МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Казанский национальный исследовательский технический университет

им. А.Н. Туполева – КАИ»

Институт компьютерных технологий и защиты информации

Отделение СПО ИКТЗИ (Колледж информационных технологий)

ЛАБОРАТОРНАЯ РАБОТА №9

по дисциплине

Основы алгоритмизации и программирования

Тема: «РАЗРАБОТКА КОМАНДНОЙ СТРОКИ ДЛЯ УПРАВЛЕНИЯ ХОДОМ ВЫПОЛНЕНИЯ ПРОГРАММЫ»

Работу выполнил

Студент гр.4232

Сафронов М.Л.

Принял

Преподаватель Хасанов А.А.

Казань 2022

**Цель работы**

Изучение алгоритмов вычисления функциональных выражений в обратной польской записи и особенностей их программной реализации

**Задание на лабораторную работу**

Модифицировать программу, реализованную на предыдущей лабораторной работе «Создание и использование библиотеки классов для графических примитивов».

Обновленная версия программы должна включать в себя следующие изменения:

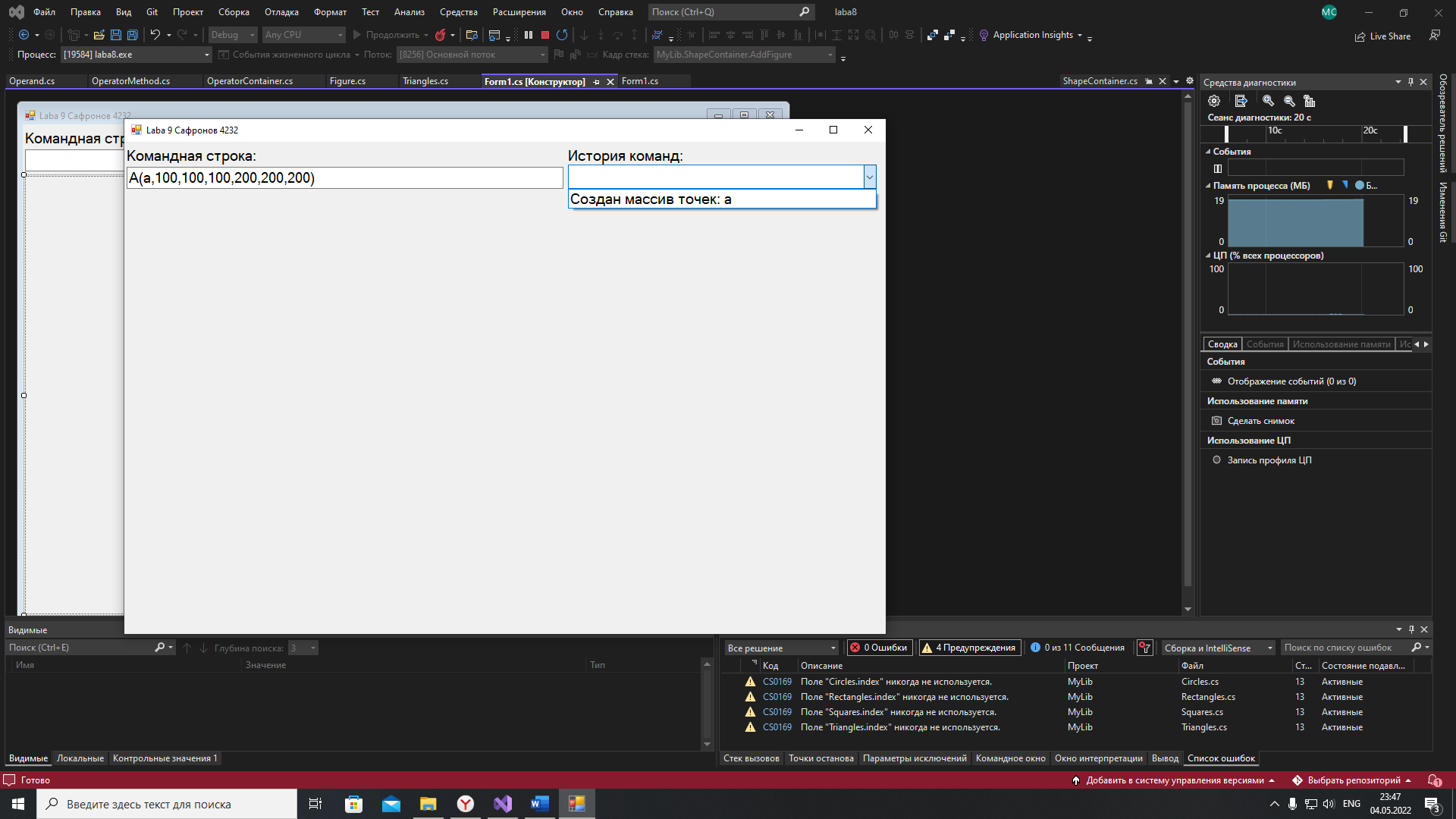
1. Удаление всех элементов управления из формы (кнопок, лейблов, полей для ввода и прочих), кроме поля рисунка PictureBox, где будет размещаться битовая карта;
2. Добавление командной строки (для ее реализации можно использовать элемент TextBox), где будут указываться команды, которые должна будет выполнять программа (прорисовка, перемещение и удаление фигур);
3. Добавить историю команд, где будут размещаться выполненные и неудачные команды.
4. Команды должны выполняться при нажатии кнопки ENTER на клавиатуре.

