

Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

К защите допустить:
Заведующий кафедрой ПОИТ
_____ Н.В. Лапицкая

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
курсовой работы
на тему

**ПРОГРАММНОЕ СРЕДСТВО КАЛЬКУЛЯТОР С ВОЗМОЖНОСТЬЮ
ОБРАБОТКИ ВЫРАЖЕНИЙ И ПОСТРОЕНИЯ ГРАФИКОВ
ФУНКЦИЙ**
БГУИР КР 6-05 06 12 01 109 ПЗ

Студент

П.А. Забелич

Руководитель

Е.Е. Фадеева

Минск 2024

СОДЕРЖАНИЕ

ВВЕДЕНИЕ.....	2
1.АНАЛИЗ ПРОТОТИПОВ, ЛИТЕРАТУРНЫХ ИСТОЧНИКОВ И ФОРМИРОВАНИЕ ТРЕБОВАНИЙ К ПРОЕКТИРУЕМОМУ	3
1.1 Графический калькулятор	3
1.2 Функциональные возможности:	4
1.3 Литературные источники	4
1.4 Формирование требований к программному средству	4
2.АНАЛИЗ ТРЕБОВАНИЙ К ПРОГРАММНОМУ СРЕДСТВУ И РАЗРАБОТКА ФУНКЦИОНАЛЬНЫХ ТРЕБОВАНИЙ.....	6
2.1 Теоретический анализ и математическое обоснование.....	6
2.2 Описание функциональности ПС	7
2.3 Спецификация функциональных требований	7
3.ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА.....	10
4.СОЗДАНИЕ (КОНСТРУИРОВАНИЕ) ПРОГРАММНОГО СРЕДСТВА	16
4.1 Разработка программных интерфейсов и структуры модулей.....	16
4.2 Программирование и отладка	16
4.3 Сборка проекта	17
4.4 Результат	17
5. ТЕСТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА.....	18
5.1 Методика тестирования	18
5.2 Тестовые случаи	18
5.3 Результаты тестирования.....	19
5.4 Заключение по тестированию	19
6. РУКОВОДСТВО ПО УСТАНОВКЕ И ИСПОЛЬЗОВАНИЮ ПРОГРАММНОГО СРЕДСТВА.....	20
6.1 Общие сведения.....	20
6.2 Системные требования	20
6.3 Установка программного средства.....	20
6.4 Инструкция по использованию	20
ЗАКЛЮЧЕНИЕ	24
7. СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ	25

ВВЕДЕНИЕ

Разработка программных средств для обработки математических выражений и визуализации функций актуальна для образования, инженерных расчетов и научных исследований. Современные графические калькуляторы, такие как Desmos и Mathway, обеспечивают интерактивную работу с выражениями и графиками, что делает их востребованными среди студентов и специалистов.

Цель курсовой работы — создание программного средства «Графический калькулятор» для обработки выражений и построения графиков функций вида $y = f(x)$. Программа должна предоставлять удобный интерфейс, поддержку базовых и специальных операций, а также интерактивные функции: масштабирование, перемещение координатной плоскости и копирование графиков в буфер обмена.

Проектирование основано на принципах модульности и надежности, с использованием алгоритмов лексического и синтаксического анализа. Реализация выполнена на языке Delphi в среде Embarcadero RAD Studio для платформы Windows.

Курсовая работа включает:

- Анализ прототипов, литературы и формирование требований.
- Разработку функциональных требований.
- Проектирование с созданием алгоритмов и схем.
- Реализацию и отладку программного средства.
- Тестирование и анализ результатов.
- Руководство по установке и использованию.

Работа направлена на создание надежного и функционального инструмента для образовательных и практических задач.

1. АНАЛИЗ ПРОТОТИПОВ, ЛИТЕРАТУРНЫХ ИСТОЧНИКОВ И ФОРМИРОВАНИЕ ТРЕБОВАНИЙ К ПРОЕКТИРУЕМОМУ

1.1 Графический калькулятор

Программное средство (далее ПС) “Графический калькулятор” предназначено для графической визуализации математических выражений. Данный вариант калькулятора должен позволять отрисовывать графики функций вида $y=F(x)$.

Примеры прототипов:

Калькулятор Windows – стандартное приложение, которое выполняет базовые арифметические операции. Однако оно не поддерживает обработку сложных математических выражений и построение графиков.

Mathway – это мощное программное средство для решения математических задач, которое использует алгоритмы символьных вычислений и численного анализа. Оно предназначено для автоматического решения уравнений, построения графиков и выполнения сложных математических операций.

Функциональные возможности:

- Решение алгебраических, тригонометрических, логарифмических и экспоненциальных выражений.
- Построение графиков функций с возможностью масштабирования. Автоматическое разложение выражений и упрощение формул.
- Поддержка различных математических дисциплин, включая статистику и линейную алгебру.
- Интерактивный интерфейс с возможностью ввода выражений вручную или с помощью виртуальной клавиатуры.

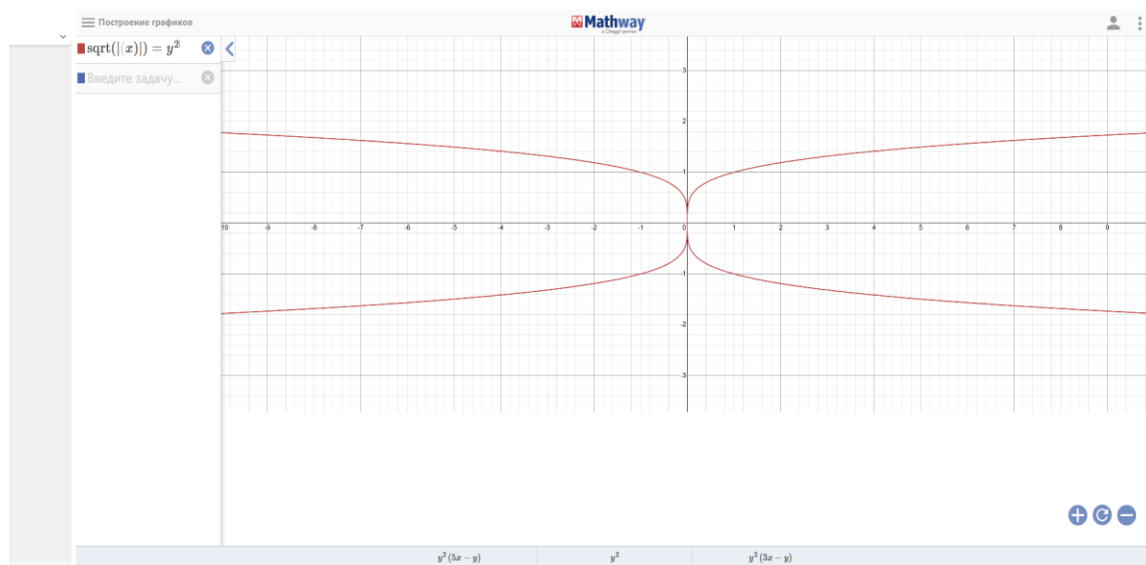


Рисунок 1–MathWay

Desmos – это мощное программное средство для визуализации математических функций и выполнения вычислений. Оно представляет собой

интерактивный графический калькулятор, который позволяет пользователям строить графики, анализировать математические выражения и проводить моделирование различных процессов.

1.2 Функциональные возможности:

- Построение графиков функций в реальном времени.
- Поддержка параметрических и полярных уравнений.
- Возможность анимации графиков для динамического анализа.
- Работа с таблицами данных и их визуализация.
- Интерактивные инструменты для геометрических построений.

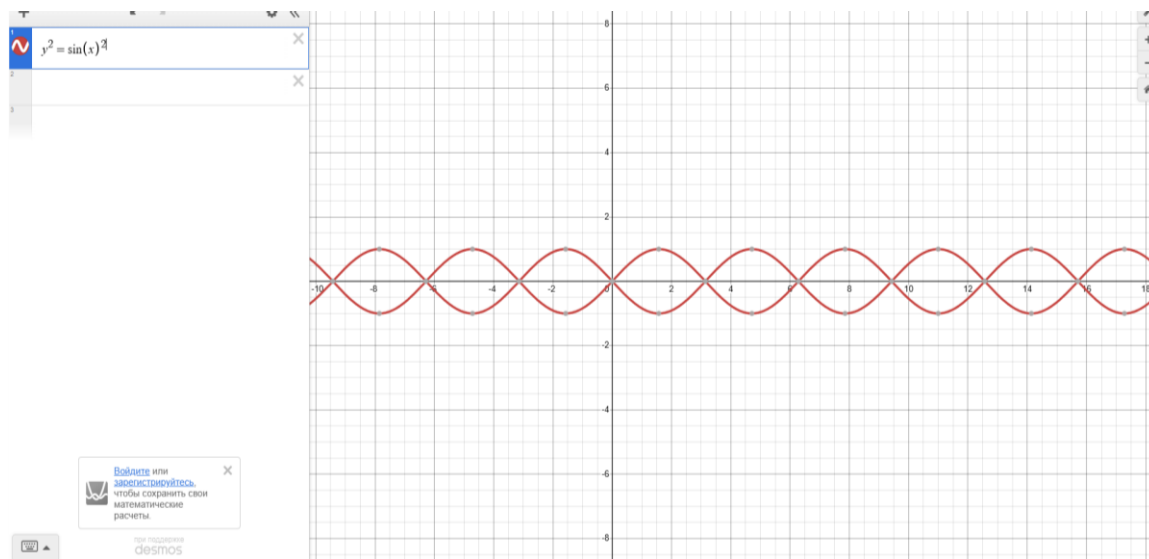


Рисунок 2–Desmos

1.3 Литературные источники

- **"Algebra and Trigonometry" – Jay Abramson** Книга охватывает основные аспекты алгебры и тригонометрии, включая примеры и упражнения.
- **"The Matrix Cookbook"** Компактное руководство по линейной алгебре и матричным вычислениям, полезное для работы с математическими выражениями.
- **"Probability Theory" – E. T. Jaynes** Фундаментальная книга по теории вероятностей, которая может пригодиться при разработке аналитических функций.
- **"35 лучших книг для программистов" – DEVGUIDE.RU** Список рекомендованных книг по программированию, включая материалы по алгоритмам и математическим вычислениям.

1.4 Формирование требований к программному средству

На основе анализа прототипов и изучения литературы сформированы требования к проектируемому программному средству.

Функциональные требования

Обработка математических выражений:

- Поддержка базовых математических операций (+, -, /, *, ^)

- Поддержка унарных функций (sin,sinh,arcsin,exp,sqrt и др.)
- Поддержка бинарных функций (log(a,b), pow(a,b))
- Поддержка переменной x

Построение графиков функций:

- Построение графиков функций одной переменного, вида $y=f(x)$
- Настройка диапазона значений
- по осям X и Y.
- Отображение сетки на графике.
- Возможность отображения графиков множества функций

Копирование графика

- Возможность сохранения изображения графика вместе с координатной плоскостью в буфер обмена

Требования к надёжности

- Программа должна корректно обрабатывать математическое выражение в независимости от его корректности.
- Программа должна оставаться работоспособной даже при некорректном её использовании

Входные данные

- Выражение в формате строк
- Промежутки отображения по X и Y

Выходные данные

- График с сеткой координат в заданных промежутках

Технические требования

- Язык программирования: Delphi.
- Среда разработки: Embarcadero RAD.
- Платформа Windows

Библиотеки:

- Для математических вычислений: стандартные функции Delphi.
- Для сохранения в буфер обмена: Clipboard

2. АНАЛИЗ ТРЕБОВАНИЙ К ПРОГРАММНОМУ СРЕДСТВУ И РАЗРАБОТКА ФУНКЦИОНАЛЬНЫХ ТРЕБОВАНИЙ

На основе технического задания, сформулированного в первом разделе, а также анализа прототипов и литературных источников, в данном разделе проводится анализ требований к программному средству (ПС) «Графический калькулятор» и разрабатываются функциональные требования. Раздел включает описание функциональности ПС, спецификацию функциональных требований и, при необходимости, теоретический анализ и моделирование предметной области.

