Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

Дисциплина: Основы алгоритмизации и программирования

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе

на тему

ПРОГРАММНОЕ СРЕДСТВО ПОСТРОЕНИЯ СИНТАКСИЧЕСКИХ ДИАГРАММ С ИСПОЛЬЗОВАНИЕМ ВЕКТОРНОЙ ГРАФИКИ

БГУИР КР 1-40 01 01 117 ПЗ

Студент: гр. 251001 Лашкин В.Н.

Руководитель:

асс. Фадеева Е.Е.

Минск 2023

Учреждение образования

«Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

УТВЕРЖДАЮ

Заведующий кафедрой ПОИТ

––––––––––––––––––––––––

(подпись)

––––––––––––––––– 2023 г.

ЗАДАНИЕ

по курсовому проектированию

Студенту  Лашкину Владиславу Николаевичу

1. Тема работы “Программное средство построения синтаксических диаграмм с использованием векторной графики”

2. Срок сдачи студентом законченной работы 31.05.2023 г.–––

3. Исходные данные к работе

- среда разработки Delphi

- входные файлы (файлы с графическими примитивами, справка пользователя)

- выходные файлы (файл с синтаксической диаграммой в векторном или растровом формате)

4. Содержание расчётно-пояснительной записки (перечень вопросов, которые подлежат разработке)

Введение.

1. Анализ прототипов, литературных источников и формирование требований к проектируемому программному средству;

2. Анализ требований к программному средству и разработка функциональных требований;

3. Проектирование программного средства;

4. Создание (конструирование) программного средства;

5. Тестирование, проверка работоспособности и анализ полученных результатов;

6. Руководство по установке и использованию;

Список используемой литературы

Заключение

5. Перечень графического материала (с точным обозначением обязательных чертежей и графиков)

1. "Редактор синтаксических диаграмм", А1, схема программы, чертеж

6. Консультант по курсовой работе

Фадеева Е.Е.

7. Дата выдачи задания 21.02.2023

8. Календарный график работы над курсовой работой на весь период проектирования (с обозначением сроков выполнения и процентом от общего объёма работы):

раздел 1 к 01.03.2023 – 15 % готовности работы;

разделы 2, 3 к 15.03.2023 – 30 % готовности работы;

разделы 4, 5 к 15.04.2023 – 60 % готовности работы;

раздел 6 к 15.05.2023 – 90 % готовности работы;

оформление пояснительной записки и графического материала к 31.05.2023 – 100 % готовности работы.

Защита курсовой работы с 01.06.2023 по 09.06.2023 г.

РУКОВОДИТЕЛЬ–––––– Е.Е.Фадеева

(подпись)

Задание принял к исполнению –––\_\_\_\_ ––

(дата и подпись студента)

# СОДЕРЖАНИЕ

[СОДЕРЖАНИЕ 4](#_Toc136520264)

[ВВЕДЕНИЕ 6](#_Toc136520265)

[1. АНАЛИЗ ПРОТОТИПОВ, ЛИТЕРАТУРНЫХ ИСТОЧНИКОВ И ФОРМИРОВАНИЕ ТРЕБОВАНИЙ К ПРОЕКТИРУЕМОМУ 7](#_Toc136520266)

[1.1. Запись правил синтаксиса языков программирования при помощи синтаксических диаграмм 7](#_Toc136520267)

[1.2. Примеры решения аналогичных задач, анализ достоинств и недостатков известных решений 8](#_Toc136520268)

[1.3. Требования к разрабатываемому программному средству 13](#_Toc136520269)

[2. АНАЛИЗ ТРЕБОВАНИЙ К ПРОГРАММНОМУ СРЕДСТВУ И РАЗРАБОТКА ФУНКЦИОНАЛЬНЫХ ТРЕБОВАНИЙ 14](#_Toc136520270)

[2.1. Описание функциональности программного средства 14](#_Toc136520271)

[2.2. Спецификация функциональных требований 16](#_Toc136520272)

[3. ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА 19](#_Toc136520273)

[3.1. Проектирование алгоритма взаимодействия пользователя с полотном 19](#_Toc136520274)

[3.2. Проектирование алгоритма рисования линий 23](#_Toc136520275)

[3.3. Проектирование алгоритма рисования фигуры 24](#_Toc136520276)

[3.4. Проектирование алгоритма отмены действия 25](#_Toc136520277)

[4. КОНСТРУИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА 26](#_Toc136520278)

[4.1. Структура программного средства 26](#_Toc136520279)

[4.2. Разработка модуля Data 26](#_Toc136520280)

[4.3. Разработка модуля Stack 27](#_Toc136520281)

[4.4. Разработка модуля Model 28](#_Toc136520282)

[4.5. Разработка модуля View 30](#_Toc136520283)

[4.6. Разработка модуля Main 32](#_Toc136520284)

[4.7. Разработка модуля frmViewGridLines 34](#_Toc136520285)

[5. ТЕСТИРОВАНИЕ, ПРОВЕРКА РАБОТОСПОСОБНОСТИ И АНАЛИЗ ПОЛУЧЕННЫХ РЕЗУЛЬТАТОВ 35](#_Toc136520286)

[6. РУКОВОДСТВО ПО УСТАНОВКЕ И ИСПОЛЬЗОВАНИЮ 42](#_Toc136520287)

[СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ 44](#_Toc136520288)

[ЗАКЛЮЧЕНИЕ 45](#_Toc136520289)

[ПРИЛОЖЕНИЕ А 46](#_Toc136520290)

[ПРИЛОЖЕНИЕ Б 47](#_Toc136520291)

[Содержание модуля Main 47](#_Toc136520292)

[Содержание модуля Model 53](#_Toc136520293)

[Содержание модуля View 58](#_Toc136520294)

[Содержание модуля Stack 65](#_Toc136520295)

[Содержание модуля Data 66](#_Toc136520296)

[Содержание модуля frmViewGridLines 67](#_Toc136520297)

# ВВЕДЕНИЕ

Языки программирования не зря называют языками. В них всё так же, как и в естественных языках, используемых людьми для общения. Каким бы ни был язык программирования, его определяют те же три элемента, что и у естественных языков: алфавит, синтаксис и семантика.

Для записи правил синтаксиса языка программирования используется специальная система обозначений – метаязык. Можно сказать, что метаязык – это язык, который используется для описания других языков. Два метаязыка, получивших наибольшее распространение, – это расширенная форма Бэкуса-Наура и синтаксические диаграммы.

Язык РБНФ более строг и точен, применяется во многих инструментах, позволяющих автоматизировать разработку отдельных частей компиляторов. Основным преимуществом РБНФ является то, что получающиеся описания представляют собой обычный текст, который легко может быть введён с клавиатуры и обработан программно. Это же является и недостатком: человеку намного проще воспринимать графическую информацию, чем текст.

Эту проблему решает ещё один популярный метаязык – синтаксические диаграммы. Синтаксические диаграммы более громоздки, однако более наглядны и просты для понимания человеком.

Цель данного курсового проектирования – разработать программное средство построения синтаксических диаграмм с использованием векторной графики.

# АНАЛИЗ ПРОТОТИПОВ, ЛИТЕРАТУРНЫХ ИСТОЧНИКОВ И ФОРМИРОВАНИЕ ТРЕБОВАНИЙ К ПРОЕКТИРУЕМОМУ

## Запись правил синтаксиса языков программирования при помощи синтаксических диаграмм

В метаязыках, описывающих синтаксис языка программирования, используются следующие понятия: метапеременная, метаконстанта, метасимвол.

Метапеременная — это элемент метаязыка, задающий произвольный фрагмент программы. Для записи метапеременных используются последовательности слов русского языка и служебных слов, между которыми находится символ подчеркивания. Метапеременные при записи заключаются в угловые скобки. Метапеременная на размеченном ребре графа означает, что этот фрагмент диаграммы должен быть детализирован подстановкой синтаксической диаграммы с именем, соответствующим данной метапеременной. Примеры записи метапеременных:

*<Шестн.\_цифра>*

*<Оператор\_While>*

*<Объявление\_переменных>*

Метаконстанта — это элемент метаязыка, задающий фрагмент программы, который в описываемом языке должен записываться строго определённым образом. В синтаксической диаграмме метаконстанте соответствует она сама. Примеры метаконстант:

*End Case For*

Метасимвол — это элемент метаязыка, задающий связь между метаконстантами и метапеременными. В синтаксических диаграммах, в отличии от РБНФ, используются только два метасимвола: «::=» и «<>».

Метасимвол «::=» используется для отделения заголовка синтаксической диаграммы от графа. Метасимвол «<>» используется для обозначения метапеременных.

Каждая синтаксическая диаграмма имеет имя, в качестве которого используется соответствующая метапеременная. Синтаксическая диаграмма представляет собой ориентированный граф с размеченными ребрами. Для разметки ребер используются метапеременные и метаконстанты. Метасимволы на синтаксической диаграмме не используются. Поэтому метаконстанты в синтаксических диаграммах в кавычки не заключаются.

Ниже рассмотрено представление в виде ориентированных графов некоторых из метасимволов языка РБНФ:

1. Выбору, альтернативе (метасимволу | (или)) соответствует разветвление в синтаксической диаграмме с последующим объединением:

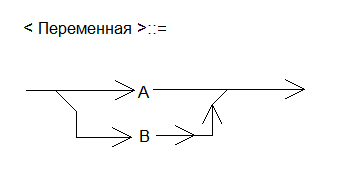


Рисунок 1.1 – пример конструкции выбора

1. Необязательной части конструкции (повторению ноль или один раз, то есть метасимволам [ ]) соответствует диаграмма:

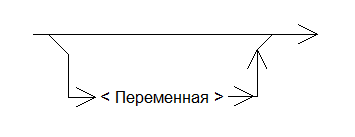


Рисунок 1.2 – пример необязательной части конструкции

1. Возможности повторения конструкций ноль, один или более раз (метасимволам { }) соответствует фрагмент синтаксической диаграммы:

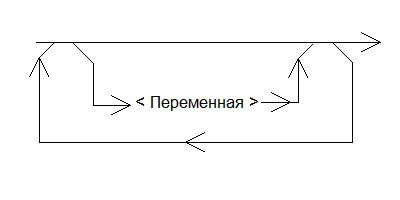


Рисунок 1.3 – пример конструкции повторения

## Примеры решения аналогичных задач, анализ достоинств и недостатков известных решений

В зарубежной литературе я обнаружил, что синтаксические диаграммы часто именуются как «Railroad diagrams» (дословно это переводится как «Схемы железной дороги»). Среди множества программных средств, работающих с векторной графикой, я могу назвать только несколько приложений, редактирующих синтаксические диаграммы.

