Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

Дисциплина: Языки программирования (ЯП)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе

на тему

ВЕКТОРНЫЙ РЕДАКТОР СХЕМ АЛГОРИТМОВ

БГУИР КР 1-40 01 01 403 ПЗ

Студент: гр. 951004 Козко Р.С.

Руководитель: асс. Шостак Е. В.

Минск 2020

Учреждение образования

«Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

УТВЕРЖДАЮ

Заведующий кафедрой ПОИТ

––––––––––––––––––––––––

(подпись)

––––––––––––––––– 2020 г.

ЗАДАНИЕ

по курсовому проектированию

Студенту Козко Роману Сергеевичу

1. Тема работы Векторный редактор схем алгоритмов

2. Срок сдачи студентом законченной работы––23.12.2020 г.–––

3. Исходные данные к работе язык программирования C++

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

Введение.

1. Анализ прототипов, литературных источников и моделирование предметной области;

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. " Векторный редактор схем алгоритмов ", А1, схема программы, чертеж.

6. Консультант по курсовой работе

Шостак Е. В.

7. Дата выдачи задания 08.09.2020

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

раздел 1 к 15.09.2020 – 15 % готовности работы;

разделы 2, 3 к 15.10.2020 – 30 % готовности работы;

разделы 4, 5 к 15.11.2020 – 60 % готовности работы;

раздел 6 к 15.12.2020 – 90 % готовности работы;

оформление пояснительной записки и графического материала к 17.12.2020 – 100 % готовности работы.

Защита курсовой работы с 14.12.2020 по 23.12.2020 г.––––––––––––––––––––

РУКОВОДИТЕЛЬ–––––– Шостак Е. В.

(подпись)

Задание принял к исполнению 08.09.2020–––\_\_\_\_\_\_––

(дата и подпись студента)

Содержание

[Содержание 4](#_Toc58713159)

[Введение 6](#_Toc58713160)

[1 Анализ прототипов, литературных источников и моделирование предметной области 7](#_Toc58713161)

[1.1 Схемы алгоритмов 7](#_Toc58713162)

[1.2 Анализ существующих аналогов 8](#_Toc58713163)

[1.3 Постановка задачи 11](#_Toc58713164)

[2 Моделирование предметной области и разработка функциональных требований 12](#_Toc58713165)

[2.1 Описание функциональных требований 12](#_Toc58713166)

[2.2 Спецификация функциональных требований 14](#_Toc58713167)

[3 Проектирование программного средства 15](#_Toc58713168)

[3.2 Проектирование динамических структур данных 15](#_Toc58713170)

[3.3 Разработка алгоритма реакции на клик пользователя по полотну 16](#_Toc58713171)

[3.4 Разработка алгоритма присоединения линии к фигуре 17](#_Toc58713172)

[3.5 Структура модулей программы 18](#_Toc58713173)

[3.6 Описание модуля Сontroller 19](#_Toc58713174)

[3.7 Описание модуля FigureModel 21](#_Toc58713175)

[3.8 Описание модуля PictureView 26](#_Toc58713176)

[3.9 Описание модуля UndoStack 27](#_Toc58713177)

[3.10 Описание класса AbstractFigure 29](#_Toc58713178)

[3.11 Описание класса LineFunc 29](#_Toc58713179)

[3.12 Описание класса RectFunc 29](#_Toc58713180)

[3.13 Описание модуля RectFunctions 29](#_Toc58713181)

[4 Тестирование, проверка работоспособности и анализ полученных результатов 30](#_Toc58713182)

[4.1 Тестирование функционала добавления фигур 30](#_Toc58713183)

[4.2 Тестирование функционала редактирования текстовых фигур и перемещения линий 32](#_Toc58713184)

[4.3 Тестирование функционала отмены изменений 33](#_Toc58713185)

[4.4 Тестирование прочего функционала программного средства 34](#_Toc58713190)

[4.5 Вывод из прохождения тестирования 35](#_Toc58713196)

[5 Руководство по установке и использованию 36](#_Toc58713197)

[Заключение 39](#_Toc58713198)

[Список литературы 40](#_Toc58713199)

[Приложение A 41](#_Toc58713200)

[Приложение В 42](#_Toc58713201)

Введение

Для того, чтобы визуально отобразить алгоритм и упростить его понимание, программисту необходимо построить его схему.

Для построения схем алгоритмов применяются разные нотации. Наиболее известные из них:

* ГОСТ 19.701-90;
* Диаграмма Насси — Шнейдермана;
* Метод Дамке.

ГОСТ 19.701-90 является наиболее популярным и позволяет отображать любые алгоритмы в отличие от своих конкурентов. Также он является более компактным.

Цель данной курсовой работы – создать векторный редактор схем алгоритмов, ориентированный на ГОСТ 19.701-90.

В ходе выполнения курсовой работы я постараюсь найти решение таким техническим вопросам как:

1. Как реализовать работу с графикой в языке программирования C++;
2. Как спроектировать наиболее удобный пользовательский интерфейс, графического редактора;

В этой пояснительной записке отображены следующие этапы написания курсовой работы:

1. Анализ прототипов, литературных источников и моделирование предметной области;
2. Анализ требований к программному средству и разработка функциональных требований;
3. Проектирование программного средства;
4. Тестирование, проверка работоспособности и анализ полученных результатов;
5. Руководство по установке и использованию.
6. Анализ прототипов, литературных источников и моделирование предметной области
   1. Схемы алгоритмов

Схема алгоритма — схема, описывающая алгоритмы или процессы, в которых отдельные шаги изображаются в виде блоков различной формы, соединенных между собой линиями, указывающими направление последовательности.

При описании алгоритма используются следующие основные элементы:

* **Процесс** — символ отображает функцию обработки данных любого вида (выполнение определённой операции или группы операций, приводящее к изменению значения, формы или размещения информации, или к определению, по которому из нескольких направлений потока следует двигаться).
* **Данные (ввод - вывод)** — символ отображает данные, носитель данных не определён.
* **Решение** — символ отображает решение или функцию переключательного типа, имеющую один вход и ряд альтернативных выходов, один и только один из которых может быть активизирован после вычисления условий, определённых внутри этого символа. Соответствующие результаты вычисления могут быть записаны по соседству с линиями, отображающими эти пути.
* **Терминатор** — символ отображает вход из внешней среды и выход во внешнюю среду (начало или конец схемы программы, внешнее использование и источник или пункт назначения данных).
* **Соединитель** — символ отображает выход в часть схемы и вход из другой части этой схемы и используется для обрыва линии и продолжения её в другом месте. Соответствующие символы-соединители должны содержать одно и то же уникальное обозначение.
* **Линия** — символ отображает поток данных или управления.

Также могут использоваться и другие элементы, но их можно отнести к дополнительным, так как любой алгоритм можно представить с помощью описанных выше.

* 1. Анализ существующих аналогов

Данная курсовая работа ориентирована на создание векторного графического редактора схем алгоритмов. Для того, чтобы иметь представление о функционале молекулярного редактора, рассмотрим уже существующие в качестве прототипов.

* + 1. Приложение Microsoft Visio

**Microsoft Visio** — векторный графический редактор блок-схем для Windows. Пример интерфейса представлен на рисунке 1.1.

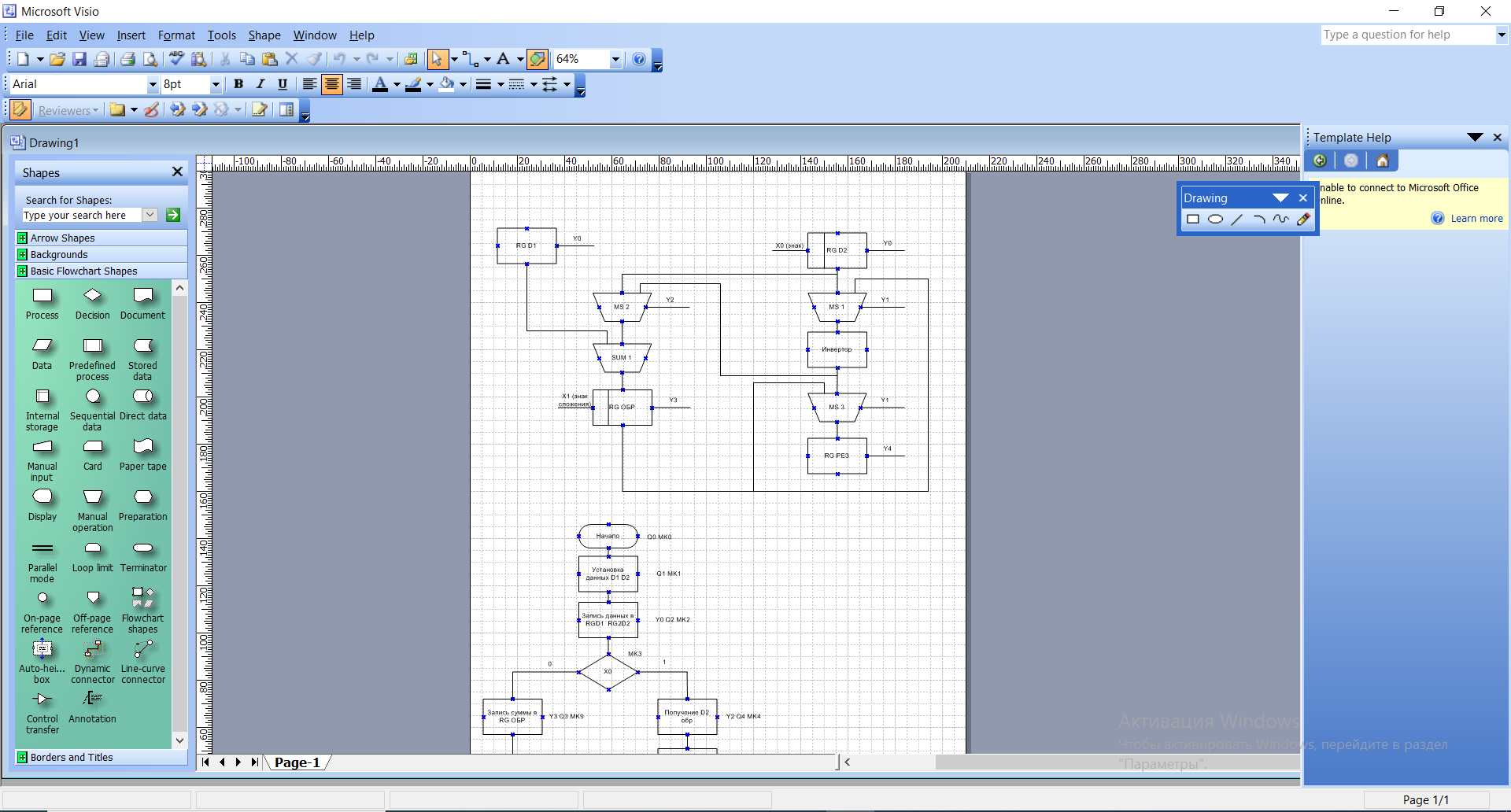


Рисунок 1.1 – Скриншот Microsoft Visio

Преимущества:

* Стабильная работа программы.
* Удобный процесс сохранения.

Недостатки:

* Устаревший пользовательский интерфейс.
  + 1. Adobe Illustrator

**Adobe Illustrator** – один из наиболее распространенных векторных графических редакторов. Разрабатывается и распространяется компанией Adobe Systems. Интерфейс программы представлен на рисунке 1.2.

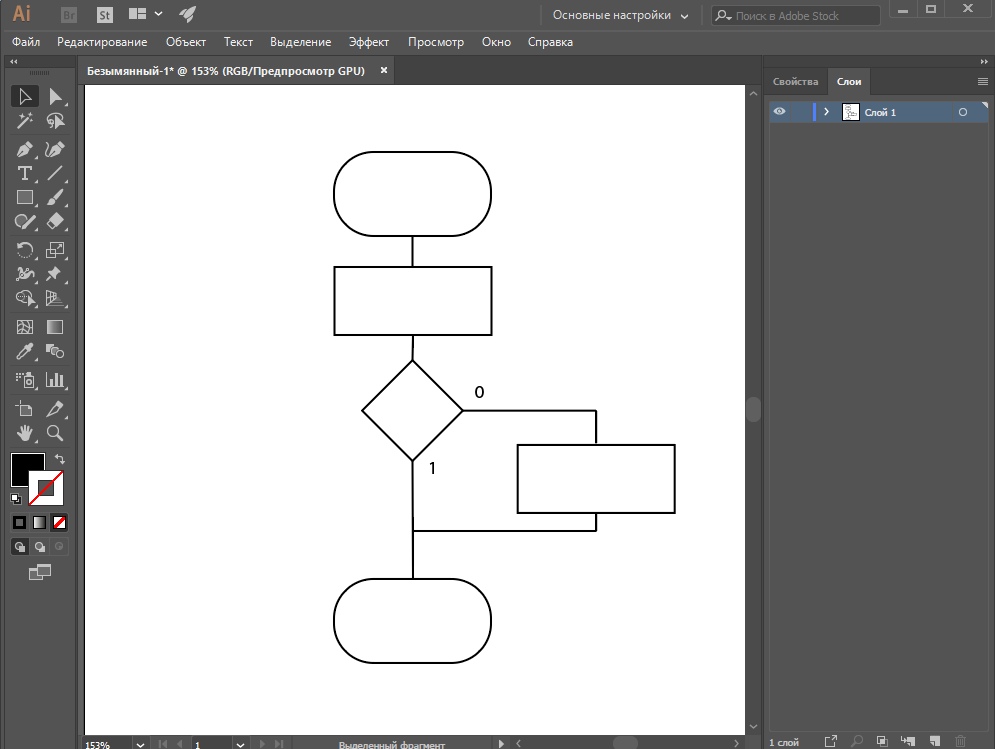


Рисунок 1.2 – Скриншот Adobe Illustrator

Преимущества:

* Удобный пользовательский интерфейс.
* Стабильная работа программы.
* Удобный процесс экспортa.

Недостатки:

* Высокая стоимость.
* Не предназначен для редактирования схем алгоритмов.
  + 1. Приложение CorelDRAW

**CorelDRAW** — графический редактор, разработанный корпорацией Corel. Интерфейс программы представлен на рисунке 1.3.

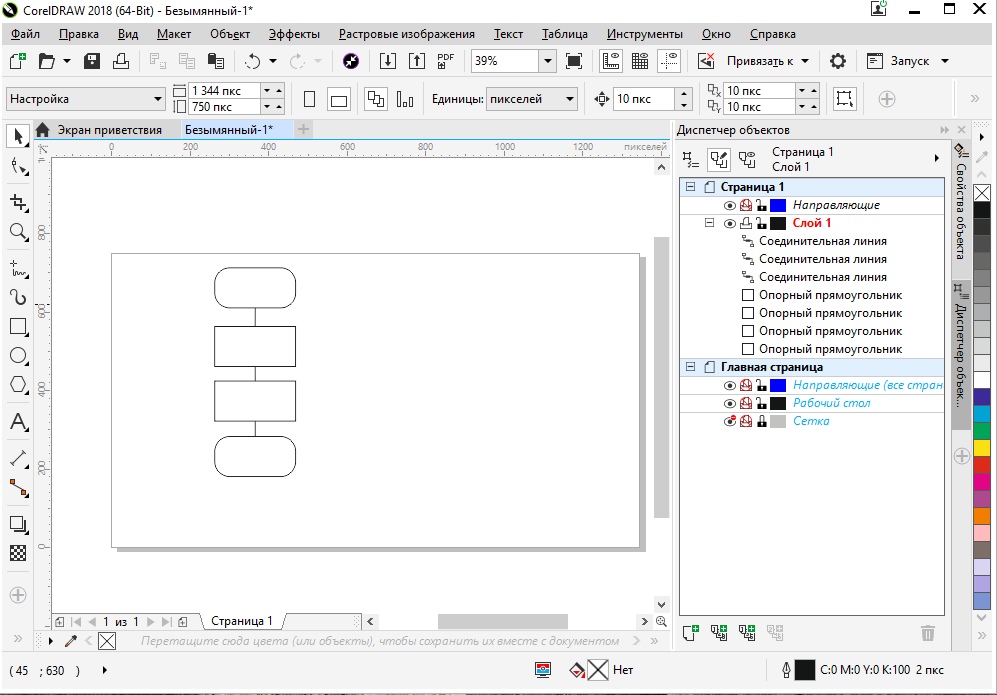


Рисунок 1.3 – Скриншот CorelDraw

Преимущества:

* Стабильная работа.
* Удобный пользовательский интерфейс.

Недостатки:

* Высокая стоимость.
* Неудобен для схем алгоритмов.
  1. Постановка задачи

Так как проектируемое программное средство предназначено для создания схем алгоритмов, необходимо создать максимально удобный интерфейс для взаимодействия с пользователем. Программа должна уметь сама рисовать элементы схемы алгоритма: действие, данные, выбор и другие.

Чтобы программу можно было считать векторным редактором, должны быть реализованы следующие функции:

* + Рисование с помощью установленных примитивных фигур;
  + Изменение размеров, положения нарисованных фигур (редактирование изображения);
  + Сохранение исходников изображения в типизированный файл с возможностью дальнейшего использования;
  + Открытие типизированного файла с исходниками;
  + Сохранение изображения в растровый формат (bmp);
  + Операции отмены действия, копирования и вставки;

Для разработки я выбрал среду Visual Studio. Данная среда является бесплатной, она позволяет разрабатывать графический пользовательский интерфейс с помощью открытой библиотеки Windows Forms. В качестве языка программирования выбран C++ в связи с поддержкой парадигмы ООП и упрощенной работы с динамическими массивами и строковым типом данных.

1. Моделирование предметной области и разработка функциональных требований
   1. Описание функциональных требований

Программное средство должно представлять переключение режимов рисования путем нажатия соответствующей иконки в ToolStrip.

Режим рисования может принимать один из следующих значений:

* Рисование
  + Процесс
  + Данные
  + Терминатор
  + Решение
  + Соединитель
  + Текст
* Указатель
* Рисование линии

В зависимости от выбранного режима пользователю должны представляться следующие возможности взаимодействия:

* Режим рисования фигур:
  + При клике по полотну программа должна отобразить выбранную фигуру по координатам клика.
* Режим рисования линий:
  + Первый клик левой кнопкой мыши по полотну должен активировать режим рисования линий. Каждое следующее нажатие левой кнопки мыши должно добавлять новую точку и соединить ее с предыдущей точкой данной линии. Нажатие правой кнопки мыши должно прекратить рисование линии.
* Режим редактирования:
  + При перемещении мыши курсор меняется в зависимости от того, на какую область фигуры он наведен и выделена ли фигура:
    - Вершина фигуры
    - Сторона фигуры
    - Центр фигуры
  + При зажатии мыши по невыделенной фигуре в данном режиме, она(фигура) должна перемещаться при перемещении курсора.
  + При выделении и зажатии мыши в этом режиме, фигура должна редактироваться по следующему принципу
    - Зажата в центре – при перемещении курсора перемещается вся фигура.
    - Зажата на вершине – при перемещении курсора перемещается вершина и стороны, которым принадлежит вершина.
    - Зажата сторона фигуры – при перемещении курсора перемещается сторона.
  + Клик по фигуре должен выделять фигуру, по которой был произведен клик. При нажатии клавиши «delete» должна удаляться выделенная фигура. Сочетания «Ctrl + C» должно помещать в специальный «программный буфер обмена» выделенную фигуру. Сочетания «Ctrl + V» - извлекать из буфера фигуру и отображать на полотне. Сочетание «Ctrl + Z» должно «откатывать» на предыдущее действие, то есть отменять последнее изменение. Если фигура (не линия) выделена, то если изменить текст в отведенном текстовом поле и нажать на поле, текст внутри фигуры должен измениться на введенный.

Также программа должна предоставлять взаимодействие с меню. В зависимости от того, по какому элементу меню был произведен клик, программа должна уметь:

* Создавать новый файл.
* Сохранять исходный файл:
  + В специальный формат.
  + В растровый формат.
* Открывать исходный файл:
  + Из специального формата.
  1. **Спецификация функциональных требований**

Главное среди функциональных требований - «Отображение фигур на полотне».

Спецификация данной функции имеет следующий вид:

* Прямоугольные фигуры имеют прозрачный фон и обводку, вершины обозначаются маленькими квадратами. Текст вписан в прямоугольник с вертикальным и горизонтальным центрированием.
* Линии должны проходить через все точки, заданные пользователем путем нажатия по полотну, в заданном пользователем порядке (порядок задается порядком нажатия).
* В конце каждой горизонтальной линии проводится стрелка.