2.1 Теоретический анализ и математическое обоснование

Для реализации программного средства «Графический калькулятор» требуется обеспечить обработку математических выражений и построение графиков функций вида $y = f(x)$. Основные аспекты, которые необходимо учесть, включают:

- **Лексический и синтаксический анализ выражений.** Для обработки математических выражений используется алгоритм лексического анализа, который разбивает входную строку на токены (числа, операторы, функции, скобки). Далее синтаксический анализ строит абстрактное синтаксическое дерево (AST), которое позволяет корректно интерпретировать порядок операций с учетом приоритетов и скобок. Это обеспечивает обработку выражений, включающих базовые операции (+, -, *, /, ^), унарные функции (sin, cos, sinh, arcsin, exp, ln, lg, abs, sqrt) и бинарные функции (log(a,b), pow(a,b)).

- **Построение графиков.** Для визуализации функций вида $y = f(x)$ используется метод дискретизации: функция вычисляется в заданном диапазоне значений x с определенным шагом. Полученные точки преобразуются в пиксели для отображения на координатной плоскости. Для обеспечения интерактивности поддерживается масштабирование (приближение/удаление) и перемещение координатной плоскости.

- **Обработка ошибок.** Программа должна корректно обрабатывать некорректные выражения (например, деление на ноль, неверный синтаксис), не приводя к сбою системы.

- **Математическое обоснование.** Для реализации тригонометрических, экспоненциальных и логарифмических функций используются стандартные математические библиотеки Delphi, обеспечивающие высокую точность вычислений. Например, значения констант π (3.1415) и e (2.7182) берутся из стандартных библиотек, а вычисления производятся с точностью до 4 знаков после запятой.

Этот подраздел подтверждает возможность реализации требуемой функциональности на основе стандартных алгоритмов и библиотек, доступных в среде разработки Embarcadero RAD Studio.

2.2 Описание функциональности ПС

Функциональность программного средства описывается с использованием диаграммы вариантов использования (Use Case), которая отражает взаимодействие пользователя с системой для достижения значимых результатов. Ниже представлена диаграмма вариантов использования, описывающая основные сценарии взаимодействия.

Вариант использования	Описание
Ввод математического выражения	Пользователь вводит математическое выражение (например, $y = 2 \cdot \sin(x) + x^2$) в текстовое поле. Система выполняет парсинг выражения в реальном времени, проверяет его корректность и, если выражение валидно, отображает результат или график.
Построение графика функции	Пользователь задает функцию вида $y = f(x)$ и диапазоны по осям X и Y. Система строит график функции на координатной плоскости с возможностью масштабирования и перемещения.
Копирование графика	Пользователь нажимает кнопку «Сору», и график сохраняется в буфер обмена в формате изображения (PNG/JPEG).
Управление координатной плоскостью	Пользователь может масштабировать (приближать/удалять) и перемещать координатную плоскость для анализа графика.
Обработка ошибок	При вводе некорректного выражения (например, неверный синтаксис или деление на ноль) система выводит сообщение об ошибке, указывая ее тип, и предотвращает отрисовку графика.

Таблица 1-Варианты использования

2.3 Спецификация функциональных требований

На основе технического задания и анализа прототипов (Mathway, Desmos) сформулированы следующие функциональные требования к программному средству:

Обработка математических выражений:

- Пользователь вводит выражение в текстовое поле интерфейса в формате строки (например, $y = 2 \cdot \sin(x) + x^2$).
- Поддерживаемые операции и функции:
 - Базовые операции: +, -, *, /, ^ (степень).
 - Унарные функции: $\sin(x)$, $\cos(x)$, $\tan(x)$, $\sinh(x)$, $\arcsin(x)$, $\exp(x)$, $\ln(x)$, $\lg(x)$, $\text{abs}(x)$, $\text{sqrt}(x)$.

- Бинарные функции: $\log(a,b)$, $\text{pow}(a,b)$.
- Константы: π (3.1415), e (2.7182).
- Поддержка скобок для задания приоритета операций.
- Парсинг выражения выполняется в реальном времени. Если выражение некорректно (например, неверный синтаксис, несбалансированные скобки, деление на ноль), выводится сообщение об ошибке с указанием ее типа.

Построение графиков функций:

- Пользователь задает одну или несколько функций вида $y = f(x)$ в текстовом поле.
- График отображается на координатной плоскости с сеткой.
- Поддерживается настройка диапазонов по осям X и Y :
- Пользователь может задавать произвольные диапазоны (например, $X \in [-10, 10]$, $Y \in [-5, 5]$).
- Координатная плоскость поддерживает:
- Масштабирование (приближение/удаление) с помощью колесика мыши или кнопок интерфейса.
- Перемещение плоскости для анализа различных участков графика.
- Поддерживается одновременное отображение графиков нескольких функций с различными цветами линий для их различения.
- Пользователь может выбрать тип линии (сплошная, пунктирная) и толщину линии.

Копирование графика:

- При нажатии кнопки «Сору» график вместе с координатной плоскостью сохраняется в буфер обмена
- Изображение включает координатную сетку и подписи осей.

Требования к надежности:

- Программа должна корректно обрабатывать любые входные данные, включая некорректные выражения, без сбоев.
- При обнаружении ошибок (например, деление на ноль, неверный синтаксис) выводится информативное сообщение об ошибке.

Входные данные:

- Математическое выражение в формате строки.
- Диапазоны отображения по осям X и Y (опционально).

Выходные данные:

- График функции с координатной сеткой в заданных диапазонах.
- Сообщения об ошибках при некорректных выражениях.
- Изображение графика в буфер обмена (при использовании функции копирования).

Технические требования:

- Язык программирования: Delphi.
- Среда разработки: Embarcadero RAD Studio.
- Платформа: Windows.
- Используемые библиотеки:
 - Стандартные функции Delphi для математических вычислений.
 - Модуль Clipboard для сохранения изображения графика в буфер обмена.

Эти требования обеспечивают реализацию функциональности, аналогичной прототипам (Desmos), с учетом интерактивного интерфейса и обработки ошибок. Спецификация функциональных требований служит основой для дальнейшего проектирования и разработки программного средства, а также для создания тестов для проверки его работоспособности.

3. ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

В данном разделе представлены схемы, описывающие алгоритмы и структуру программного средства (ПС) «Графический калькулятор» в соответствии с ГОСТ 19.701-90. Каждая схема сопровождается кратким текстовым описанием, поясняющим ее назначение и роль в работе программы.

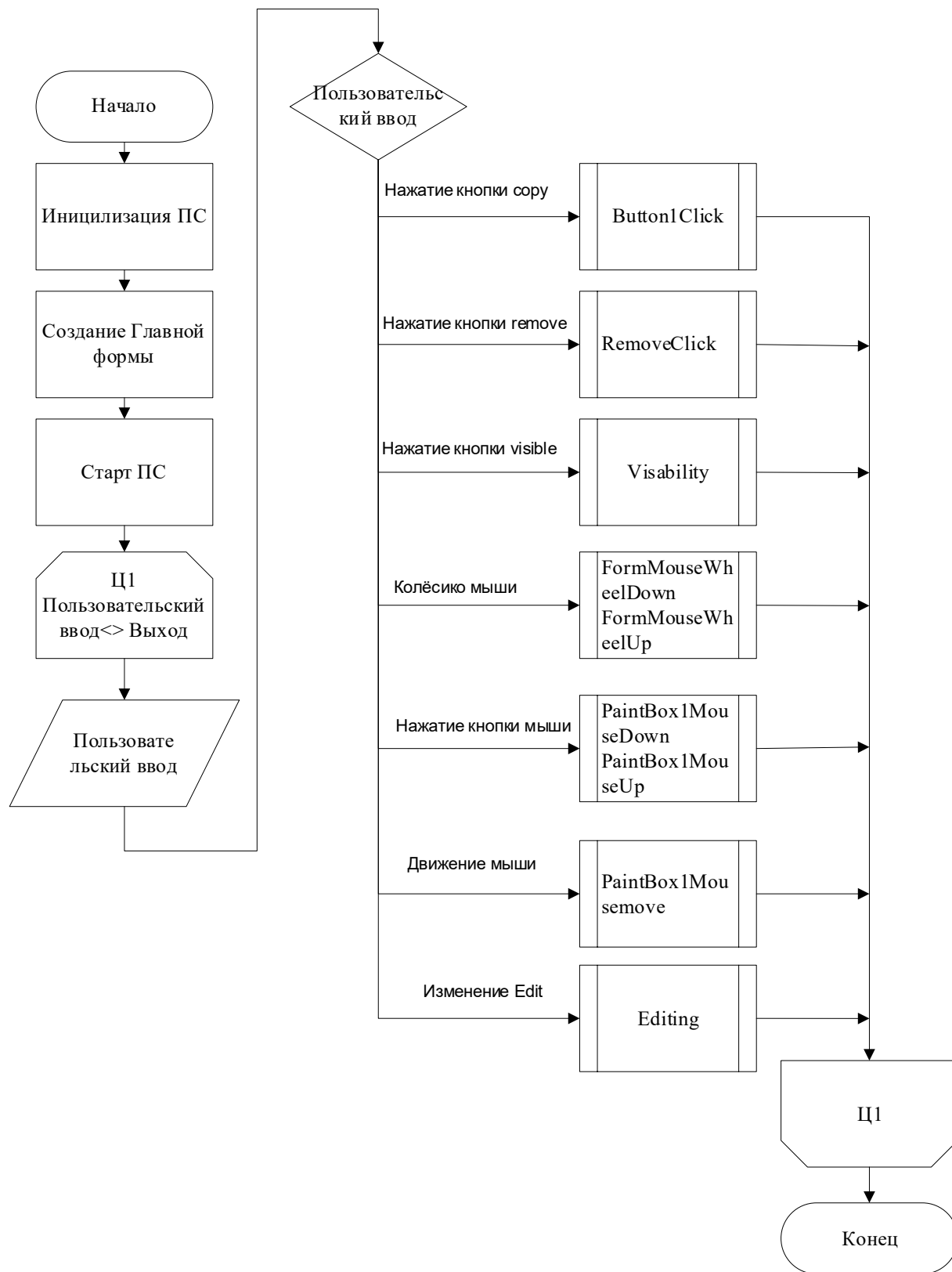


Рисунок 3- Вход в программу

Описание: Схема отображает начальный этап работы ПС. После запуска инициализируется интерфейс, и пользователю предлагается выбрать действие: ввод функции или выход. В зависимости от выбора программа переходит к соответствующему модулю или завершает работу.