**Railroad Diagram Generator**

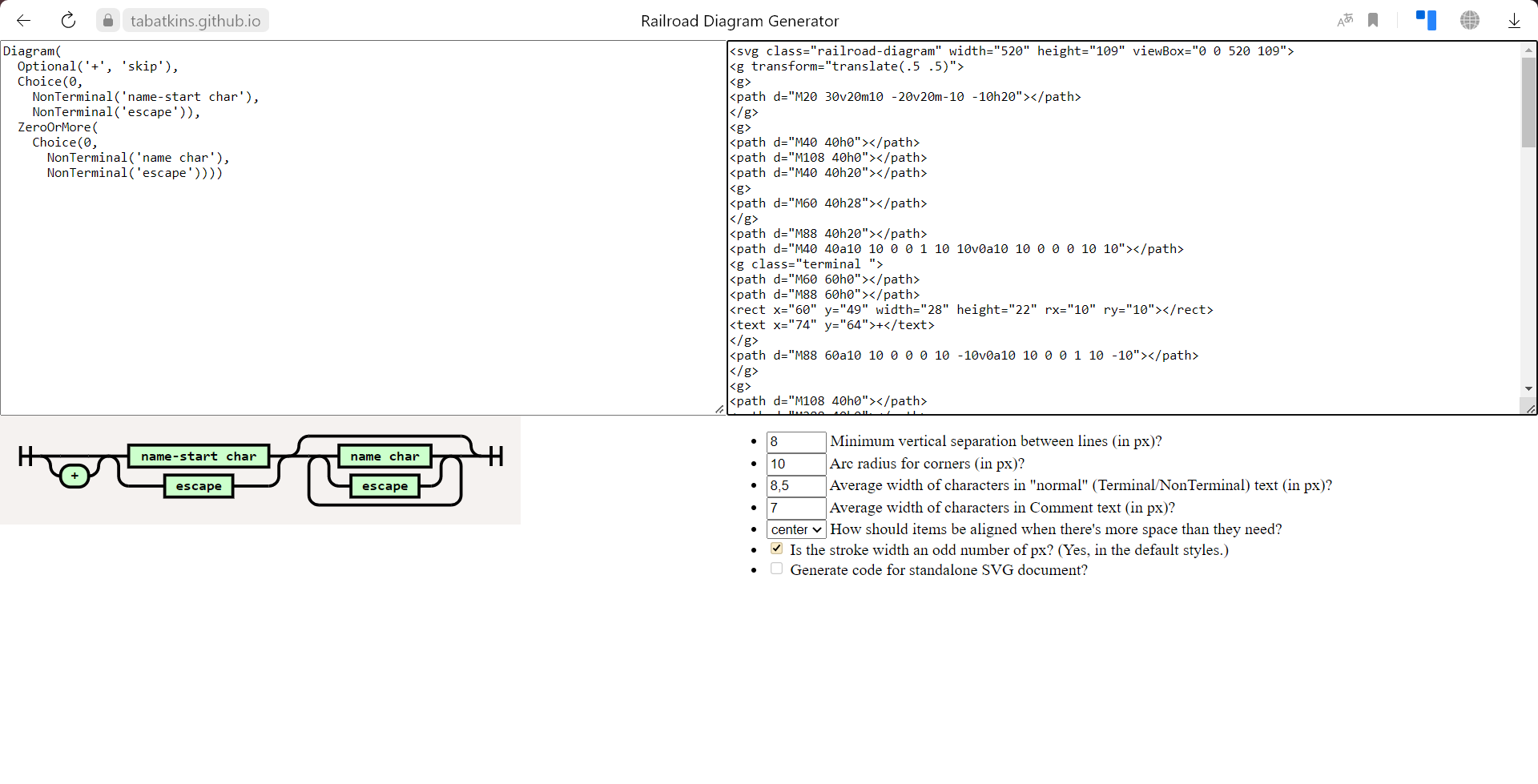


Рисунок 1.4 – генератор синтаксических диаграмм

Программа "Railroad Diagram Generator" (<https://goo.su/oqvd>) предназначена для создания синтаксических диаграмм, которые используются для визуализации грамматических конструкций, синтаксических правил и алгоритмов. Основной функционал этой программы:

1. Создание диаграммы: Позволяет пользователю создавать новую диаграмму, выбирая тип элементов и их расположение на холсте. Диаграмма может быть пустой или начальными элементами, такими как начальное состояние или заголовок.
2. Редактирование диаграммы: Позволяет пользователю добавлять, перемещать, изменять размеры и удалять элементы диаграммы. Пользователь может настраивать свойства элементов, такие как цвет, шрифт и стиль линий.
3. Типы элементов: Предоставляет набор предопределенных типов элементов, таких как стрелки, прямоугольники, овалы и текстовые блоки. Каждый тип элемента имеет свои особенности и опции настройки.
4. Соединение элементов: Позволяет пользователю соединять элементы диаграммы с помощью стрелок или линий, чтобы обозначить связи или потоки данных. Пользователь может редактировать стиль стрелок и линий, включая их тип, цвет и размер.
5. Масштабирование и перемещение: Позволяет пользователю масштабировать и перемещать диаграмму на холсте для обеспечения лучшего обзора и удобства работы.
6. Сохранение и экспорт: Позволяет пользователю сохранить созданную диаграмму в файле проекта или экспортировать ее в различные форматы, такие как изображения (PNG, JPEG) или документы (PDF, HTML).
7. Импорт и экспорт шаблонов: Предоставляет возможность импортировать существующие шаблоны диаграмм или экспортировать созданные пользователем шаблоны для дальнейшего использования или обмена с другими пользователями.
8. Поддержка грамматических правил: Позволяет пользователю определить и применить грамматические правила к элементам диаграммы, чтобы автоматически генерировать структуры или подсвечивать ошибки на основе этих правил.
9. Управление слоями: Позволяет пользователю работать с различными слоями диаграммы, чтобы упорядочить и контролировать отображение элементов.
10. Поддержка межъязыковых диаграмм: Предоставляет возможность создавать и редактировать диаграммы на различных языках, включая поддержку разных алфавитов и символов.

Достоинства:

1. Бесплатное программное средство
2. Отсутствие рекламы

Недостатки:

1. Полученные диаграммы сильно отличаются от синтаксических диаграмм, используемых для описания синтаксиса языков программирования
2. Сложный интерфейс
3. Нет возможности работы с векторной графикой
4. Необходимость доступа к интернету

К сожалению, на данный момент существует очень мало программных средств, которые создают синтаксические диаграммы с использованием векторной графики. Поэтому я выбрал некоторые популярные векторные графические редакторы.

**Adobe Illustrator**

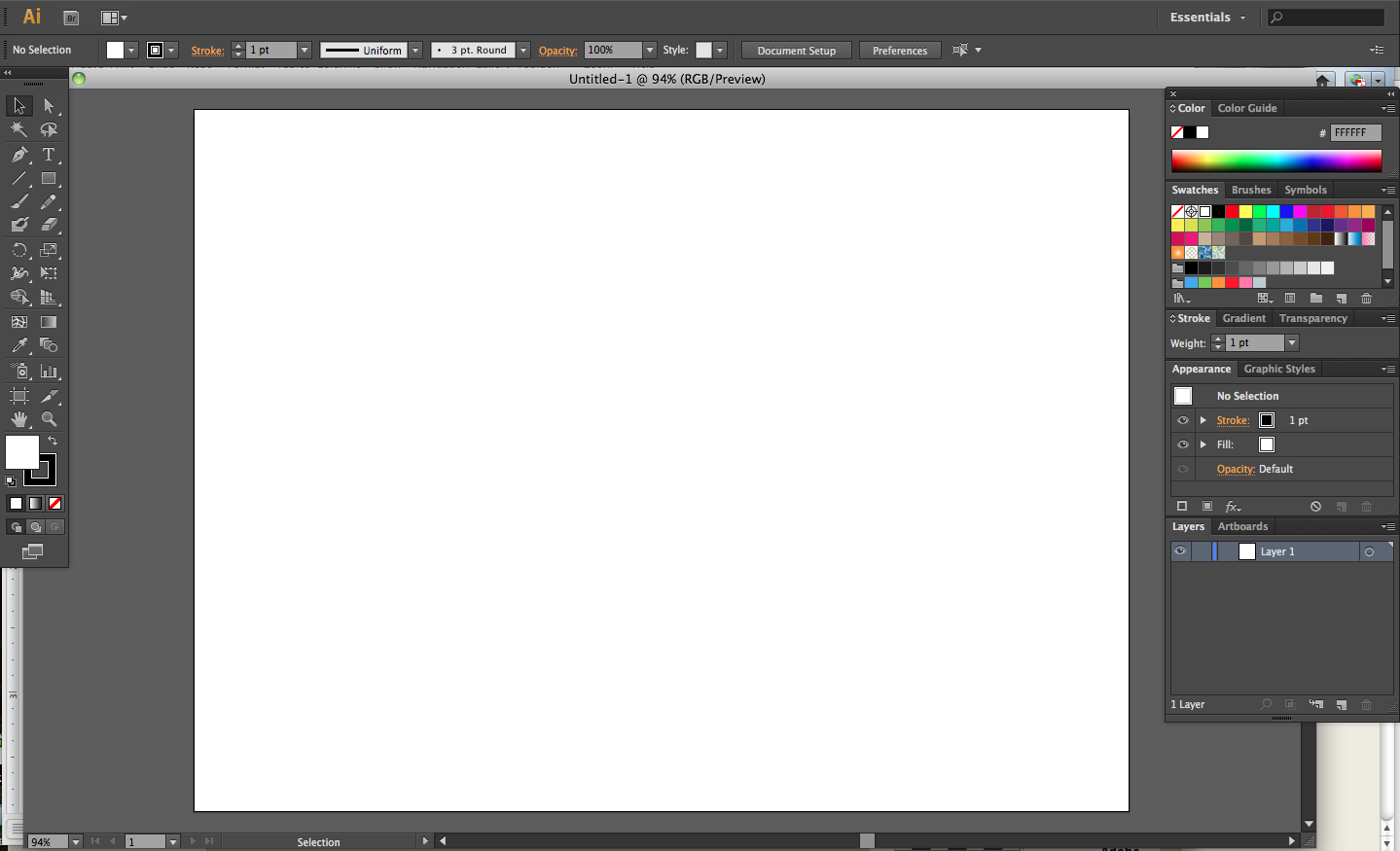


Рисунок 1.5 – интерфейс программы Adobe Illustrator

Adobe Illustrator - это векторный графический редактор, предоставляющий множество функций для создания и редактирования иллюстраций, логотипов, макетов и других векторных изображений. Он обладает широким спектром инструментов для рисования, манипулирования формами, применения эффектов и трансформаций, а также поддерживает многослойность и интеграцию с другими программами Creative Cloud. Adobe Illustrator также предлагает возможность экспорта изображений в различные форматы для печати, веба и других целей.

