Национальный исследовательский Нижегородский государственный университет  
им. Н.И. Лобачевского

Институт информационных технологий, математики и механики

Отчёт по лабораторной работе:

«Графический редактор. Плекс»

Выполнил:

Студент 2 курса факультета ИТММ,

группы 3822Б1ПМ4

Лагутов Павел Андреевич

Проверила работу:

Барышева Ирина Викторовна

г. Нижний Новгород

2024 г.

|  |  |
| --- | --- |
| Содержание |  |
| Введение…………………………………………………………………………….……………3  Постановка задачи…………………………………………………………………….…………5  Структура проекта……………………………………………………………………….………6  Руководство пользователя………………………………………………………………….…...9  Заключение……………………………………………………………………………………....16  Литература…………………………………………………………………………………….....17  Приложение………………………………………………………………………..…………….18 |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

Введение

Лабораторная работа посвящена обработке графических данных, что является важной задачей в различных областях научно-технических и производственных приложений. В ходе выполнения данной работы студенты знакомятся с проблематикой структурного (векторного) представления графической информации. Этот подход к представлению изображений позволяет существенно повысить эффективность анализа и обработки графических данных, а также обеспечить возможность их редактирования и использования в различных программах и системах, включая системы автоматизированного проектирования.

Основные аспекты, рассматриваемые при выполнении лабораторной работы, включают в себя:

1. Изучение методов хранения графической информации на ЭВМ.

2. Освоение способов редактирования графических данных.

3. Изучение применения графических данных в расчетах и моделировании.

4. Понимание принципов формирования изображений на экране дисплея или на бумаге.

5. Ознакомление с принципами построения математических моделей на основе сетевых структур данных.

Целью данной лабораторной работы является ознакомление студентов с основными принципами представления и обработки графической информации на ЭВМ, а также приобретение навыков работы с различными графическими элементами и их взаимодействия в программных системах. В результате выполнения работы студенты получат практические навыки работы с графическими данными, что поможет им в дальнейшей профессиональной деятельности в области компьютерной графики и информационных технологий.

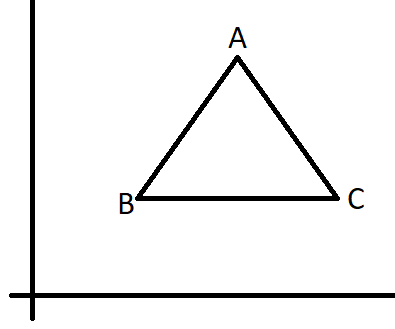
Лабораторная работа представляет собой введение в область обработки графических данных, которая является актуальной и востребованной в современном мире информационных технологий. Она предоставляет студентам возможность погрузиться в процесс работы с графическими данными, а также развить свои навыки анализа и обработки информации.

Под чертежом в рамках лабораторной работы называют графическое изображение моделируемого объекта, характеризующее его форму и размеры и выполненное по определенным правилам проектирования с применением общепринятых соглашений по изображению и обозначению графических элементам.

По чертежам изготовляют детали и производят сборку узлов, машин и конструкций. Чертеж на производстве – это основной технический документ, в котором указаны необходимые сведения о деталях, их материале, термической обработке, технические условия на изготовление. По нему проверяют качество обработки на всех стадиях технологического процесса. Различают чертеж детали, сборочный чертеж, чертеж общего вида и др.

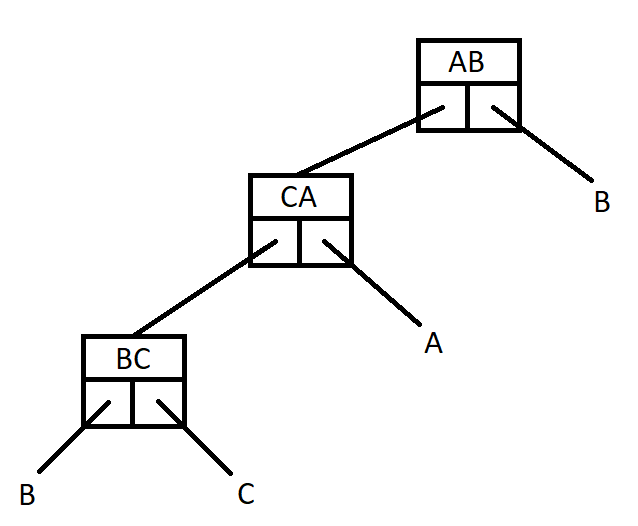
А также, в качестве чертежа будем рассматривать плоский геометрический объект, состоящий из отрезков прямых (линий) и граничных точек этих линий (представление других геометрических элементов может рассматриваться как тема дополнительных заданий). Набор линий, образующих чертеж, должен быть связным, т.е. любая линия чертежа должна иметь общую точку хотя бы с одной другой линией чертежа.

Пример возможного чертежа:



Для представления данных, описывающих структуру чертежа, предлагается использовать структуру хранения типа плекс. Плекс есть структура хранения, включающая звенья разных типов, отношения между которыми задаются при помощи сцепления. Таким образом, плекс является разновидностью многосвязного списка и используется для хранения сетевых моделей данных.

Основу плексов для представления чертежей, состоящих из линий и точек, составляют вершины (узлы), в каждой из которых располагается информация о той или иной линии чертежа (имя линия, координаты начальной и конечной точек).



Постановка задачи

В рамках лабораторной работы ставится задача разработки программного комплекса для представления, редактировании отображения на экране дисплея конкретного типа графических данных – чертежей, образуемых из ограниченного набора различных геометрических элементов (точек, линий, окружностей и т.п.).

Разрабатываемый комплекс должен обеспечивать:

* демонстрацию хранимых чертежей на экране дисплея;
* редактирование чертежей (вставку и удаление линий);
* запись (чтение) данных о чертежах в файлах.

А конкретнее, нам требуется разработать классы для работы с графическим редактором, плексом, обеспечивающим хранение данных и поддерживающим следующие операции:

1. Рисование точки
2. Изменение толщины и цвета точек
3. Рисование плекса и его редактирование
4. Сохранение плекса
5. Открытие плекса из файла

Структура проекта

В этом проекте имеются следующие классы:

1. Класс «Base.h»

Данный базовый класс объявляет типы элементов - точку и линию. Также в нём объявлены их свойства: тип, имя, цвет, толщина, кратность.  
Поля класса:

* TypeElem type; - Перечислимый тип
* string name; - Имя
* int color; - Цвет
* int width;- Толщина
* int rating; - Кратность

Методы класса:

* TypeElem getType() – Получить тип
* int getWidth() – Получить толщину
* int getRating() – Получить кратность
* string getName() – Получить имя
* int getColor() – Получить цвет
* void setType(TypeElem \_type) – Установить тип
* void setColor(int \_color) – Установить цвет
* void setRating(int \_rating) – Установить рейтинг
* void setName(const string& \_name) – Установить имя
* void setWidth(int \_width) – Установить толщину
* virtual void changeRating(int value = 1) – Установить рейтинг

1. Класс «Point.h»

Данный класс предназначен для типа точка. Он устанавливает свойства, предназначенные для точки, а именно её координаты, перемещение и рисование точки

Поля класса:

* Int x, y; - Координаты точки

Методы класса:

* int getX() – Получить координату по оси абцисса
* int getY() – Получить координату по оси ординат
* void MovePoint(int dx = 0, int dy = 0) – Сдвинуть точку на dx и dy
* void DecRating() – Уменьшить рейтинг на 1
* void IncRating() – Увеличить рейтинг на 1
* int identifyPoint(int cx, int cy) – Установить точка ли это
* void Draw(System::Drawing::Graphics^ g) – Рисование

1. Класс «Line.h»

Данный класс предназначен для типа линия. Он объявляет концы этой линии, две точки – левую и правую, а также и другие основные свойства для линии.