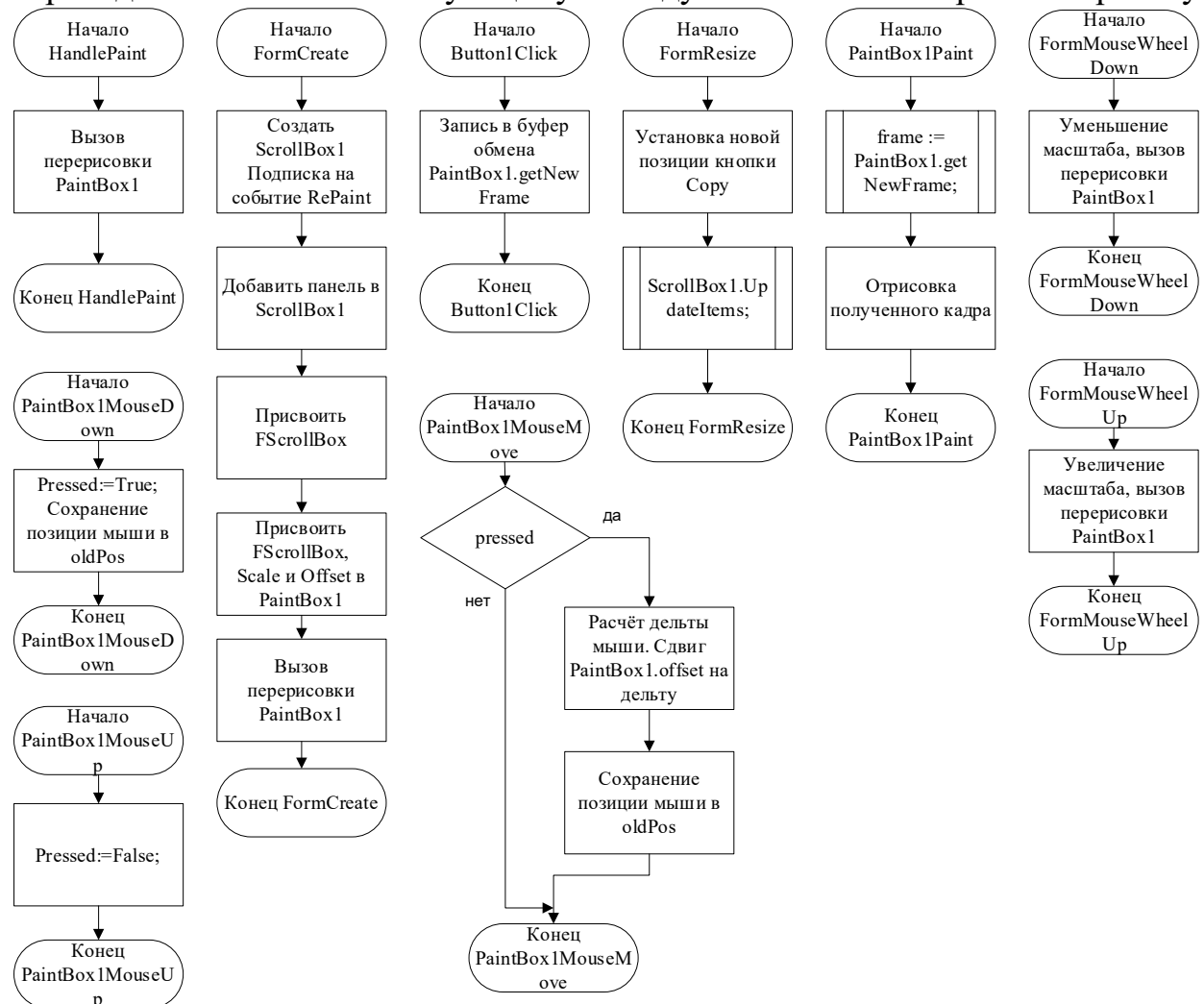


Рисунок 4-Описание главных функций

Описание: Схема иллюстрирует основные функции ПС: ввод функции $y = f(x)$, ее парсинг, проверка корректности, построение графика, управление координатной плоскостью и копирование графика в буфер обмена. Отражает последовательность выполнения ключевых операций.

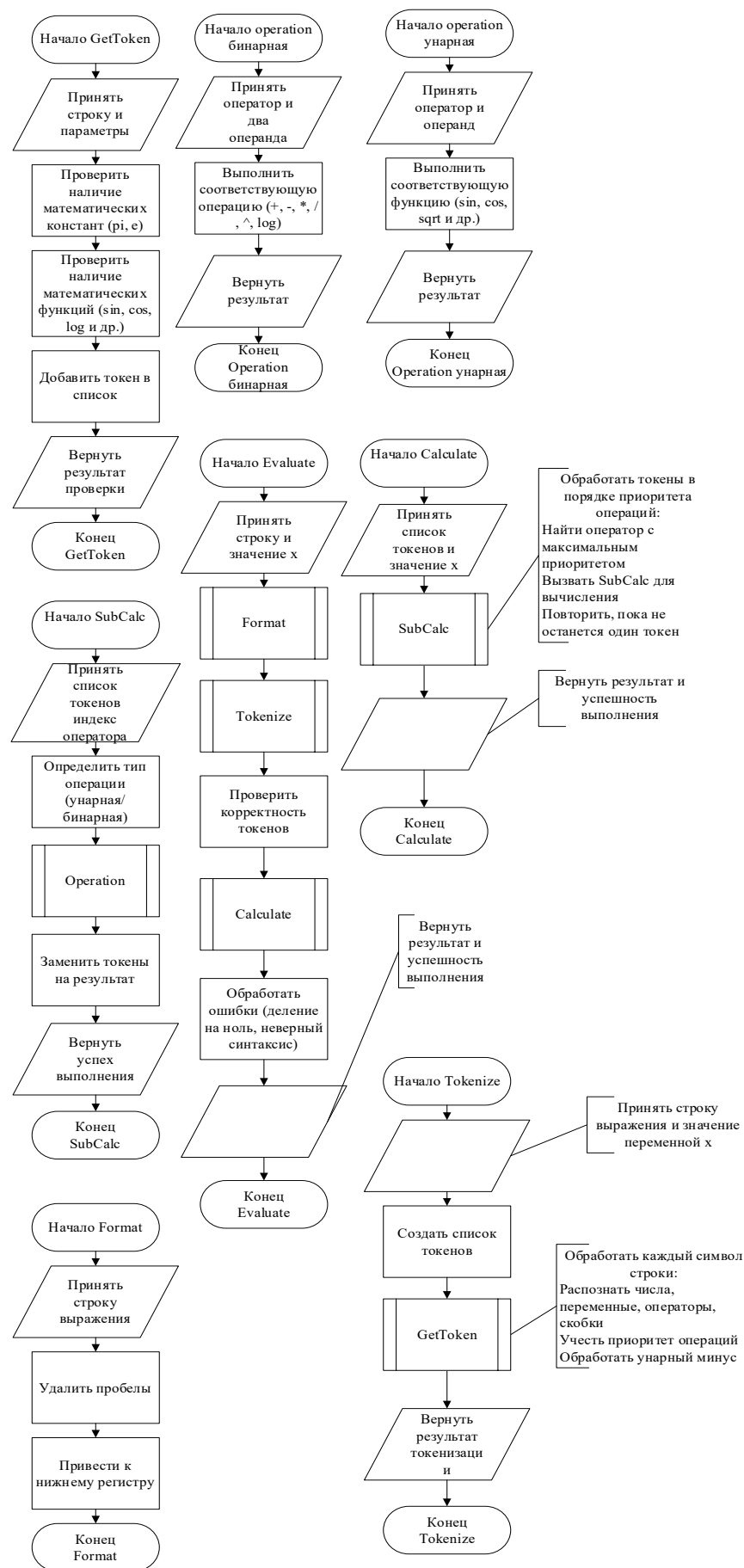


Рисунок 5-Схема Математического парсера

Описание: Схема описывает процесс обработки математического выражения. Включает этапы лексического анализа (разбиение на токены), синтаксического анализа (построение AST), проверки корректности и вычисления результата с выводом сообщения при ошибке.

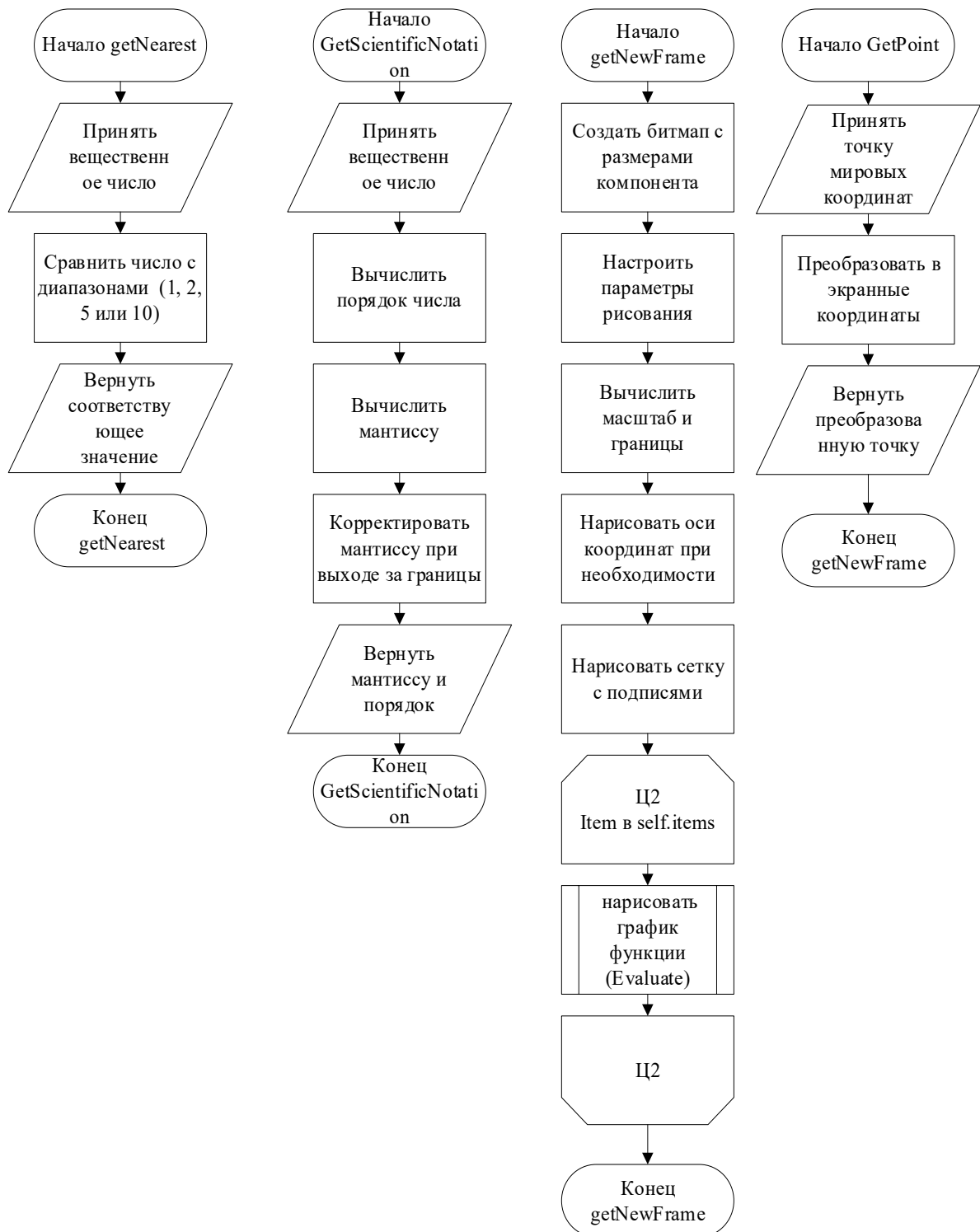


Рисунок 6-Схема работы поля

Описание: Схема показывает алгоритм работы с координатной плоскостью. Пользователь задает диапазоны осей, система дискретизирует функцию, отображает график и поддерживает интерактивное масштабирование и перемещение плоскости.

