll – выделяет весь текст.

С помощью обработчика *события OnChange* программа может контролировать вводимый текст, при необходимости фильтровать его, игнорируя недопустимые символы.

Компонент Memo представляет собой многострочный редактор, предназначен для ввода, редактирования, отображения достаточно длинного текста. Текст хранится в свойстве Lines и представляет собой пронумерованный набор строк. Нумерация начинается с нуля. Свойства:

Lines (mun string) – содержит строки текста; представляет собой пронумерованный набор строк (нумерация начинается с нуля);

Text (mun string) – содержит отображаемый компонентом текст в виде одной длинной строки. Границы строк выделяются символами #13#10 (признак eoln);

ScrollBars – определяет наличие в окне редактора полос прокрутки, может принимать одно из следущих значений:

SsNone – нет прокрутки;

SsHorizontal – горизонтальная полоса прокрутки;

SsVertical – вертикальная полоса прокрутки;

SsBoth – обе полосы;

WantReturn (mun Boolean) – если содержит true, нажатие Enter вызывает переход на новую строку, в противном случае обрабатывается системой. Для перехода на новую строку в этом случае нажать ctrl+Enter.

Многие свойства и методы аналогичны свойствам и методам Edit.

Командная кнопка Button используется для управления работой программы. Свойства:

Cancel (тип Boolean) - если true, то событие OnClick возникает при нажатии на клавишу Esc;

Default (тип Boolean) - если true, то событие OnClick возникает при нажатии на клавишу Enter;

Кнопка Button является компонентом самой Windows, поэтому не может изменять свой цвет, нет свойства Color.

Кнопка Bitbtn (страница Additional) - популярная разновидность кнопки Button. Свойство Kind определяет одну из 11 разновидностей кнопки.

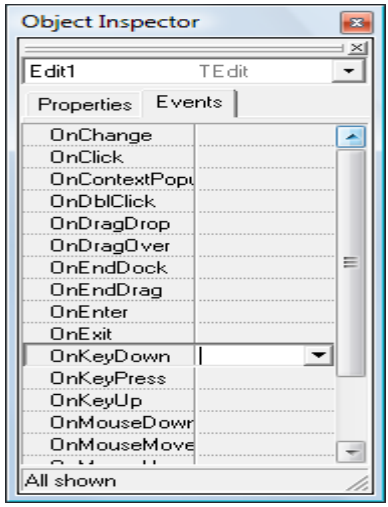
Рис. 2.1.

3. Реакция на события

Программам можно придать функциональность. Функциональность программы определяется совокупностью ее реакций на те или иные события. Каждый компонент, кроме свойств, характеризуется также набором событий, на которые он может реагировать. *Событие (Event)* – это то, что происходит во время выполнения программы. У каждого события есть имя. Например, щелчок кнопкой мыши – это событие Click, двойной щелчок кнопкой – событие DblClick, нажатие клавиши клавиатуры – событие –KeyPress.

Реакцией на событие должно быть какое-то действие. В Delphi реакция на событие реализуется как *процедура обработки события*. Чтобы программа выполняла некоторую работу в ответ на действия пользователя, нужно написать процедуру обработки соответствующего события.

Рис.3.1



Чтобы приступить к созданию процедуры обработки события, нужно сначала выбрать компонент, событие которого надо обработать (в верхней части окна *Object Inspector* отображается имя выбранного компонента). Затем в окне *Object Inspector* надо открыть вкладку *Events* (рис. 3.1). В левой колонке вкладки *Events* указаны ***свойства*** **(*property*),** значения которых определяют процедуры обработки соответствующих событий. События – свойства процедурного типа.

В программу будет добавлена процедура обработки события (ее имя появится на вкладке *Events*) и откроется окно редактора кода, в котором можно набирать инструкции реализующие обработку события.

Сформированное Delphi *имя процедуры обработки события* состоит из двух частей. Первая часть имени идентифицирует форму, содержащую компонент, для которого создана процедура обработки события. Вторая часть имени - компонент и событие. В нашем примере имя формы – Form1, имя компонента – Edit1, имя события –KeyDown. Delphi в окно кода программы добавит следующие строки:

```
procedure TForm1.Edit1KeyDown(Sender: TObject; var Key: Word;
  Shift: TShiftState);
begin
```

end;

3.1. События от мыши

Для большинства видимых компонентов определен набор обработчиков событий связанных с мышью:

```
type
  TMouseButton = (mbLeft, mbRight, mbMiddle);
        TShiftState = set of (ssSift, ssAlt, ssCtrl, ssLeft, ssRight, ssMiddle,
    ssDouble);
 TMouseEvent = procedure (Sender: TObject;
                          Button: TMouseButton;
                          Shift: TShiftState; X, Y: Integer) of object;
 TMouseMoveEvent = procedure (Sender: TObject;
                     Shift: TShiftState; X, Y: Integer) of object;
 TNotyfyEvent = procedure (Sender: TObject) of object;
    property OnMouseDown: TMouseEvent;
    property OnMouseUp: TMouseEvent;
    property OnMouseMove: TMouseMoveEvent;
    property OnClick: TNotyfyEvent;
    property OnDbClick: TNotyfyEvent;
```

Обработчики событий *OnMouseDown* и *OnMouseUp* определяют реакцию программы на соответственно нажатие и отпускание кнопки мыши, *OnMouseMove* – реакция на перемещение указателя мыши над компонентом, *OnClick* и *OnDblClick* - соответственно на щелчок и двойной щелчок левой кнопки.

Тип *TMouseButton* определяет одну из трех кнопок мыши: левую (*mbLeft*), правую (*mbRight*) и среднюю (*mbMiddle*).

Тип *TShiftState* содержит признаки, уточняющие обстоятельства возникновения события: *ssShift* – нажата клавиша *Shift*; *ssAlt* - нажата клавиша *Alt*, *ssCtrl* - нажата клавиша *Ctrl*; *ssLeft* - нажата левая кнопка мыши; *ssRight* - нажата правая кнопка мыши; *ssMiddle* - нажата средняя кнопка мыши; *ssDouble* – нажаты одновременно левая и правая кнопки.

Например, если для формы Form1 создать обработчик события *OnMouseMove*, то откроется такая процедура:

```
procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X,
Y: Integer);
begin

end;
```

Во всех обработчиках *Sender* содержит ссылку на компонент, над которым произошло событие, а X и Y- координаты указателя мыши. Во все эти процедуры передается дополнительный параметр *Shift* типа *TShiftState*.

Пример 1. Нарисовать светофор с тремя лампочками, которые реагируют на наведение указателя мыши.

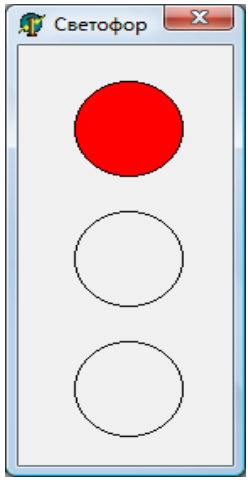
Для рисования лампочек светофора использовать компонент Shape.

Компонент Shape – стандартная фигура (страница Additional), рисует одну из простейших геометрических фигур. Вид фигуры определяется свойством TShapeType. Это свойство может принять следующие значения (прямоугольник, квадрат, скругленный прямоугольник, скругленный квадрат, эллипс, окружность):

Type TShapeType = (stRectangle, stSquare, stRoundRect, stRoundSquare, stEllipse, stSircle);

Контур фигуры определяется свойством *Pen*. Это составное свойство. Раскрыть это свойство. Свойство *Pen.Style* определяет способ вычерчивания линий. Цвет контура определяется свойством *Color*: *ClRed* – ярко-красный, *ClYellow* – ярко-желтый, *ClLime* – ярко-зеленый. Мы хотим, чтобы первоначально фигуры были не закрашенными, т. е. прозрачными. Прозрачность задается свойством *Style* → *bsClear*.

Рис. 3.2



Фигура полностью занимает все пространство компонента. Если задан круг или квадрат, а размеры компонента по горизонтали и по вертикали отличаются, фигура чертится с размером меньшего измерения.

Параметры закраски определяются свойством *Brush*. Свойство *Brush.Color* определяет цвет заполнения области. Свойство *Brush.Style* определяет стиль заполнения области.

На окне формы разместить 3 компонента Shape. Ширина и высота каждого компонента по 65. Форма фигуры задается свойством Shape. Для каждой фигуры, из раскрывающегося списка, выбрать значение *StCircle*.

Фигуры могут реагировать на наведение мыши. Событие MouseMove можно было бы обработать для каждой фигуры. Фигуры круглые, но маркеры изменения размера располагаются по сторонам прямоугольника. Система Delphi продолжает считать фигуры прямоугольными. Это означает, что при наведении указателя мыши на уголок фигуры, лампочки будут зажигаться. Чтобы этого не произошло, лампочки надо «выключить», т.е. свойствам

Enabled (Включен) каждой фигуры задать значение *false* (Нет). Для отключенных объектов никаких событий не возникает. Если указатель мыши будет наведен на лампочку, событие MouseMove будет передано другому компоненту, родительскому, в котором располагается фигура. В нашем случае это форма.

Для формы Form1 установить свойства:

Caption → Светофор, Height → 300 Width → 120.

Окно станет таким узким, что кнопки управления размером окна, присутствующие в строке заголовка, будут мешать увидеть заголовок окна целиком. Отображение стандартных кнопок задается составным свойством *BorderIcons* (Служебные кнопки). Открыть это свойство и для *biMinimize* (сворачивание формы) и *biMaxmize* (разворачивание формы на весь экран) установить значение *false.* На форме ничего не изменится, но при выполнении программы эти кнопки (сворачивания и разворачивания) исчезнут.

Запретить изменение размеров окна. Изменение размеров окна зависит от типа его рамки. Этот тип задается свойством *BorderStyle* (стиль рамки). Выбрать в раскрывающемся списке значение: *bsSingle* (тонкая) – рамка шириной в 1 пиксель. Такое окно не может изменять размеры.

Расположить объекты ровно и симметрично можно с помощью команд выравнивания.

Выделить все три фигуры нажимая на Shift, выполнить команду Edit → Align (Правка → выровнять). Открывается диалоговое окно Alignment (Выравнивание), где установить:

по горизонтали *Center on window* (центрировать в окне),

по вертикали *Space Equally* (сравнение промежутками).

Для формы обработать событие OnMouseMove. Если указатель наведен на 1-ую лампу она зажигается, если нет, то гаснет. То же самое для остальных лампочек, т.е. для каждой лампочки выполняются одни и те же действия. Их можно оформить в виде отдельной функции:

function Onshape(Sh:Tshape;X,Y:integer):TbrushStyle;

В функцию передаются следующие параметры: имя фигуры, координаты указателя мыши. Функция должна вернуть способ заполнения фигуры. Для сплошной закраски используется свойство Brash.Style → BsSolid , для гашения Brash.Style → BsClear .

Функция Onshape вызывается из процедуры FormMouseMove и поэтому должна быть описана раньше процедуры. Чтобы определить наведен ли указатель мыши на фигуру, нужно вычислить радиус фигуры. Он составляет половину ее высоты или ширины: *r:=sh.width div 2;* Чтобы найти координаты центра фигуры, к значениям левой и верхней границ нужно прибавить радиус: *cx:=sh.left+r; cy:=sh.top+r;* Далее нужно вычислить квадрат расстояния указателя мыши от центра фигуры. Если квадрат расстояния меньше r^2 , т.е. указатель мыши наведен на фигуру, возвращаемый стиль закраски будет *bsSolid*, иначе - *BsClear*.

Текст модуля

```
unit  svetofor;
interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics,
Controls, Forms,
  Dialogs, StdCtrls, Buttons, ExtCtrls;

type
  TForm1 = class(TForm)
    Shape1: TShape;
    Shape2: TShape;
    Shape3: TShape;
    procedure FormMouseMove(Sender: TObject; Shift:
TShiftState; X, Y: Integer);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

// Текст функции

function Onshape(Sh:Tshape;X,Y:integer):TbrushStyle;
var r,cx,cy:integer;
begin
       r:=sh.width div 2;  //  r – радиус фигуры
       cx:=sh.left+r;     //  cx, cy - координаты центра
    фигуры
       cy:=sh.top+r;
       onshape:=bsClear;   // функция возвращает стиль
    закраски
       if (x-cx)*(x-cx)+(y-cy)*(y-cy)<r*r then
    Onshape:=bsSolid;
end;
// Текст процедуры обработки события OnMouseMove
procedure TForm1.FormMouseMove(Sender: TObject; Shift:
TShiftState; X,  Y: Integer);
begin
       shape1.Brush.Color:=clred;
```

```
        shape1.Brush.style:=Onshape(shape1,x,y);
        shape2.Brush.Color:=clyellow;
        shape2.Brush.style:=Onshape(shape2,x,y);
        shape3.Brush.Color:=cllime;
        shape3.Brush.style:=Onshape(shape3,x,y);
end;
end.
```

3.2. События от клавиатуры

События от мыши получают любые потомки класса *TControl*. В отличие от этого события от клавиатуры получают только некоторые оконные компоненты (потомки *TWinControl*). Обработка событий связана со следующими свойствами этих компонентов:

```
type
    TSiftState = set of (ssSift, ssAlt, ssCtrl,
 ssLeft, ssRight, ssMiddle, ssDouble);
    TKeyEvent = procedure (Sender: TObject; var Key:
 Word;
    Shift: TSiftState) of object;
    TKeyPressEvent = procedure (Sender: TObject;
     var Key: Char )  of object;
    property OnKeyDown: TKeyEvent;
    property OnKeyUp: TKeyEvent;
    property OnKeyPress: TKeyPressEvent;
```

Обработчики событий *OnKeyDown* и *OnKeyUp* реагируют на нажатие большинства клавиш клавиатуры, а обработчик *OnKeyPress* – только на нажатие алфавитно-цифровых клавиш.

Эти события возникают при нажатии и отпускании клавиш клавиатуры. Обработчику передаются:

Sender – источник сообщения;

Shift - состояние специальных клавиш и кнопок мыши во время нажатия (отпускания);

Key - в обработчиках *KeyEvent* содержит виртуальный код клавиши (константы вида vk_xx, например, vk_f1, vk_escape и т.п.), фактически виртуальный код – это уникальный числовой идентификатор клавиши (виртуальные коды некоторых клавиш приведены в таблице 3.1). В обработчиках *KeyPressEvent* параметр *Key* – ASCII символ.

Таб. 3.1. Виртуальные коды некоторых клавиш

Код	Значение	Клавиша
vk_Tab	8	Tab
vk_Return	13	Enter
vk_Shift	16	Shift
vk_Escape	27	Escape
vk_0. .vk_9	48 . . 57	0 . . 9
vk_End	35	End
vk_Left	37	Курсор влево
vk_Right	39	Курсор вправо
vk_Down	40	Курсор вниз
vk_Delete	46	Delete

Пример 1. Создать программу, которая формирует массив A(n) из случайных целых чисел, вычисляет среднее арифметическое элементов массива, определяет минимальный и максимальный элементы массива. Число элементов массива ввести с компонента Edit, о завершении ввода программе сообщить нажатием на клавишу Enter. Результаты вывести на компонент Memo. Элементы массива вывести по 10 на каждой строке.

На окне формы компоненты расположить в следующем порядке (рис. 3.3):

1). Компонент панель Panel. Для компонента Panel установить следующие свойства: *Align – alBottom*, *Caption –* очистить, *Height –* 84 .

2). На панели расположить строку ввода *Edit* и кнопку *Bitbtn*. Для компонента *Edit* установить свойства: *Text –* очистить ; *Font –* установить стиль шрифта, размер и цвет. Для *bitbtn* установить свойство *Kind – BkClose*.

3). На окне формы расположить компонент *Label*. Для компонента Label установить свойства: *Align – alBottom*, *Aligment – taCenter*, *Caption –* 'Введите число элементов массива', *Height –* 30, *Font*.

4). На форме расположить компонент *Memo*. Для Memo установить свойства: *Align – alClient*, *ScrollBars – ssBoth*, *Font* – размер и цвет.