1. Проектирование программного средства
2. 1. Проектирование динамических структур данных

В первую очередь, необходимо хранить фигуры. Так как пользователь будет постоянно добавлять и удалять фигуры, удобно будет использовать динамические списков. Однако нужно предусмотреть следующие особенности:

* Фигуры с разным способом описания.
* У линий пользователь может добавлять точки.
* Информацию о фигурах нужно сохранять в файл.

Поэтому было принято решение использовать для хранения информативной части списка структуру с абстрактным классом для совместимости с линиями и прямоугольными фигурами.

От абстрактного класса я унаследовал класс для линий и еще один абстрактный класс для прямоугольных фигур. Далее определил классы-реализаторы абстрактного класса фигур. Диаграмма классов представлена на рисунке 3.1.

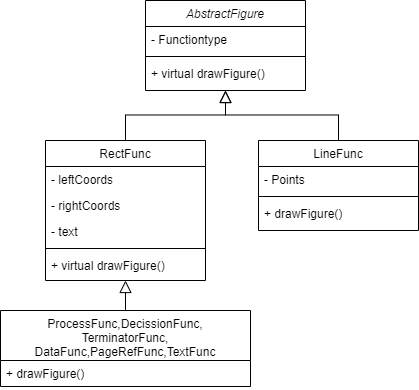


Рисунок 3.1 – диаграмма классов

Так как линии хранят в себе указатель на список точек линии, нужно придумать, как сохранять это в файл. Для этого используется структура односвязного динамического списка вершин. При такой структуре координаты преобразуются в специальный текстовый формат, а при открытии файла, декодируются и преобразуются обратно – в список точек.

Для того, чтобы предусмотреть возможность отмены действий, необходимо хранить каждое изменение. Для этого лучше всего подходит стек, т.к. нам всегда надо последнее действие. При отмене изменений будет извлекаться вершина и вершина будет перемещаться к предыдущему элементу.

Так как отменяемые действия могут быть различные, то я дополнительно храню тип действия, которое надо отменить.

Необходимо предусмотреть сохранение следующих изменений:

* Удаление фигуры.
* Добавление точки линии.
* Добавление фигур.
* Перемещение фигуры/изменение размеров фигуры
  1. Разработка алгоритма реакции на клик пользователя по полотну

После клика по полотну, необходимо обработать событие. Результат обработки зависит от текущего режима и выбранной фигуры. Режимы рисования и редактирования записаны в отдельные перечислимые типы. (Для рисования: Рисование, нет рисования, рисование линии. Для редактирования: нет редактирования, перемещение и различные редактирования размера). Если текущий режим не редактирование и не выбран указатель, то будет добавлена фигура/линия (первая точка). Если текущий режим рисование линии, то будет добавлена еще одна точка линии. Если выбран указатель, то по координате нажатия будет найдена фигура, которая была выбрана. Схема алгоритма представлена на рисунке 3.2.



Рисунок 3.2 – схема алгоритма реакции на клик пользователя по полотну

* 1. Разработка алгоритма присоединения линии к фигуре

Для упрощения рисования синтаксических диаграмм, я решил добавить возможность автоматического присоединения.

Если начальная или конечная точка линии находится очень близко к текстовой фигуре, координаты этой точки должны измениться так, чтобы точка оказалась в центре ближайшего края фигуры.

После изменения координат точки, необходимо выровнять линию. Алгоритм выравнивания линии производит перемещение всех точек, которые были до этого равны по одной из координат точке, которая переместилась.

Два алгоритма расположены на рисунке 3.3.



Рисунок 3.3 – Алгоритм проверки положения точек и выравнивания линий

* 1. Структура модулей программы

Программа состоит из следующих файлов:

1. Файлы модуля приложения Сontroller.h и Сontroller.cpp;
2. Файлы модуля класса AbstractFigure.h и AbstractFigure.cpp;
3. Файлы модуля класса LineFunc.h и LineFunc.cpp;
4. Файлы модуля класса RectFunc.h и RectFunc.cpp;
5. Файлы модуля класса RectFunctions.h и RectFunctions.cpp;
6. Файлы модуля основных типов DataTypes.h и DataTypes.cpp;
7. Файлы модуля для работы с основными списками FigureModel.h и FigureModel.cpp;
8. Файлы модуля для отрисовки PictureView.h и PictureView.cpp;
9. Файлы модуля для работы со стеком UndoStack.h и UndoStack.cpp;
   1. Описание модуля Сontroller

Модуль Сontroller является модулем для отображения окна программы и реакции на действия пользователя. Описание основных подпрограмм, находящихся в модуле Сontroller приведено в таблице 3.1.

Таблица 3.1 – основные подпрограммы модуля Сontroller

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Имя подпрограммы** | **Описание** | **Заголовок метода** | **Имя параметра** | **Назначение параметра** |
| saveBMP | Сохранение картинки в формате bmp | void saveBMP(System::String^ path) | path | Путь к файлу |
| actNew | Создание нового файла | void actNew() | - | - |
| actOpen | Открытие файла | void actOpen() | - | - |
| stopDrawLine | Остановка рисование линии | void stopDrawLine() | - | - |
| Controller\_Load | Загрузка формы | System::Void Controller\_Load(System::Object^ sender, System::EventArgs^ e) | sender | Объект, который сгенерировал событие |
| e | Аргумент для событий |

Продолжение таблицы 3.1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Имя подпрограммы** | **Описание** | **Заголовок метода** | **Имя параметра** | **Назначение параметра** |
| changeButtonState | Изменение состояния кнопки | void changeButtonState(FunctionType type) | type | Тип кнопки |
| changeCursor | Изменение курсора | void changeCursor(EditMode mode) | mode | Режим редактирования |
| pictureBox\_MouseUp | Обработка отпускания кнопки мыши. Вызов выравнивания фигур | System::Void pictureBox\_MouseUp(System::Object^ sender, System::Windows::Forms::MouseEventArgs^ e) | sender | Объект, который сгенерировал событие |
| e | Аргумент для событий |
| pictureBox\_MouseMove | Обработка движения мыши. Внутри обработчика вызывается редактирование фигур/линий, траектория линии | System::Void pictureBox\_MouseMove(System::Object^ sender, System::Windows::Forms::MouseEventArgs^ e) | sender | Объект, который сгенерировал событие |
| e | Аргумент для событий |
| pictureBox\_MouseDown | Обработка нажатие мыши.  Внутри обработчика – в зависимости от режима вызывается добавление линии/фигуры | System::Void pictureBox\_MouseDown(System::Object^ sender, System::Windows::Forms::MouseEventArgs^ e) | sender | Объект, который сгенерировал событие |
| e | Аргумент для событий |

Продолжение таблицы 3.1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Имя подпрограммы** | **Описание** | **Заголовок метода** | **Имя параметра** | **Назначение параметра** |
| FileExitMenuItem\_Click | Обработка нажатия на панель инструментов | System::Void FileExitMenuItem\_Click(System::Object^ sender, System::EventArgs^ e) | sender | Объект, который сгенерировал событие |
| e | Аргумент для событий |
| pictureBox\_DoubleClick | Обработка нажатия двойного щелчка | System::Void pictureBox\_DoubleClick(System::Object^ sender, System::EventArgs^ e) | sender | Объект, который сгенерировал событие |
| e | Аргумент для событий |
| Controller\_KeyDown | Обработка нажатий клавиш | System::Void Controller\_KeyDown(System::Object^ sender, System::Windows::Forms::KeyEventArgs^ e) | sender | Объект, который сгенерировал событие |
| e | Аргумент для событий |
| pictureBox\_Paint | Обработчик события перерисовки. Внутри – перерисовываются фигуры | System::Void pictureBox\_Paint(System::Object^ sender, System::Windows::Forms::PaintEventArgs^ e) | sender | Объект, который сгенерировал событие |
| e | Аргумент для событий |

* 1. Описание модуля FigureModel

Модуль FigureModel является модулем, содержащий основную бизнес-логику для работы то структурами данных. Описание основных подпрограмм, находящихся в модуле FigureModel приведено в таблице 3.2.

Таблица 3.2 – основные подпрограммы модуля FigureModel

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Имя подпрограммы** | **Описание** | **Заголовок метода** | **Имя параметра** | **Назначение параметра** |
| initList | Создание списка фигур | List^ initList(List^ head) | head | Указатель на голову списка |
| cleanList | Очистка списка | void cleanList(List^ head) | head | Указатель на голову списка |
| isElementInList | Проверка существование элемента в списке | bool isElementInList(List^ head , List^ el) | head | Указатель на голову списка |
| el | Существующая фигура |
| push | Добавление элемента в список | List^ push(List^ head, FunctionType fType, System::Drawing::Point points) | head | Указатель на голову списка |
| fType | Тип фигуры |
| points | Координаты цента фигуры |
| pushExistingFunc | Добавление существующего элемента в список | List^ pushExistingFunc(List^ head, AbstractFigure^ element) | head | Указатель на голову списка |
| element | Существующая фигура |

Продолжение таблицы 3.2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Имя подпрограммы** | **Описание** | **Заголовок метода** | **Имя параметра** | **Назначение параметра** |
| deleteElement | Логическое удаление элемента из списка | void deleteElement(List^ head, List^ elementToDelete) | head | Указатель на голову списка |
| elementToDelete | Существующая фигура |
| figureFromType | Создание фигуры по её типу | AbstractFigure^ figureFromType(FunctionType fType, System::Drawing::Point points) | fType | Тип фигуры |
| points | Центр фигуры |
| addLine | Добавление линии | List^ addLine(List^ head, System::Drawing::Point points) | head | Указатель на голову списка |
| points | Точка линии |
| addPoints | Добавление точек к существующей линии | Points\* addPoints(LineFunc^ line, System::Drawing::Point points) | line | Текущая линия |
| points | Точка линии |
| hitTest | Метод-события при нажатии на кнопку «Йод» на панели инструментов | List^ hitTest(List^ head, System::Drawing::Point points) | head | Указатель на голову списка |
| points | Точка клика |

Продолжение таблицы 3.2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Имя подпрограммы** | **Описание** | **Заголовок метода** | **Имя параметра** | **Назначение параметра** |
| getSelectedFuncFromRect | Поиск фигуры в области выделения | void getSelectedFuncFromRect(List^ head, List^ selectedHead, Rect rect)) | head | Указатель на голову списка |
| selectedHead | Указатель на голову списка выбранных фигур |
| rect | Прямоугольник выделения |
| transformFunc | Трансформирование функции | void transformFunc(List^ cur, System::Drawing::Point startPoints, System::Drawing::Point endPoints, EditMode mode) | cur | Текущая фигура |
| startPoints | Начальные точки |
| endPoints | Конечные точки |
| mode | Режим изменения |
| getCurEditMode | Выбор текущего режима редактирование | EditMode getCurEditMode(List^ head,List^ selectedFunc, System::Drawing::Point points) | head | Указатель на голову списка |
| selectedFunc | Выбранная фигура |
| points | Текущее положение мышки |

Продолжение таблицы 3.2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Имя подпрограммы** | **Описание** | **Заголовок метода** | **Имя параметра** | **Назначение параметра** |
| saveToFile | Сохранение файла в спец. формат | void saveToFile(List^ head, System::String^ path) | head | Указатель на голову списка |
| path | Путь к файлу |
| readFromFile | Чтение файла из спец. формата | bool readFromFile(List^ head, System::String^ path) | head | Указатель на голову списка |
| path | Путь к файлу |
| makePointOnCenter | Примагничивание к центру фигуры | void makePointOnCenter(List^ head, List^ curL) | head | Указатель на голову списка |
| curL | Текущая фигура |
| makeSmoothLine | Выравнивание линии | void makeSmoothLine(List^ curLine, System::Drawing::Point newPoints, System::Drawing::Point prevPoints) | curLine | Текущая линия |
| newPoints | Новое значение координат |
| prevPoints | Старое значение координат |

* 1. Описание модуля PictureView

Модуль PictureView является модулем, содержащий всю вспомогательную отрисовку. Описание основных подпрограмм, находящихся в модуле PictureView приведено в таблице 3.3.

Таблица 3.3 – основные подпрограммы модуля PictureView

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Имя подпрограммы** | **Описание** | **Заголовок метода** | **Имя параметра** | **Назначение параметра** |
| cleanScreen | Очистка экрана | void cleanScreen(System::Windows::Forms::PaintEventArgs^ e, int width, int height) | e | Аргумент для событий |
| width | Ширина окна |
| height | Длина окна |
| drawFunctions | Вызов рисования фигур | void drawFunctions(System::Windows::Forms::PaintEventArgs^ e, List^ head,bool isVertex) | e | Аргумент для событий |
| head | Указатель на голову списка |
| isVertex | Флаг вершин |
| drawPath | Рисование пути линии | void drawPath(System::Windows::Forms::PaintEventArgs^ e, LineFunc^ line, System::Drawing::Point points) | e | Аргумент для событий |
| line | Текущая линяя |
| points | Текущее положение мышки |
| drawSelectedRect | Отрисовка прямоугольника выделения | void drawSelectedRect(System::Windows::Forms::PaintEventArgs^ e, Rect rect) | event | Инкапсуляция события |
| rect | Прямоугольник выделения |

Продолжение таблицы 3.3

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Имя подпрограммы** | **Описание** | **Заголовок метода** | **Имя параметра** | **Назначение параметра** |
| drawVertex | Отрисовка вершины | void drawVertex(System::Windows::Forms::PaintEventArgs^ e,System::Drawing::Point points, System::Drawing::Color color) | e | Аргумент для событий |
| points | Координаты точки |
| color | Цвет |
| repaintSelectedFunc | Перерисовка вершин | void repaintSelectedFunc(System::Windows::Forms::PaintEventArgs^ e, List^ selectedFunc,bool forWrite) | e | Аргумент для событий |
| selectedFunc | Выбранная функция |
| forWrite | Флаг текста |
| drawText | Отрисовка текста в фигуре | void drawText(System::Windows::Forms::PaintEventArgs^ e,RectFunc^ cur) | e | Аргумент для событий |
| cur | Текущая фигура |

* 1. Описание модуля UndoStack

Модуль UndoStack является модулем, содержащий всю работу со стеком для отмены действий. Описание основных подпрограмм, находящихся в модуле UndoStack приведено в таблице 3.4.

Таблица 3.4 – основные подпрограммы модуля UndoStack

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Имя подпрограммы** | **Описание** | **Заголовок метода** | **Имя параметра** | **Назначение параметра** |
| stackInit | Создание стека | UndoStack^ stackInit(UndoStack^ head) | head | Указатель на голову стека |
| stackIsEmpty | Проверка пустоты стека | bool stackIsEmpty(UndoStack^ head) | head | Указатель на голову стека |
| undoStackAction | Отмена изменений | void undoStackAction(UndoStack^ cur, List^ listHead) | cur | Текущий элемент стека |
| listHead | Указатель на голову списка |
| cleanStack | Очистка стека | void cleanStack(UndoStack^ head) | head | Указатель на голову стека |
| popStack | Изъятие из стека | UndoStack^ popStack(UndoStack^ head) | head | Указатель на голову стека |
| pushStack | Добавление в стек | void pushStack(UndoStack^ head, List^ figure, UndoTypes type,System::String^ str) | head | Указатель на голову стека |
| figure | Текущая фигура |
| str | Строка для запоминания |

* 1. Описание класса AbstractFigure

Класс AbstractFigure является базовым абстрактным классом. Основная его цель – обеспечить совместимость линий и прямоугольных фигур в списке.

* 1. Описание класса LineFunc

Класс LineFunc является классом для хранения и отрисовки линий. Единственный его метод выполняет отрисовку линии и пририсовку стрелки.

* 1. Описание класса RectFunc

Класс RectFunc является абстрактным классом-наследником AbstractFigure. Он необходим для реализации обратной совместимости всех прямоугольных фигур.

* 1. Описание модуля RectFunctions

Модуль RectFunctions содержит в себе классы-реализаторы RectFunc. Для каждого класса реализован конструктор с первичной настройкой экземпляра, а также метод для отрисовки самой фигуры.

1. Тестирование, проверка работоспособности и анализ полученных результатов

В данном разделе я остановлюсь на тестировании программного средства. Тестирование было совершено на компьютере с установленной операционной системой Windows 10.

* 1. Тестирование функционала добавления фигур

Таблица 4.1 – тестирование функционала добавления фигур

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **№** | **Тестируемая функциональ-ность** | **Последовательность действий** | **Ожидаемый результат** | **Полученный результат** |
| 1 | Добавление фигуры процесса | 1.Выбор текущей фигурой «Процесс» (3 фигура).   1. 2.Кликнуть по полотну. | Добавится прямоугольная фигура |  |
| 2 | Добавление фигуры выбора | 1.Выбор текущей фигурой «Выбор» (4 фигура).   1. 2.Кликнуть по полотну. | Добавится ромбовидная фигура |  |
| 3 | Добавление фигуры данных | 1.Выбор текущей фигурой «Данные» (5 фигура).   1. 2.Кликнуть по полотну. | Добавится параллелограм |  |

Продолжение таблицы 4.1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **№** | **Тестируемая функциональ-ность** | **Последовательность действий** | **Ожидаемый результат** | **Полученный результат** |
| 4 | Добавление фигуры «Терминатор» | 1.Выбор текущей фигурой «Терминатор» (6 фигура).   1. 2.Кликнуть по полотну. | Добавится фигура, состоящая из двух прямых и двух полукругов |  |
| 5 | Добавление фигуры «Соединитель» | 1.Выбор текущей фигурой «Соединитель» (7 фигура).   1. 2.Кликнуть по полотну. | Добавится круглая фигура |  |
| 6 | Добавление фигуры «Текст» | 1.Выбор текущей фигурой «Текст» (7 фигура).   1. 2.Кликнуть по полотну. | Добавится пустая фигура |  |
| 7 | Добавление линии | 1.Выбор текущей фигурой «Линия» (2 фигура).  2.Кликнуть по полотну (ЛКМ).  3. Кликнуть второй раз по полотну на одном уровне по оси OX с прошлой точкой (ЛКМ).  4. Нажать ПКМ. | Добавится линия из двух точек. |  |

* 1. Тестирование функционала редактирования текстовых фигур и перемещения линий

Таблица 4.2 – тестирование функционала редактирования текстовых фигур и перемещения линий

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **№** | **Тестируемая функциональ-ность** | **Последовательность действий** | **Ожидаемый результат** | **Получен-ный результат** |
| 1 | Перемещение фигуры | 1. Переключиться на «указатель».  2. Навести курсор на центр фигуры или линию.  3. Зажать левую кнопку мыши.  4. Переместить курсор.  5. Отпустить курсор. | Фигура переместится | Тест пройден |
| 2 | Изменение размеров фигуры путем перетаскивания вершины | 1. Переключиться на «указатель».  2. Кликнуть по текстовой фигуре.  3. Навести курсор на вершину фигуры.  4. Зажать левую кнопку мыши.  5. Переместить курсор.  6. Отпустить курсор. | Переместится вершина, и, следовательно, изменится размер фигуры | Тест пройден |
| 3 | Перемещение стороны фигуры | 1. Переключиться на «указатель».  2. Кликнуть по текстовой фигуре.  3. Навести курсор на сторону фигуры.  4. Зажать левую кнопку мыши.  5. Переместить курсор.  6. Отпустить курсор. | Сторона передвигается и меняется размер фигуры | Тест пройден |
| 4 | Изменение текста внутри фигуры | 1. Переключиться на «указатель»  2. Кликнуть по текстовой фигуре 2 раза.  3. Вводить текст. | Текст фигуры изменится | Тест пройден |

Продолжение таблицы 4.2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **№** | **Тестируемая функциональ-ность** | **Последовательность действий** | **Ожидаемый результат** | **Получен-ный результат** |
| 5 | Копирование и вставка фигуры | 1. Нарисовать фигуры и линии.  2. Скопировать всё.  3. Вставить всё. | Все выбранные фигуры и линии вставились | Тест пройден |
| 6 | Удаление фигуры | 1. Переключиться на «указатель».  2. Кликнуть по текстовой фигуре или линии.  3. Нажать клавишу «Delete». | Фигура удалится | Тест пройден |