**Результат выполнения работы**

Описал в проекте класс Operand, в котором будет храниться значение операнда из входной строки. Описал класс OperatorMethod каторый в качестве свойст хранит делегаты на методы с определенным количеством параметров. Описал класс Operator который хранит знак оператора и переменную каждого делегата. Описал класс OperatorContainer который хранит объекты операторы.

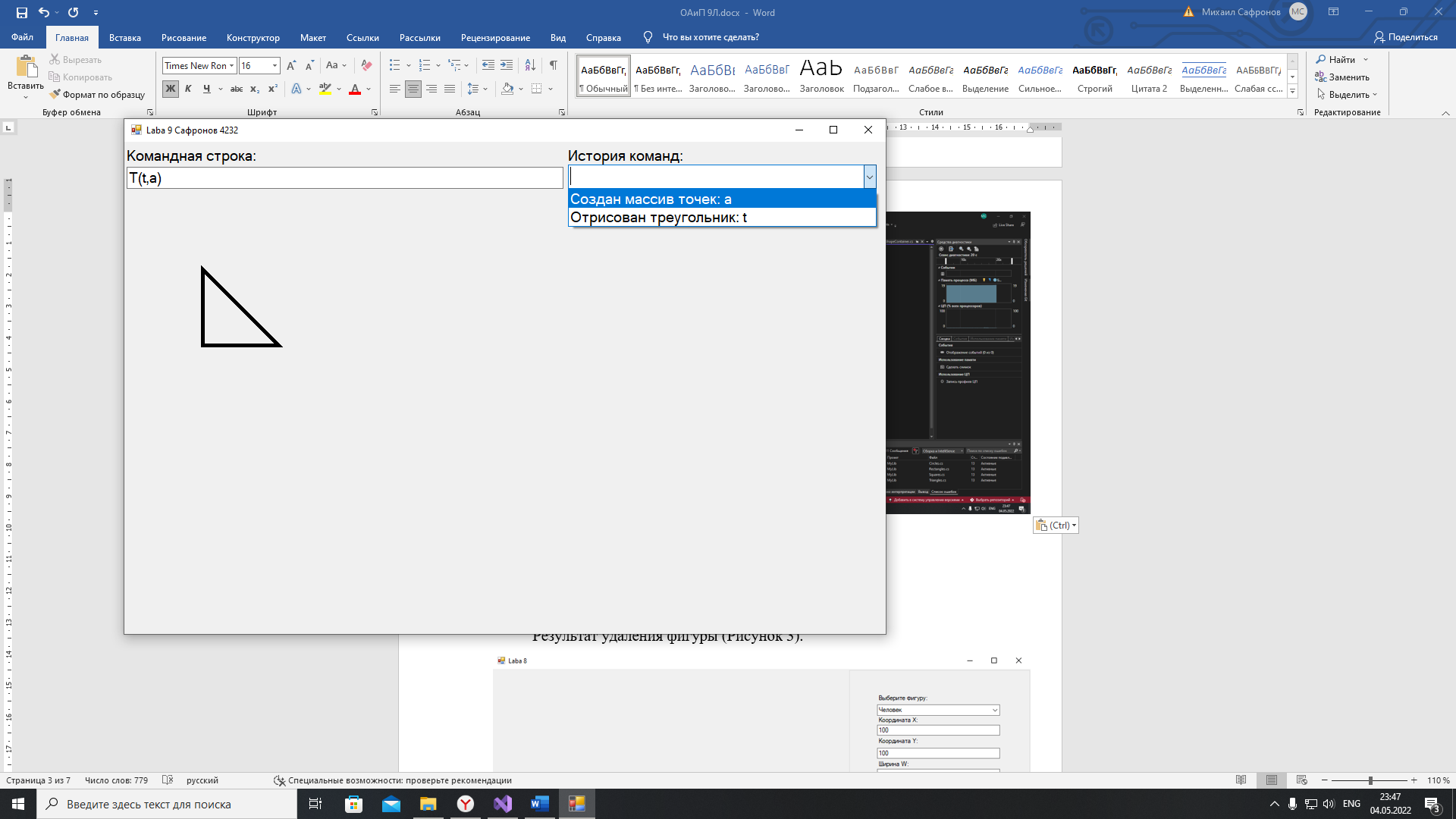
В Form1.cs определил два стека классов и словарь массивов точек. Добавил обработчик события по нажатию клавиши Enter, в котором с помощью модифицированного алгоритма обработки обратной польской записи состовляющие входной строки распределяются по стекам операторов и операндам и в дальнейшем в зависимости от введеной комманды происходит создание массива точек, отрисовка треугольника, перемещение треугольника или удаление треугольника.

