**Министерство науки и высшего образования РФ**

**Федеральное государственное бюджетное образовательное**

**учреждение высшего образования**

**Московский авиационный институт**

**(национальный исследовательский университет)**

**«МАИ»**

Кафедра 304

Пояснительная записка к курсовой работе

по дисциплине «Конструирование программного обеспечения»

Тема: «Классическое приложение Windows, реализующее векторно-матричные преобразования»

Выполнил студент группы М3О-211Б-22

Пономарев Н. А. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Принял:

Канд. техн. наук, доцент, доц. каф.304

Новиков П.В.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Москва, 2024

Содержание

[Задание 2](#_Toc168248073)

[Теоретическая часть 3](#_Toc168248074)

[Текст программы 16](#_Toc168248075)

[Файл “resource.h”: 16](#_Toc168248076)

[Файл “main\_buttons.h”: 18](#_Toc168248077)

[Файл “utility\_functions.h”: 19](#_Toc168248078)

[Файл “vector\_matrix\_operations.h”: 20](#_Toc168248079)

[Файл “dialog\_procedures.h”: 21](#_Toc168248080)

[Файл “main.cpp”: 21](#_Toc168248081)

[Результаты работы 83](#_Toc168248082)

[Вывод 100](#_Toc168248083)

[Список источников 100](#_Toc168248084)

Задание

Разработка классического приложение Windows, реализующее векторно-матричные преобразования на языке программирования C++ в среде разработки Visual Studio.

Теоретическая часть

Данная программа представляет собой классическое Windows-приложение, реализующее векторно-матричные преобразования на языке программирования C++ в среде разработки Visual Studio. Основные задачи приложения включают выполнение операций над векторами и матрицами, таких как сложение, вычитание, умножение, транспонирование, вычисление определителя и другие.

**Основные компоненты программы:**

1. Среда разработки

Программа разработана в среде Visual Studio, что обеспечивает удобную разработку, отладку и компиляцию приложения.

1. Язык программирования

Программа написана на языке C++, который позволяет эффективно управлять памятью и выполнять высокопроизводительные вычисления, необходимые для операций с матрицами и векторами.

1. Графический интерфейс пользователя (GUI)

Для создания пользовательского интерфейса используется библиотека Windows API, а также GDI+ для работы с графикой.

**Операции с векторами и матрицами:**

* Векторные операции
  1. Сложение векторов
  2. Вычитание векторов
  3. Умножение вектора на скаляр

Где - скаляр.

* 1. Скалярное произведение

Где - размерность векторов.

* 1. Векторное произведение (для 3D)
  2. Длина вектора
* Матричные операции
  1. Сложение матриц
  2. Вычитание матриц
  3. Умножение матриц​
  4. Транспонирование матрицы
  5. Обратная матрица

Где - присоединенная матрица (Матрица из дополнений).

* 1. Умножение матрицы на скаляр
  2. Возведение матрицы в степень
  3. Ранг матрицы

Ранг матрицы определяется числом ненулевых строк после приведения матрицы к ступенчатому виду. (Метод Гаусса)

* 1. Определитель матрицы

Вычисление определителя использует рекурсивный подход, основанный на разложении по строке или столбцу с использованием миноров.

**Программная реализация:**

* Определение типов:

1. **Вектор** Используется контейнер std::vector<double> для представления вектора:

using Vector = std::vector<double>;

1. **Матрица** Используется контейнер std::vector<std::vector<double>> для представления матрицы:

using Matrix = std::vector<std::vector<double>>;

* Функции для векторов:
  1. stringToVector (конвертация строки в вектор):
     + Входные данные: строка (wstring) и размер вектора (int).
     + Создать вектор заданного размера.
     + Считать элементы из строки и заполнить ими вектор.
     + Вернуть заполненный вектор.
  2. vectorToString (конвертация вектора в строку):
     + Входные данные: вектор.
     + Создать поток строк.
     + Последовательно добавить элементы вектора в поток.
     + Вернуть строку, полученную из потока.
  3. multiplyVectorByScalar (умножение вектора на скаляр):
     + Входные данные: вектор и скаляр.
     + Создать новый вектор.
     + Умножить каждый элемент вектора на скаляр.
     + Вернуть новый вектор.
  4. powerVector (возведение вектора в степень):
     + Входные данные: вектор и степень.
     + Создать новый вектор.
     + Возвести каждый элемент вектора в заданную степень.
     + Вернуть новый вектор.
  5. vectorLength (вычисление длины вектора):
     + Входные данные: вектор.
     + Вычислить квадрат каждого элемента и сложить их.
     + Взять квадратный корень из суммы.
     + Вернуть длину вектора.
  6. addVectors (сложение векторов):
     + Входные данные: два вектора.
     + Проверить, что размерности векторов совпадают.
     + Создать новый вектор.
     + Сложить соответствующие элементы векторов.
     + Вернуть новый вектор.
  7. subtractVectors (вычитание векторов):
     + Входные данные: два вектора.
     + Проверить, что размерности векторов совпадают.
     + Создать новый вектор.
     + Вычесть соответствующие элементы второго вектора из элементов первого.
     + Вернуть новый вектор.
  8. dotProduct (скалярное произведение векторов):
     + Входные данные: два вектора.
     + Проверить, что размерности векторов совпадают.
     + Создать переменную для хранения результата.
     + Вычислить сумму произведений соответствующих элементов векторов.
     + Вернуть скалярное произведение.
  9. crossProduct (векторное произведение для 3D векторов):
     + Входные данные: два 3D-вектора.
     + Проверить, что векторы имеют размерность 3.
     + Создать новый вектор.
     + Вычислить компоненты векторного произведения.
     + Вернуть новый вектор.
* Функции для матриц:
  1. stringToMatrix (конвертация строки в матрицу):
     + Входные данные: строка (wstring), количество строк и столбцов.
     + Создать матрицу заданного размера.
     + Считать элементы из строки и заполнить ими матрицу.
     + Вернуть заполненную матрицу.
  2. matrixToString (конвертация матрицы в строку):
     + Входные данные: матрица.
     + Создать поток строк.
     + Последовательно добавить элементы матрицы в поток, разделяя строки и столбцы.
     + Вернуть строку, полученную из потока.
  3. multiplyMatrixByScalar (умножение матрицы на скаляр):
     + Входные данные: матрица и скаляр.
     + Создать новую матрицу.
     + Умножить каждый элемент матрицы на скаляр.
     + Вернуть новую матрицу.
  4. transpose (транспонирование матрицы):
     + Входные данные: матрица.
     + Создать новую матрицу с транспонированными размерами.
     + Переписать элементы исходной матрицы в новую матрицу, меняя строки на столбцы.
     + Вернуть транспонированную матрицу.
  5. multiplyMatrices (умножение матриц):
     + Входные данные: две матрицы.
     + Проверить совместимость размерностей матриц для умножения.
     + Создать новую матрицу.
     + Вычислить произведение матриц.
     + Вернуть новую матрицу.
  6. powerMatrix (возведение матрицы в степень):
     + Входные данные: матрица и степень.
     + Проверить, что матрица квадратная.
     + Создать новую матрицу.
     + Вычислить произведение матрицы на саму себя нужное количество раз.
     + Вернуть новую матрицу.
  7. determinant (вычисление определителя матрицы):
     + Входные данные: матрица.
     + Проверить, что матрица квадратная.
     + Если матрица , вычислить определитель напрямую.
     + Для больших матриц рекурсивно вычислить определитель, используя миноры.
     + Вернуть определитель.
     + inverse (вычисление обратной матрицы):
     + Входные данные: матрица.
     + Проверить, что матрица квадратная.
     + Вычислить определитель матрицы.
     + Если определитель равен нулю, бросить исключение (матрица вырожденная).
     + Вычислить присоединенную матрицу.
     + Разделить каждый элемент присоединенной матрицы на определитель.
     + Вернуть обратную матрицу.
  8. addMatrices (сложение матриц):
     + Входные данные: две матрицы.
     + Проверить, что размерности матриц совпадают.
     + Создать новую матрицу.
     + Сложить соответствующие элементы матриц.
     + Вернуть новую матрицу.
  9. subtractMatrices (вычитание матриц):
     + Входные данные: две матрицы.
     + Проверить, что размерности матриц совпадают.
     + Создать новую матрицу.
     + Вычесть соответствующие элементы второй матрицы из элементов первой.
     + Вернуть новую матрицу.
  10. calculateRank (вычисление ранга матрицы):
      + Входные данные: матрица.
      + Привести матрицу к ступенчатому виду.
      + Подсчитать количество ненулевых строк.
      + Вернуть ранг матрицы.

**Оконные процедуры и диалоговые обработчики в Windows-приложении**

В Windows-приложении взаимодействие пользователя с интерфейсом обрабатывается с помощью оконных процедур и диалоговых обработчиков. Эти функции принимают и обрабатывают сообщения, которые система отправляет окну или диалогу.

Оконная процедура (Window Procedure)

Функция оконной процедуры отвечает за обработку всех сообщений, отправляемых окну.

**Параметры:**

* **Дескриптор окна (HWND hwnd):** Уникальный идентификатор окна, которому отправлено сообщение. Это позволяет функции идентифицировать, какое окно должно обработать сообщение.
* **Сообщение (UINT uMsg)**: Код сообщения, определяющий тип события или действия, которое необходимо обработать. Например, это может быть сообщение о перерисовке окна, нажатии клавиши или закрытии окна.
* **Дополнительная информация о сообщении (WPARAM wParam и LPARAM lParam)**: Эти параметры содержат дополнительную информацию, связанную с сообщением. Интерпретация этих параметров зависит от конкретного сообщения.

**Возвращаемое значение:**

* Функция возвращает значение, определяющее результат обработки сообщения.

Диалоговые обработчики (Dialog Procedures)

Диалоговые обработчики обрабатывают сообщения, отправляемые диалоговым окнам. Они обычно используются для обработки пользовательского ввода в диалоговых окнах.

**Параметры:**

* **Дескриптор диалогового окна (HWND hDlg)**: Уникальный идентификатор диалогового окна, которому отправлено сообщение.
* **Сообщение (UINT message)**: Код сообщения, определяющий тип события или действия, которое необходимо обработать.
* **Дополнительная информация о сообщении (WPARAM wParam и LPARAM lParam)**: Эти параметры содержат дополнительную информацию, связанную с сообщением. Интерпретация этих параметров зависит от конкретного сообщения.

**Возвращаемое значение:**

* Функция возвращает логическое значение, указывающее, обработано ли сообщение диалоговым окном.

Примеры диалоговых обработчиков

В программе могут быть несколько диалоговых обработчиков, каждый из которых отвечает за обработку сообщений для конкретного диалогового окна. Вот несколько примеров диалоговых обработчиков и их назначения:

1. **TransposeResultDlgProc**: обрабатывает сообщения для диалогового окна, отображающего результат транспонирования матрицы.
2. **ScalarResultDlgProc**: обрабатывает сообщения для диалогового окна, отображающего результат умножения матрицы или вектора на скаляр.
3. **ScalarInputDlgProc**: обрабатывает сообщения для диалогового окна, в котором пользователь вводит скалярное значение.
4. **PowerInputDlgProc**: обрабатывает сообщения для диалогового окна, в котором пользователь вводит степень для возведения матрицы или вектора.
5. **PowerResultDlgProc**: обрабатывает сообщения для диалогового окна, отображающего результат возведения матрицы или вектора в степень.
6. **DeterminantResultDlgProc**: обрабатывает сообщения для диалогового окна, отображающего результат вычисления определителя матрицы.
7. **VectorLengthResultDlgProc**: обрабатывает сообщения для диалогового окна, отображающего длину вектора.
8. **InverseResultDlgProc**: обрабатывает сообщения для диалогового окна, отображающего результат вычисления обратной матрицы.
9. **RankResultDlgProc**: обрабатывает сообщения для диалогового окна, отображающего ранг матрицы.
10. **MatrixOpResultDlgProc**: обрабатывает сообщения для диалогового окна, отображающего результаты операций над матрицами.
11. **VectorOpResultDlgProc**: обрабатывает сообщения для диалогового окна, отображающего результаты операций над векторами.
12. **VectorProdResultDlgProc**: обрабатывает сообщения для диалогового окна, отображающего результаты векторных произведений.

Каждый из этих обработчиков выполняет специфические действия, такие как получение ввода от пользователя, выполнение соответствующих вычислений, отображение результатов и управление элементами управления внутри диалогового окна.

**Создание окон и элементов управления в Windows-приложении:**

* Создание основного окна

Для создания основного окна приложения используется функция CreateWindowEx. Эта функция позволяет создавать окно с заданными параметрами, такими как стиль окна, позиция, размер и родительское окно.

* + Расширенный стиль окна (dwExStyle): Параметр, который определяет дополнительные стили окна. Обычно используется значение 0, что означает использование стилей по умолчанию.
  + Имя класса окна (lpClassName): Строка, которая указывает имя класса окна. Имя класса должно быть зарегистрировано ранее с помощью функции RegisterClass.
  + Текст заголовка окна (lpWindowName): Строка, которая отображается в заголовке окна.
  + Стиль окна (dwStyle): Параметр, определяющий основные стили окна, такие как WS\_OVERLAPPEDWINDOW, который создаёт стандартное перекрывающееся окно с заголовком и рамкой.
  + Позиция окна (X и Y): Координаты X и Y указывают начальную позицию окна на экране. Можно использовать значения CW\_USEDEFAULT, чтобы позволить системе выбрать позицию автоматически.
  + Размеры окна (nWidth и nHeight): Ширина и высота окна. Можно использовать CW\_USEDEFAULT для автоматического определения размеров.
  + Родительское окно (hWndParent): Дескриптор родительского окна. Для основного окна обычно указывается nullptr.
  + Дескриптор меню (hMenu): Дескриптор меню или идентификатор подменю. Для окон без меню указывается nullptr.
  + Экземпляр приложения (hInstance): Дескриптор экземпляра приложения, который создает окно. Этот параметр обычно передаётся из функции WinMain.
  + Дополнительные параметры (lpParam): Указатель на данные, которые будут переданы в оконную процедуру. Обычно используется nullptr.
* Создание кнопки

Кнопка создается с помощью функции CreateWindowEx, которая позволяет создать элемент управления кнопкой в родительском окне с заданными параметрами.

* + Расширенный стиль окна (dwExStyle): Дополнительные стили для элемента управления. Обычно используется значение 0.
  + Имя класса элемента управления (lpClassName): Имя класса элемента управления. Для кнопок используется строка "BUTTON".
  + Текст на кнопке (lpWindowName): Строка, отображаемая на кнопке.
  + Стиль окна (dwStyle): Параметр, который определяет стиль элемента управления. Например, WS\_TABSTOP | WS\_VISIBLE | WS\_CHILD | BS\_DEFPUSHBUTTON для кнопки, которая может получать фокус, видима, является дочерним элементом и имеет стиль стандартной кнопки.
  + Позиция кнопки (X и Y): Координаты позиции кнопки относительно родительского окна.
  + Размеры кнопки (nWidth и nHeight): Ширина и высота кнопки.
  + Родительское окно (hWndParent): Дескриптор родительского окна, в котором будет размещена кнопка.
  + Идентификатор кнопки (hMenu): Идентификатор кнопки, используемый для идентификации элемента управления при обработке сообщений.
  + Экземпляр приложения (hInstance): Дескриптор экземпляра приложения, в котором создается кнопка.
  + Дополнительные параметры (lpParam): Указатель на дополнительные параметры. Обычно используется nullptr.
* Создание текстового поля

Текстовое поле создается аналогично кнопке с использованием функции CreateWindowEx.

* + Расширенный стиль окна (dwExStyle): Дополнительные стили для текстового поля. Обычно используется значение 0.
  + Имя класса элемента управления (lpClassName): Имя класса элемента управления. Для текстовых полей используется строка "EDIT".
  + Начальное содержимое текстового поля (lpWindowName): Начальный текст, отображаемый в текстовом поле. Может быть nullptr для пустого поля.
  + Стиль окна (dwStyle): Параметр, определяющий стиль текстового поля. Например, WS\_CHILD | WS\_VISIBLE | WS\_BORDER | ES\_LEFT для видимого дочернего элемента с границей и выравниванием текста по левому краю.
  + Позиция текстового поля (X и Y): Координаты позиции текстового поля относительно родительского окна.
  + Размеры текстового поля (nWidth и nHeight): Ширина и высота текстового поля.
  + Родительское окно (hWndParent): Дескриптор родительского окна, в котором будет размещено текстовое поле.
  + Идентификатор текстового поля (hMenu): Идентификатор текстового поля, используемый для идентификации элемента управления при обработке сообщений.
  + Экземпляр приложения (hInstance): Дескриптор экземпляра приложения, в котором создается текстовое поле.
  + Дополнительные параметры (lpParam): Указатель на дополнительные параметры. Обычно используется nullptr.