Поля класса:

* Base\* left; - Левый конец линии
* Base\* right; - Правый конец линии

Методы класса:

* void setLeft(Base\* \_left) – Установить левый конец линии
* void setRight(Base\* \_right) – Установить правый конец линии
* Base\* getLeft() – Получить левый конец линии
* Base\* getRight() – Получить правый конец линии
* override int identifyLine(int kx, int ky) – Установить линия ли это
* void inverse() – Поменять левый и правый концы линии местами
* void Draw(System::Drawing::Graphics^ g) – Рисование

1. Класс «Plex.h»

Данный класс объединяет точки и линии в один рисунок – плекс. Также добавлены функции сдвига, сохранения, удаления плекса и считывания из текстового документа.

Поля класса:

* Base\* start; - Начало плекса

Методы класса:

* void InsertLine(TLine\* le, TLine\* ri) – Вставить линию
* void addLine(TLine\* tm) – Добавить линию по двум точкам
* TPoint\* SearchPoint(const std::string& name) – Найти точку по имени
* void Move(int dx, int dy) – Сдвинуть плекс
* void Draw(System::Drawing::Graphics^ g) - Рисование
* void saveFile(std::string path) – Сохранить плекс
* void readFile(std::string path) – Считать плекс из файла (.txt)

1. Класс «Stack.h»

Класс представляет собой реализацию стека точек и линий. Благодаря шаблонному типу, данный класс позволяет использовать различные типы данных.

Поля класса:

* T\* arr; - Указатель на массив элементов стека
* unsigned int size; - Переменная, хранящая размер выделенной памяти под массив
* unsigned int last; - Переменная, хранящая индекс свободной ячейки в массиве

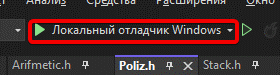
Методы класса:

* bool isEmpty() – Проверяет, пуст ли стек
* bool isFull() – Проверяет, заполнен ли стек
* T peek() – Возвращает значение верхнего элемента стека без его удаления
* T pop() – Удаляет и возвращает верхний элемент стека
* void push(T item) – Добавляет новый элемент на вершину стека.

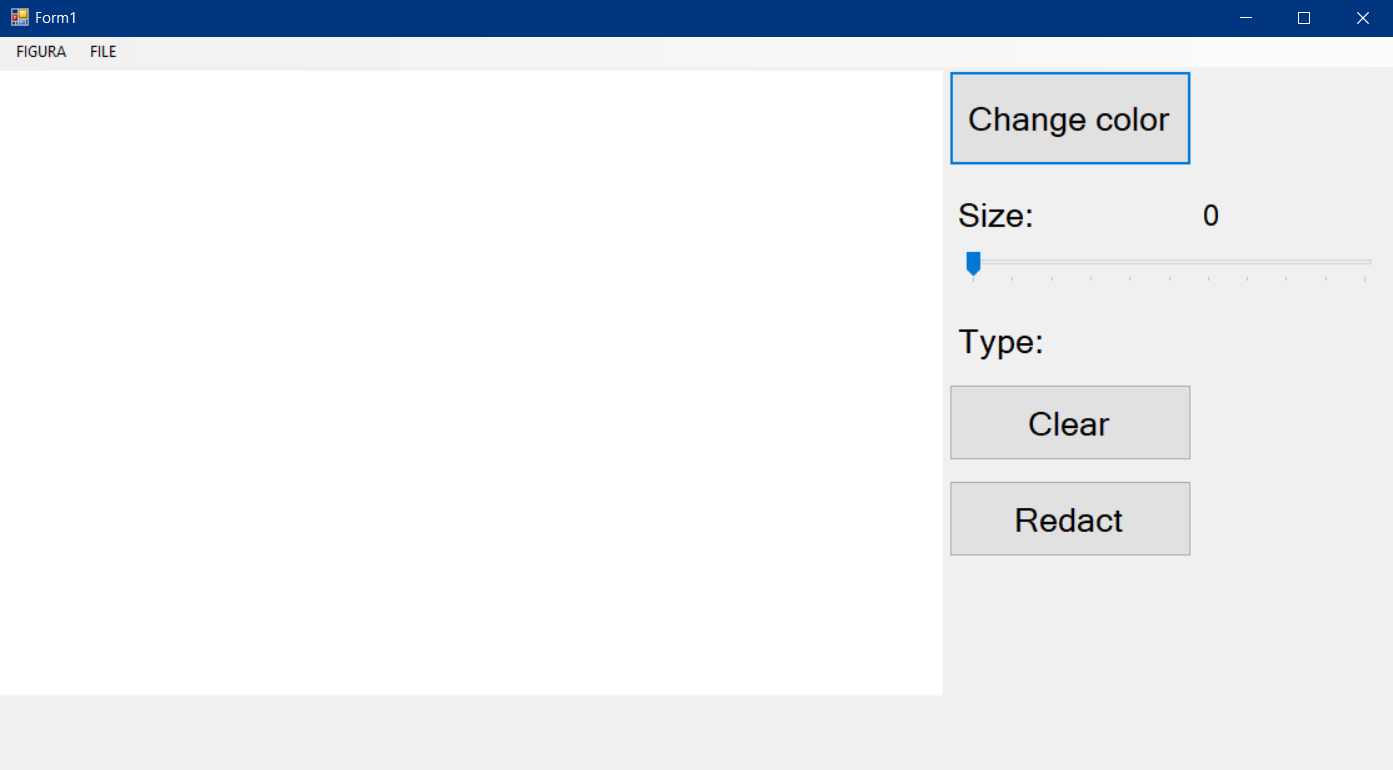
Руководство пользователя

1) Откройте файл «GraphicEditor.sln» в среде разработки Visual Studio 2022

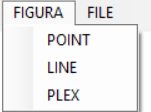
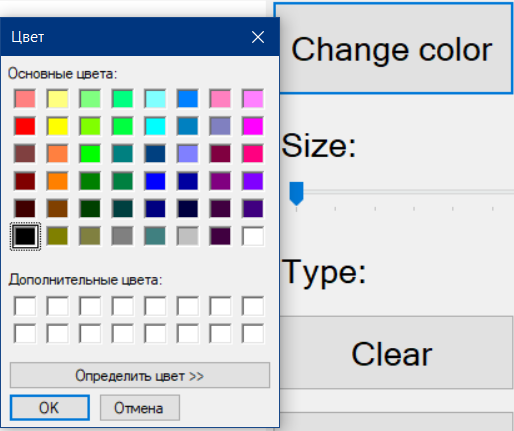
2) Запустите локальный отладчик:



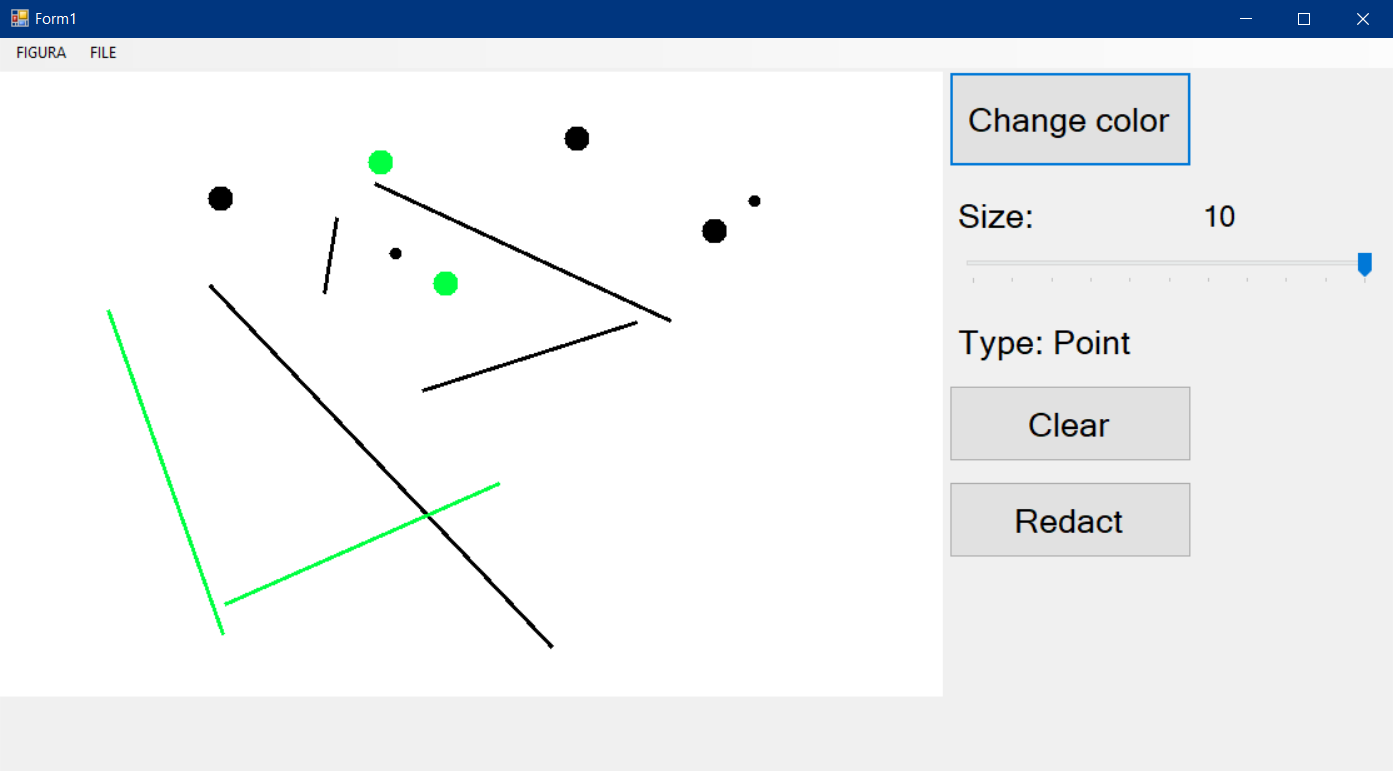
3) После этого на экране появится следующее окно:



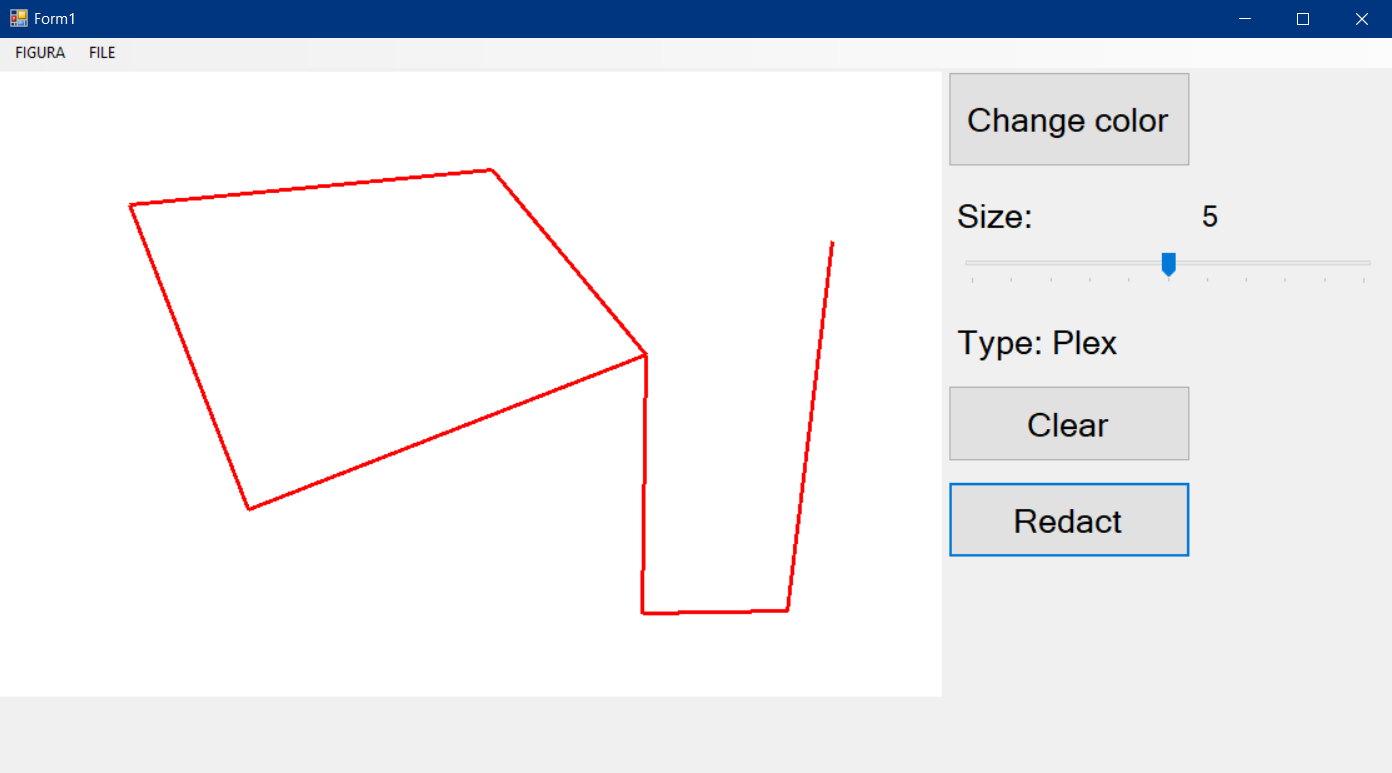
4) Вам нужно выбрать тип фигуры (линия, точка, плекс), так-же можно выбрать цвет и размеры фигуры



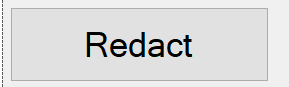
5) Добавление фигуры происходит после отпускания левой клавиши мыши