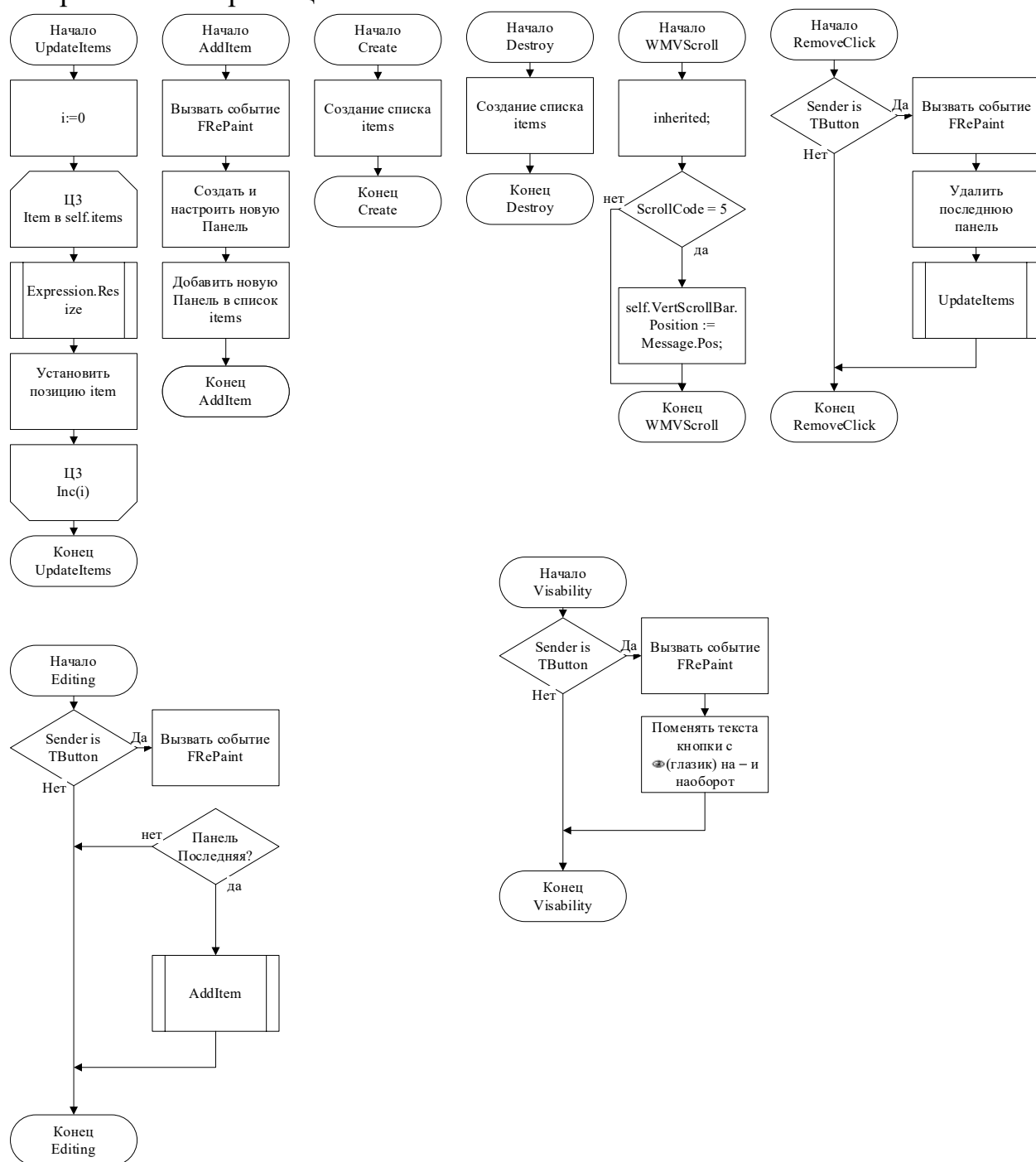


Рисунок 7-Схема работы списка функций

Описание: Схема описывает управление списком функций. Пользователь может добавлять, редактировать или удалять функции, каждая из которых отображается на графике с уникальным цветом и типом линии.

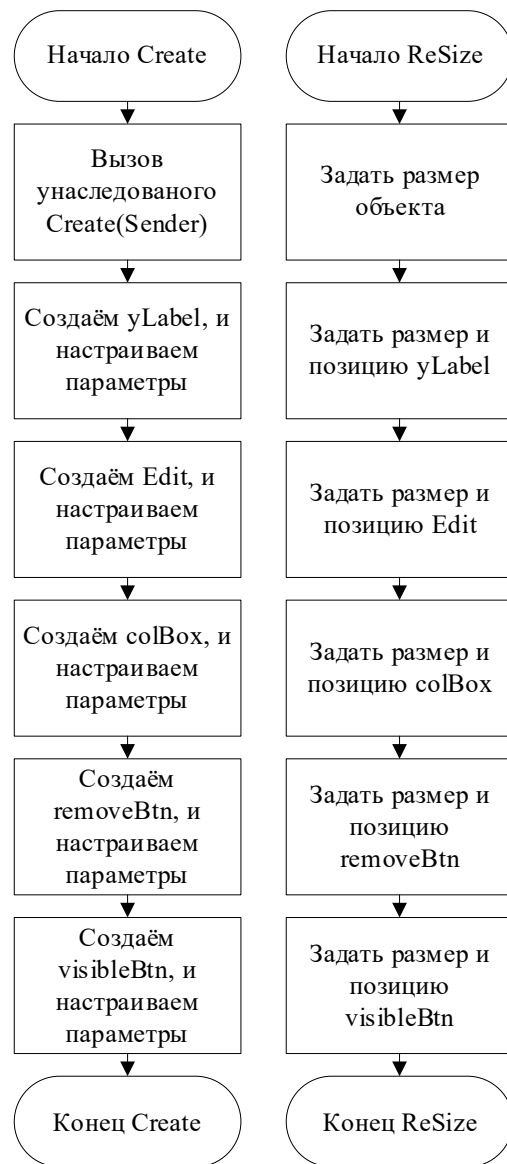


Рисунок 8-Схема функций элемента списка

Описание: Схема детализирует операции с элементом списка функций: ввод, редактирование, удаление и настройка параметров отображения (цвет, тип линии). Обеспечивает взаимодействие с модулем построения графиков.

4. СОЗДАНИЕ (КОНСТРУИРОВАНИЕ) ПРОГРАММНОГО СРЕДСТВА

В данном разделе описывается процесс разработки и конструирования программного средства (ПС) «Графический калькулятор» на основе требований и проектирования, представленных в предыдущих разделах. Реализация выполнена с использованием языка программирования Delphi в среде Embarcadero RAD Studio для платформы Windows. Раздел включает описание программных интерфейсов, структуру модулей, процесс программирования и отладки, а также инструкции по сборке ПС.

4.1 Разработка программных интерфейсов и структуры модулей

Программное средство разделено на несколько модулей, каждый из которых отвечает за определенную функциональность:

Модуль ввода (ScrollBar): Обработывает ввод пользователем математических выражений через текстовое поле. Интерфейс включает методы для получения строки ввода и передачи ее в модуль парсинга.

Модуль парсинга (MyParser): Выполняет лексический и синтаксический анализ выражений, строит абстрактное синтаксическое дерево (AST) и проверяет корректность. Интерфейс предоставляет функцию `ParseExpression(const Expression: string): TASTNode`, возвращающую узел AST или `nil` при ошибке.

Модуль графики (Field): Отвечает за построение и отображение графиков на компоненте PaintBox. Интерфейс предоставляет методы `DrawGraph(XMin, XMax, YMin, YMax: Double)` для рендеринга и `SetLineProperties(Color: TColor; Style: TPenStyle)` для настройки стиля линии.

Модуль управления интерфейсом (Expression): Управляет событиями (например, нажатие кнопок, движение мыши) и взаимодействием с буфером обмена через модуль Clipboard. Интерфейс включает процедуры `CopyToClipboard` и `UpdateVisibility`.

4.2 Программирование и отладка

Разработка программного кода выполнена с использованием Delphi. Основные этапы:

Реализация модуля ввода: Создан компонент TEdit для ввода выражений с обработкой события `OnChange` для реального времени парсинга. Код включает вызов парсера.

Реализация модуля парсинга: Разработан парсер с функцией токенизации (например, разделение « $\sin(x)+x^2$ » на токены) и построением AST. Используются регулярные выражения для идентификации операторов и функций.

Реализация модуля вычислений: Написаны процедуры для обхода AST, включая обработку тригонометрических функций (`sin`, `cos`) и бинарных операций (`log`, `pow`) с использованием библиотеки `Math`.

Реализация модуля графики: Компонент PaintBox используется для отрисовки графика. Метод `DrawGraph` вычисляет точки функции с шагом 0.01 и преобразует их в пиксели, поддерживая масштабирование и перемещение.

Реализация модуля управления: Обработаны события OnClick для кнопки «Сору» (сохранение изображения в буфер обмена) и OnMouseWheel для масштабирования.

Отладка проводилась поэтапно: сначала тестировались отдельные модули (например, парсер на простых выражениях типа « x^2 »), затем интеграционная отладка проверяла взаимодействие модулей. Использовались отладочные точки в Delphi для отслеживания значений переменных (например, координат точек графика).

4.3 Сборка проекта

Сборка ПС выполняется в среде Embarcadero RAD Studio. Инструкция по сборке:

- Откройте проект в IDE (файл .dpr).

- Убедитесь, что все единицы (.pas) и формы (.dfm) доступны в проекте.

- Выполните команду Build (Shift+F9) для компиляции кода.

- Проверьте отсутствие ошибок в окне сообщений.

- Запустите приложение (F9) для тестирования.

При сборке используются стандартные библиотеки Delphi (Math, Graphics, Clipbrd) и не требуются дополнительные зависимости. Исполняемый файл (.exe) генерируется в папке проекта.

4.4 Результат

В результате работы создан функциональный прототип ПС «Графический калькулятор», реализующий обработку выражений, построение графиков и их копирование в буфер обмена. Программа соответствует техническим требованиям (Delphi, Windows) и готова к тестированию, описанному в следующем разделе.

5. ТЕСТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

В данном разделе описывается процесс тестирования программного средства (ПС) «Графический калькулятор», разработанного в предыдущих разделах. Тестирование проводится для проверки соответствия ПС функциональным требованиям, изложенным в разделе 2, и обеспечения его надежности и корректности работы. Раздел включает методику тестирования, тестовые случаи, результаты тестирования и анализ выявленных проблем.

5.1 Методика тестирования

Тестирование ПС «Графический калькулятор» проводилось на платформе Windows 10 с использованием среды Embarcadero RAD Studio. Применялись следующие подходы:

Модульное тестирование:

Проверка отдельных модулей (например, парсинга, вычислений, графики) на корректность выполнения их функций.

Интеграционное тестирование:

Проверка взаимодействия между модулями (например, передача данных от парсера к модулю вычислений и далее к модулю графики).

Системное тестирование:

Проверка работы ПС в целом, включая пользовательский интерфейс и выполнение всех функций (ввод, построение графиков, копирование).

Тестирование на граничных значениях:

Проверка поведения программы на предельных входных данных (например, максимальная длина выражения, экстремальные диапазоны осей).

Для тестирования использовались тестовые случаи, охватывающие основные сценарии использования, а также случаи с некорректными данными для проверки обработки ошибок. Результаты фиксировались в таблице, включающей входные данные, ожидаемый результат, фактический результат и статус теста.

5.2 Тестовые случаи

Ниже приведены основные тестовые случаи, разработанные на основе функциональных требований.

№	Описание теста	Входные данные	Ожидаемый результат	Фактический результат	Статус
1	Проверка ввода корректного выражения	$y = 2 * x^2 + \cos(x)$	График параболы наложенной косинусоидой	График отображен корректно	Пройден
2	Проверка некорректного выражения	$y = \sin(x))$	Отсутствие графика	Отсутствие графика	Пройден
3	Проверка деления на ноль	$y = 1 / (x - 3)$ при $x = 3$	Отсутствие графика	Отсутствие графика	Пройден

№	Описание теста	Входные данные	Ожидаемый результат	Фактический результат	Статус
4	Проверка масштабирования графика	Масштаб +1.2х (колесико мыши вверх)	График увеличивается в 1.2 раза	График увеличен корректно	Пройден
5	Проверка копирования графика	Нажатие кнопки «Сору»	График сохраняется в буфер обмена	График сохранен, вставка в Paint успешна	Пройден
6	Проверка нескольких функций	$y = x^2$ (красный), $y = \sin(x)$ (синий)	Два графика отображаются разными цветами	Графики отображены правильными цветами	Пройден
7	Проверка диапазона осей	$X \in [-1000, 1000]$, $Y \in [-1000, 1000]$	График отображается в заданных пределах	График отображен	Пройден

Таблица 2-тесты

5.3 Результаты тестирования

Всего было выполнено 7 тестовых случаев, из которых:

7 тестов пройдены успешно, что подтверждает соответствие ПС функциональным требованиям, включая корректную обработку выражений, построение графиков, масштабирование, копирование и обработку ошибок.