Достоинства:

1. Удобный пользовательский графический интерфейс
2. Огромное количество функций
3. Множество форматов для экспорта

Недостатки:

1. Высокая стоимость программного средства
2. Неопытному пользователю будет сложно разобраться в работе приложения, чтобы создавать синтаксические диаграммы

**Inkscape**

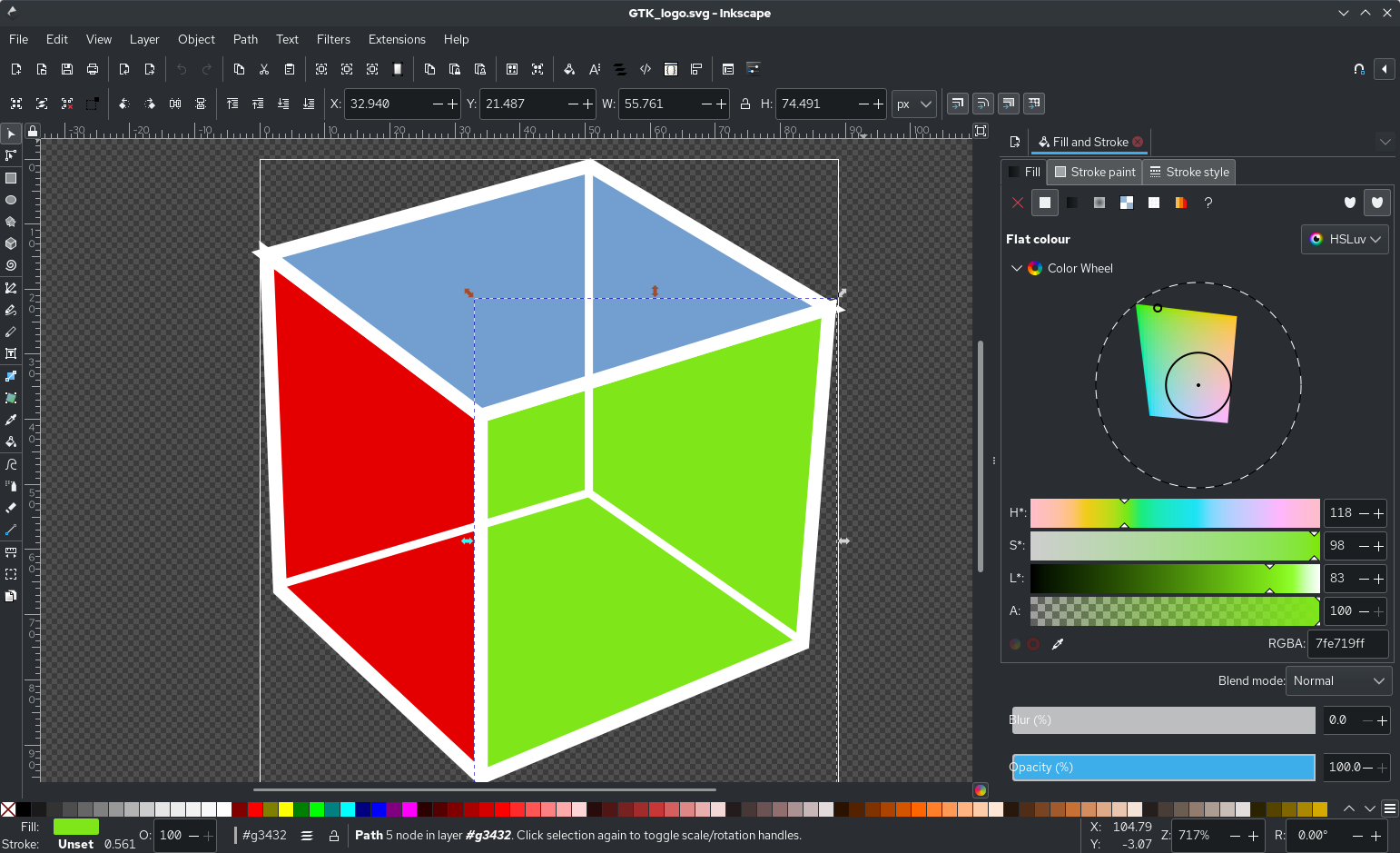


Рисунок 1.6 – интерфейс программы Inkscape

Inkscape - это бесплатный векторный графический редактор с открытым исходным кодом, предназначенный для создания и редактирования векторной графики. Он предоставляет широкий набор инструментов для рисования, создания форм, работы с текстом, применения фильтров и эффектов, а также поддерживает многослойность и масштабируемые векторные форматы. Inkscape обладает такими функциями, как вырезание, вставка и копирование объектов, выравнивание элементов, а также импорт и экспорт файлов в различных форматах, включая SVG, PDF, EPS и другие.

Достоинства:

1. Программное средство бесплатное для использования
2. Удобство использования

Недостатки:

1. Невозможность поддержки привычных горячих клавиш, если текущая раскладки клавиатуры не английская
2. Некорректный экспорт в другие форматы проприетарных графических программ (Adobe, CorelDraw)
3. Не работает поддержка шрифтов, которые реализованы без поддержки юникода
4. Программа не удобна для создания синтаксических диаграмм

## Требования к разрабатываемому программному средству

Данное курсовое проектирование посвящено разработке программного средства построения синтаксической диаграммы с использованием векторной графики. Проведя тщательный анализ существующих аналогичных программных средств, я определил основные функции, которые должно решать разрабатываемое программное средство:

1. Программное средство должно иметь максимально удобный графический пользовательский интерфейс
2. Создание синтаксических диаграмм должно производится с помощью установленных примитивных элементов диаграммы
3. Возможность изменять размеры рабочей области
4. Возможность сохранять изображение в файле
5. Возможность экспорта изображения в популярные векторные и растровые форматы

В качестве входных данных будут выступать типизированные файлы с возможностью дальнейшего использования. В качестве выходных данных будут выступать графические изображения в векторном и растровом формате.

Данное программное средство разрабатывается для операционной системы Windows 10. В качестве языка программирования был выбран Delphi, среда разработки – RAD Studio.

# АНАЛИЗ ТРЕБОВАНИЙ К ПРОГРАММНОМУ СРЕДСТВУ И РАЗРАБОТКА ФУНКЦИОНАЛЬНЫХ ТРЕБОВАНИЙ

## Описание функциональности программного средства

Программное средство должно выполнять следующий набор функций:

1. Строка меню (раздел Файл)
   1. «Новый»
      1. Когда нажата кнопка «Новый», открывается диалоговое окно, в котором пользователь может указать имя создаваемого файла, путь в котором будет размещен файл.
      2. Все файлы с готовыми синтаксическими диаграммами, которые создает программное средство, имеют расширение .dgr.
      3. После создания, в рабочей области появляется белое полотно, с которым пользователь может взаимодействовать.
   2. «Открыть»
      1. Когда нажата кнопка «Открыть», открывается диалоговое окно, в котором пользователь может, перемещаясь между папками, открыть уже существующий файл с форматом .dgr.
      2. После открытия, в рабочую область помещается полотно вместе с диаграммой.
   3. «Сохранить»
      1. При нажатии на эту кнопку все изменения будут сохранены, функции отмены действия и очищения полотна будут недоступны.
      2. Если файл был недавно создан, то появится диалоговое окно, в котором пользователь может выбрать имя файла и расположение на диске, где этот файл будет сохранен.
      3. Файл сохраняется в формате .dgr.
   4. «Сохранить как»
      1. Имеет тот же функционал, что и «Сохранить», но позволяет выбрать формат, в котором будет сохранен файл:
         1. Векторные форматы (SVG).
         2. Растровые форматы (PNG).
2. Строка меню (раздел Вид)
   1. «Линии сетки»
      1. Показывает линии сетки на полотне.
      2. При нажатии появляется новая форма, в которой пользователь может указать размер ячейки.
      3. Цвет линий сетки светло-серый.
3. Панель инструментов:
   1. Заголовок синтаксической диаграммы (значок ::=)
      1. При нажатии этой кнопки включается режим рисования.
      2. Заголовок синтаксической диаграммы появляется на полотне по координатам клика.
      3. Пользователь может изменить текст внутри фигуры.
      4. Введенный пользователем текст автоматически заключается в кавычки «<>::=».
   2. Метапеременная (значок <>)
      1. При нажатии этой кнопки включается режим рисования.
      2. Метапеременная появляется на полотне по координатам клика.
      3. Пользователь может изменить текст внутри фигуры.
      4. Метапеременная автоматически заключается в кавычки «<>».
   3. Метаконстанта (значок M)
      1. При нажатии этой кнопки включается режим рисования.
      2. Метаконстанта появляется на полотне по координатам клика.
      3. Пользователь может изменить текст внутри фигуры.
      4. Метаконстанта в кавычки не берётся.
   4. Линия (значок )
      1. При нажатии этой кнопки включается режим рисования линий, который работает следующим образом: при нажатии левой кнопки мыши ставится точка, а за курсором тянется штрихпунктирная линия синего цвета, которая подсказывает пользователю расположение будущей линии.
      2. По повторном клике по полотну ставится вторая точка. Две точки на полотке соединяются линией.
      3. Если точка конечная, то линий заканчивается стрелкой.
      4. Если линия горизонтальная, но не конечная, то рисуется стрелка на середине линии.
      5. Если линия вертикальная и начальная, то рисуется диагональный срез.
      6. Если линия вертикальная и конечная, то рисуется горизонтальный срез со стрелкой.
      7. Рисование текущей линии прекращается, когда пользователь нажмет правую кнопку мыши.
      8. Все линии рисуются перпендикулярно сторонам полотна.
   5. Отмена действия (значок отмены)
      1. В случае ошибки при построении диаграммы пользователь может отменить действие.
      2. Отмена становится недоступна, если фигур и линий на полотне нет.
   6. Очистка полотна (значок шредера)
      1. При нажатии все фигуры с полотна удаляются.
      2. Очитка полотна становится недоступна, если фигур и линий на полотне нет.
4. Рабочая область
   1. Рабочая область представляет собой белое полотно с размерами, указанными пользователем.