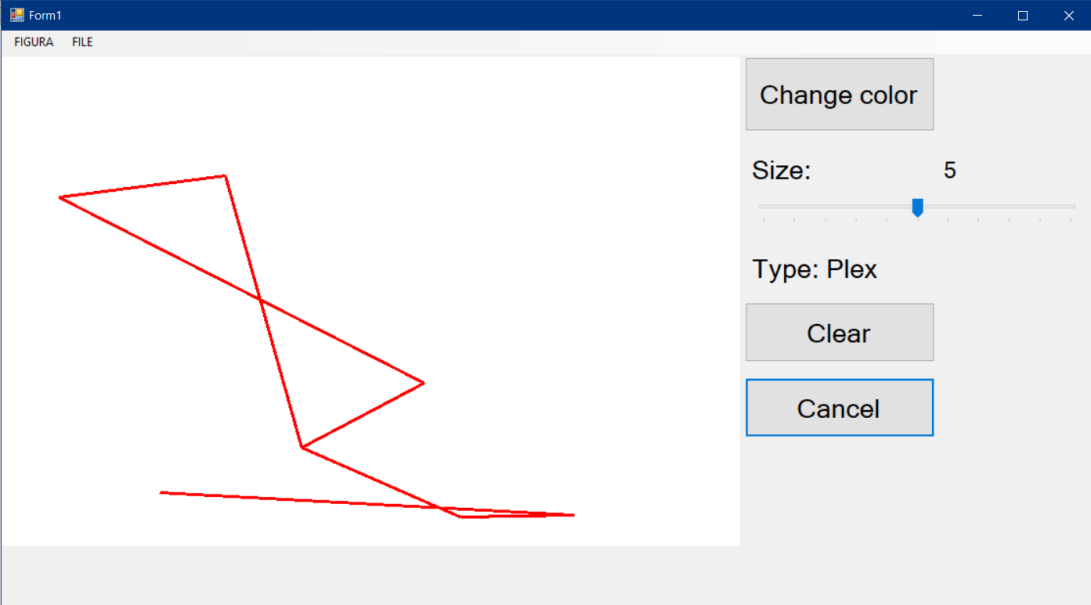
6) Перейдем к плексу. В плексе все точки связаны, поэтому рисовать надо из узлов



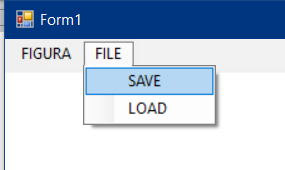
7) Разберемся с кнопкой – «Redact».



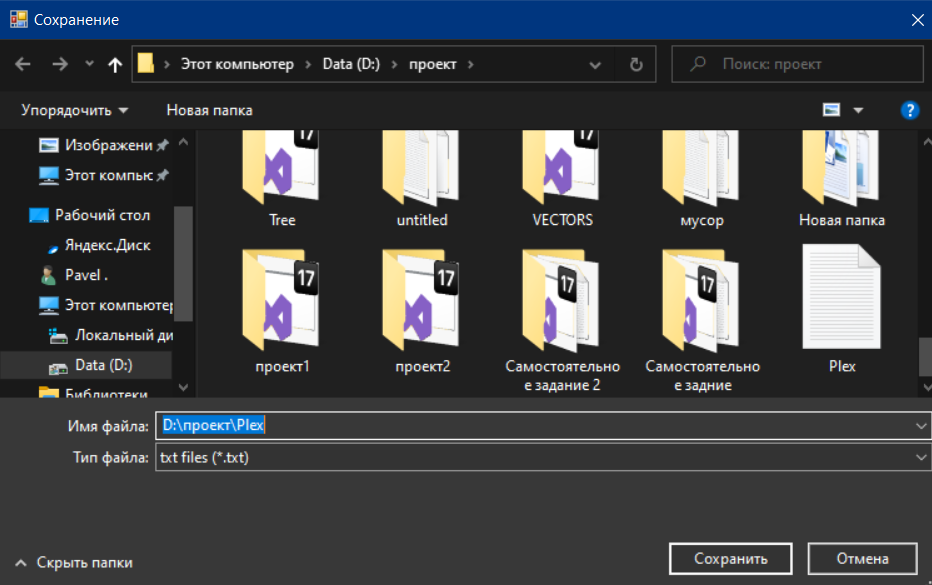
При нажатии на неё, вы сможете редактировать плекс(менять расположение узлов и двигать сам плекс):



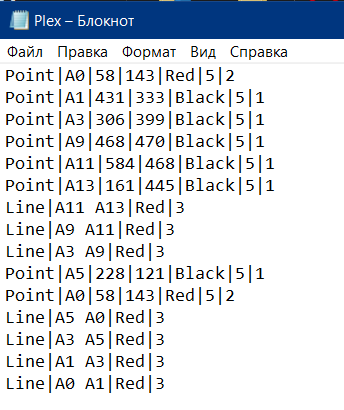
8) Следующий раздел – «FILE»: это поле позволяет работать с нашим плексом, а именно – сохранять в файл, открывать плекс из файла:



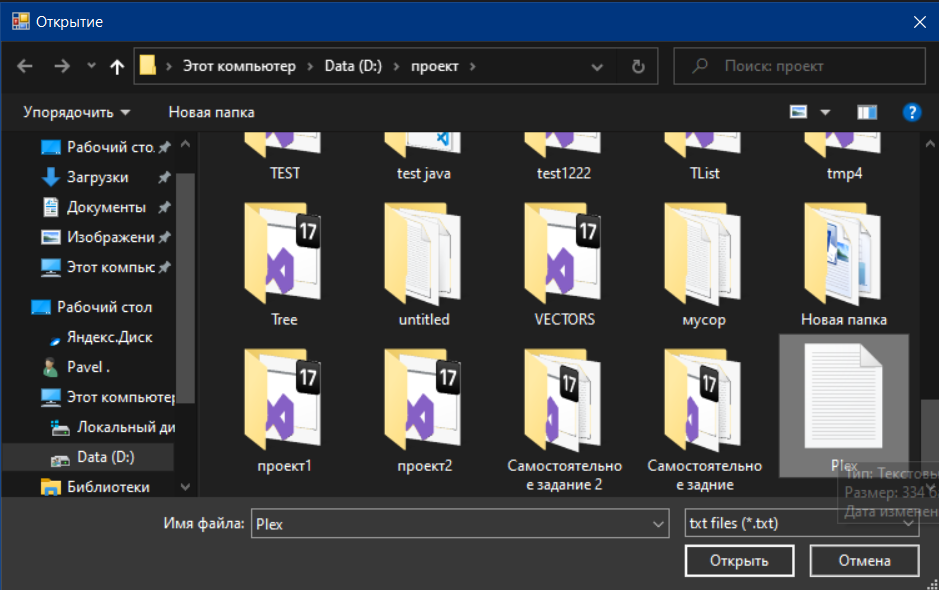
«SAVE» - позволяет сохранить наш плекс в файл. Нарисуйте любой замкнутый рисунок:

При нажатии на «Save», вы увидите окно, которое позволит вам назвать ваш файл и выбрать путь, по которому, в дальнейшем, вы найдете свой файл:

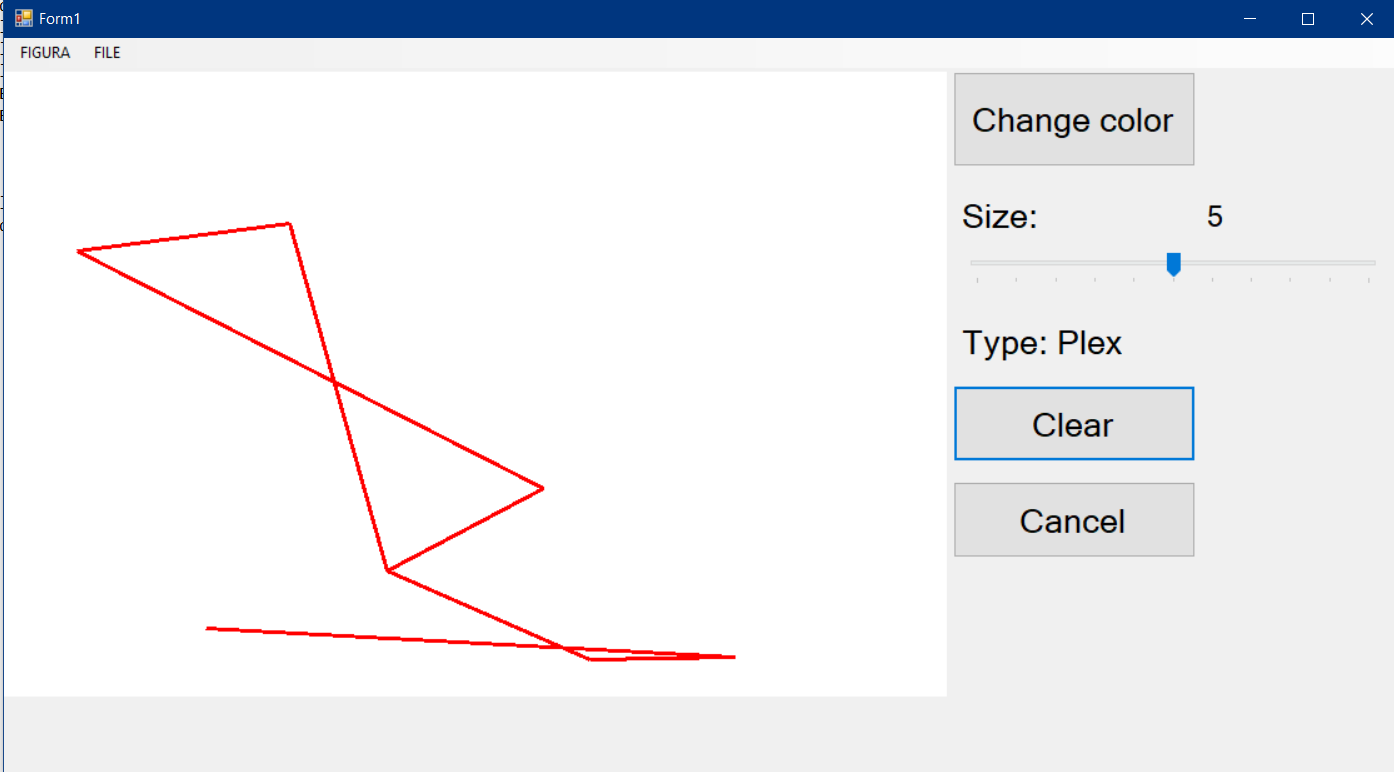
После выбора имени и нажатия на кнопку «Сохранить», вы увидите текстовый файл с вашим закодированным рисунком, который в дальнейшем может быть восстановлен:



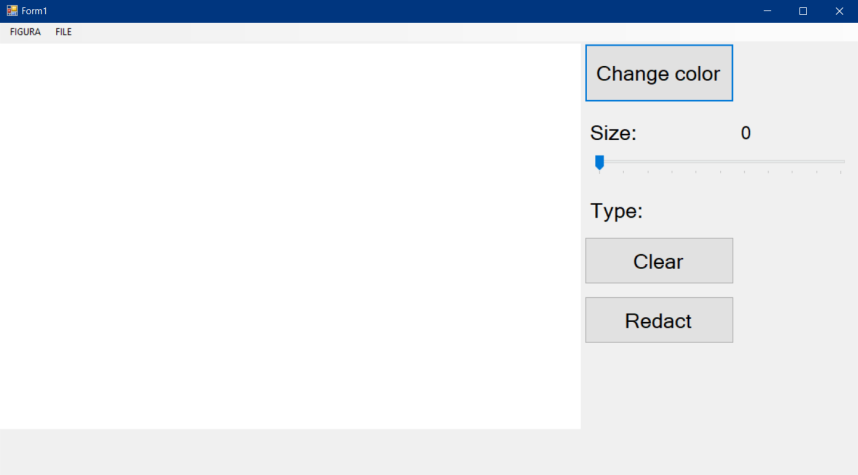
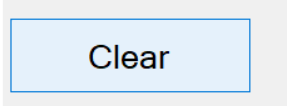
Следующая кнопка - «LOAD». Она позволяет восстановить плекс из ранее сохраненного файла:

Вы выбираете путь, по которому ранее сохраняли плекс, и выбираете файл:  


После нажатия на кнопку «Открыть», ваш плекс будет восстановлен:



Кнопка «Clear» позволяет отчистить PictureBox:



0-Заключение

В ходе выполнения лабораторной работы был разработан программный модуль для работы с геометрическими объектами, использующий структурное представление данных в виде плоских графических плексов. Основная цель работы состояла в создании функционала для представления, визуализации и обработки геометрических объектов, а также обеспечения удобного взаимодействия пользователя с программой.

На первом этапе был разработан и реализован базовый функционал модуля, включая классы для представления различных геометрических объектов и основные операции над ними. Далее был создан графический интерфейс пользователя, позволяющий визуализировать и взаимодействовать с объектами. Реализация алгоритмов обработки геометрических операций позволила осуществлять различные манипуляции с объектами.

Механизм ввода-вывода данных был реализован для обеспечения сохранения и загрузки геометрических объектов из файлов различных форматов, что повысило удобство использования программного модуля. После завершения разработки было проведено тестирование программы на различных наборах данных, что позволило выявить и исправить ошибки, а также оптимизировать работу модуля для повышения производительности.

В целом, выполнение лабораторной работы позволило приобрести практические навыки в разработке программного обеспечения для обработки графических данных, а также ознакомиться с основными аспектами работы с геометрическими объектами. Полученный опыт и знания могут быть полезными при решении различных инженерных и научно-технических задач, требующих обработки геометрических данных.Литература

1. Лабораторный практикум. / Барышева И.В., Мееров И.Б., Сысоев А.В., Шестакова Н.В. Под редакцией Гергеля В.П. Учебно-методическое пособие. – Нижний Новгород: Нижегородский госуниверситет, 2017. – 105с.
2. Дейтел П. Дж., Дейтел Х.М. Как программировать на С++. Введение в объектно-ориентированное проектирование с использованием UML. / Пер. с англ. - Москва: Издательство Бином, 2009. – 990 с.
3. Ирэ Пол ООП с использованием С++. – Киев, ДиаСофт, 1995.– 476 с. – ISBN: 5-7707-7219-0.
4. Ричард Вайнер, Льюис Пинсон, "С++ изнутри. – Киев, ДиаСофт, 1993. – 501 с. – ISBN: 5-87554-079.
5. Элджер Дж. С++: библиотека программиста. – СПб.: ЗАО Издательство Питер, 2009. – 259 с.

Приложения

* 1. «Base.h»

using System.Drawing;

using System.Security.AccessControl;

namespace GraphicEditor

{

public enum ShapeType { Point, Line, Plex};

public abstract class Shape

{

protected string name;

protected ShapeType type;

protected Color fillColor;

protected int size;

protected bool visible;

protected int identityZone;

protected int multiplicity;

public abstract void SetMultiplicity(int k);

public abstract bool HitTest(int x, int y);

public abstract void Draw(Graphics graphics);

public abstract void Shift(int deltaX, int deltaY = 0);

public string Name

{

get { return name; }

set { name = value; }

}

public ShapeType Type

{

get { return type; }

}

public Color FillColor

{

get { return fillColor; }

set { fillColor = value; }

}

public int Size

{

get { return size; }

set { SetSize(value); }

}

public bool Visible

{

get { return visible; }

set { visible = value; }

}

public int IdentityZone

{

get { return identityZone; }

set { identityZone = value; }

}

public int Multiplicity

{

get { return multiplicity; }

set { multiplicity = value; }

}

public void SetSize(int size)

{

identityZone = size \* 2;

this.size = size;

}

}

}

* 1. «Point.h»

using System;

using System.Drawing;

namespace GraphicEditor

{

public class CustomPoint : Shape

{

private int x, y;

public CustomPoint(string name, int x = 0, int y = 0, int size = 3)