* 1. Тестирование функционала отмены изменений

Таблица 4.3 – тестирование функционала отмены изменений

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **№** | **Тестируемая функциональ-ность** | **Последовательность действий** | **Ожидаемый результат** | **Получен-ный результат** |
| 1 | Отмена добавления фигуры/линии | 1. Добавить фигуру или линию.  2. Отменить изменение. | Добавленная фигура/линия удалится | Тест пройден |
| 2 | Отмена удаления фигуры/линии | 1. Добавить фигуру/линию.  2. Удалить фигуру/линию.  3. Отменить изменение. | Фигура/линия вновь появится | Тест  пройден |
| 3 | Отмена изменения размеров текстовой фигуры/линии | 1. Добавить текстовую фигуру.  2. Изменить размеры текстовой фигуры.  3. Отменить изменение. | Установятся прежние размеры | Тест  пройден |
| 4 | Отмена перемещения фигуры/линии | 1. Добавить фигуру.  2. Переместить фигуру.  3. Отменить изменения. | Фигура вернется в прежнее положение | Тест пройден |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **№** | **Тестируемая функциональ-ность** | **Последовательность действий** | **Ожидаемый результат** | **Получен-ный результат** |
| 5 | Отмена добавления точки линии | 1. Добавить несколько точек линии.  2.Закончить рисование линии.  3. Отменить изменения. | Последняя точка линии удалится | Тест пройден |
| 6 | Отмена изменений в пустом стеке | 1. Отменить изменения.  2. Добавить фигуру.  3. Отменить изменения. | Первая отмена не сработает, вторая сработает согласно п.1 | Тест пройден |

Продолжение таблицы 4.3

1. 4. Тестирование прочего функционала программного средства

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **№** | **Тестируемая функциональ-ность** | **Последовательность действий** | **Ожидаемый результат** | **Получен-ный результат** |
| 1 | Выравнивание фигуры по сравнению с другой | 1.Добавляем фигуру.  2.Добавляем вторую фигуру по вертикали немного отклонившись от первой. | Фигуры будут на одной линии | Тест пройден |
| 2 | Отмена изменений в пустом стеке | 1. Отменить изменения.  2. Добавить фигуру.  3. Отменить изменения. | Первая отмена не сработает, вторая сработает согласно п.1 | Тест пройден |
| 3 | Закрытие приложения | 1. Нарисовать что-то на полотне.  2. Закрыть приложение. | Покажется предложение сохранить изменения | Тест пройден |

Таблица 4.4 – тестирование прочего функционала программного средства

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 4 | Создание нового документа | 1. Нарисовать что-то на полотне.  2. Выбрать «Создать новый документ». | Покажется уведомление, что несохраненные изменения удалятся, если выбрано «ОК» - создастся новый документ | Тест пройден |
| 5 | Сохранение исходного файла и его открытие | 1. Нарисовать что-то на полотне.  2. Сохранить файл как cпец. заданный тип.  3. Закрыть программу.  4. Открыть файл. | Сохранение и открытие прошли успешно | Тест пройден |
| 6 | 1. Нарисовать что-то на полотне.  2. Сохранить файл как BMP.  3. Открыть файл BMP. | Схема алгоритма в формате BMP | Тест пройден |

Продолжение таблицы 4.4

1. 5. Вывод из прохождения тестирования

В ходе тестирования я проверил корректность работы программы и выяснил, что программа функционирует в соответствии с назначенными требованиями, значит она готова к использованию.

1. Руководство по установке и использованию

Для использования программы необходимо открыть приложение GOSTDiagram.exe. После открытия программы появится окно, показанное на рисунке 5.1



Рисунок 5.1 – Программное средство на устройстве под управлением Windows 10

По умолчанию при открытии выбран «Указатель». Чтобы выбрать фигуру или линию, необходимо кликнуть на соответствующую иконку в панели инструментов, рисунок 5.2.



Рисунок 5.2 – панель инструментов (перевернутый вид)

После выбора фигуры/линии можно приступать к рисованию. Если выбран один из режимов рисования текстовых фигур – каждое нажатие мыши добавляет новую фигуру. Если выбран режим «линия», то каждое нажатие ЛКМ по полотну добавляет новую точку фигуру, а ПКМ – прекращает рисование.

Для того, чтобы в фигуре оказался нужный текст, необходимо два раза кликнуть в области фигуры и ввести его. При выделении фигуры для добавления текста её углы подсвечиваются зеленым цветом. Рисунок 5.3.

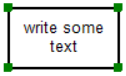


Рисунок 5.3 – текстовое поле для набора текста фигуры

Если выбран «указатель», то при наведении на фигуру меняется курсор, после чего зажатием и перемещением курсора можно переместить фигуру.

Для того, чтобы изменять размеры фигуры, необходимо сначала выбрать её, а затем навести курсор на одну из вершин или на край фигуры. При этом углы фигуры будут подсвечены синим цветом.

Также в программе присутствует меню и панель меню, изображенные на рисунке 5.4.



Рисунок 5.4 – меню и панель меню

Меню и панель меню содержит в себе:

* Создание нового файла.
* Открытие файла.
* Сохранение файла.
* Копирование фигур.
* Вставка фигур.
* Отмена изменений.

Также, чтобы сохранить синтаксическую диаграмму в формате BMP, необходимо выбрать тип файла «bmp» при сохранении. Рисунок 5.5

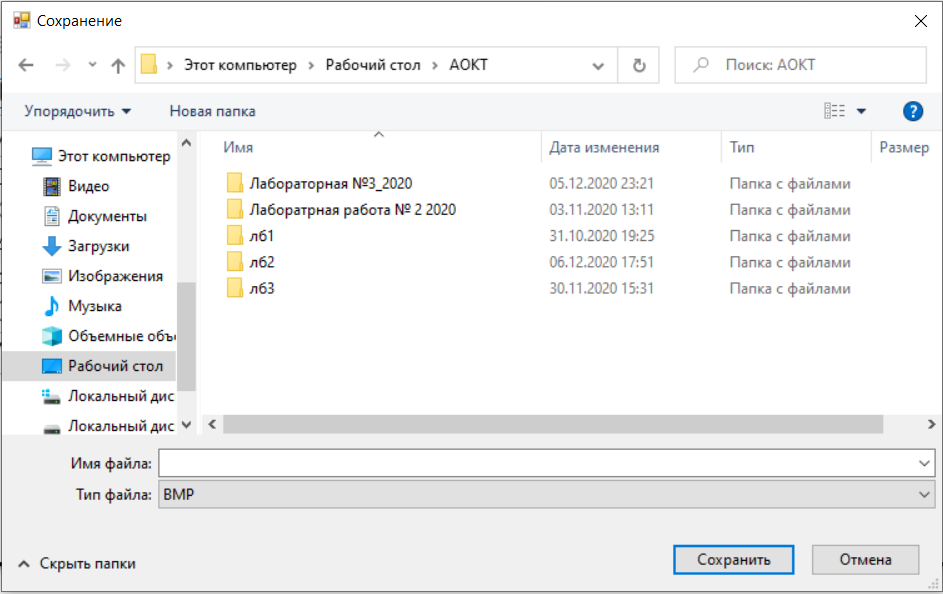


Рисунок 5.5 – сохранение в формате bmp

Заключение

В ходе данной работы было разработано приложение, имеющее достаточный функционал для построения схем алгоритмов, а также простой и удобный пользовательский интерфейс, который позволяет пользователю легко привыкнуть к приложению.

В процессе разработки, я изучил принципы работы с графикой в языке программирования С++ и принципы организации пользовательского интерфейса. Также мною получен опыт работы с парадигмой ООП и паттерном проектирования MVC.

Разработанное программное средство может быть применено программистами и архитекторами для того, чтобы нагляднее описать алгоритм, а также студентами в обучающих целях.

Список литературы

[1] Wikipedia [Электронный ресурс] – Блок-схема – Режим доступа: <https://ru.wikipedia.org/wiki/Блок-схема>

[2] Техэксперт [Электронный ресурс] – ГОСТ 19.701-90 – Режим доступа: <https://www.cgl.ucsf.edu/chimera/docs/UsersGuide/midas/vdwrad.html>

[3] Metanit [Электронный ресурс] – Введение в Windows Forms – Режим доступа: <https://metanit.com/sharp/windowsforms/1.1.php>

[4] Официальная документация C++ [Электронный ресурс] – Microsoft C++ documentation. – Режим доступа: <https://docs.microsoft.com/en-us/cpp/?view=msvc-160>

Приложение A

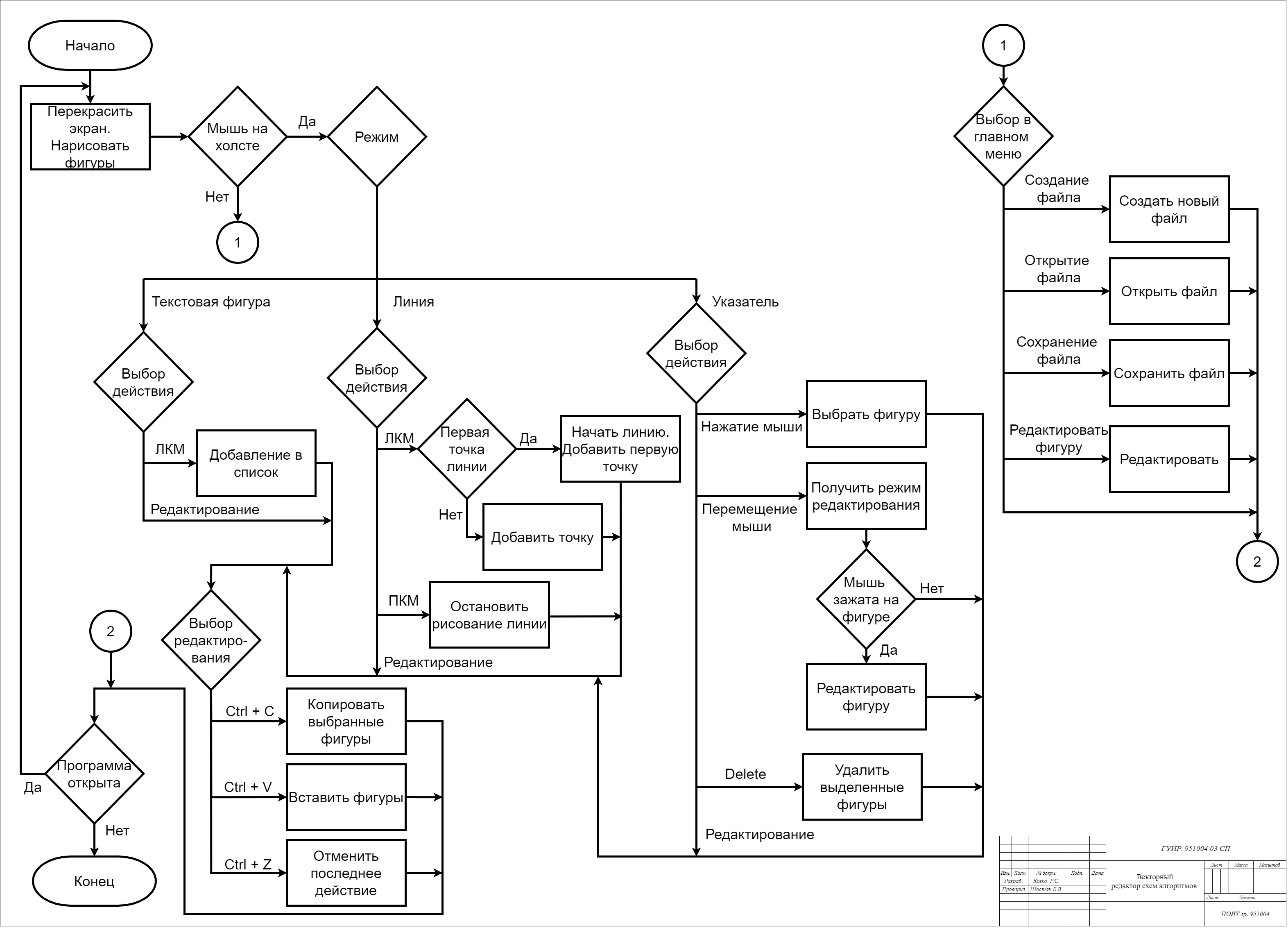


Рисунок А.1 – Схема алгоритма программы

Приложение В

**Controller.h**

#pragma once

#include "PictureView.h"

#include "DataTypes.h"

#include "FigureModel.h"

#include "RectFunctions.h"

#include "RectFunc.h"

#include "AbstractFigure.h"

#include "FigureModel.h"

#include "UndoStack.h"

FunctionType curType; // текущий тип фигуры

EditMode curEditMode; // режим редактирования

TDrawMode curDrawMode; // режим рисования

WriteType curWriteType; // режим написания

bool isChanged;

int curX, curY;

int startX, startY;

Rect selectedRect;

bool saveToBMP;

bool isMove;

namespace GOSTDiagram {

using namespace System;

using namespace System::ComponentModel;

using namespace System::Collections;

using namespace System::Windows::Forms;

using namespace System::Data;

using namespace System::Drawing;

/// <summary>

/// Сводка для Controller

/// </summary>

public ref class Controller : public System::Windows::Forms::Form

{

public:

System::String^ curPath;

List^ head;

List^ curElement;

List^ curSelectedFunc;

List^ selectedHead;

List^ copyHead;

UndoStack^ stackHead;

Controller(void)

{

InitializeComponent();

//

//TODO: добавьте код конструктора

//

this->curPath = System::String::Empty;

this->head = initList(head);

this->selectedHead = initList(selectedHead);

this->curElement = nullptr;

this->curSelectedFunc = nullptr;

this->copyHead = initList(copyHead);

this->stackHead = stackInit(stackHead);

}

protected:

/// <summary>

/// Освободить все используемые ресурсы.

/// </summary>

~Controller()

{

if (components)

{

delete components;

}

cleanList(head);

cleanList(selectedHead);

delete head;

delete copyHead;

delete selectedHead;

delete curPath;

delete stackHead;

}

private: System::Windows::Forms::MenuStrip^ menuStrip;

private: System::Windows::Forms::ToolStripMenuItem^ FileStripMenuItem;

private: System::Windows::Forms::ToolStripMenuItem^ FileNewMenuItem;

private: System::Windows::Forms::ToolStripMenuItem^ FileOpenMenuItem;

protected:

protected:

private: System::Windows::Forms::ToolStripMenuItem^ EditStripMenuItem;

private: System::Windows::Forms::ToolStripMenuItem^ FileSaveMenuItem;

private: System::Windows::Forms::ToolStripMenuItem^ FileSaveAsMenuItem;

private: System::Windows::Forms::ToolStripMenuItem^ FileExitMenuItem;

private: System::Windows::Forms::ImageList^ MenuImageList;

private: System::Windows::Forms::ToolStripMenuItem^ EditUndoMenuItem;

private: System::Windows::Forms::ToolStripMenuItem^ EditCopyMenuItem;

private: System::Windows::Forms::ToolStripMenuItem^ EditInsertMenuItem;

private: System::Windows::Forms::ToolStripSeparator^ toolStripMenuItem2;

private: System::Windows::Forms::ToolStripSeparator^ toolStripMenuItem3;

private: System::Windows::Forms::ToolStrip^ toolStripMenu;

private: System::Windows::Forms::ToolStripButton^ toolStripNewButton;

private: System::Windows::Forms::ToolStripButton^ toolStripOpenButton;

private: System::Windows::Forms::ToolStripButton^ toolStripSaveButton;

private: System::Windows::Forms::ToolStripButton^ toolStripSaveAsButton;

private: System::Windows::Forms::ToolStripSeparator^ toolStripSeparator1;

private: System::Windows::Forms::ToolStripButton^ toolStripUndoButton;

private: System::Windows::Forms::ToolStripButton^ toolStripCopyButton;

private: System::Windows::Forms::ToolStripButton^ toolStripPasteButton;

private: System::Windows::Forms::ToolStrip^ toolStripFunctions;

private: System::Windows::Forms::ToolStripButton^ toolStripPointerButton;

private: System::Windows::Forms::ToolStripButton^ toolStripLineButton;

private: System::Windows::Forms::ToolStripButton^ toolStripProcessButton;

private: System::Windows::Forms::ToolStripButton^ toolStripDecisionButton;

private: System::Windows::Forms::ToolStripButton^ toolStripDataButton;

private: System::Windows::Forms::ToolStripButton^ toolStripTerminatorButton;

private: System::Windows::Forms::ToolStripButton^ toolStripPageReferenceButton;

private: System::Windows::Forms::ToolStripButton^ toolStripTextButton;

private: System::Windows::Forms::ToolStripSeparator^ toolStripSeparator2;

private: System::Windows::Forms::ToolStripSeparator^ toolStripSeparator3;

private: System::Windows::Forms::ToolStripSeparator^ toolStripSeparator4;

private: System::Windows::Forms::ToolStripSeparator^ toolStripSeparator5;

private: System::Windows::Forms::Panel^ panel;

private: System::Windows::Forms::PictureBox^ pictureBox;

private: System::Windows::Forms::OpenFileDialog^ openFileDialog;

private: System::Windows::Forms::SaveFileDialog^ saveFileDialog;

private: System::ComponentModel::IContainer^ components;

private:

/// <summary>

/// Обязательная переменная конструктора.

/// </summary>

#pragma region Windows Form Designer generated code

/// <summary>

/// Требуемый метод для поддержки конструктора — не изменяйте

/// содержимое этого метода с помощью редактора кода.