## Спецификация функциональных требований

Среди функциональных требований есть «Заголовок синтаксической диаграммы», «Метапеременная», «Метаконстанта».

Спецификация данной функции может иметь следующий вид:

* Каждая фигура, которая помещается на полотно, имеет прозрачный фон без обводки.
* Текст внутри фигуры располагается строго в центре.
* При отмене действия последняя добавленная фигура удаляется с полотна.

Среди функциональных требований есть «Линия».

Спецификация данной функции может иметь следующий вид:

* Линии должны соединять все точки в том порядке, в котором их поставил пользователь:

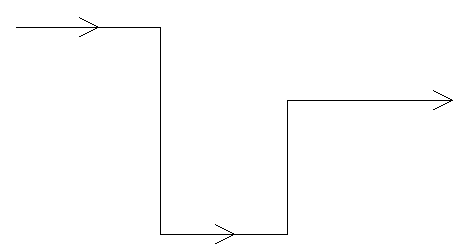


Рисунок 2.1 – Пример линии

* Если точка конечная, то линий заканчивается стрелкой:

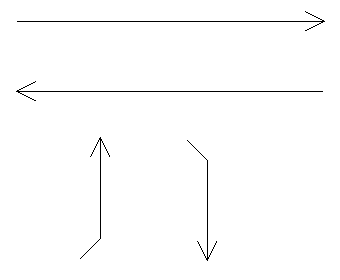


Рисунок 2.2 – Пример линий, заканчивающихся стрелкой

* Если линия горизонтальная, но не конечная, то рисуется стрелка на середине линии:

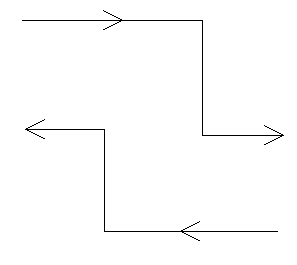


Рисунок 2.3 – Пример линий со стрелками на серединах отрезков

* Если линия вертикальная и начальная, то рисуется диагональный срез:

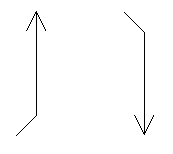


Рисунок 2.4 – Пример вертикальных начальных линий с диагональным срезом

* Если линия вертикальная и конечная, то рисуется горизонтальный срез со стрелкой:

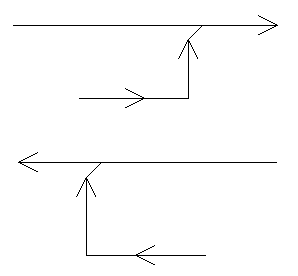


Рисунок 2.5 – Пример вертикальных конечных линий со стрелкой и диагональным срезом на конце

# ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

Проектирование программного средства состоит из проектирования следующих алгоритмов:

* Алгоритм взаимодействия пользователя с полотном
* Алгоритм рисования линий
* Алгоритм рисования фигур
* Алгоритм отмены действия

Схема программы представлена в Приложении А.

Для реализации переключения режима рисования и изменения типа фигуры я решил использовать перечислимые типы. Типы фигуры реализованы при помощи перечислимого типа TTypeFigure со значениями tfLine – тип фигуры «Линия», tfVar – тип фигуры «Метапеременная», tfConst – тип фигуры «Метаконстанта», tfHeader – тип фигуры «Заголовок синтаксической диаграммы», tfNothing – не соответствует ни одному из типов фигур, необходим для первого элемента в однонаправленном списке фигур. Режим рисования представляет собой перечислимый тип TDrawMode со значениями dmLine – режим рисования линии, dmFigure – режим рисования фигуры, dmNone – не соответствует ни одному из режимов рисования, необходим при запуске программы.

## Проектирование алгоритма взаимодействия пользователя с полотном

Пользователь взаимодействует с полотном через нажатие кнопок мыши. После нажатия левой кнопкой мыши по полотну значение координат (x, y) клика передаются в процедуру обработки клика. Если тип фигуры – «Фигура», то координаты клика передаются в функцию AddNewFigureToList, которая помещает новую фигуру на вершину однонаправленного списка фигур и возвращает указатель на эту фигуру (см. рис. 3.1). Если тип фигуры – «Линия» и это первая точка линии, то координаты клика передаются в функцию AddNewLineToList, которая помещает линию на вершину однонаправленного списка фигур, создает новый указатель на однонаправленный список точек PointHead и возвращает указатель на добавленную линию (см рис. 3.2). Если тип фигуры – «Линия» и это не первая точка линии, то координаты клика передаются в процедуру AddNewPointToList, которая помещает точку на вершину однонаправленного списка точек (см. рис. 3.3).

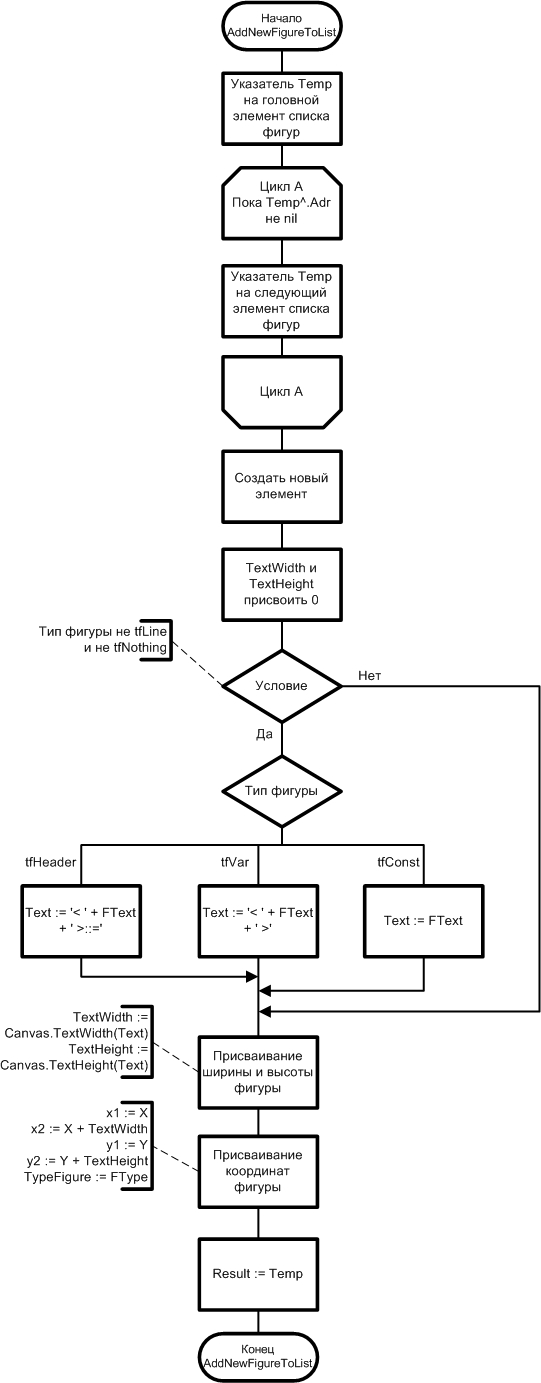


Рисунок 3.1 – Функция AddNewFigureToList



Рисунок 3.2 – Функция AddNewLineToList

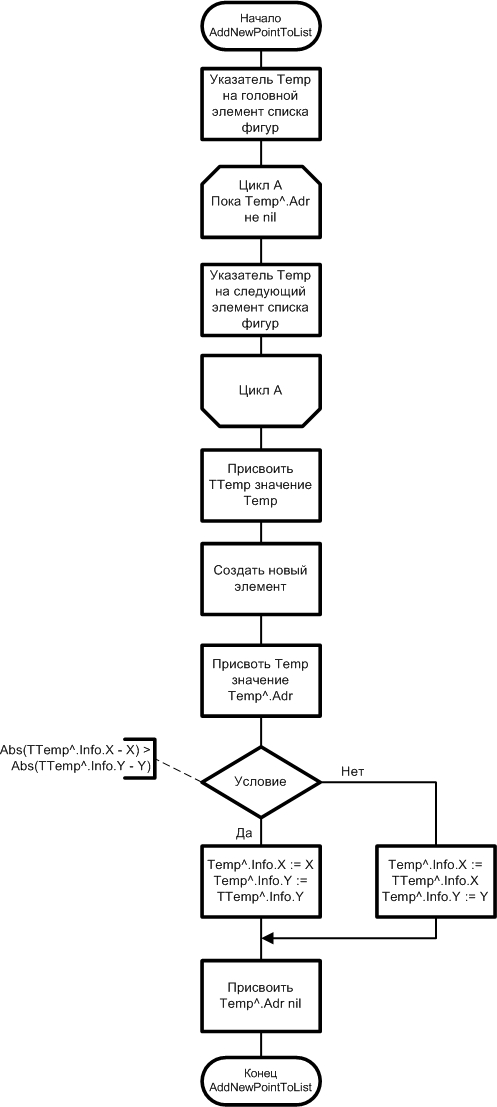


Рисунок 3.3 – Процедура AddNewPointToList

## Проектирование алгоритма рисования линий

Алгоритм рисования линий представляет собой последовательность шагов, которые определяют, каким образом будет происходить создание и отображение линий на полотне. В соответствии со спецификацией требований схема алгоритма рисования линии приведена на рисунке 3.4.