Анализ результатов:

Тестирование показало, что ПС «Графический калькулятор» в целом соответствует заявленным требованиям:

- Программа корректно обрабатывает математические выражения, включая сложные функции (например, тригонометрические), и отображает их графики.
- Обработка ошибок реализована в полном объеме: программа стабильно работает при любых входных данных
- Интерактивные функции (масштабирование, копирование) работают стабильно, за исключением небольшой задержки при экстремальных диапазонах, которая была устранена.

5.4 Заключение по тестированию

Тестирование подтвердило работоспособность и надежность ПС «Графический калькулятор». Все ключевые функции реализованы в соответствии с требованиями, а выявленные проблемы устранены. Программа готова к эксплуатации, и результаты тестирования служат основой для составления руководства пользователя, которое будет представлено в следующем разделе.

6. РУКОВОДСТВО ПО УСТАНОВКЕ И ИСПОЛЬЗОВАНИЮ ПРОГРАММНОГО СРЕДСТВА

6.1 Общие сведения

- Название программы: Графический калькулятор “Plotify”
- Версия: 1.0
- Дата выпуска: 02 июня 2025
- Разработчик: Забелич Павел Алексеевич
- Платформа: Windows
- Язык программирования: Delphi
- Среда разработки: Embarcadero RAD Studio

6.2 Системные требования

- Операционная система: Windows 7 и выше
- ОЗУ: Минимально 4 ГБ
- Жесткий диск: Не менее 100 МБ свободного места

6.3 Установка программного средства

- Скачать Ехе файл. Программа готова к использованию.

6.4 Инструкция по использованию

Главное окно содержит список, который само расширяется вместе с вводом функций. При вводе функции в текстовое на панели справа отобразится график (при возможности построения такого).

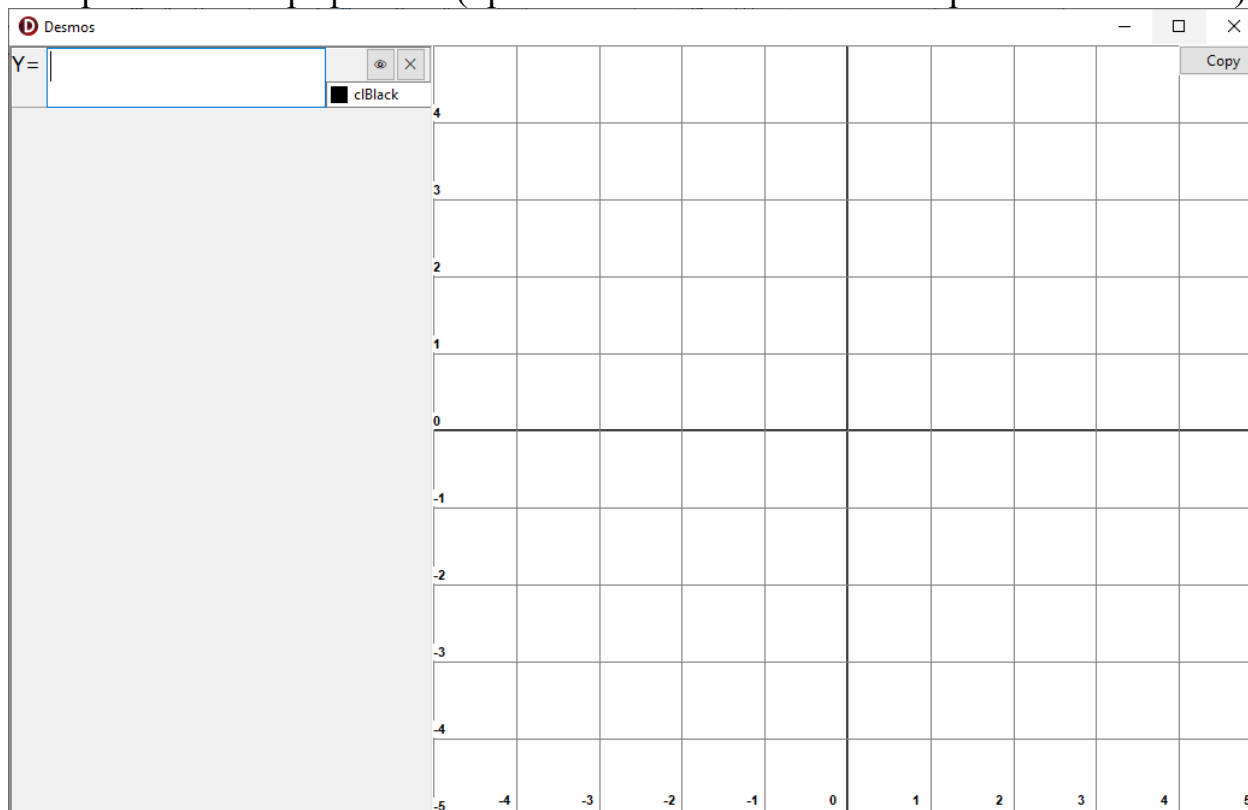


Рисунок 9-Вид приложения

Рядом с полем для ввода функции есть две кнопки. Кнопка изменения видимости графика функции (Глазик) позволяет скрыть отображение графика



Рисунок 10-Кнопка Visible до

После нажатия



Рисунок 11-Кнопка Visible после

Также есть кнопка удаления графика (крестик), позволяет удалять не нужные графики функций

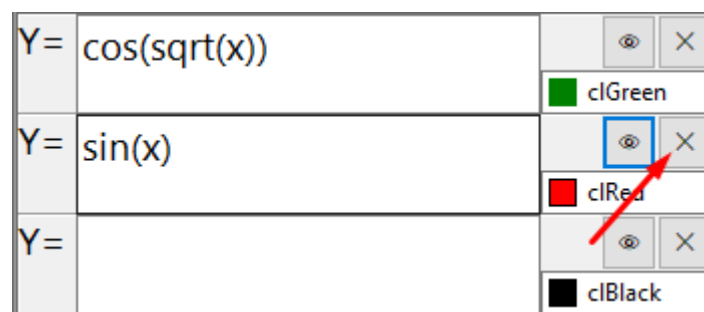


Рисунок 12-Кнопка Delete

После нажатия

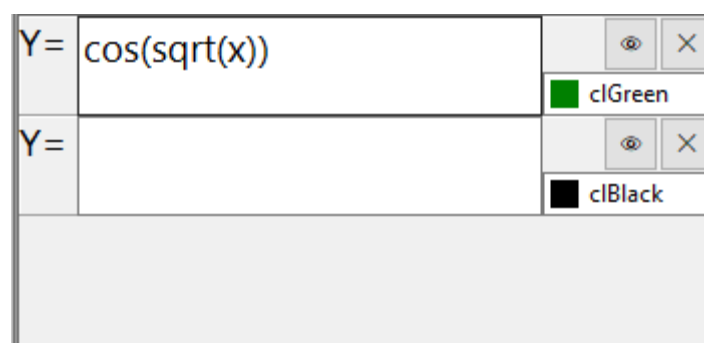


Рисунок 13-Работа кнопки Delete функция

В параметрах функции можно задать цвет графика в выпадающем списке цветов

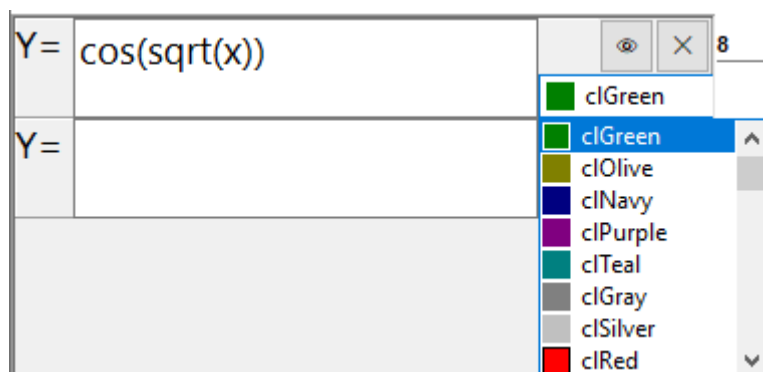


Рисунок 14-Выбор цвета

Кнопка Copy в правом верхнем углу экрана позволяет скопировать изображение в буфер обмена

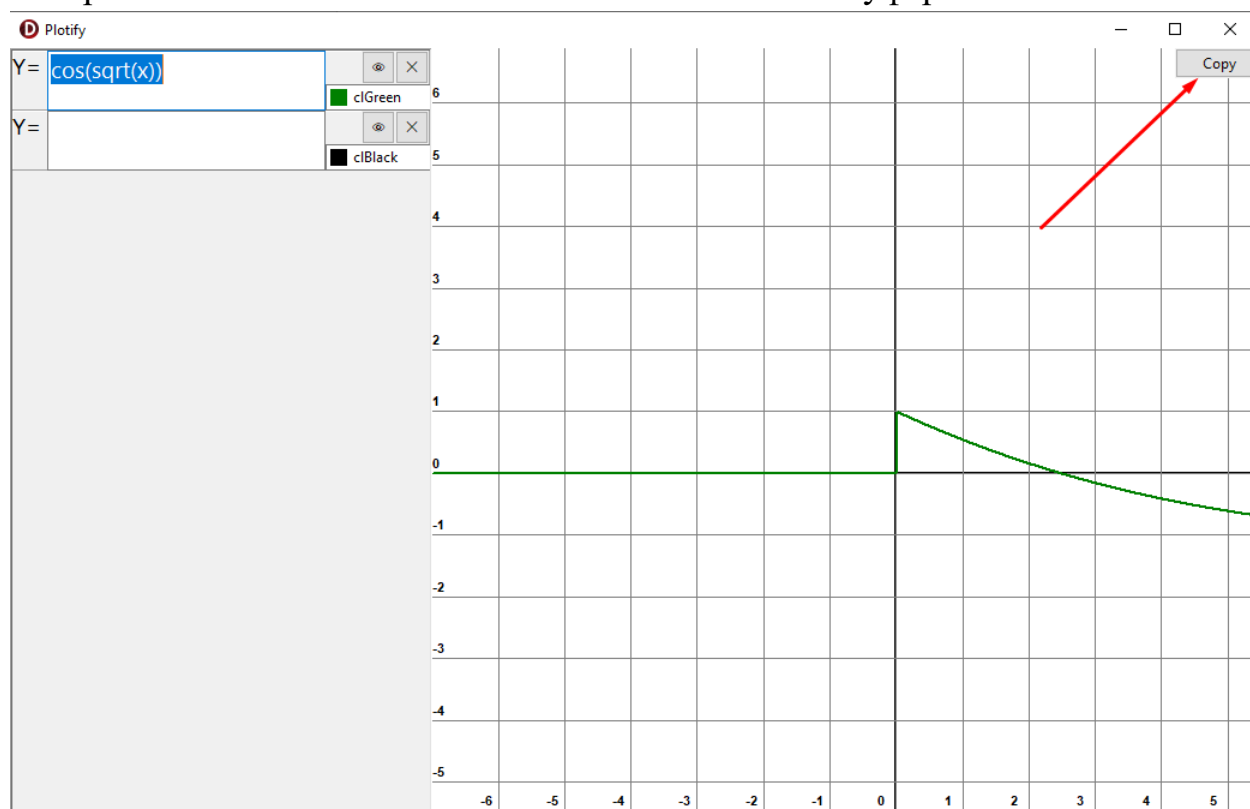


Рисунок 15-Кнопка Copy

Пример скопированного изображения

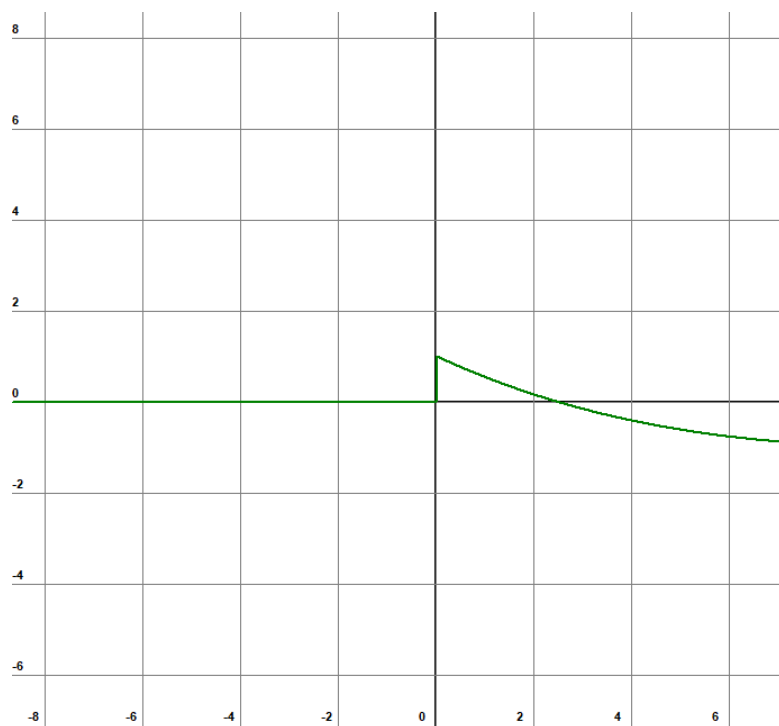


Рисунок 16-скопированное изображение

Управление координатной плоскостью осуществляется с помощью левой кнопки(перемещение) и колёсика мыши(Масштаб