Создание массива точек (Рисунок 1).



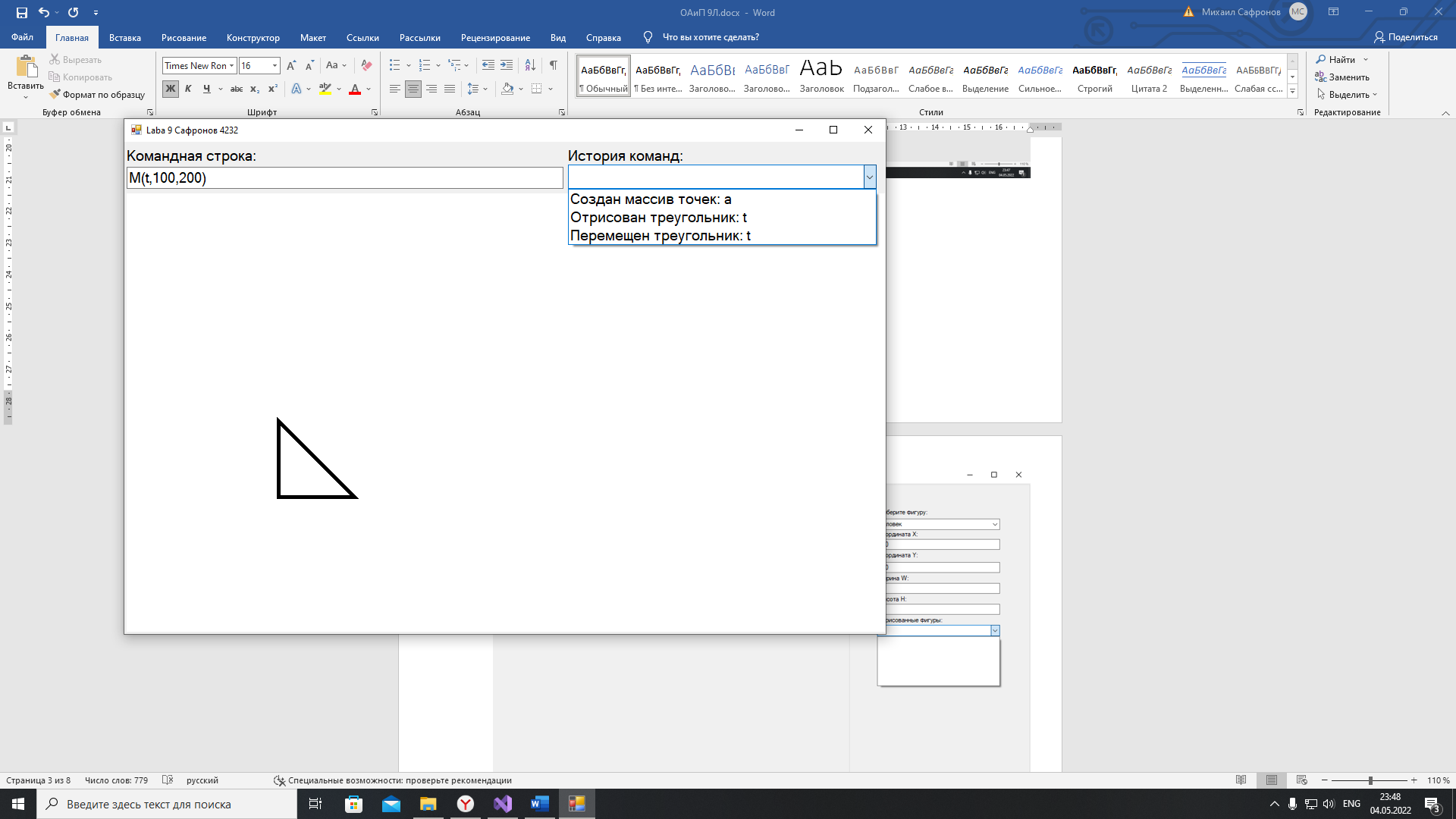
«Рисунок 1» - Создание массива точек.