{

this.name = name;

this.x = x;

this.y = y;

this.name = name;

this.size = size;

identityZone = size \* 2;

fillColor = Color.Black;

type = ShapeType.Point;

visible = true;

multiplicity = 1;

}

public override void SetMultiplicity(int k)

{

if (k > 0)

{

multiplicity++;

}

else

{

multiplicity--;

}

}

public override void Draw(Graphics g)

{

g.FillEllipse(new SolidBrush(fillColor), x - size, y - size, size \* 2, size \* 2);

}

public override void Shift(int deltaX, int deltaY = 0)

{

x += deltaX;

y += deltaY;

}

public override bool HitTest(int x, int y)

{

double distance = Math.Sqrt(Math.Pow(x - this.x, 2) + Math.Pow(y - this.y, 2));

return distance < identityZone;

}

public string PointToString()

{

return $"Point|{name}|{x}|{y}|{ColorToString()}|{size}|{multiplicity}";

}

private string ColorToString()

{

string temp = fillColor.ToString();

int startIndex = temp.IndexOf('[');

int endIndex = temp.IndexOf(']');

string result = temp.Substring(startIndex + 1, endIndex - startIndex - 1);

return result;

}

public int X

{

get { return x; }

set { x = value; }

}

public int Y

{

get { return y; }

set { y = value; }

}

}

}

* 1. «Line.h»

using System;

using System.Drawing;

namespace GraphicEditor

{

public class CustomLine : Shape {

protected Shape left;

protected Shape right;

public CustomLine(Shape left, Shape right, int size = 3) {

this.left = left;

this.right = right;

this.size = size;

identityZone = size \* 2;

fillColor = Color.Black;

type = ShapeType.Line;

visible = true;

multiplicity = 1;

name = left.Name + " " + right.Name;

}

public CustomLine() {

left = null;

right = null;

}

public CustomLine(string[] inputString) {

left = null;

right = null;

size = int.Parse(inputString[3]);

identityZone = size \* 4;

fillColor = Color.FromName(inputString[2]);

visible = true;

type = ShapeType.Line;

multiplicity = 1;

}

public CustomLine(CustomLine tmp)

{

left = null;

right = null;

name = tmp.name;

size = tmp.size;

identityZone = size \* 4;

fillColor = tmp.fillColor;

type = ShapeType.Line;

visible = true;

multiplicity = 1;

}

public CustomLine Invert()

{

CustomLine result = this;

Shape shape = result.left;

result.left = right;

result.right = shape;

result.name = left.Name + " " + right.Name;

return result;

}

public override void SetMultiplicity(int k = 0)

{

multiplicity++;

multiplicity = multiplicity % 4;

if (multiplicity == 0) multiplicity = 1;

}

public string LineToString()

{

return $"Line|{name}|{ColorToString()}|{size}";

}

private string ColorToString()

{

string temp = fillColor.ToString();

int startIndex = temp.IndexOf('[');

int endIndex = temp.IndexOf(']');

string result = temp.Substring(startIndex + 1, endIndex - startIndex - 1);

return result;

}

public override void Shift(int deltaX, int deltaY = 0)

{

left.Shift(deltaX, deltaY);

right.Shift(deltaX, deltaY);

}

public override bool HitTest(int x, int y)

{

double distance;

if (((CustomPoint)left).X == ((CustomPoint)right).X)

{

distance = Math.Abs(x - ((CustomPoint)left).X);

if (distance <= identityZone)

{

return true;

}

return false;

}

double k = (((CustomPoint)right).Y - ((CustomPoint)left).Y) / (((CustomPoint)right).X - ((CustomPoint)left).X);

double b = ((CustomPoint)left).Y - k \* ((CustomPoint)left).X;

distance = Math.Abs(k \* x - y + b) / Math.Sqrt(k \* k + 1);

if (distance <= identityZone)

{

return true;

}

return false;

}

public override void Draw(Graphics g)

{

Pen pen = new Pen(fillColor, size);

g.DrawLine(pen, ((CustomPoint)left).X, ((CustomPoint)left).Y, ((CustomPoint)right).X, ((CustomPoint)right).Y);

}

public Shape Left

{

get { return left; }

set { left = value; }

}

public Shape Right

{

get { return right; }

set { right = value; }

}

}

}

* 1. «Plex.h»

using System;

using System.Collections.Generic;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace GraphicEditor

{

internal class Plex

{

private Shape root;

public Plex()

{

root = null;

}

public Plex(Plex tmp)

{

root = new CustomLine((CustomLine)tmp.root);

Stack<CustomPoint> stackPoint = new Stack<CustomPoint>();

Stack<CustomLine> stackLine = new Stack<CustomLine>();

stackLine.Push(null);

Shape current = root;

while (current != null)

{

if (current.Type == ShapeType.Point)

{

stackPoint.Push((CustomPoint)current);

current = stackLine.Pop();

current.SetMultiplicity(1);

}

else

{

if (current.Multiplicity == 1)

{

stackLine.Push((CustomLine)current);

current = ((CustomLine)current).Left;

}

else if (current.Multiplicity == 2)

{

stackLine.Push((CustomLine)current);

current = ((CustomLine)current).Right;

}

else

{

CustomPoint right = stackPoint.Pop();

CustomPoint left = stackPoint.Peek();

switch (current.Type)

{

case ShapeType.Line:

CustomLine customLine = new CustomLine(left, right, current.Size);

customLine.FillColor = current.FillColor;

Add(customLine);

break;

}

current.SetMultiplicity(1);

current = stackLine.Pop();

if (current != null)

{

current.SetMultiplicity(1);

}

}

}

}

}

public Plex(string inputString)

{

CustomPoint[] multiplePoints = new CustomPoint[100];

int multiplePointsCount = 0;

Stack<Shape> stack = new Stack<Shape>();

string[] lines = inputString.Split('\n');

foreach (string line in lines)

{

if (string.IsNullOrWhiteSpace(line))

{

continue;

}

string[] parts = line.Split('|');

switch (parts[0])

{

case "Point":

CustomPoint customPoint = new CustomPoint(parts[1], int.Parse(parts[2]), int.Parse(parts[3]), int.Parse(parts[5]));

customPoint.Multiplicity = int.Parse(parts[6]);

Color fillColor = Color.FromName(parts[4]);

customPoint.FillColor = fillColor;

if (customPoint.Multiplicity > 1)

{

int index = -1;

int iterator = 0;

while (iterator < multiplePointsCount && index != -1)

{

if (multiplePoints[index].Name == customPoint.Name)

{

index = iterator;

}

iterator++;

}

if (index == -1)

{

stack.Push(customPoint);

multiplePoints[multiplePointsCount++] = customPoint;

}

else

{

stack.Push(multiplePoints[index]);

}

}

else

{

stack.Push(customPoint);

}

break;

case "Line":

Shape right = stack.Pop();

Shape left = stack.Pop();

CustomLine customLine = new CustomLine(parts);

customLine.Left = left;

customLine.Right = right;

stack.Push(customLine);

break;

}

}

root = stack.Pop();

}

public CustomPoint FindPoint(int x = 0, int y = 0)

{

Stack<CustomLine> stack = new Stack<CustomLine>();

stack.Push(null);

CustomPoint result = null;

Shape current = root;

while (current != null)

{

if (current.Type == ShapeType.Point)

{

if (((CustomPoint)current).HitTest(x, y))

{

result = (CustomPoint)current;

}

current = stack.Pop();

current.SetMultiplicity(1);

