

ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБЩЕОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ МОСКОВСКОЙ ОБЛАСТИ
«ОДИНЦОВСКИЙ «ДЕСЯТЫЙ ЛИЦЕЙ»

«Построитель графиков функций»

(проект)

(IT, программирование)

Выполнила:

Плотникова Мария, 9 «В» класс

Руководитель:

Деткова Людмила Анатольевна

учитель информатики

Московская область

2024 год

Содержание

Содержание	2
Паспорт проекта	4
Введение	5
Основная часть	7
1. Теория.....	7
2. Разработка интерпретатора формул.....	7
2.1. Лексический анализ текста	8
2.2. Синтаксический анализ текста	10
2.3. Построение вычисляемого выражения по заданному АСД	11
3. Построение графика функции.....	12
4. Разработка приложения.	12
5. Результаты.....	14
6. Заключение	14
Источники информации.....	17
Приложение А. Исходные тексты	18
A.1 App.xaml.cs – основной модуль приложения	18
A.2 App.xaml – визуальная часть модуля приложения	18
A.3 Модуль MainWindow. xaml.cs – главное окно	18
A.4 MainWindow.xaml – визуальная часть главного окна	26
A.5 HelpWindow.xaml.cs – окно справки	27
A.6 HelpWindow.xaml – визуальная часть окна справки	28
A.7 Lexeme.cs – определение лексем для логического анализатора	30
A.8 LexemeExtensions.cs – расширения для лексем.....	30
A.9 LexicalAnalyzerException.cs – ошибки логического анализатора	31

A.10 LexicalParser.cs – логический анализатор	31
A.11 AstNode.cs – узлы АСД.....	33
A.12 SyntacticalAnalyzerException.cs – ошибки синтаксического анализатора..	35
A.12 SyntacticalParser.cs – синтаксический анализатор	35
A.13 AnalyzerExceptionBase.cs – базовый класс для ошибок анализаторов	38
A.15 TextSpan.cs – текстовый блок.....	38
A.16 ExpressionBuilder.cs – построитель выражений	39

Паспорт проекта

Название	«Построитель графиков функций»
Автор	Плотникова Мария, ученица 9 «В» класса
Научный руководитель	Деткова Людмила Анатольевна, учитель информатики
Цели	<ol style="list-style-type: none"> 1. Приобрести навыки разработки настольных приложений. 2. Создать приложение для интерактивного построения графиков функций.
Задачи	<ol style="list-style-type: none"> 1. Изучить основные концепции создания интерпретатора языка описания математических функций. 2. Изучить основные концепции разработки двумерных графических приложений создания интерпретатора языка описания формул. 3. Установить и настроить программные средства разработки. 4. Создать редактор формул с проверкой синтаксиса вводимых данных. 5. Реализовать построение графиков функций по введенным пользователем формулам.
Даты	Идея от 26.09.2023. В период с 14.10.2023 по 07.03.2024 нашла нужную информацию и разработала программу.
Материально-техническое обеспечение	Необходим компьютер, используемое программное обеспечение бесплатное, дополнительных денежных затрат не требуется.

Введение

Проблема

Для углублённого изучения разных отраслей программирования: двумерной графики, логических и синтаксических анализаторов, разработки настольных приложений, – я решила разработать программу для интерактивного построения графиков функций.

Актуальность

Этот проект важен для меня по двум причинам. Первая – это получение новых навыков в программировании и в разработке приложений, что пригодится мне в будущей карьере.

Вторая причина – возможность предоставить любознательным школьникам инструмент для простой и удобной визуализации некоторых математических абстракций.

Цели проекта

1. Приобрести навыки разработки настольных приложений.
2. Создать приложение для интерактивного построения графиков функций.

Задачи проекта

На пути достижения целей проектов планируется решить следующие задачи:

1. Изучить основные концепции создания интерпретатора языка описания математических функций.
2. Изучить основные концепции разработки двумерных графических приложений создания интерпретатора языка описания формул.
3. Установить и настроить программные средства разработки.
4. Создать редактор формул с проверкой синтаксиса вводимых данных.
5. Реализовать построение графиков функций по введённым пользователем формулам.

Методы

Для работы над проектом я буду использовать интегрированную среду разработки (IDE) Visual Studio. Написание кода будет выполняться на языке программирования C#.

Для анализа и интерпретации функций я буду использовать метод построения абстрактного синтаксического дерева (АСД).

Для построения графиков я буду пользоваться методом дискретизации функций. В рамках настоящего проекта будут использоваться только формулы с функциями от одной переменной.

Основная часть

1. Теория

Для создания программы построения графиков функций необходимо решить две основные задачи:

1. Преобразовать формулу функции, введенную пользователем, в код на языке программирования, позволяющий вычислить значение функции для любого, заранее неизвестного значения её аргумента.
2. Преобразовать математическую функцию, определённую на бесконечном множестве действительных чисел, в растровое изображение.

2. Разработка интерпретатора формул

В моём проекте интерпретатор – это программа для преобразования текстовой записи формулы функции в другой программный код для её вычисления. Для создания интерпретатора я использую метод построения АСД.

АСД – это дерево, корнем которого является сама функция. Внутренние его вершины сопоставлены с конкретными операциями (сложение, вычитание, подфункция и т. д.), а листья – с соответствующими операндами. Таким образом, листья являются пустыми операторами и могут быть только переменными (аргументом функции) или константами.

Для построения АСД необходимо для начала определиться с допустимой грамматикой формул, которые я смогу обрабатывать. Для целей данного проекта я ограничусь формулами, содержащими элементы, перечисленные в таблице 1.

Таблица 1 – Перечень допустимых элементов формул

Элемент	Описание
число	Константа рационального числа
x	Аргумент функции
+	Операция сложения
-	Операция вычитания
*	Операция умножения
/	Операция деления
^	Операция возведения в степень

(Открывающая скобка
)	Закрывающая скобка
e	Константа числа e
pi	Константа числа π
sin	Функция синуса
cos	Функция косинуса
tg	Функция тангенса
ln	Функция натурального логарифма
lg	Функция десятичного логарифма

Общепринятый подход к построению интерпретаторов состоит в выполнении следующих преобразований:

1. Лексический анализ текста формулы.
2. Синтаксический анализ массива лексем и построение АСД.
3. Построение вычисляемого выражения по заданному АСД на целевом языке программирования.

Далее рассмотрим все этапы подробнее.

2.1. Лексический анализ текста

На этом этапе наша задача разобрать входной текст на лексемы – жёстко определённые и далее неделимые части исходного текста.

Для начала необходимо определить сами лексемы. Делаем это, основываясь на исходной таблице 1 с допустимыми элементами формул. Объединим все буквенные выражения как «идентификатор», остальные операции берём как есть. На выходе получается список, приведённый в таблице 2.

Таблица 2 – Список лексем

Элемент	Определение	Допустимая длина
Число	Последовательность цифр 0..9 с одной опциональной точкой в качестве десятичного разделителя.	≥ 1
Идентификатор	Начинается всегда с буквы A..Z , затем последовательность букв A..Z и цифр 0..9 . Целиком может обозначать имя переменной, или имя функции, или имя константы	≥ 1
+	Символ «Плюс»	1
*	Символ «Звёздочка»	1
/	Символ «Прямой слэш»	1
^	Символ «Крышка»	1
(Символ «Открывающая скобка»	1
)	Символ «Закрывающая скобка»	1

Также мы определяем, что пробелы для нас несущественны, и мы их будем просто пропускать в процессе разбора, а все прочие символы, не упомянутые в таблице 2 будем считать ошибочными.

В результате работы сопоставления символов входного текста с определениями лексем получается массив распознанных лексем, которые будут являться «сырьём» для следующего этапа – синтаксического анализа.

Например, формула $0.6 * x^2 - 2 * x - 1 + \sin(x)$ преобразуется в такой массив лексем:

```

число<0.6>
*
идентификатор<x>
^
число <2>
-
число <2>
*
идентификатор<x>
-
число <1>,
+
идентификатор<sin>
(
идентификатор<x>
)
```

2.2. Синтаксический анализ текста

На этом этапе разбора формулы мы пытаемся построить её абстрактное синтаксическое дерево. Для его построения нам необходимо определить правила, распределяющие входной массив лексем по узлам дерева. Набор этих правил называется грамматикой. Грамматика определяет общую структуру выражения и задаёт приоритет операций.

Для моего проекта я выбрала грамматику из правил, перечисленных в таблице 3. Первое правило будет корневым.

Таблица 3 – Список правил грамматики

Правило	Описание
выражение \rightarrow терм3 (("+" "-") терм3)*	Выражение, состоящее из одного или нескольких термов третьего уровня, объединённых знаками сложения и вычитания
терм3 \rightarrow терм2 (("*" "/") терм2)*	Терм третьего уровня, состоящий из одного или нескольких термов второго уровня, объединённых знаками умножения и деления
терм2 \rightarrow терм1 ("^") терм1 *	Терм второго уровня, состоящий из одного или нескольких термов первого уровня, объединённых операциями возведения в степень
терм1 \rightarrow [+ -] ("(" выражение ")" вызов идентификатор число)	Терм первого уровня, состоящий из: опционального унарного оператора плюс или минус и одного из следующих вариантов: 1. Подвыражение в круглых скобках. 2. Вызов функции. 3. Переменная, представленная идентификатором. 4. Числовой литерал.
вызов \rightarrow идентификатор "(" выражение ")"	Вызов функции, включающей её идентификатор и её аргумент в виде выражения, заключённого в круглые скобки

Синтаксический разбор начинается с первого правила (которое, становится корнем абстрактного синтаксического дерева), рекурсивно передвигаясь между правилами грамматики до полного исчерпания входного массива лексем. Если в

процессе разбора появляется недопустимая лексема, или определяется факт преждевременного исчерпания массива лексем, то синтаксический анализ завершается с ошибкой.

