**6. РУКОВОДСТВО ПО УСТАНОВКЕ И ИСПОЛЬЗОВАНИЮ**

Чтобы начать использование программы, нужно использовать установщик. Для этого достаточно запустить файл setup.exe, который представлен на рисунке 6.1.

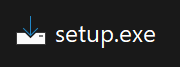


Рисунок 6.1 – Установщик программы

После запуска установщика пользователь должен выбрать режим установки в окне, представленном на рисунке 6.2.

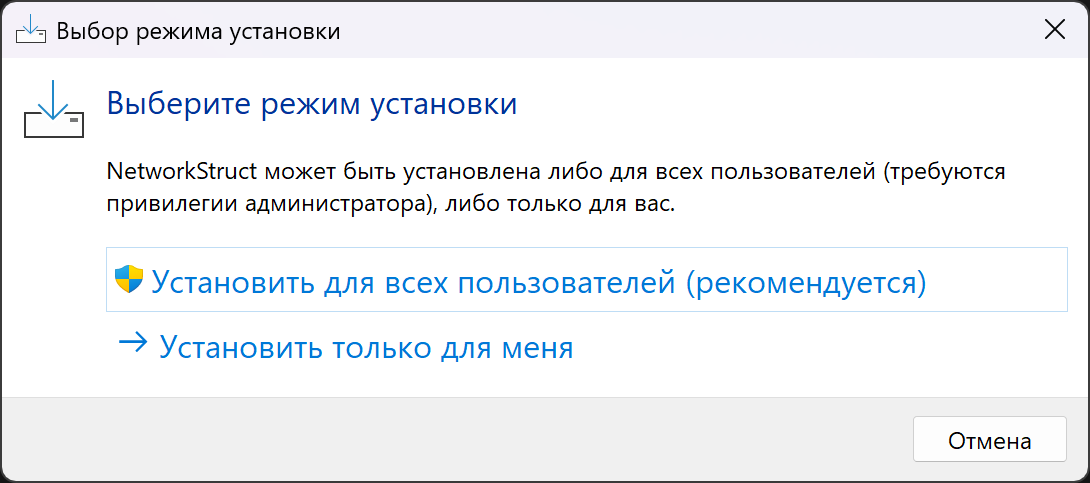


Рисунок 6.2 – Выбор режима установки

В следующем окне пользователь должен выбрать язык установщика. Выбрать можно либо русский, либо английский. Окно выбора языка установщика представлено на рисунке 6.3.

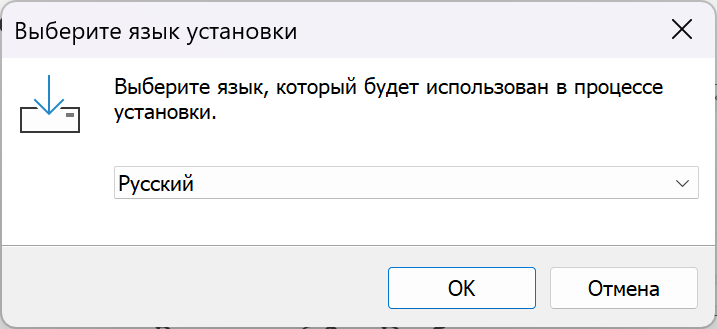


Рисунок 6.3 – Выбор языка установщика

После выбора языка пользователю предлагается выбрать место для установки приложения. Путь по умолчанию - C:\Program Files (x86)\NetworkStruct. Окно выбора пути представлено на рисунке 6.4.

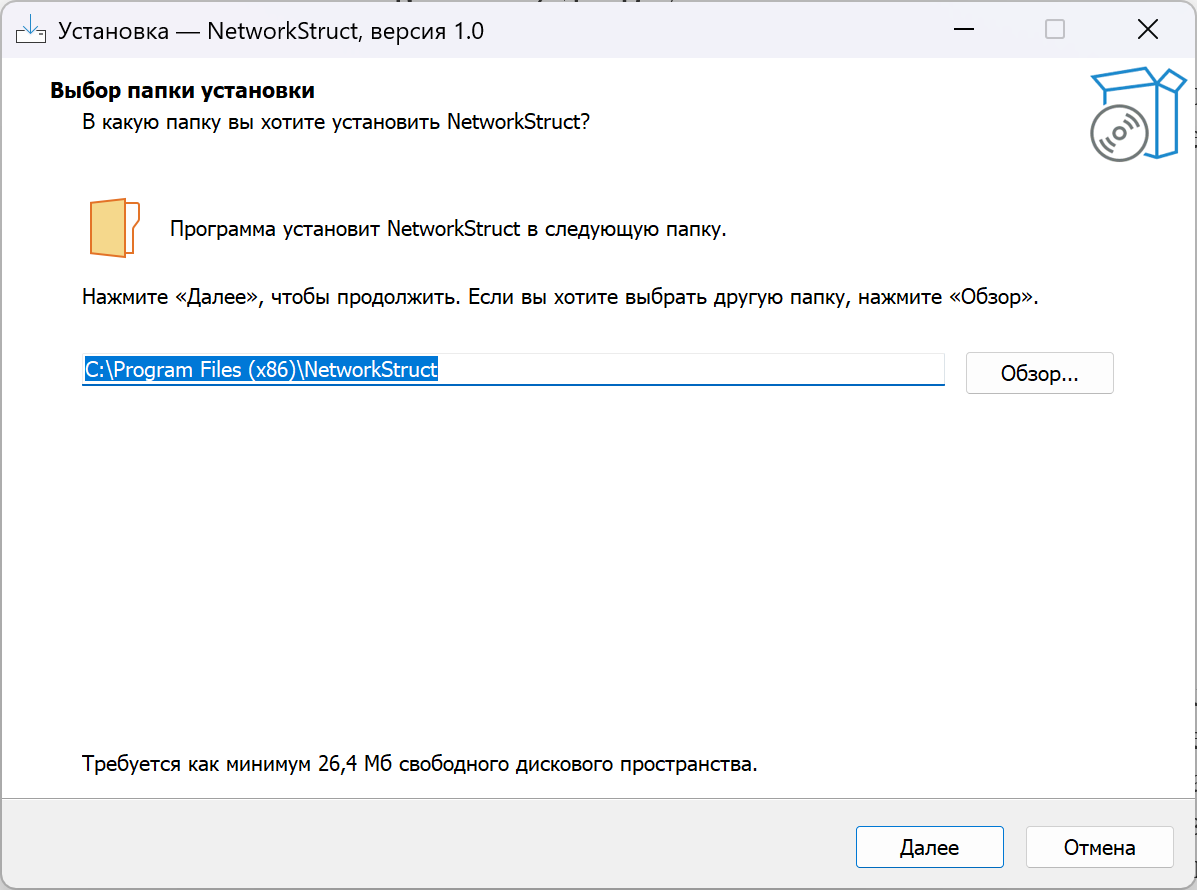


Рисунок 6.4 – Выбор пути для установки

После выбора пути открывается окно, в котором предлагается создать значок на Рабочем столе. Данное окно представлено на рисунке 6.5.

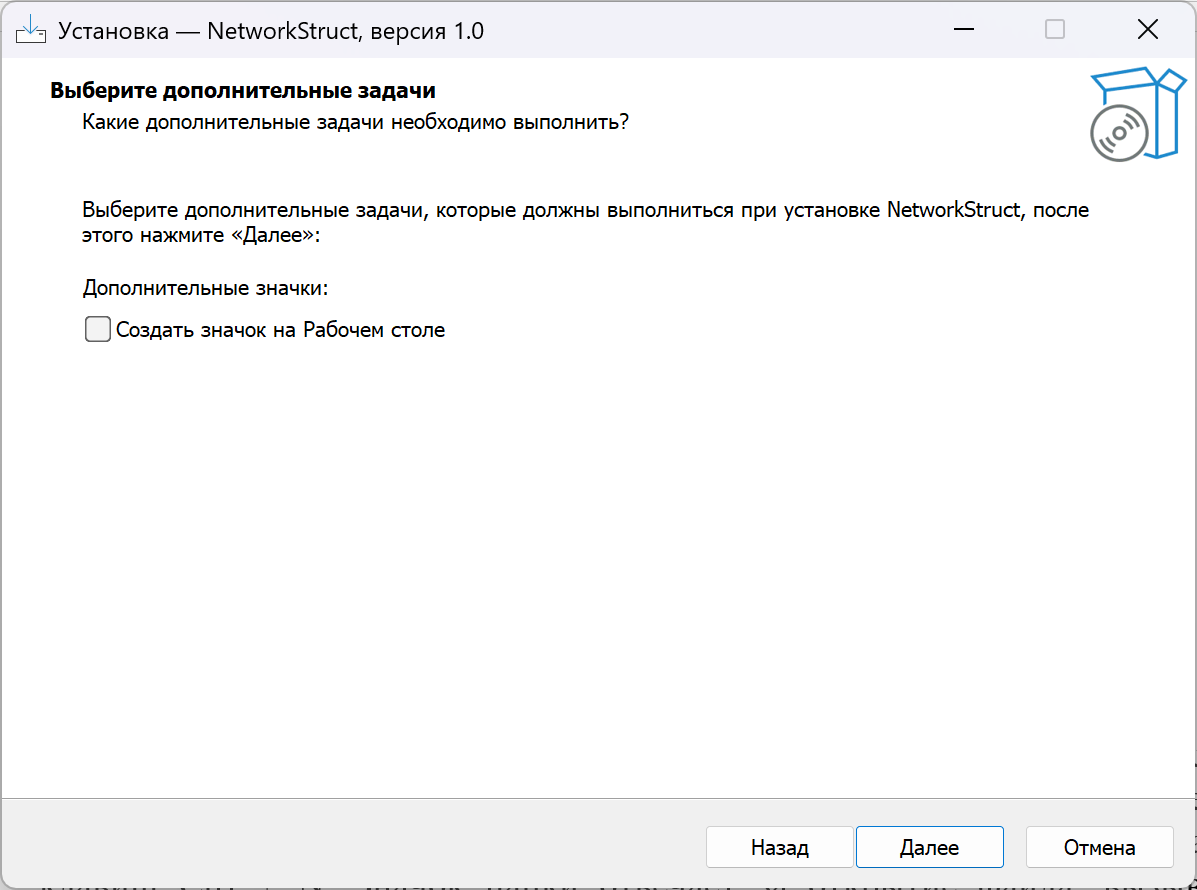


Рисунок 6.5 – Выбор дополнительных задач

Далее пользователь видит окно подтверждения установки, которое представлено на рисунке 6.6.

