МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

Юго-Западный государственный университет

(ЮЗГУ)

Кафедра вычислительной техники

РАЗРАБОТКА ОБЪЕКТНО-ОРИЕНТИРОВАННЫХ ПРИЛОЖЕНИЙ

Методические указания по выполнению лабораторных и практических работ для студентов направления подготовки 09.04.01

Курск 2016

УДК 621.3
Составитель: Э.И. Ватутин

Рецензент
Кандидат технических наук, доцент *В.С. Панищев*

Разработка объектно-ориентированных приложений: методические указания по выполнению лабораторных и практических работ по дисциплине «Технология разработки программного обеспечения» / Юго-Зап. гос. ун-т; сост.: Э.И. Ватутин; Курск, 2016. 31 с.: ил. 2.

Методические рекомендации содержат сведения по проектированию и разработке оконных приложений для решения поставленных переборных задач на современных языках программирования высокого уровня.

Предназначены для студентов направления подготовки 09.04.01 «Информатика и вычислительная техника».

Текст печатается в авторской редакции

Подписано в печать _____. Формат 60x84 1/16.
Усл. печ. л. Уч. – изд.л. Тираж 30 экз. Заказ . Бесплатно.
Юго-Западный государственный университет
305040, Курск, ул. 50 лет Октября, 94.

Содержание

Введение

Целью лабораторных работ является получение практических навыков программирования относительно сложных программ, решающих переборные задачи с использованием графического интерфейса и объектно-ориентированного подхода к проектированию.

Оконный интерфейс является удобных для большинства пользователей и включает кнопки, поля редактирования, таблицы и другие визуальные элементы управления. Большинство оконных приложений обладают хотя бы одним окном. Оконное приложение можно создать по аналогии с консольным приложением с использованием средств WinAPI (регистрация класса окна, создание элементов управления вручную, организация цикла приема сообщений), однако данный набор действий является достаточно непростым (особенно при отсутствии под рукой соответствующей документации) и размер даже простой программы будет составлять несколько сотен строк кода. Delphi предоставляет более удобный способ создания оконных приложений с использованием библиотеки VCL. Весь процесс создания программы сводится к созданию проекта оконного приложения (меню File→New→Application), размещению требуемых компонентов на форме и заданию требуемых свойств, назначению событий и написанию кода соответствующих обработчиков. В отличие от консольного приложения, состоящего из одного файла проекта (dpr), исходный код оконного приложения сосредоточен в нескольких файлах: в файле проекта и файлах модулей (pas).

Файл проекта по умолчанию имеет имя Project1.dpr и выглядит следующим образом:

```
program Project1;

uses
  Forms,
```

← список модулей состоит уже из 2 модулей: Forms содержит объект Application, имеющийся во всех оконных приложениях;

```
  Unit1 in 'Unit1.pas' {Form1};
```
Unit1 содержит код обработчиков формы Form1.

```
{$R *.res}
```
← указание линкеру включить файл ресурсов

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

Также как и у консольного приложения, в начале файла располагает необязательный заголовок. Затем следует секция подключения модулей, в которой присутствуют два модуля: `Forms` и `Unit1`. Первый из них является частью библиотеки VCL, в нем объявлен объект `Application`, который является центральным объектом любого оконного VCL-приложения и инкапсулирует в своем составе богатый набор методов по взаимодействию оконного приложения с операционной системой. Модуль `Unit1` сгенерирован средой для данного проекта и соответствует форме `Form1` (о чем подсказывает комментарий) и содержит код обработчиков событий элементов управления формы. Директива компилятора `{$R *.res}` указывает на необходимость включения в состав исполняемого файла ресурсов из файла с расширением res (имя файла ресурсов совпадает с именем файла проекта). Далее располагается основная программа, которая представляет собой три действия: инициализацию приложения, создание формы `Form1` и запуск цикла приема сообщений. Любое оконное VCL-приложение функционирует следующим образом: при каком-либо действии пользователя или событии операционной системы приложению посылается сообщение, которое в недрах метода `Run` объекта `Application` транслируется в вызов соответствующего обработчика события, определенного пользователем или назначенного библиотекой VCL по умолчанию (пример действия по умолчанию – закрытие приложения после клика мышкой на крестике в верхнем правом углу формы, происходящее независимо от наличия обработчика, определенного программистом).

Завершение работы приложения по умолчанию происходит вместе с закрытием основной формы.

Код модуля `Unit1` приведен ниже.

```
unit Unit1;                          ← заголовок файла модуля

interface                            ← интерфейсная часть модуля

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs;

type
  TForm1 = class(TForm)              ← описание класса создаваемой формы (кюда добавляются
  private                              описания размещаемых на форме компонентов и заголовки их
    { Private declarations }           обработчиков событий)
  public
    { Public declarations }
  end;

var
  Form1: TForm1;                     ← переменная (объект) формы

implementation                       ← часть реализации модуля

{$R *.dfm}                           ← директива компилятору включить dfm-файл с описанием свойств
                                       компонентов формы
                                     ← ниже добавляются обработчики по мере их создания
end.
```

Модуль не является самостоятельным приложением, а может выступать лишь как составная часть программы. Заголовок модуля начинается служебным словом `unit`, после которого следует имя модуля. В Delphi снято ограничение языка Pascal на жесткое соответствие имени модуля и имени файла модуля, однако зачастую эти имена по прежнему совпадают. В данном случае модуль имеет две секции: интерфейсную часть (начинается ключевым словом `interface`) и секцию реализации (начинается ключевым словом `implementation`). В интерфейсной части помимо целого ряда подключаемых модулей (`Windows`, `Messages`, …) присутствует описание класса формы (`TForm1`) и определена переменная формы (`Form1`). Секция реализации на данный момент содержит только директиву компилятора `{$R *.dfm}`, которая отвечает за включение в состав приложения dfm-файла с описанием формы и ее компонентов. По

мере добавления компонентов и обработчиков событий содержимое файла `Unit1.pas` будет увеличиваться, в то время как содержимое файла `Project1.dpr` в большинстве приложений изменяется достаточно редко. При необходимости в проект могут быть добавлены дополнительные модули и формы.