Формула из предыдущего раздела в итоге превращается в такое АСД:

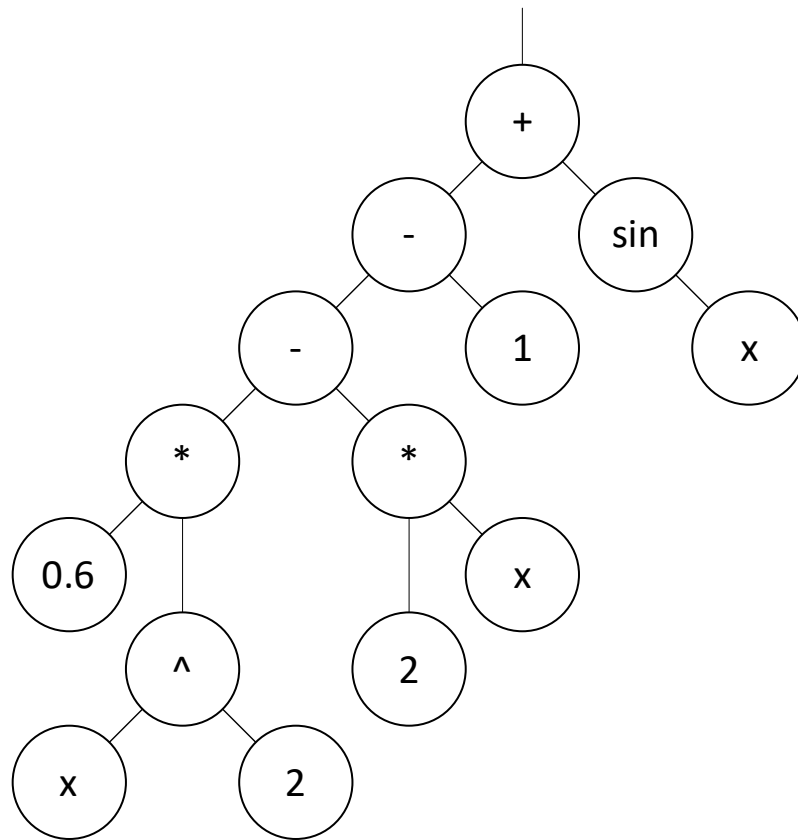


Рисунок 1 – Пример АСД

2.3. Построение вычисляемого выражения по заданному АСД

После получения абстрактного синтаксического дерева начинается этап построения вычисляемого выражения. Здесь алгоритм уже сильно зависит от целевого языка программирования. Так как я выбрала C#, то в нём есть мощная библиотека для построения динамических выражений, которая к тому же обладает похожим подходом и позволяет отобразить практически один-в-один АСД в .Net ExpressionTree.

После преобразования мы получаем динамически скомпилированный метод с сигнатурой `Func<double, double>`, представляющий собой функцию, принимающую один вещественный аргумент (нашу переменную X) и возвращающую вычисленное значение формулы для заданного значения аргумента.

3. Построение графика функции.

Для построения на экране графика ранее скомпилированного на предыдущем этапе выражения необходимо выполнить следующие действия:

1. Задать область определения функции, выбрав отрезок допустимых значений аргумента X .
2. Поскольку множество вещественных чисел бесконечно, а график строится на экране, состоящим из конечного числа пикселей, то необходимо провести дискретизацию области определений функции, заменив область определения конечным множеством значений аргумента X . Для построения графика на экране будет достаточно в качестве шага дискретизации взять один пиксель.
3. Задать область значений функций, выбрав отрезок отображаемых значений функции.
4. Двигаясь от минимального к максимальному значению аргумента X , с помощью полученного в разделе 2.3 метода, вычислить соответствующие значения координаты Y .
5. Каждую вычисленную точку (x_n, y_n) соединить отрезком с предыдущей (x_{n-1}, y_{n-1}) .

4. Разработка приложения.

Для создания графического приложения я выбрала библиотеку .NET 8 Windows Presentation Foundation (WPF), которая позволяет разрабатывать графические оконные приложения для Windows с насыщенной 3D- и 2D-графикой. Для рисования графиков функций мне как раз пригодятся возможности последней.

Я не планирую использовать никакие сторонние библиотеки для своего приложения, достаточно штатных средств фреймворка.

Так как платформа .NET позволяет использовать несколько языков программирования, то я выберу из них C# - как самый простой, мощный и распространённый.

В приложении я планирую сделать две формы: основное окно программы и окно справочной системы. Их макеты приведены на рисунке 2 и 3 соответственно.

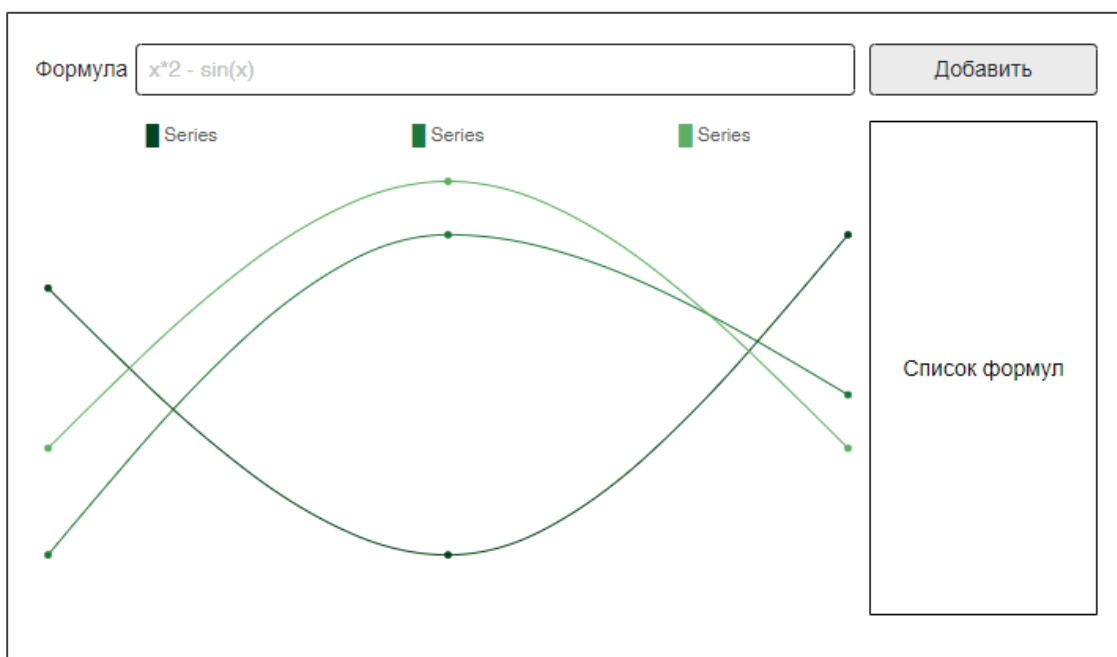


Рисунок 2 – Эскиз основного окна приложения

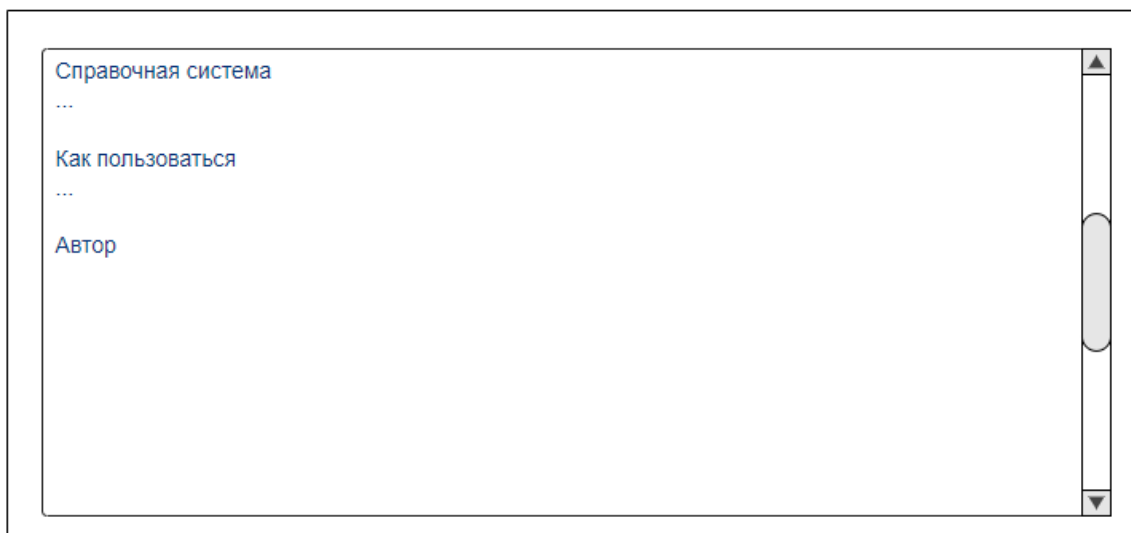


Рисунок 3 – Эскиз окна справочной системы

Исходные тексты всех модулей программы приведены в приложении А.

5. Результаты

По итогам работы над проектом были получены следующие результаты:

- изучены алгоритмы разбора формул и построения двумерных графиков.
- освоен фреймворк .NET 8 WPF для построения графических приложений.
- разработано приложение для построения графиков функций. Скриншоты приложения приведены на рисунках 4-6.

Реализованный в приложении функционал – это минимум, необходимый для отрисовки произвольных графиков. В дальнейшем приложение можно улучшать в разных направлениях, например:

- добавить поддержку поиска корней (решения уравнений);
- добавить возможность поиска экстремумов и точек перегиба;
- добавить возможность автоматического построения графиков производных.

6. Заключение

В рамках работы над проектом мне удалось достичь целей и задач проекта. Главный результат – есть работающее приложение, выполняющее заявленные функции.

