**РУСЕНСКИ УНИВЕРСИТЕТ „АНГЕЛ КЪНЧЕВ“**

**КУРСОВА РАБОТА**

ПО

ИНТЕГРИРАНИ СРЕДИ

Студент: **Бейтула Хамди Реджеб**

Факултетен номер: **103217**

Група: **26А**

Специалност: **КСТ**

Оценка**:**

Дата: 25.11.2013 г. Проверил:

/ *доц. д-р Ирина Илиева Желязкова*/

# Съдържание

[1. Задание 3](#_Toc402197772)

[2. UML проект 4](#_Toc402197773)

[2.1 Общи сведения за UML 4](#_Toc402197776)

[2.2 Delphi диаграма 6](#_Toc402197777)

[2.3 Диаграма на случаите на употреба 8](#_Toc402197778)

[2.4 Диаграма на последователностите 9](#_Toc402197779)

[2.5 Диаграма на дейностите 10](#_Toc402197780)

[2.6 Диаграма на класовете 11](#_Toc402197781)

[3. Работа с програмата 12](#_Toc402197782)

[4. Особености на програмата 16](#_Toc402197783)

[5. Използвана литература 16](#_Toc402197784)

[6. Код на програмата 16](#_Toc402197785)

[6.1 SourceCodeMain.pas 17](#_Toc402197786)

[6.2 SourceCodeAbout.pas 24](#_Toc402197787)

[6.3 VectorFileSC.pas 25](#_Toc402197788)

# Задание

Да се проектира и реализира като Windows приложение графичен редактор на логически елементи със системно меню, лента с инструменти и панел с елементи. Освен командите на менюто File (New, Open, Save, Save As, Exit), About и за изписване на текст инструменталното средство да предоставя на потребителя и следните възможности:

* Изчертаване на елемент И;
* Изчертаване на елемент ИЛИ-НЕ;
* Изчертаване на вертикална линия;
* Смяна на цвета на писалката;
* Смяна на стила на четката;
* Запазване във векторен формат и като .bmp и .jpeg файл.

# UML проект



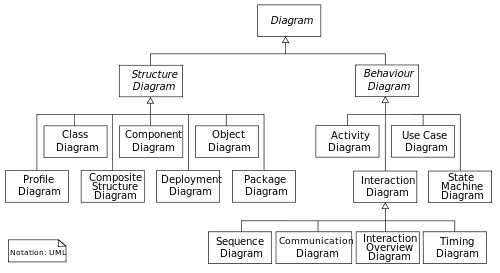
## Общи сведения за UML

Универсалният графичен език за моделиране (Unified Modeling Language – UML) представлява сравнително нов международен стандарт (ISO/IEC 19501:2005 Information Technology – Open Distributed Processing). Езикът е създаден през 1997 г. и е контролиран от Object Management Group (OMG), най-известният стандарт, на която е CORBA (Common Object Request Broker Architecture). UML включва нотация и мета-модел. *Нотацията* са графичните примитиви, използвани в моделите, т.е. това е графичният синтаксис на езика. Както при други графични езици, така и тук нотацията има малка степен на формални дефиниции. *Мета-моделът* са всички видове диаграми, дефиниращи концепцията на езика. Този модел е важен за проектантите, разглеждащи UML като език за програмиране, тъй като дефинира абстрактния му синтаксис.

UML е предназначен за специфициране, визуализиране, модифициране, конструиране и документиране на обектно-ориентирани софтуерни средства/системи (СС). Той е подходящ за всички етапи от процеса на тяхното разработване и особено подходящ за моделиране на паралелни и разпределени софтуерни системи. Важно негово предимство е платформената независимост, т.е. приложимост както за Windows, така и за web-приложения. UML представлява различни аспекти от СС като дейности, потребители, бизнес процеси, схеми за бази от данни, логически компоненти, оператори, многократно използвани софтуерни компоненти.

UML комбинира техниките за моделиране на данни с тези за бизнес, обектно и компонентно моделиране. По-точно той бе синтезиран от нотациите на Буч, обектно-ориентираните техники и обектно-ориентираното софтуерно инженерство в един общ индустриален език за моделиране на СС с реална практическа сложност. След появата му някои от тези методи възприеха неговите нотации. На базата на UML възникнаха и нови методи. UML моделите могат да се обменят чрез различни UML средства, използвайки XML формат за обмен. На настоящия етап UML моделите могат автоматично да се трансформират в други представяния (като например в Java), с помощта на т.н. QVT-подобни езици.

Като език за визуално моделиране UML се състои от 14 вида графични диаграми, разделени в две категории (Фиг. 1), съответно представящи два различни погледа върху този модел.