В сгенерированные файлы рекомендуется вносить изменения только при наличии реальной на то необходимости, при этом не следует изменять вручную идентификаторы формы и ее компонентов, дописывать обработчики событий и т.д., т.к. эти элементы помимо pas-файла модуля сохраняются также dfm-файлах, могут быть связаны с ресурсами. С большей частью этой черновой работы Delphi прекрасно справляется самостоятельно. В случае внесения необдуманных изменений можно столкнуться с серией подозрительных с первого взгляда ошибок в синтаксически правильной программе и некорректным поведением программы при запуске. Однако при этом осознанное добавление переменных, типов или подпрограмм в модуль не воспрещается.

Сгенерированным файлам проекта и модулей рекомендуется давать осмысленные имена, не забывая сохранять расширения. Каждый новый проект лучше располагать в новой папке, чтобы не возникало конфликтов с именами файлов других проектов. При необходимости любой проект можно достаточно быстро перенести в другое место.

Сгенерированное оконное приложение на данный момент ничего полезного не делает. Код библиотеки VCL отвечает за корректную обработку сообщений, благодаря чему форму можно перетаскивать в разные части экрана, изменять ее размер, минимизировать/максимизировать и т.д. В качестве примера можно наделить приложение функциональностью калькулятора, производящего сложение двух чисел. Для этого прежде всего необходимо добавить на форму следующие компоненты:

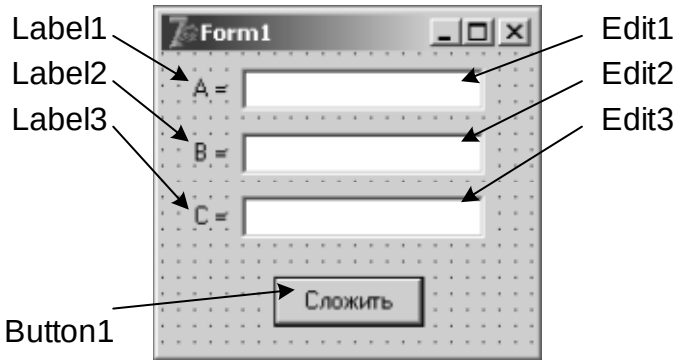

Рис. 1. Форма упрощенного калькулятора с компонентами

После размещения компонентов необходимо изменить значения некоторых полей (Caption у Label, Text у Edit) и назначить обработчик нажатия кнопки. Это можно сделать двумя способами. Наиболее простой и быстрый вариант – двойной клик мышью на компоненте, при этом Delphi сгенерирует скелет одного из обработчиков (у кнопки Button есть целый ряд событий – `OnClick`, `OnContextPopup`, `OnDragDrop` и т.д., для данного компонента будет автоматически выбран и сгенерирован обработчик `OnClick`, а, например, для компонента ComboBox в подобном случае будет сгенерирован обработчик `OnChange`). Второй способ заключается в использовании инспектора объектов. Необходимо выделить компонент, перейти на страничку Events инспектора объектов, выбрать соответствующее событие и сгенерировать для него обработчик двойным кликом мыши справа от названия события. В результате создания обработчика события Delphi сгенерирует следующий код:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
      ← действия (операторы) добавляются сюда
end;
```

На данный момент обработчик не содержит каких-либо действий. В тело обработчика необходимо добавить следующий код:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  A, B, C: Integer;                ← 3 локальные переменные
begin
  A := StrToInt(Edit1.Text);       ← ввод и вывод производятся с использованием Edit'ов
  B := StrToInt(Edit2.Text);

  C := A + B;

  Edit3.Text := IntToStr(C);
end;
```

В обработчике с использованием ключевого слова `var` объявлены три локальные переменные. Переменные `A` и `B` получают свои значения из полей ввода `Edit1` и `Edit2` (см. рис. 1), для этого используется обращение к свойству `Text` компонентов `Edit`. Свойство `Text` имеет строковый тип и несовместимо с переменными `A` и `B` целочисленного типа `Integer`, поэтому для преобразования введенной строки в число используется функция `StrToInt` (если исходные числа введены некорректно, функция `StrToInt` при выполнении преобразования возбудит исключительную ситуацию). Далее производится сложение введенных чисел, результат сохраняется в переменной `C` и затем отображается в компоненте `Edit3`, для чего используется функция `IntToStr`, преобразующая число в строку.

Результат выполнения приложения показан на рис. 2.

Рис. 2. Калькулятор в действии

Часто в оконных приложениях помимо стандартных компонентов применяются средства графического вывода, основанные на рисовании 2d-примитивов (в основном точек, линий и геометрических фигур). Для 3d-рисования разработаны специализированные наборы API (например, OpenGL и DirectX).

У каждого из визуальных компонентов в Delphi есть свойство

```
Canvas: TCanvas;
```

с помощью которого осуществляются любые действия по рисованию на компоненте. У некоторых компонентов данное свойство может быть спрятано в protected или private-части класса, что делает его недоступным напрямую.

Рисование необходимо осуществлять только тогда, когда этого требует операционная система. Например, если в программе есть две формы, одна из которых расположена поверх другой, и верхняя форма перетаскивается пользователем за заголовок окна над нижней формой, то нижняя форма будет получает уведомления о необходимости ее перерисовки. При этом операционная система Windows посылает оконному приложению сообщение `WM_PAINT`, которое транслируется библиотекой VCL в вызов обработчика события `OnPaint` одного из визуальных компонентов. Несмотря на то, что рисование с использованием свойства `Canvas` возможно практически на любом визуальном компоненте, рекомендуется его производить на специально предназначенном для этого компоненте `PaintBox`. Пример кода приведен ниже.

```
procedure TForm1.PaintBox1Paint(Sender: TObject);
begin
  with PaintBox1.Canvas do begin
    Pen.Color := clRed;              // Установка цвета карандаша
    MoveTo(0, 0);                    // Установка текущего положения карандаша
    LineTo(100, 100);                // Рисование линии
```

```
    Brush.Color := clGreen;          // Установка цвета кисти
    Rectangle(30, 30, 70, 70);       // Рисование закрашенного прямоугольника
  end;
end;
```

Рисование линий происходит карандашем (`Pen`), сплошные замкнутые фигуры (прямоугольники, эллипсы и т.п.) закрашиваются с использованием кисти (`Brush`). Для них можно задавать цвет (свойство `Color`), стиль (свойство `Style`) и другие параметры. Для задания цветов можно использовать стандартный набор констант (`clRed`, `clMagenta` и др.), а можно воспользоваться функций

```
function RGB(R, G, B: Byte): TColor;
```

которая формирует искомый цвет отталкиваясь от указанных интенсивностей красной, зеленой и синей компонент в цветовом пространстве RGB.