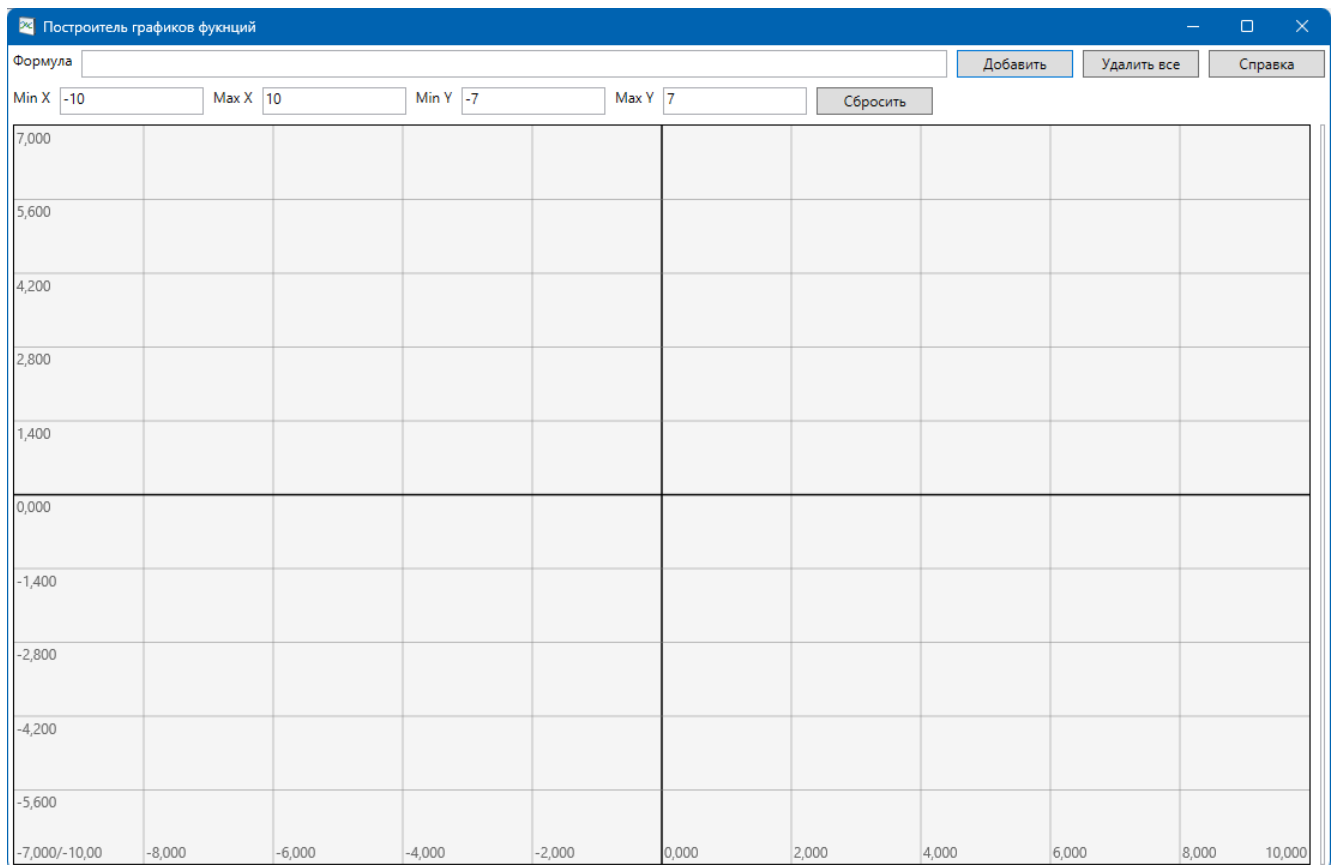


Рисунок 4 – Скриншот основного окна после запуска

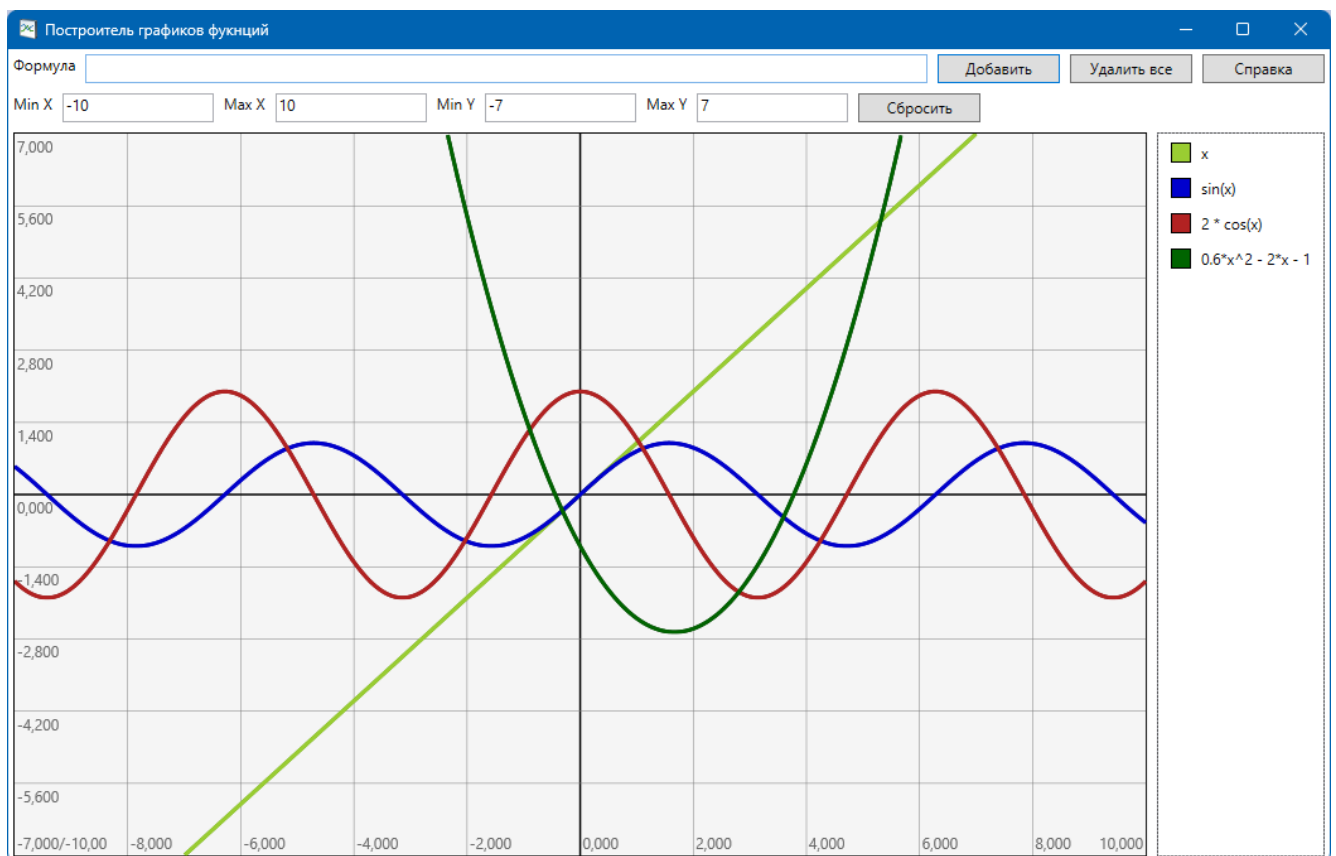


Рисунок 5 – Скриншот основного окна с несколькими построенными графиками

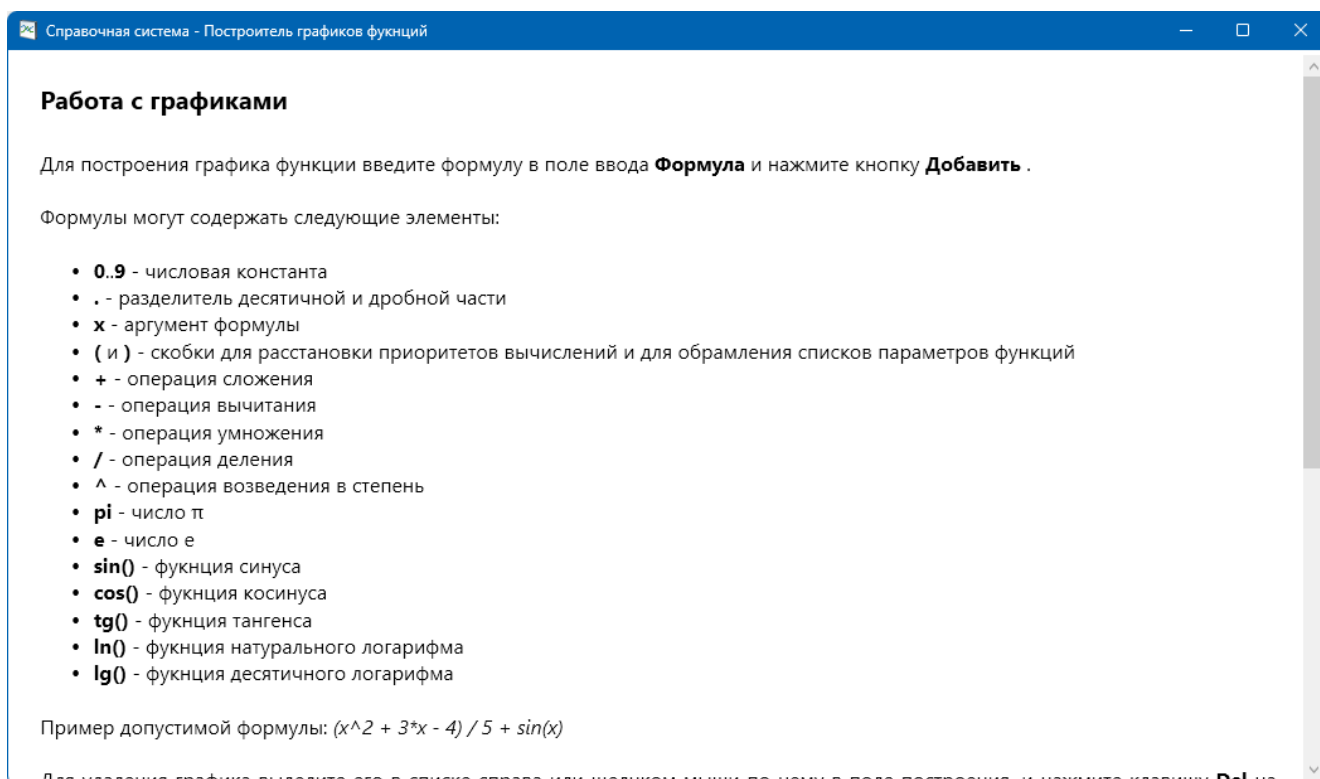


Рисунок 6 – Скриншот окна справки

Источники информации

1. Разбор формул <http://citforum.ru/programming/delphi/delphmath>.
2. Описание абстрактного синтаксического дерева
https://ru.wikipedia.org/wiki/абстрактное_синтаксическое_дерево.
3. Среда разработки Visual Studio <https://visualstudio.microsoft.com/ru/downloads>.
4. Фреймворк .NET 8 WPF <https://dotnet.microsoft.com/en-us/download/dotnet/8.0>.
5. Построение деревьев выражений LINQ <https://tyrrrz.me/blog/expression-trees>.
6. Работа с графикой в WPF
https://professorweb.ru/my/WPF/graphics_and_animation/level12/graph_animation_index.php.
7. Мак-Дональд, Мэтью. WPF: windows presentation foundation в .NET 4.5 с примерами на C# 5.0 для профессионалов. 4-е изд. : Пер. англ. – Москва : ООО «И.Д. Вильямс, 2013. – 1024 с.
8. Раттц-мл., Джозеф С. LINQ: язык интегрированных запросов в C# 2008 для профессионалов. : Пер. англ. – Москва : ООО «И.Д. Вильямс, 2008. – 560 с.