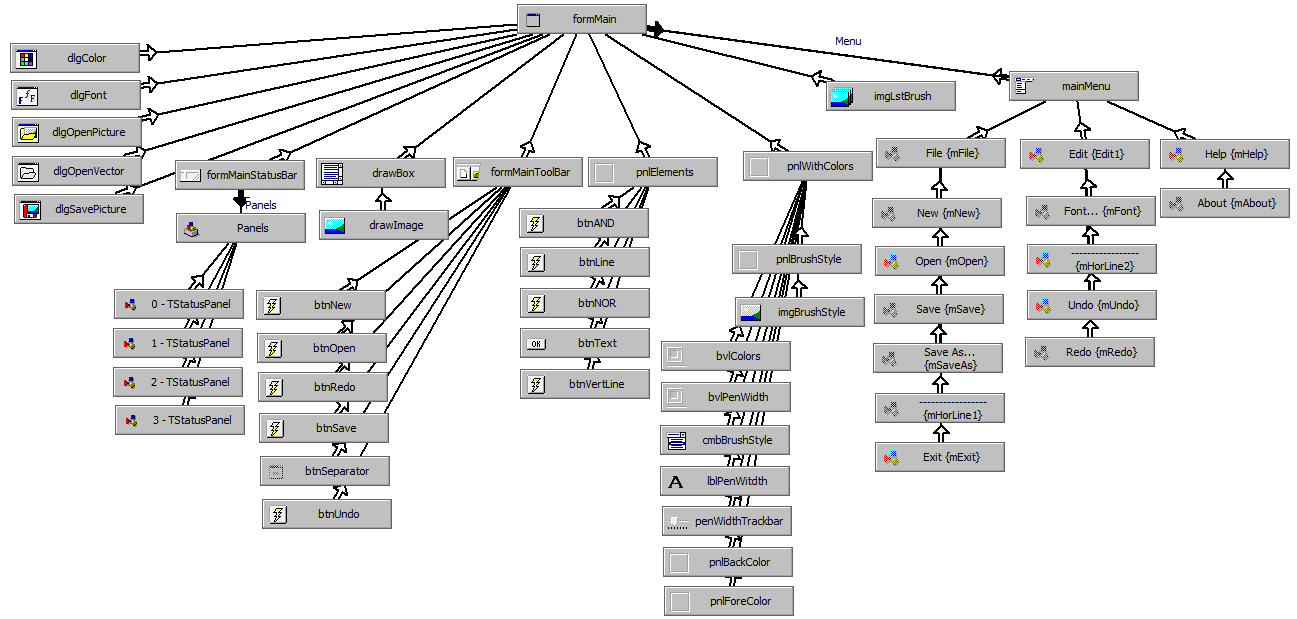


Фиг. 1 Класификация на UML диаграмите

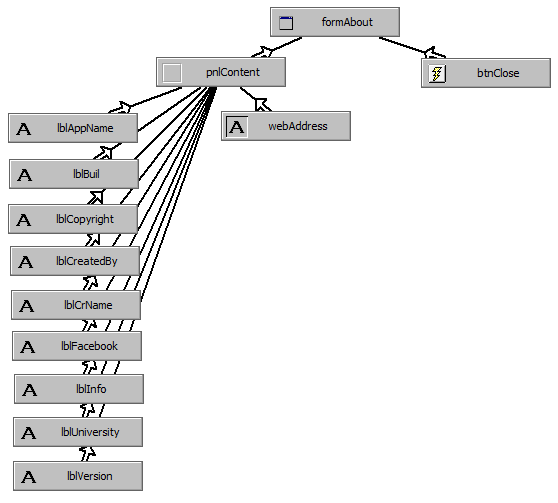
* Статичен (структурен) поглед
* - фокусира върху статичната структура на СС, използвайки термини като обекти, атрибути, операции и връзки със седем диаграми съответно на профилите, класовете, композираната структура, компонентите, обектите и пакетите.
* Динамичен (поведенчески) поглед – фокусира върху динамичното поведение на софтуерната система, показвайки сътрудничеството между обектите и промените в техните състояния. Това става с помощта на диаграмите на случаите на употреба, на дейностите и на крайните автомати. Важен подклас представляват диаграмите на взаимодействие, а именно на последователностите, комуникациите, прегледите и времевите диаграми [1].

## Delphi диаграма

На фиг. 2 и фиг. 3 са представени delphi диаграмите съответно на главната форма и формата за информация.



Фиг. 2 Delphi диаграма на главната форма



Фиг. 3 Delphi диаграма на формата 'Относно'

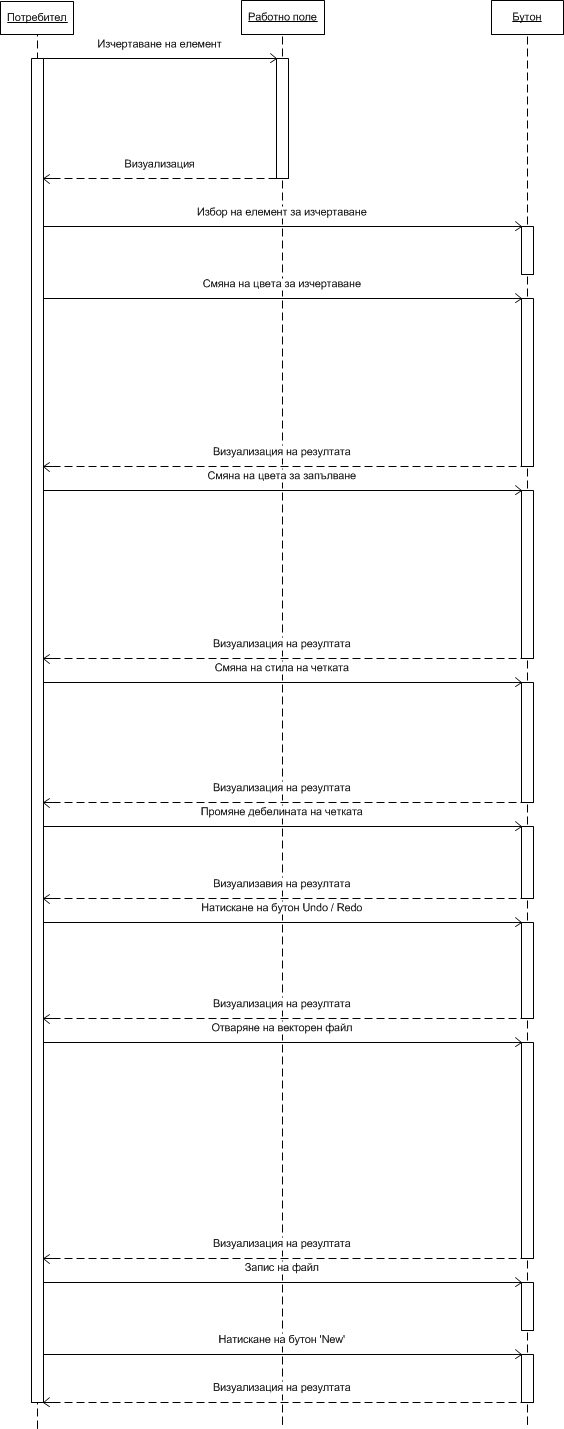
## Диаграма на случаите на употреба



Фиг. 4 Use case диаграма

Първата UML диаграма, създадена в процеса на проектиране на системата бе диаграмата на случаите на употреба (use case diagram) (фиг. 4). Тази диаграма позволява да се опишат на най-високо ниво целите на потребителя, които системата трябва да изпълнява. Тези цели не е необходимо да са задачи или действия, а може да са по общи изисквания към функционалността на системата. С други думи това е техника за определяне на функционалните изисквания на една система. Те описват типичните взаимодействия между потребителите и системата, предоставят описание на начина, по който тя се използва [3].

## Диаграма на последователностите

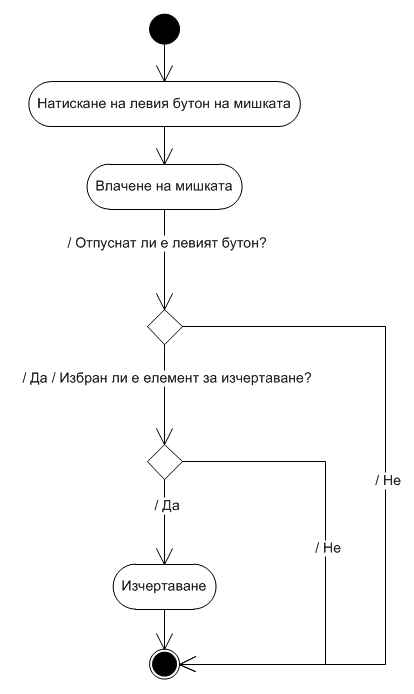


Фиг. 5 Sequence диаграма

Диаграма на последователностите (фиг. 5) се използва, за да се разгледа поведението на няколко обекта в рамките на единствен случай на употреба. Те показват добре съвместната работа между обектите. Диаграмата на последователностите показва реда, в който се случват нещата, като последователност от съобщения [4].