Довольно частой является ситуация, когда нарисовать что-либо необходимо в данный момент времени, а не в момент прихода сообщения `WM_PAINT`. Для этого можно послать указанное сообщение самому себе с использованием подпрограммы `SendMessage()`, однако более простым способом является вызов метода `Invalidate()` для визуального компонента, который необходимо перерисовать. Вызов данного метода приводит к активации обработчика события `OnPaint`, что в свою очередь приводит к обновлению графического содержимого компонента.

Если рисование выполняется однократно и занимает длительное время, то при обработке сообщения `WM_PAINT` программа может «подвисать», не реагирую на действия пользователя, что является неприятной ситуацией. Для подобных случаев существует способ рисования на виртуальном экране – специальном невизуальном классе `TBitmap`, который напрямую не отображается на форме, однако может быть быстро перерисован (полностью или частично) на любом визуальном компоненте в его обработчике

OnPaint. Указанный класс имеет свойство Canvas и рисование на нем принципиально не отличается от рисования на других визуальных компонентах (например, на PaintBox'е). Пример работы с классом TBitmap приведен ниже.

```
var
  Bmp: TBitmap;

procedure TForm1.FormCreate(Sender: TObject);
begin
  Bmp := TBitmap.Create();                  // Создание класса
  Bmp.Width := PaintBox1.ClientWidth;       // Указание его размера в пикселях
  Bmp.Height := PaintBox1.ClientHeight;

  Form1.ControlStyle := Form1.ControlStyle + [csOpaque];
end;


procedure TForm1.Button1Click(Sender: TObject);
begin
  with Bmp.Canvas do begin
    // Действия по рисованию...
  end;

  PaintBox1.Invalidate();     // Принудительная перерисовка компонента PaintBox
end;


procedure TForm1.PaintBox1Paint(Sender: TObject);
begin
  PaintBox1.Canvas.Draw(0, 0, Bmp);  // Копирование содержимого Bitmap'а на PaintBox
end;
```

При рисовании с использованием виртуального экрана иногда наблюдается неприятное моргание, которого можно избежать путем добавления двойной буферизации для формы:

```
Form1.ControlStyle := Form1.ControlStyle + [csOpaque];
```

Следует также отметить, что все действия рисования в операционной системе Windows производятся через понятие DC (Device Context). При использовании библиотеки VCL работа с DC напрямую спрятана от программиста, что достаточно удобно в большинстве случаев. Однако иногда возникает необходимость в реализации ряда дополнительных действий с использованием WinAPI, которое предоставляет гораздо больше возможностей и работает быстрее.

Пример программы

Задание. Разработать программу для построения и редактирования диагональных латинских квадратов.

Хранение данных программы реализовано в модуле `DataUnit`, который включает в своем составе объявление основного класса `TLatinSquare` для инкапсуляции действий, выполняемых с латинским квадратом. Класс включает в своем составе следующие методы (табл. 1).

Таблица 1. Методы класса `TLatinSquare`

Заголовок метода	Назначение
`function GetItem(Index1, Index2: Integer): Integer;`	Получение элемента квадрата для свойства `Item` класса
`procedure SetItem(Index1, Index2: Integer; const Value: Integer);`	Установка элемента квадрата для свойства `Item` класса
`function GetOrder: Integer;`	Возвращает порядок квадрата
`procedure SetOrder(const Value: Integer);`	Задает порядок квадрата
`procedure SetColors();`	Задает цвета фона ячеек для значений элементов квадрата
`procedure GetCollisions();`	Вычисляет число нарушений при заполнении квадрата
`function GetAllowedItems(X, Y: Integer): TElemsArr;`	Определяет множество доступных значения для указанной позиции
`function GetCellSize(): Integer;`	Возвращает размер ячейки квадрата в пикселях

`constructor Create(const Bmp: TBitmap);`	Конструктор
`procedure RandomFill();`	Случайное заполнение квадрата
`procedure Clear();`	Очистка квадрата
`procedure GreedyFill();`	Жадное заполнение квадрата
`procedure RandomFill_v2();`	Случайное заполнение квадрата, версия 2
`procedure Draw();`	Рисование квадрата на `Bitmap`'е
`procedure GetIJByXY(X, Y: Integer; var I, J: Integer);`	Получение координат ячейки по координатам в пикселях на экране
`function GetEmptyesCnt(): Integer;`	Определение числа не заполненных ячеек квадрата
`procedure FillFirstStrNormalized();`	Заполнение элементов первой строки нормализованного квадрата
`procedure BmpResized(const Bmp: TBitmap);`	Уведомление о изменении размера `Bitmap`'а
`procedure Normalize();`	Нормализация квадрата
`procedure LoadFromFile(FileName: String);`	Загрузка квадрата из файла
`procedure SaveToFile(FileName: String);`	Сохранение квадрата в файл

Программный код модуля приведен ниже.