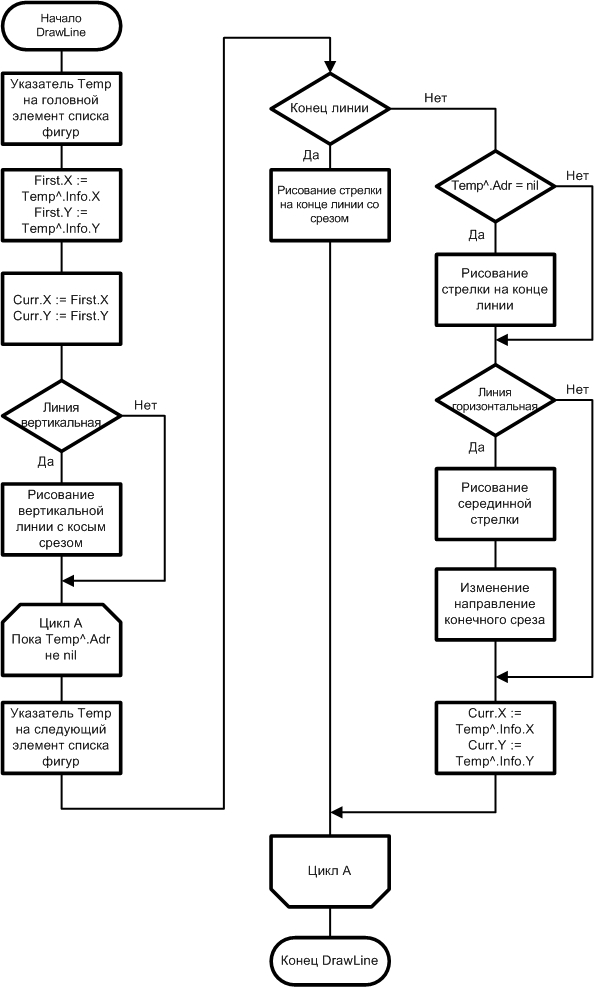


Рисунок 3.4 – Процедура DrawLine

## Проектирование алгоритма рисования фигуры

После добавления очередной фигуры в однонаправленный список фигур полотно, на котором происходит рисование, обновляется. При этом полотно очищается – на полотне рисуется белый прямоугольник с координатами равными размерам полотна –, и происходит рисования фигур и линий. Схема процедуры DrawFigure приведена на рисунке 3.5.

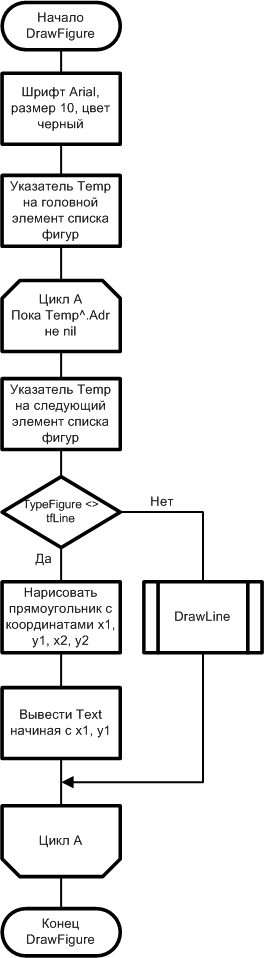


Рисунок 3.5 – Процедура DrawFigure

## Проектирование алгоритма отмены действия

Фигуры, которые пользователь поместил на полотно, находятся в однонаправленном списке фигур. Поэтому алгоритм отмены действия прост: при нажатии на кнопку «Отмена действия» или при сочетании клавиш Ctrl + Z, память, выделенная для предыдущей добавленной фигуры, освобождается и адресное поле предпоследней фигуры присваивается nil. Схема процедуры отмены действия представлен на рисунке 3.6.

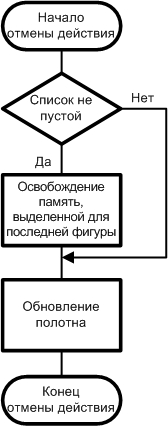


Рисунок 3.6 – Процедура отмены действия

# КОНСТРУИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

## Структура программного средства

На основании функциональных требований из предыдущих разделов разрабатываемое программное средство состоит из следующих модулей:

* Модуль Data ­– содержит в себе описание типов и структур данных, переменных, которые используются в других модулях
* Модуль Stack – содержит в себе подпрограммы для работы с однонаправленным списком фигур
* Модуль Model – содержит в себе подпрограммы добавления фигуры, линии, точки в однонаправленный список фигур, а также рисования фигуры и линии
* Модуль View – содержит в себе подпрограммы для работы с полотном, экспорта и импорта файла в растровом и векторном формате, а также сохранения фигур в типизированном файле
* Модуль Main – основной модуль, в котором содержится основная логика работы программного средства
* Модуль frmViewGridLines – модуль формы frmViewGridLines, содержит в себе подпрограммы для изменения размера ячейки сетки

## Разработка модуля Data

Таблица 4.1 – Описание основных структур данных

|  |  |  |
| --- | --- | --- |
| **Название** | **Назначение** | **Структура** |
| TDrawMode | Перечислимый тип. Предназначен для режимов рисования на полотне (рисование линии, фигуры, точки) | TDrawMode = (dmLine, dmFigure, dmPoint, dmNone) |
| TTypeFigure | Перечислимый тип. Предназначен для рисования различных типов фигур: линия, метапеременная, метаконстанта, заголовок синтаксической диаграммы | TTypeFigure = (tfLine, tfVar, tfConst, tfHeader, tfNothing) |
| TFileMode | Перечислимый тип. Предназначен для различных режимов сохранения файла (в векторном формате SVG, в формате PNG и в типизированном файле DGR) | TFileMode = (fmSvg, fmPng, fmDgr); |
| TGridMode | Перечислимый тип. Предназначен для переключения между режимами отрисовки линий сетки | TGridMode = (gmOn, gmOff) |

Таблица 4.1 – Продолжение

|  |  |  |
| --- | --- | --- |
| PPointsList | Указатель на однонаправленный список точек, которые пользователь ставит на Canvas. В информационном поле содержатся координаты точек | PPointsList = ^TPointsList;  TPointsList = record  Info: TPointsInfo;  Adr: PPointsList;  end;  TPointsInfo = record  x, y: integer;  end; |
| PFigureList | Указатель на список фигур, добавленных пользователем на полотне | PFigureList = ^FigureList;  FigureList = record  Info: TFigureInfo;  Adr: PFigureList;  end;  TFigureInfo = record  case TypeFigure: TTypeFigure of  tfVar, tfConst, tfHeader:  (x1, y1, x2, y2: integer; Text: String[100]);  tfLine:  (PointHead: PPointsList);  tfNothing:  ();  end; |
| TFigureInFile | Тип предназначен для записи информации о фигурах в типизированный файл | TFigureInFile = record  case TypeFigure: TTypeFigure of  tfVar, tfConst, tfHeader:  (x1, y1, x2, y2: integer; Text: String[100]);  tfLine:  (Point: String[200]);  tfNothing:  ();  end; |

## Разработка модуля Stack

Таблица 4.2 – Описание процедур и функций, используемых в модуле Stack

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Имя подпрограммы** | **Описание** | **Заголовок подпрограммы** | **Имя параметра** | **Назначение параметра** |
| ClearFigureList | Очищает список фигур, на который указывает FigureHead | procedure ClearFigureList(FigureHead: PFigureList); | FigureHead | Головной элемент списка фигур |

Таблица 4.2 – Продолжение

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| CreateFigureList | Создает новый список фигур и возвращает указатель на созданный список через переменную HeadList | procedure CreateFigureList(var HeadList: PFigureList); | HeadList | Головной элемент однонаправленного списка фигур |
| PopFigureList | Освобождает память, выделенную для последнего элемента в однонаправленном списке фигур | procedure PopFigureList(FigureHead: PFigureList; var Info: TFigureInfo); | FigureHead | Головной элемент однонаправленного списка фигур |
| Info | Информационное поле последнего элемента в списке фигур |
| ListIsEmpty | Проверяет наличие элементов в однонаправленном списке фигур | function ListIsEmpty(Element: PFigureList): boolean; | Element | Указатель на элемент в однонаправленном списке фигур |
| GetFigureHead | Возвращает головной элемент в однонаправленном списке фигур | function GetFigureHead(FigureHead: PFigureList): PFigureList; | FigureHead | Головной элемент однонаправленного списка фигур |

## Разработка модуля Model

Таблица 4.3 – Описание процедур и функций, используемых в модуле Model

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Имя подпрограммы** | **Описание** | **Заголовок подпрограммы** | **Имя параметра** | **Назначение параметра** |
| AddNewFigureToList | Добавляет фигуру в однонаправленный список фигур и возвращает указатель на добавленную фигуру | function AddNewFigureToList(Canvas: TCanvas; FigureHead: PFigureList;  const X, Y: Integer; const FType: TTypeFigure; const FText: String) | Canvas | Полотно |
| FigureHead | Головной элемент списка фигур |
| X, Y | Координаты клика |
| FType | Тип фигуры |
| FText | Текст внутри фигуры |

Таблица 4.3 – Продолжение

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| DrawFigure |  | procedure DrawFigure(Canvas: TCanvas; FigureHead: PFigureList); | Canvas | Полотно |
| FigureHead | Головной элемент списка фигур |
| AddNewLineToList |  | function AddNewLineToList(var FigureHead: PFigureList; X, Y: Integer)  : PFigureList; | FigureHead | Головной элемент списка фигур |
| X, Y | Координаты клика |
| PFigureList | Указатель на добавленную фигуру |
| AddNewPointToList | Добавляет новую точку в список точек, на который указывает PointsHead. Координаты точки передаются через переменные X и Y. Функция возвращает указатель на последнюю добавленную точку | function AddNewPointToList(var PointsHead: PPointsList; X, Y: Integer)  : PPointsList; | PointsHead | Вершина списка точек |
| X, Y | Координаты клика |
| DrawLine | Рисует линию | procedure DrawLine(Canvas: TCanvas; PointHead: PPointsList); | Canvas | Полотно |
| PointHead | Вершина списка точек |
| GorizArrow | Рисует горизонтальную стрелку | procedure GorizArrow(Canvas: TCanvas; X, Y: Integer; C: Integer); | Canvas | Полотно |
| X, Y | Координаты точки |
| С | Коэффициент |
| VertArrow | Рисует вертикальную стрелку | procedure VertArrow(Canvas: TCanvas; X, Y: Integer; C: Integer); | Canvas | Полотно |
| X, Y | Координаты точки |
| С | Коэффициент |