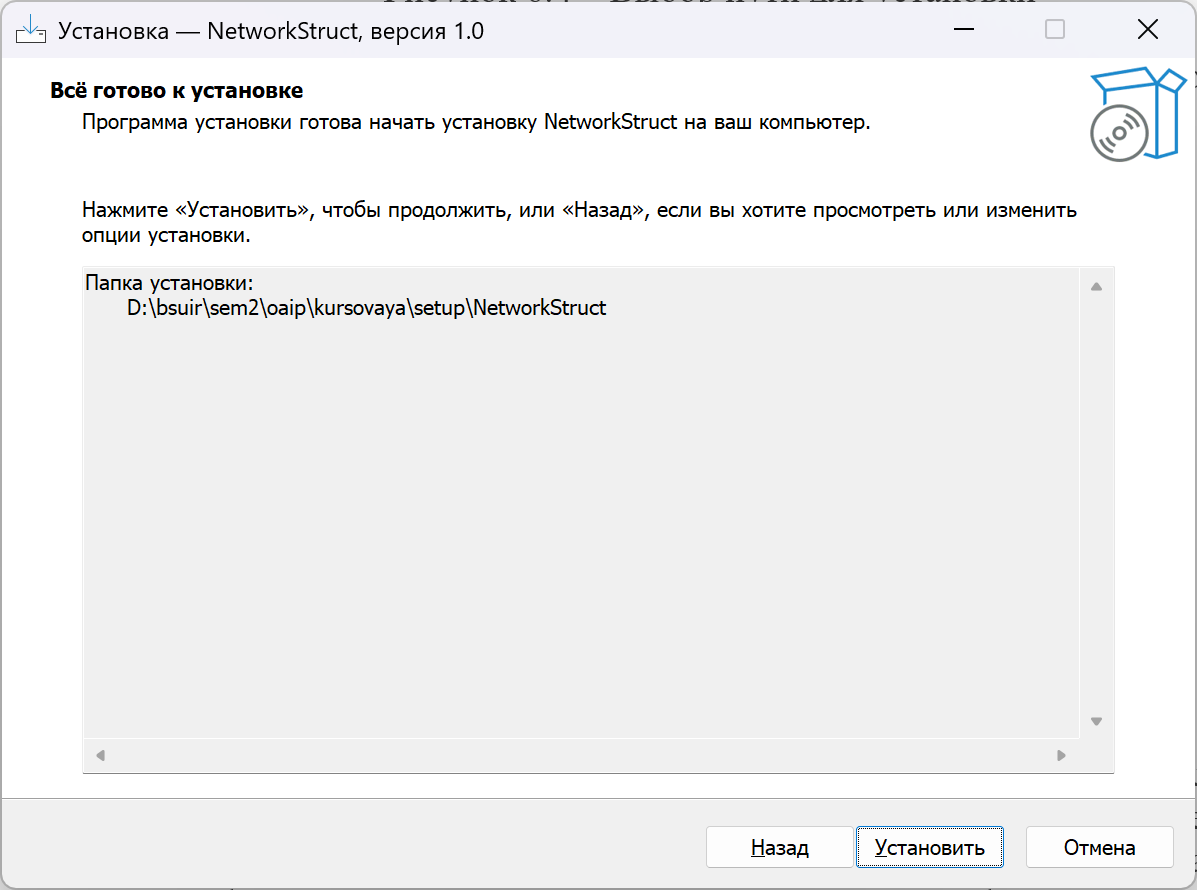


Рисунок 6.6 – Окно подтверждения установки

После установки пользователь увидит окно, представленное на рисунке 6.7, которое предложит сразу запустить программу, либо завершить установку без запуска.

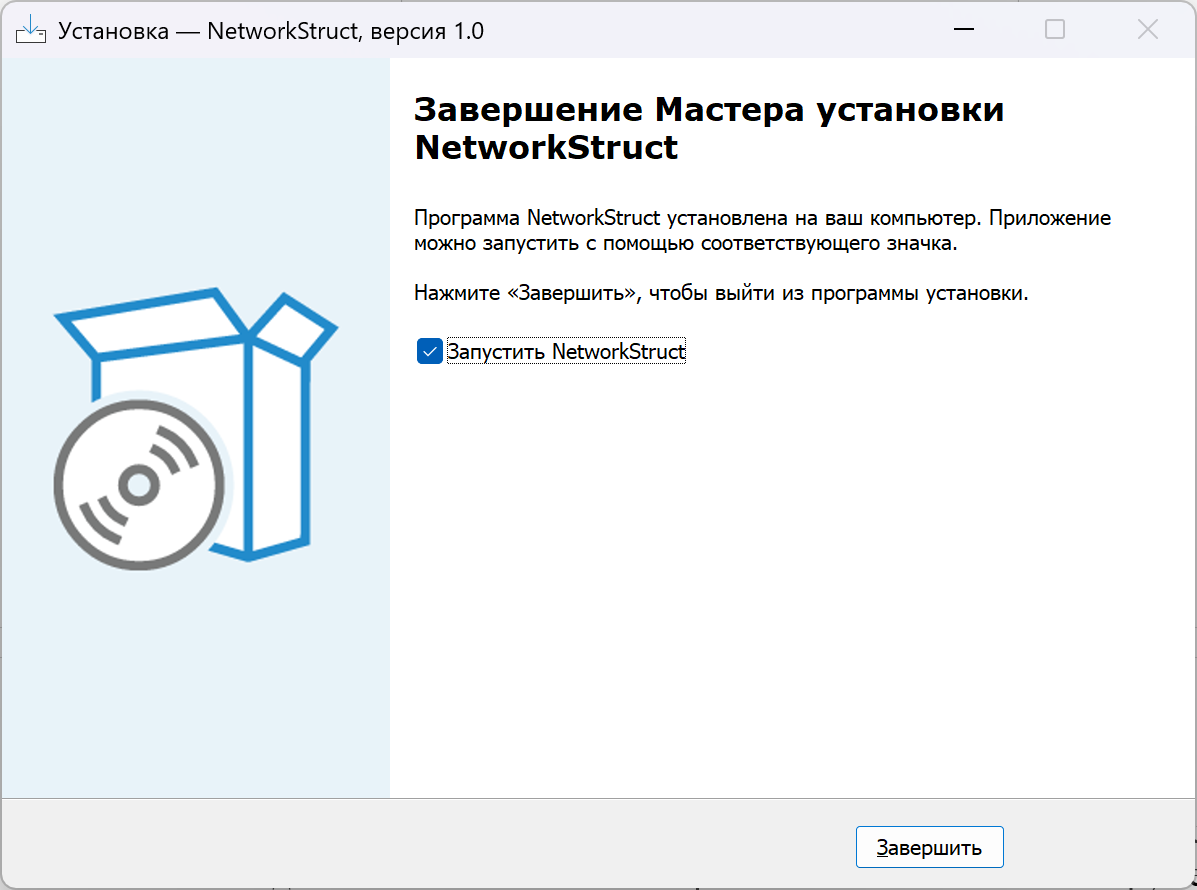


Рисунок 6.7 – Окно завершения установки

Папка, в которую была произведена установка должна содержать 7 элементов: map.bmp, Project1.exe, shipments.txt, shops.txt, unins000.dat, unins000.exe, warehouses.txt. Данная структура представлена на рисунке 6.8.

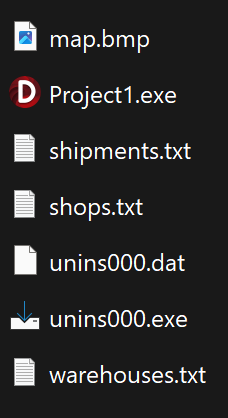


Рисунок 6.8 – Содержимое папки с программой

Map.bmp – фон (схематичное изображение карты), который отображается на главной странице программы. Project1.exe – исполняемый файл программы. Shipments.txt, shops.txt, warehouses.txt – текстовые файлы, которые хранят информацию сохраненных складов, магазинов, отгрузок. Программа не гарантирует корректную работу, при ручном изменении (если данные введены неверно) какого-либо из файлов: shipments.txt, warehouses.txt, shops.txt. Рекомендуется сохранять данные при помощи комбинации клавиш Ctrl+S или через вкладку программы файл->Сохранить. Данные в файле shipments.txt должен быть расположены в определенном формате: первая строка должна содержать количество отгрузок, последующие строки содержат информацию о каждой отгрузке в соответствующем формате. Первая строка каждой отгрузки – ее название, вторая строка – ее идентификатор, третья – идентификатор отправителя, четвертая – идентификатор получателя, пятая – название продукта, участвующего в отгрузке, шестая – количество условных единиц товара, которое участвует в отгрузке. Данные в файле shops.txt и в файле warehouses.txt должны быть также должны быть записаны в определенном формате. Первая строка файла – количество объектов (магазинов или складов), последующие строки хранят информацию о каждом объекте в определенном формате. Первая строка информации об объекте содержит название объекта, вторая – улицу, на которой расположен объект, третья – номер дома, четвертая – корпус (-1 если корпус отсутствует), пятая – вместимость объекта, шестая – занятое место, седьмая – место, зарезервированное под дальнейшие доставки, восьмая – идентификатор объекта, девятая строка – количество товаров, которые хранятся в этом объекте, далее идет описание товаров в определенном формате. Первая строка описания каждого товара - название товара, вторая – категория (пустая строка, если категория отсутствует), третья – место, занимаемое единицей товара, четвертая – количество товара, пятая – идентификатор товара, шестая – количество товара, которое отложено для дальнейших доставок.