}

else

{

if (current.Multiplicity == 1)

{

stack.Push((CustomLine)current);

current = ((CustomLine)current).Left;

}

else if (current.Multiplicity == 2)

{

stack.Push((CustomLine)current);

current = ((CustomLine)current).Right;

}

else

{

current.SetMultiplicity(1);

current = stack.Pop();

if (current != null)

{

current.SetMultiplicity(1);

}

}

}

}

return result;

}

public CustomPoint FindPoint(CustomPoint point)

{

return FindPoint(point.X, point.Y);

}

public CustomLine FindLine(int x = 0, int y = 0)

{

Stack<CustomLine> stack = new Stack<CustomLine>();

stack.Push(null);

CustomLine result = null;

Shape current = root;

while (current != null)

{

if (current.Type == ShapeType.Point)

{

if (((CustomPoint)current).HitTest(x, y))

{

result = stack.Peek();

}

current = stack.Pop();

current.SetMultiplicity(1);

}

else

{

if (current.Multiplicity == 1)

{

stack.Push((CustomLine)current);

current = ((CustomLine)current).Left;

}

else if (current.Multiplicity == 2)

{

stack.Push((CustomLine)current);

current = ((CustomLine)current).Right;

}

else

{

current.SetMultiplicity(1);

current = stack.Pop();

if (current != null)

{

current.SetMultiplicity(1);

}

}

}

}

return result;

}

public CustomLine FindLine(CustomPoint point)

{

return FindLine(point.X, point.Y);

}

public CustomPoint GetPoint(Shape line)

{

Shape current = line;

while (current.Type != ShapeType.Point)

{

current = ((CustomLine)current).Left;

}

return (CustomPoint)current;

}

public void Add(CustomLine tmp)

{

if (root == null)

{

root = tmp;

return;

}

CustomLine leftLine = null;

CustomLine rightLine = null;

CustomPoint leftPoint = FindPoint((CustomPoint)tmp.Left);

CustomPoint rightPoint = FindPoint((CustomPoint)tmp.Right);

if (leftPoint != null) leftLine = FindLine(leftPoint);

if (rightPoint != null) rightLine = FindLine(rightPoint);

if (leftPoint != null)

{

if (leftLine.Left == leftPoint)

{

string[] name = GetName(tmp.Name);

name[0] = leftLine.Left.Name;

tmp.Left = leftLine.Left;

leftLine.Left = tmp;

tmp.Name = name[0] + " " + name[1];

}

else

{

string[] name = GetName(tmp.Name);

name[0] = leftLine.Right.Name;

tmp.Left = leftLine.Right;

leftLine.Right = tmp;

tmp.Name = name[0] + " " + name[1];

}

if (rightLine != null)

{

string[] name = GetName(tmp.Name);

name[1] = rightPoint.Name;

tmp.Right = rightPoint;

rightPoint.SetMultiplicity(1);

tmp.Name = name[0] + " " + name[1];

}

}

else if (rightLine != null)

{

tmp = tmp.Invert();

if (rightLine.Right == rightPoint)

{

string[] name = GetName(tmp.Name);

name[1] = rightLine.Right.Name;

tmp.Right = rightLine.Right;

rightLine.Right = tmp;

tmp.Name = name[0] + " " + name[1];

}

else

{

string[] name = GetName(tmp.Name);

name[0] = rightLine.Left.Name;

tmp.Left = rightLine.Left;

rightLine.Left = tmp;

tmp.Name = name[0] + " " + name[1];

}

}

}

public void Draw(Graphics graphics)

{

Stack<CustomPoint> stackPoint = new Stack<CustomPoint>();

Stack<CustomLine> stackLine = new Stack<CustomLine>();

stackLine.Push(null);

Shape current = root;

while (current != null)

{

if (current.Type == ShapeType.Point)

{

stackPoint.Push((CustomPoint)current);

current = stackLine.Pop();

current.SetMultiplicity(1);

}

else

{

if (current.Multiplicity == 1)

{

stackLine.Push((CustomLine)current);

current = ((CustomLine)current).Left;

}

else if (current.Multiplicity == 2)

{

stackLine.Push((CustomLine)current);

current = ((CustomLine)current).Right;

}

else

{

CustomPoint right = stackPoint.Pop();

CustomPoint left = stackPoint.Peek();

switch (current.Type)

{

case ShapeType.Line:

CustomLine customLine = new CustomLine(left, right, current.Size);

customLine.FillColor = current.FillColor;

customLine.Draw(graphics);

break;

//case ShapeType.Rectangle:

// CustomRectangle customRectangle = new CustomRectangle(left, right, current.Size, ((CustomRectangle)current).Angle);

// customRectangle.FillColor = current.FillColor;

// customRectangle.Draw(graphics);

// break;

}

current.SetMultiplicity(1);

current = stackLine.Pop();

if (current != null)

{

current.SetMultiplicity(1);

}

}

}

}

}

public string SaveToString()

{

string result = "";

Stack<CustomLine> stackLine = new Stack<CustomLine>();

stackLine.Push(null);

Shape current = root;

while (current != null)

{

if (current.Type == ShapeType.Point)

{

result += $"{((CustomPoint)current).PointToString()}\n";

current = stackLine.Pop();

current.SetMultiplicity(1);

}

else

{

if (current.Multiplicity == 1)

{

stackLine.Push((CustomLine)current);

current = ((CustomLine)current).Left;

}

else if (current.Multiplicity == 2)

{

stackLine.Push((CustomLine)current);

current = ((CustomLine)current).Right;

}

else

{

switch

(current.Type)

{

case ShapeType.Line:

result += $"{((CustomLine)current).LineToString()}\n";

break;

//case ShapeType.Rectangle:

// result += $"{((CustomRectangle)current).RectangleToString()}\n";

// break;

}

current.SetMultiplicity(1);

current = stackLine.Pop();

if (current != null)

{

current.SetMultiplicity(1);

}

}

}

}

return result;

}

private string[] GetName(string name)

{

string[] result = name.Split(' ');

return result;

}

public void Shift(int deltaX, int deltaY = 0)

{

Shape[] multipleShape = new Shape[100];

Stack<CustomLine> stack = new Stack<CustomLine>();

stack.Push(null);

Shape current = root;

int j = 0;

while (current != null)

{

if (current.Type == ShapeType.Point)

{

bool a = true;

for (int i = 0; i < j; i++)

{

if (multipleShape[i] == current)

{

a = false;

}

}

if (a)

{

multipleShape[j] = current;

j++;

((CustomPoint)current).Shift(deltaX, deltaY);

}

current = stack.Pop();

current.SetMultiplicity(1);

}

else

{

if (current.Multiplicity == 1)

{

stack.Push((CustomLine)current);

current = ((CustomLine)current).Left;

}

else if (current.Multiplicity == 2)

{

stack.Push((CustomLine)current);

current = ((CustomLine)current).Right;

}

else

{

current.SetMultiplicity(1);

current = stack.Pop();

if (current != null)

{

current.SetMultiplicity(1);

}

}

}

}

}

public void Fill(Color FillColor)

{

Shape[] multipleShape = new Shape[100];

Stack<CustomLine> stack = new Stack<CustomLine>();

stack.Push(null);

Shape current = root;

while (current != null)

{

if (current.Type == ShapeType.Line)

{

current.FillColor = FillColor;

}

if (current.Type == ShapeType.Point)

{

current = stack.Pop();

current.SetMultiplicity(1);

}

else

{

if (current.Multiplicity == 1)

{

stack.Push((CustomLine)current);

current = ((CustomLine)current).Left;