Текст программы

**Файл “resource.h”:**

#define ID\_DIALOG 200

#define ID\_EDIT\_INPUT 201

#define ID\_RADIO\_VECTOR 202

#define ID\_RADIO\_MATRIX 203

#define ID\_BTN\_OK 204

#define ID\_BTN\_CANCEL 205

#define ID\_BTN\_NEXT 206

#define ID\_STATIC\_ROWS 207

#define ID\_STATIC\_COLS 208

#define ID\_EDIT\_ROWS 209

#define ID\_EDIT\_COLS 210

#define ID\_STATIC\_INPUT 211

#define ID\_STATIC\_RESULT\_FIRST 104

#define ID\_STATIC\_RESULT\_SECOND 105

#define IDI\_APP\_ICON 3001

#define IDB\_LOGO 1010

#define IDD\_TRANSPOSE\_RESULT 300

#define ID\_STATIC\_RESULT 301

#define ID\_BTN\_COPY 302

#define ID\_BTN\_PASTE\_FIRST 303

#define ID\_BTN\_OK 305

#define IDD\_SCALAR\_INPUT 400

#define IDD\_SCALAR\_RESULT 401

#define ID\_STATIC\_SCALAR 402

#define ID\_EDIT\_SCALAR 403

#define ID\_STATIC\_RESULT 404

#define ID\_BTN\_COPY 405

#define ID\_BTN\_PASTE\_FIRST 406

#define ID\_BTN\_OK 407

#define ID\_BTN\_CANCEL 408

#define IDD\_POWER\_INPUT 409

#define IDD\_POWER\_RESULT 410

#define ID\_STATIC\_POWER 411

#define ID\_EDIT\_POWER 412

#define ID\_STATIC\_POWER\_RESULT 413

#define ID\_BTN\_COPY\_POWER 414

#define ID\_BTN\_PASTE\_POWER\_FIRST 415

#define IDD\_DETERMINANT\_RESULT 416

#define ID\_STATIC\_DETERMINANT\_RESULT 417

#define ID\_BTN\_COPY\_DETERMINANT 418

#define IDD\_VECTOR\_LENGTH\_RESULT 420

#define ID\_STATIC\_VECTOR\_LENGTH\_RESULT 421

#define ID\_BTN\_COPY\_VECTOR\_LENGTH 422

#define IDD\_INVERSE\_RESULT 423

#define ID\_STATIC\_INVERSE\_RESULT 424

#define ID\_BTN\_COPY\_INVERSE 425

#define ID\_BTN\_PASTE\_INVERSE\_FIRST 426

#define IDD\_RANK\_RESULT 427

#define ID\_STATIC\_RANK\_RESULT 428

#define ID\_BTN\_COPY\_RANK 429

#define IDD\_MATRIX\_OP\_RESULT 430

#define ID\_STATIC\_MATRIX\_OP\_RESULT 431

#define ID\_BTN\_COPY\_MATRIX\_OP 432

#define ID\_BTN\_PASTE\_MATRIX\_OP\_FIRST 433

#define ID\_BTN\_PASTE\_MATRIX\_OP\_SECOND 434

#define ID\_BTN\_OK\_MATRIX\_OP 435

#define IDD\_VECTOR\_OP\_RESULT 436

#define ID\_STATIC\_VECTOR\_OP\_RESULT 437

#define ID\_BTN\_COPY\_VECTOR\_OP 438

#define ID\_BTN\_PASTE\_VECTOR\_OP\_FIRST 439

#define ID\_BTN\_PASTE\_VECTOR\_OP\_SECOND 440

#define ID\_BTN\_OK\_VECTOR\_OP 441

#define IDD\_VECTOR\_PROD\_RESULT 442

#define ID\_STATIC\_VECTOR\_PROD\_RESULT 443

#define ID\_BTN\_COPY\_VECTOR\_PROD 444

#define ID\_BTN\_OK\_VECTOR\_PROD 445

**Файл “main\_buttons.h”:**

#ifndef MAIN\_BUTTONS\_H

#define MAIN\_BUTTONS\_H

#define ID\_BTN\_START 1

#define ID\_BTN\_EXIT 2

#define ID\_BTN\_BACK 3

#define ID\_STATIC\_LABEL 100

#define ID\_STATIC\_CREDIT 101

#define ID\_BTN\_INPUT\_FIRST 102

#define ID\_BTN\_INPUT\_SECOND 103

#define ID\_STATIC\_TYPE\_FIRST 104

#define ID\_STATIC\_TYPE\_SECOND 105

#define ID\_STATIC\_RESULT\_FIRST 106

#define ID\_STATIC\_RESULT\_SECOND 107

#define ID\_BTN\_COPY\_FIRST 108

#define ID\_BTN\_COPY\_SECOND 109

#define ID\_BTN\_DETERMINANT\_FIRST 110

#define ID\_BTN\_INVERSE\_FIRST 111

#define ID\_BTN\_TRANSPOSE\_FIRST 112

#define ID\_BTN\_RANK\_FIRST 113

#define ID\_BTN\_MULTIPLY\_FIRST 114

#define ID\_BTN\_POWER\_FIRST 115

#define ID\_BTN\_DETERMINANT\_SECOND 116

#define ID\_BTN\_INVERSE\_SECOND 117

#define ID\_BTN\_TRANSPOSE\_SECOND 118

#define ID\_BTN\_RANK\_SECOND 119

#define ID\_BTN\_MULTIPLY\_SECOND 120

#define ID\_BTN\_POWER\_SECOND 121

#define ID\_BTN\_LENGTH\_FIRST 130

#define ID\_BTN\_MULTIPLY\_VECTOR\_FIRST 131

#define ID\_BTN\_POWER\_VECTOR\_FIRST 132

#define ID\_BTN\_LENGTH\_SECOND 133

#define ID\_BTN\_MULTIPLY\_VECTOR\_SECOND 134

#define ID\_BTN\_POWER\_VECTOR\_SECOND 135

#define ID\_BTN\_MATRIX\_MULTIPLY 200

#define ID\_BTN\_MATRIX\_ADD 201

#define ID\_BTN\_MATRIX\_SUBTRACT 202

#define ID\_BTN\_VECTOR\_ADD 203

#define ID\_BTN\_VECTOR\_SUBTRACT 204

#define ID\_BTN\_VECTOR\_DOT 205

#define ID\_BTN\_VECTOR\_CROSS 206

#define ID\_BTN\_VECTOR\_MIXED 207

#define ID\_BTN\_MATRIX\_VECTOR\_MULTIPLY 208

#endif

**Файл “utility\_functions.h”:**

#ifndef UTILITY\_FUNCTIONS\_H

#define UTILITY\_FUNCTIONS\_H

#include <windows.h>

#include <gdiplus.h>

#include <string>

extern HINSTANCE hInst;

extern ULONG\_PTR gdiplusToken;

extern Gdiplus::GdiplusStartupInput gdiplusStartupInput;

void CopyToClipboard(HWND hwnd);

void CreateControls(HWND hwnd);

void UpdateControlPositions(HWND hwnd);

void ShowPage(HWND hwnd, int page);

void UpdateResultStatic(HWND hwnd, int buttonId, const std::wstring& result, const std::wstring& type);

void ShowInputDialog(HWND hwnd, int buttonId);

Gdiplus::Image\* LoadImageFromResource(HINSTANCE hInstance, LPCWSTR resourceName);

void SetNewFont(HWND hwndDlg);

#endif // UTILITY\_FUNCTIONS\_H

**Файл “vector\_matrix\_operations.h”:**

#ifndef VECTOR\_MATRIX\_OPERATIONS\_H

#define VECTOR\_MATRIX\_OPERATIONS\_H

#include <vector>

#include <string>

using Matrix = std::vector<std::vector<double>>;

using Vector = std::vector<double>;

Vector stringToVector(const std::wstring& str, int size);

std::wstring vectorToString(const Vector& vec);

Vector multiplyVectorByScalar(const Vector& vec, double scalar);

Vector powerVector(const Vector& vec, int power);

double vectorLength(const Vector& vec);

Vector addVectors(const Vector& a, const Vector& b);

Vector subtractVectors(const Vector& a, const Vector& b);

double dotProduct(const Vector& a, const Vector& b);

Vector crossProduct(const Vector& a, const Vector& b);

double mixedProduct(const Vector& a, const Vector& b, const Vector& c);

bool checkVectorDimensions(const Vector& a, const Vector& b);

Matrix stringToMatrix(const std::wstring& str, int rows, int cols);

std::wstring matrixToString(const Matrix& matrix);

Matrix multiplyMatrixByScalar(const Matrix& matrix, double scalar);

Matrix transpose(const Matrix& matrix);

Matrix multiplyMatrices(const Matrix& A, const Matrix& B);

Matrix powerMatrix(const Matrix& matrix, int power);

double determinant(const Matrix& matrix);

Matrix getMinor(const Matrix& matrix, int row, int col);

Matrix adjugate(const Matrix& matrix);

Matrix inverse(const Matrix& matrix);

int calculateRank(Matrix matrix);

Matrix addMatrices(const Matrix& a, const Matrix& b);

Matrix subtractMatrices(const Matrix& a, const Matrix& b);

Matrix multiplyMatricesDiff(const Matrix& a, const Matrix& b);

bool isSquareMatrix(const Matrix& matrix);

bool checkMatrixDimensions(const Matrix& a, const Matrix& b);

#endif // VECTOR\_MATRIX\_OPERATIONS\_H

**Файл “dialog\_procedures.h”:**

#ifndef DIALOG\_PROCEDURES\_H

#define DIALOG\_PROCEDURES\_H

#include <windows.h>

LRESULT CALLBACK WindowProc(HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM lParam);

INT\_PTR CALLBACK DialogProc(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam);

INT\_PTR CALLBACK TransposeResultDlgProc(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam);

INT\_PTR CALLBACK ScalarResultDlgProc(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam);

INT\_PTR CALLBACK ScalarInputDlgProc(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam);

INT\_PTR CALLBACK PowerInputDlgProc(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam);

INT\_PTR CALLBACK PowerResultDlgProc(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam);

INT\_PTR CALLBACK DeterminantResultDlgProc(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam);

INT\_PTR CALLBACK VectorLengthResultDlgProc(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam);

INT\_PTR CALLBACK InverseResultDlgProc(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam);

INT\_PTR CALLBACK RankResultDlgProc(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam);

INT\_PTR CALLBACK MatrixOpResultDlgProc(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam);

INT\_PTR CALLBACK VectorOpResultDlgProc(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam);

INT\_PTR CALLBACK VectorProdResultDlgProc(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam);

#endif // DIALOG\_PROCEDURES\_H

**Файл “resource.rc”:**

// Скрипт ресурсов, созданный в Microsoft Visual C++.

//

#include "resource.h"

#include "winres.h"

IDI\_APP\_ICON ICON "icon.ico"

IDB\_LOGO BITMAP "logo.bmp"

ID\_DIALOG DIALOGEX 0, 0, 300, 300

STYLE DS\_SETFONT | DS\_MODALFRAME | WS\_POPUP | WS\_CAPTION | WS\_SYSMENU

CAPTION "Ввод элемента"

FONT 10, "MS Shell Dlg"

BEGIN

LTEXT "Выберите тип элемента:", -1, 10, 10, 280, 10

CONTROL "Вектор", ID\_RADIO\_VECTOR, "Button", BS\_AUTORADIOBUTTON, 10, 30, 100, 10

CONTROL "Матрица", ID\_RADIO\_MATRIX, "Button", BS\_AUTORADIOBUTTON, 120, 30, 100, 10

LTEXT "", ID\_STATIC\_ROWS, 10, 50, 280, 10

EDITTEXT ID\_EDIT\_ROWS, 10, 70, 280, 20, WS\_BORDER | ES\_NUMBER

LTEXT "", ID\_STATIC\_COLS, 10, 100, 280, 10

EDITTEXT ID\_EDIT\_COLS, 10, 120, 280, 20, WS\_BORDER | ES\_NUMBER

LTEXT "", ID\_STATIC\_INPUT, 10, 150, 280, 10

EDITTEXT ID\_EDIT\_INPUT, 10, 170, 280, 80, ES\_MULTILINE | WS\_BORDER | WS\_VSCROLL | ES\_WANTRETURN

DEFPUSHBUTTON "Далее", ID\_BTN\_NEXT, 50, 260, 50, 14

DEFPUSHBUTTON "Ок", ID\_BTN\_OK, 120, 260, 50, 14

PUSHBUTTON "Отмена", ID\_BTN\_CANCEL, 190, 260, 50, 14

END

IDD\_TRANSPOSE\_RESULT DIALOGEX 0, 0, 320, 240

STYLE DS\_SETFONT | DS\_MODALFRAME | WS\_POPUP | WS\_CAPTION | WS\_SYSMENU

CAPTION "Результат транспонирования"

FONT 10, "MS Shell Dlg"

BEGIN

LTEXT "", ID\_STATIC\_RESULT, 10, 10, 300, 150, SS\_LEFT | WS\_BORDER | WS\_VSCROLL | SS\_CENTER

PUSHBUTTON "Копировать", ID\_BTN\_COPY, 10, 180, 70, 14

PUSHBUTTON "Вставить", ID\_BTN\_PASTE\_FIRST, 90, 180, 100, 14

DEFPUSHBUTTON "Ок", ID\_BTN\_OK, 200, 180, 50, 14

END

IDD\_SCALAR\_INPUT DIALOGEX 0, 0, 320, 120

STYLE DS\_SETFONT | DS\_MODALFRAME | WS\_POPUP | WS\_CAPTION | WS\_SYSMENU

CAPTION "Введите число"

FONT 10, "MS Shell Dlg"

BEGIN

LTEXT "Введите число:", ID\_STATIC\_SCALAR, 10, 10, 300, 10, SS\_CENTER

EDITTEXT ID\_EDIT\_SCALAR, 10, 30, 300, 20, WS\_BORDER | ES\_NUMBER

DEFPUSHBUTTON "Ок", ID\_BTN\_OK, 60, 70, 50, 14

PUSHBUTTON "Отмена", ID\_BTN\_CANCEL, 200, 70, 50, 14

END

IDD\_SCALAR\_RESULT DIALOGEX 0, 0, 320, 240

STYLE DS\_SETFONT | DS\_MODALFRAME | WS\_POPUP | WS\_CAPTION | WS\_SYSMENU

CAPTION "Результат умножения"

FONT 10, "MS Shell Dlg"

BEGIN

LTEXT "", ID\_STATIC\_RESULT, 10, 10, 300, 150, SS\_LEFT | WS\_BORDER | WS\_VSCROLL | SS\_CENTER

PUSHBUTTON "Копировать", ID\_BTN\_COPY, 10, 180, 70, 14

PUSHBUTTON "Вставить", ID\_BTN\_PASTE\_FIRST, 90, 180, 100, 14

DEFPUSHBUTTON "Ок", ID\_BTN\_OK, 200, 180, 50, 14

END

IDD\_POWER\_INPUT DIALOGEX 0, 0, 320, 120

STYLE DS\_SETFONT | DS\_MODALFRAME | WS\_POPUP | WS\_CAPTION | WS\_SYSMENU

CAPTION "Введите степень"

FONT 10, "MS Shell Dlg"

BEGIN

LTEXT "Введите степень:", ID\_STATIC\_POWER, 10, 10, 300, 10, SS\_CENTER

EDITTEXT ID\_EDIT\_POWER, 10, 30, 300, 20, WS\_BORDER | ES\_NUMBER

DEFPUSHBUTTON "Ок", ID\_BTN\_OK, 60, 70, 50, 14

PUSHBUTTON "Отмена", ID\_BTN\_CANCEL, 200, 70, 50, 14

END

IDD\_POWER\_RESULT DIALOGEX 0, 0, 320, 240

STYLE DS\_SETFONT | DS\_MODALFRAME | WS\_POPUP | WS\_CAPTION | WS\_SYSMENU

CAPTION "Результат возведения в степень"

FONT 10, "MS Shell Dlg"

BEGIN

LTEXT "", ID\_STATIC\_POWER\_RESULT, 10, 10, 300, 150, SS\_LEFT | WS\_BORDER | WS\_VSCROLL | SS\_CENTER

PUSHBUTTON "Копировать", ID\_BTN\_COPY\_POWER, 10, 180, 70, 14

PUSHBUTTON "Вставить", ID\_BTN\_PASTE\_POWER\_FIRST, 90, 180, 100, 14

DEFPUSHBUTTON "Ок", ID\_BTN\_OK, 200, 180, 50, 14

END

IDD\_DETERMINANT\_RESULT DIALOGEX 0, 0, 320, 240

STYLE DS\_SETFONT | DS\_MODALFRAME | WS\_POPUP | WS\_CAPTION | WS\_SYSMENU

CAPTION "Результат определителя"

FONT 10, "MS Shell Dlg"

BEGIN

LTEXT "", ID\_STATIC\_DETERMINANT\_RESULT, 10, 10, 300, 150, SS\_LEFT | WS\_BORDER | WS\_VSCROLL | SS\_CENTER

PUSHBUTTON "Копировать", ID\_BTN\_COPY\_DETERMINANT, 10, 180, 70, 14

DEFPUSHBUTTON "Ок", ID\_BTN\_OK, 200, 180, 50, 14

END

IDD\_VECTOR\_LENGTH\_RESULT DIALOGEX 0, 0, 320, 240

STYLE DS\_SETFONT | DS\_MODALFRAME | WS\_POPUP | WS\_CAPTION | WS\_SYSMENU

CAPTION "Результат длины вектора"

FONT 10, "MS Shell Dlg"

BEGIN

LTEXT "", ID\_STATIC\_VECTOR\_LENGTH\_RESULT, 10, 10, 300, 150, SS\_LEFT | WS\_BORDER | WS\_VSCROLL | SS\_CENTER

PUSHBUTTON "Копировать", ID\_BTN\_COPY\_VECTOR\_LENGTH, 10, 180, 70, 14

DEFPUSHBUTTON "Ок", ID\_BTN\_OK, 200, 180, 50, 14

END

IDD\_INVERSE\_RESULT DIALOGEX 0, 0, 320, 240

STYLE DS\_SETFONT | DS\_MODALFRAME | WS\_POPUP | WS\_CAPTION | WS\_SYSMENU

CAPTION "Результат обратной матрицы"

FONT 10, "MS Shell Dlg"

BEGIN

LTEXT "", ID\_STATIC\_INVERSE\_RESULT, 10, 10, 300, 150, SS\_LEFT | WS\_BORDER | WS\_VSCROLL | SS\_CENTER

PUSHBUTTON "Копировать", ID\_BTN\_COPY\_INVERSE, 10, 180, 70, 14

PUSHBUTTON "Вставить", ID\_BTN\_PASTE\_INVERSE\_FIRST, 90, 180, 100, 14

DEFPUSHBUTTON "Ок", ID\_BTN\_OK, 200, 180, 50, 14

END

IDD\_RANK\_RESULT DIALOGEX 0, 0, 250, 150

STYLE DS\_SETFONT | DS\_MODALFRAME | WS\_POPUP | WS\_CAPTION | WS\_SYSMENU

CAPTION "Ранг Матрицы"

FONT 10, "MS Shell Dlg"

BEGIN

LTEXT "", ID\_STATIC\_RANK\_RESULT, 10, 10, 230, 60

DEFPUSHBUTTON "Копировать", ID\_BTN\_COPY\_RANK, 50, 80, 60, 14

DEFPUSHBUTTON "Ок", ID\_BTN\_OK, 120, 80, 60, 14

END

IDD\_MATRIX\_OP\_RESULT DIALOGEX 0, 0, 300, 150

STYLE DS\_SETFONT | DS\_MODALFRAME | WS\_POPUP | WS\_CAPTION | WS\_SYSMENU

CAPTION "Результат операции с матрицами"

FONT 12, "MS Shell Dlg"

BEGIN

LTEXT "", ID\_STATIC\_MATRIX\_OP\_RESULT, 10, 10, 230, 60

DEFPUSHBUTTON "Копировать", ID\_BTN\_COPY\_MATRIX\_OP, 10, 80, 60, 14

DEFPUSHBUTTON "Вставить в 1", ID\_BTN\_PASTE\_MATRIX\_OP\_FIRST, 80, 80, 60, 14

DEFPUSHBUTTON "Вставить в 2", ID\_BTN\_PASTE\_MATRIX\_OP\_SECOND, 150, 80, 60, 14

DEFPUSHBUTTON "Ок", ID\_BTN\_OK, 220, 80, 60, 14

END

IDD\_VECTOR\_OP\_RESULT DIALOGEX 0, 0, 300, 150

STYLE DS\_SETFONT | DS\_MODALFRAME | WS\_POPUP | WS\_CAPTION | WS\_SYSMENU

CAPTION "Результат операции с векторами"

FONT 12, "MS Shell Dlg"

BEGIN

LTEXT "", ID\_STATIC\_VECTOR\_OP\_RESULT, 10, 10, 230, 60

DEFPUSHBUTTON "Копировать", ID\_BTN\_COPY\_VECTOR\_OP, 10, 80, 60, 14

DEFPUSHBUTTON "Вставить в 1", ID\_BTN\_PASTE\_VECTOR\_OP\_FIRST, 80, 80, 60, 14

DEFPUSHBUTTON "Вставить в 2", ID\_BTN\_PASTE\_VECTOR\_OP\_SECOND, 150, 80, 60, 14

DEFPUSHBUTTON "Ок", ID\_BTN\_OK\_VECTOR\_OP, 220, 80, 60, 14

END

IDD\_VECTOR\_PROD\_RESULT DIALOGEX 0, 0, 250, 150

STYLE DS\_SETFONT | DS\_MODALFRAME | WS\_POPUP | WS\_CAPTION | WS\_SYSMENU

CAPTION "Результат операции с векторами"

FONT 10, "MS Shell Dlg"

BEGIN

LTEXT "", ID\_STATIC\_VECTOR\_PROD\_RESULT, 10, 10, 230, 60

DEFPUSHBUTTON "Копировать", ID\_BTN\_COPY\_VECTOR\_PROD, 10, 80, 60, 14

DEFPUSHBUTTON "Ок", ID\_BTN\_OK\_VECTOR\_PROD, 80, 80, 60, 14

END

**Файл “main.cpp”:**

#include <windows.h>

#include <vector>

#include <string>

#include <sstream>

#include <stdexcept>

#include <gdiplus.h>

#include <cmath>

#pragma comment (lib,"Gdiplus.lib")