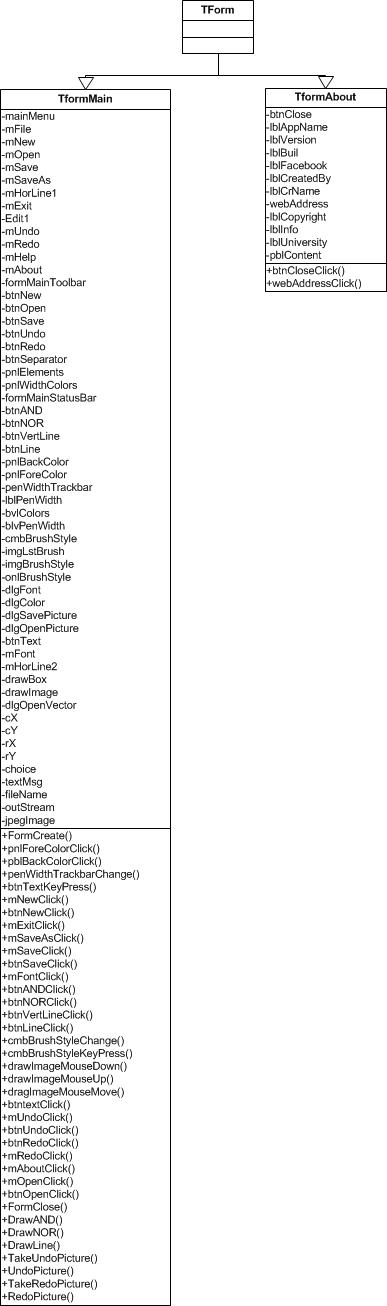
## Диаграма на дейностите

Диаграмата на дейността (activity диаграмата) показва последователността и логиката на действията, които протичат в програмната система.



Фиг. 6 Activity диаграма на логиката за изчертаване

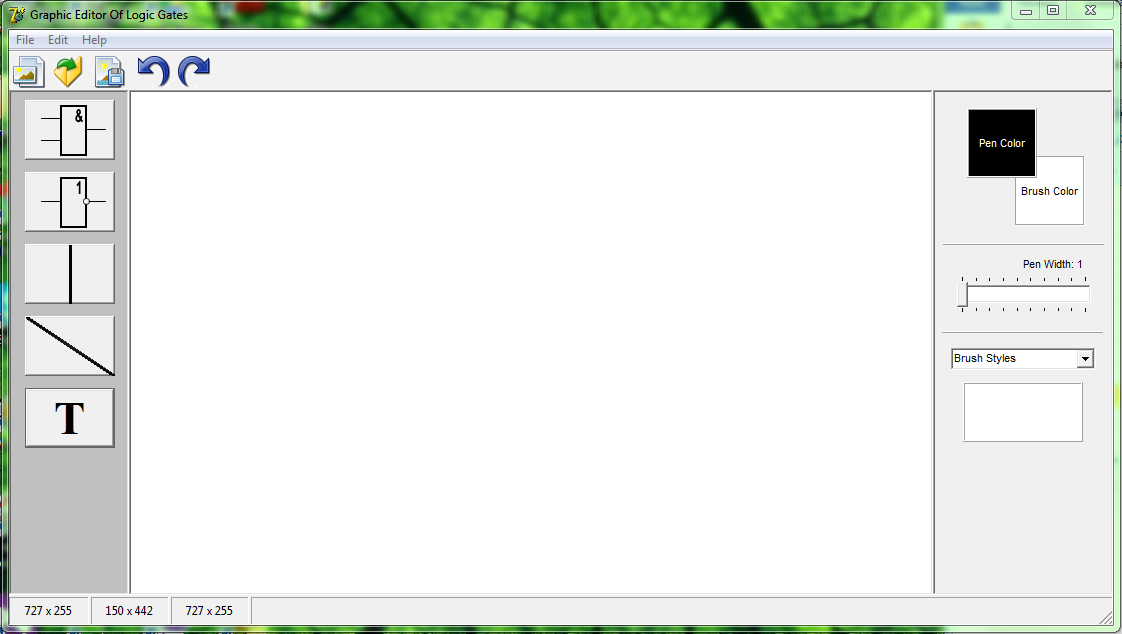
## Диаграма на класовете



Фиг. 7 Class диаграма

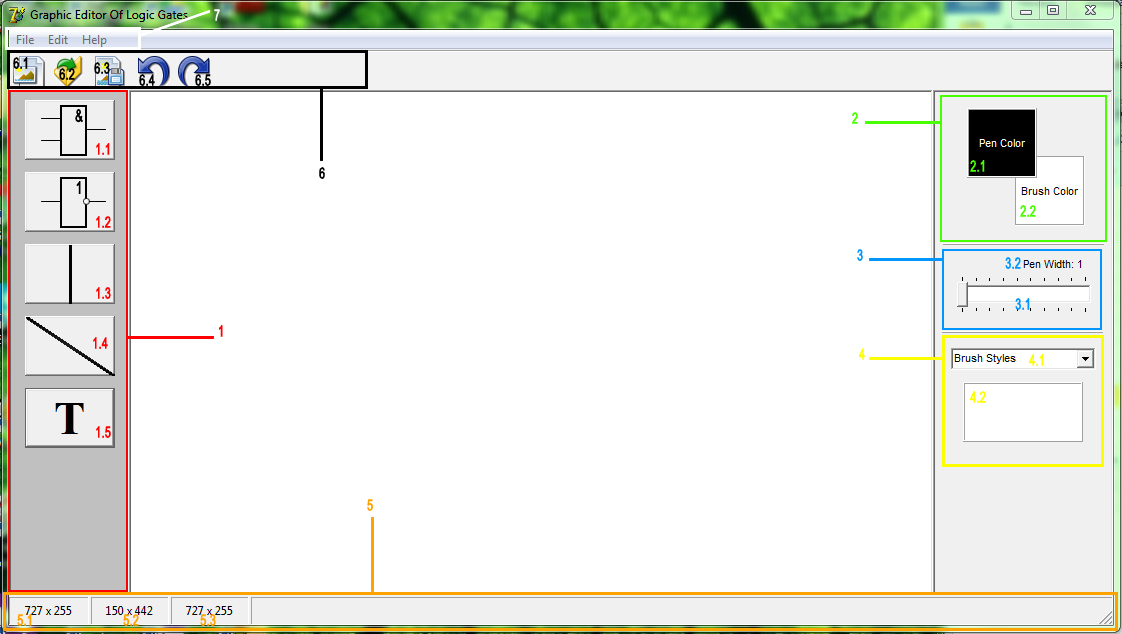
# Работа с програмата

При стартиране на програмния файл \*.ехе на екрана се визуализира прозорецът от фиг. 8.



Фиг. 8 Главен прозорец на приложението

Прозорецът на приложението е разделен визуално, основно, на седем блока (панела) фиг. 9.