DATAUNIT.PAS

```
unit DataUnit;

interface

uses
  Windows, Graphics;

type
  TLatinSquareData = array of array of Integer;

  TElemsArr = array of Integer;

  TLatinSquare = class
  private
    fLS: TLatinSquareData;
    fColors: array of TColor;
    fCollisions: array of array of Boolean;
    fBmp: TBitmap;
    function GetItem(Index1, Index2: Integer): Integer;
    procedure SetItem(Index1, Index2: Integer; const Value: Integer);
```

```
    function GetOrder: Integer;
    procedure SetOrder(const Value: Integer);
    procedure SetColors();
    procedure GetCollisions();
    function GetAllowedItems(X, Y: Integer): TElemsArr;
    function GetCellSize(): Integer;
  public
    constructor Create(const Bmp: TBitmap);

    procedure RandomFill();
    procedure Clear();
    procedure GreedyFill();
    procedure RandomFill_v2();
    procedure Draw();
    procedure GetIJByXY(X, Y: Integer; var I, J: Integer);
    function GetEmptyesCnt(): Integer;
    procedure FillFirstStrNormalized();
    procedure BmpResized(const Bmp: TBitmap);
    procedure Normalize();

    procedure LoadFromFile(FileName: String);
    procedure SaveToFile(FileName: String);

    property Items[Index1: Integer; Index2: Integer]: Integer read GetItem write SetItem;
default;
    property Order: Integer read GetOrder write SetOrder;
    property CellSize: Integer read GetCellSize;
  end;

implementation

uses
  Math, SysUtils;

{ TLatinSquare }

procedure TLatinSquare.BmpResized(const Bmp: TBitmap);
begin
  fBmp := Bmp;
end;


procedure TLatinSquare.Clear();
var
  I, J: Integer;
begin
  for I := 0 to High(fLS) do
    for J := 0 to High(fLS) do
      fLS[I][J] := -1;
end;


constructor TLatinSquare.Create(const Bmp: TBitmap);
begin
  fBmp := Bmp;
end;


procedure TLatinSquare.Draw();
var
  ACellSize, I, J, Item: Integer;
  Str: String;
begin
  GetCollisions();

  ACellSize := GetCellSize();

  with fBmp.Canvas do begin
    Pen.Color := clBlack;

    { Очистка (вывод рамки) }
    Brush.Color := clWhite;

    Rectangle(0, 0, fBmp.Width, fBmp.Height);

    Font.Size := 14;
    Font.Name := 'Arial';
    Font.Style := [fsItalic];

    { Вывод элементов }
```

```
    for I := 0 to Order-1 do
      for J := 0 to Order-1 do begin
        Item := fLS[I][J];

        if fCollisions[I][J] then
          Brush.Color := clRed
        else
          if Item >= 0 then
            Brush.Color := fColors[Item]
          else
            Brush.Color := clWhite;

        Rectangle(J*CellSize, {Bmp.Height -} I*ACellSize, (J+1)*ACellSize,
            {Bmp.Height -} (I+1)*ACellSize);

        if Item >= 0 then
          Str := IntToStr(Item)
        else
          Str := '';

        TextOut(J*ACellSize + (ACellSize - TextWidth(Str)) div 2,
           {Bmp.Height -} (I{+1})*ACellSize + (ACellSize - TextHeight(Str)) div 2, Str);
      end;
  end;
end;


procedure TLatinSquare.FillFirstStrNormalized();
var
  I: Integer;
begin
  for I := 0 to High(fLS) do
    fLS[0][I] := I;
end;


function TLatinSquare.GetAllowedItems(X, Y: Integer): TElemsArr;
var
  I, J, K: Integer;
  IsCanUse: Boolean;
begin
  Result := nil;

  for I := 0 to High(fLS) do begin
    IsCanUse := True;

    { Строка }
    for J := 0 to High(fLS) do
      if I = fLS[X][J] then begin
        IsCanUse := False;
        break;
      end;

    if not IsCanUse then
      continue;

    { Столбец }
    for J := 0 to High(fLS) do
      if I = fLS[J][Y] then begin
        IsCanUse := False;
        break;
      end;

    if not IsCanUse then
      continue;

    { Главная диагональ }
    if X = Y then begin
      for J := 0 to High(fLS) do
        if I = fLS[J][J] then begin
          IsCanUse := False;
          break;
        end;

      if not IsCanUse then
        continue;
    end;

    if X = High(fLS) - Y then begin
      { Главная диагональ }
```

```
      for J := 0 to High(fLS) do
        if I = fLS[J][High(fLS)-J] then begin
          IsCanUse := False;
          break;
        end;

      if not IsCanUse then
        continue;
    end;

    { Добавление в результат }
    SetLength(Result, Length(Result)+1);
    Result[High(Result)] := I;
  end;

end;


function TLatinSquare.GetCellSize(): Integer;
begin
  Result := Min(fBmp.Width, fBmp.Height) div Order;
end;


procedure TLatinSquare.GetCollisions();
var
  I, J, K: Integer;
begin
  { Очистка нарушений }
  for I := 0 to High(fCollisions) do
    for J := 0 to High(fCollisions) do
      fCollisions[I][J] := False;

  { Проверка нарушений }

  { 1 - дублирование в строке }
  for I := 0 to High(fCollisions) do
    for J := 0 to High(fCollisions) do
      for K := J+1 to High(fCollisions) do
        if (fLS[I][J] = fLS[I][K]) and (fLS[I][J] >= 0) then begin
          fCollisions[I][J] := True;
          fCollisions[I][K] := True;
        end;


  { 2 - дублирование в столбце }
  for I := 0 to High(fCollisions) do
    for J := 0 to High(fCollisions) do
      for K := I+1 to High(fCollisions) do
        if (fLS[I][J] = fLS[K][J]) and (fLS[I][J] >= 0) then begin
          fCollisions[I][J] := True;
          fCollisions[K][J] := True;
        end;

  { 3 - дублирование на диагоналях }
  for I := 0 to High(fCollisions) do
    for J := I+1 to High(fCollisions) do begin
      if (fLS[I][I] = fLS[J][J]) and (fLS[I][I] >= 0) then begin
        fCollisions[I][I] := True;
        fCollisions[J][J] := True;
      end;

      if (fLS[I][Order-I-1] = fLS[J][Order-J-1]) and (fLS[I][Order-I-1] >= 0) then begin
        fCollisions[I][Order-I-1] := True;
        fCollisions[J][Order-J-1] := True;
      end;
    end;

end;


function TLatinSquare.GetEmptyesCnt(): Integer;
var
  I, J: Integer;
begin
  Result := 0;

  for I := 0 to High(fLS) do
    for J := 0 to High(fLS) do
      //if fLS[I][J] < 0 then
```

```
      //  Inc(Result);

      Result := Result + Integer(fLS[I][J] < 0);
end;


procedure TLatinSquare.GetIJByXY(X, Y: Integer; var I, J: Integer);
begin
  J := X div CellSize;

  if J >= Order then
    J := -1;

  I := Y div CellSize;

  if I >= Order then
    I := -1;
end;


function TLatinSquare.GetItem(Index1, Index2: Integer): Integer;
begin
  ASSERT( (0 <= Index1) and (Index1 < Order) );
  ASSERT( (0 <= Index2) and (Index2 < Order) );

  Result := fLS[Index1][Index2];
end;


function TLatinSquare.GetOrder(): Integer;
begin
  Result := Length(fLS);
end;


procedure TLatinSquare.GreedyFill();
var
  I, J: Integer;
  AllowedItems: TElemsArr;
begin
  Clear();

  for I := 0 to High(fLS) do
    for J := 0 to High(fLS) do begin
      AllowedItems := GetAllowedItems(I, J);

      if Length(AllowedItems) > 0 then
        fLS[I][J] := AllowedItems[0];
    end;
end;


procedure TLatinSquare.LoadFromFile(FileName: String);
var
  F: Text;
  Order: Integer;
  I, J: Integer;
begin
  AssignFile(F, FileName);
  Reset(F);

  { Порядок }
  Read(F, Order);
  SetOrder(Order);

  { Элементы }
  for I := 0 to Order-1 do
    for J := 0 to Order-1 do
      Read(F, fLS[I][J]);

  CloseFile(F);
    end;


procedure TLatinSquare.Normalize();
var
  P: array of Integer;  { i <> p[i] }
  I, J, Old, New: Integer;
begin
  SetLength(P, Order);
```

```
  for I := 0 to Order-1 do begin
    Old := fLS[0][I];

    if Old >= 0 then
      P[Old] := I
    else
      P[Old] := -1;
    end;

  for I := 0 to Order-1 do
    for J := 0 to Order-1 do begin
      Old := fLS[I][J];

      if Old >= 0 then begin
        New := P[Old];
        fLS[I][J] := New;
      end;
      end;
end;


procedure TLatinSquare.RandomFill();
var
  I, J: Integer;
begin
  for I := 0 to High(fLS) do
    for J := 0 to High(fLS) do
      fLS[I][J] := Random(Order);
end;


procedure TLatinSquare.RandomFill_v2();
var
  I, J, Index: Integer;
  AllowedItems, AllowedItems2: TElemsArr;

  function Check(): Boolean;
  var
    II, JJ: Integer;
  begin
     Result := False;

    for II := I to High(fLS) do
      for JJ := 0 to High(fLS) do begin
        if fLS[II][JJ] >= 0 then
          continue;

        AllowedItems := GetAllowedItems(II, JJ);

        if Length(AllowedItems) = 1 then begin
          fLS[II][JJ] := AllowedItems[0];
          Result := True;
        end;
      end;
  end;

begin
  Clear();

  for I := 0 to High(fLS) do
    for J := 0 to High(fLS) do begin
      while Check() do
        ;

      AllowedItems := GetAllowedItems(I, J);

      Index := Random( Length(AllowedItems) );

      if Length(AllowedItems) > 0 then
        fLS[I][J] := AllowedItems[Index];
    end;
end;


procedure TLatinSquare.SaveToFile(FileName: String);
var
  F: Text;
  I, J: Integer;
begin
```

```
  AssignFile(F, FileName);
  ReWrite(F);

  { Порядок }
  Write(F, Order);

  { Элементы по порядку }
  for I := 0 to Order-1 do
    for J := 0 to Order-1 do
      Write(F, ' ', fLS[I][J]);

  CloseFile(F);
    end;


procedure TLatinSquare.SetColors();
var
  I, R, G, B: Integer;
begin
  RandSeed := 0;

  SetLength(fColors, Order);

  for I := 0 to High(fColors) do begin
    R := Random(256);
    G := Random(256);
    B := Random(256);

    fColors[I] := RGB(R, G, B);
  end;
end;


procedure TLatinSquare.SetItem(Index1, Index2: Integer; const Value: Integer);
begin
  ASSERT( (0 <= Index1) and (Index1 < Order) );
  ASSERT( (0 <= Index2) and (Index2 < Order) );

  fLS[Index1][Index2] := Value;
end;


procedure TLatinSquare.SetOrder(const Value: Integer);
begin
  SetLength(fLS, Value, Value);
  SetColors();
  SetLength(fCollisions, Value, Value);
end;


end.
```