#pragma comment(linker,"\"/manifestdependency:type='win32' \

name='Microsoft.Windows.Common-Controls' version='6.0.0.0' \

processorArchitecture='\*' publicKeyToken='6595b64144ccf1df' language='\*'\"")

#include "resource.h"

#include "main\_buttons.h"

#include "vector\_matrix\_operations.h"

#include "dialog\_procedures.h"

#include "utility\_functions.h"

HINSTANCE hInst;

bool showCopyFirst = false;

bool showCopySecond = false;

bool showMatrixButtonsFirst = false;

bool showMatrixButtonsSecond = false;

bool showVectorButtonsFirst = false;

bool showVectorButtonsSecond = false;

Matrix globalMatrixFirst;

Matrix globalMatrixSecond;

Vector globalVectorFirst;

Vector globalVectorSecond;

int globalRowsFirst = 0;

int globalColsFirst = 0;

int globalRowsSecond = 0;

int globalColsSecond = 0;

bool isFirstVector = false;

bool isSecondVector = false;

HFONT hNewFont = NULL;

ULONG\_PTR gdiplusToken;

Gdiplus::GdiplusStartupInput gdiplusStartupInput;

Gdiplus::Image\* pImage = nullptr;

Gdiplus::Image\* LoadImageFromResource(HINSTANCE hInstance, LPCWSTR resourceName)

{

HRSRC hResource = FindResource(hInstance, resourceName, RT\_BITMAP);

if (!hResource) return nullptr;

HGLOBAL hMem = LoadResource(hInstance, hResource);

if (!hMem) return nullptr;

void\* pData = LockResource(hMem);

DWORD size = SizeofResource(hInstance, hResource);

HGLOBAL hBuffer = GlobalAlloc(GMEM\_MOVEABLE, size);

void\* pBuffer = GlobalLock(hBuffer);

CopyMemory(pBuffer, pData, size);

IStream\* pStream = nullptr;

CreateStreamOnHGlobal(hBuffer, TRUE, &pStream);

Gdiplus::Image\* pImage = Gdiplus::Image::FromStream(pStream);

pStream->Release();

return pImage;

}

void SetNewFont(HWND hwndDlg) {

if (hNewFont == NULL) {

hNewFont = CreateFont(

24, // Высота шрифта (отрицательное значение указывает на высоту в пикселях)

0, 0, 0, FW\_NORMAL, FALSE, FALSE, FALSE,

DEFAULT\_CHARSET, OUT\_DEFAULT\_PRECIS, CLIP\_DEFAULT\_PRECIS,

DEFAULT\_QUALITY, DEFAULT\_PITCH | FF\_SWISS, L"Arial");

}

// Устанавливаем новый шрифт для всех элементов управления

HWND hChild = GetWindow(hwndDlg, GW\_CHILD);

while (hChild) {

SendMessage(hChild, WM\_SETFONT, (WPARAM)hNewFont, TRUE);

hChild = GetNextWindow(hChild, GW\_HWNDNEXT);

}

}

// Точка входа в приложение

int WINAPI wWinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, PWSTR pCmdLine, int nCmdShow)

{

hInst = hInstance;

// Инициализация GDI+

Gdiplus::GdiplusStartup(&gdiplusToken, &gdiplusStartupInput, NULL);

// Зарегистрируйте класс окна

const wchar\_t CLASS\_NAME[] = L"Sample Window Class";

WNDCLASSW wc = {};

wc.lpfnWndProc = WindowProc;

wc.hInstance = hInstance;

wc.lpszClassName = CLASS\_NAME;

wc.hbrBackground = (HBRUSH)(COLOR\_WINDOW + 1);

wc.hIcon = LoadIcon(hInstance, MAKEINTRESOURCE(IDI\_APP\_ICON));

RegisterClassW(&wc);

// Создайте окно

HWND hwnd = CreateWindowExW(

0,

CLASS\_NAME,

L"Vector-Matrix Transformations",

WS\_OVERLAPPEDWINDOW,

CW\_USEDEFAULT, CW\_USEDEFAULT, CW\_USEDEFAULT, CW\_USEDEFAULT,

NULL,

NULL,

hInstance,

NULL

);

if (hwnd == NULL)

{

return 0;

}

// Загрузить изображение

pImage = new Gdiplus::Image(L"logo.bmp");

if (pImage->GetLastStatus() != Gdiplus::Ok)

{

MessageBox(hwnd, L"Не удалось загрузить изображение!", L"Error", MB\_ICONERROR);

delete pImage;

pImage = nullptr;

}

ShowWindow(hwnd, nCmdShow);

// Цикл сообщений

MSG msg = {};

while (GetMessage(&msg, NULL, 0, 0))

{

TranslateMessage(&msg);

DispatchMessage(&msg);

}

// Очистка GDI+

Gdiplus::GdiplusShutdown(gdiplusToken);

return 0;

}

Vector stringToVector(const std::wstring& str, int size)

{

Vector vec(size);

std::wistringstream wiss(str);

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

{

if (!(wiss >> vec[i]))

{

throw std::runtime\_error("Неправильный формат вектора");

}

}

return vec;

}

std::wstring vectorToString(const Vector& vec)

{

std::wostringstream woss;

for (size\_t i = 0; i < vec.size(); ++i)

{

woss << vec[i];

if (i < vec.size() - 1)

{

woss << L" "; // Пространство между элементами

}

}

return woss.str();

}

Vector multiplyVectorByScalar(const Vector& vec, double scalar)

{

Vector result = vec;

for (auto& elem : result)

{

elem \*= scalar;

}

return result;

}

Vector powerVector(const Vector& vec, int power)

{

Vector result = vec;

for (auto& elem : result)

{

elem = std::pow(elem, power);

}

return result;

}

double vectorLength(const Vector& vec)

{

double sum = 0.0;

for (double elem : vec)

{

sum += elem \* elem;

}

return std::sqrt(sum);

}

Vector addVectors(const Vector& a, const Vector& b)

{

if (a.size() != b.size())

{

throw std::invalid\_argument("Векторы должны быть одного размера, чтобы их сложить");

}

Vector result(a.size());

for (size\_t i = 0; i < a.size(); ++i)

{

result[i] = a[i] + b[i];

}

return result;

}

Vector subtractVectors(const Vector& a, const Vector& b)

{

if (a.size() != b.size())

{

throw std::invalid\_argument("Векторы должны быть одного размера, чтобы их вычесть");

}

Vector result(a.size());

for (size\_t i = 0; i < a.size(); ++i)

{

result[i] = a[i] - b[i];

}

return result;

}

double dotProduct(const Vector& a, const Vector& b)

{

if (a.size() != b.size())

{

throw std::invalid\_argument("Векторы должны быть одного размера, чтобы найти их скалярное произведение");

}

double result = 0;

for (size\_t i = 0; i < a.size(); ++i)

{

result += a[i] \* b[i];

}

return result;

}

Vector crossProduct(const Vector& a, const Vector& b)

{

if (a.size() != 3 || b.size() != 3)

{

throw std::invalid\_argument("Векторное произведение определено только для трехмерных векторов");

}

Vector result(3);

result[0] = a[1] \* b[2] - a[2] \* b[1];

result[1] = a[2] \* b[0] - a[0] \* b[2];

result[2] = a[0] \* b[1] - a[1] \* b[0];

return result;

}

double mixedProduct(const Vector& a, const Vector& b, const Vector& c)

{

if (a.size() != 3 || b.size() != 3 || c.size() != 3)

{

throw std::invalid\_argument("Mixed product is defined only for 3D vectors");

}

Vector cross = crossProduct(a, b);

return dotProduct(cross, c);

}

bool checkVectorDimensions(const Vector& a, const Vector& b)

{

return a.size() == b.size();

}

Matrix transpose(const Matrix& matrix)

{

if (matrix.empty()) return Matrix();

int rows = matrix.size();

int cols = matrix[0].size();

Matrix transposed(cols, std::vector<double>(rows));

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

{

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

{

transposed[j][i] = matrix[i][j];

}

}

return transposed;

}

std::wstring matrixToString(const Matrix& matrix)

{

std::wostringstream woss;

for (const auto& row : matrix)

{

for (size\_t j = 0; j < row.size(); ++j)

{

woss << row[j];

if (j < row.size() - 1)

{

woss << L" "; // Пространство между элементами

}

}

woss << L"\n"; // Новая строка

}

return woss.str();

}

Matrix stringToMatrix(const std::wstring& str, int rows, int cols)

{

Matrix matrix(rows, std::vector<double>(cols));

std::wistringstream wiss(str);

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

{

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

{

if (!(wiss >> matrix[i][j]))

{

throw std::runtime\_error("Недопустимый формат матрицы");

}

}

}

return matrix;

}

Matrix multiplyMatrixByScalar(const Matrix& matrix, double scalar)

{

Matrix result = matrix;

for (auto& row : result)

{

for (auto& elem : row)

{

elem \*= scalar;

}

}

return result;

}

Matrix multiplyMatrices(const Matrix& A, const Matrix& B)

{

int rows = A.size();

int cols = B[0].size();

int inner = B.size();

Matrix result(rows, std::vector<double>(cols, 0.0));

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

{

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

{

for (int k = 0; k < inner; ++k)

{

result[i][j] += A[i][k] \* B[k][j];

}

}

}

return result;

}

Matrix powerMatrix(const Matrix& matrix, int power)

{

if (matrix.size() != matrix[0].size())

{

throw std::runtime\_error("Матрица должна быть квадратной, чтобы возвести ее в степень.");

}

Matrix result = matrix;

Matrix temp = matrix;

for (int p = 1; p < power; ++p)

{

result = multiplyMatrices(result, temp);

}

return result;

}

double determinant(const Matrix& matrix)

{

int n = matrix.size();

if (n == 1) return matrix[0][0];

double det = 0.0;

int sign = 1;

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

{

Matrix subMatrix(n - 1, std::vector<double>(n - 1));

for (int j = 1; j < n; ++j)

{

int subCol = 0;

for (int k = 0; k < n; ++k)

{

if (k == i) continue;

subMatrix[j - 1][subCol] = matrix[j][k];

++subCol;

}

}

det += sign \* matrix[0][i] \* determinant(subMatrix);

sign = -sign;

}

return det;

}

Matrix getMinor(const Matrix& matrix, int row, int col)

{

Matrix minorMatrix(matrix.size() - 1, std::vector<double>(matrix.size() - 1));

for (int i = 0, mi = 0; i < matrix.size(); ++i)

{

if (i == row) continue;

for (int j = 0, mj = 0; j < matrix.size(); ++j)

{

if (j == col) continue;

minorMatrix[mi][mj] = matrix[i][j];

++mj;

}

++mi;

}

return minorMatrix;

}

Matrix adjugate(const Matrix& matrix)

{

int size = matrix.size();

Matrix adj(size, std::vector<double>(size));

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

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

{

adj[j][i] = determinant(getMinor(matrix, i, j));

if ((i + j) % 2 == 1) adj[j][i] = -adj[j][i];

}

}

return adj;

}

Matrix inverse(const Matrix& matrix)

{

double det = determinant(matrix);

if (det == 0) throw std::runtime\_error("Матрица является единичной и не может иметь обратной.");

Matrix adj = adjugate(matrix);

int size = matrix.size();

Matrix inv(size, std::vector<double>(size));

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

{

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

{

inv[i][j] = adj[i][j] / det;

}

}

return inv;

}

int calculateRank(Matrix matrix) {

int m = matrix.size();

int n = matrix[0].size();

int rank = 0;

for (int row = 0; row < m; ++row) {

if (matrix[row][rank] != 0) {

for (int col = 0; col < m; ++col) {

if (col != row) {

double mult = matrix[col][rank] / matrix[row][rank];

for (int i = rank; i < n; ++i) {

matrix[col][i] -= mult \* matrix[row][i];

}

}

}

rank++;

}

if (rank == n) {

return rank;

}

}

return rank;

}

Matrix addMatrices(const Matrix& a, const Matrix& b)

{

if (a.size() != b.size() || a[0].size() != b[0].size())

{

throw std::invalid\_argument("Матрицы должны иметь одинаковые размеры, чтобы их можно было сложить");

}

int rows = a.size();

int cols = a[0].size();

Matrix result(rows, std::vector<double>(cols, 0));

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

{

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

{

result[i][j] = a[i][j] + b[i][j];

}

}

return result;

}

Matrix subtractMatrices(const Matrix& a, const Matrix& b)

{

if (a.size() != b.size() || a[0].size() != b[0].size())

{

throw std::invalid\_argument("Для вычитания матрицы должны иметь одинаковые размеры");

}

int rows = a.size();

int cols = a[0].size();

Matrix result(rows, std::vector<double>(cols, 0));

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

{

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

{

result[i][j] = a[i][j] - b[i][j];

}

}

return result;

}

Matrix multiplyMatricesDiff(const Matrix& a, const Matrix& b)

{

if (a[0].size() != b.size())

{

throw std::invalid\_argument("Матрицы нельзя перемножать");

}

int rows = a.size();

int cols = b[0].size();

int common\_dim = b.size();

Matrix result(rows, std::vector<double>(cols, 0));

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

{

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

{

for (int k = 0; k < common\_dim; ++k)

{

result[i][j] += a[i][k] \* b[k][j];

}

}

}

return result;

}

bool isSquareMatrix(const Matrix& matrix)

{

if (matrix.empty()) return false;

int numRows = matrix.size();

int numCols = matrix[0].size();

return numRows == numCols;

}

bool checkMatrixDimensions(const Matrix& a, const Matrix& b)

{

return a.size() == b.size() && (a.empty() || a[0].size() == b[0].size());

}

// Функция создания кнопок и надписей

void CreateControls(HWND hwnd)

{

// Создание надписей и кнопок

HWND hStatic = CreateWindowExW(

0,

L"STATIC",

L"Векторно-Матричный калькулятор",

WS\_CHILD | WS\_VISIBLE | SS\_CENTER,

0, 0, 0, 0, // Размеры будут обновлены позже

hwnd,

(HMENU)ID\_STATIC\_LABEL, // Label ID

(HINSTANCE)GetWindowLongPtr(hwnd, GWLP\_HINSTANCE),

NULL);

HFONT hFont = CreateFontW(

32, 0, 0, 0, FW\_BOLD, FALSE, FALSE, FALSE,

DEFAULT\_CHARSET, OUT\_DEFAULT\_PRECIS, CLIP\_DEFAULT\_PRECIS,

DEFAULT\_QUALITY, DEFAULT\_PITCH | FF\_SWISS, L"Arial");

SendMessage(hStatic, WM\_SETFONT, WPARAM(hFont), TRUE);

// Шрифт

HFONT hFontLarge = CreateFontW(

24, 0, 0, 0, FW\_BOLD, FALSE, FALSE, FALSE,

DEFAULT\_CHARSET, OUT\_DEFAULT\_PRECIS, CLIP\_DEFAULT\_PRECIS,

DEFAULT\_QUALITY, DEFAULT\_PITCH | FF\_SWISS, L"Arial");

HWND hBtnStart = CreateWindowW(

L"BUTTON",

L"Старт",

WS\_TABSTOP | WS\_VISIBLE | WS\_CHILD | BS\_DEFPUSHBUTTON,

0, 0, 0, 0,

hwnd,

(HMENU)ID\_BTN\_START,

(HINSTANCE)GetWindowLongPtr(hwnd, GWLP\_HINSTANCE), NULL);

SendMessage(hBtnStart, WM\_SETFONT, WPARAM(hFontLarge), TRUE);

HWND hBtnExit = CreateWindowW(

L"BUTTON",

L"Выход",

WS\_TABSTOP | WS\_VISIBLE | WS\_CHILD | BS\_DEFPUSHBUTTON,

0, 0, 0, 0,

hwnd,

(HMENU)ID\_BTN\_EXIT,

(HINSTANCE)GetWindowLongPtr(hwnd, GWLP\_HINSTANCE), NULL);

SendMessage(hBtnExit, WM\_SETFONT, WPARAM(hFontLarge), TRUE);

HWND hStaticCredit = CreateWindowW(

L"STATIC",

L"Made by Nikita P.",

WS\_CHILD | WS\_VISIBLE | SS\_CENTER,

0, 0, 0, 0,

hwnd,

(HMENU)ID\_STATIC\_CREDIT,

(HINSTANCE)GetWindowLongPtr(hwnd, GWLP\_HINSTANCE),

NULL);

SendMessage(hStaticCredit, WM\_SETFONT, WPARAM(hFontLarge), TRUE);

HWND hBtnBack = CreateWindowW(

L"BUTTON",

L"Назад",

WS\_TABSTOP | WS\_CHILD | BS\_DEFPUSHBUTTON,

10, 10, 100, 30,

hwnd,

(HMENU)ID\_BTN\_BACK,

(HINSTANCE)GetWindowLongPtr(hwnd, GWLP\_HINSTANCE), NULL);

SendMessage(hBtnBack, WM\_SETFONT, WPARAM(hFontLarge), TRUE);

HWND hBtnInputFirst = CreateWindowW(

L"BUTTON",

L"Ввод первого элемента",

WS\_TABSTOP | WS\_CHILD | BS\_DEFPUSHBUTTON,

0, 0, 0, 0,

hwnd,

(HMENU)ID\_BTN\_INPUT\_FIRST,

(HINSTANCE)GetWindowLongPtr(hwnd, GWLP\_HINSTANCE), NULL);

SendMessage(hBtnInputFirst, WM\_SETFONT, WPARAM(hFontLarge), TRUE);

HWND hBtnInputSecond = CreateWindowW(

L"BUTTON",

L"Ввод второго элемента",

WS\_TABSTOP | WS\_CHILD | BS\_DEFPUSHBUTTON,

0, 0, 0, 0,

hwnd,

(HMENU)ID\_BTN\_INPUT\_SECOND,

(HINSTANCE)GetWindowLongPtr(hwnd, GWLP\_HINSTANCE), NULL);

SendMessage(hBtnInputSecond, WM\_SETFONT, WPARAM(hFontLarge), TRUE);

HWND hStaticTypeFirst = CreateWindowW(

L"STATIC",

L"",

WS\_CHILD | WS\_VISIBLE | SS\_LEFT,

0, 0, 0, 0,

hwnd,

(HMENU)ID\_STATIC\_TYPE\_FIRST,

(HINSTANCE)GetWindowLongPtr(hwnd, GWLP\_HINSTANCE), NULL);

SendMessage(hStaticTypeFirst, WM\_SETFONT, WPARAM(hFontLarge), TRUE);

HWND hStaticTypeSecond = CreateWindowW(

L"STATIC",

L"",

WS\_CHILD | WS\_VISIBLE | SS\_LEFT,

0, 0, 0, 0,

hwnd,

(HMENU)ID\_STATIC\_TYPE\_SECOND,

(HINSTANCE)GetWindowLongPtr(hwnd, GWLP\_HINSTANCE), NULL);

SendMessage(hStaticTypeSecond, WM\_SETFONT, WPARAM(hFontLarge), TRUE);

HWND hResultFirst = CreateWindowW(

L"STATIC",

L"",

WS\_CHILD | WS\_VISIBLE | SS\_LEFT,

0, 0, 0, 0,

hwnd,

(HMENU)ID\_STATIC\_RESULT\_FIRST,

(HINSTANCE)GetWindowLongPtr(hwnd, GWLP\_HINSTANCE), NULL);