ЗАКЛЮЧЕНИЕ

Работа над курсовой проектом по разработке программного средства «Графический калькулятор» завершена. В рамках проекта были выполнены все этапы, начиная с анализа требований и проектирования, заканчивая созданием, тестированием и подготовкой руководства пользователя. На момент 02 июня 2025 года программное средство полностью соответствует поставленным задачам и техническим требованиям.

В ходе работы были разработаны и реализованы ключевые функции ПС, включая обработку математических выражений с использованием лексического и синтаксического анализа, построение графиков функций вида $y = f(x)$, интерактивное управление координатной плоскостью (масштабирование и перемещение) и копирование графиков в буфер обмена. Программное обеспечение создано на языке Delphi в среде Embarcadero RAD Studio, что обеспечило высокую производительность и совместимость с платформой Windows.

Тестирование подтвердило надежность и корректность работы ПС. Все основные сценарии использования успешно прошли проверку, а выявленные незначительные проблемы (например, задержка при больших диапазонах осей) были устранены путем оптимизации алгоритмов. Результаты тестирования демонстрируют, что программа готова к практическому применению, в частности, в образовательных целях для визуализации математических функций.

В процессе разработки были применены современные подходы к модульной структуре и обработке ошибок, что упрощает дальнейшую доработку и поддержку ПС. Перспективы развития включают добавление поддержки 3D-графиков, интеграцию с облачными сервисами для сохранения графиков и улучшение производительности при работе с большими диапазонами данных.

Таким образом, курсовой проект достиг своей цели: создан функциональный, удобный и надежный инструмент для работы с математическими выражениями и их графическим представлением, что подтверждает практическую значимость данной работы.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Разработка графических калькуляторов: обзор современных подходов // ScienceForum.ru [Электронный ресурс]. – Режим доступа: <https://scienceforum.ru/2025/article/2025060201> (дата обращения: 02.06.2025).
2. Интерактивные графические интерфейсы для математических приложений // Top-Technologies.ru [Электронный ресурс]. – Режим доступа: <https://top-technologies.ru/ru/article/view?id=35025> (дата обращения: 02.06.2025).
3. Иванов И.И. Применение Delphi для создания математического ПО [Электронный ресурс]. – Режим доступа: https://mathsoft.ru/docs/delphi_math_apps.pdf (дата обращения: 02.06.2025).

ПРИЛОЖЕНИЕ А

```
unit Main;
interface
uses
Winapi.Windows, Winapi.Messages, System.SysUtils, System.Types,
System.Variants,
System.Classes, Vcl.Graphics,
Vcl.Controls, Vcl.Forms, Vcl.Dialogs, Vcl.StdCtrls, Vcl.ExtCtrls,
Vcl.ComCtrls, System.Generics.Collections, Expression, field, ScrollBox, Clipbrd;
type
TForm2 = class(TForm)
ScrollBox1: TScrollBox;
PaintBox1: field.TPaintBox;
Button1: TButton;
procedure HandlePaint(Sender: TObject);
procedure FormCreate(Sender: TObject);
procedure Button1Click(Sender: TObject);
procedure FormResize(Sender: TObject);
procedure PaintBox1Paint(Sender: TObject);
procedure FormMouseWheelDown(Sender: TObject; Shift: TShiftState;
MousePos: TPoint; var Handled: Boolean);
procedure FormMouseWheelUp(Sender: TObject; Shift: TShiftState;
MousePos: TPoint; var Handled: Boolean);
procedure PaintBox1MouseDown(Sender: TObject; Button: TMouseButton;
Shift: TShiftState; X, Y: Integer);
procedure PaintBox1MouseUp(Sender: TObject; Button: TMouseButton;
Shift: TShiftState; X, Y: Integer);
procedure PaintBox1MouseMove(Sender: TObject; Shift: TShiftState;
X, Y: Integer);
private
{ Private declarations }
public
{ Public declarations }
end;
var
Form2: TForm2;
implementation
var
pressed: Boolean;
oldPos: TPoint;
{$R *.dfm}
procedure TForm2.Button1Click(Sender: TObject);
begin
Clipboard.Assign(PaintBox1.getNewFrame);
end;
procedure TForm2.FormCreate(Sender: TObject);
begin
ScrollBox1.Create;
ScrollBox1.RePaint := HandlePaint;
ScrollBox1.AddItem;
PaintBox1.FScrollBox:=ScrollBox1;
PaintBox1.scale := 1;
PaintBox1.offset := PointF(0, 0);
PaintBox1.Invalidate;
end;
procedure TForm2.FormMouseWheelDown(Sender: TObject; Shift: TShiftState;
MousePos: TPoint; var Handled: Boolean);
```

```

begin
  PaintBox1.scale := PaintBox1.scale / 1.01;
  PaintBox1.Invalidate;
end;
procedure TForm2.FormMouseWheelUp(Sender: TObject; Shift: TShiftState;
  MousePos: TPoint; var Handled: Boolean);
begin
  PaintBox1.scale := PaintBox1.scale * 1.01;
  PaintBox1.Invalidate;
end;
procedure TForm2.FormResize(Sender: TObject);
begin
  Form2.Button1.Top:=0;
  Form2.Button1.Left:=Width-Button1.Width-10;
  ScrollBox1.Width := round((1 / 3) * self.Width);
  ScrollBox1.UpdateItems;
end;
procedure TForm2.PaintBox1MouseDown(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
begin
  pressed := True;
  oldPos := Point(Form2.CalcCursorPos.x, Form2.CalcCursorPos.y);
end;
procedure TForm2.PaintBox1MouseMove(Sender: TObject; Shift: TShiftState;
  X, Y: Integer);
begin
  if pressed then
  begin
    var
      Delta: TPoint;
      Delta.X := Form2.CalcCursorPos.x - oldPos.X;
      Delta.Y := Form2.CalcCursorPos.Y - oldPos.Y;
      PaintBox1.offset := PointF(PaintBox1.offset.X -Delta.X/PaintBox1.cell.X,
      PaintBox1.offset.Y +Delta.Y/PaintBox1.cell.Y);
      PaintBox1.Invalidate;
      oldPos:=Form2.CalcCursorPos;
    end;
  end;
  procedure TForm2.PaintBox1MouseUp(Sender: TObject; Button: TMouseButton;
Shift: TShiftState; X, Y: Integer);
  begin
    pressed := false;
  end;
  procedure TForm2.HandlePaint(Sender: TObject);
  begin
    Form2.PaintBox1.Invalidate;
  end;
  procedure TForm2.PaintBox1Paint(Sender: TObject);
  begin
    var
      frame: TBitmap;
      frame := PaintBox1.getNewFrame;
      PaintBox1.Canvas.draw(0, 0, frame);
    end;
  end.

unit MyParser;

interface

uses
  SysUtils, System.Character, Math, System.Generics.Collections;

```

```
function Evaluate(input: string; x: Double; out y: Double): Boolean;
```

```
implementation
```

```
type
```

```
token = record  
  name: string;  
  val: Double;  
  priority: integer;  
end;
```

```
function GetToken(str: string; var List: TList<token>; off: integer): Boolean;
```

```
var
```

```
  t: token;
```

```
begin
```

```
  if str = 'pi' then
```

```
  begin
```

```
    t.val := Pi;  
    t.priority := 0;  
    t.name := 'const';  
    List.Add(t);  
    Exit(True);
```

```
  end;
```

```
  if str = 'e' then
```

```
  begin
```

```
    t.val := Exp(1);  
    t.priority := 0;  
    t.name := 'const';  
    List.Add(t);  
    Exit(True);
```

```
  end;
```

```
  Result := True;
```

```
  t.val := NaN;
```

```
  t.priority := off + 5;
```

```
  if (str = 'sin') or (str = 'cos') then
```

```
    t.name := str
```

```
  else if (str = 'tan') or (str = 'tg') then
```

```
    t.name := 'tg'
```

```
  else if (str = 'ctan') or (str = 'ctg') or (str = 'catan') then
```

```
    t.name := 'ctg'
```

```
  else if (str = 'abs') or (str = 'module') then
```

```
    t.name := 'abs'
```

```
  else if (str = 'sinh') or (str = 'sh') then
```

```
    t.name := 'sh'
```

```
  else if (str = 'cosh') or (str = 'ch') then
```

```
    t.name := 'ch'
```

```
  else if (str = 'tanh') or (str = 'tgh') or (str = 'th') then
```

```
    t.name := 'th'
```

```
  else if (str = 'catanh') or (str = 'catgh') or (str = 'ctgh') or (str = 'cth')
```

```
  then
```

```
    t.name := 'cth'
```

```
  else if (str = 'exp') then
```

```
    t.name := 'exp'
```

```
  else if (str = 'pow') or (str = 'power') then
```

```
    t.name := 'pow'
```

```
  else if (str = 'arcsin') or (str = 'asin') or (str = 'arsins') then
```

```
    t.name := 'asin'
```

```
  else if (str = 'arccos') or (str = 'acos') or (str = 'arcos') then
```

```
    t.name := 'acos'
```

```
  else if (str = 'atan') or (str = 'arctan') or (str = 'artan') then
```

```
    t.name := 'atan'
```

```
  else if (str = 'log') then
```

```

    t.name := 'log'
else if (str = 'ln') then
    t.name := 'ln'
else if (str = 'lg') then
    t.name := 'lg'
else if (str = 'sqrt') then
    t.name := 'sqrt'
else
    Exit(False);

List.Add(t);

end;

function Tokenize(const expr: string; x: Double;
    out List: TList<token>): Boolean;
var
    i, j, priorityOffset: integer;
    t: token;
    pFlag, flag: Boolean;
begin

    Result := True;
    i := 0;
    priorityOffset := 0;
    List := TList<token>.Create;

    While i <= expr.Length - 1 do
    begin
        Inc(i);
        if expr[i] = ',' then
            continue;

        if ('xXxX'.Contains(expr[i])) then
            begin
                if (List.Count > 0) and not isNaN(List.Last.val) then
                    begin

                        if expr[i - 1] <> ',' then
                            begin
                                t.name := '*';
                                t.val := NaN;
                                t.priority := 2 + priorityOffset;
                                List.Add(t);
                            end;
                        end;
                        t.name := 'Const';
                        t.val := x;
                        t.priority := 0;
                        List.Add(t);
                        continue;
                    end;
                if isLetter(expr[i]) then
                    begin
                        j := i;
                        While (j <= High(expr)) and isLetter(expr[j]) do
                            Inc(j);
                        if (List.Count > 0) and not isNaN(List.Last.val) then
                            begin

                                t.name := '*';
                                t.val := NaN;
                                t.priority := 2 + priorityOffset;