Для компонента Edit создать процедуру обработки события ***OnKeyPress*.** Откроется окно кода программы с заголовком обработки события, словами *begin, end*:

```
procedure TForm1.Edit1KeyPress(Sender: TObject; var Key: Char);
begin


end;
```

Напишем текст процедуры обработки события OnKeyPress. Код клавиши Enter равен 13. В программе можно проверить условие:

If key=#13 then begin key:=#0; …… end;

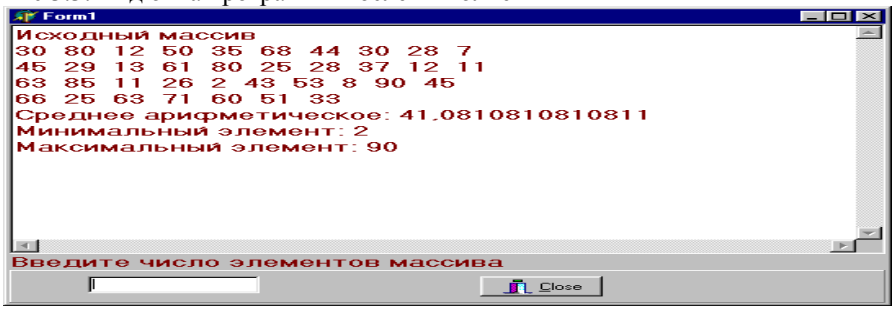
Присваивание значения **#0** параметру key означает отмену нажатия клавиши.

// Текст процедуры обработки события OnKeyPress

```
   Procedure TForm1.Edit1KeyPress(Sender: TObject; var Key:
Char);
        var a:array[1..100] of integer;
        sum,max,min,i,n:integer;  s:string;
   begin
        randomize;
        memo1.clear;
        // проверяем  нажата ли клавиша Enter
        if key=#13 then
        begin
          key:=#0;
          n:=strtoint(edit1.text);
          edit1.clear; edit1.SetFocus;
          // формируем массив из случайных чисел
          for i:=1 to n do a[i]:=random(101);
          // вычисляем сумму элементов массива
          // находим максимальный и минимальный элементы
          sum:=a[1]; min:=a[1]; max:=a[1];
          for i:=2 to n do
            begin
              sum:=sum+a[i];
              if a[i]<min then min:=a[i];
              if a[i]>max then max:=a[i];
            end;
          s:='';
          // выводим массив на экран по 10 элементов на каждой
       строке
          memo1.lines.add('Исходный массив');
          for i:=1 to n do
            begin
              s:=s+inttostr(a[i])+'  ';
              if i mod 10 =0 then
                begin
                  memo1.lines.add(s); s:='';
                end;
            end;
          // выводим оставшиеся элементов массива,
          //  если число элементов не кратно 10
           memo1.lines.add(s);
          // выводим результаты вычислений
```

```
        memo1.lines.add('Среднее арифметическое:
    '+floattostr(sum/n));
        memo1.lines.add('Минимальный элемент:
    '+inttostr(min));
        memo1.lines.add('Максимальный элемент:
    '+inttostr(max));
     end;
end;
```

Рис 3.3. Вид окна программы после выполнения



Пример 2. Создать программу, которая создает массив A из n случайных целых чисел в диапазоне от 0 до 100, отображает массив на компоненте ListBox, вычисляет среднее арифметическое выбранных элементов массива. Число элементов массива ввести с помощью компонента Edit, о завершении ввода программе сообщить нажатием на клавиши Alt+N. Чтобы вычислить среднее арифметическое выбранных элементов, нажать на кнопку OK.

Компонент ListBox представляет собой список выбора, с помощью которого можно выбрать один или несколько элементов. Свойства компонента:

Colomns – определяет количество колонок элементов в списке;

ExtendedSelect – если *ExtendedSelect=true* и *MultiSelect=true,* выбор элемента без одновременного нажатия *Ctrl* или *Alt* отменяет предыдущий выбор;

ItemIndex – содержит индекс сфокусированного элемента;

Items – содержит набор строк, показываемых в компоненте;

MultiSelect – разрешает / отменяет выбор нескольких элементов;

SelCount – содержит количество выбранных элементов;

Selected[I] – содержит признак выбора элемента с индексом I (первый элемент имеет индекс 0).

Создание элементов списка реализуется с помощью методов его свойства *Items – Add, Append, Insert.*

Для компонента *ListBox* установить следующие свойства:

Colomns – 1;

MultiSelect – true.

Для компонента *Edit* создать процедуру обработки события *OnKeyDown.*

Текст модуля

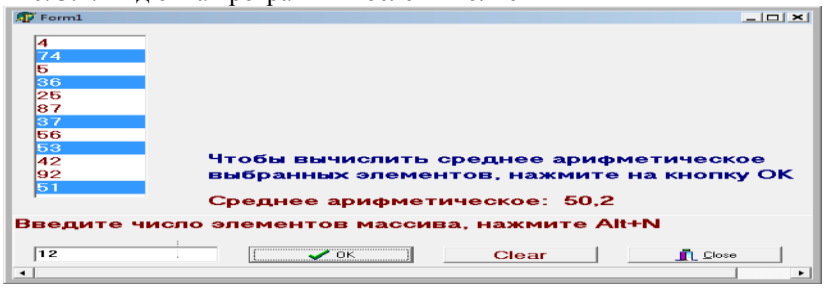
```
unit  mas3;
interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics,
Controls, Forms,
  Dialogs, MPlayer, ExtCtrls, StdCtrls, Buttons;

type
  TForm1 = class(TForm)
    Panel1: TPanel;
    Edit1: TEdit;
    ListBox1: TListBox;
    BitBtn1: TBitBtn;
    BitBtn2: TBitBtn;
    Label3: TLabel;
    BitBtn3: TBitBtn;
    BitBtn4: TBitBtn;
    Label2: TLabel;
    Label1: TLabel;
    procedure Edit1KeyDown(Sender: TObject; var Key: Word;
      Shift: TShiftState);
    procedure BitBtn2Click(Sender: TObject);
    procedure BitBtn1Click(Sender: TObject);
    procedure BitBtn4Click(Sender: TObject);
  private
            { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

Implementation
// в разделе Implementation объявляем глобальные переменные
     var a:array[1..100] of integer;
     sum,i,n:integer; sr: real; s:string;
{$R *.dfm}
// Процедура обработки события OnKeyDown
procedure TForm1.Edit1KeyDown(Sender: TObject; var Key:
Word;
Shift: TShiftState);
begin
     randomize;
```

```
    listbox1.Columns:=1;
    // проверяем, нажаты ли  клавиши  N и Alt, если да,
    //  то создаем массив и отображаем на Listbox
    if (key=ord('N')) and (ssAlt in shift) then
      begin
        key:=0;
        n:=strtoint(edit1.text);
        edit1.SetFocus;
        for i:=1 to n do a[i]:=random(101);
        for i:=1 to n do
     listbox1.Items.Add(inttostr(a[i]));
      end;
end;

// Процедура обработки события OnClick для кнопки "OK"
procedure TForm1.BitBtn4Click(Sender: TObject);
begin
    if listbox1.SelCount=0 then
      begin
        label2.Caption:='В ListBox нет выбранных
     элементов'; exit;
      end;
    sum:=0;
    // вычисляем сумму выбранных элементов
    for i:=0 to n-1 do
    if listbox1.selected[i]=true  then
 sum:=sum+strtoint(listbox1.Items[i]);
    sr:=sum/listbox1.SelCount;   // вычисляем ср.
 арифметическое
    label2.Caption:='Среднее арифметическое:
 '+floattostr(sr);
end;
procedure TForm1.BitBtn2Click(Sender: TObject);
begin
    edit1.Clear;
    listbox1.Items.Clear;
end;
procedure TForm1.BitBtn1Click(Sender: TObject);
begin
    close;
end;
end.
```

Процедура обработки события OnKeyDown проверяет, нажаты ли клавиши N и Alt, если да, то создает массив и отображает на компоненте Listbox. Процедура обработки события нажатия на кнопку "OK", проверяет, есть ли на компоненте Listbox выбранные элементы. Если нет, об этом выводится сообщение и осуществляется выход из процедуры. Если да,

вычисляет среднее арифметическое выбранных элементов и выводит его значение на метку Label2.

Рис. 3.4. Вид окна программы после выполнения



Задания для самостоятельного выполнения.

1. Создать программу, которая формирует массив A из n случайных целых чисел в диапазоне от 0 до 200, минимальный и максимальный элементы массива меняет местами. Число элементов массива ввести с помощью компонента Edit, о завершении ввода программе сообщить нажатием на клавишу Enter. Исходный массив и результат вывести на компонент Memo. Элементы массива вывести по 10 на каждой строке.

2. Сформировать одномерный массив из n элементов по следующему правилу: $a_1=1$, $a_2=1$ $a_i=2a_{i-2}+4a_{i-1}$; i=3,…,n. Подсчитать количество и сумму всех четных элементов массива. Число элементов массива ввести с помощью компонента Edit, о завершении ввода программе сообщить нажатием на клавишу Enter. Исходный массив и результат вывести на компонент Memo.

3. Сформировать одномерный массив из n элементов по следующему правилу: $a_1=5$, $a_2=9$, $a_i=a_{i-2}+3a_{i-1}$; i=3,…,n. Массив отобразить на компоненте ListBox. Подсчитать, сколько элементов, больше пятого, среди выбранных элементов массива. Число элементов массива ввести с помощью компонента Edit, о завершении ввода программе сообщить нажатием на клавиши Alt+N.

4.Сформировать одномерный массив из n элементов по следующему правилу: $a_1=3$, $a_i=5a_{i-1}/4$; i=2,…,n. Массив отобразить на компоненте ListBox. Вычислить произведение индексов выбранных элементов массива. Число элементов массива ввести с помощью компонента Edit, о завершении ввода программе сообщить нажатием на клавиши Alt+N.

5. Дан целочисленный одномерный массив, состоящий из n элементов. Массив отобразить на компоненте ListBox. Подсчитать количество и сумму элементов массива, кратных 5, среди выбранных элементов массива. Число

элементов массива ввести с помощью компонента Edit, о завершении ввода программе сообщить нажатием на клавишу Enter.

3.3. Событие от таймера

Компонент Timer (страница System), служит для отсчета интервалов реального времени. Его свойство Interval определяет интервал времени в миллисекундах, который должен пройти от включения таймера до наступления события **OnTimer.** Таймер включается при установке в свойство Enabled true. Раз, включенный таймер, всегда возбуждает событие OnTimer, пока Enabled не примет значение False.

Пример 1. Нарисовать светофор с тремя лампочками, которые загораются по очереди.

Для рисования лампочек светофора использовать компонент *Shape* (рис. 3.2).

На форме установить таймер. Свойство *Tag* определяет целочисленный параметр, который не используется Delphi и которым программист может распоряжаться по своему усмотрению. Это свойство используем для организации счетчика обращений процедуре TForm1.Timer1Timer. При каждом вхождении в процедуру значение *timer1.Tag* увеличиваем на 1. Если при делении счетчика на число лампочек, т.е. на 3 , остаток равен нулю, первую фигуру закрашиваем в ярко-красный цвет. Если остаток равен единице, вторую фигуру закрашиваем в ярко-желтый цвет. Если остаток равен двум, третью фигуру окрашиваем в ярко- зеленый цвет.

В свойство *Interval* установим интервал времени, в течении которого будут гореть лампочки светофора.

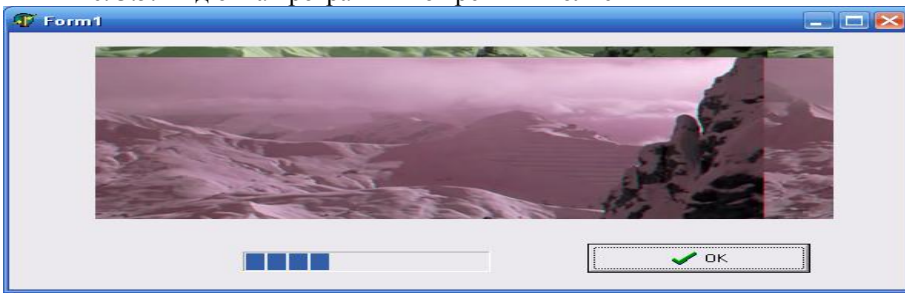
Текст процедуры обработки события OnTimer

```
procedure TForm1.Timer1Timer(Sender: TObject);
begin
    { все три фигуры делаем прозрачными, т.е. не
 закрашенными }
    shape1.Brush.Style:=bsClear;
    shape2.Brush.Style:=bsClear;
    shape3.Brush.Style:=bsClear;
    {  проверяем счетчик  }
    if timer1.Tag mod 3 =0 then
      begin
        shape1.Brush.Color:=clRed;
        shape1.Brush.Style:=bsSolid;
      end;
    if timer1.Tag mod 3 =1 then
      begin
```

```
          shape2.Brush.Color:=clYellow;
          shape2.Brush.Style:=bsSolid;
        end;
      if timer1.Tag mod 3 =2  then
        begin
          shape3.Brush.Color:=clLime;
          shape3.Brush.Style:=bsSolid;
        end;
      timer1.Tag:=timer1.Tag+1; // счетчик увеличиваем на 1
end;
```

Пример 2. Создать программу для отображения видеоклипа в течение достаточно длительного времени. На окне формы расположить компоненты Animate, ProgressBar, Timer, кнопку BitBtn.

Рис. 3.5. Вид окна программы во время выполнения



Компонент Animate (стр.Win32) представляет собой проигрыватель видеоклипов формата **AVI.** Компонент воспроизводит видео часть файла AVI и игнорирует его звуковое сопровождение. Он способен показывать лишь несжатое изображение или изображение, сжатое по методу **RLE** (Run – Length Encoding). Изображение воспроизводится в отдельном потоке команд, что освобождает ресурсы программы для выполнения необходимой работы на фоне демонстрации клипа.

Свойство *Active* разрешает / запрещает демонстрацию клипа. Во время демонстрации содержит true.

Свойство *FileName* используется для записи имени файла видеоклипа.

Для запуска видеоклипа из программы можно использовать команду

animate1.Active:=true; Для остановки видеоклипа *animate1.stop;* или *animate1.Active:=false;*

Компонент ProgressBar предназначен для отображения хода выполнения длительного по времени процесса. Свойства:

Max (тип integer) определяет максимальное значение диапазона изменения свойства Position;

Position (тип integer) – содержит текущее значение отображаемой величины.

Для компонента *ProgressBar1* в свойство *Visible* установить *false*, для *Timer1*. *Enabled* установить *false*.

Для кнопки *BitBtn1* создать обработчик события *OnClick*, где указать имя файла клипа, запустить клип, показать ProgressBar, включить Timer.

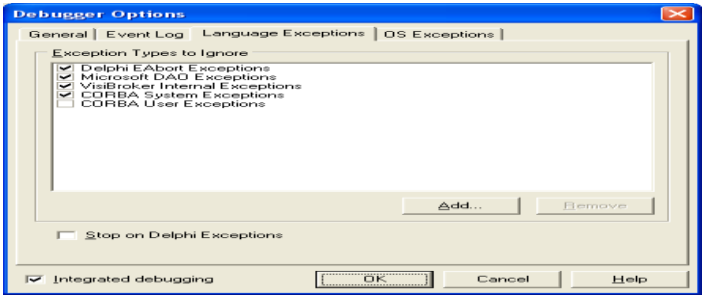
Создать обработчик события *OnTimer*, где при каждом вхождении в процедуру значение Position увеличивать на 1. Когда значение Position станет больше или равно Max, выключить Timer, спрятать ProgressBar, остановить клип.

4.Обработка исключений. Класс EXCEPTION

Класс Exception является прямым потомком базового класса TObject. Вместе со своими потомками он предназначен для обработки исключительных ситуаций (исключений), возникающих при некорректных действиях программы: например, в случае деления на ноль, при попытке открыть несуществующий файл и др.

Совет: При работе в среде Delphi при каждом исключении среда перехватывает управление программой. Полезно бывает отменить такое поведение среды. Для этого вызвать опцию *Tools – Debugger* и на странице *Language Exceptions* убрать флажок *Stop on Delphi Exceptions*.

Рис. 4.1.