После запуска приложения пользователь видит главное меню, которое показано на рисунке 6.9.



Рисунок 6.9 – Главное меню

Для загрузки данных из файла пользователь должен воспользоваться сочетанием клавиш Ctrl+O, либо вкладкой меню Файл->Загрузить, изображенной на рисунке 6.10. При этом все данные, которые уже были внесены в приложение будут удалены.

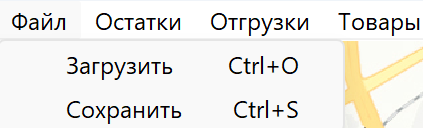


Рисунок 6.10 – Вкладка файл

После нажатия “Сохранить” появляется окно подтверждения, изображенное на рисунке 6.11.

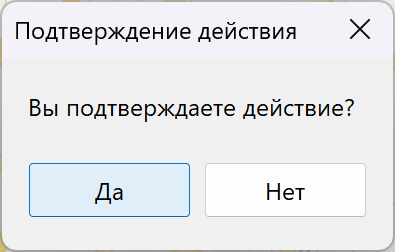


Рисунок 6.11 – Подтверждение загрузки из файлов

После загрузки файлов пользователь увидит сообщение, изображенное на рисунке 6.12.

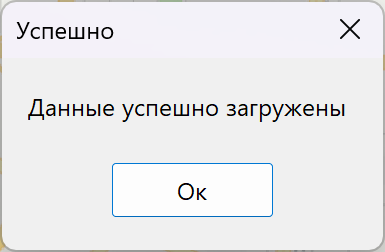


Рисунок 6.12 – Успешная загрузка данных

Программа состоит из главного меню, на котором расположены вкладки «Файл» и «Вид». Ниже главного меню располагается панель инструментов. Значок файла отвечает за создание нового файла, вызывается сочетанием клавиш Ctrl + N. Значок папки отвечает за открытие файла, вызывается горячими клавишами Ctrl + O. Значок дискеты отвечает за сохранение типизированного файла, вызывается горячими клавишами Ctrl + S. Значок дискеты с красным плюсиком отвечает за сохранение файла в растровом или векторном формате, вызывается горячими клавишами Shift + F12.

Помимо кнопок работы с файлами находится кнопки очищения полотна и отмены действия. Очищение полотна происходит при нажатии на кнопку с изображением шредера или через сочетание клавиш Ctrl + Del. Отмена действия происходит при нажатии на кнопку с синей стрелкой или через сочетание клавиш Ctrl + Z.

Справа от выше упомянутых кнопок располагаются кнопки для переключения между режимами рисования: «Заголовок синтаксической диаграммы», «Метапеременная», «Метаконстанта», «Линия». Они также могут быть выполнены через сочетания клавиш Ctrl + 1, Ctrl + 2, Ctrl + 3 и

Ctrl + 4 соответственно.

Правее находятся поля для ввода. Поле «Текст внутри фигуры» предназначено для изменения текста внутри фигуры. Поле «Ширина полотна» и «Высота полотна» позволяют пользователю изменить размеры полотна, на котором происходит рисование, подтверждаются изменения нажатием клавиши Enter.

Большую часть диалогового окна занимает белое полотно, на котором пользователь может строить синтаксические диаграммы (см. рис. 6.2).

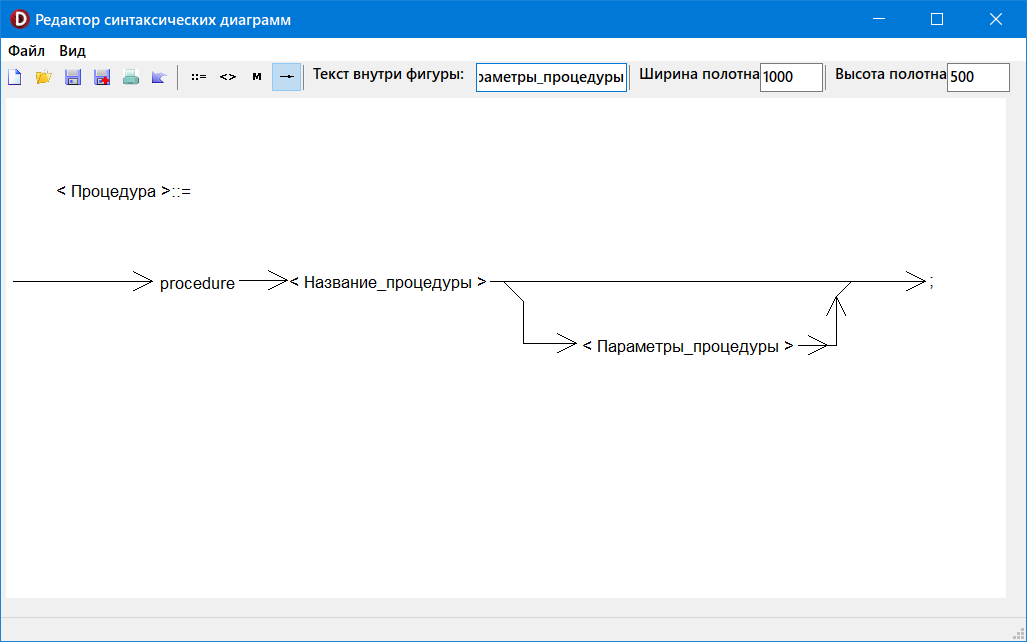


Рисунок 6.2 – Пример построенной синтаксической диаграммы

Также пользователь может экспортировать данную синтаксическую диаграмму в SVG файл. После экспорта появится файл с расширением SVG. После открытия данного файла откроется окно браузера по умолчанию. На открытой странице будет изображена синтаксическая диаграмма.

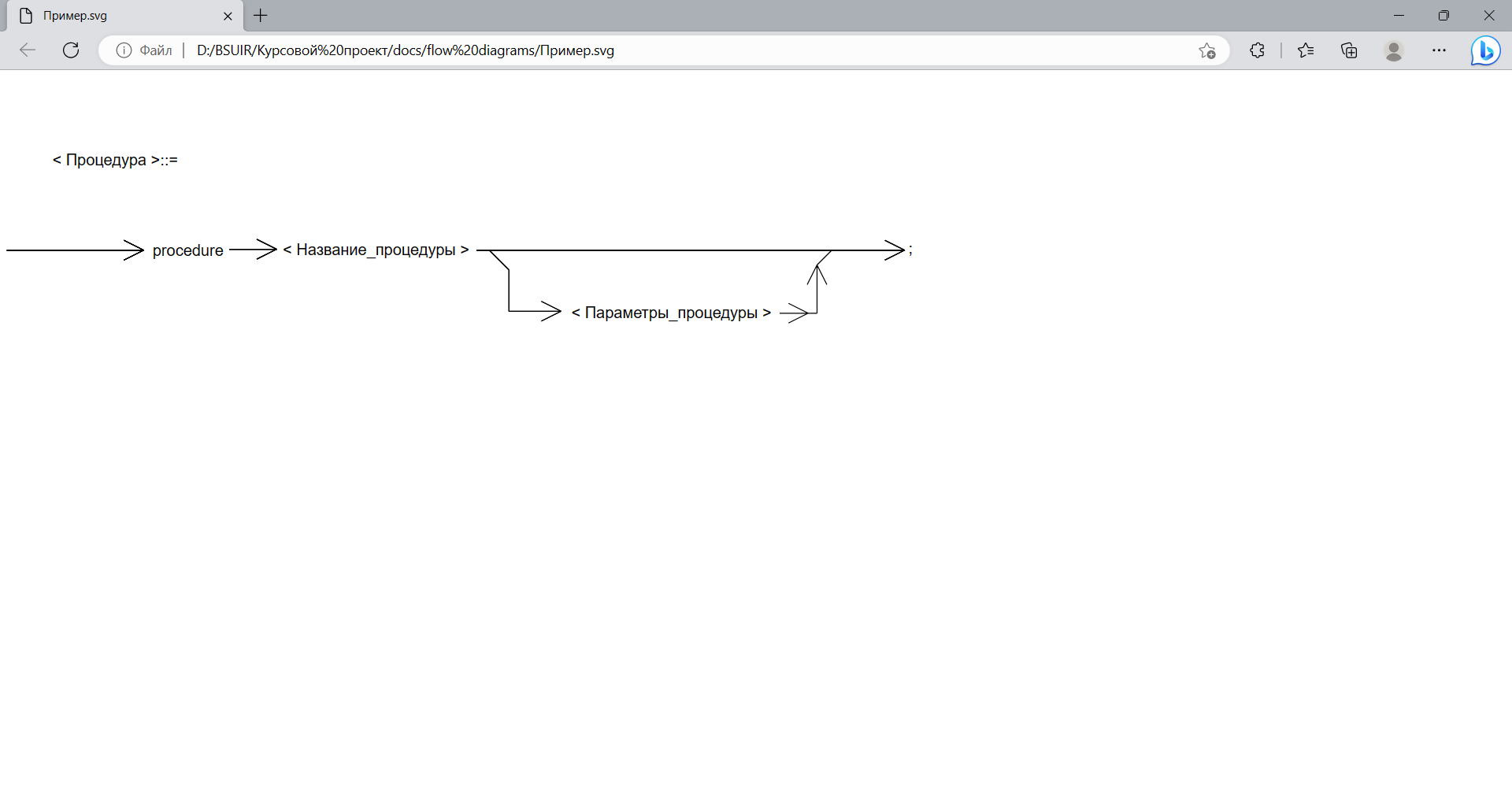


Рисунок 6.3 – Пример синтаксической диаграммы, экспортированной в SVG

# СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

[1] Глухова, Л. А, учебное пособие по дисциплине «Основы алгоритмизации и программирования» – Минск : БГУИР, 2006 г. – 195 с.