Приложение А. Исходные тексты

А.1 App.xaml.cs – основной модуль приложения

```
using System.Windows;
using System.Windows.Threading;

namespace Grapher
{
    public partial class App : Application
    {
        // Обработчик ошибок верхнего уровня
        private void Application_DispatcherUnhandledException(object sender, DispatcherUnhandledExceptionEventArgs e)
        {
            MessageBox.Show(e.Exception.Message, "Unhandled exception", MessageBoxButton.OK, MessageBoxImage.Error);
            e.Handled = true;
        }
    }
}
```

А.2 App.xaml – визуальная часть модуля приложения

```
<Application
    x:Class="Grapher.App"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="clr-namespace:Grapher"
    StartupUri="MainWindow.xaml"
    DispatcherUnhandledException="Application_DispatcherUnhandledException"
    ShutdownMode="OnMainWindowClose"
>
    <Application.Resources>

    </Application.Resources>
</Application>;
```

А.3 Модуль MainWindow.xaml.cs – главное окно

```
using Grapher.Analyzer;
using Grapher.Analyzer.Lexical;
using Grapher.Analyzer.Syntactical;
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.ComponentModel;
using System.Linq;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Animation;
using System.Windows.Media.Effects;
using System.Windows.Shapes;

namespace Grapher;

internal record class Graphic(string Formula, Func<double, double> Function, Color Color);

internal record class GraphicBuilderInfo
{
    public GraphicBuilderInfo(Graphic graphic)
    {
        Graphic = graphic;
        Geometry = new PathGeometry();
    }
    public Graphic Graphic { get; }
}
```

```

    public PathGeometry Geometry { get; }
    public PolyLineSegment? LastSegment { get; set; }
}

/// <summary>
/// Interaction logic for MainWindow.xaml
/// </summary>
public partial class MainWindow : Window, INotifyPropertyChanged
{
    private const double _defaultMinX = -10;
    private const double _defaultMinY = -7;
    private const double _defaultMaxX = +10;
    private const double _defaultMaxY = +7;

    private static readonly Color[] _colors = [Colors.Black,
        Colors.YellowGreen,
        Colors.MediumBlue,
        Colors.Firebrick,
        Colors.DarkGreen,
        Colors.Gold,
        Colors.Violet,
        Colors.SandyBrown,
        Colors.DarkCyan,
        Colors.DarkOrange,
        Colors.MediumOrchid,
        Colors.Coral,
        Colors.DodgerBlue,
        Colors.SaddleBrown
    ];

    private bool _initialized = false;
    private uint _canvasLockCount = 0U;
    private readonly ObservableCollection<Graphic> _graphics = [];
    private readonly Dictionary<Graphic, Path> _graphicPaths = [];
    private readonly Dictionary<Path, Storyboard> _pathAnimations = [];

    private double _minX;
    private double _minY;
    private double _maxX;
    private double _maxY;
    private uint _xAddMarkCount = 9;
    private uint _yAddMarkCount = 9;
    private readonly Brush _textMarkBrush;
    private readonly Brush _zeroAxisBrush;
    private readonly Brush _auxAxisBrush;

    private Point _panInitialPosition;

    public event PropertyChangedEventHandler? PropertyChanged;

    public double MinX { get => _minX; set { _minX = value; PropertyChanged?.Invoke(this, new
    PropertyChangedEventArgs(nameof(MinX))); RebuildCanvas(); } }

    public double MaxX { get => _maxX; set { _maxX = value; PropertyChanged?.Invoke(this, new
    PropertyChangedEventArgs(nameof(MaxX))); RebuildCanvas(); } }

    public double MinY { get => _minY; set { _minY = value; PropertyChanged?.Invoke(this, new
    PropertyChangedEventArgs(nameof(MinY))); RebuildCanvas(); } }

    public double MaxY { get => _maxY; set { _maxY = value; PropertyChanged?.Invoke(this, new
    PropertyChangedEventArgs(nameof(MaxY))); RebuildCanvas(); } }

    public MainWindow()
    {
        InitializeComponent();
        MinX = _defaultMinX;
        MaxX = _defaultMaxX;
        MinY = _defaultMinY;
        MaxY = _defaultMaxY;
        _legend.ItemsSource = _graphics;
        _textMarkBrush = new SolidColorBrush(Colors.DimGray);
    }
}

```

```

_zeroAxisBrush = new SolidColorBrush(Colors.Black);
_auxAxisBrush = new SolidColorBrush(Colors.Gray);

if (Keyboard.IsKeyDown(Key.LeftShift) || Keyboard.IsKeyDown(Key.RightShift))
{
    AddGraphic("x");
    AddGraphic("sin(x)");
    AddGraphic("2 * cos(x)");
    AddGraphic("0.6*x^2 - 2*x - 1");
    AddGraphic("tg(x)");
}
_initialized = true;
}

private void LockCanvas()
{
    _canvasLockCount++;
}

private void UnlockCanvas()
{
    if (_canvasLockCount > 0)
    {
        _canvasLockCount--;
        if (_canvasLockCount == 0)
            RebuildCanvas();
    }
    else
        throw new InvalidOperationException("Неверное значение счётчика блокировок холста.");
}

private void AddGraphic(string formula)
{
    // логический разбор
    var lexicalParser = new LexicalParser();
    var tokens = lexicalParser.Parse(formula);

    // синтаксический разбор
    var syntacticalParser = new SyntacticalParser();
    var ast = syntacticalParser.Parse(tokens);

    // построение выражения
    var expressionBuilder = new ExpressionBuilder();
    var function = expressionBuilder.Build(ast);

    // добавление графика
    var graphic = new Graphic(formula, function, _colors[_graphics.Count % _colors.Length]);
    _graphics.Add(graphic);
    RebuildCanvas();

    if (_initialized)
    {
        _legend.SelectedItem = graphic;
        AnimateGraphic(graphic);
    }
}

private void AddPoint(GraphicBuilderInfo builder, double i, double xScale, double yScale)
{
    var x = _minX + i / xScale;
    var y = builder.Graphic.Function(x);
    if (double.IsFinite(y))
    {
        var j = _canvas.ActualHeight - (y - _minY) * yScale;
        if (0 <= j && j < _canvas.ActualHeight)
        {
            if (builder.LastSegment is null)
            {
                builder.LastSegment = new PolyLineSegment();
                var figure = new PathFigure() { StartPoint = new Point(i, j), IsClosed = false, IsFilled = false };
                figure.Segments.Add(builder.LastSegment);
                builder.Geometry.Figures.Add(figure);
            }
        }
    }
}

```

```

        else
            builder.LastSegment.Points.Add(new Point(i, j));
        }
        else
            builder.LastSegment = null;
    }
    else
        builder.LastSegment = null;
}

public void AddTextMark(string text, double x, double y, bool moveToRight, bool moveToUp)
{
    var mark = new TextBlock() { Text = text, Foreground = _textMarkBrush };
    _canvas.Children.Add(mark);
    if (moveToRight || moveToUp)
        mark.Measure(new Size(double.PositiveInfinity, double.PositiveInfinity));

    if (moveToRight)
        Canvas.SetLeft(mark, x - mark.DesiredSize.Width);
    else
        Canvas.SetLeft(mark, x);

    if (moveToUp)
        Canvas.SetTop(mark, y - mark.DesiredSize.Height);
    else
        Canvas.SetTop(mark, y);
}

private void RebuildCanvas()
{
    if (_canvasLockCount > 0U || !_initialized)
        return;

    _canvas.Children.Clear();
    _graphicPaths.Clear();
    _pathAnimations.Clear();

    var xStepScr = 0.5;
    var xScale = _canvas.ActualWidth / (_maxX - _minX);
    var yScale = _canvas.ActualHeight / (_maxY - _minY);
    var graphicBuilder = _graphics.ToDictionary(k => k, k => new GraphicBuilderInfo(k));
    for (var i = 0.0; i < _canvas.ActualWidth; i += xStepScr)
    {
        foreach (var gb in graphicBuilder)
            // по координате X вычисляем координату Y функции и добавляем вычисленную точку
            AddPoint(gb.Value, i, xScale, yScale);
    }
    var zIndex = 1;
    foreach (var gb in graphicBuilder)
    {
        // строим линию на экране по точкам, сохранённым в предыдущем шаге
        var path = new Path() { Data = gb.Value.Geometry, ClipToBounds = true, Stroke = new SolidColorBrush(gb.Key.Color),
        StrokeThickness = 3,
        DataContext = gb.Key, Cursor = Cursors.Hand };

        // обработка щелчка ЛКМ
        path.MouseLeftButtonDown += (o, e) => { _legend.SelectedItem = path.DataContext;
        AnimateGraphic((Graphic)path.DataContext); e.Handled = true; };

        // эффект для анимации выделения
        path.Effect = new DropShadowEffect() { BlurRadius = 0, ShadowDepth = 0, Color = gb.Key.Color };

        // выводим линию на экран
        _canvas.Children.Add(path);
        _graphicPaths.Add(gb.Key, path);
        Panel.SetZIndex(path, zIndex++);
    }

    // построение координатной сетки
    BuildCoordinateGrid();
}

private void AddXAxis(double y, double yScr, Brush stroke, double thickness)
{