/// </summary>

void InitializeComponent(void)

{

this->components = (gcnew System::ComponentModel::Container());

System::ComponentModel::ComponentResourceManager^ resources = (gcnew System::ComponentModel::ComponentResourceManager(Controller::typeid));

this->menuStrip = (gcnew System::Windows::Forms::MenuStrip());

this->FileStripMenuItem = (gcnew System::Windows::Forms::ToolStripMenuItem());

this->FileNewMenuItem = (gcnew System::Windows::Forms::ToolStripMenuItem());

this->FileOpenMenuItem = (gcnew System::Windows::Forms::ToolStripMenuItem());

this->toolStripMenuItem2 = (gcnew System::Windows::Forms::ToolStripSeparator());

this->FileSaveMenuItem = (gcnew System::Windows::Forms::ToolStripMenuItem());

this->FileSaveAsMenuItem = (gcnew System::Windows::Forms::ToolStripMenuItem());

this->toolStripMenuItem3 = (gcnew System::Windows::Forms::ToolStripSeparator());

this->FileExitMenuItem = (gcnew System::Windows::Forms::ToolStripMenuItem());

this->EditStripMenuItem = (gcnew System::Windows::Forms::ToolStripMenuItem());

this->EditUndoMenuItem = (gcnew System::Windows::Forms::ToolStripMenuItem());

this->EditCopyMenuItem = (gcnew System::Windows::Forms::ToolStripMenuItem());

this->EditInsertMenuItem = (gcnew System::Windows::Forms::ToolStripMenuItem());

this->MenuImageList = (gcnew System::Windows::Forms::ImageList(this->components));

this->toolStripMenu = (gcnew System::Windows::Forms::ToolStrip());

this->toolStripNewButton = (gcnew System::Windows::Forms::ToolStripButton());

this->toolStripOpenButton = (gcnew System::Windows::Forms::ToolStripButton());

this->toolStripSaveButton = (gcnew System::Windows::Forms::ToolStripButton());

this->toolStripSaveAsButton = (gcnew System::Windows::Forms::ToolStripButton());

this->toolStripSeparator1 = (gcnew System::Windows::Forms::ToolStripSeparator());

this->toolStripUndoButton = (gcnew System::Windows::Forms::ToolStripButton());

this->toolStripCopyButton = (gcnew System::Windows::Forms::ToolStripButton());

this->toolStripPasteButton = (gcnew System::Windows::Forms::ToolStripButton());

this->toolStripFunctions = (gcnew System::Windows::Forms::ToolStrip());

this->toolStripPointerButton = (gcnew System::Windows::Forms::ToolStripButton());

this->toolStripSeparator2 = (gcnew System::Windows::Forms::ToolStripSeparator());

this->toolStripLineButton = (gcnew System::Windows::Forms::ToolStripButton());

this->toolStripSeparator3 = (gcnew System::Windows::Forms::ToolStripSeparator());

this->toolStripProcessButton = (gcnew System::Windows::Forms::ToolStripButton());

this->toolStripDecisionButton = (gcnew System::Windows::Forms::ToolStripButton());

this->toolStripDataButton = (gcnew System::Windows::Forms::ToolStripButton());

this->toolStripTerminatorButton = (gcnew System::Windows::Forms::ToolStripButton());

this->toolStripPageReferenceButton = (gcnew System::Windows::Forms::ToolStripButton());

this->toolStripSeparator4 = (gcnew System::Windows::Forms::ToolStripSeparator());

this->toolStripTextButton = (gcnew System::Windows::Forms::ToolStripButton());

this->toolStripSeparator5 = (gcnew System::Windows::Forms::ToolStripSeparator());

this->panel = (gcnew System::Windows::Forms::Panel());

this->pictureBox = (gcnew System::Windows::Forms::PictureBox());

this->openFileDialog = (gcnew System::Windows::Forms::OpenFileDialog());

this->saveFileDialog = (gcnew System::Windows::Forms::SaveFileDialog());

this->menuStrip->SuspendLayout();

this->toolStripMenu->SuspendLayout();

this->toolStripFunctions->SuspendLayout();

this->panel->SuspendLayout();

(cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->pictureBox))->BeginInit();

this->SuspendLayout();

//

// menuStrip

//

this->menuStrip->ImageScalingSize = System::Drawing::Size(15, 15);

this->menuStrip->Items->AddRange(gcnew cli::array< System::Windows::Forms::ToolStripItem^ >(2) {

this->FileStripMenuItem,

this->EditStripMenuItem

});

this->menuStrip->Location = System::Drawing::Point(0, 0);

this->menuStrip->Name = L"menuStrip";

this->menuStrip->Size = System::Drawing::Size(1082, 30);

this->menuStrip->TabIndex = 0;

this->menuStrip->Tag = L"";

this->menuStrip->Text = L"menuStrip";

//

// FileStripMenuItem

//

this->FileStripMenuItem->DropDownItems->AddRange(gcnew cli::array< System::Windows::Forms::ToolStripItem^ >(7) {

this->FileNewMenuItem,

this->FileOpenMenuItem, this->toolStripMenuItem2, this->FileSaveMenuItem, this->FileSaveAsMenuItem, this->toolStripMenuItem3,

this->FileExitMenuItem

});

this->FileStripMenuItem->Name = L"FileStripMenuItem";

this->FileStripMenuItem->Size = System::Drawing::Size(59, 26);

this->FileStripMenuItem->Text = L"Файл";

//

// FileNewMenuItem

//

this->FileNewMenuItem->Image = (cli::safe\_cast<System::Drawing::Image^>(resources->GetObject(L"FileNewMenuItem.Image")));

this->FileNewMenuItem->Name = L"FileNewMenuItem";

this->FileNewMenuItem->ShortcutKeys = static\_cast<System::Windows::Forms::Keys>((System::Windows::Forms::Keys::Control | System::Windows::Forms::Keys::N));

this->FileNewMenuItem->Size = System::Drawing::Size(264, 26);

this->FileNewMenuItem->Text = L"Новый";

this->FileNewMenuItem->Click += gcnew System::EventHandler(this, &Controller::FileNewMenuItem\_Click);

//

// FileOpenMenuItem

//

this->FileOpenMenuItem->Image = (cli::safe\_cast<System::Drawing::Image^>(resources->GetObject(L"FileOpenMenuItem.Image")));

this->FileOpenMenuItem->Name = L"FileOpenMenuItem";

this->FileOpenMenuItem->ShortcutKeys = static\_cast<System::Windows::Forms::Keys>((System::Windows::Forms::Keys::Control | System::Windows::Forms::Keys::O));

this->FileOpenMenuItem->Size = System::Drawing::Size(264, 26);

this->FileOpenMenuItem->Text = L"Открыть";

this->FileOpenMenuItem->Click += gcnew System::EventHandler(this, &Controller::FileOpenMenuItem\_Click);

//

// toolStripMenuItem2

//

this->toolStripMenuItem2->Name = L"toolStripMenuItem2";

this->toolStripMenuItem2->Size = System::Drawing::Size(261, 6);

//

// FileSaveMenuItem

//

this->FileSaveMenuItem->Image = (cli::safe\_cast<System::Drawing::Image^>(resources->GetObject(L"FileSaveMenuItem.Image")));

this->FileSaveMenuItem->Name = L"FileSaveMenuItem";

this->FileSaveMenuItem->ShortcutKeys = static\_cast<System::Windows::Forms::Keys>((System::Windows::Forms::Keys::Control | System::Windows::Forms::Keys::S));

this->FileSaveMenuItem->Size = System::Drawing::Size(264, 26);

this->FileSaveMenuItem->Text = L"Сохранить";

this->FileSaveMenuItem->Click += gcnew System::EventHandler(this, &Controller::FileSaveMenuItem\_Click);

//

// FileSaveAsMenuItem

//

this->FileSaveAsMenuItem->Image = (cli::safe\_cast<System::Drawing::Image^>(resources->GetObject(L"FileSaveAsMenuItem.Image")));

this->FileSaveAsMenuItem->Name = L"FileSaveAsMenuItem";

this->FileSaveAsMenuItem->ShortcutKeys = static\_cast<System::Windows::Forms::Keys>((System::Windows::Forms::Keys::Shift | System::Windows::Forms::Keys::F12));

this->FileSaveAsMenuItem->Size = System::Drawing::Size(264, 26);

this->FileSaveAsMenuItem->Text = L"Сохранить как";

this->FileSaveAsMenuItem->Click += gcnew System::EventHandler(this, &Controller::FileSaveAsMenuItem\_Click);

//

// toolStripMenuItem3

//

this->toolStripMenuItem3->Name = L"toolStripMenuItem3";

this->toolStripMenuItem3->Size = System::Drawing::Size(261, 6);

//

// FileExitMenuItem

//

this->FileExitMenuItem->Name = L"FileExitMenuItem";

this->FileExitMenuItem->Size = System::Drawing::Size(264, 26);

this->FileExitMenuItem->Text = L"Выйти";

this->FileExitMenuItem->Click += gcnew System::EventHandler(this, &Controller::FileExitMenuItem\_Click);

//

// EditStripMenuItem

//

this->EditStripMenuItem->DropDownItems->AddRange(gcnew cli::array< System::Windows::Forms::ToolStripItem^ >(3) {

this->EditUndoMenuItem,

this->EditCopyMenuItem, this->EditInsertMenuItem

});

this->EditStripMenuItem->Name = L"EditStripMenuItem";

this->EditStripMenuItem->Size = System::Drawing::Size(137, 26);

this->EditStripMenuItem->Text = L"Редактирование";

//

// EditUndoMenuItem

//

this->EditUndoMenuItem->Image = (cli::safe\_cast<System::Drawing::Image^>(resources->GetObject(L"EditUndoMenuItem.Image")));

this->EditUndoMenuItem->Name = L"EditUndoMenuItem";

this->EditUndoMenuItem->ShortcutKeys = static\_cast<System::Windows::Forms::Keys>((System::Windows::Forms::Keys::Control | System::Windows::Forms::Keys::Z));

this->EditUndoMenuItem->Size = System::Drawing::Size(227, 26);

this->EditUndoMenuItem->Text = L"Отменить";

this->EditUndoMenuItem->Click += gcnew System::EventHandler(this, &Controller::EditUndoMenuItem\_Click);

//

// EditCopyMenuItem

//

this->EditCopyMenuItem->Image = (cli::safe\_cast<System::Drawing::Image^>(resources->GetObject(L"EditCopyMenuItem.Image")));

this->EditCopyMenuItem->Name = L"EditCopyMenuItem";

this->EditCopyMenuItem->ShortcutKeys = static\_cast<System::Windows::Forms::Keys>((System::Windows::Forms::Keys::Control | System::Windows::Forms::Keys::C));

this->EditCopyMenuItem->Size = System::Drawing::Size(227, 26);

this->EditCopyMenuItem->Text = L"Копировать";

this->EditCopyMenuItem->Click += gcnew System::EventHandler(this, &Controller::EditCopyMenuItem\_Click);

//

// EditInsertMenuItem

//

this->EditInsertMenuItem->Image = (cli::safe\_cast<System::Drawing::Image^>(resources->GetObject(L"EditInsertMenuItem.Image")));

this->EditInsertMenuItem->Name = L"EditInsertMenuItem";

this->EditInsertMenuItem->ShortcutKeys = static\_cast<System::Windows::Forms::Keys>((System::Windows::Forms::Keys::Control | System::Windows::Forms::Keys::V));

this->EditInsertMenuItem->Size = System::Drawing::Size(227, 26);

this->EditInsertMenuItem->Text = L"Вставить";

this->EditInsertMenuItem->Click += gcnew System::EventHandler(this, &Controller::EditInsertMenuItem\_Click);

//

// MenuImageList

//

this->MenuImageList->ImageStream = (cli::safe\_cast<System::Windows::Forms::ImageListStreamer^>(resources->GetObject(L"MenuImageList.ImageStream")));

this->MenuImageList->TransparentColor = System::Drawing::Color::Transparent;

this->MenuImageList->Images->SetKeyName(0, L"New.ico");

this->MenuImageList->Images->SetKeyName(1, L"Open.ico");

this->MenuImageList->Images->SetKeyName(2, L"Save.ico");

this->MenuImageList->Images->SetKeyName(3, L"SaveAs.ico");

this->MenuImageList->Images->SetKeyName(4, L"UndoBlue.ico");

this->MenuImageList->Images->SetKeyName(5, L"Copy.ico");

this->MenuImageList->Images->SetKeyName(6, L"Paste.ico");

this->MenuImageList->Images->SetKeyName(7, L"ZoomIn.ico");

this->MenuImageList->Images->SetKeyName(8, L"ZoomOut.ico");

this->MenuImageList->Images->SetKeyName(9, L"CutBlue.ico");

this->MenuImageList->Images->SetKeyName(10, L"CutRed.ico");

this->MenuImageList->Images->SetKeyName(11, L"RedoBlue.ico");

this->MenuImageList->Images->SetKeyName(12, L"RedoGreen.ico");

this->MenuImageList->Images->SetKeyName(13, L"RedoRed.ico");

this->MenuImageList->Images->SetKeyName(14, L"UndoGreen.ico");

this->MenuImageList->Images->SetKeyName(15, L"UndoRed.ico");

//

// toolStripMenu

//

this->toolStripMenu->BackColor = System::Drawing::SystemColors::Control;

this->toolStripMenu->Font = (gcnew System::Drawing::Font(L"Segoe UI", 10.2F, System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,

static\_cast<System::Byte>(0)));

this->toolStripMenu->ImageScalingSize = System::Drawing::Size(15, 15);

this->toolStripMenu->Items->AddRange(gcnew cli::array< System::Windows::Forms::ToolStripItem^ >(8) {

this->toolStripNewButton,

this->toolStripOpenButton, this->toolStripSaveButton, this->toolStripSaveAsButton, this->toolStripSeparator1, this->toolStripUndoButton,

this->toolStripCopyButton, this->toolStripPasteButton

});

this->toolStripMenu->Location = System::Drawing::Point(0, 30);

this->toolStripMenu->Name = L"toolStripMenu";

this->toolStripMenu->Padding = System::Windows::Forms::Padding(4, 0, 4, 0);

this->toolStripMenu->Size = System::Drawing::Size(1082, 31);

this->toolStripMenu->TabIndex = 1;

this->toolStripMenu->Text = L"toolStripMenu";

//

// toolStripNewButton

//

this->toolStripNewButton->DisplayStyle = System::Windows::Forms::ToolStripItemDisplayStyle::Image;

this->toolStripNewButton->Image = (cli::safe\_cast<System::Drawing::Image^>(resources->GetObject(L"toolStripNewButton.Image")));

this->toolStripNewButton->ImageTransparentColor = System::Drawing::Color::Magenta;

this->toolStripNewButton->Name = L"toolStripNewButton";

this->toolStripNewButton->Size = System::Drawing::Size(29, 28);

this->toolStripNewButton->Click += gcnew System::EventHandler(this, &Controller::toolStripNewButton\_Click);

//

// toolStripOpenButton

//

this->toolStripOpenButton->DisplayStyle = System::Windows::Forms::ToolStripItemDisplayStyle::Image;

this->toolStripOpenButton->Image = (cli::safe\_cast<System::Drawing::Image^>(resources->GetObject(L"toolStripOpenButton.Image")));

this->toolStripOpenButton->ImageTransparentColor = System::Drawing::Color::Magenta;

this->toolStripOpenButton->Name = L"toolStripOpenButton";

this->toolStripOpenButton->Size = System::Drawing::Size(29, 28);

this->toolStripOpenButton->Click += gcnew System::EventHandler(this, &Controller::toolStripOpenButton\_Click);

//

// toolStripSaveButton

//

this->toolStripSaveButton->DisplayStyle = System::Windows::Forms::ToolStripItemDisplayStyle::Image;

this->toolStripSaveButton->Image = (cli::safe\_cast<System::Drawing::Image^>(resources->GetObject(L"toolStripSaveButton.Image")));

this->toolStripSaveButton->ImageTransparentColor = System::Drawing::Color::Magenta;

this->toolStripSaveButton->Name = L"toolStripSaveButton";

this->toolStripSaveButton->Size = System::Drawing::Size(29, 28);

this->toolStripSaveButton->Click += gcnew System::EventHandler(this, &Controller::toolStripSaveButton\_Click);

//

// toolStripSaveAsButton

//

this->toolStripSaveAsButton->DisplayStyle = System::Windows::Forms::ToolStripItemDisplayStyle::Image;

this->toolStripSaveAsButton->Image = (cli::safe\_cast<System::Drawing::Image^>(resources->GetObject(L"toolStripSaveAsButton.Image")));

this->toolStripSaveAsButton->ImageTransparentColor = System::Drawing::Color::Magenta;

this->toolStripSaveAsButton->Name = L"toolStripSaveAsButton";

this->toolStripSaveAsButton->Size = System::Drawing::Size(29, 28);

this->toolStripSaveAsButton->Click += gcnew System::EventHandler(this, &Controller::toolStripSaveAsButton\_Click);

//

// toolStripSeparator1

//

this->toolStripSeparator1->Name = L"toolStripSeparator1";

this->toolStripSeparator1->Size = System::Drawing::Size(6, 31);

//

// toolStripUndoButton

//

this->toolStripUndoButton->DisplayStyle = System::Windows::Forms::ToolStripItemDisplayStyle::Image;

this->toolStripUndoButton->Image = (cli::safe\_cast<System::Drawing::Image^>(resources->GetObject(L"toolStripUndoButton.Image")));

this->toolStripUndoButton->ImageTransparentColor = System::Drawing::Color::Magenta;

this->toolStripUndoButton->Name = L"toolStripUndoButton";

this->toolStripUndoButton->Size = System::Drawing::Size(29, 28);

this->toolStripUndoButton->Click += gcnew System::EventHandler(this, &Controller::toolStripUndoButton\_Click);

//

// toolStripCopyButton

//

this->toolStripCopyButton->DisplayStyle = System::Windows::Forms::ToolStripItemDisplayStyle::Image;

this->toolStripCopyButton->Image = (cli::safe\_cast<System::Drawing::Image^>(resources->GetObject(L"toolStripCopyButton.Image")));

this->toolStripCopyButton->ImageTransparentColor = System::Drawing::Color::Magenta;

this->toolStripCopyButton->Name = L"toolStripCopyButton";

this->toolStripCopyButton->Size = System::Drawing::Size(29, 28);

this->toolStripCopyButton->Click += gcnew System::EventHandler(this, &Controller::toolStripCopyButton\_Click);

//

// toolStripPasteButton

//

this->toolStripPasteButton->DisplayStyle = System::Windows::Forms::ToolStripItemDisplayStyle::Image;

this->toolStripPasteButton->Image = (cli::safe\_cast<System::Drawing::Image^>(resources->GetObject(L"toolStripPasteButton.Image")));

this->toolStripPasteButton->ImageTransparentColor = System::Drawing::Color::Magenta;

this->toolStripPasteButton->Name = L"toolStripPasteButton";

this->toolStripPasteButton->Size = System::Drawing::Size(29, 28);

this->toolStripPasteButton->Click += gcnew System::EventHandler(this, &Controller::toolStripPasteButton\_Click);

//

// toolStripFunctions

//

this->toolStripFunctions->Dock = System::Windows::Forms::DockStyle::Left;

this->toolStripFunctions->GripMargin = System::Windows::Forms::Padding(2, 3, 2, 3);

this->toolStripFunctions->ImageScalingSize = System::Drawing::Size(20, 20);

this->toolStripFunctions->Items->AddRange(gcnew cli::array< System::Windows::Forms::ToolStripItem^ >(12) {

this->toolStripPointerButton,

this->toolStripSeparator2, this->toolStripLineButton, this->toolStripSeparator3, this->toolStripProcessButton, this->toolStripDecisionButton,

this->toolStripDataButton, this->toolStripTerminatorButton, this->toolStripPageReferenceButton, this->toolStripSeparator4, this->toolStripTextButton,

this->toolStripSeparator5

});

this->toolStripFunctions->Location = System::Drawing::Point(0, 61);

this->toolStripFunctions->Name = L"toolStripFunctions";

this->toolStripFunctions->Padding = System::Windows::Forms::Padding(0, 1, 0, 1);

this->toolStripFunctions->Size = System::Drawing::Size(29, 492);

this->toolStripFunctions->TabIndex = 2;

this->toolStripFunctions->Text = L"toolStrip";

this->toolStripFunctions->ItemClicked += gcnew System::Windows::Forms::ToolStripItemClickedEventHandler(this, &Controller::toolStripFunctions\_ItemClicked);

//

// toolStripPointerButton

//

this->toolStripPointerButton->CheckOnClick = true;

this->toolStripPointerButton->DisplayStyle = System::Windows::Forms::ToolStripItemDisplayStyle::Image;

this->toolStripPointerButton->Image = (cli::safe\_cast<System::Drawing::Image^>(resources->GetObject(L"toolStripPointerButton.Image")));

this->toolStripPointerButton->ImageTransparentColor = System::Drawing::Color::Magenta;

this->toolStripPointerButton->Name = L"toolStripPointerButton";

this->toolStripPointerButton->Size = System::Drawing::Size(28, 24);

this->toolStripPointerButton->Tag = L"0";

//

// toolStripSeparator2

//

this->toolStripSeparator2->Name = L"toolStripSeparator2";

this->toolStripSeparator2->Size = System::Drawing::Size(28, 6);

this->toolStripSeparator2->Tag = L"notFigure";

//

// toolStripLineButton

//

this->toolStripLineButton->CheckOnClick = true;

this->toolStripLineButton->DisplayStyle = System::Windows::Forms::ToolStripItemDisplayStyle::Image;

this->toolStripLineButton->Image = (cli::safe\_cast<System::Drawing::Image^>(resources->GetObject(L"toolStripLineButton.Image")));

this->toolStripLineButton->ImageTransparentColor = System::Drawing::Color::Magenta;

this->toolStripLineButton->Name = L"toolStripLineButton";

this->toolStripLineButton->Size = System::Drawing::Size(28, 24);

this->toolStripLineButton->Tag = L"1";

//

// toolStripSeparator3

//

this->toolStripSeparator3->Name = L"toolStripSeparator3";

this->toolStripSeparator3->Size = System::Drawing::Size(28, 6);

this->toolStripSeparator3->Tag = L"notFigure";

//

// toolStripProcessButton

//

this->toolStripProcessButton->CheckOnClick = true;

this->toolStripProcessButton->DisplayStyle = System::Windows::Forms::ToolStripItemDisplayStyle::Image;

this->toolStripProcessButton->Image = (cli::safe\_cast<System::Drawing::Image^>(resources->GetObject(L"toolStripProcessButton.Image")));

this->toolStripProcessButton->ImageTransparentColor = System::Drawing::Color::Magenta;

this->toolStripProcessButton->Name = L"toolStripProcessButton";

this->toolStripProcessButton->Size = System::Drawing::Size(28, 24);

this->toolStripProcessButton->Tag = L"2";

//

// toolStripDecisionButton

//

this->toolStripDecisionButton->CheckOnClick = true;

this->toolStripDecisionButton->DisplayStyle = System::Windows::Forms::ToolStripItemDisplayStyle::Image;

this->toolStripDecisionButton->Image = (cli::safe\_cast<System::Drawing::Image^>(resources->GetObject(L"toolStripDecisionButton.Image")));

this->toolStripDecisionButton->ImageTransparentColor = System::Drawing::Color::Magenta;

this->toolStripDecisionButton->Name = L"toolStripDecisionButton";

this->toolStripDecisionButton->Size = System::Drawing::Size(28, 24);

this->toolStripDecisionButton->Tag = L"3";