SendMessage(hResultFirst, WM\_SETFONT, WPARAM(hFontLarge), TRUE);

HWND hResultSecond = CreateWindowW(

L"STATIC",

L"",

WS\_CHILD | WS\_VISIBLE | SS\_LEFT,

0, 0, 0, 0,

hwnd,

(HMENU)ID\_STATIC\_RESULT\_SECOND,

(HINSTANCE)GetWindowLongPtr(hwnd, GWLP\_HINSTANCE), NULL);

SendMessage(hResultSecond, WM\_SETFONT, WPARAM(hFontLarge), TRUE);

HWND hBtnCopyFirst = CreateWindowW(

L"BUTTON",

L"Копи",

WS\_TABSTOP | WS\_CHILD | BS\_DEFPUSHBUTTON | WS\_VISIBLE,

0, 0, 50, 30,

hwnd,

(HMENU)ID\_BTN\_COPY\_FIRST,

(HINSTANCE)GetWindowLongPtr(hwnd, GWLP\_HINSTANCE), NULL);

SendMessage(hBtnCopyFirst, WM\_SETFONT, WPARAM(hFontLarge), TRUE);

HWND hBtnCopySecond = CreateWindowW(

L"BUTTON",

L"Копи",

WS\_TABSTOP | WS\_CHILD | BS\_DEFPUSHBUTTON | WS\_VISIBLE,

0, 0, 50, 30,

hwnd,

(HMENU)ID\_BTN\_COPY\_SECOND,

(HINSTANCE)GetWindowLongPtr(hwnd, GWLP\_HINSTANCE), NULL);

SendMessage(hBtnCopySecond, WM\_SETFONT, WPARAM(hFontLarge), TRUE);

// Create buttons for matrix and vector operations

HWND hBtnDetFirst = CreateWindowW(L"BUTTON", L"Определитель", WS\_TABSTOP | WS\_CHILD | BS\_DEFPUSHBUTTON, 0, 0, 0, 0, hwnd, (HMENU)ID\_BTN\_DETERMINANT\_FIRST, (HINSTANCE)GetWindowLongPtr(hwnd, GWLP\_HINSTANCE), NULL);

SendMessage(hBtnDetFirst, WM\_SETFONT, WPARAM(hFontLarge), TRUE);

HWND hBtnInvFirst = CreateWindowW(L"BUTTON", L"Обратная", WS\_TABSTOP | WS\_CHILD | BS\_DEFPUSHBUTTON, 0, 0, 0, 0, hwnd, (HMENU)ID\_BTN\_INVERSE\_FIRST, (HINSTANCE)GetWindowLongPtr(hwnd, GWLP\_HINSTANCE), NULL);

SendMessage(hBtnInvFirst, WM\_SETFONT, WPARAM(hFontLarge), TRUE);

HWND hBtnTransFirst = CreateWindowW(L"BUTTON", L"Транспонировать", WS\_TABSTOP | WS\_CHILD | BS\_DEFPUSHBUTTON, 0, 0, 0, 0, hwnd, (HMENU)ID\_BTN\_TRANSPOSE\_FIRST, (HINSTANCE)GetWindowLongPtr(hwnd, GWLP\_HINSTANCE), NULL);

SendMessage(hBtnTransFirst, WM\_SETFONT, WPARAM(hFontLarge), TRUE);

HWND hBtnRankFirst = CreateWindowW(L"BUTTON", L"Ранг", WS\_TABSTOP | WS\_CHILD | BS\_DEFPUSHBUTTON, 0, 0, 0, 0, hwnd, (HMENU)ID\_BTN\_RANK\_FIRST, (HINSTANCE)GetWindowLongPtr(hwnd, GWLP\_HINSTANCE), NULL);

SendMessage(hBtnRankFirst, WM\_SETFONT, WPARAM(hFontLarge), TRUE);

HWND hBtnMultFirst = CreateWindowW(L"BUTTON", L"Умножить на число", WS\_TABSTOP | WS\_CHILD | BS\_DEFPUSHBUTTON, 0, 0, 0, 0, hwnd, (HMENU)ID\_BTN\_MULTIPLY\_FIRST, (HINSTANCE)GetWindowLongPtr(hwnd, GWLP\_HINSTANCE), NULL);

SendMessage(hBtnMultFirst, WM\_SETFONT, WPARAM(hFontLarge), TRUE);

HWND hBtnPowerFirst = CreateWindowW(L"BUTTON", L"Возвести в степень", WS\_TABSTOP | WS\_CHILD | BS\_DEFPUSHBUTTON, 0, 0, 0, 0, hwnd, (HMENU)ID\_BTN\_POWER\_FIRST, (HINSTANCE)GetWindowLongPtr(hwnd, GWLP\_HINSTANCE), NULL);

SendMessage(hBtnPowerFirst, WM\_SETFONT, WPARAM(hFontLarge), TRUE);

HWND hBtnDetSecond = CreateWindowW(L"BUTTON", L"Определитель", WS\_TABSTOP | WS\_CHILD | BS\_DEFPUSHBUTTON, 0, 0, 0, 0, hwnd, (HMENU)ID\_BTN\_DETERMINANT\_SECOND, (HINSTANCE)GetWindowLongPtr(hwnd, GWLP\_HINSTANCE), NULL);

SendMessage(hBtnDetSecond, WM\_SETFONT, WPARAM(hFontLarge), TRUE);

HWND hBtnInvSecond = CreateWindowW(L"BUTTON", L"Обратная", WS\_TABSTOP | WS\_CHILD | BS\_DEFPUSHBUTTON, 0, 0, 0, 0, hwnd, (HMENU)ID\_BTN\_INVERSE\_SECOND, (HINSTANCE)GetWindowLongPtr(hwnd, GWLP\_HINSTANCE), NULL);

SendMessage(hBtnInvSecond, WM\_SETFONT, WPARAM(hFontLarge), TRUE);

HWND hBtnTransSecond = CreateWindowW(L"BUTTON", L"Транспонировать", WS\_TABSTOP | WS\_CHILD | BS\_DEFPUSHBUTTON, 0, 0, 0, 0, hwnd, (HMENU)ID\_BTN\_TRANSPOSE\_SECOND, (HINSTANCE)GetWindowLongPtr(hwnd, GWLP\_HINSTANCE), NULL);

SendMessage(hBtnTransSecond, WM\_SETFONT, WPARAM(hFontLarge), TRUE);

HWND hBtnRankSecond = CreateWindowW(L"BUTTON", L"Ранг", WS\_TABSTOP | WS\_CHILD | BS\_DEFPUSHBUTTON, 0, 0, 0, 0, hwnd, (HMENU)ID\_BTN\_RANK\_SECOND, (HINSTANCE)GetWindowLongPtr(hwnd, GWLP\_HINSTANCE), NULL);

SendMessage(hBtnRankSecond, WM\_SETFONT, WPARAM(hFontLarge), TRUE);

HWND hBtnMultSecond = CreateWindowW(L"BUTTON", L"Умножить на число", WS\_TABSTOP | WS\_CHILD | BS\_DEFPUSHBUTTON, 0, 0, 0, 0, hwnd, (HMENU)ID\_BTN\_MULTIPLY\_SECOND, (HINSTANCE)GetWindowLongPtr(hwnd, GWLP\_HINSTANCE), NULL);

SendMessage(hBtnMultSecond, WM\_SETFONT, WPARAM(hFontLarge), TRUE);

HWND hBtnPowerSecond = CreateWindowW(L"BUTTON", L"Возвести в степень", WS\_TABSTOP | WS\_CHILD | BS\_DEFPUSHBUTTON, 0, 0, 0, 0, hwnd, (HMENU)ID\_BTN\_POWER\_SECOND, (HINSTANCE)GetWindowLongPtr(hwnd, GWLP\_HINSTANCE), NULL);

SendMessage(hBtnPowerSecond, WM\_SETFONT, WPARAM(hFontLarge), TRUE);

HWND hBtnLengthFirst = CreateWindowW(L"BUTTON", L"Длина", WS\_TABSTOP | WS\_CHILD | BS\_DEFPUSHBUTTON, 0, 0, 0, 0, hwnd, (HMENU)ID\_BTN\_LENGTH\_FIRST, (HINSTANCE)GetWindowLongPtr(hwnd, GWLP\_HINSTANCE), NULL);

SendMessage(hBtnLengthFirst, WM\_SETFONT, WPARAM(hFontLarge), TRUE);

HWND hBtnMultVectorFirst = CreateWindowW(L"BUTTON", L"Умножить на число", WS\_TABSTOP | WS\_CHILD | BS\_DEFPUSHBUTTON, 0, 0, 0, 0, hwnd, (HMENU)ID\_BTN\_MULTIPLY\_VECTOR\_FIRST, (HINSTANCE)GetWindowLongPtr(hwnd, GWLP\_HINSTANCE), NULL);

SendMessage(hBtnMultVectorFirst, WM\_SETFONT, WPARAM(hFontLarge), TRUE);

HWND hBtnPowerVectorFirst = CreateWindowW(L"BUTTON", L"Возвести в степень", WS\_TABSTOP | WS\_CHILD | BS\_DEFPUSHBUTTON, 0, 0, 0, 0, hwnd, (HMENU)ID\_BTN\_POWER\_VECTOR\_FIRST, (HINSTANCE)GetWindowLongPtr(hwnd, GWLP\_HINSTANCE), NULL);

SendMessage(hBtnPowerVectorFirst, WM\_SETFONT, WPARAM(hFontLarge), TRUE);

HWND hBtnLengthSecond = CreateWindowW(L"BUTTON", L"Длина", WS\_TABSTOP | WS\_CHILD | BS\_DEFPUSHBUTTON, 0, 0, 0, 0, hwnd, (HMENU)ID\_BTN\_LENGTH\_SECOND, (HINSTANCE)GetWindowLongPtr(hwnd, GWLP\_HINSTANCE), NULL);

SendMessage(hBtnLengthSecond, WM\_SETFONT, WPARAM(hFontLarge), TRUE);

HWND hBtnMultVectorSecond = CreateWindowW(L"BUTTON", L"Умножить на число", WS\_TABSTOP | WS\_CHILD | BS\_DEFPUSHBUTTON, 0, 0, 0, 0, hwnd, (HMENU)ID\_BTN\_MULTIPLY\_VECTOR\_SECOND, (HINSTANCE)GetWindowLongPtr(hwnd, GWLP\_HINSTANCE), NULL);

SendMessage(hBtnMultVectorSecond, WM\_SETFONT, WPARAM(hFontLarge), TRUE);

HWND hBtnPowerVectorSecond = CreateWindowW(L"BUTTON", L"Возвести в степень", WS\_TABSTOP | WS\_CHILD | BS\_DEFPUSHBUTTON, 0, 0, 0, 0, hwnd, (HMENU)ID\_BTN\_POWER\_VECTOR\_SECOND, (HINSTANCE)GetWindowLongPtr(hwnd, GWLP\_HINSTANCE), NULL);

SendMessage(hBtnPowerVectorSecond, WM\_SETFONT, WPARAM(hFontLarge), TRUE);

HWND hBtnMatrixMultiply = CreateWindowW(L"BUTTON", L"Умножение", WS\_TABSTOP | WS\_CHILD | BS\_DEFPUSHBUTTON, 0, 0, 0, 0, hwnd, (HMENU)ID\_BTN\_MATRIX\_MULTIPLY, (HINSTANCE)GetWindowLongPtr(hwnd, GWLP\_HINSTANCE), NULL);

SendMessage(hBtnMatrixMultiply, WM\_SETFONT, WPARAM(hFontLarge), TRUE);

HWND hBtnMatrixAdd = CreateWindowW(L"BUTTON", L"Сложение", WS\_TABSTOP | WS\_CHILD | BS\_DEFPUSHBUTTON, 0, 0, 0, 0, hwnd, (HMENU)ID\_BTN\_MATRIX\_ADD, (HINSTANCE)GetWindowLongPtr(hwnd, GWLP\_HINSTANCE), NULL);

SendMessage(hBtnMatrixAdd, WM\_SETFONT, WPARAM(hFontLarge), TRUE);

HWND hBtnMatrixSubtract = CreateWindowW(L"BUTTON", L"Вычитание", WS\_TABSTOP | WS\_CHILD | BS\_DEFPUSHBUTTON, 0, 0, 0, 0, hwnd, (HMENU)ID\_BTN\_MATRIX\_SUBTRACT, (HINSTANCE)GetWindowLongPtr(hwnd, GWLP\_HINSTANCE), NULL);

SendMessage(hBtnMatrixSubtract, WM\_SETFONT, WPARAM(hFontLarge), TRUE);

HWND hBtnVectorAdd = CreateWindowW(L"BUTTON", L"Сложение", WS\_TABSTOP | WS\_CHILD | BS\_DEFPUSHBUTTON, 0, 0, 0, 0, hwnd, (HMENU)ID\_BTN\_VECTOR\_ADD, (HINSTANCE)GetWindowLongPtr(hwnd, GWLP\_HINSTANCE), NULL);

SendMessage(hBtnVectorAdd, WM\_SETFONT, WPARAM(hFontLarge), TRUE);

HWND hBtnVectorSubtract = CreateWindowW(L"BUTTON", L"Вычитание", WS\_TABSTOP | WS\_CHILD | BS\_DEFPUSHBUTTON, 0, 0, 0, 0, hwnd, (HMENU)ID\_BTN\_VECTOR\_SUBTRACT, (HINSTANCE)GetWindowLongPtr(hwnd, GWLP\_HINSTANCE), NULL);

SendMessage(hBtnVectorSubtract, WM\_SETFONT, WPARAM(hFontLarge), TRUE);

HWND hBtnVectorDot = CreateWindowW(L"BUTTON", L"Скалярное произведение", WS\_TABSTOP | WS\_CHILD | BS\_DEFPUSHBUTTON, 0, 0, 0, 0, hwnd, (HMENU)ID\_BTN\_VECTOR\_DOT, (HINSTANCE)GetWindowLongPtr(hwnd, GWLP\_HINSTANCE), NULL);

SendMessage(hBtnVectorDot, WM\_SETFONT, WPARAM(hFontLarge), TRUE);

HWND hBtnVectorCross = CreateWindowW(L"BUTTON", L"Векторное произведение", WS\_TABSTOP | WS\_CHILD | BS\_DEFPUSHBUTTON, 0, 0, 0, 0, hwnd, (HMENU)ID\_BTN\_VECTOR\_CROSS, (HINSTANCE)GetWindowLongPtr(hwnd, GWLP\_HINSTANCE), NULL);

SendMessage(hBtnVectorCross, WM\_SETFONT, WPARAM(hFontLarge), TRUE);

HWND hBtnVectorMixed = CreateWindowW(L"BUTTON", L"Смешанное произведение", WS\_TABSTOP | WS\_CHILD | BS\_DEFPUSHBUTTON, 0, 0, 0, 0, hwnd, (HMENU)ID\_BTN\_VECTOR\_MIXED, (HINSTANCE)GetWindowLongPtr(hwnd, GWLP\_HINSTANCE), NULL);

SendMessage(hBtnVectorMixed, WM\_SETFONT, WPARAM(hFontLarge), TRUE);

HWND hBtnMatrixVectorMultiply = CreateWindowW(L"BUTTON", L"Умножение матрицы на вектор", WS\_TABSTOP | WS\_CHILD | BS\_DEFPUSHBUTTON, 0, 0, 0, 0, hwnd, (HMENU)ID\_BTN\_MATRIX\_VECTOR\_MULTIPLY, (HINSTANCE)GetWindowLongPtr(hwnd, GWLP\_HINSTANCE), NULL);

SendMessage(hBtnMatrixVectorMultiply, WM\_SETFONT, WPARAM(hFontLarge), TRUE);

}

// Функция обновления позиций кнопок управления

void UpdateControlPositions(HWND hwnd)

{

RECT rect;

GetClientRect(hwnd, &rect);

int btnWidth = 240;

int btnHeight = 30;

int spacing = 10;

int labelWidth = 300;

int labelHeight = 65;

int startX = (rect.right - btnWidth) / 2;

int startY = (rect.bottom - (2 \* btnHeight + spacing)) / 2;

HWND hStatic = GetDlgItem(hwnd, ID\_STATIC\_LABEL);

SetWindowPos(hStatic, NULL, (rect.right - labelWidth) / 2, startY - 2 \* btnHeight - 20, labelWidth, labelHeight, SWP\_NOZORDER);

SetWindowPos(GetDlgItem(hwnd, ID\_BTN\_START), NULL, startX, startY, btnWidth, btnHeight, SWP\_NOZORDER);

SetWindowPos(GetDlgItem(hwnd, ID\_BTN\_EXIT), NULL, startX, startY + btnHeight + spacing, btnWidth, btnHeight, SWP\_NOZORDER);

HWND hCredit = GetDlgItem(hwnd, ID\_STATIC\_CREDIT);

SetWindowPos(hCredit, NULL, startX, startY + 2 \* (btnHeight + spacing), btnWidth, btnHeight, SWP\_NOZORDER);

SetWindowPos(GetDlgItem(hwnd, ID\_BTN\_BACK), NULL, 10, 10, 100, 30, SWP\_NOZORDER);

int margin = rect.right / 4 - btnWidth / 2;

SetWindowPos(GetDlgItem(hwnd, ID\_BTN\_INPUT\_FIRST), NULL, margin, (rect.bottom - btnHeight) / 5, btnWidth + 20, btnHeight + 30, SWP\_NOZORDER);

SetWindowPos(GetDlgItem(hwnd, ID\_BTN\_INPUT\_SECOND), NULL, rect.right - btnWidth - margin, (rect.bottom - btnHeight) / 5, btnWidth + 20, btnHeight + 30, SWP\_NOZORDER);

SetWindowPos(GetDlgItem(hwnd, ID\_BTN\_COPY\_FIRST), NULL, margin + btnWidth + 30, (rect.bottom - btnHeight) / 5, 70, btnHeight + 30, SWP\_NOZORDER);

SetWindowPos(GetDlgItem(hwnd, ID\_BTN\_COPY\_SECOND), NULL, rect.right - margin + btnWidth / 8, (rect.bottom - btnHeight) / 5, 70, btnHeight + 30, SWP\_NOZORDER);

int resultStartY = (rect.bottom - btnHeight) / 5 + 2 \* (btnHeight + spacing);

SetWindowPos(GetDlgItem(hwnd, ID\_STATIC\_TYPE\_FIRST), NULL, margin, resultStartY - 20, rect.right - 2 \* margin, btnHeight, SWP\_NOZORDER);

SetWindowPos(GetDlgItem(hwnd, ID\_STATIC\_TYPE\_SECOND), NULL, rect.right - btnWidth - margin, resultStartY - 20, rect.right - 2 \* margin, btnHeight, SWP\_NOZORDER);

SetWindowPos(GetDlgItem(hwnd, ID\_STATIC\_RESULT\_FIRST), NULL, margin, resultStartY + 2, rect.right - margin, 6 \* btnHeight, SWP\_NOZORDER);

SetWindowPos(GetDlgItem(hwnd, ID\_STATIC\_RESULT\_SECOND), NULL, rect.right - btnWidth - margin, resultStartY, rect.right - margin, 6 \* btnHeight, SWP\_NOZORDER);

int matrixButtonYFirst = resultStartY + 6 \* btnHeight + spacing;