}

else if (current.Multiplicity == 2)

{

stack.Push((CustomLine)current);

current = ((CustomLine)current).Right;

}

else

{

current.SetMultiplicity(1);

current = stack.Pop();

if (current != null)

{

current.SetMultiplicity(1);

}

}

}

}

}

}

}

* 1. «Stack.h»

#pragma once

template <typename T>

class TStack

{

private:

T\* arr;

unsigned int size;

unsigned int last;

public:

TStack(unsigned int n = 25)

{

size = n;

last = 0;

arr = new T[size];

}

TStack(const TStack& src)

{

size = src.size;

last = src.last;

arr = new T[size];

for (int i = 0; i < size; i++)

{

arr[i] = src.arr[i];

}

}

~TStack() { }

TStack<T>& operator= (const TStack<T>& src)

{

if (size != src.size)

{

if (size != 0)

{

delete[] arr;

}

arr = new T[size];

size = src.size;

}

last = src.last;

for (int i = 0; i < size; i++)

{

arr[i] = src.arr[i];

}

return \*this;

}

bool isEmpty()

{

return (last == 0) ? true : false;

}

bool isFull()

{

return (last == size - 1) ? true : false;

}

T peek()

{

return !isEmpty() ? arr[last - 1] : T(0);

}

T pop()

{

return !isEmpty() ? arr[--last] : T(0);

}

void push(T item)

{

if (!isFull())

{

arr[last++] = item;

}

}

};

1. «Form1.h»

using System;

using System.IO;

using System.Drawing;

using System.Windows.Forms;

using System.Threading.Tasks;

namespace GraphicEditor

{

public partial class Form1 : Form

{

bool Edit = false;

bool MousePressed = false;

int pointCounter = 0;

int size = 5;

Color color = Color.Black;

ShapeType shapeType = ShapeType.Line;

CustomPoint first;

Plex plex = new Plex();

public Form1()

{

InitializeComponent();

}

private void colorButton\_Click(object sender, EventArgs e)

{

if (colorDialog1.ShowDialog() == DialogResult.OK)

{

color = colorDialog1.Color;

}

}

private void sizeTextBox\_TextChanged(object sender, EventArgs e)

{

int.TryParse(label2.Text, out size);

}

private void Update(Graphics graphics)

{

graphics.Clear(Color.White);

plex.Draw(graphics);

}

private void AddPoint(int x, int y)

{

Graphics graphics = GraphicPanel.CreateGraphics();

CustomPoint point = new CustomPoint(pointCounter++.ToString(), x, y, size);

point.FillColor = color;

point.Draw(graphics);

//label1.Text = label1.Text + "\n" + "Name= " + pointCounter.ToString() + " X=" + x.ToString() + " Y=" + y.ToString() + " SIZE=" + size.ToString() + " COLOR=" + color.ToString();

}

private void AddLine(CustomPoint left,CustomPoint right)

{

Graphics graphics=GraphicPanel.CreateGraphics();

CustomLine line = new CustomLine(left,right);

line.FillColor = color;

line.Draw(graphics);

if (shapeType == ShapeType.Plex)

{

plex.Add(line);

}

}

private void trackBar1\_Scroll(object sender, EventArgs e)

{

size = trackBar1.Value;

label2.Text=size.ToString();

}

private void pOINTToolStripMenuItem\_Click(object sender, EventArgs e)

{

shapeType = ShapeType.Point;

label3.Text = "Type: " + shapeType.ToString();

}

private void lINEToolStripMenuItem\_Click(object sender, EventArgs e)

{

shapeType = ShapeType.Line;

label3.Text = "Type: " + shapeType.ToString();

}

private void pLEXToolStripMenuItem\_Click(object sender, EventArgs e) {

shapeType = ShapeType.Plex;

label3.Text = "Type: " + shapeType.ToString();

}

void EditPoint(CustomPoint c, int localX, int localY)

{

if (!MousePressed)

{

c.Shift(

Cursor.Position.X - c.X - localX,

Cursor.Position.Y - c.Y - localY);

Graphics graphics = GraphicPanel.CreateGraphics();

graphics.Clear(Color.White);

plex.Draw(graphics);

Task.Delay(10).ContinueWith(t => EditPoint(c, localX, localY)); ;

}

}

void EditPlex(int localX, int localY)

{

if (!MousePressed)

{

plex.Shift(

Cursor.Position.X - localX,

Cursor.Position.Y - localY);

Graphics graphics = GraphicPanel.CreateGraphics();

graphics.Clear(Color.White);

plex.Draw(graphics);

localX = Cursor.Position.X;

localY = Cursor.Position.Y;

Task.Delay(10).ContinueWith(t => EditPlex(localX, localY)); ;

}

}

private void GraphicPanel\_MouseUp(object sender, MouseEventArgs e)

{

if (e.Button == MouseButtons.Left)

{

MousePressed= true;

if (!Edit)

{

Graphics graphics = GraphicPanel.CreateGraphics();

CustomPoint second = new CustomPoint($"A{pointCounter++}", e.X, e.Y, size);

switch (shapeType)

{

case ShapeType.Point:

AddPoint(e.X, e.Y);

return;

case ShapeType.Line:

AddLine(first, second);

return;

case ShapeType.Plex:

AddLine(first, second);

Update(graphics);

return;

}

}

}

}

private void GraphicPanel\_MouseDown(object sender, MouseEventArgs e)

{

if(e.Button == MouseButtons.Left)

{

//first = new CustomPoint($"A{pointCounter++}", e.X, e.Y, size);

//first.FillColor = color;

MousePressed = false;

if (!Edit)

{

first = new CustomPoint($"A{pointCounter++}", e.X, e.Y, size);

first.FillColor = color;

}

else

{

CustomPoint c;

c = plex.FindPoint(new CustomPoint($"A", e.X, e.Y, size));

if (c != null)

{

int localX = Cursor.Position.X - c.X;

int localY = Cursor.Position.Y - c.Y;

EditPoint(c, localX, localY);

}

else

{

int localX = Cursor.Position.X;

int localY = Cursor.Position.Y;

EditPlex(localX, localY);

}

}

}

}

private void GraphicPanel\_MouseMove(object sender, MouseEventArgs e)

{

if (first != null)

{

// Переместить точку и элемент управления

//first.X = e.X;

//first.Y = e.Y;

// Обновить отображение элемента управления

Control control = this.Controls[this.Controls.Count - 1];

control.Location = new Point(first.X, first.Y); // Установить позицию

}

}

private void sAVEToolStripMenuItem\_Click(object sender, EventArgs e)

{

if (saveFileDialog1.ShowDialog() == DialogResult.OK)

{

string filePath = saveFileDialog1.FileName;

string textPlex = plex.SaveToString();

File.WriteAllText(filePath, textPlex);

}

}

private void lOADToolStripMenuItem\_Click(object sender, EventArgs e)

{

if (openFileDialog1.ShowDialog() == DialogResult.OK)

{

string filePath = openFileDialog1.FileName;

plex = new Plex(File.ReadAllText(filePath));

Graphics graphics = GraphicPanel.CreateGraphics();

Update(graphics);

first = null;

}

}

private void button1\_Click(object sender, EventArgs e)

{

plex = new Plex();

pointCounter = 0;

Graphics graphics = GraphicPanel.CreateGraphics();

graphics.Clear(Color.White);

}

private void button2\_Click(object sender, EventArgs e)

{

Edit = !(Edit);

if (Edit)

{

button2.Text = "Cancel";

}

else

{

button2.Text = "Redact";

}

}

}

}