//

// toolStripDataButton

//

this->toolStripDataButton->CheckOnClick = true;

this->toolStripDataButton->DisplayStyle = System::Windows::Forms::ToolStripItemDisplayStyle::Image;

this->toolStripDataButton->Image = (cli::safe\_cast<System::Drawing::Image^>(resources->GetObject(L"toolStripDataButton.Image")));

this->toolStripDataButton->ImageTransparentColor = System::Drawing::Color::Magenta;

this->toolStripDataButton->Name = L"toolStripDataButton";

this->toolStripDataButton->Size = System::Drawing::Size(28, 24);

this->toolStripDataButton->Tag = L"4";

//

// toolStripTerminatorButton

//

this->toolStripTerminatorButton->CheckOnClick = true;

this->toolStripTerminatorButton->DisplayStyle = System::Windows::Forms::ToolStripItemDisplayStyle::Image;

this->toolStripTerminatorButton->Image = (cli::safe\_cast<System::Drawing::Image^>(resources->GetObject(L"toolStripTerminatorButton.Image")));

this->toolStripTerminatorButton->ImageTransparentColor = System::Drawing::Color::Magenta;

this->toolStripTerminatorButton->Name = L"toolStripTerminatorButton";

this->toolStripTerminatorButton->Size = System::Drawing::Size(28, 24);

this->toolStripTerminatorButton->Tag = L"5";

//

// toolStripPageReferenceButton

//

this->toolStripPageReferenceButton->CheckOnClick = true;

this->toolStripPageReferenceButton->DisplayStyle = System::Windows::Forms::ToolStripItemDisplayStyle::Image;

this->toolStripPageReferenceButton->Image = (cli::safe\_cast<System::Drawing::Image^>(resources->GetObject(L"toolStripPageReferenceButton.Image")));

this->toolStripPageReferenceButton->ImageTransparentColor = System::Drawing::Color::Magenta;

this->toolStripPageReferenceButton->Name = L"toolStripPageReferenceButton";

this->toolStripPageReferenceButton->Size = System::Drawing::Size(28, 24);

this->toolStripPageReferenceButton->Tag = L"6";

//

// toolStripSeparator4

//

this->toolStripSeparator4->Name = L"toolStripSeparator4";

this->toolStripSeparator4->Size = System::Drawing::Size(28, 6);

this->toolStripSeparator4->Tag = L"notFigure";

//

// toolStripTextButton

//

this->toolStripTextButton->CheckOnClick = true;

this->toolStripTextButton->DisplayStyle = System::Windows::Forms::ToolStripItemDisplayStyle::Image;

this->toolStripTextButton->Image = (cli::safe\_cast<System::Drawing::Image^>(resources->GetObject(L"toolStripTextButton.Image")));

this->toolStripTextButton->ImageTransparentColor = System::Drawing::Color::Magenta;

this->toolStripTextButton->Name = L"toolStripTextButton";

this->toolStripTextButton->Size = System::Drawing::Size(28, 24);

this->toolStripTextButton->Tag = L"7";

//

// toolStripSeparator5

//

this->toolStripSeparator5->Name = L"toolStripSeparator5";

this->toolStripSeparator5->Size = System::Drawing::Size(28, 6);

this->toolStripSeparator5->Tag = L"notFigure";

//

// panel

//

this->panel->AutoScroll = true;

this->panel->Controls->Add(this->pictureBox);

this->panel->Dock = System::Windows::Forms::DockStyle::Fill;

this->panel->Location = System::Drawing::Point(29, 61);

this->panel->Name = L"panel";

this->panel->Size = System::Drawing::Size(1053, 492);

this->panel->TabIndex = 3;

//

// pictureBox

//

this->pictureBox->Anchor = static\_cast<System::Windows::Forms::AnchorStyles>((((System::Windows::Forms::AnchorStyles::Top | System::Windows::Forms::AnchorStyles::Bottom)

| System::Windows::Forms::AnchorStyles::Left)

| System::Windows::Forms::AnchorStyles::Right));

this->pictureBox->Location = System::Drawing::Point(0, 0);

this->pictureBox->Name = L"pictureBox";

this->pictureBox->Size = System::Drawing::Size(1053, 500);

this->pictureBox->SizeMode = System::Windows::Forms::PictureBoxSizeMode::AutoSize;

this->pictureBox->TabIndex = 0;

this->pictureBox->TabStop = false;

this->pictureBox->Paint += gcnew System::Windows::Forms::PaintEventHandler(this, &Controller::pictureBox\_Paint);

this->pictureBox->DoubleClick += gcnew System::EventHandler(this, &Controller::pictureBox\_DoubleClick);

this->pictureBox->MouseDown += gcnew System::Windows::Forms::MouseEventHandler(this, &Controller::pictureBox\_MouseDown);

this->pictureBox->MouseMove += gcnew System::Windows::Forms::MouseEventHandler(this, &Controller::pictureBox\_MouseMove);

this->pictureBox->MouseUp += gcnew System::Windows::Forms::MouseEventHandler(this, &Controller::pictureBox\_MouseUp);

//

// openFileDialog

//

this->openFileDialog->FileName = L"openFileDialog";

this->openFileDialog->Filter = L"Все файлы|\*.\*|BMP|\*.bmp|Специальный формат|\*.rks";

//

// saveFileDialog

//

this->saveFileDialog->Filter = L"Все файлы|\*.\*|BMP|\*.bmp|Специальный формат|\*.rks";

//

// Controller

//

this->AutoScaleDimensions = System::Drawing::SizeF(8, 16);

this->AutoScaleMode = System::Windows::Forms::AutoScaleMode::Font;

this->ClientSize = System::Drawing::Size(1082, 553);

this->Controls->Add(this->panel);

this->Controls->Add(this->toolStripFunctions);

this->Controls->Add(this->toolStripMenu);

this->Controls->Add(this->menuStrip);

this->MainMenuStrip = this->menuStrip;

this->Name = L"Controller";

this->Text = L"Algorithm diagram editor";

this->Load += gcnew System::EventHandler(this, &Controller::Controller\_Load);

this->KeyDown += gcnew System::Windows::Forms::KeyEventHandler(this, &Controller::Controller\_KeyDown);

this->menuStrip->ResumeLayout(false);

this->menuStrip->PerformLayout();

this->toolStripMenu->ResumeLayout(false);

this->toolStripMenu->PerformLayout();

this->toolStripFunctions->ResumeLayout(false);

this->toolStripFunctions->PerformLayout();

this->panel->ResumeLayout(false);

this->panel->PerformLayout();

(cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->pictureBox))->EndInit();

this->ResumeLayout(false);

this->PerformLayout();

}

#pragma endregion

void saveBMP(System::String^ path) { // сохранение в формает bmp

saveToBMP = true;

curSelectedFunc = nullptr;

pictureBox->Invalidate();

Bitmap^ bmp = gcnew Bitmap(pictureBox->Width, pictureBox->Height); // создание bmp-а

pictureBox->DrawToBitmap(bmp, pictureBox->ClientRectangle);

bmp->Save(path, System::Drawing::Imaging::ImageFormat::Bmp); // сохранение bmp-а

delete bmp;

saveToBMP = false;

}

System::String^ openNFile() { // получение пути для открытия файла

openFileDialog->ShowDialog();

return openFileDialog->FileName;

}

System::String^ saveNFile() { // получение пути для сохранения нового файла

saveFileDialog->ShowDialog();

return saveFileDialog->FileName;

}

bool saveNewFile() { // сохранение нового файла

System::String^ path = saveNFile();

int pos = path->IndexOf(".");

System::String^ ext = path->Substring(pos + 1);

if ((path->Length != 0) && (ext!="bmp")){

curPath = path;

saveToFile(head, curPath); // сохранение файла в спец. формате

return true;

}

else if (path->Length != 0) {

saveBMP(path); // сохранение файла в bmp

curPath = path;

return true;

}

return false;

}

void actNew() { // создание нового файла

System::Windows::Forms::DialogResult res = MessageBox::Show("Все несохраненные данные будут удалены.Продолжить?", "Создание нового файла", MessageBoxButtons::YesNo, MessageBoxIcon::Warning);

if (res == System::Windows::Forms::DialogResult::Yes) {

isChanged = false;

curSelectedFunc = nullptr;

cleanList(head);

cleanStack(stackHead);

cleanList(copyHead);

pictureBox->Invalidate();

}

}

void actOpen() { // открытие файла

if (isChanged) { // если файла изменен - предупреждение

System::Windows::Forms::DialogResult res = MessageBox::Show("Желаете сохранить файл?В противном случае все данные будут утеряны?", "Сохранение", MessageBoxButtons::YesNoCancel, MessageBoxIcon::Warning);

if (res == System::Windows::Forms::DialogResult::Yes) {

if (!saveNewFile())

return;

}

else if (res == System::Windows::Forms::DialogResult::Cancel) {

return;

}

}

System::String^ str = openNFile();

if ((str->Length != 0) && (str->ToString() != "openFileDialog")) {

//open file

cleanList(head);

cleanStack(stackHead);

curPath = str;

if (readFromFile(head, curPath)) { // открытие файла из спец формата

isChanged = false;

}

pictureBox->Invalidate();

}

}

void actSaveAs() { // сохранить как

saveNewFile();

}

void actSave() { // сохранить сохранить

int pos = curPath->IndexOf(".");

System::String^ ext = curPath->Substring(pos + 1);

if ((curPath->Length != 0) && (ext!="bmp")) {

saveToFile(head, curPath);

}

else if (curPath->Length != 0) {

//save bmp

saveBMP(curPath);

}

else

{

saveNewFile();

}

}

// additional methods

void changeButtonState(FunctionType type) { //состояние кнопки от нажатия

switch (type)

{

case POINTER: {

toolStripPointerButton->Checked = false;

break;

}

case LINE: {

toolStripLineButton->Checked = false;

break;

}

case PROCESS: {

toolStripProcessButton->Checked = false;

break;

}

case DECISSION: {

toolStripDecisionButton->Checked = false;

break;

}

case DATA: {

toolStripDataButton->Checked = false;

break;

}

case TERMINATOR: {

toolStripTerminatorButton->Checked = false;

break;

}

case PAGEREFERENCE: {

toolStripPageReferenceButton->Checked = false;

break;

}

case TEXT: {

toolStripTextButton->Checked = false;

break;

}

}

}

void stopDrawLine() { //закончить рисование

curDrawMode = NOT\_DRAW;

pictureBox->Invalidate();

}

void changeCursor(EditMode mode) { //смена курсора в зависимости от режима редактирования

switch (mode)

{

case NOT\_EDIT:

{

pictureBox->Cursor = System::Windows::Forms::Cursors::Arrow;

break;

}

case MOVE:

{

pictureBox->Cursor = System::Windows::Forms::Cursors::SizeAll;

break;

}

case TOP:

{

pictureBox->Cursor = System::Windows::Forms::Cursors::SizeNS;

break;

}

case BOTTOM:

{

pictureBox->Cursor = System::Windows::Forms::Cursors::SizeNS;

break;

}

case RIGHT:

{

pictureBox->Cursor = System::Windows::Forms::Cursors::SizeWE;

break;

}

case LEFT:

{

pictureBox->Cursor = System::Windows::Forms::Cursors::SizeWE;

break;

}

case LEFT\_TOP:

{

pictureBox->Cursor = System::Windows::Forms::Cursors::SizeNWSE;

break;

}

case RIGHT\_TOP:

{

pictureBox->Cursor = System::Windows::Forms::Cursors::SizeNESW;

break;

}

case LEFT\_BOTTOM:

{

pictureBox->Cursor = System::Windows::Forms::Cursors::SizeNESW;

break;

}

case RIGHT\_BOTTOM:

{

pictureBox->Cursor = System::Windows::Forms::Cursors::SizeNWSE;

break;

}

}

}

private: System::Void Controller\_Load(System::Object^ sender, System::EventArgs^ e) { //начальная загрузка и конфигурация

curType = POINTER;

toolStripPointerButton->Checked = true;

isChanged = false;

curEditMode = NOT\_EDIT;

curDrawMode = NOT\_DRAW;

selectedRect.top = -1;

saveToBMP = false;

curSelectedFunc = nullptr;

isMove = false;

curWriteType = NOT\_WRITE;

//readFromFile(head,)

}

private: System::Void pictureBox\_Paint(System::Object^ sender, System::Windows::Forms::PaintEventArgs^ e) {

cleanScreen(e, pictureBox->Width, pictureBox->Height); // очищаем экран

if (!saveToBMP) // если не сохраненив в bmp ,то выводим с вершинами

drawFunctions(e, head, true);

else

drawFunctions(e, head, false);

if ((curDrawMode == DRAW\_LINE) && (curType == LINE)) { //рисуем путь для линии

drawPath(e,(LineFunc^)curElement->fig, Point(curX, curY));

}

if ((curType == POINTER) && (curEditMode == NOT\_EDIT) && (selectedRect.top != -1)) { // рисуем область выделения

drawSelectedRect(e, selectedRect);

}

if ((curSelectedFunc != nullptr) || (selectedHead->next != nullptr)) {

if (selectedHead->next != nullptr) {

List^ cur = selectedHead->next;

while (cur!=nullptr)

{

repaintSelectedFunc(e, cur,false); //выделение вершин элемента

cur = cur->next;

}

}

else if (curSelectedFunc != nullptr)

repaintSelectedFunc(e, curSelectedFunc,false);

}

if ((curSelectedFunc != nullptr) && (curWriteType == WRITE)) {

repaintSelectedFunc(e, curSelectedFunc,true); //выделение для режима записи в фигуру

}

}

private: System::Void toolStripFunctions\_ItemClicked(System::Object^ sender, System::Windows::Forms::ToolStripItemClickedEventArgs^ e) {

if (curType == LINE) { // опрежеление текущего режима фигуры и отображения выделением кнопки

stopDrawLine();

}

if (e->ClickedItem->Tag->ToString()!="notFigure")

changeButtonState(curType);

if ((e->ClickedItem->Tag->ToString()) == functionTypeToString(POINTER)){

curType = POINTER;

}

else if ((e->ClickedItem->Tag->ToString()) == functionTypeToString(LINE)) {

curType = LINE;

}

else if ((e->ClickedItem->Tag->ToString()) == functionTypeToString(PROCESS)) {

curType = PROCESS;

}

else if ((e->ClickedItem->Tag->ToString()) == functionTypeToString(DECISSION)) {

curType = DECISSION;

}

else if ((e->ClickedItem->Tag->ToString()) == functionTypeToString(DATA)) {

curType = DATA;

}

else if ((e->ClickedItem->Tag->ToString()) == functionTypeToString(TERMINATOR)) {

curType = TERMINATOR;

}

else if ((e->ClickedItem->Tag->ToString()) == functionTypeToString(PAGEREFERENCE)) {

curType = PAGEREFERENCE;

}

else if ((e->ClickedItem->Tag->ToString()) == functionTypeToString(TEXT)) {

curType = TEXT;

}

else

return;

}

//действия с главным меню

private: System::Void FileNewMenuItem\_Click(System::Object^ sender, System::EventArgs^ e) {

actNew();

}

private: System::Void toolStripNewButton\_Click(System::Object^ sender, System::EventArgs^ e) {

actNew();

}

private: System::Void toolStripOpenButton\_Click(System::Object^ sender, System::EventArgs^ e) {

actOpen();

}

private: System::Void FileOpenMenuItem\_Click(System::Object^ sender, System::EventArgs^ e) {

actOpen();

}

private: System::Void toolStripSaveButton\_Click(System::Object^ sender, System::EventArgs^ e) {

actSave();

}

private: System::Void FileSaveMenuItem\_Click(System::Object^ sender, System::EventArgs^ e) {

actSave();

}

private: System::Void FileSaveAsMenuItem\_Click(System::Object^ sender, System::EventArgs^ e) {

actSaveAs();

}

private: System::Void toolStripSaveAsButton\_Click(System::Object^ sender, System::EventArgs^ e) {

actSaveAs();

}

private: System::Void FileExitMenuItem\_Click(System::Object^ sender, System::EventArgs^ e) { // выход из программы

if (isChanged) {

System::Windows::Forms::DialogResult res = MessageBox::Show("Желаете сохранить файл?В противном случае все данные будут утеряны?", "Сохранение", MessageBoxButtons::YesNoCancel, MessageBoxIcon::Warning);

if (res == System::Windows::Forms::DialogResult::Yes) {

if (!saveNewFile())

return;

}

else if (res == System::Windows::Forms::DialogResult::Cancel) {

return;

}

}

this->Close();

}

//PaintBox click events

private: System::Void pictureBox\_MouseDown(System::Object^ sender, System::Windows::Forms::MouseEventArgs^ e) {

if (curType != POINTER) {

cleanList(selectedHead); //очистка листа выделенных фигур

curSelectedFunc = nullptr;

}

if ((curType == POINTER) && (curEditMode == NOT\_EDIT)) {

cleanList(selectedHead);

selectedRect.left = e->X; //определения начальных координат выделения

selectedRect.right = e->X;

selectedRect.top = e->Y;

selectedRect.bottom = e->Y;

}

if (curDrawMode == DRAW\_LINE) {

isChanged = true;

if (e->Button == System::Windows::Forms::MouseButtons::Left) //добавление точки к линии

{

addPoints((LineFunc^)curElement->fig, Point(e->X, e->Y));

pushStack(stackHead, curElement, POINT\_ADD\_UNDO,nullptr);

return;

}

else if (e->Button == System::Windows::Forms::MouseButtons::Right) { //останока рисования линии

stopDrawLine();

return;

}

}

else

curDrawMode = DRAW;

if ((curType != POINTER) && (curEditMode == NOT\_EDIT)) {

isChanged = true;

if (curType != LINE) {

curElement = push(head, curType,Point(e->X,e->Y)); // добавление фигуры

pushStack(stackHead, curElement, FIGURE\_INSERT\_UNDO, nullptr);

}

else if (curDrawMode != DRAW\_LINE)

{

curElement = addLine(head, Point(e->X, e->Y)); // добавление линии

pushStack(stackHead, curElement, FIGURE\_INSERT\_UNDO, nullptr);

curDrawMode = DRAW\_LINE;

}

}

else

{

startX = e->X;

startY = e->Y;

}

if (curType == POINTER) {

curSelectedFunc = hitTest(head, Point(e->X, e->Y)); // определение выделенмя фигур

if (!(isElementInList(selectedHead,curSelectedFunc))) {

cleanList(selectedHead);

}

}

curWriteType = NOT\_WRITE;

}

private: System::Void pictureBox\_MouseMove(System::Object^ sender, System::Windows::Forms::MouseEventArgs^ e) {

if ((curType == POINTER) && (curEditMode == NOT\_EDIT) && (selectedRect.top!=-1)) {

selectedRect.right = e->X;

selectedRect.bottom = e->Y; // перемещение области выделения

pictureBox->Invalidate();

return;

}

if ((curType == POINTER) && (curDrawMode == NOT\_DRAW)) {

curEditMode = getCurEditMode(head, curSelectedFunc, Point(e->X, e->Y));

changeCursor(curEditMode); // определение режима редактирования и изменение курсора

}

if ((curDrawMode == DRAW) && (curEditMode != NOT\_EDIT)) {

List^ cur = selectedHead->next;

if ((curType == POINTER) && (!isMove)) {

isMove = true;

if ((cur == nullptr) && (curSelectedFunc != nullptr)) {

// подготовка данных для стека отмены

if (curSelectedFunc->fig->type != LINE) {

RectFunc^ curF = (RectFunc^)curSelectedFunc->fig;

String^ str = pointsToStr(curF->leftCoords);

str += pointsToStr(curF->rightCoords);

// внесение в стек фигуры

pushStack(stackHead, curSelectedFunc, FUNC\_MOVE\_UNDO, str);

}

else

{

LineFunc^ curL = (LineFunc^)curSelectedFunc->fig;

Points\* points = curL->headLine->next;

System::String^ res = "";

while (points != nullptr)

{

res += pointsToStr(System::Drawing::Point(points->x, points->y));

points = points->next;

}

// внесение в стек линии

pushStack(stackHead, curSelectedFunc, LINE\_MOVE\_UNDO, res);

}

}

while (cur != nullptr)