Результат отрисовки треугольника используя только что созданный массив точек (Рисунок 2).



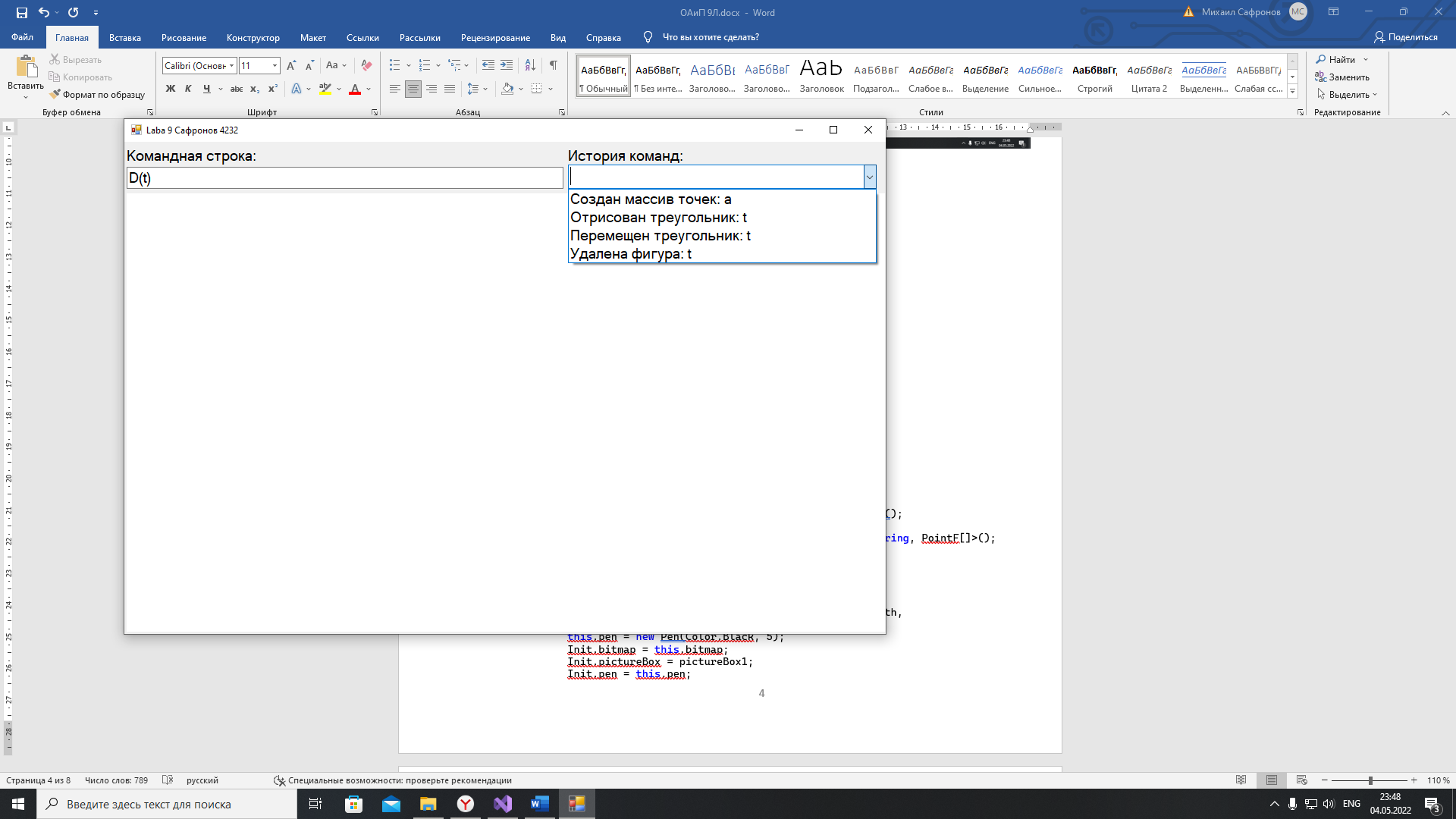
«Рисунок 2» - Отрисовка треугольника.