[2] Документация по SVG [Электронный ресурс] – Режим доступа:

<https://developer.mozilla.org/ru/docs/Web/SVG> – Дата доступа 15.05.2023

[3] Справочник по Delphi [Электронный ресурс] – Режим доступа:

<http://www.delphibasics.ru/> – Дата доступа 15.05.2023

# ЗАКЛЮЧЕНИЕ

В рамках курсового проектирования было разработано программное средство, обладающее необходимым и достаточным набором функций для построения синтаксических диаграмм с использованием векторной графики. Простой и интуитивно понятный пользовательский интерфейс позволяет пользователям быстро освоить данное программное средство и приступить к созданию синтаксических диаграмм.

Проведенное тестирование и отладка показали полную корректность работы программы.

В процессе разработки данного программного средства был использован язык программирования Delphi, средства создания векторного изображения в формате SVG.

Разработанное программное средство может быть использовано в учебных заведениях для обучения студентов особенностям синтаксиса нового языка программирования, а также разработчиками для создания синтаксических диаграмм в документации к разрабатываемому языку программирования.

# ПРИЛОЖЕНИЕ А

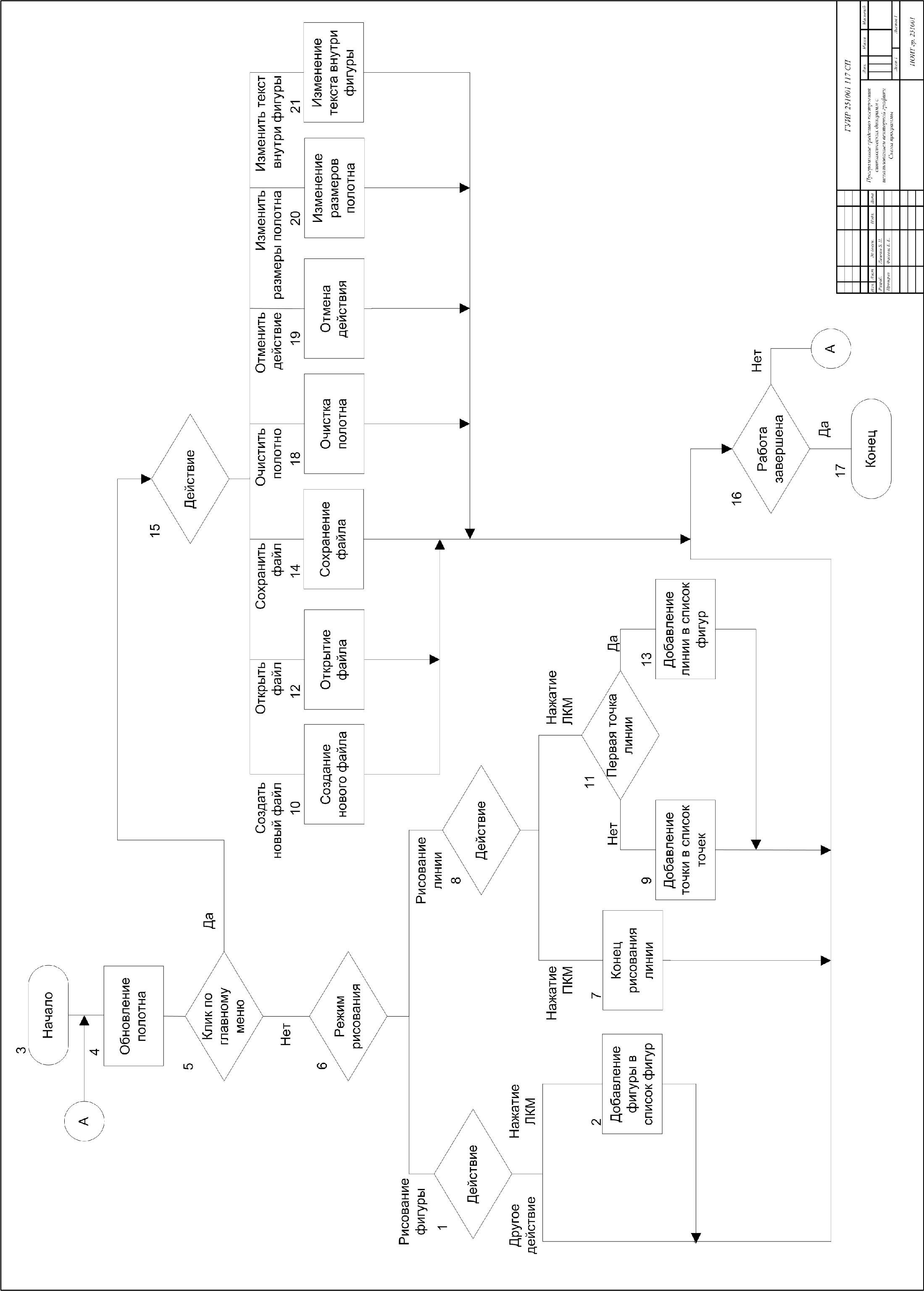


Рисунок А.1 – Схема программного средства

# ПРИЛОЖЕНИЕ Б

## Содержание модуля Main

unit Main;

interface

uses

Winapi.Windows, Winapi.Messages, System.SysUtils, System.UITypes,

System.Variants,

System.Classes, Vcl.Graphics, Vcl.Dialogs,

Vcl.Controls, Vcl.Forms, Vcl.ExtCtrls, Vcl.StdCtrls, Vcl.Menus,

Vcl.ToolWin, Vcl.ComCtrls, System.ImageList, Vcl.ImgList,

System.Actions, Vcl.ActnList, Data, Stack, Model, View;

type

TfrmMain = class(TForm)

MainMenuForm: TMainMenu;

mnFile: TMenuItem;

tbMenu: TToolBar;

MainImageList: TImageList;

MainPaintBox: TPaintBox;

MainActionList: TActionList;

MainStatusBar: TStatusBar;

actFileNew: TAction;

actFileOpen: TAction;

actFileSave: TAction;

actFileSaveAs: TAction;

actHeader: TAction;

actVariable: TAction;

actConst: TAction;

actLine: TAction;

actViewGridLines: TAction;

mnFileNew: TMenuItem;

mnFileOpen: TMenuItem;

mnFileSave: TMenuItem;

mnFileSaveAs: TMenuItem;

mnViewGridLines: TMenuItem;

tbnFileNew: TToolButton;

tbnFileOpen: TToolButton;

tbnFileSave: TToolButton;

tbnFileSaveAs: TToolButton;

tbnSeparator: TToolButton;

tbnHeader: TToolButton;

tbnVariable: TToolButton;

tbnConst: TToolButton;

tbnLine: TToolButton;

lbText: TLabel;

edtText: TEdit;

MainSaveDialog: TSaveDialog;

tbnClearCanvas: TToolButton;

actClearCanvas: TAction;

MainOpenDialog: TOpenDialog;

actUndo: TAction;

tbnUndo: TToolButton;

mnView: TMenuItem;

lbWidth: TLabel;

edtWidth: TEdit;

lbHeight: TLabel;

edtHeight: TEdit;

procedure FormCreate(Sender: TObject);

procedure FormDestroy(Sender: TObject);

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

procedure MainPaintBoxPaint(Sender: TObject);

procedure MainPaintBoxMouseDown(Sender: TObject; Button: TMouseButton;

Shift: TShiftState; X, Y: Integer);

procedure MainPaintBoxMouseMove(Sender: TObject; Shift: TShiftState;

X, Y: Integer);

procedure actFileNewExecute(Sender: TObject);

procedure actFileOpenExecute(Sender: TObject);

procedure actFileSaveExecute(Sender: TObject);

procedure actFileSaveAsExecute(Sender: TObject);

procedure actVariableExecute(Sender: TObject);

procedure actLineExecute(Sender: TObject);

procedure actConstExecute(Sender: TObject);

procedure actHeaderExecute(Sender: TObject);

procedure actViewGridLinesExecute(Sender: TObject);

procedure actClearCanvasExecute(Sender: TObject);

procedure actUndoExecute(Sender: TObject);

procedure edtWidthKeyDown(Sender: TObject; var Key: Word;

Shift: TShiftState);

procedure edtHeightKeyDown(Sender: TObject; var Key: Word;

Shift: TShiftState);

private

public

end;

var

frmMain: TfrmMain;

FigureHead, FigureCurr: PFigureList;

TypeFigure: TTypeFigure;

GridMode: TGridMode;