4.1. Защищенные блоки

Для обработки исключений в Delphi предусмотрен механизм защищенного блока.

```
Try                          try
< операторы >                < операторы >
Except                       Finally
< Обработка исключений >     < операторы >
Else                         End;
< операторы >
end;
```

Защищенный блок начинается зарезервированным словом *try* **(**попытаться выполнить) и завершается словом *end*. Существуют два типа защищенных блоков – *Except* (исключить) и *Finally* (в завершении), отличающихся способом обработки исключения. В блоке *Except* порядок выполнения операторов таков: сначала выполняются операторы секции *Try… Except*; . Если операторы выполнены без возникновения исключительной ситуации, работа защищенного блока на этом прекращается и управление получает оператор, стоящий за словом *end.* Если при выполнении части *try* возникает исключение, управление получает обработчик в секции *Except*, а если таковой не найден – первый из операторов, стоящих за словом *Else*.

В блоке *Finally* операторы в секции *Finally… end* получают управление всегда, независимо от того, возникло ли исключение в секции *try … Finally* или нет. Если исключение возникло, все операторы в секции *try…finally*, стоящие за виновником исключения, пропускаются и управление получает первый оператор секции *Finally… end*. Если исключения не было, этот оператор получает управление после выполнения последнего оператора секции *try…finally*.

Обработчики исключений в блоке *Except* имеют такой синтаксис:

On < класс исключения > **do** <оператор >;

Здесь **on, do** – зарезервированные слова; < класс исключения > - класс обработки исключения; <оператор > - любой оператор, кроме оператора передачи управления goto на метку вне блока *Except*.

Поиск нужного обработчика осуществляется с начала списка вниз, пока не встретиться класс, способный обрабатывать исключение данного типа. Если подходящего класса не обнаружено, управление передается операторам, стоящим за словом *Else.*

Защищенные блоки могут вкладываться друг в друга на неограниченную глубину.

Важно помнить, что ищется самый первый из возможно нескольких обработчиков, класс которого способен обрабатывать данное исключение. Если, например, в списке первым стоит *EAbort* , который может обработать любое исключение, ни один стоящих за ним обработчиков никогда не получит управления. Точно также, если указан обработчик класса *EIntError*, за ним бесполезно размещать обработчики *EDivByZero, EIntOverflow.*

```
Try
….
except
// не имеет смысла делать так
On EIntError do …. ;
On ERangeError do … ;
On EDivByZero  do …;
// надо так
On ERangeError do … ;
```

```
On EDivByZero  do …;
On EIntError do
End;
```

4.2. Стандартные классы исключений

Таблица 4.1. Некоторые стандартные классы исключений

Класс	Родитель	Обрабатываемое исключение
EAbort	*Exception*	Реализует «тихую» (без какого-либо сообщения) обработку любого исключения
EConvertError	*Exception*	Ошибка преобразования в функциях StrToInt или StrToFloat
EIntError	*Exception*	Любая ошибка в целочисленных вычислениях
EDivByZero	*EIntError*	Ошибка целочисленного деления на ноль
ERangeError	*EIntError*	Целочисленный результат превышает емкость целого типа данных
EIntOverflow	*EIntError*	Ошибка целочисленного переполнения
EMathError	*Exeption*	Любая ошибка при вычислениях с плавающей точкой
EZeroDivide	*EMathError*	Вещественное деление на ноль
EFCreateError	*EStreamError*	Ошибка при создании файла
EinOutError	*Exception*	Любая ошибка в файловых операциях
EMenuError	*Exeption*	Ошибка при работе программы с меню
EOverflof	*EMathError*	Результат операций с плавающей точкой слишком велик, чтобы уместиться в регистрах сопроцессора

Пример. Создать программу, которая над двумя данными числами производит указанное арифметическое действие.

Для указания знаков арифметических действий использовать компонент ComboBox. ***Компонент Combobox*** (страница Standard) - комбинированный список выбора. Представляет собой комбинацию списка выбора и текстового редактора. С помощью списка выбора пользователь может выбрать один или несколько элементов выбора. Свойства:

Items (mun string) содержит набор строк, показываемых в компоненте;

Multiselect(mun boolean) разрешает, отменяет выбор нескольких элементов;

Itemindex(mun integer) содержит индекс сфокусированного элемента. Первый элемент имеет индекс 0. Если не выбрана ни одна строка itemindex=-1.

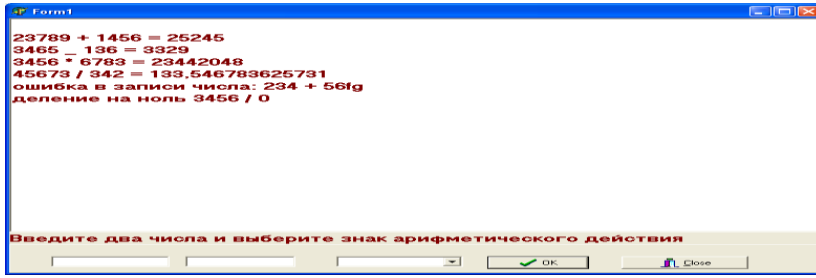
На форме расположить компонент Panel, на панели расположить две строки ввода (Edit), компонент Combobox и две командные кнопки Ok и Close.

Над панелью расположить компонент Label и многострочный редактор Memo.

Текст процедуры обработки события OnClick

```
procedure TForm1.BitBtn1Click(Sender: TObject);
var
    x,y,z:real;
    i: integer;
    s:string;
begin
    i:=combobox1.ItemIndex;
    if (edit1.text='')or(edit2.text='')or(i<0) then
      begin
        memo1.lines.add('Данные не введены');   exit;
      end;
    s:=trim(edit1.text)+' '+combobox1.items[i]+'
 '+trim(edit2.text);
    try    //начало защищенного блока
    x:=strtofloat(trim(edit1.text));
    y:=strtofloat(trim(edit2.text));
    case i of
      0: z:=x+y;
      1: z:=x-y;
      2: z:=x*y;
      3: z:=x/y;
    end;
    //  вывод результатов
     memo1.lines.add(s+'  =  '+floattostr(z))
    // обработка исключений
    except
    on EConvertError do
        memo1.lines.add(' ошибка в записи числа: '+s);
    on EZeroDivide do
        memo1.lines.add('деление на ноль  '+s);
    end;     // конец защищенного блока
        edit1.clear; edit2.clear;
        edit1.setfocus;
        combobox1.itemindex:=-1;
end;
```

Рис. 4.3. Вид окна программы после выполнения



В процедуре обработки события OnClick предусмотрены обработки исключений. При выполнении преобразования строк, введенных с помощью компонентов Edit1 и Edit2, при выполнении функции *StrToFloat* возможно возникновение ошибочной ситуации, если преобразуемые данные не могут быть приведены к требуемому виду. Такая ситуация обрабатывается с помощью обработчика исключений *EConvertError*. При возникновении исключения на экран выводится сообщение об ошибке, и управление получает первый оператор, стоящий после защищенного блока.

При выполнении инструкции z:= x/y; , если делитель равен нулю, возникает ошибочная ситуация. Такая ситуация обрабатывается с помощью обработчика исключений *EZeroDivide*. При возникновении исключения на экран выводится сообщение об ошибке, и управление получает первый оператор, стоящий после защищенного блока.

Задания для самостоятельного выполнения. Предусмотреть обработку исключительных ситуаций. Результаты вывести на многострочный редактор Memo.

1. Дано натуральное число n. Найти произведение ненулевых цифр этого числа.
2. Протабулировать функцию $F(x)=\begin{cases} -x^2+3x+9, \text{если } x\le 3 \\ \dfrac{x}{x^2-1}, \text{если } x>3 \end{cases}$ на отрезке [a; b] с шагом h.
3. Вычислить сумму $s=x+\sum_{i=1}^{\infty}\frac{x^{2i+1}}{(2i+1)!}$ с точностью ε.
4. Вычислить сумму $S=\sum_{i=1}^{n}\frac{x+i}{3i^3}$.

5. Дано натуральное число n. Проверить, является ли оно трехзначным, кратным пяти.

6. Дано натуральное число n. Найти сумму квадратов всех его цифр.

5. Различные способы ввода и обработки одномерного массива

Под вводом массива понимается процесс получения от пользователя (или из файла) во время работы программы, значений элементов массива. Далее рассматриваются три варианта организации ввода массива с использованием компонентов Memo, Edit и таблицы StringGrid.

5.1. Ввод одномерного массива с помощью многострочного редактора Memo

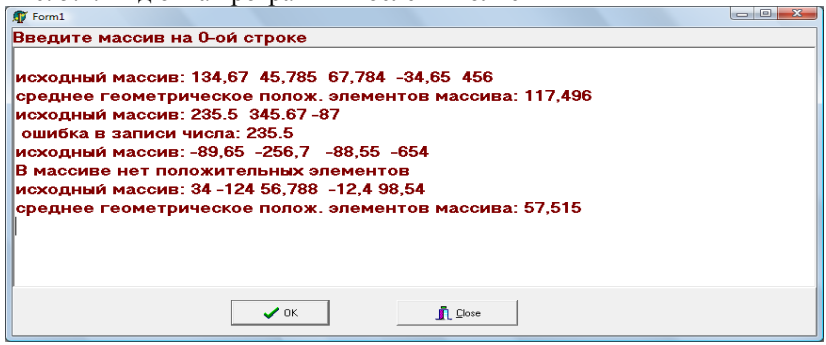
Пример 1. Дан числовой массив. Элементы массива ввести на нулевой строке многострочного редактора Memo. При вводе элементы массива разделить одним или несколькими пробелами. Найти среднее геометрическое положительных элементов массива.

Текст процедуры обработки события OnClick

```
procedure TForm1.BitBtn1Click(Sender: TObject);
    var i, j, n, k:integer;  p,  sg: real;
    s,b:string; a:array[1..10] of real;
begin
    if  memo1.Lines[0]='' then
      begin
        memo1.Lines.Add('массив не введен'); exit
      end;
    s:=memo1.Lines[0];
    memo1.SetFocus;  memo1.Lines.Add('исходный массив:
 '+s);
    memo1.Lines[0]:='';
    //удаление лишних пробелов между числами
    i:=1;  // i – номер символа в строке
    while i<=length(s)-1 do
      begin
        if (s[i]=' ') and (s[i+1]=' ') then
          begin
            delete(s,i,1); i:=i-1;
          end;
        i:=i+1;
      end;
    try
```

```
    p:=1;  k:=0;  // p – произведение,  к – число
 положительных элементов
     b:='';   i:=1;  // i – номер символа в строке
    j:=1;         // j – индекс элемента в массиве
    //выделение чисел и запись в массив
    while i<=length(s) do
      begin
        while (s[i]<>' ') and (i<=length(s))  do
          begin
            b:=b+s[i]; i:=i+1;
          end;
        a[j]:=strtofloat(b);  i:=i+1; j:=j+1; b:='';
      end;
    n:=j-1;
    for i:=1 to n do
      if a[i]>0 then begin p:=p*a[i];  k:=k+1; end;
                sg:=exp((1/k)*ln(p));
    memo1.Lines.Add('среднее геометрическое положительных
 элементов массива: '+ floattostrf(sg,ffFixed,7,3));
    memo1.SetFocus;
    except
      on EConvertError do   memo1.lines.add(' ошибка в
   записи числа: '+b);
      on  EMathError do   memo1.lines.add('В массиве нет
   положительных элементов');
     end;
end;
```

Рис. 5.1. Вид окна программы после выполнения



Процедура обработки события OnClick проверяет, введены ли данные на нулевую строку компонента Memo, если нет, то выводится об этом сообщение и осуществляется выход из процедуры. Если да, введенные данные считываются с нулевой строки компонента Memo и записываются в строку s и выводятся на компонент Memo1. Выделяются элементы массива

из строки s, для этого организуются два вложенных цикла: внешний и внутренний. Внешний цикл выполняется, пока номер обрабатываемого символа меньше длины строки s. Внутренний цикл выполняется пока i-ый символ не пробел и пока не обработаны все символы строки. Внутренний цикл выделяет числа из строки s и записывает в массив. Вычисляется произведение и определяется число положительных элементов. Вычисляется среднее геометрическое положительных элементов массива.

При преобразовании введенных данных из строчного типа в числовой возможно возникновение ошибочной ситуации, если введенные данные не могут быть приведены к требуемому типу. Такую ситуацию обрабатывает обработчик исключений *EConvertError*. При возникновении исключения на экран выводится сообщение об ошибке и процедура обработки события OnClick завершает работу.

Если в массиве нет положительных элементов при выполнении инструкции sg:=exp((1/k)*ln(p)); (где p – произведение положительных элементов массива, а k - их количество возникает ошибочная ситуация. Такую ситуацию обрабатывает обработчик исключений EMathError. При возникновении исключения на экран выводится сообщение «В массиве нет положительных элементов» и процедура обработки события OnClick завершает работу.

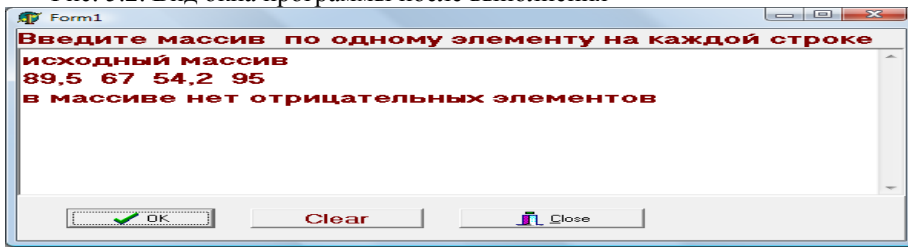
Пример 2. Дан числовой массив. Массив ввести с помощью компонента Memo по одному элементу на каждой строке, начиная с нулевой. Найти среднее арифметическое отрицательных элементов массива. Исходный массив и результат вывести на многострочный редактор Memo.

Текст процедуры обработки события OnClick для кнопки “OK”

```
 procedure TForm1.BitBtn1Click(Sender: TObject);
    var a:array[1..20]of real;
    n,i,k:integer; s1:string; sr, s:real;
begin
    n:=memo1.Lines.Count;   { определяем количество
 введенных строк}
    if n=0 then
      begin
        memo1.Lines.add('массив не введен');
        exit; { Выход из процедуры }
      end;
    try
    for i:=1 to n do a[i]:=strtofloat(memo1.lines[i-1]);
    memo1.Clear;
    memo1.lines.Add('исходный массив');
    s1:='';
    for i:=1 to n do s1:=s1+floattostr(a[i])+'  ';
```

```
    memo1.Lines.add(s1);
    // находим количество и сумму отрицательных элементов
 массива
    k:=0;  s:=0;
    for i:=1 to n do
    if a[i]<0 then
      begin
        s:=s+a[i]; k:=k+1;
      end;
    sr:=s/k;
    // ср. арифм. выводим с двумя знаками после запятой
    memo1.Lines.Add('среднее арифметическое отрицательных
 элементов: '+ floattostrf(sr,ffFixed,6,2));
    except   // обработка ошибочных ситуаций
    on EConvertError do // обработка ошибки преобразования
        memo1.lines.add(' ошибка в записи числа'+b);
    on  EMathError do { обработка ситуации , когда делимое
 и делитель равны нулю}
     memo1.lines.add('в массиве нет отрицательных
 элементов ')
    end;
end;
```

Рис. 5.2. Вид окна программы после выполнения



Процедура обработки события OnClick определяет количество введенных элементов, если данные не введены, выводит об этом сообщение и осуществляется выход из процедуры. Введенные данные записываются в массив, и исходный массив выводится на экран. Вычисляется среднее арифметическое отрицательных элементов.

При преобразовании введенных данных из строчного типа в числовой возможно возникновение ошибочной ситуации, если введенные данные не могут быть приведены к требуемому типу. Данная ситуация обрабатывается с помощью обработчика исключений *EConvertError*. При возникновении исключения на экран выводится сообщение об ошибке и процедура обработки события OnClick завершает работу.

Если в массиве нет отрицательных элементов, то при вычислении среднего арифметического делимое и делитель равны нулю. При

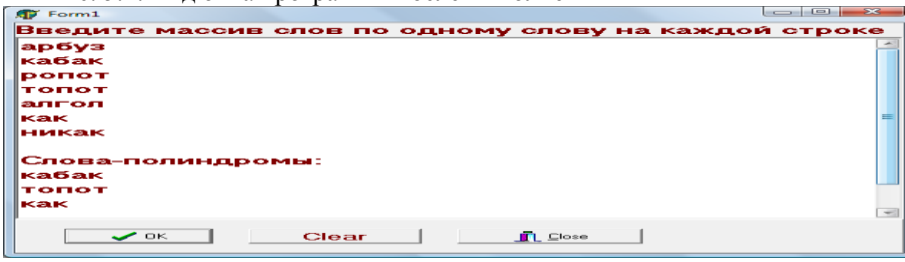
выполнении инструкции sr:=s/k; возникает ошибочная ситуация. Данная ситуация обрабатывается с помощью обработчика исключений *EMathError*. При возникновении исключения на экран выводится сообщение «В массиве нет отрицательных элементов» и процедура обработки события OnClick завершает работу.