## Разработка модуля View

Таблица 4.4 – Описание процедур и функций, используемых в модуле View

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Имя подпрограммы** | **Описание** | **Заголовок подпрограммы** | **Имя параметра** | **Назначение параметра** |
| ClearCanvas | Очищает Canvas | procedure ClearCanvas(Canvas: TCanvas; const CanvasWigth, CanvasHeight: Integer); | Canvas | Полотно, на котором происходит рисование |
| CanvasWigth, CanvasHeight | Ширина и высота полотна |
| Streak | Рисует сетку на полотне | procedure Streak(Canvas: TCanvas; PointHead: PPointsList; X, Y: Integer); | Canvas | Полотно, на котором происходит рисование |
| PointHead | Вершина списка точек |
| X, Y | Координаты точки |
| SaveFile | Сохраняет в файл | function SaveFile(FileMode: TFileMode; MainSaveDialog: TSaveDialog): String; | FileMode | Файловый режим |
| MainSaveDialog | Диалог сохранения |
| OpenFile | Открывает файл | function OpenFile(FileMode: TFileMode; MainOpenDialog: TOpenDialog): String; | FileMode | Файловый режим |
| MainOpenDialog | Диалог открытия |
| OpenCanvasFromDGR | Открытие из типизированного файла | procedure OpenCanvasFromDGR(const FileName: string; FigureHead: PFigureList); | FileName | Файловый режим |
| FigureHead | Головной элемент списка фигур |
| SaveCanvasToPNG | Сохранить в PNG файле | procedure SaveCanvasToPNG(const FileName: string; Canvas: TCanvas;  CanvasWidth, CanvasHeight: Integer); | FileName | Название файла |
| Canvas | Полотно |
| CanvasWidth,  CanvasHeight | Ширина и высота полотна |

Таблица 4.4 – Продолжение

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| SaveCanvasToSVG | Сохранить в SVG файле | procedure SaveCanvasToSVG(const FileName: string; Width, Height: Integer;  FigureHead: PFigureList); | FileName | Название файла |
| Width, Height | Ширина и высота полотна |
| FigureHead | Головной элемент списка фигур |
| ListToStr | Преобразовать список точек в строку | function ListToStr(fPoints: PPointsList): String; | fPoints | Элемент списка точек |
| ReplaceChar | Заменить недопустимые символы в SVG файле | function ReplaceChar(Text: UTF8String): UTF8String; | Text | Текст |
| SVGMoveTo | Перемещает перо на SVG полотне | procedure SVGMoveTo(var List: UTF8String; X, Y: Integer); | List | Строка |
| X, Y | Координаты точки |
| SVGLineTo | Рисует линию на SVG полотне | procedure SVGLineTo(var List: UTF8String; X, Y: Integer); | List | Строка |
| X, Y | Координаты точки |
| SVGGorizArrow | Рисует горизонтальную стрелку на SVG полотне | procedure SVGGorizArrow(var List: UTF8String; X, Y: Integer; C: Integer); | List | Строка |
| X, Y | Координаты точки |
| С | Коэффициент |
| SVGVertArrow | Рисует вертикальную стрелку на SVG полотне | procedure SVGVertArrow(var List: UTF8String; X, Y: Integer; C: Integer); | List |  |
| X, Y | Координаты точки |
| С | Коэффициент |

## Разработка модуля Main

Таблица 4.5 – Описание процедур и функций, используемых в модуле Main

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Имя подпрограммы** | **Описание** | **Заголовок подпрограммы** | **Имя параметра** | **Назначение параметра** |
| MainPaintBoxPaint | Предназначен для рисования фигур на полотне | procedure MainPaintBoxPaint(Sender: TObject); | Sender | Объект, вызвавший событие |
| MainPaintBoxMouseDown | Обработчик события нажатия на PaintBox. В зависимости от режима рисования на полотне рисуется либо линия, либо фигура (метаконстанта, метапеременная или заголовок синтаксической диаграммы) | procedure MainPaintBoxMouseDown(Sender: TObject; Button: TMouseButton;  Shift: TShiftState; X, Y: Integer); | Sender | Объект, вызвавший событие |
| Button | Указывает, какая кнопка мыши была нажата |
| Shift | Указывает на состояние клавиш Shift, Ctrl и Alt во время события |
| X, Y | Координаты клика мышью |
| actFileNewExecute | При нажатии на «Новый» полотно очищается и новый файл сохраняется по пути, который указал пользователь | procedure actFileNewExecute(Sender: TObject); | Sender | Объект, вызвавший событие |
| actFileOpenExecute | Разгружает файл по пути, который указал пользователь | procedure actFileOpenExecute(Sender: TObject); | Sender | Объект, вызвавший событие |
| actFileSaveExecute | Сохраняет файл | procedure actFileSaveExecute(Sender: TObject); | Sender | Объект, вызвавший событие |
| actFileSaveAsExecute | Сохраняет файл, пользователь указывает тип файла | procedure actFileSaveAsExecute(Sender: TObject); | Sender | Объект, вызвавший событие |
| actVariableExecute | Тип фигуры становится tfVar | procedure actVariableExecute(Sender: TObject); | Sender | Объект, вызвавший событие |
| actLineExecute | Тип фигуры становится tfLine | procedure actLineExecute(Sender: TObject); | Sender | Объект, вызвавший событие |

Таблица 4.5 – Продолжение

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| actConstExecute | Тип фигуры становится tfConst | procedure actConstExecute(Sender: TObject); | Sender | Объект, вызвавший событие |
| actHeaderExecute | Тип фигуры становится tfHeader | procedure actHeaderExecute(Sender: TObject); | Sender | Объект, вызвавший событие |
| actClearCanvasExecute | Очищает полотно и список фигур | procedure actClearCanvasExecute(Sender: TObject); | Sender | Объект, вызвавший событие |
| actViewGridLinesExecute | Рисует сетку на полотне | procedure actViewGridLinesExecute(Sender: TObject); | Sender | Объект, вызвавший событие |
| actUndoExecute | Отменяет последнее действие | procedure actUndoExecute(Sender: TObject); | Sender | Объект, вызвавший событие |
| FormCreate | Создание основной формы | procedure FormCreate(Sender: TObject); | Sender | Объект, вызвавший событие |
| actClearCanvasExecute | Очищает полотно | procedure actClearCanvasExecute(Sender: TObject); | Sender | Объект, вызвавший событие |
| FormDestroy | Уничтожение формы | procedure FormDestroy(Sender: TObject); | Sender | Объект, вызвавший событие |
| MainPaintBoxMouseMove | Предназначен для рисования штрихпунктирной линии | Procedure MainPaintBoxMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer); | Sender | Объект, вызвавший событие |
| Shift | Указывает на состояние клавиш Shift, Ctrl и Alt во время события |
| X, Y | Координаты клика мышью |

Таблица 4.5 – Продолжение

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| FormClose | Обработчик события закрытия формы. Проверяет, сохранены ли изменения в файл. Если нет, то пользователю будет предложено сохранить изменения в файл | procedure FormClose(Sender: TObject; var Action: TCloseAction); | Sender | Объект, вызвавший событие |
| Action | Определяет, каким образом должна закрыться форма |

## Разработка модуля frmViewGridLines

Таблица 4.6 – Описание подпрограмм модуля frmViewGridLines

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Имя подпрограммы** | **Описание** | **Заголовок подпрограммы** | **Имя параметра** | **Назначение параметра** |
| btnOkClick | Вывод результата «ОК» и изменение размера ячейки | procedure btnOkClick(Sender: TObject); | Sender | Объект, вызвавший событие |
| btnCancelClick | Вывод результата «Отмена» | procedure btnCancelClick(Sender: TObject); | Sender | Объект, вызвавший событие |
| FormKeyDown | Изменение размера ячейки | procedure FormKeyDown(Sender: TObject; var Key: Word; Shift: TShiftState); | Sender | Объект, вызвавший событие |
| Key | Код клавиши |
| Shift | Указывает на состояние клавиш Shift, Ctrl и Alt во время события |

Полный код программного средства размещен в Приложении Б.

# ТЕСТИРОВАНИЕ, ПРОВЕРКА РАБОТОСПОСОБНОСТИ И АНАЛИЗ ПОЛУЧЕННЫХ РЕЗУЛЬТАТОВ

Таблица 5.1 – Тестирование функционала программного средства

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Специфика тестирования** | **Номер теста** | **Вводимые данные** | **Ожидаемый результат** | **Полученный результат** |
| Запуск программы | 1 | Двойной щелчок левой кнопкой мыши по программе | Появление окна программы с полотном | Тест пройден |
| Создание нового файла | 2 | Нажатие сочетания клавиш Ctrl + N, нажатие «No» при появлении предупреждающего окна | Появление и закрытие предупреждающего окна | Тест пройден |
| Создание нового файла | 3 | Нажатие на кнопку «Новый», нажатие «No» при появлении предупреждающего окна | Появление и закрытие предупреждающего окна | Тест пройден |
| Создание нового файла | 4 | Нажатие на кнопку «Новый», нажатие «Yes» при появлении предупреждающего окна | Появление предупреждающего окна, диалогового окна сохранения, успешное сохранение файла в формате DGR по назначенному пути | Тест пройден |
| Открытие файла | 5 | Нажатие сочетания клавиш Ctrl + O | Появление предупреждающего окна, диалогового окна открытия, успешное открытие DGR файла по назначенному пути, появление фигур на полотне | Тест пройден |
| Открытие файла | 6 | Нажатие на кнопку «Открыть» | Появление предупреждающего окна, диалогового окна открытия, успешное открытие DGR файла по назначенному пути, появление фигур на полотне | Тест пройден |