Результат перемещения фигуры (Рисунок 3).



«Рисунок 3» - Перемещение фигуры.

Результат удаления фигуры (Рисунок 4).



«Рисунок 4» - Удаление фигуры.

**Листинг программы**

**Form1.cs**

using System;

using System.Collections.Generic;

using System.Drawing;

using System.Windows.Forms;

using MyLib;

using laba8.laba9;

namespace laba8

{

public partial class Form1 : Form

{

Bitmap bitmap;

Pen pen;

ShapeContainer cont = new ShapeContainer();

private Stack<Operator> operators = new Stack<Operator>();

private Stack<Operand> operands = new Stack<Operand>();

Dictionary<string, PointF[]> points = new Dictionary<string, PointF[]>();

public Form1()

{

InitializeComponent();

this.bitmap = new Bitmap(pictureBox1.ClientSize.Width, pictureBox1.ClientSize.Height);

this.pen = new Pen(Color.Black, 5);

Init.bitmap = this.bitmap;

Init.pictureBox = pictureBox1;

Init.pen = this.pen;

}

private void tbInputString\_KeyDown(object sender, KeyEventArgs e)

{

if (e.KeyCode == Keys.Enter)

{

try

{

foreach (var inputChar in tbInputString.Text)

{

if (IsNotOperation(inputChar))

{

if (!char.IsDigit(inputChar))

{

operands.Push(new Operand(operands.Pop().value.ToString() + inputChar));

}

else

{

operands.Push(new Operand(operands.Pop().value.ToString() + inputChar));

}

}

else if (operands.Count == 0 || operands.Peek().value.ToString() != "")

{

operands.Push(new Operand(""));

}

switch (inputChar)

{

case '(':

{

operators.Push(OperatorContainer.FindOperator(inputChar));

break;

}

case ')':

{

do

{

if (operators.Peek().symbolOperator == '(')

{

operators.Pop();

break;

}

if (operators.Count == 0)

{

break;

}

} while (operators.Peek().symbolOperator != ')');

if (operators.Peek() != null)

{

operands.Pop();

SelectingPerformingOperation(operators.Peek());

operands = new Stack<Operand>();

operators = new Stack<Operator>();

}

else

{

MessageBox.Show("Введенной операции не существует");

cbHistory.Items.Add("Попытака ввода несуществубщей команды: '" + tbInputString.Text + "'");

}

break;

}

default:

{

if (inputChar == 'A' || inputChar == 'T' || inputChar == 'M' || inputChar == 'D')

{

if (operators.Count == 0)

{

operators.Push(OperatorContainer.FindOperator(inputChar));

}

break;

}

break;

}

}

}

}

catch (Exception ex)

{

MessageBox.Show(ex.ToString());

cbHistory.Items.Add("Попытака ввода команды: '" + tbInputString.Text + "'");

}

}

}

private bool IsNotOperation(char item)

{

if (!(item == 'A' || item == 'T' || item == 'M' || item == 'D' || item == ',' || item == '(' || item == ')'))

return true;

else

return false;

}

private void SelectingPerformingOperation(Operator op)

{

switch (op.symbolOperator)

{

case 'A':

{

//A(nameA, x1, y1, x2, y2, x3, y3)

int y2 = Convert.ToInt32(Convert.ToString(operands.Pop().value));

int x2 = Convert.ToInt32(Convert.ToString(operands.Pop().value));

int y1 = Convert.ToInt32(Convert.ToString(operands.Pop().value));

int x1 = Convert.ToInt32(Convert.ToString(operands.Pop().value));

int y0 = Convert.ToInt32(Convert.ToString(operands.Pop().value));

int x0 = Convert.ToInt32(Convert.ToString(operands.Pop().value));

string name = Convert.ToString(operands.Pop().value);

PointF[] pointF = new PointF[3];

pointF[0].X = x0; pointF[0].Y = y0;

pointF[1].X = x1; pointF[1].Y = y1;

pointF[2].X = x2; pointF[2].Y = y2;

cbHistory.Items.Add("Создан массив точек: " + name);

points.Add(name,pointF);

break;