Фиг. 9 Панели

1. Панел с елементи:
   1. Бутон за избор на логически елемент И;
   2. Бутон за избор на логически елемент ИЛИ-НЕ;
   3. Бутон за избор на вертикална линия;
   4. Бутон за избор на линия (нормална);
   5. Бутон за избор на текст;
2. Панел за избор на цвят:
   1. Бутон за избор на цвят за изчертаване (Pen Color);
   2. Бутон за избор на цвят за запълване (Brush Color);
3. Панел за избор на дебелината на четката:
   1. Трак-бар за промяна на дебелината;
   2. Етикет за визуализиране на текущата дебелина на четката;
4. Панел за избор на стила на четката:
   1. Падащо меню за избор на стила на четката;
   2. Панел, където се визуализира избраният стил на четката;
5. Лента на състоянието (фиг. 10):



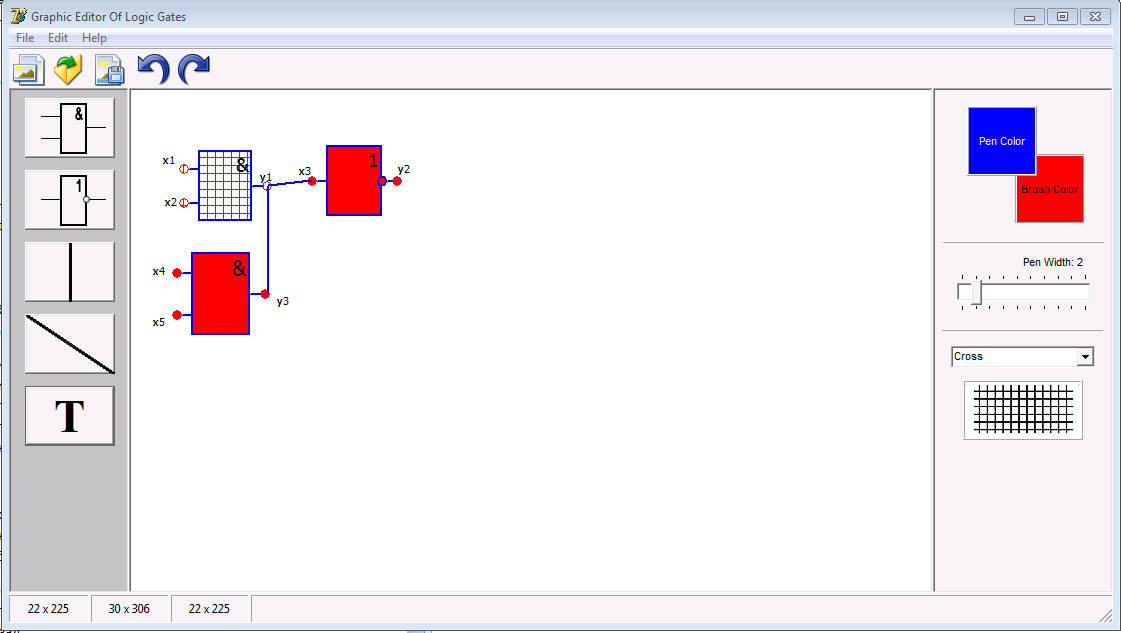
Фиг. 10 Status bar

* 1. Показва координатите на курсора при натискане на левия бутон на мишката;
  2. Показва текущите координати на курсора;
  3. Показва координатите на курсора при отпускане на левия бутон на мишката;

1. Лента с инструменти:
   1. Бутон за създаване на нов файл;
   2. Бутон за отваряне на файл;
   3. Бутон за записване;
   4. Бутон за преминаване стъпка назад – Undo;
   5. Бутон за преминаване стъпка напред – Redo;
2. Главно меню:

* File – съдържа бутони за – създаване, отваряне, записване на файл и изход от приложението;
* Edit – undo, redo бутони и Font – за избор на шрифт;
* Help – съдържа бутон About – за информация относно приложението;

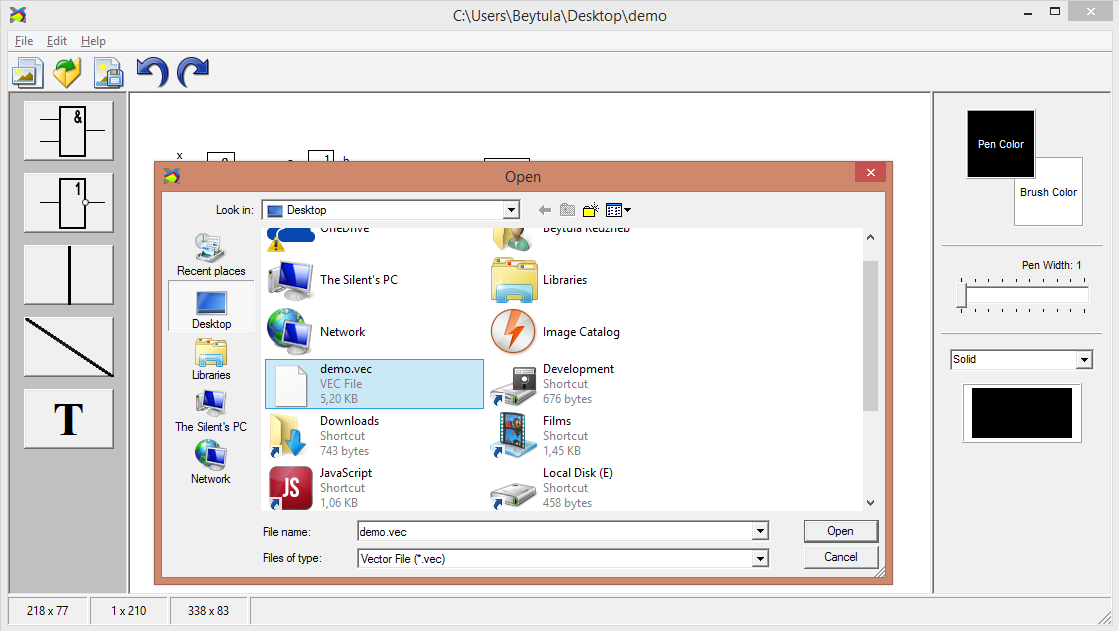
За изчертаване на логически елемент (или линия) е необходимо *първо* да се избере съответният елемент от панела с елементите; *второ* ако е необходимо да се направят промени (цвят, дебелина, стил на четката) е необходимо да се избере съответният панел (2, 3, 4); *трето* кликва се с левия бутон на мишката в работното поле и мишката се мести надясно и надолу (при изчертване на логически елемент) и бутонът се отпуска на желаната позиция – елементът е изчертан.



Фиг. 11 Пример

Ако се избере бутонът за писане на текст (1.5) – необходимо е да се кликне на желаната позиция в работното поле и да се въведе желаният текст. При нужда на промяна на цвета, стила и т.н. на текста е необходимо да се избере Edit->Font.

Бутоните Undo и Redo се използват, когато примерно е начертан грешен елемент и е необходимо връщате стъпка назад – Undo или пък стъпка напред – Redo. Бутоните могат да се използват за връщане на max 4 стъпки напред/назад.