SetWindowPos(GetDlgItem(hwnd, ID\_BTN\_DETERMINANT\_FIRST), NULL, margin, matrixButtonYFirst, btnWidth, btnHeight, SWP\_NOZORDER);

SetWindowPos(GetDlgItem(hwnd, ID\_BTN\_INVERSE\_FIRST), NULL, margin, matrixButtonYFirst + btnHeight + spacing, btnWidth, btnHeight, SWP\_NOZORDER);

SetWindowPos(GetDlgItem(hwnd, ID\_BTN\_TRANSPOSE\_FIRST), NULL, margin, matrixButtonYFirst + 2 \* (btnHeight + spacing), btnWidth, btnHeight, SWP\_NOZORDER);

SetWindowPos(GetDlgItem(hwnd, ID\_BTN\_RANK\_FIRST), NULL, margin, matrixButtonYFirst + 3 \* (btnHeight + spacing), btnWidth, btnHeight, SWP\_NOZORDER);

SetWindowPos(GetDlgItem(hwnd, ID\_BTN\_MULTIPLY\_FIRST), NULL, margin, matrixButtonYFirst + 4 \* (btnHeight + spacing), btnWidth, btnHeight, SWP\_NOZORDER);

SetWindowPos(GetDlgItem(hwnd, ID\_BTN\_POWER\_FIRST), NULL, margin, matrixButtonYFirst + 5 \* (btnHeight + spacing), btnWidth, btnHeight, SWP\_NOZORDER);

int matrixButtonYSecond = resultStartY + 6 \* btnHeight + spacing;

SetWindowPos(GetDlgItem(hwnd, ID\_BTN\_DETERMINANT\_SECOND), NULL, rect.right - btnWidth - margin, matrixButtonYSecond, btnWidth, btnHeight, SWP\_NOZORDER);

SetWindowPos(GetDlgItem(hwnd, ID\_BTN\_INVERSE\_SECOND), NULL, rect.right - btnWidth - margin, matrixButtonYSecond + btnHeight + spacing, btnWidth, btnHeight, SWP\_NOZORDER);

SetWindowPos(GetDlgItem(hwnd, ID\_BTN\_TRANSPOSE\_SECOND), NULL, rect.right - btnWidth - margin, matrixButtonYSecond + 2 \* (btnHeight + spacing), btnWidth, btnHeight, SWP\_NOZORDER);

SetWindowPos(GetDlgItem(hwnd, ID\_BTN\_RANK\_SECOND), NULL, rect.right - btnWidth - margin, matrixButtonYSecond + 3 \* (btnHeight + spacing), btnWidth, btnHeight, SWP\_NOZORDER);

SetWindowPos(GetDlgItem(hwnd, ID\_BTN\_MULTIPLY\_SECOND), NULL, rect.right - btnWidth - margin, matrixButtonYSecond + 4 \* (btnHeight + spacing), btnWidth, btnHeight, SWP\_NOZORDER);

SetWindowPos(GetDlgItem(hwnd, ID\_BTN\_POWER\_SECOND), NULL, rect.right - btnWidth - margin, matrixButtonYSecond + 5 \* (btnHeight + spacing), btnWidth, btnHeight, SWP\_NOZORDER);

int vectorButtonYFirst = resultStartY + 6 \* btnHeight + spacing;

SetWindowPos(GetDlgItem(hwnd, ID\_BTN\_LENGTH\_FIRST), NULL, margin, vectorButtonYFirst, btnWidth, btnHeight, SWP\_NOZORDER);

SetWindowPos(GetDlgItem(hwnd, ID\_BTN\_MULTIPLY\_VECTOR\_FIRST), NULL, margin, vectorButtonYFirst + btnHeight + spacing, btnWidth, btnHeight, SWP\_NOZORDER);

SetWindowPos(GetDlgItem(hwnd, ID\_BTN\_POWER\_VECTOR\_FIRST), NULL, margin, vectorButtonYFirst + 2 \* (btnHeight + spacing), btnWidth, btnHeight, SWP\_NOZORDER);

int vectorButtonYSecond = resultStartY + 6 \* btnHeight + spacing;

SetWindowPos(GetDlgItem(hwnd, ID\_BTN\_LENGTH\_SECOND), NULL, rect.right - btnWidth - margin, vectorButtonYSecond, btnWidth, btnHeight, SWP\_NOZORDER);

SetWindowPos(GetDlgItem(hwnd, ID\_BTN\_MULTIPLY\_VECTOR\_SECOND), NULL, rect.right - btnWidth - margin, vectorButtonYSecond + btnHeight + spacing, btnWidth, btnHeight, SWP\_NOZORDER);

SetWindowPos(GetDlgItem(hwnd, ID\_BTN\_POWER\_VECTOR\_SECOND), NULL, rect.right - btnWidth - margin, vectorButtonYSecond + 2 \* (btnHeight + spacing), btnWidth, btnHeight, SWP\_NOZORDER);

int operationButtonY = resultStartY + 6 \* btnHeight + spacing;

SetWindowPos(GetDlgItem(hwnd, ID\_BTN\_MATRIX\_MULTIPLY), NULL, (rect.right - btnWidth) / 2, operationButtonY, btnWidth + 50, btnHeight, SWP\_NOZORDER);

SetWindowPos(GetDlgItem(hwnd, ID\_BTN\_MATRIX\_ADD), NULL, (rect.right - btnWidth) / 2, operationButtonY + btnHeight + spacing, btnWidth + 50, btnHeight, SWP\_NOZORDER);

SetWindowPos(GetDlgItem(hwnd, ID\_BTN\_MATRIX\_SUBTRACT), NULL, (rect.right - btnWidth) / 2, operationButtonY + 2 \* (btnHeight + spacing), btnWidth + 50, btnHeight, SWP\_NOZORDER);

SetWindowPos(GetDlgItem(hwnd, ID\_BTN\_VECTOR\_ADD), NULL, (rect.right - btnWidth) / 2, operationButtonY, btnWidth + 50, btnHeight, SWP\_NOZORDER);

SetWindowPos(GetDlgItem(hwnd, ID\_BTN\_VECTOR\_SUBTRACT), NULL, (rect.right - btnWidth) / 2, operationButtonY + btnHeight + spacing, btnWidth + 50, btnHeight, SWP\_NOZORDER);

SetWindowPos(GetDlgItem(hwnd, ID\_BTN\_VECTOR\_DOT), NULL, (rect.right - btnWidth) / 2, operationButtonY + 2 \* (btnHeight + spacing), btnWidth + 50, btnHeight, SWP\_NOZORDER);

SetWindowPos(GetDlgItem(hwnd, ID\_BTN\_VECTOR\_CROSS), NULL, (rect.right - btnWidth) / 2, operationButtonY + 3 \* (btnHeight + spacing), btnWidth + 50, btnHeight, SWP\_NOZORDER);

SetWindowPos(GetDlgItem(hwnd, ID\_BTN\_VECTOR\_MIXED), NULL, (rect.right - btnWidth) / 2, operationButtonY + 4 \* (btnHeight + spacing), btnWidth + 50, btnHeight, SWP\_NOZORDER);

SetWindowPos(GetDlgItem(hwnd, ID\_BTN\_MATRIX\_VECTOR\_MULTIPLY), NULL, (rect.right - btnWidth) / 2 - 15, operationButtonY, btnWidth + 90, btnHeight, SWP\_NOZORDER);

}

// Функция переключения видимости страницы

void ShowPage(HWND hwnd, int page)

{

ShowWindow(GetDlgItem(hwnd, ID\_STATIC\_LABEL), SW\_HIDE);

ShowWindow(GetDlgItem(hwnd, ID\_BTN\_START), SW\_HIDE);

ShowWindow(GetDlgItem(hwnd, ID\_BTN\_EXIT), SW\_HIDE);

ShowWindow(GetDlgItem(hwnd, ID\_BTN\_BACK), SW\_HIDE);

ShowWindow(GetDlgItem(hwnd, ID\_STATIC\_CREDIT), SW\_HIDE);

ShowWindow(GetDlgItem(hwnd, ID\_BTN\_INPUT\_FIRST), SW\_HIDE);

ShowWindow(GetDlgItem(hwnd, ID\_BTN\_INPUT\_SECOND), SW\_HIDE);

ShowWindow(GetDlgItem(hwnd, ID\_STATIC\_TYPE\_FIRST), SW\_HIDE);

ShowWindow(GetDlgItem(hwnd, ID\_STATIC\_TYPE\_SECOND), SW\_HIDE);

ShowWindow(GetDlgItem(hwnd, ID\_STATIC\_RESULT\_FIRST), SW\_HIDE);

ShowWindow(GetDlgItem(hwnd, ID\_STATIC\_RESULT\_SECOND), SW\_HIDE);

ShowWindow(GetDlgItem(hwnd, ID\_BTN\_COPY\_FIRST), SW\_HIDE);

ShowWindow(GetDlgItem(hwnd, ID\_BTN\_COPY\_SECOND), SW\_HIDE);

ShowWindow(GetDlgItem(hwnd, ID\_BTN\_DETERMINANT\_FIRST), SW\_HIDE);

ShowWindow(GetDlgItem(hwnd, ID\_BTN\_INVERSE\_FIRST), SW\_HIDE);

ShowWindow(GetDlgItem(hwnd, ID\_BTN\_TRANSPOSE\_FIRST), SW\_HIDE);

ShowWindow(GetDlgItem(hwnd, ID\_BTN\_RANK\_FIRST), SW\_HIDE);

ShowWindow(GetDlgItem(hwnd, ID\_BTN\_MULTIPLY\_FIRST), SW\_HIDE);

ShowWindow(GetDlgItem(hwnd, ID\_BTN\_POWER\_FIRST), SW\_HIDE);

ShowWindow(GetDlgItem(hwnd, ID\_BTN\_DETERMINANT\_SECOND), SW\_HIDE);

ShowWindow(GetDlgItem(hwnd, ID\_BTN\_INVERSE\_SECOND), SW\_HIDE);

ShowWindow(GetDlgItem(hwnd, ID\_BTN\_TRANSPOSE\_SECOND), SW\_HIDE);

ShowWindow(GetDlgItem(hwnd, ID\_BTN\_RANK\_SECOND), SW\_HIDE);

ShowWindow(GetDlgItem(hwnd, ID\_BTN\_MULTIPLY\_SECOND), SW\_HIDE);

ShowWindow(GetDlgItem(hwnd, ID\_BTN\_POWER\_SECOND), SW\_HIDE);

ShowWindow(GetDlgItem(hwnd, ID\_BTN\_LENGTH\_FIRST), SW\_HIDE);

ShowWindow(GetDlgItem(hwnd, ID\_BTN\_MULTIPLY\_VECTOR\_FIRST), SW\_HIDE);

ShowWindow(GetDlgItem(hwnd, ID\_BTN\_POWER\_VECTOR\_FIRST), SW\_HIDE);

ShowWindow(GetDlgItem(hwnd, ID\_BTN\_LENGTH\_SECOND), SW\_HIDE);

ShowWindow(GetDlgItem(hwnd, ID\_BTN\_MULTIPLY\_VECTOR\_SECOND), SW\_HIDE);

ShowWindow(GetDlgItem(hwnd, ID\_BTN\_POWER\_VECTOR\_SECOND), SW\_HIDE);

ShowWindow(GetDlgItem(hwnd, ID\_BTN\_MATRIX\_MULTIPLY), SW\_HIDE);

ShowWindow(GetDlgItem(hwnd, ID\_BTN\_MATRIX\_ADD), SW\_HIDE);

ShowWindow(GetDlgItem(hwnd, ID\_BTN\_MATRIX\_SUBTRACT), SW\_HIDE);

ShowWindow(GetDlgItem(hwnd, ID\_BTN\_VECTOR\_ADD), SW\_HIDE);

ShowWindow(GetDlgItem(hwnd, ID\_BTN\_VECTOR\_SUBTRACT), SW\_HIDE);

ShowWindow(GetDlgItem(hwnd, ID\_BTN\_VECTOR\_DOT), SW\_HIDE);

ShowWindow(GetDlgItem(hwnd, ID\_BTN\_VECTOR\_CROSS), SW\_HIDE);

ShowWindow(GetDlgItem(hwnd, ID\_BTN\_VECTOR\_MIXED), SW\_HIDE);

ShowWindow(GetDlgItem(hwnd, ID\_BTN\_MATRIX\_VECTOR\_MULTIPLY), SW\_HIDE);

if (page == 0)

{

ShowWindow(GetDlgItem(hwnd, ID\_STATIC\_LABEL), SW\_SHOW);

ShowWindow(GetDlgItem(hwnd, ID\_BTN\_START), SW\_SHOW);

ShowWindow(GetDlgItem(hwnd, ID\_BTN\_EXIT), SW\_SHOW);

ShowWindow(GetDlgItem(hwnd, ID\_STATIC\_CREDIT), SW\_SHOW);

}

else {

ShowWindow(GetDlgItem(hwnd, ID\_BTN\_BACK), SW\_SHOW);

ShowWindow(GetDlgItem(hwnd, ID\_BTN\_INPUT\_FIRST), SW\_SHOW);

ShowWindow(GetDlgItem(hwnd, ID\_BTN\_INPUT\_SECOND), SW\_SHOW);

ShowWindow(GetDlgItem(hwnd, ID\_STATIC\_TYPE\_FIRST), SW\_SHOW);

ShowWindow(GetDlgItem(hwnd, ID\_STATIC\_TYPE\_SECOND), SW\_SHOW);

ShowWindow(GetDlgItem(hwnd, ID\_STATIC\_RESULT\_FIRST), SW\_SHOW);

ShowWindow(GetDlgItem(hwnd, ID\_STATIC\_RESULT\_SECOND), SW\_SHOW);

if (showCopyFirst)

{

ShowWindow(GetDlgItem(hwnd, ID\_BTN\_COPY\_FIRST), SW\_SHOW);

}

if (showCopySecond)

{

ShowWindow(GetDlgItem(hwnd, ID\_BTN\_COPY\_SECOND), SW\_SHOW);

}

if (showMatrixButtonsFirst)

{

ShowWindow(GetDlgItem(hwnd, ID\_BTN\_DETERMINANT\_FIRST), SW\_SHOW);

ShowWindow(GetDlgItem(hwnd, ID\_BTN\_INVERSE\_FIRST), SW\_SHOW);

ShowWindow(GetDlgItem(hwnd, ID\_BTN\_TRANSPOSE\_FIRST), SW\_SHOW);

ShowWindow(GetDlgItem(hwnd, ID\_BTN\_RANK\_FIRST), SW\_SHOW);

ShowWindow(GetDlgItem(hwnd, ID\_BTN\_MULTIPLY\_FIRST), SW\_SHOW);

ShowWindow(GetDlgItem(hwnd, ID\_BTN\_POWER\_FIRST), SW\_SHOW);

}

if (showMatrixButtonsSecond)

{

ShowWindow(GetDlgItem(hwnd, ID\_BTN\_DETERMINANT\_SECOND), SW\_SHOW);

ShowWindow(GetDlgItem(hwnd, ID\_BTN\_INVERSE\_SECOND), SW\_SHOW);

ShowWindow(GetDlgItem(hwnd, ID\_BTN\_TRANSPOSE\_SECOND), SW\_SHOW);

ShowWindow(GetDlgItem(hwnd, ID\_BTN\_RANK\_SECOND), SW\_SHOW);

ShowWindow(GetDlgItem(hwnd, ID\_BTN\_MULTIPLY\_SECOND), SW\_SHOW);

ShowWindow(GetDlgItem(hwnd, ID\_BTN\_POWER\_SECOND), SW\_SHOW);

}

if (showVectorButtonsFirst)

{

ShowWindow(GetDlgItem(hwnd, ID\_BTN\_LENGTH\_FIRST), SW\_SHOW);

ShowWindow(GetDlgItem(hwnd, ID\_BTN\_MULTIPLY\_VECTOR\_FIRST), SW\_SHOW);

ShowWindow(GetDlgItem(hwnd, ID\_BTN\_POWER\_VECTOR\_FIRST), SW\_SHOW);

}

if (showVectorButtonsSecond)

{

ShowWindow(GetDlgItem(hwnd, ID\_BTN\_LENGTH\_SECOND), SW\_SHOW);

ShowWindow(GetDlgItem(hwnd, ID\_BTN\_MULTIPLY\_VECTOR\_SECOND), SW\_SHOW);

ShowWindow(GetDlgItem(hwnd, ID\_BTN\_POWER\_VECTOR\_SECOND), SW\_SHOW);

}

if (showMatrixButtonsFirst && showMatrixButtonsSecond)

{

ShowWindow(GetDlgItem(hwnd, ID\_BTN\_MATRIX\_MULTIPLY), SW\_SHOW);

ShowWindow(GetDlgItem(hwnd, ID\_BTN\_MATRIX\_ADD), SW\_SHOW);

ShowWindow(GetDlgItem(hwnd, ID\_BTN\_MATRIX\_SUBTRACT), SW\_SHOW);

}

if (showVectorButtonsFirst && showVectorButtonsSecond)

{

ShowWindow(GetDlgItem(hwnd, ID\_BTN\_VECTOR\_ADD), SW\_SHOW);

ShowWindow(GetDlgItem(hwnd, ID\_BTN\_VECTOR\_SUBTRACT), SW\_SHOW);

ShowWindow(GetDlgItem(hwnd, ID\_BTN\_VECTOR\_DOT), SW\_SHOW);

ShowWindow(GetDlgItem(hwnd, ID\_BTN\_VECTOR\_CROSS), SW\_SHOW);

ShowWindow(GetDlgItem(hwnd, ID\_BTN\_VECTOR\_MIXED), SW\_SHOW);

}

if ((showMatrixButtonsFirst && showVectorButtonsSecond) || (showVectorButtonsFirst && showMatrixButtonsSecond))

{

ShowWindow(GetDlgItem(hwnd, ID\_BTN\_MATRIX\_VECTOR\_MULTIPLY), SW\_SHOW);

}

}

}

// Функция обновления страницы при вводе элементов

void UpdateResultStatic(HWND hwnd, int buttonId, const std::wstring& result, const std::wstring& type)

{

if (buttonId == ID\_BTN\_INPUT\_FIRST)

{

ShowWindow(GetDlgItem(hwnd, ID\_BTN\_DETERMINANT\_FIRST), SW\_HIDE);

ShowWindow(GetDlgItem(hwnd, ID\_BTN\_INVERSE\_FIRST), SW\_HIDE);

ShowWindow(GetDlgItem(hwnd, ID\_BTN\_TRANSPOSE\_FIRST), SW\_HIDE);

ShowWindow(GetDlgItem(hwnd, ID\_BTN\_RANK\_FIRST), SW\_HIDE);

ShowWindow(GetDlgItem(hwnd, ID\_BTN\_MULTIPLY\_FIRST), SW\_HIDE);

ShowWindow(GetDlgItem(hwnd, ID\_BTN\_POWER\_FIRST), SW\_HIDE);

ShowWindow(GetDlgItem(hwnd, ID\_BTN\_LENGTH\_FIRST), SW\_HIDE);

ShowWindow(GetDlgItem(hwnd, ID\_BTN\_MULTIPLY\_VECTOR\_FIRST), SW\_HIDE);

ShowWindow(GetDlgItem(hwnd, ID\_BTN\_POWER\_VECTOR\_FIRST), SW\_HIDE);

InvalidateRect(hwnd, NULL, TRUE);

SetWindowText(GetDlgItem(hwnd, ID\_STATIC\_TYPE\_FIRST), type.c\_str());