Пример 3. Дан массив слов. Массив ввести с помощью компонента Memo по одному слову на каждой строке. Выяснить, есть ли в этом массиве слова-палиндромы (т. е. слова, одинаково читающиеся слева направо и справа налево). Если есть, вывести их на экран.

Текст процедуры обработки события OnClick для кнопки «ОК»

```
procedure TForm1.BitBtn1Click(Sender: TObject);
    var
    b: array[1..10] of string ;
    i, n, k, j: integer;
    s1:string;
begin
    k:=0;
    memo1.setfocus;
    n:=memo1.Lines.Count;         { определяем количество
 введенных строк }
    if n=0 then
      begin
        memo1.Lines.add('массив не введен');  exit;
      end;
    for i:=1 to n do b[i]:=memo1.lines[i-1];
    memo1.lines.Add('Слова-полиндромы:');
    for i:=1 to n do
      begin
        s1:='';   // в s1 записываем перевернутое слово
        // i – номер элемента массива b, j – номер символа
     в слове b[i]
        for j:= length(b[i]) downto 1 do s1:=s1+b[i,j];
        if s1=b[i] then
          begin  // слово-полиндром выводим на экран
            memo1.lines.Add(b[i]); k:=k+1;
            // к –счетчик слов-полиндромов
          end;
      end;
    if k=0 then memo1.lines.Add('Таких слов нет');
end;
```

Рис. 5.4. Вид окна программы после выполнения



5.2. Ввод одномерного массива с помощью редактора Edit

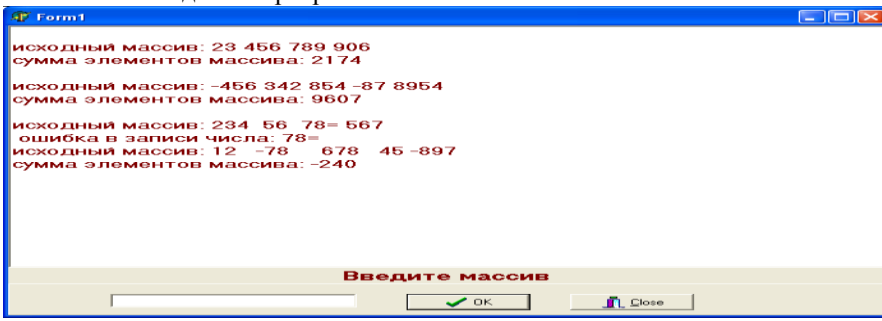
Пример. Дан массив из целых чисел. Найти сумму элементов массива. Элементы массива ввести с помощью однострочного редактора Edit. При вводе элементы массива разделить одним или несколькими пробелами.

Текст процедуры обработки события OnClick для кнопки «OK”

```
procedure TForm1.BitBtn1Click(Sender: TObject);
    var sum,i,j,n:integer;
    s,b:string;   a:array[1..10] of integer;
begin
    s:=edit1.Text;   memo1.Lines.Add('исходный массив:
 '+s);
    edit1.clear; edit1.setfocus;
    i:=1; //удаление лишних пробелов между числами
    while i<=length(s)-1 do
      begin
        if (s[i]=' ') and (s[i+1]=' ') then
          begin
            delete(s,i,1);
             i:=i-1;
          end;
        i:=i+1;
      end;
    try
    i:=1;  b:=''; sum:=0; // i - номер символа в строке
    j:=1;     // j - номер элемента в массиве
    //выделение чисел из строки s и запись в массив
    while i<=length(s) do
      begin
        while (s[i]<>' ') and (i<=length(s))  do
          begin
            b:=b+s[i];
             i:=i+1;
          end;
```

```
          a[j]:=strtoint(b); sum:=sum+a[j]; i:=i+1; j:=j+1;
       b:='';
        end;
      memo1.Lines.Add('сумма элементов массива:
  '+inttostr(sum));
      except
        on EConvertError do  // обработка ошибки
    преобразования
        memo1.lines.add(' ошибка в записи числа: '+b);
      end;
end;
```

Рис. 5.5. Вид окна программы после выполнения



Процедура обработки события OnClick считывает введенные данные с компонента Edit1 и записывает в строку s и выводит на компонент Memo1. Выделяются элементы массива из строки s, для этого организуются два вложенных цикла: внешний и внутренний. Внешний цикл выполняется, пока номер обрабатываемого символа меньше длины строки s. Внутренний цикл выполняется пока i-ый символ не пробел и пока не обработаны все символы строки. Он выделяет числа из строки s и записывает в массив, вычисляет сумму элементов массива.

При преобразовании введенных данных из строчного типа в числовой возможно возникновение ошибочной ситуации. Такая ситуация обрабатывается с помощью обработчика исключений *EConvertError*. При возникновении исключения на экран выводится сообщение об ошибке и процедура обработки события OnClick завершает работу.

5.3. Ввод одномерного массива с помощью таблицы StringGrid

Пример. Дан числовой массив A(n). Найти сумму положительных, произведение отрицательных элементов массива. Массив ввести с помощью таблицы StringGrid.

Для ввода массива удобно использовать компонент StringGrid. ***Компонент StringGrid*** (стр. Additional) представляет собой таблицу, ячейки которой содержат строки символов. Таблица делится на две части – фиксированную и рабочую. Фиксированная часть служит для показа заголовков столбцов и строк. Обычно фиксированная часть таблицы занимает крайний левый столбец и верхнюю строку таблицы. Однако с помощью свойств *FixedCols* и *FixedRows* можно задать другое количество фиксированных столбцов и строк (если эти свойства имеют значение 0, таблица не содержит фиксированной части). Рабочая часть – эта остальная часть таблицы. Она может содержать произвольное количество столбцов и строк.

Центральным свойством компонента является *Cells* – двумерный массив ячеек, каждая из которых может содержать произвольный текст. Конкретная ячейка определяется номером столбца и номером строки, на пересечении которых она находится (нумерация начинается с нуля).

Свойства компонента *StringGrid:*

Name - имя компонента;

ColCount - количество столбцов таблицы;

RowCount - количество строк таблицы;

Cells - соответствующий таблице двумерный массив;

FixedCols - количество зафиксированных слева столбцов таблицы;

FixedRows - количество зафиксированных сверху строк таблицы

Options.goEditing - признак допустимости редактирования содержимого ячеек таблицы. True – редактирование разрешено, False –запрещено;

Options.goTab - разрешает (True) или запрещает (False) использование клавиши <Tab> для перемещения курсора в следующую ячейку таблицы;

Options.goAlwaysShowEdit - признак нахождения компонента в режиме редактирования. Если значение свойства *false*, то для того чтобы в ячейке появился курсор, надо начинать набирать текст, нажать клавишу <F2> или сделать щелчок мышью.

Для таблицы StringGrid установить следующие свойства:

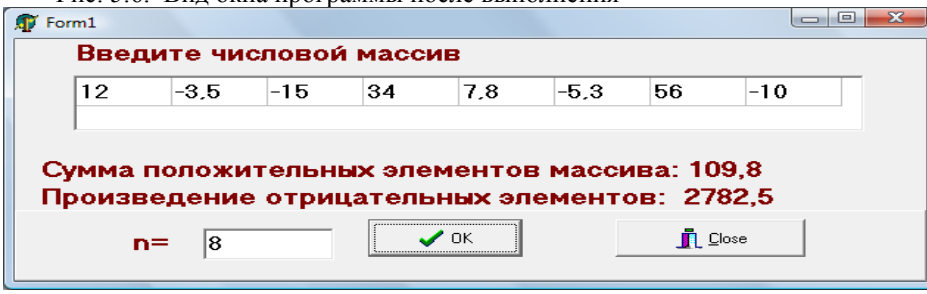
FixedCols – 0;	Options.goEditing – true;
FixedRows – 0;	Options.goTab – true;
RowCount – 1;	Options.goAlwaysShowEdit – true;

Текст процедуры обработки события OnClick для кнопки «OK”

```
procedure TForm1.BitBtn1Click(Sender: TObject);
var  i,j,n:integer;  sum,p:real;
     s,b:string;
     a:array[1..20] of real;
begin
     try
```

```
    n:=strtoint(edit1.Text);
    StringGrid1.ColCount:=n;
    for i:=1 to n do if  StringGrid1.cells[i-1,0]<>'' then
    a[i]:= strtofloat(StringGrid1.cells[i-1,0])
    else a[i]:=0;
    sum:=0; p:=1;
    for i:=1   to n do  if a[i]>=0 then sum:=sum+a[i] else
 p:=p*a[i];
    Label3.Caption:=' Сумма положительных элементов
 массива: '+floattostr(sum) + #13  +'   Произведение
 отрицательных элементов:  '+floattostr(p);
    except
    on EConvertError do
    Label3.caption:=' Ошибка в записи числа
 '+StringGrid1.cells[i-1,0];
    end;
end;
```

Рис. 5.6. Вид окна программы после выполнения



Задания для самостоятельного выполнения (предусмотреть обработку исключений).

1. Дан числовой массив. Найти сумму нечетных элементов массива, отрицательные элементы заменить их квадратами. Элементы массива ввести с нулевой строки компонента Memo .

2. Дан одномерный числовой массив A. Элементы массива ввести с нулевой строки компонента Memo. Сформировать новый массив B, удалив минимальный и максимальный элементы массива A.

3. Дан целочисленный одномерный массив. Элементы массива ввести с помощью компонента Memo по одному элементу на каждой строке, начиная с нулевой. Найти сумму элементов, стоящих перед максимальным.

4. Дан одномерный массив A. Элементы массива ввести с помощью компонента Memo по одному элементу на каждой строке, начиная с нулевой. Сформировать новый массив B, переместив каждый элемент массива A на 6 позиций влево. (Примечание, при сдвиге элементы, стоящие на первой позиции переносятся на последнюю позицию).

5. Дан одномерный массив A из слов. Массив ввести с помощью компонента Memo. Найти все слова, имеющие заданное окончание.

6. Дан одномерный массив A из слов. Массив ввести с помощью компонента Memo. Найти все слова, начинающиеся с заданной приставки.

7. Дан одномерный массив A из слов. Массив ввести с помощью компонента Memo. Найти все слова, которые начинаются и заканчиваются одной и той же буквой.

8. Дан числовой массив. Найти сумму квадратов отрицательных элементов массива. Элементы массива ввести с помощью компонента Edit.

9. Дан целочисленный одномерный массив. Элементы массива ввести с помощью компонента Edit. Найти минимальный элемент среди элементов стоящих на четных позициях массива.

10. Дан массив A(n). Массив ввести с помощью таблицы StringGrid. Вычислить среднее геометрическое первых пяти положительных элементов массива.

11. Дан целочисленный одномерный массив A(n). Массив ввести с помощью таблицы StringGrid. Элементы, кратные 7, увеличить на значение их индекса.

6. Различные способы ввода и обработки двумерного массива

6.1. Ввод двумерного массива с помощью редактора Memo

Пример. Дан двумерный массив A(n,m). Найти число отрицательных элементов массива. Массив ввести с помощью многострочного редактора Memo с нулевой и первой строк. Исходный массив и результат вывести на Memo.

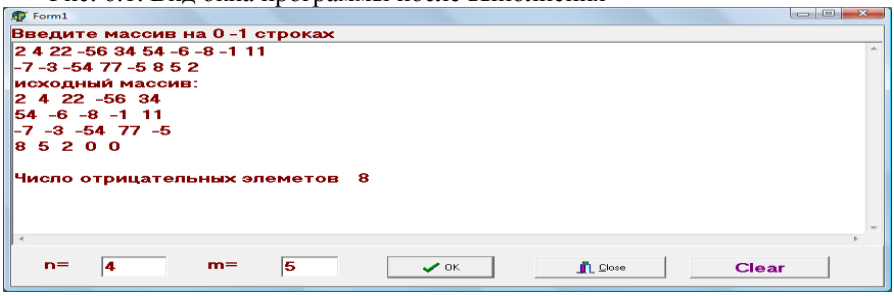
Текст процедуры обработки события OnClick для кнопки «OK”

```
procedure TForm1.BitBtn1Click(Sender: TObject);
var
    i,j,n,m,k:integer;
    s,b:string;
    a:array[1..10,1..10] of integer;
begin
    memo1.SetFocus;
    if  (memo1.Lines[0]='') or (edit1.text='') or
 (edit2.text='') then
      begin
        memo1.Lines.Add('данные не введены');  exit
      end;
    s:=''; for i:=1 to 2 do
    s:=s+memo1.Lines[i-1]+' '; // в строку  s записываем
 //  элементы массива
```

```
   n:=strtoint(edit1.text);
   m:=strtoint(edit2.text);
   // инициализируем массив
   for i:=1 to n do
   for j:=1 to m do   a[i,j]:=0;
   // из строки s удаляем лишние пробелы
   while i<=length(s)-1 do
     begin
       if (s[i]=' ') and (s[i+1]=' ') then
         begin
           delete(s,i,1); i:=i-1;
         end;
       i:=i+1;
     end;
   try
   // i - номер строки, j  - номер столбца массива
   // k - номер символа в строке s
   i:=1;  j:=1; k:=1;  b:='';
   // из строки s выделяем числа и записываем в массив
   while (k<=length(s)) and (i*j-1<n*m) do
     begin
       while (s[k]<>' ') and (k<=length(s))  do
         begin
           b:=b+s[k]; k:=k+1;
         end;
         a[i,j]:=strtoint(b);  k:=k+1;
         if j<m then j:=j+1 else
           begin
             j:=1;  i:=i+1;
           end;
       b:='';
     end;
   memo1.Lines.Add('исходный массив: ');
   // исходный массив выводим на экран
   for i:=1 to n do
     begin
       s:='';
       for j:=1 to m do
       s:=s+inttostr(a[i,j])+'  ';
       memo1.Lines.add(s);
     end;
   // определяем число отрицательных элементов
   k1:=0;
   for i:=1 to n do
   for j:=1 to m do
   if a[i,j]<0 then k1:=k1+1 ;
   memo1.Lines.add('     ');
   memo1.Lines.add('Число отрицательных элементов
'+inttostr(k1));
```

```
      except
        on EConvertError do
        memo1.lines.add('ошибка в записи числа: '+b);
      end;
end;
```

Рис. 6.1. Вид окна программы после выполнения



Процедура обработки события OnClick проверяет, введены ли исходные данные, если нет, выводит об этом сообщение и осуществляется выход из процедуры. Если да, введенные данные с нулевой и первой строк компонента Memo записываются в строку s. Число строк и столбцов массива считываются с помощью компонентов Edit1 и Edit2. После этого инициализируется массив, т.е. всем элемента массива присваиваются нулевые значения. Выделяются элементы массива из строки s, для этого организуются два вложенных цикла: внешний и внутренний. Внешний цикл выполняется, пока номер обрабатываемого символа меньше длины строки s и не выделено требуемое количество элементов (n*m). Внутренний цикл выполняется пока k-ый символ не пробел и пока не обработаны все символы строки. Он выделяет числа из строки s и записывает в массив. Если количество введенных чисел окажется меньше чем n*m, недостающие элементы окажутся равными нулю. Исходный массив выводится на экран в виде матрицы, вычисляется число отрицательных элементов массива.

При преобразовании введенных данных из строчного типа в числовой возможно возникновение ошибочной ситуации. Данная ситуация обрабатывается с помощью обработчика исключений *EConvertError*. При возникновении исключения на экран выводится сообщение об ошибке и процедура обработки события OnClick завершает работу.

6.2. Ввод двумерного массива с помощью таблицы StringGrid

Пример. В таблице записаны результаты работы студентов за семестр: количество баллов за самостоятельные работы, за контрольные работы №1 и №2 и за тестирование (таб.6.1). Эти данные ввести в ячейки компонента

StringGrid, оттуда переписать в массив A(n,7) , где n –число строк массива, число столбцов равно 7. Подсчитать общее количество баллов, набранных студентами за семестр, и определить допущен ли студент к зачету. Студент допускается к зачету, если набрал не менее 33 баллов. Результаты вычислений записать в соответствующие столбцы StringGrid.