{

// подготовка данных для стека отмены

if (cur->fig->type != LINE)

{

RectFunc^ curF = (RectFunc^)cur->fig;

String^ str = pointsToStr(curF->leftCoords);

str += pointsToStr(curF->rightCoords);

// внесение в стек фигуры

pushStack(stackHead, cur, FUNC\_MOVE\_UNDO, str);

}

else

{

LineFunc^ curL = (LineFunc^)cur->fig;

Points\* points = curL->headLine->next;

System::String^ res = "";

while (points != nullptr)

{

res += pointsToStr(System::Drawing::Point(points->x, points->y));

points = points->next;

}

// внесение в стек линии

pushStack(stackHead, cur, LINE\_MOVE\_UNDO, res);

}

cur = cur->next;

}

}

// трансформирование элементов

cur = selectedHead->next;

if ((curSelectedFunc != nullptr) && (cur==nullptr))

transformFunc(curSelectedFunc, Point(e->X, e->Y), Point(startX, startY), curEditMode);

else

while (cur != nullptr) {

transformFunc(cur, Point(e->X, e->Y), Point(startX, startY), MOVE);

cur = cur->next;

}

startX = e->X;

startY = e->Y;

if (curEditMode != NOT\_EDIT)

pictureBox->Invalidate();

}

if ((curDrawMode == DRAW\_LINE) && (curType == LINE)) {

pictureBox->Invalidate();

curX = e->X;

curY = e->Y; // координаты для прокладывания пути линии

}

}

private: System::Void pictureBox\_MouseUp(System::Object^ sender, System::Windows::Forms::MouseEventArgs^ e) {

if ((curType == POINTER) && (curEditMode == NOT\_EDIT)) {

cleanList(selectedHead);

getSelectedFuncFromRect(head, selectedHead, selectedRect); //поиск элементов для выделения в области

selectedRect.top = -1;

}

if (curDrawMode != DRAW\_LINE) {

curDrawMode = NOT\_DRAW;

}

if (isMove) {

isMove = false;

}

// постановка фигур на одну вертикальную линию

if ((curSelectedFunc != nullptr) && (curSelectedFunc->fig->type != LINE))

makeFuncOnOneLine(head, curSelectedFunc);

else if ((curElement != nullptr) && (curElement->fig->type != LINE))

makeFuncOnOneLine(head, curElement);

// примагничивание в фигуре линии

if ((curSelectedFunc != nullptr) && (curSelectedFunc->fig->type == LINE))

makePointOnCenter(head, curSelectedFunc);

else if ((curElement != nullptr) && (curElement->fig->type == LINE))

makePointOnCenter(head, curElement);

pictureBox->Invalidate();

}

private: System::Void Controller\_KeyDown(System::Object^ sender, System::Windows::Forms::KeyEventArgs^ e) {

// обработка нажатия

if (curWriteType != WRITE) {

if (curSelectedFunc != nullptr) {

pushExistingFunc(selectedHead, curSelectedFunc->fig);

}

if (e->KeyCode == System::Windows::Forms::Keys::Delete) { // определение кнопки

List^ cur = selectedHead->next;

while (cur != nullptr)

{

pushStack(stackHead, cur, DELETE\_UNDO, nullptr);

deleteElement(head, cur); // удаление фигуры

cur = cur->next;

}

cleanList(selectedHead);

curSelectedFunc = nullptr;

pictureBox->Invalidate();

}

}

else

{

// редактирование текста в фигуре

// в зависимости от кода клавиши выбирается действие

if ((curSelectedFunc != nullptr) && (curSelectedFunc->fig->type != LINE)) {

RectFunc^ curF = (RectFunc^)curSelectedFunc->fig;

//8 delete | 13 enter | 32 space | 37 left | 39 right | 40 down | 38 up | 107 + | 109 - |106 \* |111 /

if (e->KeyValue == 13) {

curF->text += "\n";

} else if (e->KeyValue == 8){

if (curF->text->Length !=0)

curF->text = curF->text->Remove(curF->text->Length-1);

}

else if(e->KeyValue == 32) {

curF->text += " ";

}

else if (e->KeyValue == 107) {

curF->text += "+";

}

else if(e->KeyValue == 109) {

curF->text += "-";

}

else if (e->KeyValue == 106) {

curF->text += "\*";

}

else if (e->KeyValue == 111) {

curF->text += "/";

}

else

if (e->KeyValue>=65 && e->KeyValue <= 90)

curF->text += ((Char)(e->KeyValue + ('a'-'A'))).ToString();

else

curF->text += (Char(e->KeyValue)).ToString();

pictureBox->Invalidate();

}

}

}

private: System::Void pictureBox\_DoubleClick(System::Object^ sender, System::EventArgs^ e) {

curWriteType = WRITE; // переход в режим редактирования текста

isMove = false;

cleanList(selectedHead);

}

void actCopy() {

// копирования фигуры в список

cleanList(copyHead);

List^ cur = selectedHead->next;

while (cur != nullptr) {

pushExistingFunc(copyHead, cur->fig);

cur = cur->next;

}

}

int copyShift = 25;

void actPaste() {

// вставка фигуры

if (copyHead->next == nullptr) {

return;

}

List^ curCopy = copyHead->next;

cleanList(selectedHead);

while (curCopy != nullptr) {

// создание копии линии и смены её координат для удобства нахождения

if (curCopy->fig->type == LINE) {

LineFunc^ absL = gcnew LineFunc();

LineFunc^ copyL = (LineFunc^)curCopy->fig;

absL->headLine = new Points;

absL->headLine->next = nullptr;

absL->width = copyL->width;

Points\* points = copyL->headLine->next;

while (points != nullptr) {

addPoints(absL, System::Drawing::Point(points->x + copyShift, points->y + copyShift));

points = points->next;

}

pushExistingFunc(head, (AbstractFigure^)absL);

List^ curL = pushExistingFunc(selectedHead, (AbstractFigure^)absL);

pushStack(stackHead, curL, FIGURE\_INSERT\_UNDO, nullptr);

}

else {

// создание копии фигуры и смены её координат для удобства нахождения

AbstractFigure^ absF = figureFromType(curCopy->fig->type, System::Drawing::Point(0, 0));

RectFunc^ rectF = (RectFunc^)absF;

RectFunc^ copyF = (RectFunc^)curCopy->fig;

rectF->leftCoords.X = copyF->leftCoords.X + copyShift;

rectF->leftCoords.Y = copyF->leftCoords.Y + copyShift;

rectF->rightCoords.X = copyF->rightCoords.X + copyShift;

rectF->rightCoords.Y = copyF->rightCoords.Y + copyShift;

rectF->text = copyF->text;

pushExistingFunc(head, (AbstractFigure^)rectF);

List^ curL = pushExistingFunc(selectedHead, (AbstractFigure^)rectF);

pushStack(stackHead, curL, FIGURE\_INSERT\_UNDO, nullptr);

}

curCopy = curCopy->next;

}

pictureBox->Invalidate();

}

// функции редактирования

private: System::Void toolStripCopyButton\_Click(System::Object^ sender, System::EventArgs^ e) {

EditCopyMenuItem\_Click(sender,e);

}

private: System::Void toolStripPasteButton\_Click(System::Object^ sender, System::EventArgs^ e) {

EditInsertMenuItem\_Click(sender, e);

}

private: System::Void EditCopyMenuItem\_Click(System::Object^ sender, System::EventArgs^ e) {

actCopy();

}

private: System::Void EditInsertMenuItem\_Click(System::Object^ sender, System::EventArgs^ e) {

actPaste();

}

void actUndo(){

// отмена действия

UndoStack^ cur = popStack(stackHead);

undoStackAction(cur, head); // после доставания из стека выполняем отмену

if (cur != nullptr && cur->figure->fig->type == LINE) {

LineFunc^ line = (LineFunc^)cur->figure->fig;

if (line->headLine->next == nullptr) {

curDrawMode = NOT\_DRAW;

deleteElement(head, cur->figure);

}

}

cleanList(selectedHead);

curSelectedFunc = nullptr;

pictureBox->Invalidate();

}

private: System::Void EditUndoMenuItem\_Click(System::Object^ sender, System::EventArgs^ e) {

actUndo();

}

private: System::Void toolStripUndoButton\_Click(System::Object^ sender, System::EventArgs^ e) {

EditUndoMenuItem\_Click(sender, e);

}

};

}

**AbstractFigure.h**

#pragma once

#include "DataTypes.h"

// Абстрактный базовый класс

ref class AbstractFigure {

public:

virtual void drawFigure(System::Windows::Forms::PaintEventArgs^ e, bool isVertex) = 0;

FunctionType type;

AbstractFigure();

virtual ~AbstractFigure();

};

**AbstractFigure.cpp**

#include "AbstractFigure.h"

AbstractFigure::AbstractFigure() {

}

AbstractFigure::~AbstractFigure() {

}

**DataTypes.h**

#pragma once

enum WriteType { //типы для записи в фигуру

WRITE,

NOT\_WRITE

};

enum FunctionType{ //типы фигур

POINTER,

LINE,

PROCESS,

DECISSION,

DATA,

TERMINATOR,

PAGEREFERENCE,

TEXT

};

enum TDrawMode // типы режимов рисования

{

DRAW,

NOT\_DRAW,

DRAW\_LINE

};

enum EditMode // типы режимов редактировния

{

NOT\_EDIT,

MOVE,

LEFT,

RIGHT,

TOP,

BOTTOM,

LEFT\_TOP,

RIGHT\_TOP,

LEFT\_BOTTOM,

RIGHT\_BOTTOM,

};

struct Rect // структура для прямоугольой области

{

int top,bottom,right,left;

};

System::String^ functionTypeToString(FunctionType v); // перевод типа фигуры в строку

**DataTypes.cpp**

#include "DataTypes.h"

System::String^ functionTypeToString(FunctionType v)

{

switch (v)

{

case POINTER: return "0"; break;

case LINE: return "1"; break;

case PROCESS: return "2"; break;

case DECISSION: return "3"; break;

case DATA: return "4"; break;

case TERMINATOR: return "5"; break;

case PAGEREFERENCE: return "6"; break;

case TEXT: return "7"; break;

}

}

**FigureModel.h**

#pragma once

#include "AbstractFigure.h"

#include "DataTypes.h"

#include "LineFunc.h"

ref struct List // структура списка

{

AbstractFigure^ fig;

List^ next;

};

// основные функции для работы со списком и бизнесс логикой

List^ initList(List^);

void cleanList(List^);

List^ push(List^, FunctionType, System::Drawing::Point);

List^ pushExistingFunc(List^, AbstractFigure^);

bool isElementInList(List^, List^);

void deleteElement(List^, List^);

List^ addLine(List^, System::Drawing::Point);

Points\* addPoints(LineFunc^, System::Drawing::Point);

AbstractFigure^ figureFromType(FunctionType fType, System::Drawing::Point points);

//define selected Figure

List^ hitTest(List^ head, System::Drawing::Point points);

void getSelectedFuncFromRect(List^, List^, Rect);

int max(int, int);

int min(int, int);

void swap(int& first, int& second);

void transformFunc(List^, System::Drawing::Point, System::Drawing::Point, EditMode);

EditMode getCurEditMode(List^, List^, System::Drawing::Point);

void saveToFile(List^, System::String^);

bool readFromFile(List^, System::String^);

System::String^ pointsToStr(System::Drawing::Point points);

System::Drawing::Point parseStringToPoints(System::String^& str);

void makeFuncOnOneLine(List^ head, List^ cur);

void makeSmoothLine(List^ curLine, System::Drawing::Point newPoints, System::Drawing::Point prevPoints);

void makePointOnCenter(List^ head, List^ curL);

**FigureModel.cpp**

#include "FigureModel.h"

#include "RectFunctions.h"

#define \_USE\_MATH\_DEFINES

#include <math.h>

using namespace std;

int closeObject = 5;

List^ initList(List^ head) { //Создание списка

head = gcnew List;

head->next = nullptr;

return head;

}

void cleanList(List^ head) { //очистка списка

List^ temp = nullptr;

List^ cur = head->next;

while (cur!=nullptr)

{

temp = cur;

delete temp->fig;

delete temp;

cur = cur->next;

}

head->next = nullptr;

}

List^ push(List^ head, FunctionType fType, System::Drawing::Point points){ //добавление элемента в список

List^ cur = head;

while (cur->next!=nullptr)

{

cur = cur->next;

}

cur->next = gcnew List;

cur = cur->next;

cur->next = nullptr;

//add figure

cur->fig = figureFromType(fType,points); //создание фигуры исходя из её типа

return cur;

}

List^ pushExistingFunc(List^ head, AbstractFigure^ element) { //добавление в список существующей фигуры

List^ cur = head;

while (cur->next != nullptr)

{

cur = cur->next;

}

cur->next = gcnew List;

cur = cur->next;

cur->next = nullptr;

cur->fig = element;

return cur;

}

bool isElementInList(List^ head , List^ el) { //проверка содержится ли элемент в списке

List^ cur = head->next;

while (cur != nullptr){

if (cur->fig == el->fig)

return true;

cur = cur->next;

}

return false;

}

void deleteElement(List^ head, List^ elementToDelete) { //логическое удаление элемента из списка

List^ cur = head;

List^ curEl;

while (cur->next!=nullptr)

{

curEl = cur->next;

if (curEl->fig == elementToDelete->fig) {

cur->next = curEl->next;

return;

}

else

cur = cur->next;

}

}

AbstractFigure^ figureFromType(FunctionType fType, System::Drawing::Point points) { //создание фигуры на основе типа

RectFunc^ cur;

switch (fType)

{

case PROCESS:

cur = gcnew ProcessFunc(points);

break;

case DECISSION:

cur = gcnew DecissionFunc(points);

break;

case DATA:

cur = gcnew DataFunc(points);

break;

case TERMINATOR:

cur = gcnew TerminatorFunc(points);

break;

case PAGEREFERENCE:

cur = gcnew PageRefFunc(points);

break;

case TEXT:

cur = gcnew TextFunc(points);

break;

}

return cur;

}

//Line Model

List^ addLine(List^ head, System::Drawing::Point points) { //создание линии

List^ cur = head;

while (cur->next != nullptr)

{

cur = cur->next;

}

cur->next = gcnew List;

cur = cur->next;

cur->next = nullptr;

LineFunc^ line = gcnew LineFunc;

line->headLine = new Points;

line->headLine->next = nullptr;

line->width = 2.0f;

addPoints(line,points); //добавление точек

cur->fig = line;

return cur;

}

Points\* addPoints(LineFunc^ line, System::Drawing::Point points) { //добавление точек к существующей линии

Points \*cur = line->headLine;

while (cur->next != nullptr)

{

cur = cur->next;

}

int curX, curY;

if (cur != line->headLine) {

curX = cur->x;

curY = cur->y;

// Запрещаем проводить прямую под углом

if (((points.X - curX) != 0) && (atan(abs((points.Y - curY) / (points.X - curX))) < M\_PI\_4)) {

points.Y = curY;

}

else

{

points.X = curX;

}

}

cur->next = new Points;

cur = cur->next;

cur->next = nullptr;

cur->x = points.X;

cur->y = points.Y;

return cur;

}

List^ hitTest(List^ head, System::Drawing::Point points) { //нахождение фигуры по клику на нее

int closeObject = 5;

if (head == nullptr) {

return nullptr;

}

List^ cur = head->next;

while (cur != nullptr)

{

if (cur->fig->type != LINE) {

RectFunc^ curF = (RectFunc^)cur->fig;

if ((points.X >= curF->leftCoords.X) && (points.X <= curF->rightCoords.X) && (points.Y >= curF->leftCoords.Y) && //анализируем область

(points.Y <= curF->rightCoords.Y)) {

return cur;

}

}

else

{

LineFunc^ curL = (LineFunc^)cur->fig;

Points\* curP = curL->headLine->next;

while (curP != nullptr)

{

if ((abs(points.X - curP->x) <= closeObject) && // если это вершина

(abs(points.Y - curP->y) <= closeObject)) {

return cur;

}

if (curP->next != nullptr) { // если проверяем точку на линии

if (((abs(points.Y - curP->y) <= closeObject) && // горизонтальная

(points.X >= min(curP->x, curP->next->x)) &&

(points.X <= max(curP->x, curP->next->x))) ||

((abs(points.X - curP->x) <= closeObject) && // вертикальная

(points.Y >= min(curP->y, curP->next->y)) &&

(points.Y <= max(curP->y, curP->next->y)))) {

return cur;

};

}

curP = curP->next;

}

}

cur = cur->next;

}

return nullptr;

}

int max(int first, int second) {

if (first > second)

return first;

else

return second;

}

int min(int first, int second) {

if (first < second)

return first;

else

return second;

}

void swap(int &first, int &second) {

int t = first;

first = second;

second = t;

}

void getSelectedFuncFromRect(List^ head, List^ selectedHead, Rect rect) { //обнаружение всех фигур из выделенной области

if (rect.right < rect.left) {

swap(rect.right, rect.left);

}

if (rect.top < rect.bottom) {

swap(rect.top, rect.bottom);

}

List^ cur = head->next;

while (cur!=nullptr)

{

if (cur->fig->type != LINE) {

RectFunc^ curF = (RectFunc^)cur->fig;

if ((((curF->leftCoords.X > rect.left) && (curF->leftCoords.X < rect.right)) // попала ли фигура в область

|| ((curF->rightCoords.X > rect.left)) && (curF->rightCoords.X < rect.right))

&& (((curF->leftCoords.Y < rect.top) && (curF->leftCoords.Y > rect.bottom)

) || ((curF->rightCoords.Y < rect.top) &&

(curF->rightCoords.Y > rect.bottom)))) {

pushExistingFunc(selectedHead, curF);

}

}

else

{

LineFunc^ curL = (LineFunc^)cur->fig;

Points\* curP = curL->headLine->next;

while (curP != nullptr)

{

if ((curP->x > rect.left) && (curP->x < rect.right) && // попала ли точка линии в область

(curP->y < rect.top) && (curP->y > rect.bottom)) {

//добавление фигуры

pushExistingFunc(selectedHead, curL);

break;

}

curP = curP->next;

}

}

cur = cur->next;

}

}

void transformFunc(List^ cur, System::Drawing::Point startPoints, System::Drawing::Point endPoints, EditMode mode) {

int dX = endPoints.X - startPoints.X; //изменение положение/размеров фигур

int dY = endPoints.Y - startPoints.Y;

if (cur != nullptr) {

switch (mode)

{

case MOVE:

{

if (cur->fig->type != LINE) {

RectFunc^ curF = (RectFunc^)cur->fig; // перемещение фигуры

curF->leftCoords.X -= dX;

curF->rightCoords.X -= dX;

curF->leftCoords.Y -= dY;

curF->rightCoords.Y -= dY;

}

else

{

LineFunc^ curL = (LineFunc^)cur->fig;

Points\* curP = curL->headLine->next;

while (curP != nullptr) {

curP->x = curP->x - dX; // перемещение линии

curP->y = curP->y - dY;

curP = curP->next;

}

}

break;

}

case TOP:

{

// смещаем верхнюю сторону

RectFunc^ curF = (RectFunc^)cur->fig;

curF->leftCoords.Y = curF->leftCoords.Y - dY;

break;

}

case BOTTOM:

{

RectFunc^ curF = (RectFunc^)cur->fig;

// Смещаем нижнюю сторону

curF->rightCoords.Y = curF->rightCoords.Y - dY;

break;

}

case RIGHT:

{

RectFunc^ curF = (RectFunc^)cur->fig;

// Смещаем правую сторону

curF->rightCoords.X = curF->rightCoords.X - dX;

break;

}

case LEFT:

{

RectFunc^ curF = (RectFunc^)cur->fig;

curF->leftCoords.X = curF->leftCoords.X - dX;

break;

}

case LEFT\_TOP:

{

RectFunc^ curF = (RectFunc^)cur->fig;

curF->leftCoords.X = curF->leftCoords.X - dX;

curF->leftCoords.Y = curF->leftCoords.Y - dY;

break;

}

case RIGHT\_TOP:

{

RectFunc^ curF = (RectFunc^)cur->fig;

curF->rightCoords.X = curF->rightCoords.X - dX;

curF->leftCoords.Y = curF->leftCoords.Y - dY;

break;

}

case LEFT\_BOTTOM:

{

RectFunc^ curF = (RectFunc^)cur->fig;

curF->leftCoords.X = curF->leftCoords.X - dX;

curF->rightCoords.Y = curF->rightCoords.Y - dY;

break;