DrawMode: TDrawMode;

FileWasChanged: Boolean;

FirstPoint: Boolean;

implementation

uses

frmViewGridLines;

{$R \*.dfm}

procedure TfrmMain.actFileNewExecute(Sender: TObject);

var

Temp: Integer;

Path: String;

const

Text = 'Все несохраненные изменения будут утеряны. Продолжить?';

begin

Temp := MessageDlg(Text, mtWarning, [mbYes, mbNo], 0);

if Temp = mrYes then

begin

ClearFigureList(FigureHead);

FileWasChanged := false;

MainPaintBox.Repaint;

Path := SaveFile(fmDgr, MainSaveDialog);

if Path <> '' then

begin

FirstPoint := true;

SaveCanvasToDGR(Path, FigureHead);

end;

end;

end;

procedure TfrmMain.actFileOpenExecute(Sender: TObject);

var

Path: String;

begin

Path := OpenFile(fmDgr, MainOpenDialog);

if Path <> '' then

begin

ClearFigureList(FigureHead);

tbnUndo.Enabled := false;

OpenCanvasFromDGR(Path, FigureHead);

FileWasChanged := false;

MainPaintBox.Repaint;

end;

end;

procedure TfrmMain.actFileSaveAsExecute(Sender: TObject);

var

FileName: string;

TempGridMode: TGridMode;

begin

MainOpenDialog := TSaveDialog.Create(Self);

try

MainOpenDialog.Title := 'Сохранить файл как';

MainOpenDialog.Filter :=

'SVG файлы (\*.svg)|\*.svg|PNG файлы (\*.png)|\*.png|Все файлы (\*.\*)|\*.\*';

MainOpenDialog.DefaultExt := 'svg';

if MainOpenDialog.Execute then

begin

FileName := MainOpenDialog.FileName;

TempGridMode := GridMode;

GridMode := gmOff;

with MainPaintBox do

begin

Repaint;

case MainOpenDialog.FilterIndex of

1:

SaveCanvasToSVG(FileName, Width, Height, FigureHead);

2:

SaveCanvasToPNG(FileName, Canvas, Width, Height);

end;

end;

GridMode := TempGridMode;

FileWasChanged := false;

MainPaintBox.Repaint;

end;

finally

MainOpenDialog.Free;

end;

FileWasChanged := false;

end;

procedure TfrmMain.actFileSaveExecute(Sender: TObject);

var

Path: String;

begin

Path := SaveFile(fmDgr, MainSaveDialog);

if Path <> '' then

begin

SaveCanvasToDGR(Path, FigureHead);

FirstPoint := true;

end;

FileWasChanged := false;

end;

procedure TfrmMain.actClearCanvasExecute(Sender: TObject);

begin

ClearFigureList(FigureHead);

FirstPoint := true;

tbnClearCanvas.Enabled := true;

tbnUndo.Enabled := false;

MainPaintBox.Repaint;

end;

procedure TfrmMain.actConstExecute(Sender: TObject);

begin

FirstPoint := true;

TypeFigure := tfConst;

DrawMode := dmFigure;

tbnHeader.Down := false;

tbnVariable.Down := false;

tbnConst.Down := true;

tbnLine.Down := false;

end;

procedure TfrmMain.actHeaderExecute(Sender: TObject);

begin

FirstPoint := true;

TypeFigure := tfHeader;

DrawMode := dmFigure;

tbnHeader.Down := true;

tbnVariable.Down := false;

tbnConst.Down := false;

tbnLine.Down := false;

end;

procedure TfrmMain.actLineExecute(Sender: TObject);

begin

TypeFigure := tfLine;

DrawMode := dmLine;

tbnHeader.Down := false;

tbnVariable.Down := false;

tbnConst.Down := false;

tbnLine.Down := true;

end;

procedure TfrmMain.actUndoExecute(Sender: TObject);

var

FigureInfo: TFigureInfo;

begin

if not ListIsEmpty(FigureHead) then

PopFigureList(FigureHead, FigureInfo);

if ListIsEmpty(FigureHead) then

begin

tbnUndo.Enabled := false;

tbnClearCanvas.Enabled := false;

end;

MainPaintBox.Repaint;

end;

procedure TfrmMain.actVariableExecute(Sender: TObject);

begin

FirstPoint := true;

TypeFigure := tfVar;

DrawMode := dmFigure;

tbnHeader.Down := false;

tbnVariable.Down := true;

tbnConst.Down := false;

tbnLine.Down := false;

end;

procedure TfrmMain.actViewGridLinesExecute(Sender: TObject);

var

Res: Word;

begin

if GridMode = gmOff then

GridMode := gmOn

else

GridMode := gmOff;

if GridMode = gmOn then

begin

Res := frmGrid.ShowModal;

if Res = mrNo then

GridMode := gmOff

else

with MainPaintBox do

ClearCanvas(Canvas, Width, Height, GridMode, GridSize);

end;

MainPaintBox.Repaint;

end;

procedure TfrmMain.edtHeightKeyDown(Sender: TObject; var Key: Word;

Shift: TShiftState);

begin

if Key = VK\_RETURN then

begin

if StrToInt(edtHeight.Text) > 100 then

begin

frmMain.Constraints.MinHeight := StrToInt(edtHeight.Text) + 150;

frmMain.Height := StrToInt(edtHeight.Text) + 150;

end;

MainPaintBox.Repaint;

end;

end;

procedure TfrmMain.edtWidthKeyDown(Sender: TObject; var Key: Word;

Shift: TShiftState);

begin

if Key = VK\_RETURN then

begin

if StrToInt(edtWidth.Text) > 1000 then

begin

frmMain.Constraints.MinWidth := StrToInt(edtWidth.Text) + 30;

frmMain.Width := StrToInt(edtWidth.Text) + 30;

end;

MainPaintBox.Repaint;

end;

end;

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

var

Temp: Integer;

Path: String;

const

Text = 'Файл был изменен. Сохранить изменения?';

begin

if FileWasChanged then

begin

Temp := MessageDlg(Text, mtWarning, [mbYes, mbNo, mbCancel], 0);

case Temp of

mrYes:

begin

Path := SaveFile(fmPng, MainSaveDialog);

if Path <> '' then

begin

with MainPaintBox do

SaveCanvasToPNG(Path, Canvas, Width, Height);

FileWasChanged := false;

Action := caFree;

end

else

Action := caNone;

end;

mrNo:

Action := caFree;

mrCancel:

Action := caNone;

end;

end;

end;

procedure TfrmMain.FormCreate(Sender: TObject);

begin

GridMode := gmOff;

DrawMode := dmNone;

CreateFigureList(FigureHead);

FileWasChanged := false;

TypeFigure := tfNothing;

FirstPoint := true;

GridSize := 20;

tbnUndo.Enabled := false;

tbnClearCanvas.Enabled := false;

MainPaintBox.Width := 1000;

MainPaintBox.Height := 500;

end;

procedure TfrmMain.FormDestroy(Sender: TObject);

begin

ClearFigureList(FigureHead);

Dispose(FigureHead);

end;

procedure TfrmMain.MainPaintBoxMouseDown(Sender: TObject; Button: TMouseButton;

Shift: TShiftState; X, Y: Integer);

begin

if DrawMode = dmFigure then

case Button of

TMouseButton.mbLeft:

begin

FigureCurr := AddNewFigureToList(MainPaintBox.Canvas, FigureHead, X,

Y, TypeFigure, edtText.Text);

tbnUndo.Enabled := true;

tbnClearCanvas.Enabled := true;

FileWasChanged := true;

end;

end;

if DrawMode = dmLine then

case Button of

TMouseButton.mbLeft:

begin

if not FirstPoint then

begin

AddNewPointToList(FigureCurr^.Info.PointHead, X, Y);

tbnUndo.Enabled := true;

tbnClearCanvas.Enabled := true;

FileWasChanged := true;

end

else

begin

FigureCurr := AddNewLineToList(FigureHead, X, Y);

tbnUndo.Enabled := true;

tbnClearCanvas.Enabled := true;

FirstPoint := false;

FileWasChanged := true;

end;

end;

TMouseButton.mbRight:

FirstPoint := true;

end;

MainPaintBox.Repaint;

end;

procedure TfrmMain.MainPaintBoxMouseMove(Sender: TObject; Shift: TShiftState;

X, Y: Integer);

begin

if (TypeFigure = tfLine) and not FirstPoint then

begin

MainPaintBox.Repaint;

Streak(MainPaintBox.Canvas, FigureCurr^.Info.PointHead, X, Y);

end;

end;

procedure TfrmMain.MainPaintBoxPaint(Sender: TObject);

begin

with MainPaintBox do

ClearCanvas(Canvas, Width, Height, GridMode, GridSize);

DrawFigure(MainPaintBox.Canvas, FigureHead);

MainPaintBox.Width := StrToInt(edtWidth.Text);

MainPaintBox.Height := StrToInt(edtHeight.Text);

end;

end.