Фиг. 12 Зареждане на векторен файл

Потребителят има възможност за запис (Save и Save As) на създадената от него графика в три различни файла: \*.bmp, \*.jpeg и векторен файл \*.vec. А след зареждане (Open) на векторния файл може да си я редактира (фиг. 12).

Векторният файл е от тип запис. Всеки запис съдържа следните полета, чиито стойности зависят от елемента или от свойството на елемента:

Element = record

name: string[20]; *{име на свойство/елемент/операция}*

parameters: array[1..4] of integer; *{позиция/размер/стойност}*

textMsg: string[100];  *{за нечислoва стойност}*

end;

Например за „линия“ полетата могат да имат следните стойности:

name = ‘NLine’;

parameters[1] = 50; // начална точка по Х

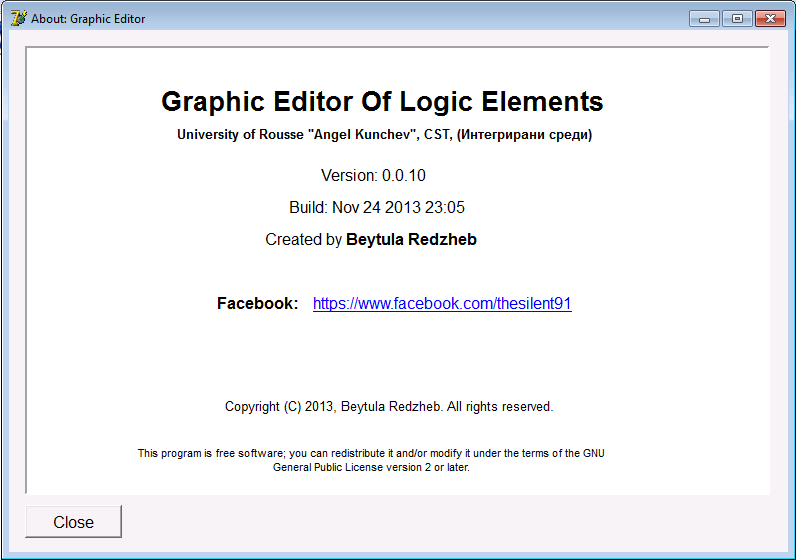
parameters[2] = 50; // начална точка по Y

parameters[3] = 150; // крайна точка по Х

parameters[4] = 140; // крайна точка по Y

textMsg = ‘’;

При избор на Help->About се визуализира прозорецът от фиг. 13, който дава информация за приложението.



Фиг. 13 За програмата

За изход от програмата може да се избере File->Exit или клавишната комбинация Alt+F4.

# Особености на програмата

* Размер – около 1 MB;
* ОС – Windows XP, 7, 8;
* Програмата е тествана на ОС Windows 7 Ultimate x64 и Windows 8 Professional x64 при резолюция на екрана 1920x1080 px.
* С инсталационен файл (*setup.exe*).

# Използвана литература

1. <http://en.wikipedia.org/wiki/Unified_Modeling_Language>, 2013

1. <http://www.delphibasics.co.uk/>, 2013
2. <http://refg.tu-sofia.bg/~bogi/StopFak/SoftEngr/uml_lab1.pdf>, 2013
3. http://refg.tu-sofia.bg/~bogi/StopFak/SoftEngr/uml\_lab3.pdf

# Код на програмата

## SourceCodeMain.pas

unit SourceCodeMain;

interface

uses

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

Dialogs, Menus, ToolWin, ComCtrls, Buttons, ExtCtrls, StdCtrls, ImgList,

ExtDlgs, Jpeg, VectorFileSC, ShellApi;

type

TformMain = class(TForm)

mainMenu: TMainMenu;

mFile: TMenuItem;

mNew: TMenuItem;

mOpen: TMenuItem;

mSave: TMenuItem;

mSaveAs: TMenuItem;

mHorLine1: TMenuItem;

mExit: TMenuItem;

Edit1: TMenuItem;

mUndo: TMenuItem;

mRedo: TMenuItem;

mHelp: TMenuItem;

mAbout: TMenuItem;

formMainToolBar: TToolBar;

btnNew: TSpeedButton;

btnOpen: TSpeedButton;

btnSave: TSpeedButton;

btnUndo: TSpeedButton;

btnRedo: TSpeedButton;

btnSeparator: TToolButton;

pnlElements: TPanel;

pnlWithColors: TPanel;

formMainStatusBar: TStatusBar;

btnAND: TSpeedButton;

btnNOR: TSpeedButton;

btnVertLine: TSpeedButton;

btnLine: TSpeedButton;

pnlBackColor: TPanel;

pnlForeColor: TPanel;

penWidthTrackbar: TTrackBar;

lblPenWitdth: TLabel;

bvlColors: TBevel;

bvlPenWidth: TBevel;

cmbBrushStyle: TComboBox;

imgLstBrush: TImageList;

imgBrushStyle: TImage;

pnlBrushStyle: TPanel;

dlgFont: TFontDialog;

dlgColor: TColorDialog;

dlgSavePicture: TSavePictureDialog;

dlgOpenPicture: TOpenPictureDialog;

btnText: TButton;

mFont: TMenuItem;

mHorLine2: TMenuItem;

drawBox: TScrollBox;

drawImage: TImage;

dlgOpenVector: TOpenDialog;

procedure FormCreate(Sender: TObject);

procedure pnlForeColorClick(Sender: TObject);

procedure pnlBackColorClick(Sender: TObject);

procedure penWidthTrackbarChange(Sender: TObject);

procedure btnTextKeyPress(Sender: TObject; var Key: Char);

procedure mNewClick(Sender: TObject);

procedure btnNewClick(Sender: TObject);

procedure mExitClick(Sender: TObject);

procedure mSaveAsClick(Sender: TObject);

procedure mSaveClick(Sender: TObject);

procedure btnSaveClick(Sender: TObject);

procedure mFontClick(Sender: TObject);

procedure btnANDClick(Sender: TObject);

procedure btnNORClick(Sender: TObject);

procedure btnVertLineClick(Sender: TObject);

procedure btnLineClick(Sender: TObject);

procedure cmbBrushStyleChange(Sender: TObject);

procedure cmbBrushStyleKeyPress(Sender: TObject; var Key: Char);

procedure drawImageMouseDown(Sender: TObject; Button: TMouseButton;

Shift: TShiftState; X, Y: Integer);

procedure drawImageMouseUp(Sender: TObject; Button: TMouseButton;