SetWindowText(GetDlgItem(hwnd, ID\_STATIC\_RESULT\_FIRST), result.c\_str());

ShowWindow(GetDlgItem(hwnd, ID\_BTN\_COPY\_FIRST), SW\_SHOW);

showCopyFirst = true;

if (type == L"Матрица")

{

ShowWindow(GetDlgItem(hwnd, ID\_BTN\_DETERMINANT\_FIRST), SW\_SHOW);

ShowWindow(GetDlgItem(hwnd, ID\_BTN\_INVERSE\_FIRST), SW\_SHOW);

ShowWindow(GetDlgItem(hwnd, ID\_BTN\_TRANSPOSE\_FIRST), SW\_SHOW);

ShowWindow(GetDlgItem(hwnd, ID\_BTN\_RANK\_FIRST), SW\_SHOW);

ShowWindow(GetDlgItem(hwnd, ID\_BTN\_MULTIPLY\_FIRST), SW\_SHOW);

ShowWindow(GetDlgItem(hwnd, ID\_BTN\_POWER\_FIRST), SW\_SHOW);

showMatrixButtonsFirst = true;

showVectorButtonsFirst = false;

}

else if (type == L"Вектор")

{

ShowWindow(GetDlgItem(hwnd, ID\_BTN\_LENGTH\_FIRST), SW\_SHOW);

ShowWindow(GetDlgItem(hwnd, ID\_BTN\_MULTIPLY\_VECTOR\_FIRST), SW\_SHOW);

ShowWindow(GetDlgItem(hwnd, ID\_BTN\_POWER\_VECTOR\_FIRST), SW\_SHOW);

showVectorButtonsFirst = true;

showMatrixButtonsFirst = false;

}

}

else if (buttonId == ID\_BTN\_INPUT\_SECOND)

{

ShowWindow(GetDlgItem(hwnd, ID\_BTN\_DETERMINANT\_SECOND), SW\_HIDE);

ShowWindow(GetDlgItem(hwnd, ID\_BTN\_INVERSE\_SECOND), SW\_HIDE);

ShowWindow(GetDlgItem(hwnd, ID\_BTN\_TRANSPOSE\_SECOND), SW\_HIDE);

ShowWindow(GetDlgItem(hwnd, ID\_BTN\_RANK\_SECOND), SW\_HIDE);

ShowWindow(GetDlgItem(hwnd, ID\_BTN\_MULTIPLY\_SECOND), SW\_HIDE);

ShowWindow(GetDlgItem(hwnd, ID\_BTN\_POWER\_SECOND), SW\_HIDE);

ShowWindow(GetDlgItem(hwnd, ID\_BTN\_LENGTH\_SECOND), SW\_HIDE);

ShowWindow(GetDlgItem(hwnd, ID\_BTN\_MULTIPLY\_VECTOR\_SECOND), SW\_HIDE);

ShowWindow(GetDlgItem(hwnd, ID\_BTN\_POWER\_VECTOR\_SECOND), SW\_HIDE);

InvalidateRect(hwnd, NULL, TRUE);

SetWindowText(GetDlgItem(hwnd, ID\_STATIC\_TYPE\_SECOND), type.c\_str());

SetWindowText(GetDlgItem(hwnd, ID\_STATIC\_RESULT\_SECOND), result.c\_str());

ShowWindow(GetDlgItem(hwnd, ID\_BTN\_COPY\_SECOND), SW\_SHOW);

showCopySecond = true;

if (type == L"Матрица")

{

ShowWindow(GetDlgItem(hwnd, ID\_BTN\_DETERMINANT\_SECOND), SW\_SHOW);

ShowWindow(GetDlgItem(hwnd, ID\_BTN\_INVERSE\_SECOND), SW\_SHOW);

ShowWindow(GetDlgItem(hwnd, ID\_BTN\_TRANSPOSE\_SECOND), SW\_SHOW);

ShowWindow(GetDlgItem(hwnd, ID\_BTN\_RANK\_SECOND), SW\_SHOW);

ShowWindow(GetDlgItem(hwnd, ID\_BTN\_MULTIPLY\_SECOND), SW\_SHOW);

ShowWindow(GetDlgItem(hwnd, ID\_BTN\_POWER\_SECOND), SW\_SHOW);

showMatrixButtonsSecond = true;

showVectorButtonsSecond = false;

}

else if (type == L"Вектор")

{

ShowWindow(GetDlgItem(hwnd, ID\_BTN\_LENGTH\_SECOND), SW\_SHOW);

ShowWindow(GetDlgItem(hwnd, ID\_BTN\_MULTIPLY\_VECTOR\_SECOND), SW\_SHOW);

ShowWindow(GetDlgItem(hwnd, ID\_BTN\_POWER\_VECTOR\_SECOND), SW\_SHOW);

showVectorButtonsSecond = true;

showMatrixButtonsSecond = false;

}

}

if (showMatrixButtonsFirst && showMatrixButtonsSecond)

{

ShowWindow(GetDlgItem(hwnd, ID\_BTN\_VECTOR\_ADD), SW\_HIDE);

ShowWindow(GetDlgItem(hwnd, ID\_BTN\_VECTOR\_SUBTRACT), SW\_HIDE);

ShowWindow(GetDlgItem(hwnd, ID\_BTN\_VECTOR\_DOT), SW\_HIDE);

ShowWindow(GetDlgItem(hwnd, ID\_BTN\_VECTOR\_CROSS), SW\_HIDE);

ShowWindow(GetDlgItem(hwnd, ID\_BTN\_VECTOR\_MIXED), SW\_HIDE);

ShowWindow(GetDlgItem(hwnd, ID\_BTN\_MATRIX\_VECTOR\_MULTIPLY), SW\_HIDE);

ShowWindow(GetDlgItem(hwnd, ID\_BTN\_MATRIX\_MULTIPLY), SW\_SHOW);

ShowWindow(GetDlgItem(hwnd, ID\_BTN\_MATRIX\_ADD), SW\_SHOW);

ShowWindow(GetDlgItem(hwnd, ID\_BTN\_MATRIX\_SUBTRACT), SW\_SHOW);

}

else if (showVectorButtonsFirst && showVectorButtonsSecond)

{

ShowWindow(GetDlgItem(hwnd, ID\_BTN\_MATRIX\_MULTIPLY), SW\_HIDE);

ShowWindow(GetDlgItem(hwnd, ID\_BTN\_MATRIX\_ADD), SW\_HIDE);

ShowWindow(GetDlgItem(hwnd, ID\_BTN\_MATRIX\_SUBTRACT), SW\_HIDE);

ShowWindow(GetDlgItem(hwnd, ID\_BTN\_MATRIX\_VECTOR\_MULTIPLY), SW\_HIDE);

ShowWindow(GetDlgItem(hwnd, ID\_BTN\_VECTOR\_ADD), SW\_SHOW);

ShowWindow(GetDlgItem(hwnd, ID\_BTN\_VECTOR\_SUBTRACT), SW\_SHOW);

ShowWindow(GetDlgItem(hwnd, ID\_BTN\_VECTOR\_DOT), SW\_SHOW);

ShowWindow(GetDlgItem(hwnd, ID\_BTN\_VECTOR\_CROSS), SW\_SHOW);

ShowWindow(GetDlgItem(hwnd, ID\_BTN\_VECTOR\_MIXED), SW\_HIDE); //no 3rd vector

}

else if ((showMatrixButtonsFirst && showVectorButtonsSecond) || (showVectorButtonsFirst && showMatrixButtonsSecond))

{

ShowWindow(GetDlgItem(hwnd, ID\_BTN\_VECTOR\_ADD), SW\_HIDE);

ShowWindow(GetDlgItem(hwnd, ID\_BTN\_VECTOR\_SUBTRACT), SW\_HIDE);

ShowWindow(GetDlgItem(hwnd, ID\_BTN\_VECTOR\_DOT), SW\_HIDE);

ShowWindow(GetDlgItem(hwnd, ID\_BTN\_VECTOR\_CROSS), SW\_HIDE);

ShowWindow(GetDlgItem(hwnd, ID\_BTN\_VECTOR\_MIXED), SW\_HIDE);

ShowWindow(GetDlgItem(hwnd, ID\_BTN\_MATRIX\_MULTIPLY), SW\_HIDE);

ShowWindow(GetDlgItem(hwnd, ID\_BTN\_MATRIX\_ADD), SW\_HIDE);

ShowWindow(GetDlgItem(hwnd, ID\_BTN\_MATRIX\_SUBTRACT), SW\_HIDE);

ShowWindow(GetDlgItem(hwnd, ID\_BTN\_MATRIX\_VECTOR\_MULTIPLY), SW\_HIDE); //do not exist

}

}

// Функция отображения диалога ввода

void ShowInputDialog(HWND hwnd, int buttonId)

{

DialogBoxParam(hInst, MAKEINTRESOURCE(ID\_DIALOG), hwnd, DialogProc, buttonId);

}

// Функция копирования текста в буфер обмена

void CopyToClipboard(HWND hwnd)

{

int len = GetWindowTextLength(hwnd);

if (len > 0)

{

HGLOBAL hMem = GlobalAlloc(GHND, (len + 1) \* sizeof(TCHAR));

if (hMem)

{

LPTSTR pMem = (LPTSTR)GlobalLock(hMem);

if (pMem)

{

GetWindowText(hwnd, pMem, len + 1);

GlobalUnlock(hMem);

OpenClipboard(hwnd);

EmptyClipboard();

SetClipboardData(CF\_UNICODETEXT, hMem);

CloseClipboard();

}

}

}

}

INT\_PTR CALLBACK TransposeResultDlgProc(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam)

{

static std::wstring resultText;

static int elementId;

switch (message)

{

case WM\_INITDIALOG:

{

std::pair<std::wstring\*, int>\* params = (std::pair<std::wstring\*, int>\*)lParam;

resultText = \*(params->first);

elementId = params->second;

SetDlgItemText(hDlg, ID\_STATIC\_RESULT, resultText.c\_str());

SetNewFont(hDlg);

}

return (INT\_PTR)TRUE;

case WM\_COMMAND:

switch (LOWORD(wParam))

{

case ID\_BTN\_COPY:

if (OpenClipboard(hDlg))

{

EmptyClipboard();

size\_t size = (resultText.size() + 1) \* sizeof(wchar\_t);

HGLOBAL hMem = GlobalAlloc(GMEM\_MOVEABLE, size);

memcpy(GlobalLock(hMem), resultText.c\_str(), size);

GlobalUnlock(hMem);

SetClipboardData(CF\_UNICODETEXT, hMem);

CloseClipboard();

}

break;

case ID\_BTN\_PASTE\_FIRST:

if (elementId == ID\_BTN\_TRANSPOSE\_FIRST)

{

globalMatrixFirst = stringToMatrix(resultText, globalMatrixFirst.size(), globalMatrixFirst[0].size());

UpdateResultStatic(GetParent(hDlg), ID\_BTN\_INPUT\_FIRST, resultText, L"Матрица");

}

else if (elementId == ID\_BTN\_TRANSPOSE\_SECOND)

{

globalMatrixSecond = stringToMatrix(resultText, globalMatrixSecond.size(), globalMatrixSecond[0].size());

UpdateResultStatic(GetParent(hDlg), ID\_BTN\_INPUT\_SECOND, resultText, L"Матрица");

}

break;

case ID\_BTN\_OK:

EndDialog(hDlg, LOWORD(wParam));

return (INT\_PTR)TRUE;

}

break;

}

return (INT\_PTR)FALSE;

}

INT\_PTR CALLBACK ScalarInputDlgProc(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam)

{

static double\* scalar;

switch (message)

{

case WM\_INITDIALOG:

scalar = (double\*)lParam;

return (INT\_PTR)TRUE;

SetNewFont(hDlg);

case WM\_COMMAND:

switch (LOWORD(wParam))

{

case ID\_BTN\_OK:

{

wchar\_t buffer[1024];

GetDlgItemText(hDlg, ID\_EDIT\_SCALAR, buffer, 1024);

\*scalar = \_wtof(buffer);

EndDialog(hDlg, LOWORD(wParam));

return (INT\_PTR)TRUE;

}

case ID\_BTN\_CANCEL:

EndDialog(hDlg, LOWORD(wParam));

return (INT\_PTR)TRUE;

}

break;

}

return (INT\_PTR)FALSE;

}

INT\_PTR CALLBACK ScalarResultDlgProc(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam)

{

static std::wstring resultText;

static int elementId;

switch (message)

{

case WM\_INITDIALOG:

{

std::pair<std::wstring\*, int>\* params = (std::pair<std::wstring\*, int>\*)lParam;

resultText = \*(params->first);

elementId = params->second;

SetDlgItemText(hDlg, ID\_STATIC\_RESULT, resultText.c\_str());

SetNewFont(hDlg);

}

return (INT\_PTR)TRUE;

case WM\_COMMAND:

switch (LOWORD(wParam))

{

case ID\_BTN\_COPY:

if (OpenClipboard(hDlg))

{

EmptyClipboard();

size\_t size = (resultText.size() + 1) \* sizeof(wchar\_t);

HGLOBAL hMem = GlobalAlloc(GMEM\_MOVEABLE, size);

memcpy(GlobalLock(hMem), resultText.c\_str(), size);

GlobalUnlock(hMem);

SetClipboardData(CF\_UNICODETEXT, hMem);

CloseClipboard();

}

break;

case ID\_BTN\_PASTE\_FIRST:

if (elementId == ID\_BTN\_MULTIPLY\_FIRST)

{

globalMatrixFirst = stringToMatrix(resultText, globalMatrixFirst.size(), globalMatrixFirst[0].size());

UpdateResultStatic(GetParent(hDlg), ID\_BTN\_INPUT\_FIRST, resultText, L"Матрица");

}

else if (elementId == ID\_BTN\_MULTIPLY\_SECOND)

{

globalMatrixSecond = stringToMatrix(resultText, globalMatrixSecond.size(), globalMatrixSecond[0].size());

UpdateResultStatic(GetParent(hDlg), ID\_BTN\_INPUT\_SECOND, resultText, L"Матрица");

}

else if (elementId == ID\_BTN\_MULTIPLY\_VECTOR\_FIRST)

{

globalVectorFirst = stringToVector(resultText, globalVectorFirst.size());

UpdateResultStatic(GetParent(hDlg), ID\_BTN\_INPUT\_FIRST, resultText, L"Вектор");

}

else if (elementId == ID\_BTN\_MULTIPLY\_VECTOR\_SECOND)

{

globalVectorSecond = stringToVector(resultText, globalVectorSecond.size());

UpdateResultStatic(GetParent(hDlg), ID\_BTN\_INPUT\_SECOND, resultText, L"Вектор");

}

break;

case ID\_BTN\_OK:

EndDialog(hDlg, LOWORD(wParam));

return (INT\_PTR)TRUE;

}

break;

}

return (INT\_PTR)FALSE;

}

INT\_PTR CALLBACK PowerInputDlgProc(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam)

{

static int\* power;

switch (message)

{

case WM\_INITDIALOG:

power = (int\*)lParam;

return (INT\_PTR)TRUE;

SetNewFont(hDlg);

case WM\_COMMAND:

switch (LOWORD(wParam))

{

case ID\_BTN\_OK:

{

wchar\_t buffer[1024];

GetDlgItemText(hDlg, ID\_EDIT\_POWER, buffer, 1024);

\*power = \_wtoi(buffer);

EndDialog(hDlg, LOWORD(wParam));

return (INT\_PTR)TRUE;

}

case ID\_BTN\_CANCEL:

EndDialog(hDlg, LOWORD(wParam));

return (INT\_PTR)TRUE;

}

break;

}

return (INT\_PTR)FALSE;

}

INT\_PTR CALLBACK PowerResultDlgProc(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam)

{

static std::wstring resultText;

static int elementId;

switch (message)

{

case WM\_INITDIALOG:

{

std::pair<std::wstring\*, int>\* params = (std::pair<std::wstring\*, int>\*)lParam;

resultText = \*(params->first);

elementId = params->second;

SetDlgItemText(hDlg, ID\_STATIC\_POWER\_RESULT, resultText.c\_str());

SetNewFont(hDlg);

}

return (INT\_PTR)TRUE;

case WM\_COMMAND:

switch (LOWORD(wParam))

{

case ID\_BTN\_COPY\_POWER:

if (OpenClipboard(hDlg))

{

EmptyClipboard();

size\_t size = (resultText.size() + 1) \* sizeof(wchar\_t);

HGLOBAL hMem = GlobalAlloc(GMEM\_MOVEABLE, size);

memcpy(GlobalLock(hMem), resultText.c\_str(), size);

GlobalUnlock(hMem);

SetClipboardData(CF\_UNICODETEXT, hMem);

CloseClipboard();

}

break;

case ID\_BTN\_PASTE\_POWER\_FIRST:

if (elementId == ID\_BTN\_POWER\_FIRST)

{

globalMatrixFirst = stringToMatrix(resultText, globalMatrixFirst.size(), globalMatrixFirst[0].size());

UpdateResultStatic(GetParent(hDlg), ID\_BTN\_INPUT\_FIRST, resultText, L"Матрица");

}

else if (elementId == ID\_BTN\_POWER\_SECOND)

{

globalMatrixSecond = stringToMatrix(resultText, globalMatrixSecond.size(), globalMatrixSecond[0].size());

UpdateResultStatic(GetParent(hDlg), ID\_BTN\_INPUT\_SECOND, resultText, L"Матрица");

}

else if (elementId == ID\_BTN\_POWER\_VECTOR\_FIRST)

{

globalVectorFirst = stringToVector(resultText, globalVectorFirst.size());

UpdateResultStatic(GetParent(hDlg), ID\_BTN\_INPUT\_FIRST, resultText, L"Вектор");

}

else if (elementId == ID\_BTN\_POWER\_VECTOR\_SECOND)

{

globalVectorSecond = stringToVector(resultText, globalVectorSecond.size());

UpdateResultStatic(GetParent(hDlg), ID\_BTN\_INPUT\_SECOND, resultText, L"Вектор");

}

break;

case ID\_BTN\_OK:

EndDialog(hDlg, LOWORD(wParam));

return (INT\_PTR)TRUE;

}

break;

}

return (INT\_PTR)FALSE;

}

INT\_PTR CALLBACK DeterminantResultDlgProc(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam)

{

static std::wstring resultText;

static int elementId;

switch (message) {

case WM\_INITDIALOG:

{

std::pair<std::wstring\*, int>\* params = (std::pair<std::wstring\*, int>\*)lParam;

resultText = \*(params->first);

elementId = params->second;

SetDlgItemText(hDlg, ID\_STATIC\_DETERMINANT\_RESULT, resultText.c\_str());

SetNewFont(hDlg);

}

return (INT\_PTR)TRUE;

case WM\_COMMAND:

switch (LOWORD(wParam))

{

case ID\_BTN\_COPY\_DETERMINANT:

if (OpenClipboard(hDlg))

{

EmptyClipboard();

size\_t size = (resultText.size() + 1) \* sizeof(wchar\_t);

HGLOBAL hMem = GlobalAlloc(GMEM\_MOVEABLE, size);

memcpy(GlobalLock(hMem), resultText.c\_str(), size);

GlobalUnlock(hMem);

SetClipboardData(CF\_UNICODETEXT, hMem);

CloseClipboard();

}

break;

case ID\_BTN\_OK:

EndDialog(hDlg, LOWORD(wParam));

return (INT\_PTR)TRUE;

}

break;

}

return (INT\_PTR)FALSE;