}

case RIGHT\_BOTTOM:

{

RectFunc^ curF = (RectFunc^)cur->fig;

curF->rightCoords.X = curF->rightCoords.X - dX;

curF->rightCoords.Y = curF->rightCoords.Y - dY;

break;

}

}

}

}

EditMode getCurEditMode(List^ head,List^ selectedFunc, System::Drawing::Point points) { // определяем текущий режим редактирования

if (head->next == nullptr)

return NOT\_EDIT;

List^ cur = head->next;

while (cur != nullptr) {

if (cur->fig->type != LINE) {

RectFunc^ curF = (RectFunc^)cur->fig;

if ((selectedFunc != nullptr) && (curF == selectedFunc->fig)) {

if ((abs(points.Y - curF->leftCoords.Y) < (closeObject)) &&

(abs(points.X - curF->leftCoords.X) < (closeObject)))

// Левая верхняя вершина

return LEFT\_TOP;

if ((abs(points.Y - curF->leftCoords.Y) < (closeObject)) &&

(abs(points.X - curF->rightCoords.X) < (closeObject)))

// Правая верхняя вершина

return RIGHT\_TOP;

if ((abs(points.Y - curF->rightCoords.Y) < (closeObject)) &&

(abs(points.X - curF->leftCoords.X) < (closeObject))) {

// Левая нижняя вершина

return LEFT\_BOTTOM;

}

if ((abs(points.Y - curF->rightCoords.Y) < (closeObject)) &&

(abs(points.X - curF->rightCoords.X) < (closeObject)))

// Правая нижняя вершина

return RIGHT\_BOTTOM;

if ((points.X >= curF->leftCoords.X) && (points.X <= curF->rightCoords.X) && ((abs(points.Y - curF->leftCoords.Y) < (closeObject))

|| (abs(points.Y - curF->rightCoords.Y) < (closeObject))))

// Горизонтальная сторона

if (abs(points.Y - curF->leftCoords.Y) < (closeObject))

return TOP;

else

return BOTTOM;

if ((points.Y >= curF->leftCoords.Y) && (points.Y <= curF->rightCoords.Y) && ((abs(points.X - curF->leftCoords.X) < (closeObject))

|| (abs(points.X - curF->rightCoords.X) < (closeObject)))) {

// Вертикальная сторона

if (abs(points.X - curF->leftCoords.X) < (closeObject))

return LEFT;

else

return RIGHT;

}

}

if ((curF->leftCoords.X < points.X) && (curF->rightCoords.X > points.X) &&

(curF->leftCoords.Y < points.Y) && (curF->rightCoords.Y > points.Y))

{

return MOVE;

}

}

else

{ // для линии

LineFunc^ curL = (LineFunc^)cur->fig;

Points\* curP = curL->headLine->next;

while (curP != nullptr) {

if (curP->next != nullptr)

if (((abs(points.Y - curP->y) <= closeObject) && // горизонтальная

(points.X >= min(curP->x, curP->next->x)) &&

(points.X <= max(curP->x, curP->next->x))) ||

((abs(points.X - curP->x) <= closeObject) && // вертикальная

(points.Y >= min(curP->y, curP->next->y)) &&

(points.Y <= max(curP->y, curP->next->y))))

return MOVE;

curP = curP->next;

}

}

cur = cur->next;

}

return NOT\_EDIT;

}

System::String^ pointsToStr(System::Drawing::Point points) { //перевод точек в строку

System::String^ res = (points.X).ToString();

res = res + "|" + (points.Y).ToString()+"|";

return res;

}

void saveToFile(List^ head, System::String^ path) { // сохранение в файл

List^ cur = head->next;

System::IO::StreamWriter^ file = gcnew System::IO::StreamWriter(path);

file->WriteLine("RKZFILE"); // уникальное начало файла

while (cur!=nullptr)

{

if (cur->fig->type != LINE) {

RectFunc^ curF = (RectFunc^)cur->fig;

System::String^ res = "";

res += functionTypeToString(curF->type) + "|"; // помещаем тип

res += curF->text + "|"; // помещаем тест

res += pointsToStr(curF->leftCoords);

res += pointsToStr(curF->rightCoords); // помещаем координаты

file->WriteLine(res);

}

else

{

LineFunc^ line = (LineFunc^)cur->fig;

Points\* points = line->headLine->next;

System::String^ res = "";

res += functionTypeToString(line->type) + "|"; // помещаем тип

while (points!=nullptr)

{

res += pointsToStr(System::Drawing::Point(points->x, points->y)); // помещаем координаты

points = points->next;

}

file->WriteLine(res);

}

cur = cur->next;

}

file->Close();

}

System::Drawing::Point parseStringToPoints(System::String^ &str) { //выделяем из строки точки

System::Drawing::Point points;

int pos = str->IndexOf("|");

int len = str->Length - pos;

System::String^ temp = str->Remove(pos, len);

str = str->Remove(0, pos + 1);

points.X = int::Parse(temp); //по Х

pos = str->IndexOf("|");

len = str->Length - pos;

temp = str->Remove(pos, len);

str = str->Remove(0, pos + 1);

points.Y = int::Parse(temp); //по У

return points;

}

bool readFromFile(List^ head, System::String^ path) { //чтение из файла

System::String^ res = "";

if (System::IO::File::Exists(path)) {

System::IO::StreamReader^ file = gcnew System::IO::StreamReader(path);

res = file->ReadLine();

if (res == "RKZFILE") { //определяем есть ли уникальная метка файла

while (!file->EndOfStream)

{

res = file->ReadLine();

switch ((FunctionType)(res[0] - '0')) // выделяем тип фигуры

{

case LINE:{

res = res->Remove(0, 2);

List^ line = addLine(head, parseStringToPoints(res));

while (res->Length != 0) {

System::Drawing::Point points = parseStringToPoints(res);

addPoints((LineFunc^)line->fig, points);

}

break;

}

default: //если не линия,то прямоугольная фигура

AbstractFigure^ fig = figureFromType((FunctionType)(res[0]-'0'), System::Drawing::Point(0, 0));

fig->type = (FunctionType)(res[0] - '0');

res = res->Remove(0, 2);

int pos = res->IndexOf("|");

RectFunc^ rf = (RectFunc^)fig;

int len = res->Length - pos;

System::String^ temp = res->Remove(pos,len);

rf->text = temp;

res = res->Remove(0, pos+1);

rf->leftCoords = parseStringToPoints(res);

rf->rightCoords = parseStringToPoints(res);

pushExistingFunc(head, fig);

break;

}

}

file->Close();

return true;

}

else {

file->Close(); // ошибки в случае неверной метки

System::Windows::Forms::MessageBox::Show("Не верный формат или файл поврежден", "Ошибка!",

System::Windows::Forms::MessageBoxButtons::OK, System::Windows::Forms::MessageBoxIcon::Warning);

return false;

}

}

else

return false;

}

void makeFuncOnOneLine(List^ head, List^ cur) { // автоматическое выравнивание фигуры по вертикали к существующим

RectFunc^ cFunc = (RectFunc ^)cur->fig;

int curX = cFunc->leftCoords.X + (cFunc->rightCoords.X - cFunc->leftCoords.X) / 2;

List^ curL = head->next;

while (curL != nullptr) {

if (curL->fig->type == LINE) {

curL = curL->next;

continue;

}

else if (cur != curL)

{

RectFunc^ rectFunc = (RectFunc^)curL->fig;

int nCurX = rectFunc->leftCoords.X + (rectFunc->rightCoords.X - rectFunc->leftCoords.X) / 2;

if (abs(nCurX - curX) <= closeObject \* 3) {

cFunc->leftCoords.X = rectFunc->leftCoords.X;

cFunc->rightCoords.X = rectFunc->rightCoords.X;

return;

}

}

curL = curL->next;

}

}

void makeSmoothLine(List^ curLine, System::Drawing::Point newPoints, System::Drawing::Point prevPoints) { //выравнивание линии при автоматическом изменении координат её точек

LineFunc^ line = (LineFunc^)curLine->fig;

Points\* curP = line->headLine->next;

while (curP != nullptr) {

if (((curP->x != newPoints.X) && (curP->y != newPoints.Y)) &&

(((curP->y == prevPoints.Y)) || ((curP->x == prevPoints.X)) ||

((curP->x == prevPoints.X) && (curP->y == prevPoints.Y))))

{

if (curP->y == prevPoints.Y)

curP->y = newPoints.Y;

if (curP->x == prevPoints.X)

curP->x = newPoints.X;

}

curP = curP->next;

}

}

void makePointOnCenter(List^ head, List^ curL) { // крепление линии к центру фигуры

bool fF = false;

bool fE = false;

Points pP;

List^ cur = head->next;

LineFunc^ line = (LineFunc^)curL->fig;

Points \*curP = line->headLine->next;

if (curP == nullptr)

return;

Points \*curPF = curP;

while (curP->next != nullptr)

{

curP = curP->next;

}

Points\* curPE = curP;

while (cur!=nullptr) {

if (curL == cur)

{

cur = cur->next;

continue;

}

if (cur->fig->type != LINE)

{

RectFunc^ curF = (RectFunc^)cur->fig;

int curX = curF->leftCoords.X + (curF->rightCoords.X - curF->leftCoords.X) / 2; //для крепления к низу и верху

if (((abs(curPF->y - curF->rightCoords.Y) <= (closeObject+3 )) || // поиск подходящей координаты фигуры

(abs(curPF->y - curF->leftCoords.Y) <= (closeObject+3))) &&

(abs(curPF->x - curX) <= (closeObject+3 )) && (!fF))

{

fF = true;

pP.x = curPF->x;

pP.y = curPF->y;

if (abs(curPF->y - curF->rightCoords.Y) <= (closeObject+3))

curPF->y = curF->rightCoords.Y;

else

curPF->y = curF->leftCoords.Y;

curPF->x = curX;

makeSmoothLine(curL, System::Drawing::Point(curPF->x,curPF->y), System::Drawing::Point(pP.x, pP.y)); // выравнивание точек линии при прикреплении

}

else if (((abs(curPE->y - curF->rightCoords.Y) <= (closeObject +3)) || // поиск подходящей координаты фигуры

(abs(curPE->y - curF->leftCoords.Y) <= (closeObject+3 ))) &&

(abs(curPE->x - curX) <= (closeObject+3 )) && (!fE))

{

fE = true;

pP.x = curPE->x;

pP.y = curPE->y;

if (abs(curPE->y - curF->leftCoords.Y) <= (closeObject+3))

curPE->y = curF->leftCoords.Y;

else

curPE->y = curF->rightCoords.Y;

curPE->x = curX;

makeSmoothLine(curL, System::Drawing::Point(curPE->x, curPE->y), System::Drawing::Point(pP.x, pP.y));

}

if (fF && fE)

return;

int curY = curF->leftCoords.Y + (curF->rightCoords.Y - curF->leftCoords.Y) / 2; //для крепления к левой и правой стороне

if (((abs(curPF->x - curF->rightCoords.X) <= (closeObject + 3)) ||

(abs(curPF->x - curF->leftCoords.X) <= (closeObject + 3))) && // поиск подходящей координаты фигуры

(abs(curPF->y - curY) <= (closeObject + 3)) && (!fF))

{

fF = true;

pP.x = curPF->x;

pP.y = curPF->y;

if (abs(curPF->x - curF->rightCoords.X) <= (closeObject + 3))

curPF->x = curF->rightCoords.X;

else

curPF->x = curF->leftCoords.X;

curPF->y = curY;

makeSmoothLine(curL, System::Drawing::Point(curPF->x, curPF->y), System::Drawing::Point(pP.x, pP.y));

}

else if (((abs(curPE->x - curF->rightCoords.X) <= (closeObject + 3)) ||

(abs(curPE->x - curF->leftCoords.X) <= (closeObject + 3))) && // поиск подходящей координаты фигуры

(abs(curPE->y - curY) <= (closeObject + 3)) && (!fE))

{

fE = true;

pP.x = curPE->x;

pP.y = curPE->y;

if (abs(curPE->x - curF->leftCoords.X) <= (closeObject + 3))

curPE->x = curF->leftCoords.X;

else

curPE->x = curF->rightCoords.X;

curPE->y = curY;

makeSmoothLine(curL, System::Drawing::Point(curPE->x, curPE->y), System::Drawing::Point(pP.x, pP.y));

}

if (fF && fE)

return;

}

cur = cur->next;

}

}

**LineFunc.h**

#pragma once

#include "AbstractFigure.h"

struct Points // структура точек линии

{

int x, y;

Points \*next;

};

// класс линии

ref class LineFunc :

public AbstractFigure

{

public:

int width;

Points\* headLine;

LineFunc();

~LineFunc();

void drawFigure(System::Windows::Forms::PaintEventArgs^ e, bool isVertex) override;

};

**LineFunc.cpp**

#include "LineFunc.h"

#include "PictureView.h"

LineFunc::LineFunc() {

this->type = LINE;

}

LineFunc::~LineFunc() {

}

const int arrowWidth = 7;

const int arrowLength = 14;

void LineFunc::drawFigure(System::Windows::Forms::PaintEventArgs^ e, bool isVertex){

// рисование линии

Points\* cur = this->headLine->next;

System::Drawing::Pen^ pen = gcnew System::Drawing::Pen(System::Drawing::Color::Black);

pen->Width = this->width;

Points\* lastPoint = nullptr;

Points\* prevLastPoint = nullptr;

if (cur != nullptr)

while (cur->next != nullptr)

{

// рисование по точкам

e->Graphics->DrawLine(pen,cur->x,cur->y,cur->next->x,cur->next->y);

if (isVertex) {

drawVertex(e, System::Drawing::Point(cur->x, cur->y), System::Drawing::Color::Black);

}

prevLastPoint = cur;

lastPoint = cur->next;

cur = cur->next;

}

// рисование стрелки в зависимости от направления линии

if (lastPoint!=nullptr && prevLastPoint!=nullptr) {

if (lastPoint->x == prevLastPoint->x) { // горизонтальная линия

if (lastPoint->y > prevLastPoint->y) {

e->Graphics->DrawLine(pen, lastPoint->x - arrowWidth, lastPoint->y - arrowLength, lastPoint->x, lastPoint->y);

e->Graphics->DrawLine(pen, lastPoint->x + arrowWidth, lastPoint->y - arrowLength, lastPoint->x, lastPoint->y);

}

else {

e->Graphics->DrawLine(pen, lastPoint->x - arrowWidth, lastPoint->y + arrowLength, lastPoint->x, lastPoint->y);

e->Graphics->DrawLine(pen, lastPoint->x + arrowWidth, lastPoint->y + arrowLength, lastPoint->x, lastPoint->y);

}

}

else { // вертикальная линия

if (lastPoint->x > prevLastPoint->x) {

e->Graphics->DrawLine(pen, lastPoint->x - arrowLength, lastPoint->y + arrowWidth, lastPoint->x, lastPoint->y);

e->Graphics->DrawLine(pen, lastPoint->x - arrowLength, lastPoint->y - arrowWidth, lastPoint->x, lastPoint->y);

}

else {

e->Graphics->DrawLine(pen, lastPoint->x + arrowLength, lastPoint->y + arrowWidth, lastPoint->x, lastPoint->y);

e->Graphics->DrawLine(pen, lastPoint->x + arrowLength, lastPoint->y - arrowWidth, lastPoint->x, lastPoint->y);

}

}

}

// рисование вершин

if (isVertex) {

drawVertex(e, System::Drawing::Point(cur->x, cur->y), System::Drawing::Color::Black);

}

delete pen;

}

**PictureView.h**

#pragma once

#include "FigureModel.h"

#include "LineFunc.h"

#include "RectFunc.h"

#include "DataTypes.h"

// основные методы для отрисовки

void cleanScreen(System::Windows::Forms::PaintEventArgs^ e, int width, int height);

void drawFunctions(System::Windows::Forms::PaintEventArgs^ e, List^ head,bool);

void drawPath(System::Windows::Forms::PaintEventArgs^,LineFunc^, System::Drawing::Point);

void drawSelectedRect(System::Windows::Forms::PaintEventArgs^ ,Rect);

void drawVertex(System::Windows::Forms::PaintEventArgs^,System::Drawing::Point, System::Drawing::Color);

void repaintSelectedFunc(System::Windows::Forms::PaintEventArgs^, List^, bool);

void drawText(System::Windows::Forms::PaintEventArgs^,RectFunc^);

**PictureView.cpp**

#include "PictureView.h"

#include "DataTypes.h"

#define \_USE\_MATH\_DEFINES

#include <math.h>

using namespace System::Drawing;

void cleanScreen(System::Windows::Forms::PaintEventArgs^ e, int width, int height){

e->Graphics->FillRectangle(Brushes::White,0,0,width,height); // очистка экрана белым цветом

}

void drawFunctions(System::Windows::Forms::PaintEventArgs^ e, List^ head,bool isVertex) {

List^ cur = head->next; // вызов отрисовки элементов

while (cur != nullptr)

{

cur->fig->drawFigure(e,isVertex);

cur = cur->next;

}

}

void drawPath(System::Windows::Forms::PaintEventArgs^ e, LineFunc^ line, System::Drawing::Point points) {

// отрисовка пути линии

if (line->headLine == nullptr) {

return;

}

Points \*cur = line->headLine;

while (cur->next != nullptr)

{

cur = cur->next;

}

int curX = cur->x;

int curY = cur->y;

// Запрещаем проводить прямую под углом

if (((points.X - curX) != 0) && (atan(abs((points.Y - curY) / (points.X - curX))) < M\_PI\_4)) {

points.Y = curY;

}

else

{

points.X = curX;

}

System::Drawing::Pen^ pen = gcnew System::Drawing::Pen(System::Drawing::Color::Black);

pen->Width = 2.0f;

pen->DashStyle = System::Drawing::Drawing2D::DashStyle::Dash;

e->Graphics->DrawLine(pen, cur->x, cur->y, points.X, points.Y);

delete pen;

}

void drawSelectedRect(System::Windows::Forms::PaintEventArgs^ e, Rect rect) {

// отрисовка выделительного прямоугольника

System::Drawing::Pen^ pen = gcnew System::Drawing::Pen(System::Drawing::Color::Black);

pen->DashStyle = System::Drawing::Drawing2D::DashStyle::Dash;

if ((rect.right > rect.left) && (rect.top > rect.bottom))

e->Graphics->DrawRectangle(pen, rect.left, rect.bottom, (rect.right - rect.left), (rect.top - rect.bottom));

else if ((rect.left > rect.right) && (rect.bottom > rect.top))

e->Graphics->DrawRectangle(pen, rect.right, rect.top, (rect.left - rect.right), (rect.bottom - rect.top));

else if (rect.right > rect.left)

e->Graphics->DrawRectangle(pen, rect.left, rect.top, (rect.right - rect.left), (rect.bottom - rect.top));

else if (rect.left > rect.right)

e->Graphics->DrawRectangle(pen, rect.right, rect.bottom, (rect.left - rect.right), (rect.top - rect.bottom));

delete pen;

}

void drawVertex(System::Windows::Forms::PaintEventArgs^ e,System::Drawing::Point points, System::Drawing::Color color) {

// отрисовка углов элементов

System::Drawing::Pen^ pen = gcnew System::Drawing::Pen(color);

pen->Width = 1.0f;

System::Drawing::Brush^ brush;

if (color == System::Drawing::Color::Black) {

// выбор цвета заливки

brush = gcnew System::Drawing::SolidBrush(System::Drawing::Color::White);

}

else

{

brush = gcnew System::Drawing::SolidBrush(color);

}

e->Graphics->DrawRectangle(pen, float(points.X - 4), float(points.Y - 4), 4\*sqrt(2), 4 \* sqrt(2));

e->Graphics->FillRectangle(brush, float(points.X - 3), float(points.Y - 3), 3 \* sqrt(2), 3 \* sqrt(2));

delete brush;

delete pen;

}

void repaintSelectedFunc(System::Windows::Forms::PaintEventArgs^ e, List^ selectedFunc,bool forWrite) {

// перерисовка углов для выделенной фигуры

System::Drawing::Color color;

if (forWrite) {

color = System::Drawing::Color::Green;

}

else {

color = System::Drawing::Color::Blue;