## Содержание модуля Model

unit Model;

interface

uses

Vcl.Graphics, System.SysUtils, Data, Stack;

function AddNewFigureToList(Canvas: TCanvas; FigureHead: PFigureList;

const X, Y: Integer; const FType: TTypeFigure; const FText: String)

: PFigureList;

procedure DrawFigure(Canvas: TCanvas; FigureHead: PFigureList);

function AddNewLineToList(var FigureHead: PFigureList; X, Y: Integer)

: PFigureList;

procedure AddNewPointToList(PointsHead: PPointsList; X, Y: Integer);

procedure DrawLine(Canvas: TCanvas; PointHead: PPointsList);

procedure GorizArrow(Canvas: TCanvas; X, Y: Integer; C: Integer);

procedure VertArrow(Canvas: TCanvas; X, Y: Integer; C: Integer);

implementation

// ПОДПРОГРАММЫ ДЛЯ РИСОВАНИЯ ФИГУР

procedure DrawFigure(Canvas: TCanvas; FigureHead: PFigureList);

var

Temp: PFigureList;

begin

Canvas.Font.Name := 'Arial';

Canvas.Font.Size := 10;

Canvas.Font.Color := clBlack;

Temp := FigureHead;

while Temp^.Adr <> nil do

begin

Temp := Temp^.Adr;

with Temp^.Info do

begin

if TypeFigure <> tfLine then

begin

Canvas.Pen.Color := clWhite;

Canvas.Brush.Color := clWhite;

Canvas.Rectangle(x1, y1, x2, y2);

Canvas.TextOut(x1, y1, String(Text));

Canvas.Pen.Color := clBlack;

end

else

DrawLine(Canvas, Temp^.Info.PointHead);

end;

end;

end;

function AddNewFigureToList(Canvas: TCanvas; FigureHead: PFigureList;

const X, Y: Integer; const FType: TTypeFigure; const FText: String)

: PFigureList;

var

Temp: PFigureList;

TextWidth, TextHeight: Integer;

begin

Temp := FigureHead;

while Temp^.Adr <> nil do

Temp := Temp^.Adr;

New(Temp^.Adr);

Temp := Temp^.Adr;

Temp^.Adr := nil;

TextWidth := 0;

TextHeight := 0;

with Temp^.Info do

begin

if (FType <> tfLine) and (FType <> tfNothing) then

begin

case FType of

tfHeader:

Text := '< ' + ShortString(Trim(FText)) + ' >::=';

tfVar:

Text := '< ' + ShortString(Trim(FText)) + ' >';

tfConst:

Text := ShortString(Trim(FText));

end;

TextWidth := Canvas.TextWidth(String(Text));

TextHeight := Canvas.TextHeight(String(Text));

end;

x1 := X;

x2 := X + TextWidth;

y1 := Y;

y2 := Y + TextHeight;

TypeFigure := FType;

end;

Result := Temp;

end;

// ПОДПРОГРАММЫ ДЛЯ РИСОВАНИЯ ЛИНИЙ

function AddNewLineToList(var FigureHead: PFigureList; X, Y: Integer)

: PFigureList;

var

Temp: PFigureList;

begin

Temp := FigureHead;

while Temp^.Adr <> nil do

Temp := Temp^.Adr;

New(Temp^.Adr);

Temp := Temp^.Adr;

Temp^.Adr := nil;

Temp^.Info.TypeFigure := tfLine;

New(Temp^.Info.PointHead);

Temp^.Info.PointHead^.Adr := nil;

Temp^.Info.PointHead^.Info.X := X;

Temp^.Info.PointHead^.Info.Y := Y;

Result := Temp;

end;

procedure DrawLine(Canvas: TCanvas; PointHead: PPointsList);

var

Temp: PPointsList;

First, Curr: TPointsInfo;

DistX, DistY, Slice: Integer;

EndOfLine: Boolean;

begin

Canvas.Pen.Width := 1;

Temp := PointHead;

if Temp^.Adr <> nil then

begin

First.X := Temp^.Info.X;

First.Y := Temp^.Info.Y;

Canvas.MoveTo(Temp^.Info.X, Temp^.Info.Y);

Curr.X := First.X;

Curr.Y := First.Y;

// Если первая линия - вертикальная, то рисовать косой срез

if (Temp^.Adr <> nil) and (First.X = Temp^.Adr^.Info.X) and

(First.Y <> Temp^.Adr^.Info.Y) and (Curr.Y = Temp^.Info.Y) then

begin

if First.Y - Temp^.Adr^.Info.Y < 0 then

Slice := 1

else

Slice := -1;

Canvas.MoveTo(Temp^.Info.X - 20, Temp^.Info.Y);

Canvas.LineTo(Temp^.Info.X, Temp^.Info.Y + Slice \* 20);

Canvas.MoveTo(Temp^.Info.X, Temp^.Info.Y + Slice \* 20);

end;

EndOfLine := False;

while Temp^.Adr <> nil do

begin

Temp := Temp^.Adr;

if EndOfLine then

begin

// Если конечная линия вертикальная, то ставить стрелку на конце и срез

Canvas.LineTo(Temp^.Info.X, Temp^.Info.Y + Slice \* 15);

Canvas.MoveTo(Temp^.Info.X, Temp^.Info.Y + Slice \* 15);

Canvas.LineTo(Temp^.Info.X + 15, Temp^.Info.Y);

VertArrow(Canvas, Temp^.Info.X, Temp^.Info.Y + Slice \* 15, Slice);

end

else

begin

Canvas.LineTo(Temp^.Info.X, Temp^.Info.Y);

Canvas.MoveTo(Temp^.Info.X, Temp^.Info.Y);

// Рисует стрелку на конце линии

if Temp^.Adr = nil then

begin

DistX := Temp^.Info.X - Curr.X;

if DistX > 0 then

GorizArrow(Canvas, Temp^.Info.X, Temp^.Info.Y, 1)

else if DistX < 0 then

GorizArrow(Canvas, Temp^.Info.X, Temp^.Info.Y, -1);

DistY := Temp^.Info.Y - Curr.Y;

if DistY > 0 then

VertArrow(Canvas, Temp^.Info.X, Temp^.Info.Y, -1)

else if DistY < 0 then

VertArrow(Canvas, Temp^.Info.X, Temp^.Info.Y, 1);

end;

// Рисовать стрелку на середине линии

if (Temp^.Adr <> nil) and (Temp^.Info.X <> First.X) and

(Temp^.Info.X = Temp^.Adr^.Info.X) and

(Temp^.Info.Y <> Temp^.Adr^.Info.Y) and (Curr.X <> Temp^.Info.X) then

begin

if Temp^.Adr^.Adr = nil then

EndOfLine := True;

DistX := Temp^.Info.X - Curr.X;

if Abs(DistX) > 20 then

begin

if DistX > 0 then

DistX := 1

else

DistX := -1;

GorizArrow(Canvas, Temp^.Info.X + 10 \* DistX -

(Temp^.Info.X - Curr.X) div 2, Temp^.Info.Y, DistX);

Canvas.MoveTo(Temp^.Info.X, Temp^.Info.Y);

end;

// Изменить направление конечного среза

if Temp^.Info.Y - Temp^.Adr^.Info.Y < 0 then

Slice := -1

else

Slice := 1;

end

else

EndOfLine := False;

Curr.X := Temp^.Info.X;

Curr.Y := Temp^.Info.Y;

end;

end;

end;

Canvas.Pen.Width := 1;

end;

procedure AddNewPointToList(PointsHead: PPointsList; X, Y: Integer);

var

TTemp, Temp: PPointsList;

begin

Temp := PointsHead;

while Temp^.Adr <> nil do

Temp := Temp^.Adr;

TTemp := Temp;

New(Temp^.Adr);

Temp := Temp^.Adr;

if Abs(TTemp^.Info.X - X) > Abs(TTemp^.Info.Y - Y) then

begin

Temp^.Info.X := X;

Temp^.Info.Y := TTemp^.Info.Y;

end

else

begin

Temp^.Info.X := TTemp^.Info.X;

Temp^.Info.Y := Y;

end;

Temp^.Adr := nil;

end;

procedure GorizArrow(Canvas: TCanvas; X, Y: Integer; C: Integer);

begin

Canvas.MoveTo(X, Y);

Canvas.LineTo(X - 20 \* C, Y - 10);

Canvas.MoveTo(X, Y);

Canvas.LineTo(X - 20 \* C, Y + 10);

Canvas.MoveTo(X, Y);

end;

procedure VertArrow(Canvas: TCanvas; X, Y: Integer; C: Integer);

begin

Canvas.MoveTo(X, Y);

Canvas.LineTo(X - 10, Y + 20 \* C);

Canvas.MoveTo(X, Y);

Canvas.LineTo(X + 10, Y + 20 \* C);

Canvas.MoveTo(X, Y);

end;

end.