Таб. 6.1.

Фамилия	Самост. работы	КР №1	КР №2	Результат тестир.	Общ.кол. баллов	Допуск к зачету
Алексеева	8	7	6	9		
Гарипова	7	10	8	8		
Гунина	9	10	10	10		
Крылова	8	7	4	7		
Ткаченко	6	10	9	8		
Шубина	10	12	12	10		

Для ввода исходных данных и отображения результатов используется компонент StringGrid. Для компонента StringGrid установить следующие свойства:

FixedCols – 0; Options.goEditing – true;
FixedRows – 1; Options.goTabs – true;
Options.goAlwaysShowEditing – true;

Ячейки первой фиксированной строки используются в качестве заголовков таблиц. Во время создания формы приложения нельзя установить значения элементов массива Cells, т. к. элементы массива доступны только во время выполнения программы. Поэтому значения элементов массива Cells, соответствующих первой строке, устанавливает процедура обработки события OnActivate, которое происходит во время активизации формы приложения.

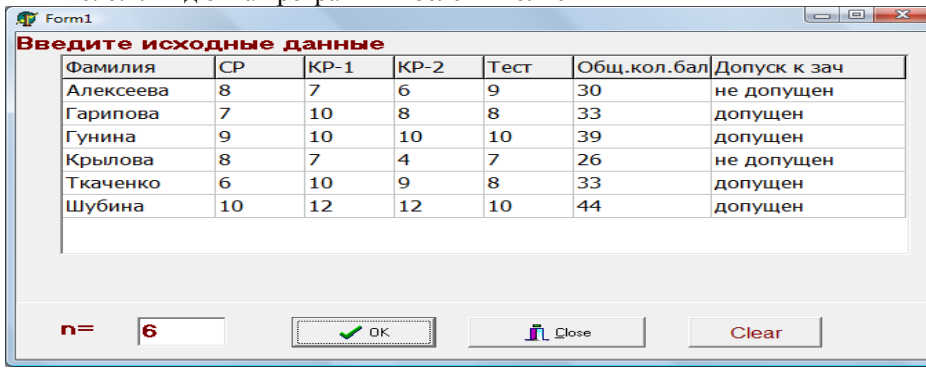
Текст модуля

```
unit masstringgrid;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics,
Controls, Forms,
  Dialogs, StdCtrls, Buttons, ComCtrls, Grids, ExtCtrls;

type
  TForm1 = class(TForm)
    Panel1: TPanel;
```

```
    Edit1: TEdit;
    BitBtn1: TBitBtn;
    BitBtn2: TBitBtn;
    StringGrid1: TStringGrid;
    Label1: TLabel;
    procedure FormActivate(Sender: TObject);
    procedure BitBtn1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation
{$R *.dfm}

procedure TForm1.FormActivate(Sender: TObject);
begin
    StringGrid1.cells[0,0]:='Фамилия';
    StringGrid1.cells[1,0]:='СР';
    StringGrid1.cells[2,0]:='КР-1';
    StringGrid1.cells[3,0]:='КР-2';
    StringGrid1.cells[4,0]:='Тест';
    StringGrid1.cells[5,0]:='Общ.кол.бал.';
    StringGrid1.cells[6,0]:='Допуск к зач'
end;

procedure TForm1.BitBtn1Click(Sender: TObject);
    var
    i,j,n,s,k1:integer; b: string;
    a:array[0..20,0..7] of string;
begin
    try
    b:= edit1.Text;
    n:=strtoint(edit1.Text);
    // кол-во строк  на 1 больше, т.к. есть фиксированная
 строка
    StringGrid1.RowCount:=n+1;
    StringGrid1.colCount:=7;
    // запись данных из   StringGrid в массив
    for i:=0 to n do
    for j:=0 to 4 do
    a[i,j]:= StringGrid1.cells[j,i];
    // вычисляем общее кол-во баллов
    // выставляем признак допуска к зачету
    for i:=1 to n do
```

```
      begin
        s:=0;
        for j:=1 to 4 do
          begin
            b:=a[i,j]; // запоминаем элемент массива для
          обработки ошибки преобразования
            s:=s+strtoint(a[i,j]);
          end;
        a[i,5]:=inttostr(s);
        if   inttostr(s)>='33' then a[i,6]:='допущен'
                               else a[i,6]:='не допущен';
      end;
    { запись общего кол-ва баллов и признака допуска к
 зачету в StringGrid }
    for i:=1 to n do
    for j:=5 to 6 do
    StringGrid1.cells[j,i]:=a[i,j];
    except
      on EConvertError do
      Label3.caption:=' Ошибка в записи числа'+b;
    end;
end;

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

procedure TForm1.BitBtn3Click(Sender: TObject);
begin
    Label3.caption:='';
    edit1.Text:='';
end;
```

Рис. 6.2. Вид окна программы после выполнения



Фамилия	СР	КР-1	КР-2	Тест	Общ.кол.бал	Допуск к зач
Алексеева	8	7	6	9	30	не допущен
Гарипова	7	10	8	8	33	допущен
Гунина	9	10	10	10	39	допущен
Крылова	8	7	4	7	26	не допущен
Ткаченко	6	10	9	8	33	допущен
Шубина	10	12	12	10	44	допущен

Сначала программа считывает в массив *a* данные, которые введены в ячейки компонента StringGrid. Затем обрабатывает массив, т.е. вычисляет общее количество баллов для каждого студента и определяет, допущен ли студент к зачету. Полученные значения записываются в последующие столбцы массива *a*. Эти данные также записываются в ячейки компонента StringGrid.

7. Динамические массивы

Если заранее неизвестно, сколько будет введено данных в массив, или объем данных собираемых для массива значительно меняется, то в подобных ситуациях можно создать динамический массив. При объявлении таких массивов границы индексов не указываются:

```
var
  a: array of integer;
  b: array of array of real;
```

В этом примере динамический массив *a* имеет одно измерение, *b* – два. Указание границ индексов по каждому измерению осуществляется во время выполнения программы с помощью функции *SetLength.* Например, в ходе выполнения оператора *SetLength(a,3);* одномерный динамический массив *a* получит память, достаточную для размещения трех целочисленных значений. Нижняя граница индексов по любому измерению динамического массива всегда равна 0, поэтому верхней границей индексов для *a* станет 2.

Фактически идентификатор динамического массива ссылается на указатель, содержащий адрес первого байта памяти, выделенной для размещения массива. Поэтому для освобождения этой памяти достаточно присвоить идентификатору массива значение ***NIL*** (другим способом является использование процедуры *Finalize*):

```
var
    a, b : array of integer;
begin
    //  Распределяем память
    SetLength(a,10);
    SetLength(b,20);
    // Используем массивы
    . . . . .
    //  Освобождаем память
    a:= NIL;
    Finalize (b) ;
end;
```

При изменении границы индексов уже существующего динамического массива сначала резервируется нужная для размещения нового массива память, затем элементы старого массива переносятся в новый.

В многомерных массивах сначала устанавливается длина его первого измерения, затем второго и т.д. Например, у двумерного динамического массива строки тоже являются динамическими массивами, и длины строк тоже должны быть установлены с помощью функции Setlength.

```
var
  b : array of array of  real; // двумерный динамический
массив
begin
  // устанавливаем количество строк массива
  setlength(b,3);
  // задаем длину каждой строки
  Setlength (b(0), 3);
  Setlength (b(1), 3);
  Setlength (b(2),3);
  . . . . .
end;
```

В отличие от обычных массивов динамические массивы могут иметь *разную* длину по второму и следующим измерениям. В предыдущем примере определен квадратный массив 3x3. Можно было, например, создать треугольный массив:

```
  setlength(b,3);
  // задаем длину каждой строки
  Setlength (b(0), 3);
  Setlength (b(1), 4);
  Setlength (b(2), 5);
```

Если заранее неизвестно, сколько будет введено данных в двумерный массив, например, количество строк и столбцов массива вводятся во время выполнения программы, границы изменения индексов массива можно задать следующим образом:

```
Var
  // объявляем двумерный динамический массив
  a : array of array of integer;
begin
  n:=strtoint(edit1.Text); // вводим  количество строк
  m:=strtoint(edit2.Text); //  вводим  количество столбцов
  //  устанавливаем количество строк массива
  setlength(a,n);
  //  задаем количество элементов каждой строки
  for i:=0 to n-1 do
  setlength(a[i],m);
  . . . . .
end;
```

Пример 1. Дан числовой массив A(n). Число элементов массива ввести с помощью компонента Edit. Элементы массива ввести с помощью таблицы StringGrid и отобразить на компоненте ListBox. В зависимости от выбранных

независимых переключателей выполнить одну или несколько из следующих действий: вычислить сумму, произведение и среднее арифметическое выбранных элементов массива. Для этого на форме установить три независимых переключателя CheckBox. Длину массива и количество столбцов StringGrid увеличить на количество выбранных переключателей. Результаты вычислений записать в добавленные элементы массива и отобразить на компоненте StringGrid.

Компонент CheckBox - независимый переключатель, используется для того чтобы пользователь мог указать свое решение типа *Да/Нет* или *Да/Нет/Не знаю* (в последнем случае в окошке компонента устанавливается флаг выбора, но само окошко закрашивается в серый цвет). Это решение отражается в свойстве *State* компонента, доступном как для чтения, так и для записи. В составе диалогового окна может быть несколько компонентов CheckBox. Состояние любого из них не зависит от состояния остальных, поэтому такие переключатели называются независимыми.

Свойства компонента:

AllowGrayed (тип Boolean) – разрешает / запрещает использование в переключателе третьего состояния *cbGrayed* (Не знаю);

Caption – содержит связанный с компонентом текст – надпись переключателя;

Checked – содержит выбор пользователя типа *Да/Нет*. Если *Checked* равно *true*, то переключатель выбран, т.е. *State = cbChecked.* Если *Checked* равно *fulse*, то *State* равно *cbUnChecked* или *cbGrayed.* Установка *Checked* в *true* во время выполнения программы автоматически переключает *State* в *cbChecked.*

State – содержит состояния компонента: *cbUnChecked* – нет, *cbChecked* – да, *cbGrayed* – не знаю. Все три состояния допускаются, если значение *AllowGrayed* равно true.

Для компонента *CheckBox1* в свойство *Caption* установить «Сумма»; для компонента *CheckBox2* в свойство *Caption* установить «Произведение»; для компонента *CheckBox3* в свойство *Caption* установить «Среднее арифметическое».

Для компонента *StringGrid* установить следующие свойства:

1) в нашем примере фиксированная часть *StringGrid* не используется, поэтому в свойства FixedCols и FixedRows установить 0;

2) в свойства Options.goEditing, Options.goTab, Options.goAlwaysShowEdit установить true

Текст модуля

```
unit dinammas1;

interface

uses
```

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

  type
    TForm1 = class(TForm)
      Panel1: TPanel;
      BitBtn1: TBitBtn;
      BitBtn2: TBitBtn;
      Label1: TLabel;
      StringGrid1: TStringGrid;
      Edit1: TEdit;
      Label2: TLabel;
      Label3: TLabel;
      ListBox1: TListBox;
      BitBtn3: TBitBtn;
      BitBtn4: TBitBtn;
      Label4: TLabel;
      Label5: TLabel;
      CheckBox1: TCheckBox;
      CheckBox2: TCheckBox;
      CheckBox3: TCheckBox;
      procedure BitBtn1Click(Sender: TObject);
      procedure BitBtn2Click(Sender: TObject);
      procedure BitBtn4Click(Sender: TObject);
      procedure BitBtn3Click(Sender: TObject);
    private
      { Private declarations }
    public
      { Public declarations }
    end;

  var
    Form1: TForm1;

  implementation
      var
      i,j,n:integer;  sum,p:real;
      s,b:string;
      a:array of real;
  {$R *.dfm}

  // Процедура обработки события нажатия на кнопку «ОК»

  procedure TForm1.BitBtn1Click(Sender: TObject);
  begin
      try
      n:=strtoint(edit1.Text);
      SetLength(a,n); // устанавливаем длину массива
```

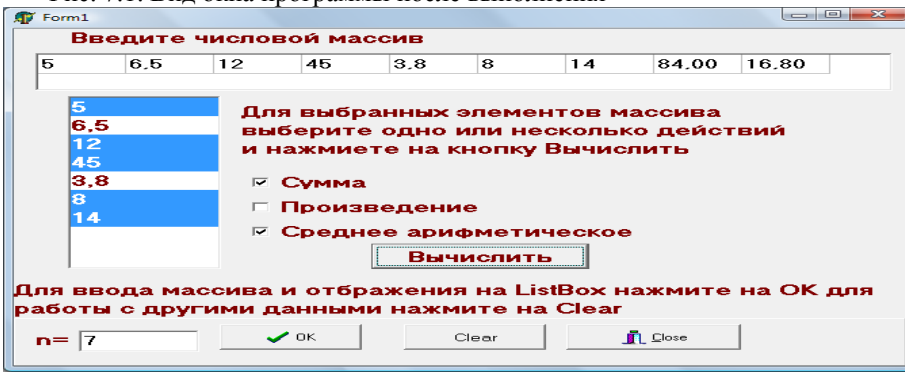
```
        StringGrid1.ColCount:=n;
        StringGrid1.RowCount:=1;
        // запись данных из ячеек таблицы StringGrid в массив
        for i:=0 to n-1 do if StringGrid1.cells[i,0]<>'' then
        a[i]:= strtofloat(StringGrid1.cells[i,0])
     else
        a[i]:=0;
        // Отображение массива на компоненте Listbox
        for i:=0 to n-1 do
     listbox1.Items.Add(floattostr(a[i]));
        except
          on EConvertError  do // ошибка преобразования строки
       в число
          Label5.caption:=' Ошибка в записи
       числа'+StringGrid1.cells[i,0];
        end;
    end;

    {Процедура обработки события OnClick для кнопки
«Вычислить»}

    procedure TForm1.BitBtn4Click(Sender: TObject);
        var sum, p,sr: real;
        j, k: integer;
    begin
        Label5.Caption:='';
        k:=0;
        if listbox1.SelCount=0 then
          begin
            Label5.caption:='B    ListBox    нет    выбранных
         элементов';   exit;
          end;
        // определяем количество выбранных переключателей
        if CheckBox1.Checked then k:=k+1;
        if CheckBox2.Checked then k:=k+1;
        if CheckBox3.Checked then k:=k+1;
        {длину массива увеличиваем на количество выбранных
     переключателей}
        SetLength(a,n+k);
        { на количество выбранных переключателей увеличиваем
     кол-во столбцов StringGrid }
        StringGrid1.ColCount:=n+k;
        j:=n; // j – индекс для записи добавленных элементов
        {Если выбран первый переключатель, вычисляем сумму
     выбранных элементов массива }
        if CheckBox1.Checked then
          begin
            sum:=0;
            for i:=0 to n-1 do
```