Shift: TShiftState; X, Y: Integer);

procedure drawImageMouseMove(Sender: TObject; Shift: TShiftState; X,

Y: Integer);

procedure btnTextClick(Sender: TObject);

procedure mUndoClick(Sender: TObject);

procedure btnUndoClick(Sender: TObject);

procedure btnRedoClick(Sender: TObject);

procedure mRedoClick(Sender: TObject);

procedure mAboutClick(Sender: TObject);

procedure mOpenClick(Sender: TObject);

procedure btnOpenClick(Sender: TObject);

procedure FormClose(Sender: TObject; var Action: TCloseAction);

private

{ Private declarations }

public

{ Public declarations }

cX, cY: integer;

rX, rY: integer;

choice: byte;

textMsg: string;

fileName: string;

outStream: TFileStream;

jpegImage: TJpegImage;

procedure DrawAND(bX, bY, eX, eY: integer);

procedure DrawNOR(bX, bY, eX, eY: integer);

procedure DrawLine(bX, bY, eX, eY: integer);

procedure TakeUndoPicture;

procedure UndoPicture;

procedure TakeRedoPicture;

procedure RedoPicture;

end;

TPictureArray = array of TPicture;

TLink = record

x: integer;

y: integer;

r: real;

end;

TEllipseLink = array [1..100] of TLink;

var

formMain: TformMain;

eLink: TEllipseLink;

linkCounter: integer = 0;

aPicUndo: TPictureArray;

aPicRedo: TPictureArray;

openedAndEdit, saveOrNo: boolean;

implementation

uses SourceCodeAbout;

{$R \*.dfm}

procedure TformMain.TakeUndoPicture;

var

i: integer;

begin

if Length(aPicUndo) = 4 then begin

for i := 0 to 2 do begin

aPicUndo[i].Assign(aPicUndo[i + 1]);

end;

end

else begin

Setlength(aPicUndo, Length(aPicUndo) + 1);

aPicUndo[Length(aPicUndo) - 1] := TPicture.Create;

end;

aPicUndo[Length(aPicUndo) - 1].Assign(drawImage.Picture);

end;

procedure TformMain.UndoPicture;

begin

if Length(aPicUndo) = 0 then

exit;

TakeRedoPicture;

drawImage.Picture.Assign(aPicUndo[Length(aPicUndo) - 1]);

aPicUndo[Length(aPicUndo) - 1].Free;

SetLength(aPicUndo, Length(aPicUndo) - 1);

if (elementsCount > 0) then

Dec(elementsCount);

end;

procedure TformMain.TakeRedoPicture;

var

i: integer;

begin

if Length(aPicRedo) = 4 then begin

for i := 0 to 2 do begin

aPicRedo[i].Assign(aPicRedo[i + 1]);

end;

end

else begin

Setlength(aPicRedo, Length(aPicRedo) + 1);

aPicRedo[Length(aPicRedo) - 1] := TPicture.Create;

end;

aPicRedo[Length(aPicRedo) - 1].Assign(drawImage.Picture);

end;

procedure TformMain.RedoPicture;

begin

if Length(aPicRedo) = 0 then

exit;

TakeUndoPicture;

drawImage.Picture.Assign(aPicRedo[Length(aPicRedo) - 1]);

aPicRedo[Length(aPicRedo) - 1].Free;

SetLength(aPicRedo, Length(aPicRedo) - 1);

if (elementsCount < lastStateVal) then

Inc(elementsCount);

end;

procedure TformMain.DrawAND(bX, bY, eX, eY: integer);

var

nColor: TColor;

penW, fontSz: integer;

begin

rX := eX - bX;

rY := eY - bY;

with drawImage.Canvas do begin

fontSz := Font.Size;

Rectangle(bX, bY, eX, eY);

penW := Pen.Width;

Pen.Width := 1;

nColor := Pen.Color;

Pen.Color := clRed;

Ellipse(bX - 20, bY + Round(rY \* 0.25) + 5, bX - 10, bY + Round(rY \* 0.25) - 5);

Pen.Color := nColor;

Pen.Width := penW;

MoveTo(bX - 10, bY + Round(rY \* 0.25));

LineTo(bX, bY + Round(rY \* 0.25));

penW := Pen.Width;

Pen.Width := 1;

nColor := Pen.Color;

Pen.Color := clRed;

Ellipse(bX - 20, bY + Round(rY \* 0.75 ) + 5, bX - 10, bY + Round(rY \* 0.75) - 5);

Pen.Color := nColor;

Pen.Width := penW;

MoveTo(bX - 10, bY + Round(rY \* 0.75));

LineTo(bX, bY + Round(rY \* 0.75));

penW := Pen.Width;

Pen.Width := 1;

nColor := Pen.Color;

Pen.Color := clRed;

Ellipse(eX + 10, bY + Round(rY \* 0.5) + 5, eX + 20, bY + Round(rY \* 0.5) - 5);

Pen.Color := nColor;

Pen.Width := penW;

MoveTo(eX, bY + Round(rY \* 0.5));

LineTo(eX + 10, bY + Round(rY \* 0.5));

Font.Size := Round(rY \* 0.2);

TextOut(eX - (Font.Size + Round(0.05 \* rY)), bY + Round(0.03 \* rY), '&');

Font.Size := fontSz;

end;

end;

procedure TformMain.DrawNOR(bX, bY, eX, eY: integer);

var

nColor: TColor;

penW, fontSz: integer;

begin

rX := eX - bX;

rY := eY - bY;

with drawImage.Canvas do begin

fontSz := Font.Size;

Rectangle(bX, bY, eX, eY);

penW := Pen.Width;

Pen.Width := 1;

nColor := Pen.Color;

Pen.Color := clRed;

Ellipse(bX - 20, bY + Round(rY \* 0.5) + 5, bX - 10, bY + Round(rY \* 0.5) - 5);

Pen.Color := nColor;

Pen.Width := penW;

Ellipse(eX - 5, bY + Round(rY \* 0.5) + 5, eX + 5, bY + Round(rY \* 0.5) - 5);

MoveTo(bX - 10, bY + Round(rY \* 0.5));

LineTo(bX, bY + Round(rY \* 0.5));

penW := Pen.Width;

Pen.Width := 1;

nColor := Pen.Color;

Pen.Color := clRed;

Ellipse(eX + 10, bY + Round(rY \* 0.5) + 5, eX + 20, bY + Round(rY \* 0.5) - 5);

Pen.Color := nColor;

Pen.Width := penW;

MoveTo(eX + 5, bY + Round(ry \* 0.5));

LineTo(eX + 10, bY + Round(rY \* 0.5));

Font.Size := Round(rY \* 0.2);

TextOut(eX - Font.Size, bY + Round(0.03 \* rY), '1');

Font.Size := fontSz;

end;

end;

procedure TformMain.DrawLine(bX, bY, eX, eY: integer);

begin

with drawImage.Canvas do begin

MoveTo(bX, bY);