## Содержание модуля View

unit View;

interface

uses

Vcl.Graphics, Vcl.Dialogs, System.SysUtils, System.StrUtils, System.Types,

System.UITypes,

PngImage, Data;

procedure ClearCanvas(Canvas: TCanvas; const Width, Height: Integer;

GridMode: TGridMode; const GridSize: Integer);

procedure Streak(Canvas: TCanvas; PointHead: PPointsList; X, Y: Integer);

function SaveFile(FileMode: TFileMode; MainSaveDialog: TSaveDialog): String;

function OpenFile(FileMode: TFileMode; MainOpenDialog: TOpenDialog): String;

procedure OpenCanvasFromDGR(const FileName: string; FigureHead: PFigureList);

procedure SaveCanvasToDGR(const FileName: string; FigureHead: PFigureList);

procedure SaveCanvasToPNG(const FileName: string; Canvas: TCanvas;

CanvasWidth, CanvasHeight: Integer);

procedure SaveCanvasToSVG(const FileName: string; Width, Height: Integer;

FigureHead: PFigureList);

function ListToStr(fPoints: PPointsList): String;

function ReplaceChar(Text: UTF8String): UTF8String;

// ПОДПРОГРАММЫ ДЛЯ РИСОВАНИЯ В SVG ФАЙЛЕ

procedure SVGMoveTo(var List: UTF8String; X, Y: Integer);

procedure SVGLineTo(var List: UTF8String; X, Y: Integer);

procedure SVGGorizArrow(var List: UTF8String; X, Y: Integer; C: Integer);

procedure SVGVertArrow(var List: UTF8String; X, Y: Integer; C: Integer);

implementation

procedure ClearCanvas(Canvas: TCanvas; const Width, Height: Integer;

GridMode: TGridMode; const GridSize: Integer);

var

TempColor, TempPen: TColor;

i: Integer;

begin

if GridMode = gmOff then

with Canvas do

begin

TempPen := Pen.Color;

Pen.Color := clWhite;

TempColor := Brush.Color;

Brush.Color := clWhite;

Rectangle(0, 0, Width, Height);

Pen.Color := TempPen;

Brush.Color := TempColor;

end

else

with Canvas do

begin

TempColor := Canvas.Brush.Color;

Brush.Color := clWhite;

Rectangle(0, 0, Width, Height);

Brush.Color := TempColor;

TempColor := Pen.Color;

Pen.Color := $9E9E9E;

for i := 0 to Width div GridSize do

begin

MoveTo(i \* GridSize, 0);

LineTo(i \* GridSize, Height);

end;

for i := 0 to Height div GridSize do

begin

MoveTo(0, i \* GridSize);

LineTo(Width, i \* GridSize);

end;

Pen.Color := TempColor;

end;

end;

procedure Streak(Canvas: TCanvas; PointHead: PPointsList; X, Y: Integer);

var

Temp: PPointsList;

TempStyle: TPenStyle;

TempColor: TColor;

begin

Temp := PointHead;

while Temp^.Adr <> nil do

Temp := Temp^.Adr;

if abs(X - Temp^.Info.X) > abs(Y - Temp^.Info.Y) then

Y := Temp^.Info.Y

else

X := Temp^.Info.X;

TempStyle := Canvas.Pen.Style;

Canvas.Pen.Style := psDashDot;

TempColor := Canvas.Pen.Color;

Canvas.Pen.Color := clBlue;

Canvas.MoveTo(Temp^.Info.X, Temp^.Info.Y);

Canvas.LineTo(X, Y);

Canvas.Pen.Color := TempColor;

Canvas.Pen.Style := TempStyle;

end;

procedure SaveCanvasToDGR(const FileName: string; FigureHead: PFigureList);

var

FigureFile: file of TFigureInFile;

fRec: TFigureInFile;

fPoints: PPointsList;

fTemp: PFigureList;

begin

AssignFile(FigureFile, FileName);

ReWrite(FigureFile);

fTemp := FigureHead;

while fTemp^.Adr <> nil do

begin

fTemp := fTemp^.Adr;

fRec.TypeFigure := fTemp^.Info.TypeFigure;

if fRec.TypeFigure = tfLine then

begin

fPoints := fTemp^.Info.PointHead;

fRec.Point := ShortString(ListToStr(fPoints));

end

else

begin

fRec.Text := fTemp^.Info.Text;

fRec.x1 := fTemp^.Info.x1;

fRec.y1 := fTemp^.Info.y1;

fRec.x2 := fTemp^.Info.x2;

fRec.y2 := fTemp^.Info.y2;

end;

Write(FigureFile, fRec);

end;

CloseFile(FigureFile);

end;

procedure SaveCanvasToPNG(const FileName: string; Canvas: TCanvas;

CanvasWidth, CanvasHeight: Integer);

var

Bitmap: TBitmap;

Png: TPngImage;

begin

Bitmap := TBitmap.Create;

try

Bitmap.Width := CanvasWidth;

Bitmap.Height := CanvasHeight;

Bitmap.Canvas.CopyRect(Rect(0, 0, CanvasWidth, CanvasHeight), Canvas,

Rect(0, 0, CanvasWidth, CanvasHeight));

Png := TPngImage.Create;

try

Png.Assign(Bitmap);

Png.SaveToFile(FileName);

finally

Png.Free;

end;

finally

Bitmap.Free;

end;

end;

procedure SaveCanvasToSVG(const FileName: string; Width, Height: Integer;

FigureHead: PFigureList);

var

SVGFile: TextFile;

fTemp: PFigureList;

pTemp: PPointsList;

First, Curr: TPointsInfo;

Text, Points: UTF8String;

DistX, DistY, Slice: Integer;

EndOfLine: Boolean;

begin

AssignFile(SVGFile, FileName, CP\_UTF8);

ReWrite(SVGFile);

// открытие svg контейнера

WriteLn(SVGFile, '<?xml version="1.0" encoding="UTF-8"?>');

WriteLn(SVGFile, '<svg xmlns="http://www.w3.org/2000/svg" width="', Width,

'" height="', Height, '" viewBox="0 0 ', Width, ' ', Height, '">');

Points := '';

fTemp := FigureHead;

while fTemp^.Adr <> nil do

begin

fTemp := fTemp^.Adr;

// Алгоритм рисования линий в SVG файле такой же, как и на Canvas

// только методы Canvas.MoveTo и Canvas.LineTo заменены на

// SVGMoveTo и SVGLineTo соответственно

if fTemp^.Info.TypeFigure = tfLine then

begin

pTemp := fTemp^.Info.PointHead;

if pTemp^.Adr <> nil then

begin

First.X := pTemp^.Info.X;

First.Y := pTemp^.Info.Y;

SVGMoveTo(Points, pTemp^.Info.X, pTemp^.Info.Y);

Curr.X := First.X;

Curr.Y := First.Y;

// Если первая линия - вертикальная, то рисовать косой срез

if (pTemp^.Adr <> nil) and (First.X = pTemp^.Adr^.Info.X) and

(First.Y <> pTemp^.Adr^.Info.Y) and (Curr.Y = pTemp^.Info.Y) then

begin

if First.Y - pTemp^.Adr^.Info.Y < 0 then

Slice := 1

else

Slice := -1;

SVGMoveTo(Points, pTemp^.Info.X - 20, pTemp^.Info.Y);

SVGLineTo(Points, pTemp^.Info.X, pTemp^.Info.Y + Slice \* 20);

SVGMoveTo(Points, pTemp^.Info.X, pTemp^.Info.Y + Slice \* 20);

end;

EndOfLine := False;

while pTemp^.Adr <> nil do

begin

pTemp := pTemp^.Adr;

if EndOfLine then

begin

// Если конечная линия вертикальная, то ставить стрелку на конце и срез

SVGLineTo(Points, pTemp^.Info.X, pTemp^.Info.Y + Slice \* 15);

SVGMoveTo(Points, pTemp^.Info.X, pTemp^.Info.Y + Slice \* 15);

SVGLineTo(Points, pTemp^.Info.X + 15, pTemp^.Info.Y);

SVGVertArrow(Points, pTemp^.Info.X, pTemp^.Info.Y + Slice \*

15, Slice);

end

else

begin

SVGLineTo(Points, pTemp^.Info.X, pTemp^.Info.Y);

SVGMoveTo(Points, pTemp^.Info.X, pTemp^.Info.Y);

// Рисует стрелку на конце линии

if pTemp^.Adr = nil then

begin

DistX := pTemp^.Info.X - Curr.X;

if DistX > 0 then

SVGGorizArrow(Points, pTemp^.Info.X, pTemp^.Info.Y, 1)

else if DistX < 0 then

SVGGorizArrow(Points, pTemp^.Info.X, pTemp^.Info.Y, -1);

DistY := pTemp^.Info.Y - Curr.Y;

if DistY > 0 then

SVGVertArrow(Points, pTemp^.Info.X, pTemp^.Info.Y, -1)

else if DistY < 0 then

SVGVertArrow(Points, pTemp^.Info.X, pTemp^.Info.Y, 1);

end;

if (pTemp^.Adr <> nil) and (pTemp^.Info.X <> First.X) and

(pTemp^.Info.X = pTemp^.Adr^.Info.X) and

(pTemp^.Info.Y <> pTemp^.Adr^.Info.Y) and (Curr.X <> pTemp^.Info.X)

then

begin

if pTemp^.Adr^.Adr = nil then

EndOfLine := True;

DistX := pTemp^.Info.X - Curr.X;

if abs(DistX) > 20 then

begin

if DistX > 0 then

DistX := 1

else

DistX := -1;

SVGGorizArrow(Points, pTemp^.Info.X + 10 \* DistX -

(pTemp^.Info.X - Curr.X) div 2, pTemp^.Info.Y, DistX);

SVGMoveTo(Points, pTemp^.Info.X, pTemp^.Info.Y);

end;

if pTemp^.Info.Y - pTemp^.Adr^.Info.Y < 0 then

Slice := -1

else

Slice := 1;

end

else

EndOfLine := False;

Curr.X := pTemp^.Info.X;

Curr.Y := pTemp^.Info.Y;

end;

end;

end;

// вся информация о точках заносится в path контейнер

WriteLn(SVGFile, '<path d="', Points,

' Z" fill="none" stroke="black" stroke-width="1" />');

end

else

begin

// открытие text контейнера

Text := ReplaceChar(AnsiToUTF8(String(fTemp^.Info.Text)));

WriteLn(SVGFile, '<text x="', fTemp^.Info.x1 + 2 \* 34, '" y="',

fTemp^.Info.y1 + 9,

'" text-anchor="middle" alignment-baseline="middle" fill="black" font-family="Arial" font-size="16"> ',

Text, ' </text>');

end;

end;

// закрытие svg контейнера

WriteLn(SVGFile, '</svg>');

CloseFile(SVGFile);

end;

function SaveFile(FileMode: TFileMode; MainSaveDialog: TSaveDialog): String;

begin

Result := '';

if FileMode = fmDgr then

begin

MainSaveDialog.FileName := 'FileName.dgr';