Таблица 5.1 – Продолжение

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Сохранение файла | 7 | Нажатие сочетания клавиш Ctrl + S | Появление диалогового окна сохранения, успешное сохранение файла в формате DGR по назначенному пути | Тест пройден |
| Сохранение файла | 8 | Нажатие на кнопку «Сохранить» | Появление диалогового окна сохранения, успешное сохранение файла в формате DGR по назначенному пути | Тест пройден |
| Сохранение файла как | 9 | Нажатие сочетания клавиш Shift + F12 | Появление диалогового окна сохранения | Тест пройден |
| Сохранение файла как | 10 | Нажатие на кнопку «Сохранить как», выбор «SVG» при появлении диалогового окна сохранения | Появление диалогового окна сохранения, успешное сохранение файла в формате SVG по назначенному пути | Тест пройден |
| Сохранение файла как | 11 | Нажатие на кнопку «Сохранить как», выбор «PNG» при появлении диалогового окна сохранения | Появление диалогового окна сохранения, успешное сохранение файла в формате PNG по назначенному пути | Тест пройден |
| Очистка полотна | 12 | Нажатие сочетания клавиш Ctrl + Del | Удаление фигур с полотна | Тест пройден |
| Очистка полотна | 13 | Нажатие кнопки «Очистить полотно» | Удаление фигур с полотна | Тест пройден |
| Отмена действия | 14 | Нажатие сочетания клавиш Ctrl + Z | Удаление с полотна последней добавленной фигуры | Тест пройден |
| Отмена действия | 15 | Нажатие кнопки «Отмена действия» | Удаление с полотна последней добавленной фигуры | Тест пройден |
| Заголовок синтаксической диаграммы | 16 | Нажатие сочетания клавиш Ctrl + 1 | Переключение режима рисования в режим рисования фигур | Тест пройден |

Таблица 5.1 – Продолжение

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Заголовок синтаксической диаграммы | 17 | Нажатие кнопки «Заголовок синтаксической диаграммы» | Переключение режима рисования в режим рисования фигур | Тест пройден |
| Заголовок синтаксической диаграммы | 18 | Нажатие кнопки «Заголовок синтаксической диаграммы», нажатие левой кнопки мыши по полотну | Переключение режима рисования в режим рисования фигур, появление заголовка синтаксической диаграммы | Тест пройден |
| Заголовок синтаксической диаграммы | 19 | Нажатие кнопки «Заголовок синтаксической диаграммы», ввод текста в поле для ввода, нажатие левой кнопки мыши по полотну | Переключение режима рисования в режим рисования фигур, появление заголовка синтаксической диаграммы с текстом внутри, заданным пользователем | Тест пройден |
| Метапеременная | 20 | Нажатие сочетания клавиш Ctrl + 2 | Переключение режима рисования в режим рисования фигур | Тест пройден |
| Метапеременная | 21 | Нажатие кнопки «Метапеременная» | Переключение режима рисования в режим рисования фигур | Тест пройден |
| Метапеременная | 22 | Нажатие кнопки «Метапеременная», нажатие левой кнопки мыши по полотну | Переключение режима рисования в режим рисования фигур, появление метапеременной | Тест пройден |
| Метапеременная | 23 | Нажатие кнопки «Метапеременная», ввод текста в поле для ввода, нажатие левой кнопки мыши по полотну | Переключение режима рисования в режим рисования фигур, появление метапеременной с текстом внутри, заданным пользователем | Тест пройден |
| Метаконстанта | 24 | Нажатие сочетания клавиш Ctrl + 3 | Переключение режима рисования в режим рисования фигур | Тест пройден |

Таблица 5.1 – Продолжение

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Метаконстанта | 25 | Нажатие кнопки «Метаконстанта» | Переключение режима рисования в режим рисования фигур | Тест пройден |
| Метаконстанта | 26 | Нажатие кнопки «Метаконстанта», нажатие левой кнопки мыши по полотну | Переключение режима рисования в режим рисования фигур, появление метаконстанты | Тест пройден |
| Метаконстанта | 27 | Нажатие кнопки «Метаконстанта», ввод текста в поле для ввода, нажатие левой кнопки мыши по полотну | Переключение режима рисования в режим рисования фигур, появление метаконстанты с текстом внутри, заданным пользователем | Тест пройден |
| Линия | 28 | Нажатие сочетания клавиш Ctrl + 4 | Переключение режима рисования в режим рисования линии | Тест пройден |
| Линия | 29 | Нажатие кнопки «Линия» | Переключение режима рисования в режим рисования линии | Тест пройден |
| Линия | 30 | Нажатие кнопки «Метаконстанта», нажатие левой кнопки мыши по полотну | Переключение режима рисования в режим рисования линии, появление синей штрихпунктирной линии, которая тянется за курсором | Тест пройден |
| Линия | 31 | Нажатие кнопки «Метаконстанта», нажатие левой кнопки мыши по полотну, нажатие правой кнопки мыши | Переключение режима рисования в режим рисования линии, удаление синей штрихпунктирной линии | Тест пройден |
| Линия | 32 | Нажатие кнопки «Метаконстанта», начальная линия горизонтальная | Переключение режима рисования в режим рисования линии, появление горизонтальной линии со стрелкой на конце | Тест пройден |

Таблица 5.1 – Продолжение

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Линия | 33 | Нажатие кнопки «Метаконстанта», начальная линия вертикальная | Переключение режима рисования в режим рисования линии, появление вертикальной линии со стрелкой на конце и диагональным срезом вначале | Тест пройден |
| Линия | 34 | Нажатие кнопки «Метаконстанта», начальная линия горизонтальная, вторая линия вертикальная | Переключение режима рисования в режим рисования линии, появление горизонтальной линии со стрелкой посередине, вертикальной линии со стрелкой и диагональным срезом на конце | Тест пройден |
| Линия | 35 | Нажатие кнопки «Метаконстанта», начальная линия вертикальная, вторая линия горизонтальная | Переключение режима рисования в режим рисования линии, появление вертикальной линии с диагональным срезом вначале, горизонтальной линии со стрелкой на конце | Тест пройден |
| Линия | 36 | Нажатие кнопки «Метаконстанта», начальная линия горизонтальная, вторая линия вертикальная, третья линия горизонтальная | Переключение режима рисования в режим рисования линии, появление горизонтальной линии со стрелкой посередине, вертикальной линии, горизонтальной линии со стрелкой на конце | Тест пройден |

Таблица 5.1 – Продолжение

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Линия | 37 | Нажатие кнопки «Метаконстанта», начальная линия вертикальная, вторая линия горизонтальная, третья линия вертикальная | Переключение режима рисования в режим рисования линии, появление вертикальной линии с диагональным срезом в начале, горизонтальной линии с диагональным срезом посередине, вертикальной линии со стрелкой и диагональным срезом на конце | Тест пройден |
| Изменение размеров полотна | 38 | Задание новой ширины полотна, нажатие клавиши Enter | Полотно меняет свой размер, диалоговое окно меняет свой размер | Тест пройден |
| Изменение размеров полотна | 39 | Задание новой высоты полотна, нажатие клавиши Enter | Полотно меняет свой размер, диалоговое окно меняет свой размер | Тест пройден |
| Линии сетки | 40 | Нажатие сочетания клавиш Ctrl + W | Появление диалогового окна с вводом размера ячейки | Тест пройден |
| Линии сетки | 41 | Нажатие кнопки «Линии сетки» | Появление диалогового окна с вводом размера ячейки | Тест пройден |
| Линии сетки | 42 | Нажатие кнопки «Линии сетки», нажатие «Отмена» | Появление диалогового окна с вводом размера ячейки. Изменения с полотном не произошли | Тест пройден |
| Линии сетки | 43 | Нажатие кнопки «Линии сетки», задание размера ячейки, нажатие «ОК» | Появление диалогового окна с вводом размера ячейки. Появление на полотне сетки из параллельных сторонам полотна линий светло-серого цвета | Тест пройден |

Таблица 5.1 – Продолжение

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Линии сетки | 44 | Нажатие кнопки «Линии сетки», задание размера ячейки, нажатие «ОК», повторное нажатие кнопки «Линии сетки» | После повторного нажатия кнопки линии сетки с полотна удаляются | Тест пройден |
| Завершение программы | 45 | Нажатие на «Х» в правом верхнем углу окна | Появление предупреждающего окна, завершение программы | Тест пройден |

Все тесты прошли успешно. Сбои в работе программы при прохождении тестов не были обнаружены. Программа полностью исправна и готова к использованию.

# РУКОВОДСТВО ПО УСТАНОВКЕ И ИСПОЛЬЗОВАНИЮ

Чтобы начать использование программы, не нужно использовать установщик, достаточно запустить файл SyntaxDiagram.exe. После запуска программы появится диалоговое окно, представленное на рисунке 6.1.

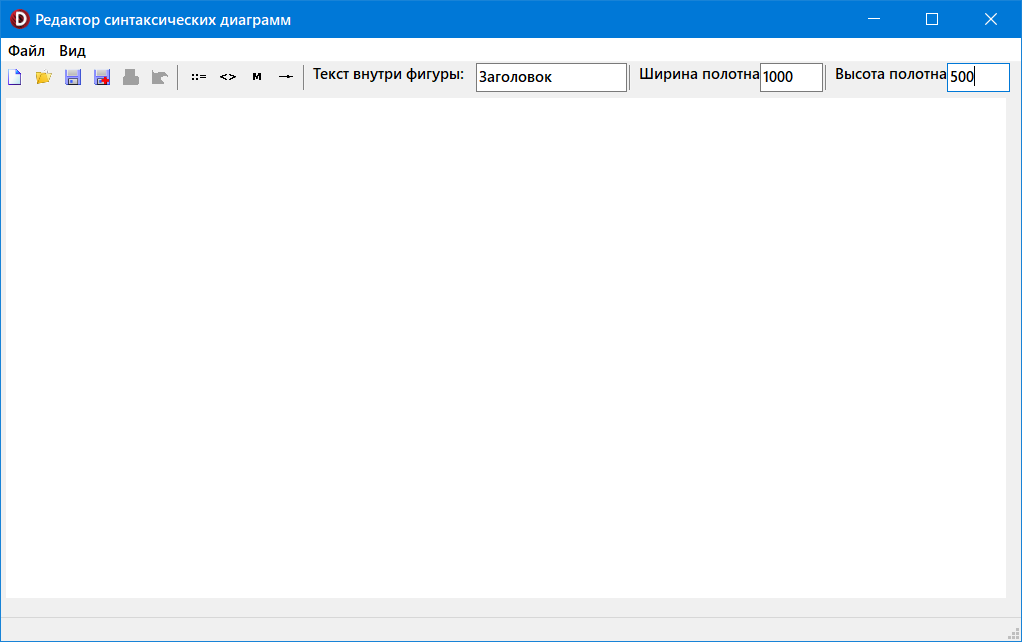


Рисунок 6.1 – Диалоговое окно программы

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).