LineTo(eX, eY);

end;

end;

procedure TformMain.FormCreate(Sender: TObject);

begin

rX := 0;

rY := 0;

drawImage.Canvas.Pen.Color := clBlack;

drawImage.Canvas.Brush.Color := clWhite;

openedAndEdit := false;

saveOrNo := false;

end;

procedure TformMain.pnlForeColorClick(Sender: TObject);

begin

if (dlgColor.Execute) then begin

pnlForeColor.Color := dlgColor.Color;

drawImage.Canvas.Pen.Color := dlgColor.Color;

addElement('ForeColor', dlgColor.Color);

if (pnlForeColor.Color = clWhite) then

pnlForeColor.Font.Color := clBlack

else

pnlForeColor.Font.Color := clWhite;

end

end;

procedure TformMain.pnlBackColorClick(Sender: TObject);

begin

if (dlgColor.Execute) then begin

pnlBackColor.Color := dlgColor.Color;

drawImage.Canvas.Brush.Color := dlgColor.Color;

addElement('BackColor', dlgColor.Color);

if (pnlBackColor.Color = clBlack) then

pnlBackColor.Font.Color := clWhite

else

pnlBackColor.Font.Color := clBlack;

end;

end;

procedure TformMain.penWidthTrackbarChange(Sender: TObject);

begin

lblPenWitdth.Caption := 'Pen Width: ' + IntToStr(penWidthTrackbar.Position);

end;

procedure TformMain.btnTextKeyPress(Sender: TObject; var Key: Char);

begin

if (choice = 5) then begin

textMsg := textMsg + Key;

saveOrNo := true;

addElement('Text', textMsg, cX, cY);

drawImage.Canvas.TextOut(cX, cY, textMsg);

end;

end;

procedure TformMain.mNewClick(Sender: TObject);

begin

fileName := '';

drawImage.Picture := nil;

end;

procedure TformMain.btnNewClick(Sender: TObject);

begin

mNewClick(Sender);

end;

procedure TformMain.mExitClick(Sender: TObject);

begin

formMain.Close;

end;

procedure TformMain.mSaveAsClick(Sender: TObject);

begin

if (dlgSavePicture.Execute) then begin

fileName := dlgSavePicture.FileName;

formMain.Caption := fileName;

drawImage.Picture.SaveToFile(fileName + '.bmp');

jpegImage := TJpegImage.Create;

jpegImage.Assign(drawImage.Picture.Bitmap);

outStream := TFileStream.Create(fileName + '.jpg' ,fmOpenWrite or fmCreate);

jpegImage.SaveToStream(outStream);

outStream.Free;

jpegImage.Free;

saveVectorFile(fileName + '.vec', openedAndEdit);

saveOrNo := false;

end;

end;

procedure TformMain.mSaveClick(Sender: TObject);

begin

if (fileName = '') then

mSaveAsClick(Sender)

else begin

drawImage.Picture.SaveToFile(fileName + '.bmp');

jpegImage := TJpegImage.Create;

jpegImage.Assign(drawImage.Picture.Bitmap);

outStream := TFileStream.Create(fileName + '.jpg' ,fmOpenWrite or fmCreate);

jpegImage.SaveToStream(outStream);

outStream.Free;

jpegImage.Free;

saveVectorFile(fileName + '.vec', openedAndEdit);

saveOrNo := false;

end;

end;

procedure TformMain.btnSaveClick(Sender: TObject);

begin

mSaveClick(Sender);

end;

procedure TformMain.mFontClick(Sender: TObject);

begin

if (dlgFont.Execute) then

drawImage.Canvas.Font := dlgFont.Font;

end;

procedure TformMain.btnANDClick(Sender: TObject);

begin

choice := 1;

end;

procedure TformMain.btnNORClick(Sender: TObject);

begin

choice := 2;

end;

procedure TformMain.btnVertLineClick(Sender: TObject);

begin

choice := 3;

end;

procedure TformMain.btnLineClick(Sender: TObject);

begin

choice := 4;

end;

procedure TformMain.cmbBrushStyleChange(Sender: TObject);

begin

if (cmbBrushStyle.Text = 'Solid') then begin

drawImage.Canvas.Brush.Style := bsSolid;

addElement('BrushStyle', 'Solid', 0, 0);

imgBrushStyle.Picture := nil;

imgLstBrush.GetBitmap(0, imgBrushStyle.Picture.Bitmap);

end

else if (cmbBrushStyle.Text = 'BDiagonal') then begin

drawImage.Canvas.Brush.Style := bsBDiagonal;

addElement('BrushStyle', 'BDiagonal', 0, 0);

imgBrushStyle.Picture := nil;

imgLstBrush.GetBitmap(6, imgBrushStyle.Picture.Bitmap);

end

else if (cmbBrushStyle.Text = 'FDiagonal') then begin

drawImage.Canvas.Brush.Style := bsFDiagonal;

addElement('BrushStyle', 'FDiagonal', 0, 0);

imgBrushStyle.Picture := nil;

imgLstBrush.GetBitmap(1, imgBrushStyle.Picture.Bitmap);

end

else if (cmbBrushStyle.Text = 'Cross') then begin

drawImage.Canvas.Brush.Style := bsCross;

addElement('BrushStyle', 'Cross', 0, 0);

imgBrushStyle.Picture := nil;

imgLstBrush.GetBitmap(2, imgBrushStyle.Picture.Bitmap);

end

else if (cmbBrushStyle.Text = 'DiagCross') then begin

drawImage.Canvas.Brush.Style := bsDiagCross;

addElement('BrushStyle', 'DiagCross', 0, 0);

imgBrushStyle.Picture := nil;

imgLstBrush.GetBitmap(3, imgBrushStyle.Picture.Bitmap);

end

else if (cmbBrushStyle.Text = 'Horizontal') then begin

drawImage.Canvas.Brush.Style := bsHorizontal;

addElement('BrushStyle', 'Horizontal', 0, 0);

imgBrushStyle.Picture := nil;

imgLstBrush.GetBitmap(4, imgBrushStyle.Picture.Bitmap);

end

else if (cmbBrushStyle.Text = 'Vertical') then begin

drawImage.Canvas.Brush.Style := bsVertical;

addElement('BrushStyle', 'Vertical', 0, 0);

imgBrushStyle.Picture := nil;

imgLstBrush.GetBitmap(5, imgBrushStyle.Picture.Bitmap);

end;

end;

procedure TformMain.cmbBrushStyleKeyPress(Sender: TObject; var Key: Char);

begin

Key := #0;

end;

procedure TformMain.drawImageMouseDown(Sender: TObject;

Button: TMouseButton; Shift: TShiftState; X, Y: Integer);

begin

cX := X;

cY := Y;

formMainStatusBar.Panels[0].Text := IntToStr(X) + ' x ' + IntToStr(Y);

if (drawImage.Canvas.Pen.Width <> penWidthTrackbar.Position) then begin

addElement('PenWidth', penWidthTrackbar.Position);

drawImage.Canvas.Pen.Width := penWidthTrackbar.Position;

end;

end;

procedure TformMain.drawImageMouseUp(Sender: TObject; Button: TMouseButton;

Shift: TShiftState; X, Y: Integer);

begin

TakeUndoPicture;

formMainStatusBar.Panels[2].Text := IntToStr(X) + ' x ' + IntToStr(Y);

if (choice = 1) then begin

addElement('AND', cX, cY, X, Y);

DrawAND(cX, cY, X, Y);

saveOrNo := true;

end

else if (choice = 2) then begin

addElement('NOR', cX, cY, X, Y);