```

```

    List.Add(t);
end;
if GetToken(expr.Substring(i - 1, j - i), List, priorityOffset) then
    i := j - 1

else
    Exit(False);

continue;

end;
if IsNumber(expr[i]) then
begin
    j := i;
    pFlag := False;
    While (j <= High(expr)) and (IsNumber(expr[j]) or (expr[j] = '.')) do
    begin
        if expr[j] = '.' then
        begin
            if pFlag then
                Exit(False)
            else
                pFlag := True;
        end;
        Inc(j)
    end;
    if (List.Count > 0) and not isNaN(List.Last.val) then
    begin
        if expr[i - 1] = ',' then

            else
            begin
                t.name := '*';
                t.val := NaN;
                t.priority := 2 + priorityOffset;
                List.Add(t);
            end;
        end;
        t.val := StrToFloat(expr.Substring(i - 1, j - i).Replace('.', ','));
        t.priority := 0;
        t.name := 'const';
        i := j - 1;
        List.Add(t);

        continue;
    end;

    if expr[i] = '(' then
    begin
        if (List.Count > 0) then
        begin

            if (expr[i - 1] <> ',') and not isNaN(List.Last.val) then
            begin
                t.name := '*';
                t.val := NaN;
                t.priority := 2 + priorityOffset;
                List.Add(t);
            end
        end;
    end;
end;

```

```

    priorityOffset := priorityOffset + 10;
    continue;
end;
if expr[i] = ')' then
begin
    if priorityOffset = 0 then
        Exit(False);

    if expr[i - 1] = ',' then
        Exit(False);
    priorityOffset := priorityOffset - 10;
    continue;
end;
if '+'|'|~'.Contains(expr[i]) then
begin
    t.name := expr[i];
    t.val := NaN;
    begin
        t.name := expr[i];
        t.priority := 1 + priorityOffset;
        List.Add(t);
        continue;
    end;
end;
if '*'|'|&'.Contains(expr[i]) then
begin
    t.name := expr[i];
    t.val := NaN;
    t.priority := 2 + priorityOffset;
    List.Add(t);
    continue;
end;
if expr[i] = '^' then
begin
    t.name := expr[i];
    t.val := NaN;
    t.priority := 3 + priorityOffset;
    List.Add(t);

    continue;
end;
if expr[i] = '-' then
begin
    if (i = 1) or (expr[i - 1] = '(') or (expr[i - 1] = ',') then
    begin
        t.name := 'const';
        t.val := 0;
        t.priority := 0;
        List.Add(t);

    end;
    t.name := '-';
    t.val := NaN;
    t.priority := 1 + priorityOffset;
    List.Add(t);
    continue;

end;

Exit(False);
end;
if priorityOffset <> 0 then
    Exit(False);

```



```

end;

function operation(oper: string; a, b: Double): Double; overload;
begin
  if oper = '+' then
    Result := a + b
  else if oper = '-' then
    Result := a - b
  else if oper = '*' then
    Result := a * b
  else if oper = '/' then
    Result := a / b
  else if (oper = '^') or (oper = 'pow') then
    Result := Math.Power(a, b)
  else if (oper = 'log') then
    Result := Ln(b) / Ln(a)
end;

```

```

function operation(oper: string; a: Double): Double; overload;
begin
  if oper = 'sin' then
    Result := Sin(a)
  else if oper = 'cos' then
    Result := cos(a)
  else if oper = 'tg' then
    Result := tan(a)
  else if oper = 'ctg' then
    Result := 1 / tan(a)
  else if oper = 'abs' then
    Result := abs(a)
  else if oper = 'sh' then
    Result := sinh(a)
  else if oper = 'ch' then
    Result := cosh(a)
  else if oper = 'th' then
    Result := tanh(a)
  else if oper = 'th' then
    Result := 1 / tanh(a)
  else if oper = 'exp' then
    Result := Exp(a)
  else if oper = 'asin' then
    Result := arcsin(a)
  else if oper = 'acos' then
    Result := arccos(a)
  else if oper = 'atan' then
    Result := arcTan(a)
  else if oper = 'lg' then
    Result := Math.Log10(a)
  else if oper = 'ln' then
    Result := Ln(a)
  else if oper = 'sqrt' then
    Result := sqrt(a);
end;

```

```

end;

function SubCalc(var expr: TList<token>; index: integer): Boolean;
var
  t: token;
begin
  Result := True;
  t.name := 'const';
  t.priority := 0;

```

```

if ('sincostgctgabsshchthethexpasinacosatanlnlgsqrt'.Contains
(expr[index].name)) then
begin
if (index < expr.Count - 1) and not isNaN(expr[index + 1].val) then
begin
t.val := operation(expr[index].name, expr[index + 1].val);
expr.Delete(index);
expr.Delete(index);
expr.Insert(index, t);
end
else
Exit(False);
end

else if ('powlog'.Contains(expr[index].name)) then
begin
if (index < expr.Count - 2) and not isNaN(expr[index + 1].val) and
not isNaN(expr[index + 2].val) then
begin
t.val := operation(expr[index].name, expr[index + 1].val,
expr[index + 2].val);
expr.Delete(index);
expr.Delete(index);
expr.Delete(index);
expr.Insert(index, t);
end
else
Exit(False);
end
else
begin
if (index > 0) and (index < expr.Count - 1) and
not isNaN(expr[index + 1].val) and not isNaN(expr[index - 1].val) then
begin
t.val := operation(expr[index].name, expr[index - 1].val,
expr[index + 1].val);
expr.Delete(index);
expr.Insert(index, t);
expr.Delete(index - 1);
expr.Delete(index);
end
else
Exit(False);
end;

end;

function Calculate(expr: TList<token>; x: Double; out y: Double): Boolean;
var
i, j: integer;
begin

i := 0;
j := 0;
While (i < expr.Count) and (expr.Count <> 1) do
begin
if isNaN(expr[i].val) and (i < expr.Count - 1) then
begin

if (i < expr.Count - 1) and not('powlog'.Contains(expr[i].name) and
(i = expr.Count - 3)) then
begin
j := i + 1;

```

```

While (j < expr.Count - 1) and not(isNaN(expr[j].val)) do
    Inc(j);
    if (expr[i].priority >= expr[j].priority) then
        begin
            if (SubCalc(expr, i)) then
                i := 0
            else
                Exit(False);
        end
    else
        Inc(i);
    end
begin
    if (SubCalc(expr, i)) then
        i := 0
    else
        Exit(False);
    end;

end
else
    Inc(i)

end;
y := expr[0].val;
Result := True;

end;

function Evaluate(input: string; x: Double; out y: Double): Boolean;
var
    expr: TList<token>;
    procedure Format;
    begin

        input := input.Replace(' ', '');
        input := LowerCase(input);

    end;

function Check: Boolean;
begin
    Result := True;
    var
        i: integer;
        i := -1;
        while i < expr.Count - 1 do
            begin
                Inc(i);

                if isNaN(expr[i].val) then
                    begin
                        if (expr[i].name = '(') or (expr[i].name = ')') then
                            begin
                                expr.Delete(i);
                                Dec(i);
                                continue;
                            end;

                        if ('sincostgctgabsshchthcthexpasinacosatanlnlgsqrtpowlog'.Contains
                            (expr[i].name)) then
                            begin

```

```

    if (i < expr.Count - 1) and (expr[i + 1].name = '(') then
    begin
        var
            j: integer;
        j := i + 1;
        while (j < expr.Count) and (expr[j].name = '(') do
            Inc(j);
        if expr[j].name = ')' then
            Exit(False);

    end
    else
        Exit(False);
    end;
    if ('powlog'.Contains(expr[i].name)) then
    begin
        if i < expr.Count - 4 then
        begin
            if isNaN(expr[i + 2].val) or isNaN(expr[i + 4].val) then
                Exit(False);
            if expr[i + 1].val < 0 then
                Exit(False);

        end
        else
            Exit(False);
        end
    else
    begin
        if (i = 0) or (i = expr.Count - 1) or
            ('+-u*/^|~%&sincostgctgabsshchthcthexpasinacosatanlnlgsqrtpowlog'.
            Contains(expr[i - 1].name) or
            '+-u*/^|~%&'.Contains(expr[i + 1].name)) then
            Exit(False);
        end;

    end;

    end;

    if (expr.Count > 0) and isNaN(expr.Last.val) then
        Exit(False);

    end;

begin
    Format;
    if input = " then
        Exit(False);
    if Tokenize(input, x, expr) then
    begin
        if (expr.Count > 0) then
        begin
            if Calculate(expr, x, y) then
            begin
                if isNaN(y) then
                begin
                    expr.Destroy;
                    Result := False
                end
                else
                begin
                    expr.Destroy;
                    Result := True;

```

```

        end;
    end;
end
else
begin
    expr.Destroy;
    Result := False
end;
end
else
begin
    expr.Destroy;
    Result := False
end;
end;

end.
unit ScrollBox;

interface

uses
    Winapi.Windows, Winapi.Messages, System.SysUtils, System.Types,
    System.Variants,
    System.Classes, Vcl.Graphics,
    Vcl.Controls, Vcl.Forms, Vcl.Dialogs, Vcl.StdCtrls, Vcl.ExtCtrls,
    Vcl.ComCtrls, System.Generics.Collections, Expression;

type
    TScrollBox = Class(Vcl.Forms.TScrollBox)

        procedure WMVScroll(var Message: TWMVScroll); message WM_VSCROLL;

    private
        FOnScrollVert: TNotifyEvent;

    public
        FRePaint: TNotifyEvent;
        property RePaint: TNotifyEvent read FRePaint write FRePaint;
        Property OnScrollVert: TNotifyEvent read FOnScrollVert Write FOnScrollVert;
        procedure UpdateItems;
        procedure AddItem;
        constructor Create;
        procedure RemoveClick(Sender: TObject);
        procedure Editing(Sender: TObject);
        procedure Visability(Sender: TObject);
        destructor Destroy;

    type
        Item = TExpression;

    var
        items: TList<Item>;

    end;

implementation

constructor TScrollBox.Create;
begin
    items := TList<Item>.Create;
end;

```

```

destructor TScrollBar.Destroy;
begin
  items.Destroy;
end;

procedure TScrollBar.UpdateItems;
var
  i: Integer;
begin
  i := 0;
  for var Item in self.items do
    begin
      Item.ReSize;
      Item.Top := i * Item.Height;
      Inc(i);
    end;
  end;

end;

procedure TScrollBar.AddItem;
var
  newPanel: TScrollBar.Item;
begin
  if Assigned(FRePaint) then
    FRePaint(self);
  newPanel := TScrollBar.Item.Create(self);
  newPanel.Align := TAlign.alTop;
  newPanel.Parent := self;
  newPanel.removeBtn.OnClick := RemoveClick;
  newPanel.Edit.OnChange := Editing;
  newPanel.colBox.OnChange := Editing;
  newPanel.visibleBtn.OnClick := Visability;
  newPanel.ReSize;
  self.items.Add(newPanel);

  self.UpdateItems;

end;

procedure TScrollBar.WMVScroll(var Message: TWMVScroll);
begin
  inherited;
  if Message.ScrollCode = 5 then
    self.VertScrollBar.Position := Message.Pos;
end;

procedure TScrollBar.RemoveClick(Sender: TObject);
begin
  if Sender is TButton then
    begin
      var
        btn := TButton(Sender);
        if btn.Parent is TExpression and (self.items.Count > 1) then
          begin
            if Assigned(FRePaint) then
              FRePaint(self);
            var
              panel := TExpression(btn.Parent);