Проект включает в своем составе две формы. Первая форма приведена ниже (рис. 3), она является вспомогательной и служит для изменения порядка квадрата, с которым производится работа.

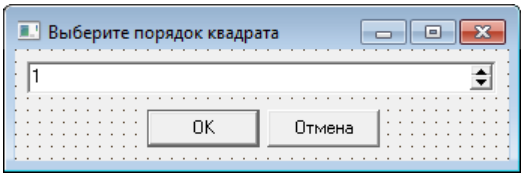

Рис. 3. Форма задания порядка квадрата

Форма включает в своем составе один обработчик `OnKeyDown` нажатия клавиш на клавиатуре для завершения ввода по нажатию на клавишу Enter или досрочного закрытия формы по нажатию на клавишу Esc.

SETORDERFORMUNIT.PAS

```
unit SetOrderFormUnit;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, Spin;

type
  TSetOrderForm = class(TForm)
    Button1: TButton;
    Button2: TButton;
    SpinEdit: TSpinEdit;
    procedure FormKeyDown(Sender: TObject; var Key: Word; Shift: TShiftState);
  end;

var
  SetOrderForm: TSetOrderForm;

implementation

{$R *.dfm}

procedure TSetOrderForm.FormKeyDown(Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  case Key of
    27: ModalResult := mrCancel;
    13: ModalResult := mrOK;
  end;
end;


end.
```

Также проект включает основную форму, с помощью которой производятся основные действия с редактируемым квадратом (рис. 4).