MainSaveDialog.Filter := 'DGR files (\*.dgr)|\*.dgr|All files (\*.\*)|\*.\*';

MainSaveDialog.DefaultExt := 'dgr';

end;

if MainSaveDialog.Execute then

begin

Result := MainSaveDialog.FileName;

end;

end;

function OpenFile(FileMode: TFileMode; MainOpenDialog: TOpenDialog): String;

begin

Result := '';

if FileMode = fmDgr then

begin

MainOpenDialog.FileName := 'FileName.dgr';

MainOpenDialog.Filter := 'DGR files (\*.dgr)|\*.dgr|All files (\*.\*)|\*.\*';

MainOpenDialog.DefaultExt := 'dgr';

end;

if MainOpenDialog.Execute then

begin

Result := MainOpenDialog.FileName;

end;

end;

procedure OpenCanvasFromDGR(const FileName: string; FigureHead: PFigureList);

var

FigureFile: file of TFigureInFile;

fTemp: PFigureList;

pTemp: PPointsList;

fInfo: TFigureInFile;

CoordStr: String;

begin

AssignFile(FigureFile, FileName);

Reset(FigureFile);

fTemp := FigureHead;

while not EOF(FigureFile) do

begin

New(fTemp^.Adr);

fTemp := fTemp^.Adr;

fTemp^.Adr := nil;

Read(FigureFile, fInfo);

fTemp^.Info.TypeFigure := fInfo.TypeFigure;

if fInfo.TypeFigure = tfLine then

begin

New(fTemp^.Info.PointHead);

pTemp := fTemp^.Info.PointHead;

pTemp^.Adr := nil;

if fInfo.Point <> '' then

begin

Delete(fInfo.Point, 1, 1);

fInfo.Point := fInfo.Point + '"';

end;

CoordStr := String(Copy(fInfo.Point, 1,

pos('"', String(fInfo.Point)) - 1));

Delete(fInfo.Point, 1, pos('"', String(fInfo.Point)) + 1);

if (fInfo.Point <> '') or (CoordStr <> '') then

begin

pTemp^.Info.X := StrToInt(Copy(CoordStr, 1, pos(',', CoordStr) - 1));

pTemp^.Info.Y := StrToInt(Copy(CoordStr, pos(',', CoordStr) + 1,

Length(CoordStr)));

pTemp^.Adr := nil;

end;

while (Length(fInfo.Point) <> 0) and (fInfo.Point <> '"') do

begin

CoordStr := String(Copy(fInfo.Point, 1,

pos('"', String(fInfo.Point)) - 1));

Delete(fInfo.Point, 1, pos('"', String(fInfo.Point)) + 1);

if (fInfo.Point <> '') or (CoordStr <> '') then

begin

New(pTemp^.Adr);

pTemp := pTemp^.Adr;

pTemp^.Adr := nil;

pTemp^.Info.X := StrToInt(Copy(CoordStr, 1, pos(',', CoordStr) - 1));

pTemp^.Info.Y := StrToInt(Copy(CoordStr, pos(',', CoordStr) + 1,

Length(CoordStr)));

end;

end;

end

else

with fTemp^.Info do

begin

Text := fInfo.Text;

x1 := fInfo.x1;

y1 := fInfo.y1;

x2 := fInfo.x2;

y2 := fInfo.y2;

end;

end;

Close(FigureFile);

end;

function ListToStr(fPoints: PPointsList): String;

begin

Result := '';

Result := Result + String('"' + IntToStr(fPoints^.Info.X) + ',' +

IntToStr(fPoints^.Info.Y) + '"');

while fPoints^.Adr <> nil do

begin

fPoints := fPoints^.Adr;

Result := Result + '"' + IntToStr(fPoints^.Info.X) + ',' +

IntToStr(fPoints^.Info.Y) + '"';

end;

end;

function ReplaceChar(Text: UTF8String): UTF8String;

begin

Result := UTF8Encode(StringReplace(UTF8ToString(Text), '<', '&lt;',

[rfReplaceAll]));

end;

procedure SVGMoveTo(var List: UTF8String; X, Y: Integer);

begin

List := List + 'M ' + AnsiToUTF8(IntToStr(X)) + ' ' +

AnsiToUTF8(IntToStr(Y)) + ' ';

end;

procedure SVGLineTo(var List: UTF8String; X, Y: Integer);

begin

List := List + 'L ' + AnsiToUTF8(IntToStr(X)) + ' ' +

AnsiToUTF8(IntToStr(Y)) + ' ';

end;

procedure SVGGorizArrow(var List: UTF8String; X, Y: Integer; C: Integer);

begin

SVGMoveTo(List, X, Y);

SVGLineTo(List, X - 20 \* C, Y - 10);

SVGMoveTo(List, X, Y);

SVGLineTo(List, X - 20 \* C, Y + 10);

SVGMoveTo(List, X, Y);

end;

procedure SVGVertArrow(var List: UTF8String; X, Y: Integer; C: Integer);

begin

SVGMoveTo(List, X, Y);

SVGLineTo(List, X - 10, Y + 20 \* C);

SVGMoveTo(List, X, Y);

SVGLineTo(List, X + 10, Y + 20 \* C);

SVGMoveTo(List, X, Y);

end;

end.

## Содержание модуля Stack

unit Stack;

interface

uses

Data, Vcl.Graphics;

procedure CreateFigureList(var HeadList: PFigureList);

procedure ClearFigureList(FigureHead: PFigureList);

function PopFigureList(FigureHead: PFigureList; var Info: TFigureInfo): boolean;

function ListIsEmpty(Element: PFigureList): boolean;

function GetFigureHead(FigureHead: PFigureList): PFigureList;

implementation

uses

Main;

procedure CreateFigureList(var HeadList: PFigureList);

begin

New(HeadList);

HeadList^.Info.TypeFigure := tfNothing;

HeadList^.Adr := nil;

end;

procedure ClearFigureList(FigureHead: PFigureList);

var

Temp: PFigureList;

begin

while FigureHead^.Adr <> nil do

begin

Temp := FigureHead^.Adr;

FigureHead^.Adr := Temp^.Adr;

Dispose(Temp);

end;

FigureHead^.Info.TypeFigure := tfNothing;

FigureHead^.Adr := nil;

end;

function PopFigureList(FigureHead: PFigureList; var Info: TFigureInfo): boolean;

var

Temp, TTemp: PFigureList;

begin

Temp := FigureHead;

TTemp := FigureHead;

Result := False;

while Temp^.Adr <> nil do

begin

TTemp := Temp;

Temp := Temp^.Adr;

end;

if Temp <> FigureHead then

begin

Info := Temp^.Info;

TTemp^.Adr := nil;

Dispose(Temp);

Result := true;

end;

end;

function ListIsEmpty(Element: PFigureList): boolean;

begin

Result := (Element^.Info.TypeFigure = tfNothing) and (Element^.Adr = nil);

end;

function GetFigureHead(FigureHead: PFigureList): PFigureList;

begin

Result := FigureHead;

end;

end.

## Содержание модуля Data

unit Data;

interface

type

TDrawMode = (dmLine, dmFigure, dmNone);

TTypeFigure = (tfLine, tfVar, tfConst, tfHeader, tfNothing);

TFileMode = (fmSvg, fmPng, fmDgr);

TGridMode = (gmOn, gmOff);

TPointsInfo = record

x, y: integer;

end;

PPointsList = ^TPointsList;

TPointsList = record

Info: TPointsInfo;

Adr: PPointsList;

end;

TFigureInfo = record

case TypeFigure: TTypeFigure of

tfVar, tfConst, tfHeader:

(x1, y1, x2, y2: integer; Text: String[100]);

tfLine:

(PointHead: PPointsList);

tfNothing:

();

end;

TFigureInFile = record

case TypeFigure: TTypeFigure of

tfVar, tfConst, tfHeader:

(x1, y1, x2, y2: integer; Text: String[100]);

tfLine:

(Point: String[200]);

tfNothing:

();

end;

PFigureList = ^FigureList;

FigureList = record

Info: TFigureInfo;

Adr: PFigureList;

end;

var

GridSize: integer;

implementation

end.

## Содержание модуля frmViewGridLines

unit frmViewGridLines;

interface

uses

Winapi.Windows, Winapi.Messages, System.SysUtils, System.Variants,

System.Classes, Vcl.Graphics,

Vcl.Controls, Vcl.Forms, Vcl.Dialogs, Vcl.StdCtrls, Data;

type

TfrmGrid = class(TForm)

lbQuestion: TLabel;

edtNumber: TEdit;

btnOk: TButton;

btnCancel: TButton;

procedure btnOkClick(Sender: TObject);

procedure btnCancelClick(Sender: TObject);

procedure FormKeyDown(Sender: TObject; var Key: Word; Shift: TShiftState);

private

public

end;

var

frmGrid: TfrmGrid;

implementation

{$R \*.dfm}

procedure TfrmGrid.btnCancelClick(Sender: TObject);

begin

ModalResult := mrNo;

end;

procedure TfrmGrid.btnOkClick(Sender: TObject);

begin

GridSize := StrToInt(edtNumber.Text);

ModalResult := mrOk;

end;

procedure TfrmGrid.FormKeyDown(Sender: TObject; var Key: Word;

Shift: TShiftState);

begin

if Key = VK\_RETURN then

begin

GridSize := StrToInt(edtNumber.Text);

ModalResult := mrOk;

end;

if Key = VK\_ESCAPE then

begin

ModalResult := mrNo;

end;

end;

end.