```

```

var axis = new Line() { X1 = 0, Y1 = yScr, X2 = _canvas.ActualWidth, Y2 = yScr, Stroke = stroke, StrokeThickness = thickness };
_canvas.Children.Add(axis);
AddTextMark(y.ToString("f3"), 2, yScr + 2, false, false);
}

private void AddYAxis(double x, double xScr, Brush stroke, double thickness)
{
    var axis = new Line() { X1 = xScr, Y1 = 0, X2 = xScr, Y2 = _canvas.ActualHeight, Stroke = stroke, StrokeThickness = thickness };
    _canvas.Children.Add(axis);
    AddTextMark(x.ToString("f3"), xScr + 2, _canvas.ActualHeight - 2, false, true);
}

public void BuildCoordinateGrid()
{
    // комбинированная метка в нижнем левом углу
    AddTextMark(_minY.ToString("f3") + "/" + MinX.ToString("f"), 2, _canvas.ActualHeight - 2, false, true);

    // метки по оси X и вертикальные линии
    AddTextMark(MaxX.ToString("f3"), _canvas.ActualWidth - 2, _canvas.ActualHeight - 2, true, true);

    var xMarkStep = (MaxX - MinX) / (_xAddMarkCount + 1);
    var xMarkScrStep = _canvas.ActualWidth / (_xAddMarkCount + 1);
    double xStart;
    double xStartScr;
    if (MinX < 0 && 0 < MaxX)
    {
        xStart = 0;
        xStartScr = _canvas.ActualWidth * (-MinX / (MaxX - MinX));
        AddYAxis(xStart, xStartScr, _zeroAxisBrush, 1.5);
    }
    else
    {
        xStart = MinX;
        xStartScr = 0;
    }

    if (_xAddMarkCount > 1)
    {
        for (var i = -_xAddMarkCount; i < _xAddMarkCount; i++)
        {
            if (i == -1 && MinX < 0 && 0 < MaxX)
                continue;
            var x = xStart + (i + 1) * xMarkStep;
            var xScr = xStartScr + (i + 1) * xMarkScrStep;
            if (40 < xScr && xScr < _canvas.ActualWidth - 40)
                AddYAxis(x, xScr, _auxAxisBrush, 0.5);
        }
    }

    // метки по оси Y и горизонтальные линии
    AddTextMark(MaxY.ToString("f3"), 2, 2, false, false);

    var _yMarkStep = (MaxY - MinY) / (_yAddMarkCount + 1);
    var _yMarkScrStep = _canvas.ActualHeight / (_yAddMarkCount + 1);
    double yStart;
    double yStartScr;
    if (MinY < 0 && 0 < MaxY)
    {
        yStart = 0;
        yStartScr = _canvas.ActualHeight * (MaxY / (MaxY - MinY));
        AddXAxis(yStart, yStartScr, _zeroAxisBrush, 1.5);
    }
    else
    {
        yStart = MinY;
        yStartScr = _canvas.ActualHeight;
    }

    if (_yAddMarkCount > 1)
    {
        for (var i = -_yAddMarkCount; i < _yAddMarkCount; i++)
        {
            if (i == -1 && MinY < 0 && 0 < MaxY)
                continue;

```

```

        var y = yStart - (i + 1) * _yMarkStep;
        var yScr = yStartScr + (i + 1) * _yMarkScrStep;
        if (40 < yScr && yScr < _canvas.ActualHeight - 40)
            AddXAxis(y, yScr, _auxAxisBrush, 0.5);
    }
}
/*
if (_yAddMarkCount > 0)
{
    var _yMarkScrDistance = _canvas.ActualHeight / (_yAddMarkCount + 1);
    for (var j = 0U; j < _yAddMarkCount; j++)
    {
        var yPoint = _maxY - (j + 1) * (_maxY - _minY) / (_yAddMarkCount + 1);
        var yPointScr = (j + 1) * _yMarkScrDistance;
        var axis = new Line() { X1 = 0, Y1 = yPointScr, X2 = _canvas.ActualWidth, Y2 = yPointScr, Stroke = _auxAxisBrush,
StrokeThickness = 0.5 };
        _canvas.Children.Add(axis);
        AddTextMark(yPoint.ToString("f"), 2, yPointScr + 2, false, false);
    }
}
if (MinY <= 0 && 0 <= MaxY)
{
    // создаём линию для оси X
    var yPointSrc = _canvas.ActualHeight * (MaxY / (MaxY - MinY));
    var zeroXAxis = new Line() { X1 = 0, Y1 = yPointSrc, X2 = _canvas.ActualWidth, Y2 = yPointSrc, Stroke = _zeroAxisBrush,
StrokeThickness = 1.5 };
    _canvas.Children.Add(zeroXAxis);
}
*/
}

private void _clearButton_Click(object sender, RoutedEventArgs e)
{
    _graphics.Clear();
    RebuildCanvas();
}

private void _addButton_Click(object sender, RoutedEventArgs e)
{
    AddGraphic(_editor.Text);
}

private void _canvas_SizeChanged(object sender, SizeChangedEventArgs e)
{
    RebuildCanvas();
}

private void _canvas_MouseLeftButtonDown(object sender, MouseButtonEventArgs e)
{
    if (Mouse.Captured is null)
    {
        e.Handled = true;
        _panInitialPosition = e.GetPosition(_canvas);
        Mouse.Capture(_canvas);
        Mouse.OverrideCursor = Cursors.ScrollAll;
    }
}

private void _canvas_MouseMove(object sender, MouseEventArgs e)
{
    if (Mouse.Captured == _canvas)
    {
        var newPosition = e.GetPosition(_canvas);
        DoPanCanvas(newPosition);
    }
}

private void _canvas_MouseLeftButtonUp(object sender, MouseButtonEventArgs e)
{
    if (Mouse.Captured == _canvas)
    {
        e.Handled = true;
        ReleaseCanvasCapture(e.GetPosition(_canvas));
    }
}

```

```

private void _canvas_LostMouseCapture(object sender, MouseEventArgs e)
{
    Mouse.OverrideCursor = null;
}

private void DoPanCanvas(Point currentPosition)
{
    // обработка сдвига координатной плоскости мышью
    var screenDeltaX = currentPosition.X - _panInitialPosition.X;
    var screenDeltaY = currentPosition.Y - _panInitialPosition.Y;
    if (screenDeltaX != 0 || screenDeltaY != 0)
    {
        var deltaX = - screenDeltaX / _canvas.ActualWidth * (_maxX - _minX);
        var deltaY = screenDeltaY / _canvas.ActualHeight * (_maxY - _minY);
        LockCanvas();
        try
        {
            {
                MinX += deltaX;
                MaxX += deltaX;
                MinY += deltaY;
                MaxY += deltaY;
                _panInitialPosition = currentPosition;
            }
        }
        finally
        {
            {
                UnlockCanvas();
            }
        }
    }
}

private void ReleaseCanvasCapture(Point? lastPoint)
{
    {
        if (lastPoint is not null)
            DoPanCanvas(lastPoint.Value);
        Mouse.Capture(null);
        Mouse.OverrideCursor = null;
    }
}

private void Window_KeyDown(object sender, KeyEventArgs e)
{
    {
        if (e.Key == Key.Escape)
        {
            {
                if (Mouse.Captured == _canvas)
                {
                    // отмена захвата мыши по нажатию ESC
                    e.Handled = true;
                    ReleaseCanvasCapture(null);
                }
            }
        }
        else if (e.Key == Key.Delete)
        {
            if (_legend.SelectedItem is Graphic g)
            {
                {
                    _graphics.Remove(g);
                    RebuildCanvas();
                }
            }
        }
    }
}

private void _resetCoordinatesButton_Click(object sender, RoutedEventArgs e)
{
    {
        LockCanvas();
        try
        {
            {
                // установка начальных значений координат
                MinX = _defaultMinX;
                MaxX = _defaultMaxX;
                MinY = _defaultMinY;
                MaxY = _defaultMaxY;
                e.Handled = true;
            }
        }
        finally
        {
            {
                UnlockCanvas();
            }
        }
    }
}

```

```

    }
}

private void AnimateGraphic(Graphic graphic)
{
    // Анимация подсвечивания графика при выделении
    var path = _graphicPaths[graphic];
    if (!_pathAnimations.TryGetValue(path, out var sb))
    {
        var storyboard = new Storyboard();
        storyboard.FillBehavior = FillBehavior.Stop;
        var glowAnimation = new DoubleAnimation();
        glowAnimation.By = 30;
        glowAnimation.AutoReverse = true;
        glowAnimation.Duration = TimeSpan.FromSeconds(0.5);
        storyboard.SetTarget(glowAnimation, path);
        storyboard.SetTargetProperty(glowAnimation, new PropertyPath("Effect.BlurRadius"));
        storyboard.Children.Add(glowAnimation);
        //storyboard.Completed += (o, e) => path.Effect = null;
        _pathAnimations[path] = storyboard;
        storyboard.Begin();
    }
    else
    {
        if (sb.GetCurrentState() == ClockState.Stopped)
            sb.Begin();
    }
}

private void _legend_SelectionChanged(object sender, SelectionChangedEventArgs e)
{
    if (!_initialized)
        return;

    foreach (var g in e.AddedItems.OfType<Graphic>())
    {
        AnimateGraphic(g);
        var path = _graphicPaths[g];
        // отображаем выделенную фигуру поверх остальных
        var zCoords = _canvas.Children.OfType<Path>()
            .Where(x => x != path)
            .Select(x => Panel.GetZIndex(x)).ToArray();
        var z = zCoords.Length > 0 ? zCoords.Max() : 0;
        Panel.SetZIndex(path, z + 1);
    }
}

private void _canvas_PreviewMouseWheel(object sender, MouseWheelEventArgs e)
{
    if (e.Delta == 0)
        return;

    LockCanvas();
    try
    {
        // при нажатом Ctrl - масштабируем
        if (Keyboard.IsKeyDown(Key.LeftCtrl) || Keyboard.IsKeyDown(Key.RightCtrl))
        {
            var scaleFactor = 1 - 0.2 * Math.Sign(e.Delta);
            MinX *= scaleFactor;
            MinY *= scaleFactor;
            MaxX *= scaleFactor;
            MaxY *= scaleFactor;
        }
        // при нажатом Shift - прокручиваем влево/вправо
        else if (Keyboard.IsKeyDown(Key.LeftShift) || Keyboard.IsKeyDown(Key.RightShift))
        {
            var delta = (MaxX - MinX) * 0.05 * Math.Sign(e.Delta);
            MinX -= delta;
            MaxX -= delta;
        }
        // иначе прокручиваем вверх/вниз
    }
    else
    {

```

```

        var delta = (MaxY - MinY) * 0.05 * Math.Sign(e.Delta);
        MinY += delta;
        MaxY += delta;
    }
}
finally
{
    UnlockCanvas();
}
e.Handled = true;
}

private void _helpButton_Click(object sender, RoutedEventArgs e)
{
    HelpWindow.ShowWindow();
    e.Handled = true;
}
}