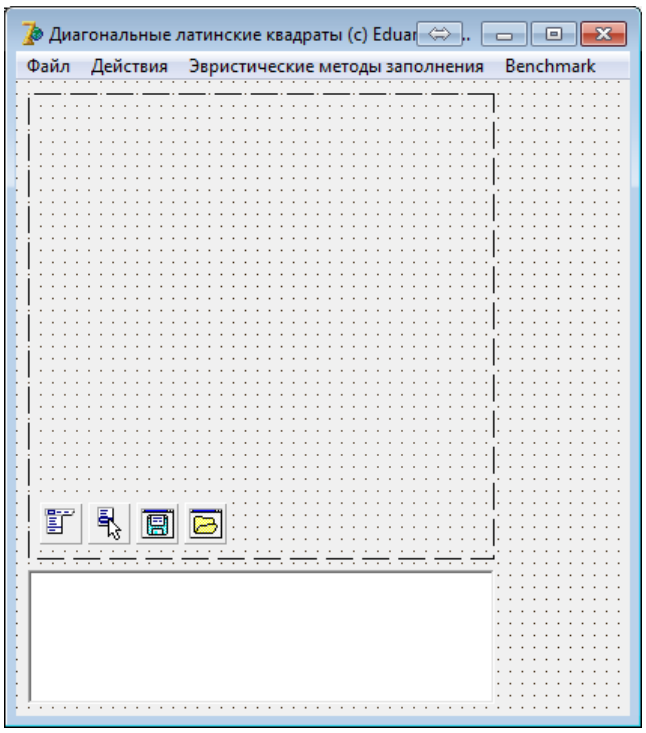

Рис. 4. Основная форма проекта

Она включает в своем составе главное меню `MainMenu`, компонент `PaintBox` для рисования квадрата, поле `Memo` для вывода поясняющей текстовой информации, контекстное меню `PopupMenu`, и диалоги для выбора имени файла при загрузке и сохранении квадрата `OpenDialog` и `SaveDialog`. Код модуля, включающий описание компонентов формы и обработчиков событий, приведен ниже.