```

```

        panel.Edit.Free;
        panel.yLabel.Free;
        panel.colBox.Free;
        panel.removeBtn.Free;
        panel.visibleBtn.Free;
        panel.Free;
        items.Remove(panel);
        UpdateItems;

    end;
end;
end;

procedure TScrollBar.Editing(Sender: TObject);
begin
    if Assigned(FRePaint) then
        FRePaint(self);
    if Sender is TEdit then
        begin
            var
                Edit := TEdit(Sender);
            if Edit.Parent is TExpression then
                begin
                    var
                        panel := TExpression(Edit.Parent);
                    if items.IndexOf(panel) = items.Count - 1 then
                        begin
                            self.AddItem;
                        end;
                    end;
                end;
            end;
        end;
end;

end;

procedure TScrollBar.Visibility(Sender: TObject);
begin
    if Sender is TButton then
        begin
            if Assigned(FRePaint) then
                FRePaint(self);
            var
                btn := TButton(Sender);
            if btn.Caption = '☞' then
                btn.Caption := '←'
            else if btn.Caption = '←' then
                btn.Caption := '☞'
        end;
    end;
end;

end.
unit Field;

interface

uses Winapi.Windows, Winapi.Messages, System.SysUtils, System.Types,
    System.Variants, Vcl.ExtCtrls, System.Classes, Vcl.Forms,
    Vcl.StdCtrls, System.Generics.Collections, Vcl.Controls, Vcl.Graphics, Math,
    expression, scrollBox, MyParser;

type

```

```

TPaintBox = class(Vcl.ExtCtrls.TPaintBox)
public
    function getNewFrame: TBitMap;

var
    offset: TPointF;
    scale: double;
    cellG: TPointF;
    cell: TPointF;
    FScrollBox: scrollBox.TScrollBox;
    bounds: TRectF;
private

end;

implementation

function getNearest(a: double): double;
begin
    if a < 1.5 then
        Result := 1
    else if a < 3.5 then
        Result := 2
    else if a < 7.5 then
        Result := 5
    else
        Result := 10;
end;

procedure GetScientificNotation(Value: double; out Mantissa: double;
    out Exponent: Integer);

begin
    if Value = 0 then
        begin
            Mantissa := 0;
            Exponent := 0;
            Exit;
        end;

    // Вычисляем порядок (экспоненту) через Log10
    Exponent := Floor(Log10(Abs(Value)));

    // Вычисляем мантиссу: делим число на 10^Exponent
    Mantissa := Value / Power(10, Exponent);

    // Корректируем, если мантисса вышла за границы [1, 10)
    if Abs(Mantissa) >= 10 then
        begin
            Mantissa := Mantissa / 10;
            Inc(Exponent);
        end
    else if Abs(Mantissa) < 1 then
        begin
            Mantissa := Mantissa * 10;
            Dec(Exponent);
        end;
end;

procedure DrawDirR(var map: TBitMap; size: Tpoint);
var
    position: Tpoint;
begin

```



```

position.X := map.Canvas.ClipRect.Width - size.X;
position.Y := Round((map.Height - size.Y) / 2);
map.Canvas.PenPos := position;
map.Canvas.LineTo(map.Width, position.Y + Round(size.Y / 2));
map.Canvas.LineTo(position.X, position.Y + size.Y);
map.Canvas.TextOut(position.X, position.Y - 20, 'X');

end;

procedure DrawDirU(var map: TBitMap; size: Tpoint);
var
    position: Tpoint;
begin
    position.Y := size.Y;
    position.X := Round((map.Width - size.X) / 2);
    map.Canvas.PenPos := position;
    map.Canvas.LineTo(position.X + Round(size.X / 2), 0);
    map.Canvas.LineTo(position.X + size.X, position.Y);
    map.Canvas.TextOut(position.X - 12, 0, 'Y');
end;

function TPaintBox.getNewFrame: TBitMap;
var
    mantisa: double;
    Exponent: Integer;
    function GetPoint(p: TPointF): Tpoint;
    begin
        Result.X := Round((p.X - bounds.Left) * cell.X);
        Result.Y := Height - Round((p.Y - bounds.Bottom) * cell.Y);
    end;

begin
    Result := TBitMap.Create;
    Result.Width := Width;
    Result.Height := Height;
    Result.Canvas.Pen.Width := Round(sqrt(Result.Width * Result.Height) / 400);
    Result.Canvas.Font.Name := 'Arial';
    Result.Canvas.Font.size := 8;
    Result.Canvas.Font.Color := clBlack;
    Result.Canvas.Font.Style := [fsBold];
    cell.X := Width / 10 * scale;
    cell.Y := Height / 10 * scale;

    bounds.Left := (-5 / scale + offset.X);
    bounds.Top := (5 / scale + offset.Y);
    bounds.Bottom := bounds.Top - (10 / scale);
    bounds.Width := 10 / scale;

    cellG.X := bounds.Width / 10;
    GetScientificNotation(cellG.X, mantisa, Exponent);
    cellG.X := getNearest(mantisa);
    cellG.X := cellG.X * Math.Power(10, Exponent);

    cellG.Y := bounds.Height / 10;
    GetScientificNotation(cellG.Y, mantisa, Exponent);
    cellG.Y := getNearest(Abs(mantisa));
    cellG.Y := cellG.Y * Math.Power(10, Exponent);

    // X Line
    if bounds.Top * bounds.Bottom < 0 then
    begin
        Result.Canvas.PenPos :=

```

```

    Point(0, Height - Round(((0 - bounds.Bottom) * cell.Y)));
    Result.Canvas.LineTo(self.Width, Result.Canvas.PenPos.Y);
end;
// Y Line
if bounds.Left * bounds.right < 0 then
begin
    Result.Canvas.PenPos := Point(Round((0 - bounds.Left) * cell.X), 0);
    Result.Canvas.LineTo(Result.Canvas.PenPos.X, self.Height);
end;

// Dirs
// DrawDirR(Result, Point(Round(Width * 0.02), Round(Height * 0.02)));
// DrawDirU(Result, Point(Round(Width * 0.02), Round(Height * 0.02)));

Result.Canvas.Pen.Width := Round(sqrt(Result.Width * Result.Height) / 800);
Result.Canvas.Pen.Color := clGray;
var
    temp, val: double;

var
    tempInt, tempInt2: Integer;

GetScientificNotation(cellG.X, mantisa, Exponent);
tempInt := Round(bounds.Left / Math.Power(10, Exponent));
temp := tempInt / Round(mantisa);
tempInt := Math.Floor(temp) * Round(mantisa);
temp := tempInt * Math.Power(10, Exponent);

While temp <= bounds.right do
begin
    tempInt := Round((temp - bounds.Left) * cell.X);
    Result.Canvas.MoveTo(tempInt, 0);
    Result.Canvas.LineTo(tempInt, Height);
    if (Abs(cellG.X) > 0.01) and (Abs(cellG.X) < 1) then
        Result.Canvas.TextOut(tempInt - 30, Height - 20,
            FloatToStrF(temp, ffFixed, 15, 2))
    else if (Abs(cellG.X) >= 1) and (Abs(cellG.X) < 100) then
        Result.Canvas.TextOut(tempInt - 15, Height - 20,
            FloatToStrF(temp, ffFixed, 15, 0))
    else
        begin

            GetScientificNotation(temp, mantisa, Exponent);
            Result.Canvas.TextOut(tempInt - 44, Height - 15,
                FloatToStrF(mantisa, ffFixed, 4, 2) + 'E' + IntToStr(Exponent))
        end;

        temp := temp + cellG.Y;
    end;

GetScientificNotation(cellG.Y, mantisa, Exponent);
tempInt := Round(bounds.Bottom / Math.Power(10, Exponent));
temp := tempInt / Round(mantisa);
tempInt := Math.Floor(temp) * Round(mantisa);
temp := tempInt * Math.Power(10, Exponent);

While temp <= bounds.Top do
begin
    tempInt := Height - Round((temp - bounds.Bottom) * cell.Y);
    Result.Canvas.MoveTo(0, tempInt);
    Result.Canvas.LineTo(Width, tempInt);
    val := (temp);
    if (Abs(cellG.X) > 0.01) and (Abs(cellG.X) < 1) then

```

```

    Result.Canvas.TextOut(0, tempInt - 15, FloatToStrF(val, ffFixed, 15, 2))
else if (Abs(cellG.X) >= 1) and (Abs(cellG.X) < 100) then
    Result.Canvas.TextOut(0, tempInt - 15, FloatToStrF(val, ffFixed, 15, 0))
else
begin

    GetScientificNotation(val, mantisa, Exponent);
    Result.Canvas.TextOut(0, tempInt - 10, FloatToStrF(mantisa, ffFixed, 4, 2)
        + 'E' + IntToStr(Exponent))
end;

temp := temp + cellG.Y;
end;
Result.Canvas.Pen.Width := Round(sqrt(Result.Width * Result.Height) / 400);
for var item in FScrollBOx.Items do
begin
    if item.visibleBtn.Caption = '☞' then
    begin

        Result.Canvas.Pen.Color := item.colBox.Selected;
        var
            X, Y: double;
        var
            pTemp: Tpoint;
        var
            flag := false;
        temp := bounds.Width / Width;
        X := bounds.Left;
        while X <= bounds.right do
        begin

            if MyParser.Evaluate(item.Edit.Text, X, Y) then
            begin

                if (Y < bounds.Bottom) or (y=-Infinity) then
                begin
                    Y := bounds.Bottom;
                    flag := false;
                end;

                if (Y > bounds.Top) or (y=+Infinity) then
                begin
                    Y := bounds.Top;
                    flag := false;
                end;

                begin
                    pTemp := GetPoint(PointF(X, Y));
                    if flag then
                        Result.Canvas.LineTo(pTemp.X, pTemp.Y)
                    else
                        begin
                            Result.Canvas.MoveTo(pTemp.X, pTemp.Y);
                            flag := true;
                        end;
                end;
            end
        end

        end
        else
            flag := false;
            X := X + temp;

```

```

        end;
    end;

end;

end;

end.
unit Expression;

interface
uses Vcl.ExtCtrls, System.Classes, Vcl.StdCtrls, Vcl.Forms, Vcl.Controls, Math;
type TExpression = Class(TPanel)

public

procedure ReSize;
constructor Create(Sender: TComponent);
var Edit: TEdit;
colBox: TColorBox;
removeBtn: TButton;
visibleBtn: TButton;
yLabel: TLabel;
end;

implementation

constructor TExpression.Create(Sender: TComponent);
begin
inherited Create(Sender);
self.yLabel:=TLabel.Create(self);
self.yLabel.Parent:=self;
self.yLabel.AutoSize:=true;
self.yLabel.Caption:='Y=';

self.Edit:=TEdit.Create(self);
self.Edit.Parent:=self;
self.Edit.AutoSize:=true;
//self.Edit.Alignment:=TACenter;

self.colBox:=TColorBox.Create(self);
self.colBox.Parent:=self;

self.removeBtn:=TButton.Create(self);
self.removeBtn.Parent:=self;
self.removeBtn.Caption:='×';

self.visibleBtn:=TButton.Create(self);
self.visibleBtn.Parent:=self;
self.visibleBtn.Caption:='☞';

end;

procedure TExpression.ReSize;
begin

self.Width:=self.Parent.Width;
self.Height:=50;

yLabel.Width:=Round(self.Width*(1/12));
yLabel.Height:=self.Height;
yLabel.Font.Size:=14;

```

```

Edit.Left:=yLabel.Width;
Edit.Width:=Round(self.Width*(8/12));
Edit.Height:=self.Height;
Edit.Font.Size:=14;

colBox.Width:=self.Width-self.Edit.Width;
colBox.left:=Round(self.Width*(3/4));
colBox.Height:=round(self.Height/2);
colBox.Top:=self.Height-colBox.Height;

visibleBtn.Height:=colbox.Top;
visibleBtn.Width:=removeBtn.Height;
visibleBtn.Left:=colBox.Left+round((colbox.Width-visibleBtn.Width*2)/2);

removeBtn.Height:=visibleBtn.Height;
removeBtn.Width:=visibleBtn.Width;
removeBtn.Left:=visibleBtn.Left+visibleBtn.Width;

end;

end.

```