DrawNOR(cX, cY, X, Y);

saveOrNo := true;

end

else if (choice = 3) then begin

// Draw Vertical Line

addElement('VLine', cX, cY, cX, Y);

DrawLine(cX, cY, cX, Y);

saveOrNo := true;

end

else if (choice = 4) then begin

// Draw Normal Line

addElement('NLine', cX, cY, X, Y);

DrawLine(cX, cY, X, Y);

saveOrNo := true;

end

else if (choice = 5) then begin

textMsg := '';

end;

end;

procedure TformMain.drawImageMouseMove(Sender: TObject; Shift: TShiftState;

X, Y: Integer);

begin

formMainStatusBar.Panels[1].Text := IntToStr(X) + ' x ' + IntToStr(Y);

end;

procedure TformMain.btnTextClick(Sender: TObject);

begin

choice := 5;

end;

procedure TformMain.mUndoClick(Sender: TObject);

begin

UndoPicture;

end;

procedure TformMain.btnUndoClick(Sender: TObject);

begin

UndoPicture;

end;

procedure TformMain.btnRedoClick(Sender: TObject);

begin

RedoPicture;

end;

procedure TformMain.mRedoClick(Sender: TObject);

begin

RedoPicture;

end;

procedure TformMain.mAboutClick(Sender: TObject);

begin

formAbout.Show;

end;

procedure TformMain.mOpenClick(Sender: TObject);

var

elem: Element;

X, Y: integer;

begin

if (dlgOpenVector.Execute) then begin

mNewClick(Sender);

openedAndEdit := true;

fileName := dlgOpenVector.FileName;

formMain.Caption := fileName;

AssignFile(fileHandler, fileName);

Reset(fileHandler);

while not eof(fileHandler) do begin

Read(fileHandler, elem);

cX := elem.parameters[1];

cY := elem.parameters[2];

X := elem.parameters[3];

Y := elem.parameters[4];

if (elem.name = 'PenWidth') then begin

drawImage.Canvas.Pen.Width := cX;

penWidthTrackbar.Position := cX;

end

else if (elem.name = 'AND') then begin

DrawAND(cX, cY, X, Y);

end

else if (elem.name = 'NOR') then begin

DrawNOR(cX, cY, X, Y);

end

else if (elem.name = 'NLine') then begin

DrawLine(cX, cY, X, Y);

end

else if (elem.name = 'VLine') then begin

DrawLine(cX, cY, X, Y);

end

else if (elem.name = 'ForeColor') then begin

drawImage.Canvas.Pen.Color := cX;

pnlForeColor.Color := cX;

end

else if (elem.name = 'BackColor') then begin

drawImage.Canvas.Brush.Color := cX;

pnlBackColor.Color := cX;

end

else if (elem.name = 'BrushStyle') then begin

cmbBrushStyle.Text := elem.textMsg;

cmbBrushStyleChange(Sender);

end

else if (elem.name = 'Text') then begin

drawImage.Canvas.TextOut(cX, cY, elem.textMsg);

end;

end;

CloseFile(fileHandler);

end;

end;

procedure TformMain.btnOpenClick(Sender: TObject);

begin

mOpenClick(Sender);

end;

procedure TformMain.FormClose(Sender: TObject; var Action: TCloseAction);

var

result: integer;

begin

if (saveOrNo) then

result := MessageBox(0, PChar('Do you want to save the changes?'),

PChar('Confirm'),

MB\_YESNO + MB\_ICONQUESTION);

if (result = mrYes) then

mSaveClick(Sender);

end;

end.

## SourceCodeAbout.pas

unit SourceCodeAbout;

interface

uses

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

Dialogs, StdCtrls, ExtCtrls, Buttons, ShellAPI;

type

TformAbout = class(TForm)

lblAppName: TLabel;

pnlContent: TPanel;

lblVersion: TLabel;

lblBuil: TLabel;

lblFacebook: TLabel;

lblCreatedBy: TLabel;

lblCrName: TLabel;

webAddress: TStaticText;

lblCopyright: TLabel;

lblInfo: TLabel;

lblUniversity: TLabel;

btnClose: TSpeedButton;

procedure btnCloseClick(Sender: TObject);

procedure webAddressClick(Sender: TObject);

private

{ Private declarations }

public

{ Public declarations }

end;

var

formAbout: TformAbout;

implementation

{$R \*.dfm}

procedure TformAbout.btnCloseClick(Sender: TObject);

begin

Close;

end;

procedure TformAbout.webAddressClick(Sender: TObject);

begin

{$IFDEF MSWINDOWS}

ShellExecute(0, 'OPEN', PChar('https://www.facebook.com/thesilent91'), '', '', SW\_SHOWNORMAL);

{$ENDIF MSWINDOWS}

{$IFDEF POSIX}

\_system(PAnsiChar('open ' + AnsiString('https://www.facebook.com/thesilent91')));

{$ENDIF POSIX}

end;

end.

## VectorFileSC.pas

unit VectorFileSC;

interface

type

Element = record

name: string[20];

parameters: array[1..4] of integer;

textMsg: string[100];

end;

var

elements: array[1..100] of Element;

elementsCount, lastStateVal: byte;

fileHandler: file of Element;

procedure addElement(name: string; p1, p2, p3, p4: integer); overload;

procedure addElement(name: string; p: integer); overload;

procedure addElement(name, txtMSG: string; p1, p2: integer); overload;

procedure saveVectorFile(filename: string; openedAndEdit: boolean);

implementation

procedure addElement(name: string; p1, p2, p3, p4: integer); overload;

begin

Inc(elementsCount);

lastStateVal := elementsCount;

elements[elementsCount].name := name;

elements[elementsCount].parameters[1] := p1;

elements[elementsCount].parameters[2] := p2;

elements[elementsCount].parameters[3] := p3;

elements[elementsCount].parameters[4] := p4;

end;

procedure addElement(name: string; p: integer); overload;

begin

Inc(elementsCount);

lastStateVal := elementsCount;

elements[elementsCount].name := name;

elements[elementsCount].parameters[1] := p;

end;

procedure addElement(name, txtMSG: string; p1, p2: integer); overload;

begin

Inc(elementsCount);

lastStateVal := elementsCount;

elements[elementsCount].name := name;

elements[elementsCount].textMsg := txtMSG;

elements[elementsCount].parameters[1] := p1;

elements[elementsCount].parameters[2] := p2;

end;

procedure saveVectorFile(fileName: string; openedAndEdit: boolean);

var

i: integer;

begin

AssignFile(fileHandler, fileName);

if (openedAndEdit) then begin

Reset(fileHandler);

Seek(fileHandler, FileSize(fileHandler));

for i := 1 to elementsCount do

Write(fileHandler, elements[i]);

end

else begin

Rewrite(fileHandler);

for i := 1 to elementsCount do

Write(fileHandler, elements[i]);

end;

Close(fileHandler);

end;

end.

[2]