```

A.4 MainWindow.xaml – визуальная часть главного окна

```

<Window
    x:Class="Grapher.MainWindow"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    xmlns:local="clr-namespace:Grapher"
    mc:Ignorable="d"
    Title="Построитель графиков функций" Height="768" Width="1024" WindowStartupLocation="CenterScreen"
    WindowState="Maximized"
    PreviewKeyDown="Window_KeyDown"
>
    <Grid>
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="*/"/>
        </Grid.RowDefinitions>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="*/"/>
            <ColumnDefinition Width="Auto"/>
        </Grid.ColumnDefinitions>
        <Grid Grid.Row="0" Grid.ColumnSpan="2">
            <Grid.ColumnDefinitions>
                <ColumnDefinition Width="Auto" />
                <ColumnDefinition Width="*/" />
                <ColumnDefinition Width="Auto" />
                <ColumnDefinition Width="Auto" />
                <ColumnDefinition Width="Auto" />
            </Grid.ColumnDefinitions>
            <TextBlock Grid.Column="0" Margin="4">Формула</TextBlock>
            <TextBox x:Name="_editor" Grid.Column="1" Margin="4"></TextBox>
            <Button x:Name="_addButton" Grid.Column="2" Height="23" Width="97" Margin="4" Click="_addButton_Click"
                IsDefault="True" ToolTip="Добавить новый график функции с указанной формулой" >Добавить</Button>
            <Button x:Name="_clearButton" Grid.Column="3" Height="23" Width="97" Margin="4" Click="_clearButton_Click"
                ToolTip="Удалить все построенные графики функций">Удалить все</Button>
            <Button x:Name="_helpButton" Grid.Column="4" Height="23" Width="97" Margin="4" Click="_helpButton_Click"
                ToolTip="Открыть справочную систему">Справка</Button>
        </Grid>
        <StackPanel Grid.Row="1" Grid.ColumnSpan="2" Orientation="Horizontal">
            <TextBlock Margin="4">Min X</TextBlock>
            <TextBox Margin="4" MinWidth="120" Text="{Binding Mode=TwoWay, UpdateSourceTrigger=PropertyChanged, Delay=200,
                Path=MinX, RelativeSource={RelativeSource Mode=FindAncestor, AncestorType=Window}}">
                ToolTip="Минимальное значение координаты X"/>
            <TextBlock Margin="4">Max X</TextBlock>
            <TextBox Margin="4" MinWidth="120" Text="{Binding Mode=TwoWay, UpdateSourceTrigger=PropertyChanged, Delay=200,
                Path=MaxX, RelativeSource={RelativeSource Mode=FindAncestor, AncestorType=Window}}">
                ToolTip="Максимальное значение координаты X"/>
            <TextBlock Margin="4">Min Y</TextBlock>
            <TextBox Margin="4" MinWidth="120" Text="{Binding Mode=TwoWay, UpdateSourceTrigger=PropertyChanged, Delay=200,
                Path=MinY, RelativeSource={RelativeSource Mode=FindAncestor, AncestorType=Window}}">
                ToolTip="Минимальное значение координаты Y"/>
            <TextBlock Margin="4">Max Y</TextBlock>
            <TextBox Margin="4" MinWidth="120" Text="{Binding Mode=TwoWay, UpdateSourceTrigger=PropertyChanged, Delay=200,
                Path=MaxY, RelativeSource={RelativeSource Mode=FindAncestor, AncestorType=Window}}">
                ToolTip="Максимальное значение координаты Y"/>
        </StackPanel>
    </Grid>

```

```

        ToolTip="Максимальное значение координаты X" />
    <TextBlock Margin="4">Max Y</TextBlock>
    <TextBox Margin="4" MinWidth="120" Text="{Binding Mode=TwoWay, UpdateSourceTrigger=PropertyChanged, Delay=200,
Path=MaxY, RelativeSource={RelativeSource Mode=FindAncestor, AncestorType=Window}}"
        ToolTip="Максимальное значение координаты Y" />
    <Button x:Name="_resetCoordinatesButton" Height="23" Width="97" Margin="4" Click="_resetCoordinatesButton_Click"
ToolTip="Сбросить значения координат в исходное состояние">Сбросить</Button>
</StackPanel>
<Border Grid.Row="2" Grid.Column="0" BorderThickness="1" BorderBrush="Black" Margin="4">
    <Grid>
        <Grid.RowDefinitions>
            <RowDefinition Height="*" />
            <RowDefinition Height="Auto" />
        </Grid.RowDefinitions>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="Auto" />
            <ColumnDefinition Width="*" />
        </Grid.ColumnDefinitions>
        <Canvas x:Name="_canvas" Grid.Row="0" Grid.Column="1" Background="WhiteSmoke" Cursor="Cross"
            SizeChanged="_canvas_SizeChanged"
            MouseLeftButtonDown="_canvas_MouseLeftButtonDown" MouseMove="_canvas_MouseMove"
            MouseLeftButtonUp="_canvas_MouseLeftButtonUp" LostMouseCapture="_canvas_LostMouseCapture"
            PreviewMouseWheel="_canvas_PreviewMouseWheel" />
    </Grid>
</Border>
<ListBox Grid.Row="2" Grid.Column="1" x:Name="_legend" Margin="4" SelectionChanged="_legend_SelectionChanged">
    <ListBox.ItemTemplate>
        <DataTemplate>
            <DockPanel LastChildFill="True">
                <Rectangle DockPanel.Dock="Left" Margin="4" MinHeight="16" MinWidth="16" Stroke="Black">
                    <Rectangle.Fill>
                        <SolidColorBrush Color="{Binding Path=Color}" />
                    </Rectangle.Fill>
                    <Rectangle>
                        <TextBlock Margin="4" Text="{Binding Path=Formula}" />
                    </Rectangle>
                </DockPanel>
            </DataTemplate>
        </ListBox.ItemTemplate>
    </ListBox>
</Grid>
</Window>

```

A.5 HelpWindow.xaml.cs – окно справки

```

using System;
using System.Windows;

namespace Grapher
{
    /// <summary>
    /// Interaction logic for HelpWindow.xaml
    /// </summary>
    public partial class HelpWindow : Window
    {
        private static HelpWindow? _instance;

        public static void ShowWindow()
        {
            if (_instance is null)
            {
                _instance = new();
                _instance.Show();
            }
            else
                _instance.Activate();
        }

        public HelpWindow()
        {
            InitializeComponent();
        }
    }
}

```

```

private void Window_Closed(object sender, EventArgs e)
{
    if (_instance == this)
        _instance = null;
}
}
}

```

A.6 HelpWindow.xaml – визуальная часть окна справки

```

<Window x:Class="Grapher.HelpWindow"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    xmlns:local="clr-namespace:Grapher"
    mc:Ignorable="d"
    Title="Справочная система - Построитель графиков функций" Height="1024" Width="1280"
    WindowStartupLocation="CenterScreen"
    Closed="Window_Closed"
>
    <FlowDocumentScrollViewer Margin="4" HorizontalScrollBarVisibility="Disabled" VerticalScrollBarVisibility="Auto">
        <FlowDocument IsOptimalParagraphEnabled="True" IsHyphenationEnabled="True" FontSize="16" FontFamily="Segoe UI"
        >
            <Paragraph FontSize="20">
                <Bold>Работа с графиками</Bold>
            </Paragraph>
            <Paragraph>
                Для построения графика функции введите формулу в поле ввода
                <Bold>Формула</Bold> и нажмите кнопку
                <Bold>Добавить</Bold> .
            </Paragraph>
            <Paragraph>
                Формулы могут содержать следующие элементы:
            </Paragraph>
            <List MarkerStyle="Disc" >
                <ListItem>
                    <Paragraph>
                        <Bold>0</Bold>..<Bold>9</Bold> - числовая константа
                    </Paragraph>
                </ListItem>
                <ListItem>
                    <Paragraph>
                        <Bold>.</Bold> - разделитель десятичной и дробной части
                    </Paragraph>
                </ListItem>
                <ListItem>
                    <Paragraph>
                        <Bold>x</Bold> - аргумент формулы
                    </Paragraph>
                </ListItem>
                <ListItem>
                    <Paragraph>
                        <Bold>(</Bold> и
                        <Bold>)</Bold> - скобки для расстановки приоритетов вычислений и для обрамления списков параметров функций
                    </Paragraph>
                </ListItem>
                <ListItem>
                    <Paragraph>
                        <Bold>+</Bold> - операция сложения
                    </Paragraph>
                </ListItem>
                <ListItem>
                    <Paragraph>
                        <Bold>-</Bold> - операция вычитания
                    </Paragraph>
                </ListItem>
                <ListItem>
                    <Paragraph>
                        <Bold>*</Bold> - операция умножения
                    </Paragraph>
                </ListItem>
            </List>
        </FlowDocument>
    </FlowDocumentScrollViewer>
</Window>

```

<ListItem>
 <Paragraph>
 <Bold>/</Bold> - операция деления
 </Paragraph>
 </ListItem>
 <ListItem>
 <Paragraph>
 <Bold>^</Bold> - операция возведения в степень
 </Paragraph>
 </ListItem>
 <ListItem>
 <Paragraph>
 <Bold>pi</Bold> - число π
 </Paragraph>
 </ListItem>
 <ListItem>
 <Paragraph>
 <Bold>e</Bold> - число e
 </Paragraph>
 </ListItem>
 <ListItem>
 <Paragraph>
 <Bold>sin()</Bold> - функция синуса
 </Paragraph>
 </ListItem>
 <ListItem>
 <Paragraph>
 <Bold>cos()</Bold> - функция косинуса
 </Paragraph>
 </ListItem>
 <ListItem>
 <Paragraph>
 <Bold>tg()</Bold> - функция тангенса
 </Paragraph>
 </ListItem>
 <ListItem>
 <Paragraph>
 <Bold>ln()</Bold> - функция натурального логарифма
 </Paragraph>
 </ListItem>
 <ListItem>
 <Paragraph>
 <Bold>lg()</Bold> - функция десятичного логарифма
 </Paragraph>
 </ListItem>
 </List>
 <Paragraph>
 Пример допустимой формулы:
 <Italic>(x^2 + 3*x - 4) / 5 + sin(x)</Italic>
 </Paragraph>
 <Paragraph>
 Для удаления графика выделите его в списке справа или щелчком мыши по нему в поле построения, и нажмите клавишу
 <Bold>Del</Bold> на клавиатуре. Для удаления всех построенных графиков нажмите кнопку
 <Bold>Удалить все</Bold> справа от редактора формул.
 </Paragraph>
 <Paragraph FontSize="20">
 <Bold>Управление координатной сеткой</Bold>
 </Paragraph>
 <Paragraph>
 Для произвольного перемещения рабочей области построителя перемещайте мышь с зажатой левой клавишей. Также
 можно использовать колесо прокрутки для вертикального перемещения рабочей области. Если при этом удерживать нажатой
 клавишу <Bold>Shift</Bold>, то перемещение будет выполняться в горизонтальном направлении.
 </Paragraph>
 <Paragraph>
 Для изменения масштаба рабочей области построителя вращайте колесо мыши, удерживая нажатой клавишу
 <Bold>Ctrl</Bold>.
 </Paragraph>
 <Paragraph>
 Для точной установки заданных координат рабочей области воспользуйтесь полями ввода <Bold>Min X</Bold>,
 <Bold>Max X</Bold>, <Bold>Min Y</Bold> и <Bold>Max Y</Bold>.
 </Paragraph>
 <Paragraph>
 Для сброса координат рабочей области в начальные значения нажмите кнопку <Bold>Сбросить</Bold>.
 </Paragraph>

```

<Paragraph FontSize="20">
  <Bold>Автор</Bold>
</Paragraph>
<Paragraph>
  Мария Плотникова, 9В класс, 2023-2024 г.
</Paragraph>
</FlowDocument>
</FlowDocumentScrollView>
</Window>