```
        if listbox1.selected[i]=true  then
        sum:=sum+strtofloat(listbox1.Items[i]);
        a[j]:= sum;  j:=j+1;
      end;
    {Если выбран второй переключатель, вычисляем
 произведение }
    if CheckBox2.Checked then
      begin
        p:=1;
        for i:=0 to n-1 do
        if listbox1.selected[i]=true  then
     p:=p*strtofloat(listbox1.Items[i]);
        a[j]:=p;        j:=j+1;
      end;
    {Если выбран третий переключатель, вычисляем среднее
 арифметическое }
    if CheckBox3.Checked then
      begin
        sum:=0;
        for i:=0 to n-1 do
        if listbox1.selected[i]=true  then
        sum:=sum+strtofloat(listbox1.Items[i]);
        sr:=sum/listbox1.SelCount;
        a[j]:=sr;  j:=j+1;
      end;
    //  вывод добавленных элементов массива  в StringGrid
    for i:=n  to j-1 do
 StringGrid1.cells[i,0]:=floattostrf(a[i],ffFixed,5,2);
    a:=Nil; //  освобождаем память
end;
procedure TForm1.BitBtn2Click(Sender: TObject);
begin
    close;
end;

procedure TForm1.BitBtn3Click(Sender: TObject);
begin
    Edit1.Clear;
    ListBox1.Clear;
    for i:=0 to  StringGrid1.ColCount-1 do
    StringGrid1.cells[i,0]:='';
    CheckBox1.State:=cbUnChecked;
    CheckBox2.State:=cbUnChecked;
    CheckBox3.State:=cbUnChecked;
    Label5.Caption:='';
end;
end.
```

Процедура обработки события OnClick для кнопки «ОК» считывает с компонента Edit количество элементов массива, устанавливает длину массива и количество столбцов StringGrid, данные введенные в ячейки компонента StringGrid записывает в массив и отображает на компоненте ListBox. Предусмотрена обработка исключения. При записи введенных данных в массив (при выполнении функции StrToFloat) возможно исключение EConvertError, если в ячейки StringGrid будут введены данные, которые не могут быть приведены к требуемому виду. При возникновении исключения на экран выводится сообщение об ошибке и процедура обработки события OnClick завершает работу.

Процедура обработки события OnClick для кнопки «Вычислить» проверяет, есть ли выбранные элементы на компоненте ListBox, если нет, выводит об этом сообщение и осуществляется выход из процедуры. Если есть, определяется количество выбранных переключателей, длина массива и количество столбцов StringGrid увеличивается на количество выбранных переключателей. Если выбран первый переключатель, вычисляется сумма выбранных элементов, если второй – вычисляется произведение, если третий – среднее арифметическое. Результаты вычислений записываются в добавленные элементы массива и записываются в добавленные столбцы StringGrid.

Рис. 7.1. Вид окна программы после выполнения



Пример 2. Дан двумерный целочисленный массив A размера n*m. Массив ввести с помощью таблицы StringGrid. Число строк и столбцов массива увеличить на единицу. Вычислить сумму элементов каждой строки и записать в добавленный столбец. Сумму элементов каждого столбца записать в добавленную строку. Результаты вычислений записать в соответствующие строку и столбец StringGrid.

Для StringGrid установить следующие свойства:

FixedCols – 0;
FixedRows – 0;

Options.goEditing – true;
Options.goTabs – true;
Options.goAlwaysShowEditing – true;

Текст процедуры обработки события OnClick для кнопки «OK”

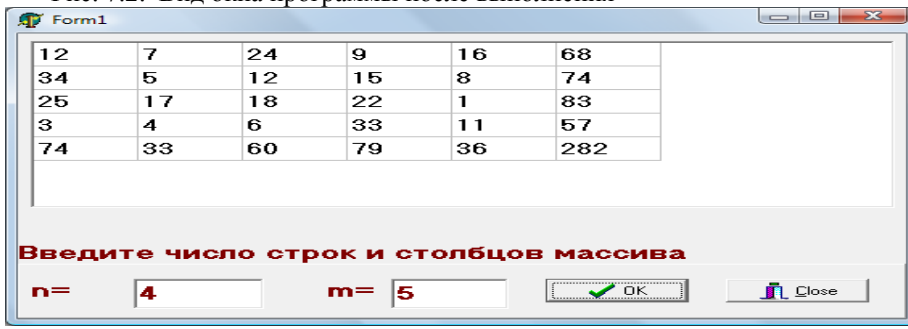
```
procedure TForm1.BitBtn2Click(Sender: TObject);
    var i,j,n,m,s, s1: integer;
    // объявляем динамический массив
    a:array of array of integer;
begin
    try
    n:=strtoint(edit1.Text);
    m:=strtoint(edit2.Text);
    //  устанавливаем количество строк
    setlength(a,n);
    //  задаем количество элементов каждой строки
    for i:=0 to n-1 do
    setlength(a[i],m);
    StringGrid1.RowCount:=n+1;
    StringGrid1.colCount:=m+1;
    // записываем данные из   StringGrid в массив
    for i:=0 to n-1 do
    for j:=0 to m-1 do
    a[i,j]:= strtoint(StringGrid1.cells[j,i]);
    //  Число строк и столбцов массива увеличиваем на
 единицу.
    setlength(a,n+1);
    for i:=0 to n do
    setlength(a[i],m+1);
    // вычисляем сумму элементов каждой строки
    for i:=0 to n-1 do
      begin
        s:=0;
        for j:=0 to m-1 do
        s:=s+a[i,j];
        a[i,m]:=s;
      end;
    // вычисляем сумму элементов каждого столбца
    for j:=0 to m do
      begin
        s1:=0;
        for i:=0 to n do
        s1:=s1+a[i,j];
        a[n,j]:=s1;
      end;
    // результаты вычислений записываем в StringGrid
```

```
      for i:=0 to n do
      StringGrid1.cells[m,i]:=inttostr(a[i,m]);
      for j:=0 to m do
      StringGrid1.cells[j,n]:=inttostr(a[n,j]);
      except
        On EConvertError do
        Label4.caption:='Ошибка в записи числа'+
    StringGrid1.cells[j,i];
      end;
      a:=Nil; // освобождаем память
end;
```

В процедуре обработки события OnClick для кнопки «OK» объявляется динамический массив. С компонентов Edit1 и Edit2 считываются количество строк и столбцов массива. С помощью функции SetLength устанавливаются количество строк и количество элементов каждой строки массива, устанавливаются количество строк и столбцов StringGrid, данные введенные в ячейки компонента StringGrid записываются в массив. Количество строк и столбцов массива увеличивается на единицу. Вычисляется сумма элементов каждой строки и записывается в добавленный столбец массива. Сумма элементов каждого столбца записывается в добавленную строку. Результаты вычислений записываются в компонент StringGrid.

Предусмотрена обработка исключения. При записи введенных данных в массив (при выполнении функции StrToInt) возможно исключение *EConvertError*. При возникновении исключения на экран выводится сообщение об ошибке и управление получает оператор стоящий после защищенного блока.

Рис. 7.2. Вид окна программы после выполнения



Задания для самостоятельного выполнения

1. В таблице записаны фамилии студентов и экзаменационные оценки за первый семестр. Эти данные ввести в ячейки таблицы StringGrid, оттуда переписать в массив. Подсчитать среднюю оценку, и студентам, имеющим

средний балл не ниже четырех начислить стипендию. Среднюю оценку и начисленную стипендию записать в последующие столбцы массива и таблицы StringGrid.

2. В таблице записаны фамилии студентов и средняя зарплата отца, матери и число членов семьи. Эти данные ввести в ячейки таблицы StringGrid, оттуда переписать в массив. Подсчитать доход на одного члена семьи и начислить социальную стипендию. Социальная стипендия начисляется, если доходы семьи студента, ниже прожиточного минимума. Доход на одного члена семьи и начисленную стипендию записать в последующие столбцы массива и таблицы StringGrid.

3. В таблице записаны фамилии абитуриентов и баллы по ЕГЭ по математике, информатике и иностранному языку. Эти данные ввести в ячейки таблицы StringGrid, оттуда переписать в массив. Вычислить общую сумму баллов. Общую сумму баллов и отметку о зачислении абитуриента в ВУЗ записать в последующие столбцы массива и StringGrid. Если сумма баллов больше или равна проходному и баллы по информатике не ниже 75, абитуриент зачисляется.

4. В таблице записаны фамилии, возраст и рост учеников. Эти данные ввести в ячейки таблицы StringGrid, оттуда переписать в массив. Отметку о записи ученика в баскетбольную секцию записать в последующие столбцы массива и StringGrid. В баскетбольную секцию принимают детей с ростом не менее 160 см, возраст не должен превышать 13 лет.

5. В таблице записаны фамилии жильцов и расход электроэнергии за текущий месяц в Квт/ч. Эти данные ввести в ячейки компонента StringGrid, оттуда переписать в массив. Подсчитать плату для каждого жильца, если компания по снабжению электроэнергией взимает плату по тарифу: к1 рублей за 1 Квт/ч за первые 500 Квт/ч; к2 рублей за 1 Квт/ч, если потребление свыше 500 Квт/ч. Подсчитать сумму оплаты и записать в последующий столбец массива и StringGrid.

6. В таблице записаны фамилии студентов и экзаменационные оценки за превый семестр. Эти данные ввести в ячейки компонента StringGrid, оттуда переписать в массив. В зависимости от выбранных независимых переключателей выполнить одну или несколько из следующих действий: вычислить среднюю оценку и качество успеваемости по данному предмету. Для этого на форме установить два независимых переключателя CheckBox. Число строк массива и таблицы StringGrid увеличить на количество выбранных переключателей. Результаты вычислений записать в добавленные элементы массива и отобразить на компоненте StringGrid. (Для определения качества успеваемости число студентов, сдавших все экзамены на хорошо и отлично, разделить на общее количество студентов и умножить на 100%).

8. Работа с текстовыми файлами

В состав Windows входят ряд типовых диалоговых окон, таких как окно выбора загружаемого файла, окно выбора шрифта, окно настройки принтера и т.д. В Delphi реализованы классы, объекты которых дают удобные способы создания и использования таких окон.

Работа со стандартными диалоговыми окнами осуществляется в три этапа.

Вначале на форму помещается соответствующий компонент и осуществляется настройка его свойств. На втором этапе осуществляется вызов стандартного для диалогов метода Execute, который создает и показывает на экране диалоговое окно. Вызов этого метода обычно располагается внутри обработчика какого-либо события. Например, обработчик пункта меню *Открыть файл* может вызвать метод Execute диалога OpenDialog, обработчик пункта меню *Сохранить* может вызвать такой же метод у компонента SaveDialog. Только после обращения к Execute на экране появляется соответствующее диалоговое окно. Окно диалога является модальным окном, поэтому сразу после обращения к Execute дальнейшее выполнение программы приостанавливается до тех пор, пока пользователь не закроет окно. Окна бывают модальные и немодальные. Немодальное окно работает одновременно с первым, они обычно открываются в одном методе и закрываются в другом. Модальное окно полностью берет на себя управление программой, поэтому оператор следующий за обращением к методу Execute получит управление только после закрытия этого окна. Execute – логическая функция, она возвращает в программу True, если результат диалога с пользователем был успешным.

Проанализировав результат Execute, программа может выполнить третий этап - использование введенных с помощью диалогового окна данных – имени файла, настроек принтера, выбранного шрифта и т.д.

Компоненты OpenDialog и SaveDialog – диалоги открытия и сохранения файлов (страница DIALOGS)

Свойство FileName (тип String) содержит маршрут поиска и выбранный файл при успешном завершении диалога. Программа может использовать это свойство для доступа к файлу с целью читать из него данные (OpenDialog) или записывать в него (SaveDialog). Пользователь может ввести произвольное имя и, следовательно, указать несуществующий файл. Для записи это не имеет значения, но при чтении отсутствие файла может привести к аварийному завершению программы. Чтобы избежать этого, можно использовать механизм обработки исключительных ситуаций или можно проверить существование файла глобальной функцией FileExists.

Например,

```
Procedure …;
```

```
    Var  f:TextFile; S:string;
    Begin
    //
    OpenDialog1.Execute and FileExists(OpenDialog1.FileName)
then
    Begin
    AssignFile(f, OpenDialog1.FileName);
    Reset (f);
    Memo1.lines.clear;
    While not eof(f) do
    begin
        Readln(f,s);
        Memo1.lines.add(s);
    end;
    Closefile(f);
    end;
    end;
```

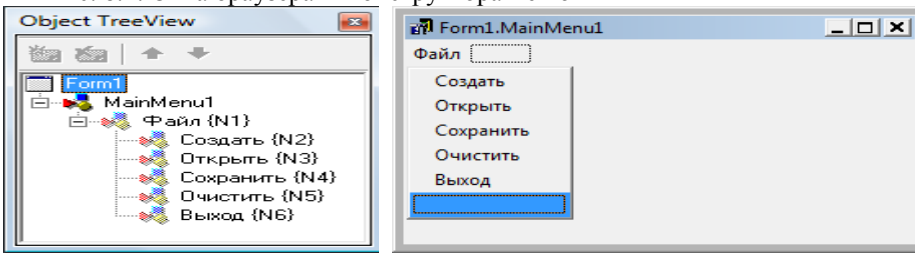
Свойство Filter (тип string) используется для фильтрации (отбора) файлов. Например, оператор

OpenDialog1.Filter:='|Текстовые файлы |*.txt|Файлы Паскаля|*.pas'; задает две маски - для отбора файлов с расширениями pas и txt.

Пример. Создать простейший текстовый редактор, который работает с меню (рис. 8.1).

На окне формы расположить компоненты MainMenu, Memo, OpenDialog, SaveDialog. Для компонента Memo установить следующие свойства: Align → alClient ; ScrollBars → ssBoth (обе полосы прокрутки); Font – изменение надписи (размер, стиль, шрифт, цвет). Для открытия и сохранения файлов использовать стандартные диалоговые окна.

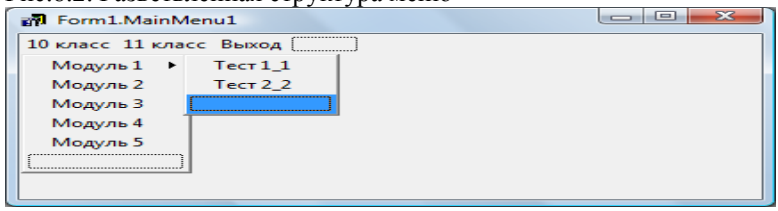
Рис. 8.1. Окна браузера и конструктора меню



Компонент MainMenu – главное меню формы. После установки компонента на форму необходимо создать его опции (пункты). Для этого следует дважды щелкнуть по компоненту левой кнопкой мыши, либо щелкнуть в правой половине строки *Items* Инспектора Объектов. Открывается окно конструктора меню (рис. 8.1).

Для создания опций перейти в окно Инспектора Объектов и ввести текст опции в строке Caption, после чего нажать Enter – опция готова, можно перейти к следующей опции. Каждая опция главного меню может раскрываться в список подпунктов. Для создания подпункта щелкнуть мышью ниже опции и ввести первый подпункт. Для создания разветвленных меню, т.е. таких, у которых подпункты вызывают новые списки, щелкнуть по подпункту и нажать *Ctrl+Вправо*, где *Вправо* – клавиша смещения курсора вправо. На рисунке 8.2 показан пример разветвленной структуры меню.

Рис.8.2. Разветвленная структура меню



Текст модуля

```
unit textfile;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics,
Controls, Forms,
  Dialogs, StdCtrls, Buttons, ComCtrls, Menus;

type
  TForm1 = class(TForm)
    Memo1: TMemo;
    MainMenu1: TMainMenu;
    OpenDialog1: TOpenDialog;
    SaveDialog1: TSaveDialog;
    N1: TMenuItem;
    N2: TMenuItem;
    N3: TMenuItem;
    N4: TMenuItem;
    N5: TMenuItem;
    N6: TMenuItem;
    procedure N1Click(Sender: TObject);
    procedure N2Click(Sender: TObject);
    procedure N3Click(Sender: TObject);
    procedure N4Click(Sender: TObject);
    procedure N5Click(Sender: TObject);
  private
    { Private declarations }
```

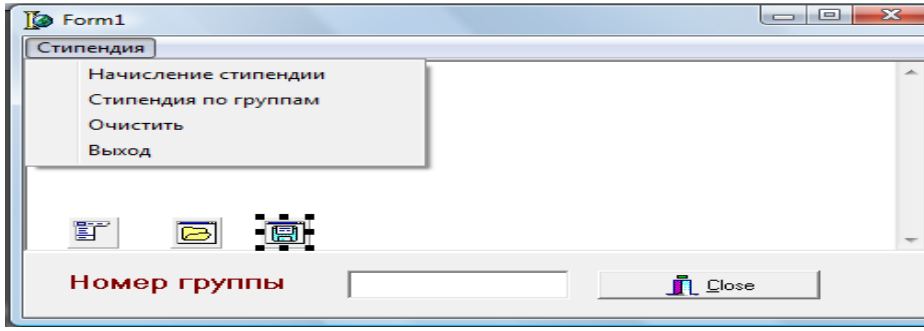
```
  public
    { Public declarations }
  end;

var
  Form1: TForm1;
implementation
    var n,i:integer;
    f:textfile;
    s:string;
{$R *.dfm}

procedure TForm1.N2Click(Sender: TObject);
begin
    //  создание файла
    memo1.Clear;
    memo1.SetFocus;
end;
procedure TForm1.N4Click(Sender: TObject);
begin    //сохранение файла
    if savedialog1.Execute then
      begin
        assignfile(f,savedialog1.filename);
        rewrite(f);
        n:=memo1.lines.Count;
        for i:=1 to n do writeln(f,memo1.lines[i-1]);
        closefile(f);
      end;
end;
procedure TForm1.N5Click(Sender: TObject);
begin   // очистка Memo
    memo1.Clear;
    memo1.SetFocus;
end;
procedure TForm1.N6Click(Sender: TObject);
begin  // выход
    close
end;
procedure TForm1.N3Click(Sender: TObject);
begin   //  открытие файла
    if opendialog1.Execute and
 fileexists(opendialog1.FileName) then
      begin
        assignfile(f,opendialog1.FileName);
        reset(f);
        memo1.Clear;
        while not eof(f) do
          begin
            readln(f,s);
```