MAINUNIT.PAS

```
unit MainUnit;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms, Dialogs, ExtCtrls,
Menus, DataUnit, StdCtrls, MyUtils;

type
  TMainForm = class(TForm)
    MainMenu: TMainMenu;
    test1: TMenuItem;
    PaintBox: TPaintBox;
    PopupMenu: TPopupMenu;
    RandomSearchV1MenuItem: TMenuItem;
    GreedyMenuItem: TMenuItem;
    RandomSearchV2MenuItem: TMenuItem;
    RandomSearchV3MenuItem: TMenuItem;
    N2: TMenuItem;
```

```
    ClearMenuItem: TMenuItem;
    Memo: TMemo;
    ExitMenuItem: TMenuItem;
    SaveAsBmp: TMenuItem;
    SaveDialog: TSaveDialog;
    FillFirstStrNormalizedMenuItem: TMenuItem;
    SetOrderMenuItem: TMenuItem;
    NormalizeSquareMenuItem: TMenuItem;
    N1: TMenuItem;
    SaveMenuItem: TMenuItem;
    LoadMenuItem: TMenuItem;
    OpenDialog: TOpenDialog;
    Benchmark1: TMenuItem;
    GenerateDls9MenuItem: TMenuItem;
    GenerateDls9PgoMenuItem: TMenuItem;
    procedure PaintBoxPaint(Sender: TObject);
    procedure FormShow(Sender: TObject);
    procedure PaintBoxMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
    procedure MenuItemClick(Sender: TObject);
    procedure RandomSearchV1MenuItemClick(Sender: TObject);
    procedure GreedyMenuItemClick(Sender: TObject);
    procedure RandomSearchV2MenuItemClick(Sender: TObject);
    procedure RandomSearchV3MenuItemClick(Sender: TObject);
    procedure ClearMenuItemClick(Sender: TObject);
    procedure ExitMenuItemClick(Sender: TObject);
    procedure SaveAsBmpClick(Sender: TObject);
    procedure FillFirstStrNormalizedMenuItemClick(Sender: TObject);
    procedure PaintBoxClick(Sender: TObject);
    procedure SetOrderMenuItemClick(Sender: TObject);
    procedure FormResize(Sender: TObject);
    procedure NormalizeSquareMenuItemClick(Sender: TObject);
    procedure SaveMenuItemClick(Sender: TObject);
    procedure LoadMenuItemClick(Sender: TObject);
    procedure FormMouseWheel(Sender: TObject; Shift: TShiftState;
      WheelDelta: Integer; MousePos: TPoint; var Handled: Boolean);
    procedure GenerateDls9MenuItemClick(Sender: TObject);
    procedure GenerateDls9PgoMenuItemClick(Sender: TObject);
  private
    fBmp: TBitmap;
    fLS: TLatinSquare;
    fCurrI, fCurrJ: Integer;
    fpc: TFpDataCollection;

    procedure Update();
    procedure FillPopupMenu();
  end;

var
  MainForm: TMainForm;

implementation

uses
  SetOrderFormUnit, Profiler;

{$R *.dfm}

procedure TMainForm.PaintBoxPaint(Sender: TObject);
begin
  PaintBox.Canvas.Draw(0, 0, fBmp);
end;


procedure TMainForm.FormShow(Sender: TObject);
const
  N = 10;
begin
  fBmp := TBitmap.Create();
  fBmp.Width := PaintBox.ClientWidth;
  fBmp.Height := PaintBox.ClientHeight;

  fLS := TLatinSquare.Create(fBmp);
  fLS.Order := N;
  fLS.Clear();
  //fLS.RandomFill();

  //fLS.Draw(fBmp);
  //PaintBox1.Invalidate();
  Update();
```

```
  { 0 ... N-1 }
  FillPopupMenu();
end;


procedure TMainForm.PaintBoxMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
//var
//  I, J: Integer;
begin
  fLS.GetIJByXY(X, Y, fCurrI, fCurrJ);

  //Caption := IntToStr(I) + ' ' + IntToStr(J);
end;


procedure TMainForm.MenuItemClick(Sender: TObject);
var
  I: Integer;
begin
  I := TMenuItem(Sender).Tag;

  if (fCurrI < 0) or (fCurrI >= fLS.Order) or (fCurrJ < 0) or (fCurrJ >= fLS.Order) then
    exit;

  fLS[fCurrI, fCurrJ] := I;

  Update();

  //ShowMessage(IntToStr(I));
end;


procedure TMainForm.RandomSearchV1MenuItemClick(Sender: TObject);
begin
  fLS.Clear();
  fLS.RandomFill();
  //fLS.Items[0, 3] := -1;

  Update();
end;


procedure TMainForm.GreedyMenuItemClick(Sender: TObject);
begin
  fLS.GreedyFill();

  Update();
end;


procedure TMainForm.RandomSearchV2MenuItemClick(Sender: TObject);
begin
  fLS.RandomFill_v2();

  Update();
end;


procedure TMainForm.RandomSearchV3MenuItemClick(Sender: TObject);
var
  I: Integer;
begin
  for I := 1 to 1000 do begin
    fLS.RandomFill_v2();

    if fLS.GetEmptyesCnt() = 0 then begin
      Memo.Lines.Add('Случайный перебор v3: квадрат заполнен с ' + IntToStr(I) + ' попытки');
      break;
    end;
  end;

  //fLS.Draw(fBmp);
  //PaintBox1.Invalidate();
  Update();
end;


procedure TMainForm.ClearMenuItemClick(Sender: TObject);
begin
  fLS.Clear();
```

```
  Update();
end;


procedure TMainForm.Update();
begin
  fLS.Draw();
  PaintBox.Invalidate();
end;


procedure TMainForm.ExitMenuItemClick(Sender: TObject);
begin
  Self.Close();
end;


procedure TMainForm.SaveAsBmpClick(Sender: TObject);
var
  FileName: String;
begin
  SaveDialog.Filter := 'Изображение (*.bmp)|*.bmp';

  if SaveDialog.Execute() then begin
    FileName := SaveDialog.FileName;

    if ExtractFileExt(FileName) = '' then
      FileName := FileName + '.bmp';

    { Подгонка под размер квадрата }
    fBmp.Width := fLS.CellSize * fLS.Order;
    fBmp.Height := fLS.CellSize * fLS.Order;

    fBmp.SaveToFile(FileName);

    fBmp.Width := PaintBox.ClientWidth;
    fBmp.Height := PaintBox.ClientHeight;
  end;
end;


procedure TMainForm.FillFirstStrNormalizedMenuItemClick(Sender: TObject);
begin
  fLS.FillFirstStrNormalized();

  Update();
end;


procedure TMainForm.PaintBoxClick(Sender: TObject);
begin
  { Определение координат }
  if (fCurrI < 0) or (fCurrI >= fLS.Order) or (fCurrJ < 0) or (fCurrJ >= fLS.Order) then
    exit;

  { Изменение элемента }
  fLS[fCurrI, fCurrJ] := (fLS[fCurrI, fCurrJ] + 1) mod fLS.Order;

  Update();
end;


procedure TMainForm.SetOrderMenuItemClick(Sender: TObject);
begin
  if SetOrderForm.ShowModal() <> mrOK then
    exit;

  fLS.Order := SetOrderForm.SpinEdit.Value;
  fLS.Clear();

  FillPopupMenu();

  Update();
end;


procedure TMainForm.FillPopupMenu();
var
  Item: TMenuItem;
  I: Integer;
```

```
begin
  PopupMenu.Items.Clear();

  { Очистить }
  Item := TMenuItem.Create(MainForm);
  Item.Caption := 'Очистить ячейку';
  Item.OnClick := MenuItemClick;
  Item.Tag := -1;

  PopupMenu.Items.Add(Item);

  { Номера }
  for I := 0 to fLS.Order-1 do begin
    Item := TMenuItem.Create(MainForm);
    Item.Caption := IntToStr(I);
    Item.OnClick := MenuItemClick;
    Item.Tag := I;

    PopupMenu.Items.Add(Item);
  end;
end;


procedure TMainForm.FormResize(Sender: TObject);
begin
  PaintBox.Width := MainForm.Width - 32;
  Memo.Width := MainForm.Width - 32;

  PaintBox.Height := MainForm.Height - 170;
  Memo.Top := MainForm.Height - 155;

  fBmp.Free();
  fBmp := TBitmap.Create();
  fBmp.Width := PaintBox.ClientWidth;
  fBmp.Height := PaintBox.ClientHeight;

  fLS.BmpResized(fBmp);

  Update();
end;


procedure TMainForm.NormalizeSquareMenuItemClick(Sender: TObject);
begin
  fLS.Normalize();

  Update();
end;


procedure TMainForm.SaveMenuItemClick(Sender: TObject);
var
  FileName: String;
begin
  SaveDialog.Filter := 'Текстовый файл (*.txt)|*.txt';

  if SaveDialog.Execute() then begin
    FileName := SaveDialog.FileName;

    if ExtractFileExt(FileName) = '' then
      FileName := FileName + '.txt';

    fLS.SaveToFile(FileName);
  end;
end;


procedure TMainForm.LoadMenuItemClick(Sender: TObject);
begin
  OpenDialog.Filter := 'Текстовый файл (*.txt)|*.txt';

  if OpenDialog.Execute() then begin
    fLS.LoadFromFile(OpenDialog.FileName);

    Update();
  end;
end;
```

```
procedure TMainForm.FormMouseWheel(Sender: TObject; Shift: TShiftState; WheelDelta: Integer;
MousePos: TPoint; var Handled: Boolean);
begin
  //Caption := IntToStr(WheelDelta);

  { Определение координат }
  if (fCurrI < 0) or (fCurrI >= fLS.Order) or (fCurrJ < 0) or (fCurrJ >= fLS.Order) then
    exit;

  if WheelDelta > 0 then
    WheelDelta := 1
  else
    WheelDelta := -1;

  { Изменение элемента }
  fLS[fCurrI, fCurrJ] := (fLS[fCurrI, fCurrJ] + WheelDelta + fLS.Order) mod fLS.Order;

  Update();
end;


procedure MeasureNThreads(AppName: String; ThreadsCnt: Integer);
var
  Handle: THandle;
  Handles: array [1..100] of THandle;
  I: Integer;
begin
  { 1 процесс }
  {MyCreateProcess(AppName, Handle);

  WaitForSingleObject(Handle, INFINITE);}

  { N процессов }
  //SetLength(Handles, ThreadsCnt);

  for I := 1 to ThreadsCnt do
    MyCreateProcess(AppName, Handles[I]);

  WaitForMultipleObjects(ThreadsCnt, @Handles[1], True, INFINITE);

  //Handles := nil;
end;


procedure PerformMeasureExperiment(ExeName: String);
var
  AppName: String;
  ProcessorsCnt: Integer;
  I: Integer;
  Pace: Double;
begin
  Screen.Cursor := crHourGlass;

  AppName := ExtractFilePath(ParamStr(0)) + ExeName;
  ProcessorsCnt := GetNumberOfProcessors();

  MainForm.Memo.Lines.Add( IntToStr(ProcessorsCnt) + ' processors found, ' +
IntToStr(ProcessorsCnt*2) + ' tests will be launched');

  for I := 1 to ProcessorsCnt*2 do begin
    StartProf();

    MeasureNThreads(AppName, I);

    StopProf();

    Pace := 1.0 * I / (1.0 * ProfResult() / ProcFreq / 10000000) / 1000;
    MainForm.fpc.Add(Pace);
    MainForm.Memo.Lines.Add( IntToStr(I) + ' thread(s): ' + FloatToStrF(Pace, ffFixed, 10, 2) +
'k DLS/s' );

    Application.ProcessMessages();
  end;

  MainForm.Memo.Lines.Add('OK');

  Screen.Cursor := crDefault;
end;
```

```
procedure TMainForm.GenerateDls9MenuItemClick(Sender: TObject);
begin
  fpc := TFpDataCollection.Create(0);

  Memo.Lines.Add('Base test');
  PerformMeasureExperiment('dls9.exe');

  fpc.SaveToFile('dls9.dat');
  fpc.Free();
end;


procedure TMainForm.GenerateDls9PgoMenuItemClick(Sender: TObject);
begin
  fpc := TFpDataCollection.Create(0);

  Memo.Lines.Add('PGO test');
  PerformMeasureExperiment('dls9pgo.exe');

  fpc.SaveToFile('dls9pgo.dat');
  fpc.Free();
end;


end.
```