}

case 'T':

{

//T(name, nameA)

string nameA = Convert.ToString(operands.Pop().value);

string nameT = Convert.ToString(operands.Pop().value);

PointF[] pointF = new PointF[3];

foreach (var point in points)

{

if(point.Key == nameA)

pointF = point.Value;

}

Triangles b = new Triangles(pointF, nameT);

op = new Operator(b.Draw, 'T');

op.operatorMethod();

ShapeContainer.AddFigure(b);

cbHistory.Items.Add("Отрисован треугольник: " + b.name);

break;

}

case 'M':

{

//M(name, dx, dy)

int dy = Convert.ToInt32(Convert.ToString(operands.Pop().value));

int dx = Convert.ToInt32(Convert.ToString(operands.Pop().value));

string name = Convert.ToString(operands.Pop().value);

for (int i = 0; i < ShapeContainer.figureList.Count; i++)

{

if (ShapeContainer.figureList[i].name == name)

{

if(ShapeContainer.figureList[i].MoveTo(dx, dy) == true)

{

cbHistory.Items.Add("Перемещен треугольник: " + ShapeContainer.figureList[i].name);

}

else

{

cbHistory.Items.Add("Неудалось переместить фигуру: " + ShapeContainer.figureList[i].name);

}

}

}

break;

}

case 'D':

{

//D(name)

string nameF = Convert.ToString(operands.Pop().value);

var forDelete = ShapeContainer.figureList.Find(f => f.name == nameF);

ShapeContainer.figureList.Remove(forDelete);

cbHistory.Items.Add("Удалена фигура: " + nameF);

Graphics g = Graphics.FromImage(Init.bitmap);

g.Clear(Color.White);

pictureBox1.Invalidate();

for (int i = 0; i < ShapeContainer.figureList.Count; i++)

{

ShapeContainer.figureList[i].Draw();

}

break;

}

}

}

}

}

**Operand.cs**

namespace laba8.laba9

{

public class Operand

{

public object value;

public Operand(object NewValue)

{

this.value = NewValue;

}

}

}

**Operator.cs**

namespace laba8.laba9

{

public class Operator : OperatorMethod

{

public char symbolOperator;

public EmptyOperatorMethod operatorMethod = null;

public BinaryOperatorMethod binaryOperator = null;

public TrinaryOperatorMethod trinaryOperator = null;

public SevenaryOperatorMethod sevenaryOperator = null;

public Operator(EmptyOperatorMethod operatorMethod, char symbolOperator)

{

this.operatorMethod = operatorMethod;

this.symbolOperator = symbolOperator;

}

public Operator(BinaryOperatorMethod binaryOperator, char symbolOperator)

{

this.binaryOperator = binaryOperator;

this.symbolOperator = symbolOperator;

}

public Operator(TrinaryOperatorMethod trinaryOperator, char symbolOperator)

{

this.trinaryOperator = trinaryOperator;

this.symbolOperator = symbolOperator;

}

public Operator(SevenaryOperatorMethod sevenaryOperator, char symbolOperator)

{

this.sevenaryOperator = sevenaryOperator;

this.symbolOperator = symbolOperator;

}

public Operator(char symbolOperator)

{

this.symbolOperator = symbolOperator;

}

}

}

**OperatorContainer.cs**

using System.Collections.Generic;

namespace laba8.laba9

{

public static class OperatorContainer

{

public static List<Operator> operators = new List<Operator>();

static OperatorContainer()

{

operators.Add(new Operator('A'));

operators.Add(new Operator('T'));

operators.Add(new Operator('M'));

operators.Add(new Operator('D'));

operators.Add(new Operator(','));

operators.Add(new Operator('('));

operators.Add(new Operator(')'));

}

public static Operator FindOperator(char s)

{

foreach (Operator op in operators)

{

if (op.symbolOperator == s)

{

return op;

}

}

return null;

}

}

}

**OperatorMethod.cs**

namespace laba8.laba9

{

public class OperatorMethod

{

public delegate void EmptyOperatorMethod();

public delegate void UnaryOperatorMethod(object operand);

public delegate void BinaryOperatorMethod(object operand1, object operand2);

public delegate void TrinaryOperatorMethod(object operand1, object operand2, object operand3);

public delegate void SevenaryOperatorMethod(object operand1, object operand2, object operand3, object operand4, object operand5, object operand6, object operand7);

}

}