```

A.7 Lexeme.cs – определение лексем для логического анализатора

```

namespace Grapher.Analyzer.Lexical;

public enum LexemeType
{
    None,
    Whitespace,
    Plus, Minus, Asterisk, Slash, Power,
    OpeningParenthesis, ClosingParenthesis, Comma,
    Identifier, DoubleLiteral
}

public class Lexeme
{
    public Lexeme(LexemeType type, TextSpan position)
    {
        Type = type;
        Position = position;
    }

    public override string ToString()
    {
        return $"{Type}, position: {Position}";
    }

    public LexemeType Type { get; }

    public TextSpan Position { get; }
}

public sealed class Lexeme<T> : Lexeme
{
    public Lexeme(LexemeType type, TextSpan position, T content)
        : base(type, position)
    {
        Content = content;
    }

    public override string ToString()
    {
        return $"{Type}, position: {Position}, value: {Content}";
    }

    public T Content { get; }
}

```

A.8 LexemeExtensions.cs – расширения для лексем

```

using System;

namespace Grapher.Analyzer.Lexical;

/// <summary>
/// Extension methods for the <see cref="Lexeme"/> class
/// </summary>
public static class LexemeExtensions
{

```

```

public static T GetContent<T>(this Lexeme lexeme)
{
    if (lexeme.GetType() == typeof(Lexeme<T>))
    {
        var result = ((Lexeme<T>)lexeme).Content;
        return result;
    }
    else
        throw new ArgumentException("Токен не содержит значения требуемого типа");
}
}

```

A.9 LexicalAnalyzerException.cs – ошибки логического анализатора

```

namespace Grapher.Analyzer.Lexical;

public sealed class LexicalAnalyzerException : AnalyzerException
{
    private LexicalAnalyzerException(string message, TextSpan errorPosition)
        : base(message, errorPosition) { }

    public static LexicalAnalyzerException UnknownCharacterError(TextSpan errorPosition)
        => new($"Нераспознанный символ входной строки в позиции {errorPosition.Start}", errorPosition);

    public static LexicalAnalyzerException ExtraDecimalSeparator(TextSpan errorPosition)
        => new($"Лишний десятичный разделитель в позиции {errorPosition.Start}", errorPosition);
}

```

A.10 LexicalParser.cs – логический анализатор

```

using System;
using System.Collections.Generic;
using System.Globalization;

namespace Grapher.Analyzer.Lexical;

public sealed class LexicalParser
{
    private List<Func<string, int, Lexeme?>> _evaluators = [];

    private Lexeme? ParseWhitespace(string input, int pos)
    {
        Lexeme? result = null;
        if (char.IsWhiteSpace(input[pos]))
        {
            var newPos = pos;
            do
            {
                newPos++;
            }
            while (newPos < input.Length && char.IsWhiteSpace(input[newPos]));
            result = new Lexeme(LexemeType.Whitespace, new TextSpan(pos, newPos - pos));
        }
        return result;
    }

    private Lexeme? ParseOperator(string input, int pos)
    {
        Lexeme? result = null;
        switch (input[pos])
        {
            case '+':
                result = new Lexeme(LexemeType.Plus, new TextSpan(pos, 1));
                break;
            case '-':
                result = new Lexeme(LexemeType.Minus, new TextSpan(pos, 1));
                break;
            case '*':

```

```

        result = new Lexeme(LexemeType.Asterisk, new TextSpan(pos, 1));
        break;
    case '/':
        result = new Lexeme(LexemeType.Slash, new TextSpan(pos, 1));
        break;
    case '^':
        result = new Lexeme(LexemeType.Power, new TextSpan(pos, 1));
        break;
    case '(':
        result = new Lexeme(LexemeType.OpeningParenthesis, new TextSpan(pos, 1));
        break;
    case ')':
        result = new Lexeme(LexemeType.ClosingParenthesis, new TextSpan(pos, 1));
        break;
    case ',':
        result = new Lexeme(LexemeType.Comma, new TextSpan(pos, 1));
        break;
    }
    return result;
}

private Lexeme? ParseIdentifier(string input, int pos)
{
    Lexeme? result = null;
    var ch = input[pos];
    if (char.IsLetter(ch))
    {
        var newPos = pos;
        do
        {
            newPos++;
        }
        while (newPos < input.Length && char.IsLetterOrDigit(input[newPos]));
        result = new Lexeme<string>(LexemeType.Identifier, new TextSpan(pos, newPos - pos),
            input[pos..newPos]);
    }
    return result;
}

private Lexeme? ParseNumberLiteral(string input, int pos)
{
    Lexeme? result = null;
    if (char.IsDigit(input[pos]))
    {
        var newPos = pos;
        var foundPeriod = false;
        do
        {
            newPos++;
            if (newPos < input.Length)
            {
                if (input[newPos] == '.')
                {
                    if (!foundPeriod)
                    {
                        foundPeriod = true;
                        newPos++;
                    }
                }
                else
                {
                    throw LexicalAnalyzerException.ExtraDecimalSeparator(new TextSpan(newPos, 1));
                }
            }
        }
        while (newPos < input.Length && char.IsDigit(input[newPos]));
        result = new Lexeme<double>(LexemeType.DoubleLiteral, new TextSpan(pos, newPos - pos),
            double.Parse(input.AsSpan(pos, newPos - pos), CultureInfo.InvariantCulture));
    }
    return result;
}

public LexicalParser()
{
    _evaluators.Add(ParseWhitespace);
    _evaluators.Add(ParseNumberLiteral);
    _evaluators.Add(ParseOperator);
    _evaluators.Add(ParseIdentifier);
}

```

```

    }

    public IReadOnlyCollection<Lexeme> Parse(string input)
    {
        var result = new List<Lexeme>();
        var i = 0;
        while (i < input.Length)
        {
            Lexeme? lexeme = null;
            foreach (var evaluator in _evaluators)
            {
                lexeme = evaluator(input, i);
                if (lexeme != null)
                {
                    i += lexeme.Position.Length;
                    break;
                }
            }
            if (lexeme != null)
                result.Add(lexeme);
            else
                throw LexicalAnalyzerException.UnknownCharacterError(new TextSpan(i, 1));
        }
        return result;
    }
}

```

A.11 AstNode.cs – узлы АСД

```

using System.Collections.Generic;
using System.Linq;

namespace Grapher.Analyzer.Syntactical;

public enum AstNodeType { Expression, FunctionCall, BinaryOperation, UnaryOperation, Variable, Literal }

public abstract class AstNode
{
    private readonly List<AstNode> _children;

    protected AstNode(AstNodeType type)
    {
        _children = new List<AstNode>();
        Type = type;
    }

    protected void AddChild(AstNode child)
    {
        _children.Add(child);
        child.Parent = this;
    }

    public override string ToString()
    {
        return Type.ToString();
    }

    public IReadOnlyList<AstNode> Children { get => _children; }

    public AstNode? Parent { get; protected set; }

    public AstNodeType Type { get; }
}

public abstract class AstExpression : AstNode
{
    public AstExpression(AstNodeType type) : base(type) { }
}

```

```

public class AstLiteral<T> : AstExpression
{
    public AstLiteral(T value)
        : base(AstNodeType.Literal)
    {
        Value = value;
    }

    public T Value { get; }
}

public class AstVariable : AstExpression
{
    public AstVariable(string identifier)
        : base(AstNodeType.Variable)
    {
        Identifier = identifier;
    }

    public string Identifier { get; }
}

public class AstFunctionCall : AstExpression
{
    public AstFunctionCall(string identifier, IEnumerable<AstExpression> parameters)
        : base(AstNodeType.FunctionCall)
    {
        Identifier = identifier;
        foreach (var parameter in parameters)
            AddChild(parameter);
    }

    public string Identifier { get; }

    public IReadOnlyList<AstExpression> Parameters { get => Children.Cast<AstExpression>().ToList(); }
}

public enum UnaryOperator { Plus, Minus }

public class AstUnaryOperation : AstExpression
{
    public AstUnaryOperation(UnaryOperator @operator, AstExpression operand)
        : base(AstNodeType.UnaryOperation)
    {
        Operator = @operator;
        AddChild(operand);
    }

    public UnaryOperator Operator { get; }
    public AstExpression Operand { get => (AstExpression)Children[0]; }
}

public enum BinaryOperator { Addition, Subtraction, Multiplication, Division, Power }

public class AstBinaryOperation : AstExpression
{
    public AstBinaryOperation(AstExpression left, BinaryOperator @operator, AstExpression right)
        : base(AstNodeType.BinaryOperation)
    {
        Operator = @operator;
        AddChild(left);
        AddChild(right);
    }

    public AstExpression Left { get => (AstExpression)Children[0]; }

    public BinaryOperator Operator { get; }

    public AstExpression Right { get => (AstExpression)Children[1]; }
}

```

A.12 SyntacticalAnalyzerException.cs – ошибки синтаксического анализатора

```
namespace Grapher.Analyzer.Syntactical;

public sealed class SyntacticalAnalyzerException : AnalyzerException
{
    private SyntacticalAnalyzerException(string message, TextSpan errorPosition)
        : base(message, errorPosition) { }

    public static SyntacticalAnalyzerException UnclosedParenthesesError(TextSpan errorPosition)
        => new SyntacticalAnalyzerException($"Не найдена закрывающая круглая скобка", errorPosition);

    public static SyntacticalAnalyzerException ExpectingIdentifierError(TextSpan errorPosition)
        => new SyntacticalAnalyzerException($"Ожидается идентификатор", errorPosition);

    public static SyntacticalAnalyzerException ExpectingExpressionError(TextSpan errorPosition)
        => new SyntacticalAnalyzerException($"Ожидается выражение", errorPosition);

    public static SyntacticalAnalyzerException UnpxpectedToken(TextSpan errorPosition)
        => new SyntacticalAnalyzerException($"Неожиданный символ после окончания выражения", errorPosition);
}
```