}

if (selectedFunc->fig->type != LINE) {

RectFunc^ curF = (RectFunc^)selectedFunc->fig;

drawVertex(e, System::Drawing::Point(curF->leftCoords.X, curF->leftCoords.Y), color);

drawVertex(e, System::Drawing::Point(curF->rightCoords.X, curF->leftCoords.Y), color);

if (curF->type != DATA) {

//отрисовка углов для всех фигур кроме даты

drawVertex(e, System::Drawing::Point(curF->leftCoords.X, curF->rightCoords.Y), color);

drawVertex(e, System::Drawing::Point(curF->rightCoords.X, curF->rightCoords.Y), color);

}

else {

//для даты

drawVertex(e, System::Drawing::Point(curF->rightCoords.X - 29, curF->rightCoords.Y), color);

drawVertex(e, System::Drawing::Point(curF->leftCoords.X - 29, curF->rightCoords.Y), color);

}

return;

}

else

{

//отрисовка углов для линии

LineFunc^ curL = (LineFunc^)selectedFunc->fig;

Points\* curP = curL->headLine->next;

while (curP!=nullptr)

{

drawVertex(e, System::Drawing::Point(curP->x, curP->y), System::Drawing::Color::Blue);

curP = curP->next;

}

}

}

void drawText(System::Windows::Forms::PaintEventArgs^ e,RectFunc^ cur) {

//отрисовка текста в фигуре

System::Drawing::RectangleF rectF = System::Drawing::RectangleF(cur->leftCoords.X,

cur->leftCoords.Y,

cur->rightCoords.X - cur->leftCoords.X,

cur->rightCoords.Y - cur->leftCoords.Y);

System::Drawing::Brush^ brush = gcnew System::Drawing::SolidBrush(System::Drawing::Color::Black);

System::Drawing::Font^ drawFont = gcnew System::Drawing::Font("Arial", 10); //выбор шрифта

System::Drawing::StringFormat^ drawFormat = gcnew System::Drawing::StringFormat();

if (!(cur->type == DATA))

drawFormat->Alignment = System::Drawing::StringAlignment::Center; //центрирование

else

drawFormat->Alignment = System::Drawing::StringAlignment::Near;

drawFormat->LineAlignment = System::Drawing::StringAlignment::Center;

e->Graphics->DrawString(cur->text, drawFont, brush, rectF, drawFormat); //вывод текста

delete drawFont;

delete drawFormat;

delete brush;

}

**RectFunc.h**

#pragma once

#include "AbstractFigure.h"

// базовый класс для всех фигур

ref class RectFunc : public AbstractFigure

{

public:

System::Drawing::Point leftCoords;

System::Drawing::Point rightCoords;

System::String^ text;

RectFunc();

RectFunc(System::Drawing::Point, System::Drawing::Point);

~RectFunc();

virtual void drawFigure(System::Windows::Forms::PaintEventArgs^ e, bool isVertex) override = 0;

};

**RectFunc.cpp**

#include "RectFunc.h"

RectFunc::RectFunc() {

this->text = nullptr;

}

RectFunc::~RectFunc() {

}

RectFunc::RectFunc(System::Drawing::Point leftP, System::Drawing::Point rightP) {

// конструктор для фигур

this->leftCoords = leftP;

this->rightCoords = rightP;

}

**RectFunctions.h**

#pragma once

#include "RectFunc.h"

// Классы для всех фигур

ref class ProcessFunc : RectFunc

{

public:

ProcessFunc();

ProcessFunc(System::Drawing::Point points);

~ProcessFunc();

void drawFigure(System::Windows::Forms::PaintEventArgs^ e,bool isVertex) override;

private:

};

ref class DecissionFunc : RectFunc

{

public:

DecissionFunc();

DecissionFunc(System::Drawing::Point);

~DecissionFunc();

void drawFigure(System::Windows::Forms::PaintEventArgs^ e, bool isVertex) override;

private:

};

ref class TerminatorFunc : RectFunc

{

public:

TerminatorFunc();

TerminatorFunc(System::Drawing::Point);

~TerminatorFunc();

void drawFigure(System::Windows::Forms::PaintEventArgs^ e, bool isVertex) override;

private:

};

ref class DataFunc : RectFunc

{

public:

DataFunc();

DataFunc(System::Drawing::Point);

~DataFunc();

void drawFigure(System::Windows::Forms::PaintEventArgs^ e, bool isVertex) override;

private:

};

ref class PageRefFunc : RectFunc

{

public:

PageRefFunc();

PageRefFunc(System::Drawing::Point);

~PageRefFunc();

void drawFigure(System::Windows::Forms::PaintEventArgs^ e, bool isVertex) override;

private:

};

ref class TextFunc : RectFunc

{

public:

TextFunc();

TextFunc(System::Drawing::Point);

~TextFunc();

void drawFigure(System::Windows::Forms::PaintEventArgs^ e, bool isVertex) override;

private:

};

**RectFunctions.cpp**

#include "RectFunctions.h"

#include "DataTypes.h"

#define \_USE\_MATH\_DEFINES

#include <math.h>

#include "PictureView.h"

ProcessFunc::ProcessFunc() {

}

ProcessFunc::ProcessFunc(System::Drawing::Point points)

{

// конструктор с начальными настройками

this->leftCoords.X = points.X - 44;

this->leftCoords.Y = points.Y - 24;

this->rightCoords.X = points.X + 44;

this->rightCoords.Y = points.Y + 24;

this->type = PROCESS;

}

ProcessFunc::~ProcessFunc()

{

}

void ProcessFunc::drawFigure(System::Windows::Forms::PaintEventArgs^ e,bool isVertex) {

// рисование фигуры и углов при надобности

System::Drawing::Pen^ pen = gcnew System::Drawing::Pen(System::Drawing::Color::Black);

pen->Width = 2.0f;

e->Graphics->DrawRectangle(pen, this->leftCoords.X, this->leftCoords.Y, (this->rightCoords.X - this->leftCoords.X),(this->rightCoords.Y - this->leftCoords.Y));

if (text != nullptr) {

drawText(e, this);

}

if (isVertex) {

drawVertex(e, System::Drawing::Point(this->leftCoords.X, this->leftCoords.Y), System::Drawing::Color::Black);

drawVertex(e, System::Drawing::Point(this->rightCoords.X, this->leftCoords.Y), System::Drawing::Color::Black);

drawVertex(e, System::Drawing::Point(this->leftCoords.X, this->rightCoords.Y), System::Drawing::Color::Black);

drawVertex(e, System::Drawing::Point(this->rightCoords.X, this->rightCoords.Y), System::Drawing::Color::Black);

}

delete pen;

}

DecissionFunc::DecissionFunc()

{

}

DecissionFunc::DecissionFunc(System::Drawing::Point points)

{

// конструктор с начальными настройками

this->leftCoords.X = points.X - 44;

this->leftCoords.Y = points.Y - 24;

this->rightCoords.X = points.X + 44;

this->rightCoords.Y = points.Y + 24;

this->type = DECISSION;

}

DecissionFunc::~DecissionFunc()

{

}

void DecissionFunc::drawFigure(System::Windows::Forms::PaintEventArgs^ e, bool isVertex) {

// рисование фигуры и углов при надобности

System::Drawing::Pen^ pen = gcnew System::Drawing::Pen(System::Drawing::Color::Black);

pen->Width = 2.0f;

array<System::Drawing::Point>^ poligonPoints = gcnew array<System::Drawing::Point>{

System::Drawing::Point(this->leftCoords.X, this->leftCoords.Y + (this->rightCoords.Y - this->leftCoords.Y) / 2),

System::Drawing::Point(this->leftCoords.X + (this->rightCoords.X- this->leftCoords.X)/2, (int)this->leftCoords.Y),

System::Drawing::Point(this->rightCoords.X, (int)this->leftCoords.Y + (this->rightCoords.Y - this->leftCoords.Y) / 2),

System::Drawing::Point(this->rightCoords.X - (this->rightCoords.X - this->leftCoords.X) / 2, (int)this->rightCoords.Y)};

e->Graphics->DrawPolygon(pen,poligonPoints);

if (text != nullptr) {

drawText(e, this);

}

if (isVertex) {

drawVertex(e, System::Drawing::Point(this->leftCoords.X, this->leftCoords.Y), System::Drawing::Color::Black);

drawVertex(e, System::Drawing::Point(this->rightCoords.X, this->leftCoords.Y), System::Drawing::Color::Black);

drawVertex(e, System::Drawing::Point(this->leftCoords.X, this->rightCoords.Y), System::Drawing::Color::Black);

drawVertex(e, System::Drawing::Point(this->rightCoords.X, this->rightCoords.Y), System::Drawing::Color::Black);

}

delete pen;

delete poligonPoints;

}

TerminatorFunc::TerminatorFunc()

{

}

TerminatorFunc::TerminatorFunc(System::Drawing::Point points)

{

// конструктор с начальными настройками

this->leftCoords.X = points.X - 44;

this->leftCoords.Y = points.Y - 16;

this->rightCoords.X = points.X + 44;

this->rightCoords.Y = points.Y + 16;

this->type = TERMINATOR;

}

TerminatorFunc::~TerminatorFunc()

{

}

void TerminatorFunc::drawFigure(System::Windows::Forms::PaintEventArgs^ e, bool isVertex) {

// рисование фигуры и углов при надобности

System::Drawing::Pen^ pen = gcnew System::Drawing::Pen(System::Drawing::Color::Black);

pen->Width = 2.0f;

e->Graphics->DrawArc(pen, this->leftCoords.X, this->leftCoords.Y, ((this->rightCoords.X - this->leftCoords.X)\*(4.0 / 11.0)),

(this->rightCoords.Y - this->leftCoords.Y), 90, 180);

e->Graphics->DrawLine(pen, this->leftCoords.X + (this->rightCoords.X - this->leftCoords.X) / 6,

this->leftCoords.Y+1, this->rightCoords.X - (this->rightCoords.X - this->leftCoords.X) / 6, this->leftCoords.Y+1);

e->Graphics->DrawArc(pen, this->rightCoords.X- ((this->rightCoords.X - this->leftCoords.X)\*(4.0/11.0)), this->leftCoords.Y,

((this->rightCoords.X - this->leftCoords.X) \* (4.0 / 11.0)),

(this->rightCoords.Y - this->leftCoords.Y),90,-180);

e->Graphics->DrawLine(pen, this->leftCoords.X+ (this->rightCoords.X - this->leftCoords.X)/6, this->rightCoords.Y,

this->rightCoords.X - (this->rightCoords.X - this->leftCoords.X)/6, this->rightCoords.Y);

if (text != nullptr) {

drawText(e, this);

}

if (isVertex) {

drawVertex(e, System::Drawing::Point(this->leftCoords.X, this->leftCoords.Y), System::Drawing::Color::Black);

drawVertex(e, System::Drawing::Point(this->rightCoords.X, this->leftCoords.Y), System::Drawing::Color::Black);

drawVertex(e, System::Drawing::Point(this->rightCoords.X, this->rightCoords.Y), System::Drawing::Color::Black);

drawVertex(e, System::Drawing::Point(this->leftCoords.X, this->rightCoords.Y), System::Drawing::Color::Black);

}

delete pen;

}

DataFunc::DataFunc()

{

}

DataFunc::DataFunc(System::Drawing::Point points)

{

// конструктор с начальными настройками

this->leftCoords.X = points.X - 29;

this->leftCoords.Y = points.Y - 24;

this->rightCoords.X = points.X + 59;

this->rightCoords.Y = points.Y + 24;

this->type = DATA;

}

DataFunc::~DataFunc()

{

}

void DataFunc::drawFigure(System::Windows::Forms::PaintEventArgs^ e, bool isVertex) {

// рисование фигуры и углов при надобности

System::Drawing::Pen^ pen = gcnew System::Drawing::Pen(System::Drawing::Color::Black);

pen->Width = 2.0f;

array<System::Drawing::Point>^ poligonPoints = gcnew array<System::Drawing::Point>{

System::Drawing::Point(this->leftCoords.X, this->leftCoords.Y),

System::Drawing::Point(this->rightCoords.X,this->leftCoords.Y),

System::Drawing::Point(this->rightCoords.X - 29, this->rightCoords.Y),

System::Drawing::Point(this->leftCoords.X - 29, this->rightCoords.Y)};

e->Graphics->DrawPolygon(pen, poligonPoints);

if (text != nullptr) {

drawText(e, this);

}

if (isVertex) {

drawVertex(e, System::Drawing::Point(this->leftCoords.X, this->leftCoords.Y), System::Drawing::Color::Black);

drawVertex(e, System::Drawing::Point(this->rightCoords.X, this->leftCoords.Y), System::Drawing::Color::Black);

drawVertex(e, System::Drawing::Point(this->rightCoords.X - 29, this->rightCoords.Y), System::Drawing::Color::Black);

drawVertex(e, System::Drawing::Point(this->leftCoords.X - 29, this->rightCoords.Y), System::Drawing::Color::Black);

}

delete pen;

delete poligonPoints;

}

PageRefFunc::PageRefFunc()

{

}

PageRefFunc::PageRefFunc(System::Drawing::Point points)

{

// конструктор с начальными настройками

this->leftCoords.X = points.X - 16;

this->leftCoords.Y = points.Y - 16;

this->rightCoords.X = points.X + 16;

this->rightCoords.Y = points.Y + 16;

this->type = PAGEREFERENCE;

}

PageRefFunc::~PageRefFunc()

{

}

void PageRefFunc::drawFigure(System::Windows::Forms::PaintEventArgs^ e, bool isVertex) {

// рисование фигуры и углов при надобности

System::Drawing::Pen^ pen = gcnew System::Drawing::Pen(System::Drawing::Color::Black);

pen->Width = 2.0f;

e->Graphics->DrawEllipse(pen, this->leftCoords.X, this->leftCoords.Y, (this->rightCoords.X - this->leftCoords.X),

(this->rightCoords.Y - this->leftCoords.Y));

if (text != nullptr) {

drawText(e, this);

}

if (isVertex) {

drawVertex(e, System::Drawing::Point(this->leftCoords.X, this->leftCoords.Y), System::Drawing::Color::Black);

drawVertex(e, System::Drawing::Point(this->rightCoords.X, this->leftCoords.Y), System::Drawing::Color::Black);

drawVertex(e, System::Drawing::Point(this->leftCoords.X, this->rightCoords.Y), System::Drawing::Color::Black);

drawVertex(e, System::Drawing::Point(this->rightCoords.X, this->rightCoords.Y), System::Drawing::Color::Black);

}

delete pen;

}

TextFunc::TextFunc() {

}

TextFunc::TextFunc(System::Drawing::Point points)

{

// конструктор с начальными настройками

this->leftCoords.X = points.X - 44;

this->leftCoords.Y = points.Y - 24;

this->rightCoords.X = points.X + 44;

this->rightCoords.Y = points.Y + 24;

this->type = PROCESS;

}

TextFunc::~TextFunc()

{

}

void TextFunc::drawFigure(System::Windows::Forms::PaintEventArgs^ e, bool isVertex) {

// рисование фигуры и углов при надобности

System::Drawing::Pen^ pen = gcnew System::Drawing::Pen(System::Drawing::Color::Black);

pen->Width = 2.0f;

if (text != nullptr) {

drawText(e, this);

}

if (isVertex) {

drawVertex(e, System::Drawing::Point(this->leftCoords.X, this->leftCoords.Y), System::Drawing::Color::Black);

drawVertex(e, System::Drawing::Point(this->rightCoords.X, this->leftCoords.Y), System::Drawing::Color::Black);

drawVertex(e, System::Drawing::Point(this->leftCoords.X, this->rightCoords.Y), System::Drawing::Color::Black);

drawVertex(e, System::Drawing::Point(this->rightCoords.X, this->rightCoords.Y), System::Drawing::Color::Black);

}

delete pen;

}

**UndoStack.h**

#pragma once

#include "FigureModel.h"

enum UndoTypes // типы действий ,которые можно отменить

{

FUNC\_MOVE\_UNDO,

LINE\_MOVE\_UNDO,

POINT\_ADD\_UNDO,

FIGURE\_INSERT\_UNDO,

DELETE\_UNDO,

};

ref struct UndoStack // структура стека

{

List^ figure;

UndoTypes type;

System::String^ str;

UndoStack^ next;

};

// основные функции для работы со стеком

UndoStack^ stackInit(UndoStack^ head);

bool stackIsEmpty(UndoStack^ head);

void pushStack(UndoStack^ head, List^ figure, UndoTypes type, System::String^ str);

UndoStack^ popStack(UndoStack^ head);

void cleanStack(UndoStack^ head);

void undoStackAction(UndoStack^ cur,List^ listHead);

**UndoStack.cpp**

#include "UndoStack.h"

#include "LineFunc.h"

#include "RectFunc.h"

UndoStack^ stackInit(UndoStack^ head) { // создание стека

head = gcnew UndoStack;

head->str = nullptr;

head->next = nullptr;

return head;

}

bool stackIsEmpty(UndoStack^ head) { // проверка стека на пустоту

if (head->next == nullptr)

return true;

return false;

}

void pushStack(UndoStack^ head, List^ figure, UndoTypes type,System::String^ str) { // добавление в стек

UndoStack^ newEl = gcnew UndoStack;

newEl->figure = figure;

newEl->type = type;

newEl->str = str;

newEl->next = nullptr;

if (stackIsEmpty(head)) {

head->next = newEl;

return;

}

newEl->next = head->next;

head->next = newEl;

}

UndoStack^ popStack(UndoStack^ head) { // изъятие из стека

if (!stackIsEmpty(head)) {

UndoStack^ cur = head->next;

head->next = head->next->next;

//delete vertex after using!

return cur;

}

else

return nullptr;

}

void cleanStack(UndoStack^ head) { // очистка стека

UndoStack^ cur = head->next;

while (cur != nullptr) {

UndoStack^ temp = cur;

cur = cur->next;

//delete temp->str;

delete temp;

}

}

void undoStackAction(UndoStack^ cur, List^ listHead){ // действия отмены

if (cur != nullptr) {

switch (cur->type){

// перебор различных вариантов и реагирование на них

case FUNC\_MOVE\_UNDO: {

RectFunc^ curF = (RectFunc^)cur->figure->fig;

System::String^ str = cur->str;

curF->leftCoords = parseStringToPoints(str);

curF->rightCoords = parseStringToPoints(str);

break;

}

case LINE\_MOVE\_UNDO: {

System::String^ str = cur->str;

LineFunc^ curL = (LineFunc^)cur->figure->fig;

Points\* pointsP = curL->headLine->next;

while (pointsP != nullptr) {

System::Drawing::Point points = parseStringToPoints(str);

pointsP->x = points.X;

pointsP->y = points.Y;

pointsP = pointsP->next;

}

}

break;

case POINT\_ADD\_UNDO: {

LineFunc^ line = (LineFunc^)cur->figure->fig;

Points\* points = line->headLine;

if (points->next != nullptr && points->next->next != nullptr)

{

while (points->next->next != nullptr) {

points = points->next;

}

delete points->next;

points->next = nullptr;

}

break;

}

case FIGURE\_INSERT\_UNDO:

deleteElement(listHead,cur->figure);

if (cur->figure->fig->type == LINE) {

LineFunc^ line = (LineFunc^)cur->figure->fig;

//delete line->headLine->next;

line->headLine->next = nullptr;

}

break;

case DELETE\_UNDO:

pushExistingFunc(listHead, cur->figure->fig);

break;

}

delete cur;

}

}

ВЕДОМОСТЬ

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Обозначение | | | | Наименование | | | | Дополнительные сведения | | | |
|  | | | | Текстовые документы | | | |  | | | |
|  | | | |  | | | |  | | | |
| БГУИР КР 1–40 01 01 403 ПЗ | | | | Пояснительная записка | | | | 95 с. | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | | Графические документы | | | |  | | | |
|  | | | |  | | | |  | | | |
| ГУИР 951004 403 СП | | | | Схема программы «Векторный редактор схем алгоритмов» | | | | Формат А1 | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  |  |  |  |  | БГУИР КР 1-40 01 01 403 Д1 | | | | | | |
|  |  |  |  |  |
|  |  |  |  |  |
| Изм. | Л. | № докум. | Подп. | Дата | Векторный редактор схем алгоритмов  Ведомость курсовой  работы |  | | | | Лист | Листов |
| Разраб. | | Козко Р.С. |  |  | Т |  | |  | 95 | 95 |
| Пров. | | Шостак Е.В. |  |  | Кафедра ПОИТ  гр. 951004 | | | | | |
|  | |  |  |  |
|  | |  |  |  |
|  | |  |  |  |