```
            memo1.Lines.Add(s);
          end;
        closefile(f);
      end
      else
        memo1.Lines.Add('файл не открыт')
end;
end.
```

Задание для самостоятельного выполнения.

Создать текстовый файл «Результаты сессии», на первой строке которого содержатся номер семестра, учебный год, размер обычной стипендии, размер повышенной стипендии. На каждой строке начиная со второй, содержатся следующие сведения: номер группы, фамилия студента, экзаменационные оценки за семестр (не менее четырех оценок).

Рис. 8.3.



Программа работает с меню (рис. 8.3). По команде меню «Начисление стипендии» программа начисляет стипендию студентам в зависимости от результатов сессии и создает новый текстовый файл «Стипендия», на первой строке которого содержатся номер семестра, учебный год. На каждой строке начиная со второй, содержатся следующие сведения: номер группы, фамилия студента, экзаменационные оценки за семестр, размер начисленной стипендии. Если студент все экзамены сдал на хорошо и отлично, ему начисляется обычная стипендия, если все экзамены сдал на отлично, то - повышенная. Если хотя бы один экзамен сдан на удовлетворительно, студенту стипендия не начисляется.

По команде меню «Стипендия по группам» программа выдает список студентов по запрашиваемой группе в такой форме: первая строка заголовка: «Список начисления стипендии по группе <номер группы> за I семестр 1911/12 учебного года», следующая строка заголовка: « фамилия оценка 1 оценка 2 оценка 3 оценка 4 стипендия». Далее – список студентов с указанием оценок и суммы стипендии.

9. Программы со многими формами

Форма является основным строительным блоком в Delphi. В каждом проекте Delphi предусмотрена хотя бы одна форма, связанная с модулем кода программы пользователя и появляется на экране в момент старта программы. Сложные проекты могут иметь сколько угодно форм, с каждой связан свой модуль, который решает какую-то конкретную задачу. Каждая из форм появляется на экране по мере надобности.

Разновидности форм определяются значениями их свойства ***FormStyle***, а также разнообразием форм-заготовок, хранящихся в репозитории (архиве) Delphi.

Стиль формы задается одним из значений свойства

TFormStyle = (fsNormal, fsMDIChild, fsMDIForm, fsStayOnTop);

Property FormStyle: TFormStyle;

Стиль *fsNormal* определяет обычную форму, использующуюся для решения самых различных задач, в том числе – для общего управления всей программой (главная форма).

Стили *fsMDIChild, fsMDIForm* используются при создании так называемых многодокументных приложений в стиле *MDI (Multi Document Interface).* Этот стиль предполагает создание главного окна MDI (его обычно называют рамочным), внутри которого по мере надобности появляются дочерние окна. Дочерние окна , подобно дочерним элементам контейнера, не могут выходить за пределы главного рамочного окна. Для создания рамочного окна используется стиль *fsMDIForm*, а для создания дочернего окна стиль *fsMDIChild*.

Стиль *fsStayOnTop* предназначен для окон**,** которые всегда должны располагаться над всеми другими окнами программы. В момент активизации окна оно обычно становится видимым, даже если перед этим его загораживали другие раскрытые окна. Понятно, что этот стиль используется в исключительных случаях, когда окно содержит что-то, требующее повышенного внимания пользователя.

Современные многооконные приложения чаще всего строятся в стиле *SDI (Single Document Interface),*который в противоположность MDI не накладывает ограничений на положение и размеры вспомогательных форм (в стиле SDI реализована среда Delphi). Для создания форм в этом случае используется стиль *fsNormal.*

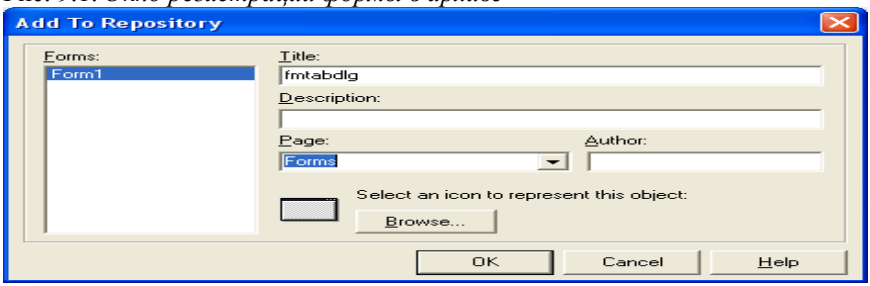
В репозитории Delphi множество стандартных форм-заготовок, предназначенных для решения конкретных задач. Доступ к репозиторию открывает опция меню ***File/New***. Помимо универсальной пустой формы Form репозиторий содержит специализированные формы, некоторые из которых приведены в таблице 9.1.

Таб. 9.1.

Название	Страница	Назначение
About box	Forms	Окно о программе
Dual List Box	Forms	Диалоговое окно с двумя компонентами ListBox. Используется для гибкого управления списками.
Tabbed Pages	Forms	Заготовка для многостраничного диалогового окна с закладками, кнопками Ok, Cancel, Help.
Password Dialog	Dialogs	Диалоговое окно с редактором TEdit, кнопками Ok, Cancel для ввода паролей.
Dialog Wizard	Dialogs	Мастер создания диалоговых окон.

Архив Delphi хранится в папке ***Objrepos.*** Чтобы созданную форму перенести в архив ее надо сохранить в этой папке (C:\Program Files\Borland\Objrepos). Сохранение формы в этой папке еще не обеспечивает включение ее в архив. Созданную форму нужно зарегистрировать в архиве. Для этого щелкнуть по форме правой кнопкой. В появившемся меню выбрать ***Add to Repository***. В открывшемся окне в строке ***Title*** набрать имя формы, в строке ***Page*** выбрать ***Forms***, нажать на кнопку «OK» (рис. 9.1). Когда форма зарегистрирована в любой момент ее можно выбрать с помощью опции ***File / New.***

Рис. 9.1. *Окно регистрации формы в архиве*



Компонент форма - FORM

Некоторые свойства формы:

Active (тип Boolean)– содержит True, если окно активно;

BorderIcons (тип TBorderIcons) - определяет наличие кнопок в заголовке окна;

BorderStyle (тип TFormBorderStyle) – определяет стиль рамки окна;

ModalResult (тип TModalResult) – для модального окна содержит результат диалога.

Некоторые методы формы:

Procedure Close – закрывает окно, для главного окна завершает работу программы;

Procedure Print – печатает окно на принтере;

Procedure SetFocus – передает фокус ввода форме;

Procedure Show – показывает форму в немодальном режиме;

Function ShowModal – показывает окно в модальном режиме и возвращает результат диалога.

Некоторые события формы:

OnActivate – возникает в момент активизации окна;

OnCreate – возникает при создании окна, но до появления его на экране;

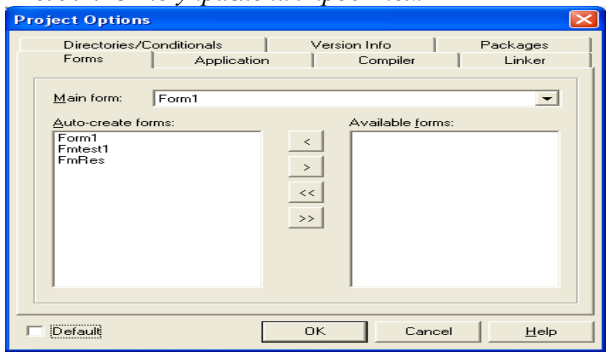
OnResize – возникает при изменении размеров окна;

OnShow – возникает при появлении окна на экране.

Создание и использование форм

Для подключения новой формы к проекту достаточно обратиться к репозиторию и выбрать нужную разновидность формы. Самая первая форма подключенная к проекту (стандартное имя формы Form1) становится главным окном программы. Окно этой формы автоматически появляется на экране в момент старта программы. Любое окно можно сделать главным. Для этого нужно обратиться к опции *Projects / Options* и раскрыв список *Mainform*, выбрать нужную форму (рис. 9.2).

Рис. 9.2. *Окно управления проектом*



Когда программа работает со многими окнами, каждое следующее окно становится видно только после обращения к его методу *Show* или *Showmodal.*

Чтобы обратиться к этим методам нужно сослаться на объект-окно, который автоматически объявляется в интерфейсном разделе связанного с окном модуля. Главное окно, в свою очередь, тоже должно знать о существовании другого окна. Это достигается ссылкой на модуль окна в предложении Uses. Например, если в ходе выполнения одного из методов главного окна программа захочет вызвать окно с именем fmtest1 (это окно связано с модулем test1), в разделе ***implementation*** главного модуля должно быть следующее предложение

implementation
Uses test1;

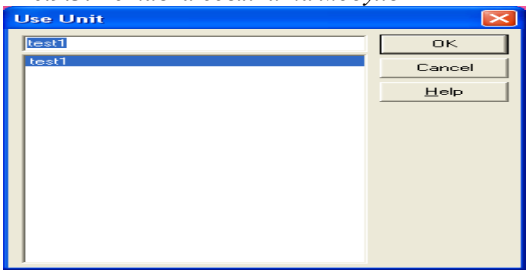
После чего окно можно вызвать на экран:

fmtest1.Showmodal;

или

fmtest1.Show;

Рис.9.3. *Вставка ссылки на модуль*



Вставку ссылку на модуль можно автоматизировать. Для этого активизировать главное окно после чего выполнить опцию *File / Use Unit* . В появившемся диалоговом окне (рис. 9.3) выбрать нужный модуль (в нашем примере test1) и нажать на OK. При этом после слова **implementation** вставляется ссылка **Uses** test1**.** Точно так же открыв второе окно, выполнить опцию *File / Use Unit* .

При вызове метода **show** второе окно появляется на экране и работает одновременно с первым. Такие окна называются немодальными, они всегда открываются в одном методе и закрываются в другом. В отличие от этого обращение к **Showmodal** создает модальное окно. Модальные окна всегда требуют от пользователя принятия какого-либо решения. С их помощью реализуется диалог с пользователем и с их помощью пользователь может сообщать о принятом решении. В момент закрытия этого окна число, соответствующее решению пользователя, помещается в **modalresult.** Некоторые стандартные кнопки (*Ok, Yes, No, Cancel* и т.п.) автоматически помещают нужное число в ModalResult и закрывают окно. В других случаях об этом должен заботиться программист. Вызывающая программа получает значение ModalResult как значение функции ShowModal и может тут же его анализировать и использовать:

fmtest1.Showmodal;
fmres.Edit1.text:= inttostr(fmtest1.modalresult);

Модальное окно должно закрываться методом **hide**. Этот метод не меняет значения modalresult. А **close** всегда в modalresult помещает число 2. Если в вызывающую программу нужно передать нестандартный модальный результат, следует, например, написать:

Modalresult:=ball;
Hide;

10. Создание многооконного проекта

Пример. Создать многооконный проект, который предлагает учащимся десятого класса тесты по информатике. Каждый тест содержит не менее пяти вопросов и не менее четырех вариантов ответов на каждый вопрос. Приложение работает с меню (рис. 10.5). Запрашивает у ученика фамилию и класс. Ученик из меню выбирает тест, на экране появляется тестовое задание с вопросами и вариантами ответов. Для создания теста использовать компонент *TabbedNoteBook* – набор страниц с закладками. На каждой странице закладки для отображения вопроса установить компонент *Label*, для задания вариантов ответов – компонент *RadioGroup* – группу зависимых переключателей. После ответа на все вопросы ученик нажимает на кнопку «ОК» на форме и на экране появляется окно с результатами тестирования. Результаты тестирования также записываются в текстовый файл. Каждое тестовое задание размещается на отдельной форме и связанный с ней модуль обрабатывает результаты.

Для создания тестовых заданий заранее создать форму (заготовку) и разместить в архиве Delphi.

Создание формы "Тестирование" и перенос ее в архив Delphi

Компонент TabbedNoteBook – набор страниц с закладками. Представляет многостраничный блокнот с закладками. Основные свойства:

ActivePage (тип string) – определяет имя активной страницы блокнота;

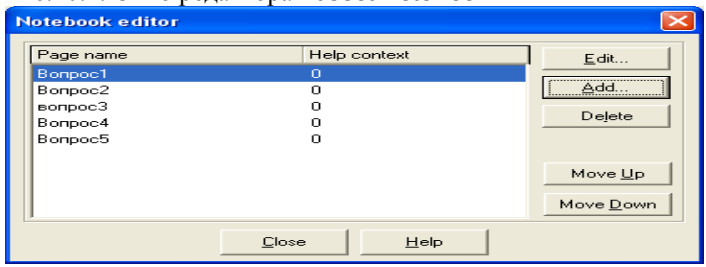
PageIndex (тип integer) - содержит номер активной страницы блокнота;

Pages (тип TStrings) – содержит набор строк с именами страниц. Pages – ключевое свойство. С помощью его методов Add, Delete, Find и т.д. можно добавлять и удалять страницы, отыскивать нужную и т.д.

Событие *OnChange* возникает при смене страницы.

На форму из страницы *WIN 3.1* вставить компонент *TabbedNoteBook*. Для добавления новых страниц открыть редактор editor (свойство ***Pages***). Нажав на кнопку ***Edit*** первую страницу переименовать на «вопрос1»**.** Нажав на кнопку ***Add*** добавить еще четыре страницы (рис. 10.1).

Рис.10.1. Окно редактора TabbedNoteBook



Используя свойство **ActivePage** активизировать страницу **«вопрос1»**. На этой странице вставить метку **Label**. Для метки установить свойства: Caption – очистить, Align – altop , Aligment – TaCenter, Height – 70, Font – изменить размер и шрифт надписи, Autozise – false, WordWrap - true. Вставить компонент ***RadioGroup*** (Caption – ‘Варианты ответов’). Все эти элементы скопировать на каждую страницу.

Компонент RadioGroup (стр. *Standard)* – группа зависимых переключателей. Представляет собой специальный контейнер, предназначенный для размещения зависимых переключателей класса *TRadioButton*. Каждый размещаемый в нем переключатель помещается в специальный список Items и доступен по индексу, что упрощает обслуживание группы. Основные свойства:

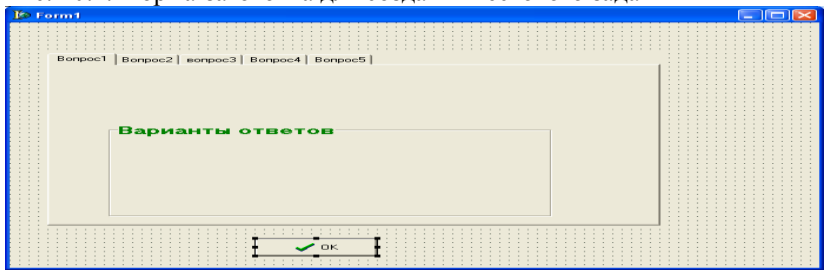
Colomns (тип integer) – определяет количество столбцов переключателей;

ItemIndex (integer) – содержит индекс выбранного переключателя, нумерация переключателей начинается с нуля, если не один из переключателей не выбран имеет значение -1;

Items (TString) – содержит список строк с заголовками элементов.

На форме разместить кнопку **‘OK**’ (рис. 10.2).

Рис. 10.2. Форма-заготовка для создания тестового задания

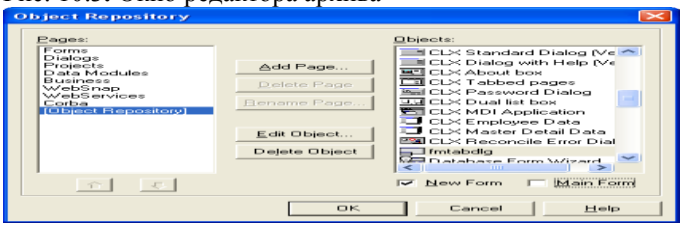


После этого созданную форму сохранить с именем **Tabdlg** в папке Objrepos (c:\program Files\Borland\Delphi6\Objrepos\tabdlg). Созданную форму зарегистрировать в архиве, щелкнуть по форме правой кнопкой. В появившемся меню выбрать ***Add to Repository***. В открывшемся окне в строке

Title набрать **fmtabdlg**, в строке Page выбрать Forms, нажать на кнопку «OK» . **Проект не сохранять!**

Чтобы при создании новой формы (при выполнении команды –***NEW FORM***) открывалась форма Tabdlg выполнить команду ***Tools – Repository***. В открывшемся окне в области Pages выделить строку *Object Repository*, в области Objects выделить свою форму и активизировать флажок ***New Form*** (рис. 10.3).

Рис. 10.3**.** Окно редактора архива



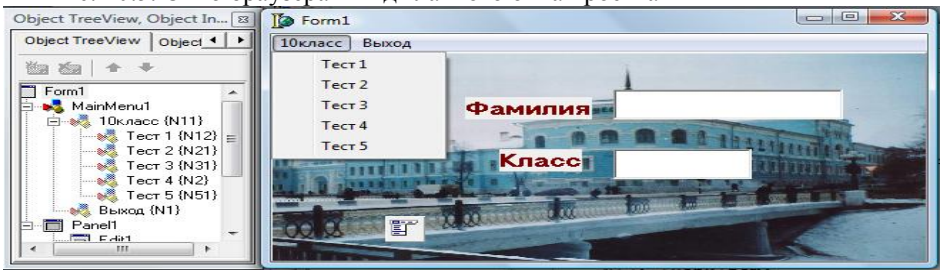
Создание главного окна и проекта программы.

Выполнить команду ***New Application***. На экране появится форма Form1. На окне формы поместить компонент ***MainMenu.*** Сформировать опции меню (рис. 10.5). На окне формы поместить Panel. Для панели установить свойства: Caption – очистить, Align – AlClient. На панели из страницы ***Additional*** поместить компонент ***Image***. Рисунок должен занимать всю панель. Для Image1 установить свойства: Align – AlClient, Stretch – true (растяжка). Если это свойство включено, то рисунок будет подгоняться под размер области просмотра. В эту область загрузить рисунок. Используя свойство ***Picture*** открыть окно загрузки рисунка .

Рис. 10.4. Окно загрузки рисунка

Запросить у ученика фамилию и класс. Для этого на панели поместить компоненты *Label* и *Edit*. Сохранить программу под именем **inform.pas**, проект под именем **Test.dpr**.

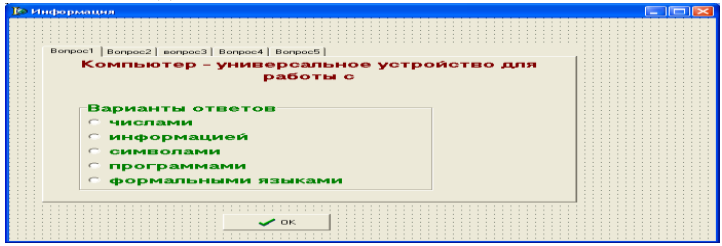
Рис. 10.5. Окно браузера и вид главного окна проекта



Создание модуля TEST1.

Открыть проект **Test.dpr**. Выполнить команду ***New Form***. На экране должна появиться форма Tabdlg. Для формы установить свойства: Name – fmtest1, Caption – ‘Информация’. Многостраничную закладку *TabbedNoteBook* заполнить вопросами и вариантами ответов.

Рис. 10.6. Вид окна fmtest1



Для того чтобы при выполнении теста, каждый раз открывалась закладка «вопрос1», для формы ***fmtest1*** обработать событие ***OnActivate***. В теле процедуры ввести:

```
TabbedNoteBook1.PageIndex:=0; или
TabbedNoteBook1.ActivePage:='вопрос1';
```

Два раза щелкнуть на кнопке «ОК», установленной на форме. Процедура обработки события OnClick для кнопки «ОК» должна подсчитать число правильных ответов. В открывшемся окне кода программы ввести:

В разделе описаний

```
var ball:integer;
```

В теле процедуры:

```
ball:=0;
If radiogroup1.itemindex=1 then ball:=ball+1;
If radiogroup2.itemindex=1 then ball:=ball+1;
If radiogroup3.itemindex=2 then ball:=ball+1;
If radiogroup4.itemindex=0 then ball:=ball+1;
If radiogroup5.itemindex=2 then ball:=ball+1;
modalresult:=ball;      //  число правильных ответов
hide;
```

Связать главную форму с формой **fmtest1,** для обеих форм выполнить команду ***File - Use unit***. Модуль сохранить под именем **Test1.pas.**

Создание модуля RES.

Создать окно для отображения результата. Выполнить команду **Tools – Repository**, отыскать свою форму **fmtabdlg** и снять флажок **New Form**. Выполнить команду New Form. На экране появится форма Form2. Для формы установить свойства: ***Name – fmres, Caption – 'Результат'.***

На окне результата поместить панель, на панели рисунок. Вывести сообщение: <число предложенных вопросов> и <число правильных ответов>. Поместить кнопку «ОК» (рис. 6.7). При нажатии на кнопку «ОК» окно должно закрыться. Для кнопки "ОК" создать процедуру обработки события OnClick, в открывшемся окне кода программы ввести: ***Hide;***

Связать главную форму с формой **fmres,** для обеих форм выполнить команду ***File - Use unit***. Модуль сохранить под именем ***Res.pas***.

Рис. 10.7. Вид окна результатов

Обработка события «Выбор пункта меню Тест 1».

Перейти на главное окно. Открыть меню, выбрать пункт **Тест 1**, щелкнуть по этому пункту. При выборе пункта меню Тест 1 на экране должна появиться форма с вопросами и вариантами ответов для теста 1. В открывшемся окне кода программы в разделе описаний ввести:

```
var b:integer;
```

В теле процедуры:

```
fmtest1.Showmodal;        // показать форму fmtest1
b:=fmtest1.modalresult;
fmres.Edit1.text:='5';
fmres.Edit2.text:= inttostr(b);
fmres.show;
```

// Запись результатов тестирования в текстовый файл

// ……..

Результаты тестирования записывать в текстовый файл. Файл открыть в режиме Append. В файл записывать: класс, фамилию, тему тестирования, число предложенных вопросов, число правильных ответов, дату и время проведения тестирования. Текстовый файл создать заранее. Тему тестирования взять из заголовка окна *fmtest1.caption*.

Функция *DateToStr(Date)* - возвращает текущую дату.

Функция *TimeToStr(Time*) – возвращает текущее время.

Сохранить внесенные изменения и запустить программу на выполнение. Чтобы каждый раз при запуске программы на выполнение, автоматически сохранялся последний вариант программы, выполнить команду *Tools – Environment Options*. Активизировать страницу *Preferences*, в области *AutoSave Options* активизировать флажки.

Если при очередной загрузке проекта, он загрузиться без подключенных к нему модулей, загрузить их, выполнив команду ***Project / Add to Project.***

Задания для самостоятельного выполнения

а). Пока ученик не введет свои данные меню сделать недоступным. В меню в пункт **класс10- Тест 3** добавить подпункты **Тест 3_1** и **Тест 3_2.**

б). Запись результатов в текстовый файл и вывод результатов на окно fmres оформить в виде отдельной процедуры. Процедуре в качестве параметра передать имя соответствующего окна тестирования. Пример заголовка процедуры:

```
Procedure zapres(fm:TForm1);
...
begin
…..
fm.showmodal;          // открывает модальное окно с
вопросами теста 1
fmres.edit2.text:=inttostr(fm.modalresult);      {  выводит
число правильны ответов на окно fmres. }
.....
end;
```

Пример вызова процедуры из процедуры обработки пункта меню Тест1: **zapres(fmtest1);**

в). Создать модули **Test3_1** и **Test3_2**, подключить их к проекту. Для вывода результатов использовать окно **fmres**. Результаты тестирования добавить в тот же текстовый файл.

г). Имя текстового файла ввести через параметр при запуске программы на выполнение. При запуске программы из среды **Delphi** выполнить команду ***Run – Parameters.*** В открывшемся диалоговом окне в поле ***Parameters*** вести имя текстового файла.

Если программа будет запускаться из среды **Windows**, создать исполняемый файл. Для этого сначала выполнить команду ***Project / Options*** открыть закладку ***Directories / Conditionals***; в поле ***Output Directory*** выбрать папку, куда будет записываться exe – файл. Для компиляции всех модулей проекта и создания исполняемого файла выполнить команду ***Build Test.*** Для исполняемого файла создать ярлык, для ярлыка открыть контекстное меню, выполнить команду ***Свойства*** и в поле ***Объект***, после имени программы, ввести имя текстового файла, заключив его в кавычки (например,

D:\test\test.exe 'D\rezult.txt'). Текст, находящийся в поле ***Объект*** вкладки ***Ярлык*** диалогового окна ***Свойства,*** называется командной строкой**.**

Рис.10.8. Окно Run Parametrs

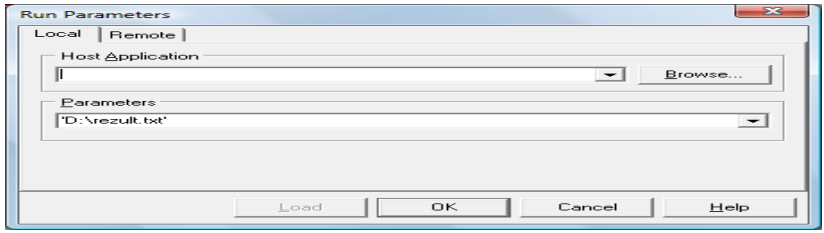
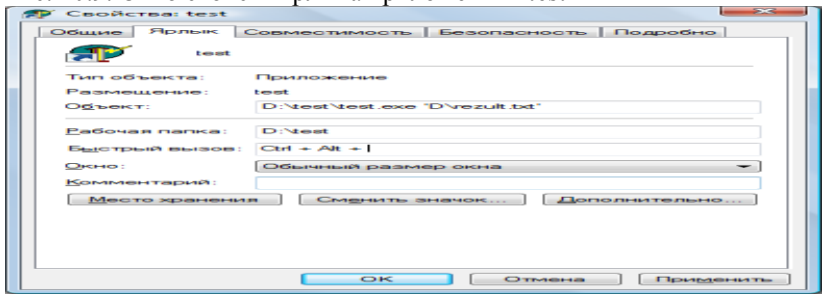


Рис. 10.9. Окно свойств ярлыка приложения «test»



Программа может получить параметр, указанный в командной строке запуска программы, как значение функции *ParamStr(n)*, где n – номер параметра. Количество параметров командной строки находится в глобальной переменной *ParamCount*. Для нашей программы значение переменной ParamCount равно 1, а функции ParamStr(1) –'D:\rezult.txt'. Пример фрагмента программы, обеспечивающего прием параметра из командной строки:

```
    If  ParamCount=0 then begin   ShowMessage('не задан
текстовый файл');        Exit; end;
    Fname:= ParamStr(1);
    . . . . .
    AssignFile(f, Fname);
    .....
```

Текст модуля inform

```
    unit inform;

    interface

    uses
      Windows, Messages, SysUtils, Variants, Classes, Graphics,
Controls, Forms,
```

```
  Dialogs, Menus, StdCtrls, jpeg, ExtCtrls, ComCtrls;

type
  TForm1 = class(TForm)
    MainMenu1: TMainMenu;
    N11: TMenuItem;
    N12: TMenuItem;
    N21: TMenuItem;
    N31: TMenuItem;
    N2: TMenuItem;
    N51: TMenuItem;
    N1: TMenuItem;
    Panel1: TPanel;
    Image1: TImage;
    Label1: TLabel;
    Edit1: TEdit;
    Label3: TLabel;
    Edit2: TEdit;
    procedure N12Click(Sender: TObject);
    procedure N1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

uses res, test1;
     var b:integer;
     f:textfile;
     s:string;

{$R *.dfm}

procedure TForm1.N12Click(Sender: TObject);
begin
     if (edit1.text='') or (edit2.text='') then  exit;
     fmtest1.Showmodal;
     b:=fmtest1.modalresult;
     fmres.label4.caption:=inttostr(b)+'   ';
     fmres.Label3.caption:= '5';
     fmres.show;
     // запись результатов тестирования в текстовый файл
     AssignFile(f,'restest.txt');
     append(f);
```

```
        s:='класс '+edit2.text+' '+edit1.text+'  тема
     '+fmtest1.caption+'  число предложенных вопросов  5 ';
        writeln(f,s);
        s:=' число правильных ответов  '+inttostr(b)+' дата
     '+datetostr(date)+' время '+timetostr(time);
        writeln(f,s);
        closefile(f);
    end;
    procedure TForm1.N1Click(Sender: TObject);
    begin
        close;
    end;
    end.

    Текст модуля test1
    unit test1;

    interface

    uses
      Windows, Messages, SysUtils, Variants, Classes, Graphics,
Controls, Forms,
      Dialogs, StdCtrls, Buttons, ExtCtrls, ComCtrls, TabNotBk;

    type
      TFmtest1 = class(TForm)
        TabbedNotebook1: TTabbedNotebook;
        RadioGroup1: TRadioGroup;
        Label1: TLabel;
        Label2: TLabel;
        Label3: TLabel;
        Label4: TLabel;
        Label5: TLabel;
        BitBtn1: TBitBtn;
        RadioGroup1: TRadioGroup;
        RadioGroup2: TRadioGroup;
         RadioGroup3: TRadioGroup;
        RadioGroup4: TRadioGroup;
        RadioGroup5: TRadioGroup;
        procedure BitBtn1Click(Sender: TObject);
        procedure FormActivate(Sender: TObject);
      private
        { Private declarations }
      public
        { Public declarations }
      end;

    var
      Fmtest1: TFmtest1;
```

```
implementation

uses inform;

{$R *.dfm}

procedure TFmtest1.BitBtn1Click(Sender: TObject);
      var ball: integer;
begin
    ball:=0;
    If radiogroup1.itemindex=1 then ball:=ball+1;
    If radiogroup2.itemindex=1 then ball:=ball+1;
    If radiogroup3.itemindex=2 then ball:=ball+1;
    If radiogroup4.itemindex=0 then ball:=ball+1;
    If radiogroup5.itemindex=2 then ball:=ball+1;
    modalresult:=ball;
    hide;
end;
procedure TFmtest1.FormActivate(Sender: TObject);
begin
    TabbedNoteBook1.PageIndex:=0;
end;
end.

Текст модуля res
unit res;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics,
Controls, Forms,
  Dialogs, StdCtrls, Buttons, jpeg, ExtCtrls;

type
  Tfmres = class(TForm)
    Panel1: TPanel;
    Image1: TImage;
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    Label4: TLabel;
    BitBtn1: TBitBtn;
    procedure BitBtn1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
```

```
  end;

var
  fmres: Tfmres;

implementation

uses inform;

{$R *.dfm}

procedure Tfmres.BitBtn1Click(Sender: TObject);
begin
    Form1.Edit1.Clear;
    Form1.Edit2.Clear;
    close;
end;
end.
```

Литература

1. Фаронов В.В. Delphi 6 Учебный курс. - М., «Нолидж», 2002.
2. Культин Н.Б. Delphi 7. Основы программирования в Delphi 7. – СПб: БХВ - Петербург, 2009. – 640с.
3. Могилев А.В., Пак Н.И., Хеннер Е.К. Информатика: Учебное пособие для студентов педвузов / под ред. Е.К. Хеннера. – М.: ACADEMIA, 1999.
4. Могилев А.В., Пак Н.И., Хеннер Е.К. Практикум по информатике: Учеб. пособие для студ. пед. вузов/ Под ред. Е.К. Хеннера. – М.: Изд. центр "Академия", 2001. -608 с.
5. Симанович С., Евсеев Г. Занимательное программирование Delphi. – М., «АСТ-Пресс книга», 2001. – 367 с .
6. Халитова З.Р., Хисматуллина Н.А. Методическое руководство по программированию. Одномерный массив. Задачи и решения. – Казань, КГПУ, 2003. – 26 с.
7. Халитова З.Р., Хисматуллина Н.А. Методическое руководство по программированию. Двумерный массив. Задачи и решения. – Казань, КГПУ, 2003. – 30 с.