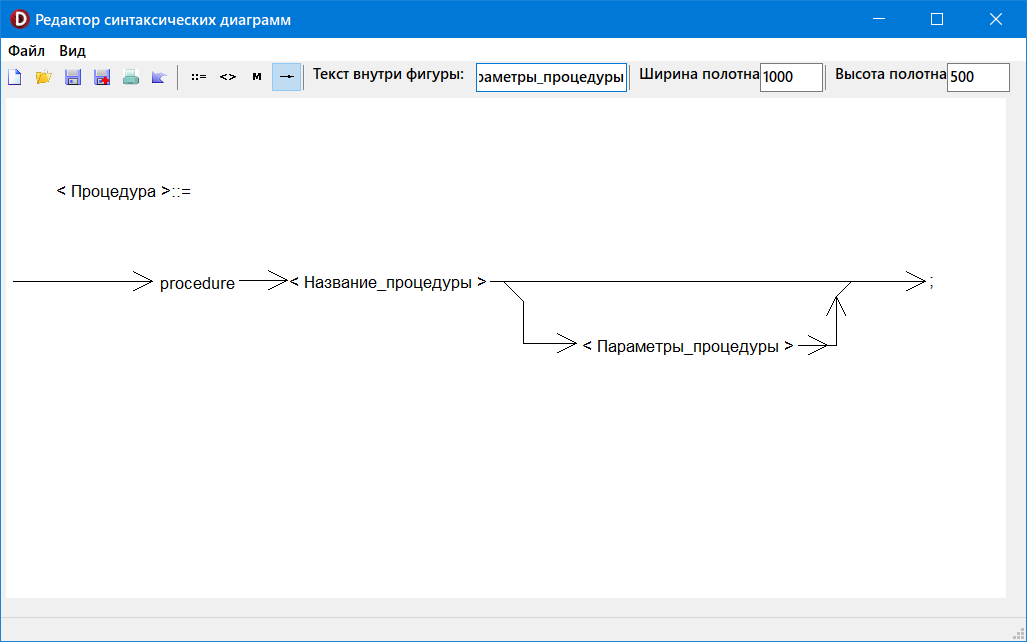


Рисунок 6.2 – Пример построенной синтаксической диаграммы

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.

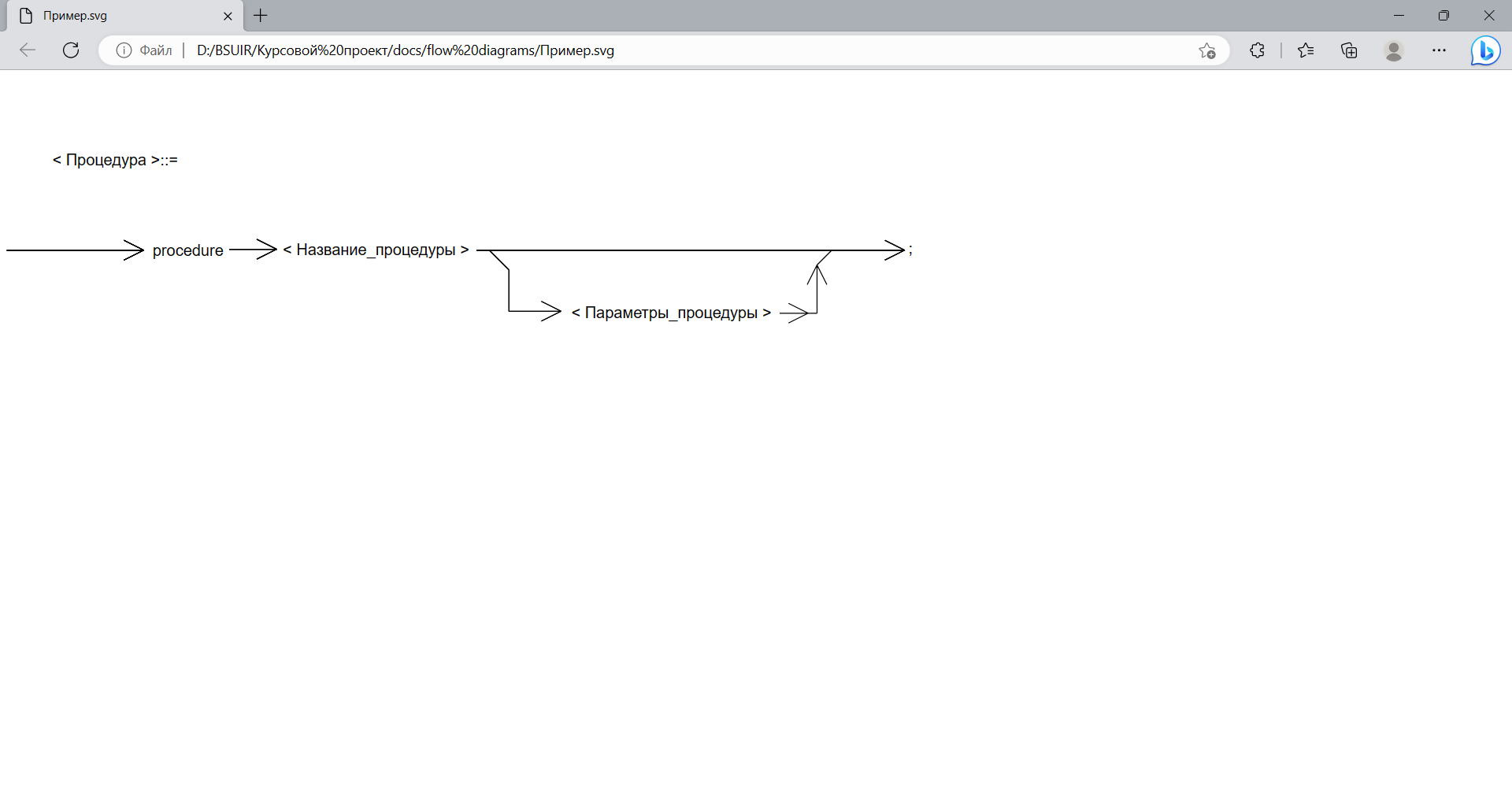


Рисунок 6.3 – Пример синтаксической диаграммы, экспортированной в SVG

# 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

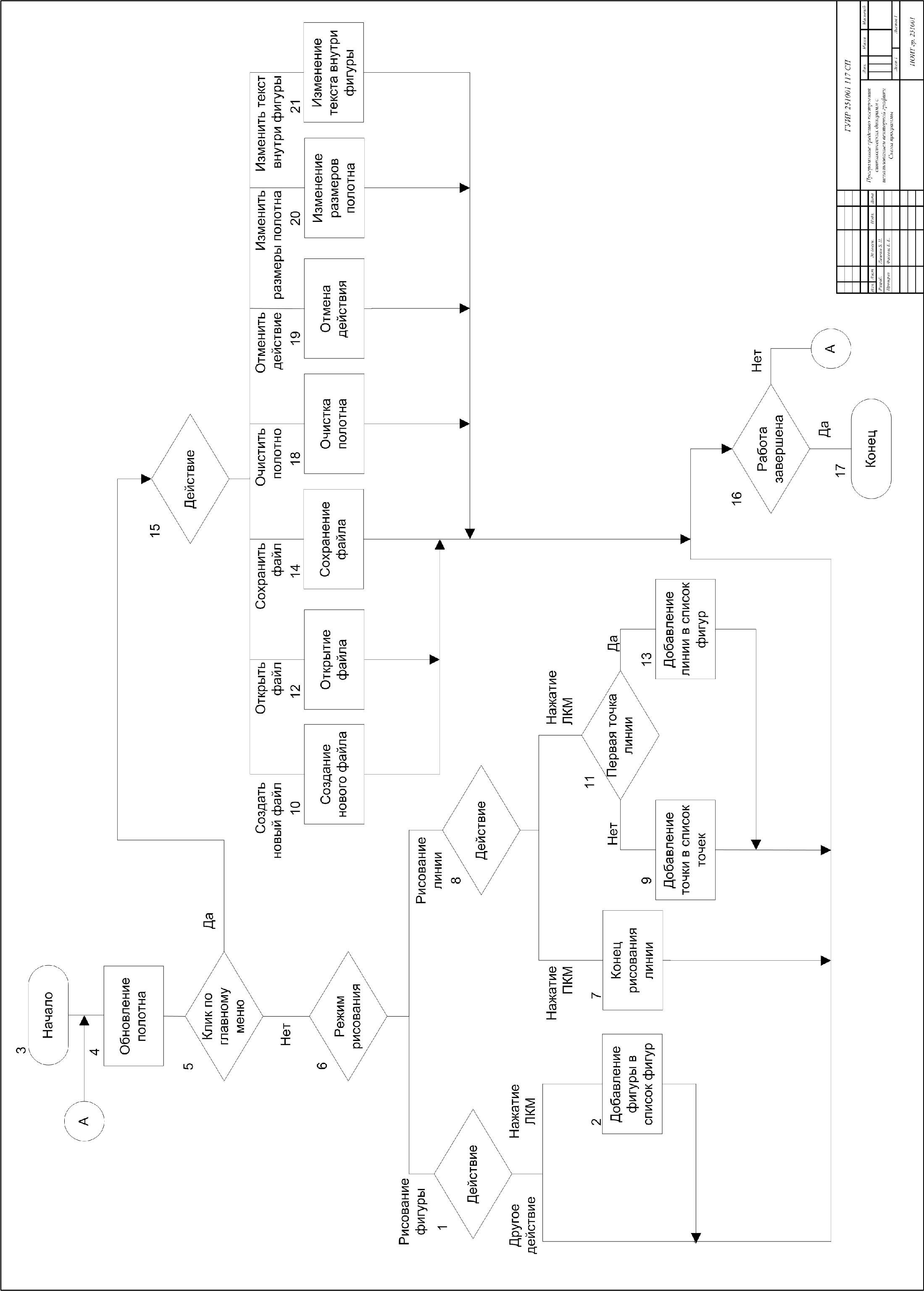


Рисунок А.1 – Схема программного средства

# 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.