A.12 SyntacticalParser.cs – синтаксический анализатор

```
using Grapher.Analyzer.Lexical;
using System.Collections.Generic;
using System.Linq;

namespace Grapher.Analyzer.Syntactical;

public sealed class SyntacticalParser
{
    private static readonly HashSet<LexemeType> _triviaLexemeTypes = new[] { LexemeType.Whitespace }.ToHashSet();
    private readonly List<Lexeme> _lexemes;
    private int _lexemeIndex;
    private Lexeme _emptyLexeme;

    public SyntacticalParser()
    {
        _lexemes = new List<Lexeme>();
        _emptyLexeme = new Lexeme(LexemeType.None, new TextSpan(0, 0));
    }

    private Lexeme Lexeme(int offset = 0)
    {
        Lexeme result;
        var index = _lexemeIndex + offset;
        if (0 <= index && index < _lexemes.Count)
            result = _lexemes[index];
        else
            result = _emptyLexeme;
        return result;
    }

    private void Move(int offset)
    {
        _lexemeIndex += offset;
    }

    private bool Eol { get => _lexemeIndex >= _lexemes.Count; }

    // выражение -> терм3(( "+" | "-" ) терм3)*
    // терм3 -> терм2 (( "*" | "/" ) терм2 )*
    // терм2 -> терм1 ( "^" ) терм1)*
    // терм1 -> [+|-] (" выражение ") | вызов | идентификатор | число)
    // вызов -> идентификатор "(" выражение ")"

    // терм1 -> [+|-] (" выражение ") | вызов | идентификатор | число)
}
```

```

private AstExpression BuildTerm1()
{
    AstExpression term;
    UnaryOperator? unaryOperator = null;
    switch (Lexeme().Type)
    {
        case LexemeType.Plus:
            unaryOperator = UnaryOperator.Plus;
            Move(+1);
            break;
        case LexemeType.Minus:
            unaryOperator = UnaryOperator.Minus;
            Move(+1);
            break;
    }

    switch (Lexeme().Type)
    {
        case LexemeType.OpeningParenthesis:
            var opening = Lexeme().Position.Start;
            Move(+1);
            term = BuildExpression();
            if (Lexeme().Type == LexemeType.ClosingParenthesis)
                Move(+1);
            else
                throw SyntacticalAnalyzerException.UnclosedParenthesesError(new TextSpan(opening,
                    Lexeme().Position.Start - opening));
            break;
        case LexemeType.Identifier:
            var identifier = Lexeme().GetContent<string>();
            Move(+1);
            if (Lexeme().Type == LexemeType.OpeningParenthesis)
            {
                var opening2 = Lexeme().Position.Start;
                var parameters = new List<AstExpression>();
                do
                {
                    Move(+1);
                    var param = BuildExpression();
                    parameters.Add(param);
                } while (Lexeme().Type == LexemeType.Comma);

                if (Lexeme().Type == LexemeType.ClosingParenthesis)
                {
                    term = new AstFunctionCall(identifier, parameters);
                    Move(+1);
                }
                else
                    throw SyntacticalAnalyzerException.UnclosedParenthesesError(new TextSpan(opening2,
                        Lexeme().Position.Start - opening2));
            }
            else
                term = new AstVariable(identifier);
            break;
        case LexemeType.DoubleLiteral:
            term = new AstLiteral<double>(Lexeme().GetContent<double>());
            Move(+1);
            break;
        default:
            throw SyntacticalAnalyzerException.ExpectingExpressionError(new TextSpan(Lexeme().Position.Start, 0));
    }

    // навешиваем унарную операцию, если таковая была
    if (unaryOperator.HasValue)
        term = new AstUnaryOperation(unaryOperator.Value, term);

    return term;
}

// терм2 -> терм1 ( "^" ) терм1 *
private AstExpression BuildTerm2()
{
    AstExpression powTerm;
    var left = BuildTerm1();

```

```

while (true)
{
    AstExpression right;
    var lexemeType = Lexeme().Type;
    if (lexemeType == LexemeType.Power)
    {
        Move(+1);
        right = BuildTerm1();
        powTerm = new AstBinaryOperation(left, BinaryOperator.Power, right);
        left = powTerm;
    }
    else
    {
        powTerm = left;
        break;
    }
}
return powTerm;
}

// терм3 -> терм2 (( "*" | "/" ) терм2 )*
private AstExpression BuildTerm3()
{
    AstExpression group;
    var left = BuildTerm2();
    while (true)
    {
        AstExpression right;
        var lexemeType = Lexeme().Type;
        if (lexemeType == LexemeType.Asterisk)
        {
            Move(+1);
            right = BuildTerm2();
            group = new AstBinaryOperation(left, BinaryOperator.Multiplication, right);
            left = group;
        }
        else if (lexemeType == LexemeType.Slash)
        {
            Move(+1);
            right = BuildTerm2();
            group = new AstBinaryOperation(left, BinaryOperator.Division, right);
            left = group;
        }
        else
        {
            group = left;
            break;
        }
    }
    return group;
}

// выражение -> терм3 (( "+" | "-" ) терм3)*
private AstExpression BuildExpression()
{
    AstExpression expression;
    var left = BuildTerm3();
    while (true)
    {
        var lexemeType = Lexeme().Type;
        AstExpression right;
        if (lexemeType == LexemeType.Plus)
        {
            Move(+1);
            right = BuildTerm3();
            expression = new AstBinaryOperation(left, BinaryOperator.Addition, right);
            left = expression;
        }
        else if (lexemeType == LexemeType.Minus)
        {
            Move(+1);
            right = BuildTerm3();
            expression = new AstBinaryOperation(left, BinaryOperator.Substraction, right);

```

```

        left = expression;
    }
    else
    {
        expression = left;
        break;
    }
}
return expression;
}
private AstExpression BuildTopLevelExpression()
{
    var expression = BuildExpression();
    if (Eol)
        return expression;
    else
        throw SyntacticalAnalyzerException.UnpxpectedToken(new TextSpan(Lexeme().Position.Start, 0));
}

public AstExpression Parse(IEnumerable<Lexeme> lexemes)
{
    _lexemes.Clear();
    _lexemes.AddRange(lexemes.Where(t => !_triviaLexemeTypes.Contains(t.Type)));
    _lexemeIndex = 0;
    var result = BuildTopLevelExpression();
    return result;
}
}

```

A.13 AnalyzerExceptionBase.cs – базовый класс для ошибок анализаторов

```

using System;

namespace Grapher.Analyzer;

public abstract class AnalyzerException : ApplicationException
{
    protected AnalyzerException(string message, TextSpan? errorPosition)
        : base(message)
    {
        ErrorPosition = errorPosition;
    }

    public TextSpan? ErrorPosition { get; }
}

```

A.15 TextSpan.cs – текстовый блок

```

namespace Grapher.Analyzer;

public readonly struct TextSpan
{
    public TextSpan(int start, int length)
    {
        Start = start;
        Length = length;
    }

    public override string ToString()
    {
        return $"({Start}, {Length})";
    }

    public int Start { get; }

    public int Length { get; }
}

```

A.16 ExpressionBuilder.cs – построитель выражений

```
using Grapher.Analyzer.Syntactical;
using System;
using System.Linq.Expressions;
using System.Reflection;

namespace Grapher.Analyzer;

public class ExpressionBuilder
{
    private readonly Expression _pi = Expression.Constant(Math.PI);
    private readonly Expression _e = Expression.Constant(Math.E);
    private readonly ParameterExpression _x = Expression.Parameter(typeof(double));
    private readonly MethodInfo _sinMethod = typeof(Math).GetMethod(nameof(Math.Sin));
    private readonly MethodInfo _cosMethod = typeof(Math).GetMethod(nameof(Math.Cos));
    private readonly MethodInfo _tgMethod = typeof(Math).GetMethod(nameof(Math.Tan));
    private readonly MethodInfo _lnMethod = typeof(Math).GetMethod(nameof(Math.Log), [typeof(double)]);
    private readonly MethodInfo _lgMethod = typeof(Math).GetMethod(nameof(Math.Log10));
    private readonly MethodInfo _logMethod = typeof(Math).GetMethod(nameof(Math.Log), [typeof(double), typeof(double)]);

    private Expression VisitNode(AstNode node) => node switch
    {
        AstLiteral<double> literal => Expression.Constant(literal.Value),
        AstVariable variable => variable.Identifier.ToLowerInvariant() switch
        {
            "pi" => _pi,
            "e" => _e,
            "x" => _x,
            _ => throw new ArgumentException($"Неизвестная переменная: {variable.Identifier}.")
        },
        AstFunctionCall call when call.Parameters.Count != 1 => throw new ArgumentException($"Поддерживаются только функции с одним аргументом."),
        AstFunctionCall call => call.Identifier.ToLowerInvariant() switch
        {
            "sin" => Expression.Call(null, _sinMethod, VisitNode(call.Parameters[0])),
            "cos" => Expression.Call(null, _cosMethod, VisitNode(call.Parameters[0])),
            "tg" => Expression.Call(null, _tgMethod, VisitNode(call.Parameters[0])),
            "ln" => Expression.Call(null, _lnMethod, VisitNode(call.Parameters[0])),
            "lg" => Expression.Call(null, _lgMethod, VisitNode(call.Parameters[0])),
            _ => throw new ArgumentException($"Неизвестная функция: {call.Identifier}.")
        },
        AstUnaryOperation uo => uo.Operator switch
        {
            UnaryOperator.Plus => VisitNode(uo.Operand),
            UnaryOperator.Minus => Expression.Negate(VisitNode(uo.Operand)),
            _ => throw new ArgumentException($"Неизвестный унарный оператор: {uo.Operator}.")
        },
        AstBinaryOperation bo => bo.Operator switch
        {
            BinaryOperator.Addition => Expression.Add(VisitNode(bo.Left), VisitNode(bo.Right)),
            BinaryOperator.Subtraction => Expression.Subtract(VisitNode(bo.Left), VisitNode(bo.Right)),
            BinaryOperator.Multiplication => Expression.Multiply(VisitNode(bo.Left), VisitNode(bo.Right)),
            BinaryOperator.Division => Expression.Divide(VisitNode(bo.Left), VisitNode(bo.Right)),
            BinaryOperator.Power => Expression.Call(null, typeof(Math).GetMethod(nameof(Math.Pow)), VisitNode(bo.Left), VisitNode(bo.Right)),
            _ => throw new ArgumentException($"Неизвестный бинарный оператор: {bo.Operator}.")
        },
        _ => throw new ArgumentException($"Неизвестный элемент выражения: {node.Type}.")
    };

    public Func<double, double> Build(AstExpression ast)
    {
        var expression = VisitNode(ast);
        var lb = Expression.Lambda<Func<double, double>>(expression, _x);
        var lambda = lb.Compile();
        return lambda;
    }
}
```