Пример латинского квадрата, сформированного методом случайного перебора и открытого в разработанной программе, приведен на рис. 5.

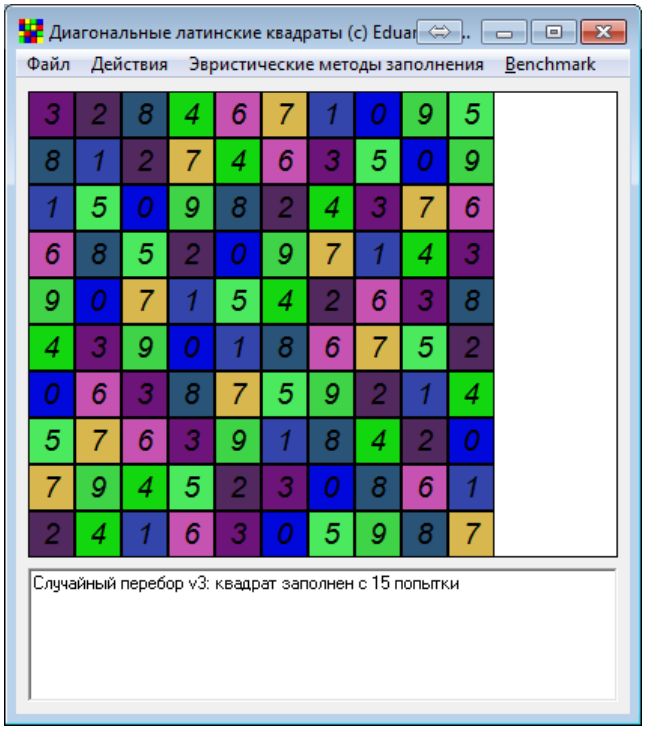

Рис. 5. Пример работы программы

Индивидуальные варианты заданий

1. В заданном ориентированном графе найти кратчайший путь между парой указанных вершин.
2. В заданном неориентированном графе найти все пути между указанной парой вершин. Повторное посещение уже пройденных вершин в путях не допускается.
3. Гамильтонов путь – это путь, однократно проходящий через все вершины графа. В заданном ориентированной графе найти кратчайший гамильтонов путь, соединяющий указанную пару вершин.
4. В заданном неориентированной графе найти все гамильтоновы пути, соединяющие указанную пару вершин.
5. Гамильтонов цикл – это цикл, однократно проходящий через все вершины графа. В заданном неориентированном графе найти кратчайший гамильтонов цикл.
6. В заданном ориентированном графе найти все гамильтоновы циклы.
7. Хроматическим числом неориентированного графа называется минимальное число цветов, в которое можно раскрасить вершины графа так, чтобы соединенные ребром вершины были раскрашены в разные цвета. Определить хроматическое число для заданного неориентированного графа.
8. Найти минимальную раскраску заданного неориентированного графа (см. предыдущее задание).
9. Графы называются изоморфными, если из одного можно получить другой путем перенумерации его вершин. Определить, является ли заданная пара графов изоморфной.
10. Определить, содержит ли заданный граф G подграф $\tilde{G}$, изоморфный графу G'.

11. Для графов G_1 и G_2 найти максимальный по включению изоморфный подграф $\tilde{G}$.
12. Автоморфизмом графа $G = \langle A, V \rangle$ называется такая перестановка σ его вершин, в ходе которой получается граф $G' = \langle \sigma(A), V \rangle$, изоморфный графу *G*. Найти все автоморфизмы заданного графа *G*.
13. Для неориентированного графа заданы веса ребер. Найти такое разбиение графа на *N* непустых подграфов, чтобы сумма ребер, связывающих полученные подграфы, была минимальна.
14. Максимальным независимым множеством называется такое максимальное по включению подмножество вершин графа, в котором ни одна пара вершин не соединена ребром. Для заданного графа найти максимальное независимое множество.
15. Для заданного неориентированного графа найти все полносвязные подграфы из *N* вершин.
16. Для заданного неориентированного графа найти максимальный по включению полносвязный подграф.
17. Для заданного дерева найти все поддеревья, входящие в его состав. Определить их число.
18. Для заданного графа найти все связные подграфы, входящие в его состав. Определить их число.
19. Диаметр графа – это кратчайшее расстояние между парой наиболее удаленных друг от друга вершин графа. Для заданного графа определить его диаметр.
20. Граф называется связным, если для каждой пары вершин существует соединяющий их путь. Определить, является ли заданный неориентированный граф связным.

Содержание отчета

1. Титульный лист.
2. Цель работы.
3. Индивидуальное задание.
4. Краткое описание стратегии решения.
5. Листинг программы.
6. Тестовые примеры, результаты тестирования.
7. Выводы.

Библиографический список

1. Емельянов С.Г., Ватутин Э.И., Панищев В.С., Титов В.С. Процедурно-модульное программирование на Delphi: учебное пособие. М.: Аргамак-Медиа, 2014. 352 с.
2. Зотов И.В., Ватутин Э.И., Борзов Д.Б. Процедурно-ориентированное программирование на C++: учебное пособие. Курск: КурскГТУ, 2008. 211 с.