}

INT\_PTR CALLBACK VectorLengthResultDlgProc(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam)

{

static std::wstring resultText;

static int elementId;

switch (message)

{

case WM\_INITDIALOG:

{

std::pair<std::wstring\*, int>\* params = (std::pair<std::wstring\*, int>\*)lParam;

resultText = \*(params->first);

elementId = params->second;

SetDlgItemText(hDlg, ID\_STATIC\_VECTOR\_LENGTH\_RESULT, resultText.c\_str());

SetNewFont(hDlg);

}

return (INT\_PTR)TRUE;

case WM\_COMMAND:

switch (LOWORD(wParam))

{

case ID\_BTN\_COPY\_VECTOR\_LENGTH:

if (OpenClipboard(hDlg))

{

EmptyClipboard();

size\_t size = (resultText.size() + 1) \* sizeof(wchar\_t);

HGLOBAL hMem = GlobalAlloc(GMEM\_MOVEABLE, size);

memcpy(GlobalLock(hMem), resultText.c\_str(), size);

GlobalUnlock(hMem);

SetClipboardData(CF\_UNICODETEXT, hMem);

CloseClipboard();

}

break;

case ID\_BTN\_OK:

EndDialog(hDlg, LOWORD(wParam));

return (INT\_PTR)TRUE;

}

break;

}

return (INT\_PTR)FALSE;

}

INT\_PTR CALLBACK InverseResultDlgProc(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam)

{

static std::wstring resultText;

static int elementId;

switch (message)

{

case WM\_INITDIALOG:

{

std::pair<std::wstring\*, int>\* params = (std::pair<std::wstring\*, int>\*)lParam;

resultText = \*(params->first);

elementId = params->second;

SetDlgItemText(hDlg, ID\_STATIC\_INVERSE\_RESULT, resultText.c\_str());

SetNewFont(hDlg);

}

return (INT\_PTR)TRUE;

case WM\_COMMAND:

switch (LOWORD(wParam))

{

case ID\_BTN\_COPY\_INVERSE:

if (OpenClipboard(hDlg))

{

EmptyClipboard();

size\_t size = (resultText.size() + 1) \* sizeof(wchar\_t);

HGLOBAL hMem = GlobalAlloc(GMEM\_MOVEABLE, size);

memcpy(GlobalLock(hMem), resultText.c\_str(), size);

GlobalUnlock(hMem);

SetClipboardData(CF\_UNICODETEXT, hMem);

CloseClipboard();

}

break;

case ID\_BTN\_PASTE\_INVERSE\_FIRST:

if (elementId == ID\_BTN\_INVERSE\_FIRST)

{

globalMatrixFirst = stringToMatrix(resultText, globalMatrixFirst.size(), globalMatrixFirst[0].size());

UpdateResultStatic(GetParent(hDlg), ID\_BTN\_INPUT\_FIRST, resultText, L"Матрица");

}

else if (elementId == ID\_BTN\_INVERSE\_SECOND)

{

globalMatrixSecond = stringToMatrix(resultText, globalMatrixSecond.size(), globalMatrixSecond[0].size());

UpdateResultStatic(GetParent(hDlg), ID\_BTN\_INPUT\_SECOND, resultText, L"Матрица");

}

break;

case ID\_BTN\_OK:

EndDialog(hDlg, LOWORD(wParam));

return (INT\_PTR)TRUE;

}

break;

}

return (INT\_PTR)FALSE;

}

INT\_PTR CALLBACK RankResultDlgProc(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam)

{

static std::wstring resultText;

static int elementId;

switch (message)

{

case WM\_INITDIALOG:

{

std::pair<std::wstring\*, int>\* params = (std::pair<std::wstring\*, int>\*)lParam;

resultText = \*(params->first);

elementId = params->second;

SetDlgItemText(hDlg, ID\_STATIC\_RANK\_RESULT, resultText.c\_str());

SetNewFont(hDlg);

}

return (INT\_PTR)TRUE;

case WM\_COMMAND:

switch (LOWORD(wParam))

{

case ID\_BTN\_COPY\_RANK:

if (OpenClipboard(hDlg))

{

EmptyClipboard();

size\_t size = (resultText.size() + 1) \* sizeof(wchar\_t);

HGLOBAL hMem = GlobalAlloc(GMEM\_MOVEABLE, size);

memcpy(GlobalLock(hMem), resultText.c\_str(), size);

GlobalUnlock(hMem);

SetClipboardData(CF\_UNICODETEXT, hMem);

CloseClipboard();

}

break;

case ID\_BTN\_OK:

EndDialog(hDlg, LOWORD(wParam));

return (INT\_PTR)TRUE;

}

break;

}

return (INT\_PTR)FALSE;

}

INT\_PTR CALLBACK MatrixOpResultDlgProc(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam)

{

static std::wstring resultText;

static int operationType; // 1 for addition, 2 for subtraction, 3 for multiplication

switch (message)

{

case WM\_INITDIALOG:

{

std::pair<std::wstring\*, int>\* params = (std::pair<std::wstring\*, int>\*)lParam;

resultText = \*(params->first);

operationType = params->second;

SetDlgItemText(hDlg, ID\_STATIC\_MATRIX\_OP\_RESULT, resultText.c\_str());

SetNewFont(hDlg);

}

return (INT\_PTR)TRUE;

case WM\_COMMAND:

switch (LOWORD(wParam))

{

case ID\_BTN\_COPY\_MATRIX\_OP:

if (OpenClipboard(hDlg))

{

EmptyClipboard();

size\_t size = (resultText.size() + 1) \* sizeof(wchar\_t);

HGLOBAL hMem = GlobalAlloc(GMEM\_MOVEABLE, size);

memcpy(GlobalLock(hMem), resultText.c\_str(), size);

GlobalUnlock(hMem);

SetClipboardData(CF\_UNICODETEXT, hMem);

CloseClipboard();

}

break;

case ID\_BTN\_PASTE\_MATRIX\_OP\_FIRST:

// Paste result into the first element

globalMatrixFirst = stringToMatrix(resultText, globalMatrixFirst.size(), globalMatrixFirst[0].size());

UpdateResultStatic(GetParent(hDlg), ID\_BTN\_INPUT\_FIRST, resultText, L"Матрица");

break;

case ID\_BTN\_PASTE\_MATRIX\_OP\_SECOND:

// Paste result into the second element

globalMatrixSecond = stringToMatrix(resultText, globalMatrixSecond.size(), globalMatrixSecond[0].size());

UpdateResultStatic(GetParent(hDlg), ID\_BTN\_INPUT\_SECOND, resultText, L"Матрица");

break;

case ID\_BTN\_OK:

EndDialog(hDlg, LOWORD(wParam));

return (INT\_PTR)TRUE;

}

break;

}

return (INT\_PTR)FALSE;

}

INT\_PTR CALLBACK VectorOpResultDlgProc(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam)

{

static std::wstring resultText;

static int operationType; // 1 for addition, 2 for subtraction

switch (message)

{

case WM\_INITDIALOG:

{

std::pair<std::wstring\*, int>\* params = (std::pair<std::wstring\*, int>\*)lParam;

resultText = \*(params->first);

operationType = params->second;

SetDlgItemText(hDlg, ID\_STATIC\_VECTOR\_OP\_RESULT, resultText.c\_str());

SetNewFont(hDlg);

}

return (INT\_PTR)TRUE;

case WM\_COMMAND:

switch (LOWORD(wParam))

{

case ID\_BTN\_COPY\_VECTOR\_OP:

if (OpenClipboard(hDlg))

{

EmptyClipboard();

size\_t size = (resultText.size() + 1) \* sizeof(wchar\_t);

HGLOBAL hMem = GlobalAlloc(GMEM\_MOVEABLE, size);

memcpy(GlobalLock(hMem), resultText.c\_str(), size);

GlobalUnlock(hMem);

SetClipboardData(CF\_UNICODETEXT, hMem);

CloseClipboard();

}

break;

case ID\_BTN\_PASTE\_VECTOR\_OP\_FIRST:

// Paste result into the first element

globalVectorFirst = stringToVector(resultText, globalVectorFirst.size());

UpdateResultStatic(GetParent(hDlg), ID\_BTN\_INPUT\_FIRST, resultText, L"Вектор");

break;

case ID\_BTN\_PASTE\_VECTOR\_OP\_SECOND:

// Paste result into the second element

globalVectorSecond = stringToVector(resultText, globalVectorSecond.size());

UpdateResultStatic(GetParent(hDlg), ID\_BTN\_INPUT\_SECOND, resultText, L"Вектор");

break;

case ID\_BTN\_OK\_VECTOR\_OP:

EndDialog(hDlg, LOWORD(wParam));

return (INT\_PTR)TRUE;

}

break;

}

return (INT\_PTR)FALSE;

}

INT\_PTR CALLBACK VectorProdResultDlgProc(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam)

{

static std::wstring resultText;

switch (message)

{

case WM\_INITDIALOG:

{

resultText = \*(std::wstring\*)lParam;

SetDlgItemText(hDlg, ID\_STATIC\_VECTOR\_PROD\_RESULT, resultText.c\_str());

SetNewFont(hDlg);

}

return (INT\_PTR)TRUE;

case WM\_COMMAND:

switch (LOWORD(wParam))

{

case ID\_BTN\_COPY\_VECTOR\_PROD:

if (OpenClipboard(hDlg))

{

EmptyClipboard();

size\_t size = (resultText.size() + 1) \* sizeof(wchar\_t);

HGLOBAL hMem = GlobalAlloc(GMEM\_MOVEABLE, size);

memcpy(GlobalLock(hMem), resultText.c\_str(), size);

GlobalUnlock(hMem);

SetClipboardData(CF\_UNICODETEXT, hMem);

CloseClipboard();

}

break;

case ID\_BTN\_OK\_VECTOR\_PROD:

EndDialog(hDlg, LOWORD(wParam));

return (INT\_PTR)TRUE;

}

break;

}

return (INT\_PTR)FALSE;

}

LRESULT CALLBACK WindowProc(HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM lParam)

{

static int currentPage = 0;

switch (uMsg)

{

case WM\_CREATE:

CreateControls(hwnd);

UpdateControlPositions(hwnd);

ShowPage(hwnd, currentPage);

return 0;

case WM\_SIZE:

UpdateControlPositions(hwnd);

InvalidateRect(hwnd, NULL, TRUE);

return 0;

case WM\_CTLCOLORSTATIC:

{

HDC hdcStatic = (HDC)wParam;

SetBkMode(hdcStatic, TRANSPARENT);

return (INT\_PTR)GetStockObject(NULL\_BRUSH);

}

case WM\_COMMAND:

switch (LOWORD(wParam))

{

case ID\_BTN\_START:

currentPage = 1;

ShowPage(hwnd, currentPage);

InvalidateRect(hwnd, NULL, TRUE);

break;

case ID\_BTN\_EXIT:

PostQuitMessage(0);

break;

case ID\_BTN\_BACK:

currentPage = 0;

ShowPage(hwnd, currentPage);

InvalidateRect(hwnd, NULL, TRUE);

break;

case ID\_BTN\_INPUT\_FIRST:

case ID\_BTN\_INPUT\_SECOND:

ShowInputDialog(hwnd, LOWORD(wParam));

break;

case ID\_BTN\_TRANSPOSE\_FIRST:

if (!isFirstVector && !globalMatrixFirst.empty())

{

Matrix transposed = transpose(globalMatrixFirst);

std::wstring result = matrixToString(transposed);

std::pair<std::wstring\*, int> params = { &result, ID\_BTN\_TRANSPOSE\_FIRST };

DialogBoxParam(hInst, MAKEINTRESOURCE(IDD\_SCALAR\_RESULT), hwnd, TransposeResultDlgProc, (LPARAM)&params);

}

break;

case ID\_BTN\_TRANSPOSE\_SECOND:

if (!isSecondVector && !globalMatrixSecond.empty())

{

Matrix transposed = transpose(globalMatrixSecond);

std::wstring result = matrixToString(transposed);

std::pair<std::wstring\*, int> params = { &result, ID\_BTN\_TRANSPOSE\_SECOND };

DialogBoxParam(hInst, MAKEINTRESOURCE(IDD\_SCALAR\_RESULT), hwnd, TransposeResultDlgProc, (LPARAM)&params);

}

break;

case ID\_BTN\_MULTIPLY\_VECTOR\_FIRST:

case ID\_BTN\_MULTIPLY\_VECTOR\_SECOND:

case ID\_BTN\_MULTIPLY\_FIRST:

case ID\_BTN\_MULTIPLY\_SECOND:

{

int buttonId = LOWORD(wParam);

bool isVector = (buttonId == ID\_BTN\_MULTIPLY\_VECTOR\_FIRST || buttonId == ID\_BTN\_MULTIPLY\_VECTOR\_SECOND);

double scalar = 0.0;

// Show scalar input dialog

if (DialogBoxParam(hInst, MAKEINTRESOURCE(IDD\_SCALAR\_INPUT), hwnd, ScalarInputDlgProc, (LPARAM)&scalar) == ID\_BTN\_OK)

{

if (buttonId == ID\_BTN\_MULTIPLY\_FIRST || buttonId == ID\_BTN\_MULTIPLY\_VECTOR\_FIRST)

{

if (!isFirstVector && !globalMatrixFirst.empty())

{

Matrix resultMatrix = multiplyMatrixByScalar(globalMatrixFirst, scalar);

std::wstring result = matrixToString(resultMatrix);

std::pair<std::wstring\*, int> params = { &result, buttonId };

DialogBoxParam(hInst, MAKEINTRESOURCE(IDD\_SCALAR\_RESULT), hwnd, ScalarResultDlgProc, (LPARAM)&params);

}

else if (isFirstVector) {

Vector resultVector = multiplyVectorByScalar(globalVectorFirst, scalar);

std::wstring result = vectorToString(resultVector);

std::pair<std::wstring\*, int> params = { &result, buttonId };

DialogBoxParam(hInst, MAKEINTRESOURCE(IDD\_SCALAR\_RESULT), hwnd, ScalarResultDlgProc, (LPARAM)&params);

}

}

else if (buttonId == ID\_BTN\_MULTIPLY\_SECOND || buttonId == ID\_BTN\_MULTIPLY\_VECTOR\_SECOND)

{

if (!isSecondVector && !globalMatrixSecond.empty())

{

Matrix resultMatrix = multiplyMatrixByScalar(globalMatrixSecond, scalar);

std::wstring result = matrixToString(resultMatrix);

std::pair<std::wstring\*, int> params = { &result, buttonId };

DialogBoxParam(hInst, MAKEINTRESOURCE(IDD\_SCALAR\_RESULT), hwnd, ScalarResultDlgProc, (LPARAM)&params);

}

else if (isSecondVector)

{

Vector resultVector = multiplyVectorByScalar(globalVectorSecond, scalar);

std::wstring result = vectorToString(resultVector);

std::pair<std::wstring\*, int> params = { &result, buttonId };

DialogBoxParam(hInst, MAKEINTRESOURCE(IDD\_SCALAR\_RESULT), hwnd, ScalarResultDlgProc, (LPARAM)&params);

}

}

}

}

break;

case ID\_BTN\_POWER\_VECTOR\_FIRST:

case ID\_BTN\_POWER\_VECTOR\_SECOND:

case ID\_BTN\_POWER\_FIRST:

case ID\_BTN\_POWER\_SECOND:

{

int buttonId = LOWORD(wParam);

bool isVector = (buttonId == ID\_BTN\_POWER\_VECTOR\_FIRST || buttonId == ID\_BTN\_POWER\_VECTOR\_SECOND);

int power = 0;

// Show power input dialog

if (DialogBoxParam(hInst, MAKEINTRESOURCE(IDD\_POWER\_INPUT), hwnd, PowerInputDlgProc, (LPARAM)&power) == ID\_BTN\_OK)

{

if (buttonId == ID\_BTN\_POWER\_FIRST || buttonId == ID\_BTN\_POWER\_VECTOR\_FIRST)

{

if (!isFirstVector && !globalMatrixFirst.empty())

{

if (!isSquareMatrix(globalMatrixFirst))

{

MessageBox(hwnd, L"Матрица должна быть квадратной для возведения в степень.", L"Ошибка", MB\_ICONERROR);

break;

}

Matrix resultMatrix = powerMatrix(globalMatrixFirst, power);

std::wstring result = matrixToString(resultMatrix);

std::pair<std::wstring\*, int> params = { &result, buttonId };

DialogBoxParam(hInst, MAKEINTRESOURCE(IDD\_POWER\_RESULT), hwnd, PowerResultDlgProc, (LPARAM)&params);

}

else if (isFirstVector)

{

Vector resultVector = powerVector(globalVectorFirst, power);

std::wstring result = vectorToString(resultVector);

std::pair<std::wstring\*, int> params = { &result, buttonId };

DialogBoxParam(hInst, MAKEINTRESOURCE(IDD\_POWER\_RESULT), hwnd, PowerResultDlgProc, (LPARAM)&params);

}

}

else if (buttonId == ID\_BTN\_POWER\_SECOND || buttonId == ID\_BTN\_POWER\_VECTOR\_SECOND)

{

if (!isSecondVector && !globalMatrixSecond.empty())

{

if (!isSquareMatrix(globalMatrixSecond))

{

MessageBox(hwnd, L"Матрица должна быть квадратной для возведения в степень.", L"Ошибка", MB\_ICONERROR);

break;

}

Matrix resultMatrix = powerMatrix(globalMatrixSecond, power);

std::wstring result = matrixToString(resultMatrix);

std::pair<std::wstring\*, int> params = { &result, buttonId };

DialogBoxParam(hInst, MAKEINTRESOURCE(IDD\_POWER\_RESULT), hwnd, PowerResultDlgProc, (LPARAM)&params);

}

else if (isSecondVector)

{

Vector resultVector = powerVector(globalVectorSecond, power);

std::wstring result = vectorToString(resultVector);

std::pair<std::wstring\*, int> params = { &result, buttonId };

DialogBoxParam(hInst, MAKEINTRESOURCE(IDD\_POWER\_RESULT), hwnd, PowerResultDlgProc, (LPARAM)&params);

}

}

}

}

break;

case ID\_BTN\_DETERMINANT\_FIRST:

case ID\_BTN\_DETERMINANT\_SECOND:

{

int buttonId = LOWORD(wParam);

if ((buttonId == ID\_BTN\_DETERMINANT\_FIRST && !isFirstVector && !globalMatrixFirst.empty()) ||

(buttonId == ID\_BTN\_DETERMINANT\_SECOND && !isSecondVector && !globalMatrixSecond.empty()))

{

const Matrix& matrix = (buttonId == ID\_BTN\_DETERMINANT\_FIRST) ? globalMatrixFirst : globalMatrixSecond;

if (!isSquareMatrix(matrix))

{

MessageBox(hwnd, L"Матрица должна быть квадратной для вычисления определителя.", L"Ошибка", MB\_ICONERROR);

break;

}

try

{

double det = determinant(matrix);

std::wstring result = L"Определитель: " + std::to\_wstring(det);

std::pair<std::wstring\*, int> params = { &result, buttonId };

DialogBoxParam(hInst, MAKEINTRESOURCE(IDD\_DETERMINANT\_RESULT), hwnd, DeterminantResultDlgProc, (LPARAM)&params);

}

catch (const std::exception& e)

{

MessageBox(hwnd, L"Произошла ошибка при вычислении определителя.", L"Ошибка", MB\_ICONERROR);

}

}

}

break;

case ID\_BTN\_LENGTH\_FIRST:

case ID\_BTN\_LENGTH\_SECOND:

{

int buttonId = LOWORD(wParam);

if ((buttonId == ID\_BTN\_LENGTH\_FIRST && isFirstVector && !globalVectorFirst.empty()) ||

(buttonId == ID\_BTN\_LENGTH\_SECOND && isSecondVector && !globalVectorSecond.empty()))

{

const Vector& vec = (buttonId == ID\_BTN\_LENGTH\_FIRST) ? globalVectorFirst : globalVectorSecond;

double length = vectorLength(vec);

std::wstring result = L"Длина вектора: " + std::to\_wstring(length);

std::pair<std::wstring\*, int> params = { &result, buttonId };

DialogBoxParam(hInst, MAKEINTRESOURCE(IDD\_VECTOR\_LENGTH\_RESULT), hwnd, VectorLengthResultDlgProc, (LPARAM)&params);

}

}

break;

case ID\_BTN\_INVERSE\_FIRST:

case ID\_BTN\_INVERSE\_SECOND:

{

int buttonId = LOWORD(wParam);

if ((buttonId == ID\_BTN\_INVERSE\_FIRST && !isFirstVector && !globalMatrixFirst.empty()) ||

(buttonId == ID\_BTN\_INVERSE\_SECOND && !isSecondVector && !globalMatrixSecond.empty()))

{

const Matrix& matrix = (buttonId == ID\_BTN\_INVERSE\_FIRST) ? globalMatrixFirst : globalMatrixSecond;

if (!isSquareMatrix(matrix))

{

MessageBox(hwnd, L"Матрица должна быть квадратной, чтобы существовала ее обратная матрица.", L"Error", MB\_ICONERROR);

break;

}

try

{

Matrix invMatrix = inverse(matrix);

std::wstring result = matrixToString(invMatrix);

std::pair<std::wstring\*, int> params = { &result, buttonId };

DialogBoxParam(hInst, MAKEINTRESOURCE(IDD\_INVERSE\_RESULT), hwnd, InverseResultDlgProc, (LPARAM)&params);

}

catch (const std::runtime\_error& e)

{

MessageBox(hwnd, L"Матрица является вырожденной (определитель равнен 0), у такой матрицы нет обратной матрицы.", L"Error", MB\_ICONERROR);

}

}

}

break;

case ID\_BTN\_RANK\_FIRST:

case ID\_BTN\_RANK\_SECOND:

{

int buttonId = LOWORD(wParam);

if ((buttonId == ID\_BTN\_RANK\_FIRST && !isFirstVector && !globalMatrixFirst.empty()) ||

(buttonId == ID\_BTN\_RANK\_SECOND && !isSecondVector && !globalMatrixSecond.empty()))

{

const Matrix& matrix = (buttonId == ID\_BTN\_RANK\_FIRST) ? globalMatrixFirst : globalMatrixSecond;

try

{

int rank = calculateRank(matrix);

std::wstring result = L"Ранг матрицы: " + std::to\_wstring(rank);

std::pair<std::wstring\*, int> params = { &result, buttonId };

DialogBoxParam(hInst, MAKEINTRESOURCE(IDD\_RANK\_RESULT), hwnd, RankResultDlgProc, (LPARAM)&params);

}

catch (const std::exception& e)

{

MessageBox(hwnd, L"Произошла ошибка при вычислении ранга.", L"Ошибка", MB\_ICONERROR);

}

}

}

break;

case ID\_BTN\_MATRIX\_MULTIPLY:

case ID\_BTN\_MATRIX\_ADD:

case ID\_BTN\_MATRIX\_SUBTRACT:

{

int buttonId = LOWORD(wParam);

if (!isFirstVector && !isSecondVector && !globalMatrixFirst.empty() && !globalMatrixSecond.empty())

{

try

{

Matrix resultMatrix;

if (buttonId == ID\_BTN\_MATRIX\_MULTIPLY)

{

resultMatrix = multiplyMatricesDiff(globalMatrixFirst, globalMatrixSecond);

}

else if (buttonId == ID\_BTN\_MATRIX\_ADD)

{

resultMatrix = addMatrices(globalMatrixFirst, globalMatrixSecond);

}

else if (buttonId == ID\_BTN\_MATRIX\_SUBTRACT)

{

resultMatrix = subtractMatrices(globalMatrixFirst, globalMatrixSecond);

}

std::wstring result = matrixToString(resultMatrix);

std::pair<std::wstring\*, int> params = { &result, buttonId };

DialogBoxParam(hInst, MAKEINTRESOURCE(IDD\_MATRIX\_OP\_RESULT), hwnd, MatrixOpResultDlgProc, (LPARAM)&params);

}

catch (const std::exception& e) {

MessageBox(hwnd, L"Возникла какая-то проблема.", L"Error", MB\_ICONERROR);

}

}

}

break;

case ID\_BTN\_VECTOR\_ADD:

case ID\_BTN\_VECTOR\_SUBTRACT:

{

int buttonId = LOWORD(wParam);

if (isFirstVector && isSecondVector && !globalVectorFirst.empty() && !globalVectorSecond.empty())

{

try

{

Vector resultVector;

if (buttonId == ID\_BTN\_VECTOR\_ADD)

{

resultVector = addVectors(globalVectorFirst, globalVectorSecond);

}

else if (buttonId == ID\_BTN\_VECTOR\_SUBTRACT)

{

resultVector = subtractVectors(globalVectorFirst, globalVectorSecond);

}

std::wstring result = vectorToString(resultVector);

std::pair<std::wstring\*, int> params = { &result, buttonId };

DialogBoxParam(hInst, MAKEINTRESOURCE(IDD\_VECTOR\_OP\_RESULT), hwnd, VectorOpResultDlgProc, (LPARAM)&params);

}

catch (const std::exception& e)

{

MessageBox(hwnd, L"Возникла какая-то проблема.", L"Error", MB\_ICONERROR);

}

}

}

break;

case ID\_BTN\_VECTOR\_DOT:

case ID\_BTN\_VECTOR\_CROSS:

case ID\_BTN\_VECTOR\_MIXED:

{

int buttonId = LOWORD(wParam);

if (isFirstVector && isSecondVector && !globalVectorFirst.empty() && !globalVectorSecond.empty())

{

try

{

std::wstring result;

if (buttonId == ID\_BTN\_VECTOR\_DOT)

{

double dotProd = dotProduct(globalVectorFirst, globalVectorSecond);

result = L"Скалярное произведение: " + std::to\_wstring(dotProd);

}

else if (buttonId == ID\_BTN\_VECTOR\_CROSS)

{

Vector crossProd = crossProduct(globalVectorFirst, globalVectorSecond);

result = L"Векторное произведение: " + vectorToString(crossProd);

}

else if (buttonId == ID\_BTN\_VECTOR\_MIXED)

{

double mixedProd = mixedProduct(globalVectorFirst, globalVectorSecond, globalVectorSecond);

result = L"Смешанное произведение: " + std::to\_wstring(mixedProd);

}

DialogBoxParam(hInst, MAKEINTRESOURCE(IDD\_VECTOR\_PROD\_RESULT), hwnd, VectorProdResultDlgProc, (LPARAM)&result);

}

catch (const std::exception& e) {

MessageBox(hwnd, L"Возникла какая-то проблема.", L"Error", MB\_ICONERROR);

}

}

}

break;

}

return 0;

case WM\_GETMINMAXINFO:

{

MINMAXINFO\* pInfo = (MINMAXINFO\*)lParam;

pInfo->ptMinTrackSize.x = 1200;

pInfo->ptMinTrackSize.y = 800;

return 0;

}

case WM\_DESTROY:

if (pImage)

{

delete pImage;

}

PostQuitMessage(0);

return 0;

case WM\_PAINT:

{

PAINTSTRUCT ps;

HDC hdc = BeginPaint(hwnd, &ps);

HBRUSH hBrush = CreateSolidBrush(RGB(255, 255, 255));

FillRect(hdc, &ps.rcPaint, hBrush);

DeleteObject(hBrush);

if (currentPage == 0 && pImage != nullptr)

{

Gdiplus::Graphics graphics(hdc);

int imgWidth = pImage->GetWidth() / 2.5;

int imgHeight = pImage->GetHeight() / 2.5;

int x = (ps.rcPaint.right - imgWidth) / 2;

int y = 20;

graphics.DrawImage(pImage, x, y, imgWidth, imgHeight);

}

if (currentPage == 1 && pImage != nullptr)

{

Gdiplus::Graphics graphics(hdc);

int imgWidth = pImage->GetWidth() / 2.5;

int imgHeight = pImage->GetHeight() / 2.5;

int x = (ps.rcPaint.right - imgWidth) / 2 + 30;

int y = 20;

graphics.DrawImage(pImage, x, y, imgWidth, imgHeight);

}

EndPaint(hwnd, &ps);

}

return 0;

}

return DefWindowProc(hwnd, uMsg, wParam, lParam);

}

INT\_PTR CALLBACK DialogProc(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam) {

static int step = 0;

static int buttonId;

static BOOL isVector;

static HWND hwndParent;

HWND hChild = GetWindow(hDlg, GW\_CHILD);

switch (message) {

case WM\_INITDIALOG:

step = 0;

buttonId = lParam;

hwndParent = GetParent(hDlg);

CheckRadioButton(hDlg, ID\_RADIO\_VECTOR, ID\_RADIO\_MATRIX, ID\_RADIO\_VECTOR);

ShowWindow(GetDlgItem(hDlg, ID\_EDIT\_ROWS), SW\_HIDE);

ShowWindow(GetDlgItem(hDlg, ID\_EDIT\_COLS), SW\_HIDE);

ShowWindow(GetDlgItem(hDlg, ID\_STATIC\_ROWS), SW\_HIDE);

ShowWindow(GetDlgItem(hDlg, ID\_STATIC\_COLS), SW\_HIDE);

ShowWindow(GetDlgItem(hDlg, ID\_EDIT\_INPUT), SW\_HIDE);

ShowWindow(GetDlgItem(hDlg, ID\_STATIC\_INPUT), SW\_HIDE);

SetNewFont(hDlg);

return (INT\_PTR)TRUE;

case WM\_COMMAND:

if (LOWORD(wParam) == ID\_BTN\_NEXT)

{

if (step == 0)

{

isVector = IsDlgButtonChecked(hDlg, ID\_RADIO\_VECTOR);

step = 1;

ShowWindow(GetDlgItem(hDlg, ID\_EDIT\_ROWS), SW\_SHOW);

ShowWindow(GetDlgItem(hDlg, ID\_STATIC\_ROWS), SW\_SHOW);

if (isVector)

{

SetDlgItemText(hDlg, ID\_STATIC\_ROWS, L"Введите размерность вектора:");

}

else

{

ShowWindow(GetDlgItem(hDlg, ID\_EDIT\_COLS), SW\_SHOW);

ShowWindow(GetDlgItem(hDlg, ID\_STATIC\_COLS), SW\_SHOW);

SetDlgItemText(hDlg, ID\_STATIC\_ROWS, L"Введите количество строк:");

SetDlgItemText(hDlg, ID\_STATIC\_COLS, L"Введите количество столбцов:");

}

}

else if (step == 1)

{

wchar\_t buffer[10];

GetDlgItemText(hDlg, ID\_EDIT\_ROWS, buffer, 10);

int rows = \_wtoi(buffer);

if (isVector)

{

if (buttonId == ID\_BTN\_INPUT\_FIRST)

{

globalVectorFirst = Vector(rows);

isFirstVector = true;

}

else if (buttonId == ID\_BTN\_INPUT\_SECOND)

{

globalVectorSecond = Vector(rows);

isSecondVector = true;

}

}

else

{

GetDlgItemText(hDlg, ID\_EDIT\_COLS, buffer, 10);

int cols = \_wtoi(buffer);

if (buttonId == ID\_BTN\_INPUT\_FIRST)

{

globalMatrixFirst = Matrix(rows, std::vector<double>(cols));

isFirstVector = false;

}

else if (buttonId == ID\_BTN\_INPUT\_SECOND)

{

globalMatrixSecond = Matrix(rows, std::vector<double>(cols));

isSecondVector = false;

}

}

step = 2;

ShowWindow(GetDlgItem(hDlg, ID\_EDIT\_ROWS), SW\_HIDE);

ShowWindow(GetDlgItem(hDlg, ID\_EDIT\_COLS), SW\_HIDE);

ShowWindow(GetDlgItem(hDlg, ID\_STATIC\_ROWS), SW\_HIDE);

ShowWindow(GetDlgItem(hDlg, ID\_STATIC\_COLS), SW\_HIDE);

ShowWindow(GetDlgItem(hDlg, ID\_EDIT\_INPUT), SW\_SHOW);

ShowWindow(GetDlgItem(hDlg, ID\_STATIC\_INPUT), SW\_SHOW);

if (isVector)

{

SetDlgItemText(hDlg, ID\_STATIC\_INPUT, L"Введите элементы вектора:");

}

else

{

SetDlgItemText(hDlg, ID\_STATIC\_INPUT, L"Введите элементы матрицы (построчно):");

}

}

return (INT\_PTR)TRUE;

}

else if (LOWORD(wParam) == ID\_BTN\_OK)

{

if (step != 2)

{

MessageBox(hDlg, L"Пожалуйста, завершите ввод данных.", L"Ошибка", MB\_OK | MB\_ICONERROR);

return (INT\_PTR)TRUE;

}

wchar\_t inputBuffer[1024];

GetDlgItemText(hDlg, ID\_EDIT\_INPUT, inputBuffer, 1024);

try

{

std::wstring result;

std::wstring type;

if (isVector)

{

if (buttonId == ID\_BTN\_INPUT\_FIRST)

{

globalVectorFirst = stringToVector(inputBuffer, globalVectorFirst.size());

result = vectorToString(globalVectorFirst);

type = L"Вектор";

}

else if (buttonId == ID\_BTN\_INPUT\_SECOND)

{

globalVectorSecond = stringToVector(inputBuffer, globalVectorSecond.size());

result = vectorToString(globalVectorSecond);

type = L"Вектор";

}

}

else

{

int rows = (buttonId == ID\_BTN\_INPUT\_FIRST) ? globalMatrixFirst.size() : globalMatrixSecond.size();

int cols = (buttonId == ID\_BTN\_INPUT\_FIRST) ? globalMatrixFirst[0].size() : globalMatrixSecond[0].size();

if (buttonId == ID\_BTN\_INPUT\_FIRST)

{

globalMatrixFirst = stringToMatrix(inputBuffer, rows, cols);

result = matrixToString(globalMatrixFirst);

type = L"Матрица";

}

else if (buttonId == ID\_BTN\_INPUT\_SECOND)

{

globalMatrixSecond = stringToMatrix(inputBuffer, rows, cols);

result = matrixToString(globalMatrixSecond);

type = L"Матрица";

}

}

UpdateResultStatic(hwndParent, buttonId, result, type);

}

catch (const std::exception& e)

{

MessageBox(hDlg, L"Ошибка ввода данных", L"Ошибка", MB\_OK | MB\_ICONERROR);

return (INT\_PTR)TRUE;

}

EndDialog(hDlg, LOWORD(wParam));

return (INT\_PTR)TRUE;

}

else if (LOWORD(wParam) == ID\_BTN\_CANCEL)

{

EndDialog(hDlg, LOWORD(wParam));

return (INT\_PTR)TRUE;

}

break;

}

return (INT\_PTR)FALSE;

}

Результаты работы

При запуске программы появляется начальный экран приложения.

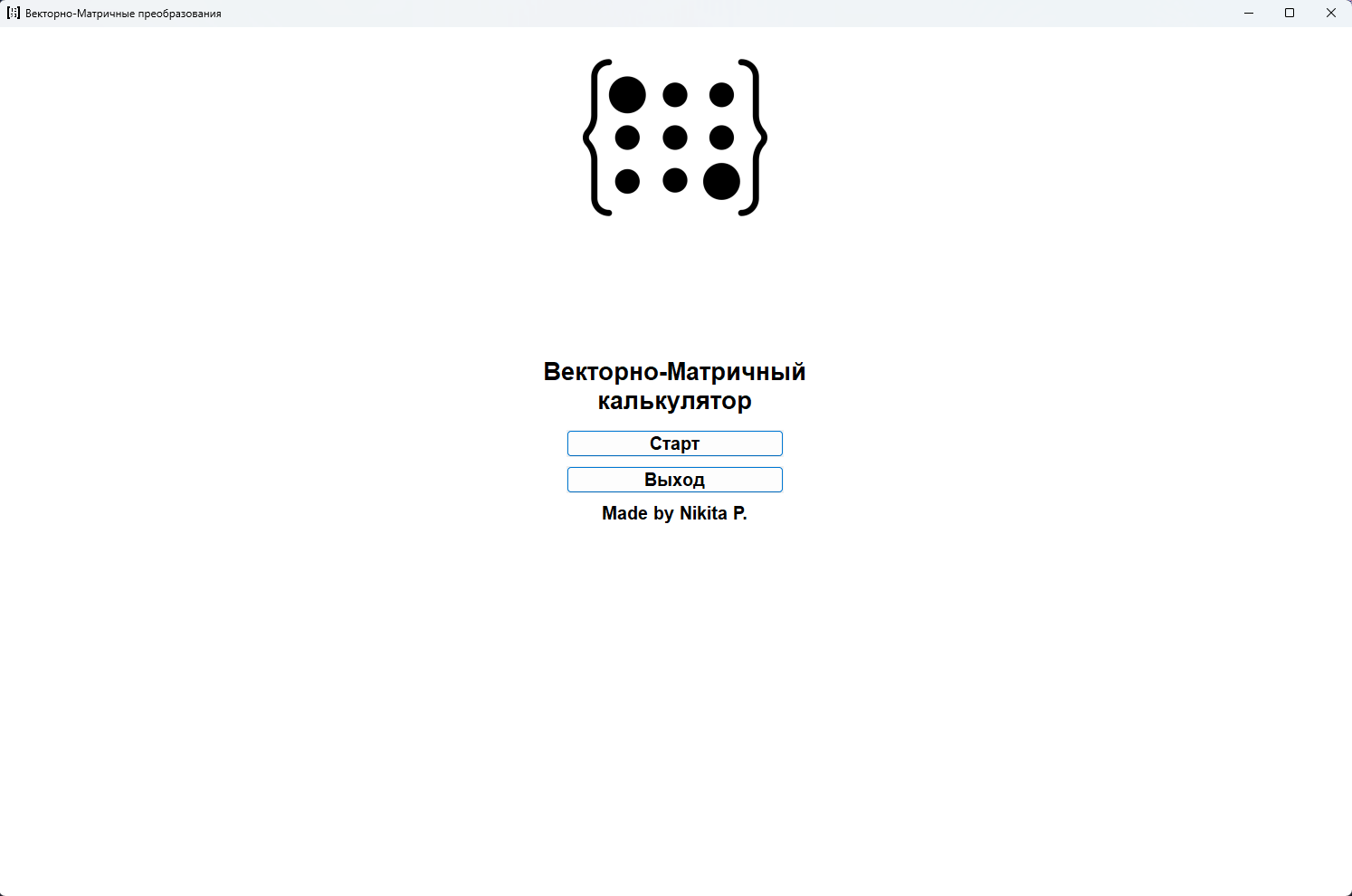


Рис. 1 – начальный экран приложения.

На начальном экране отображаются две кнопки: "Начать" и "Выход". При нажатии кнопки "Начать" открывается главный экран, где можно вводить элементы. Нажатие кнопки "Выход" завершает работу программы.

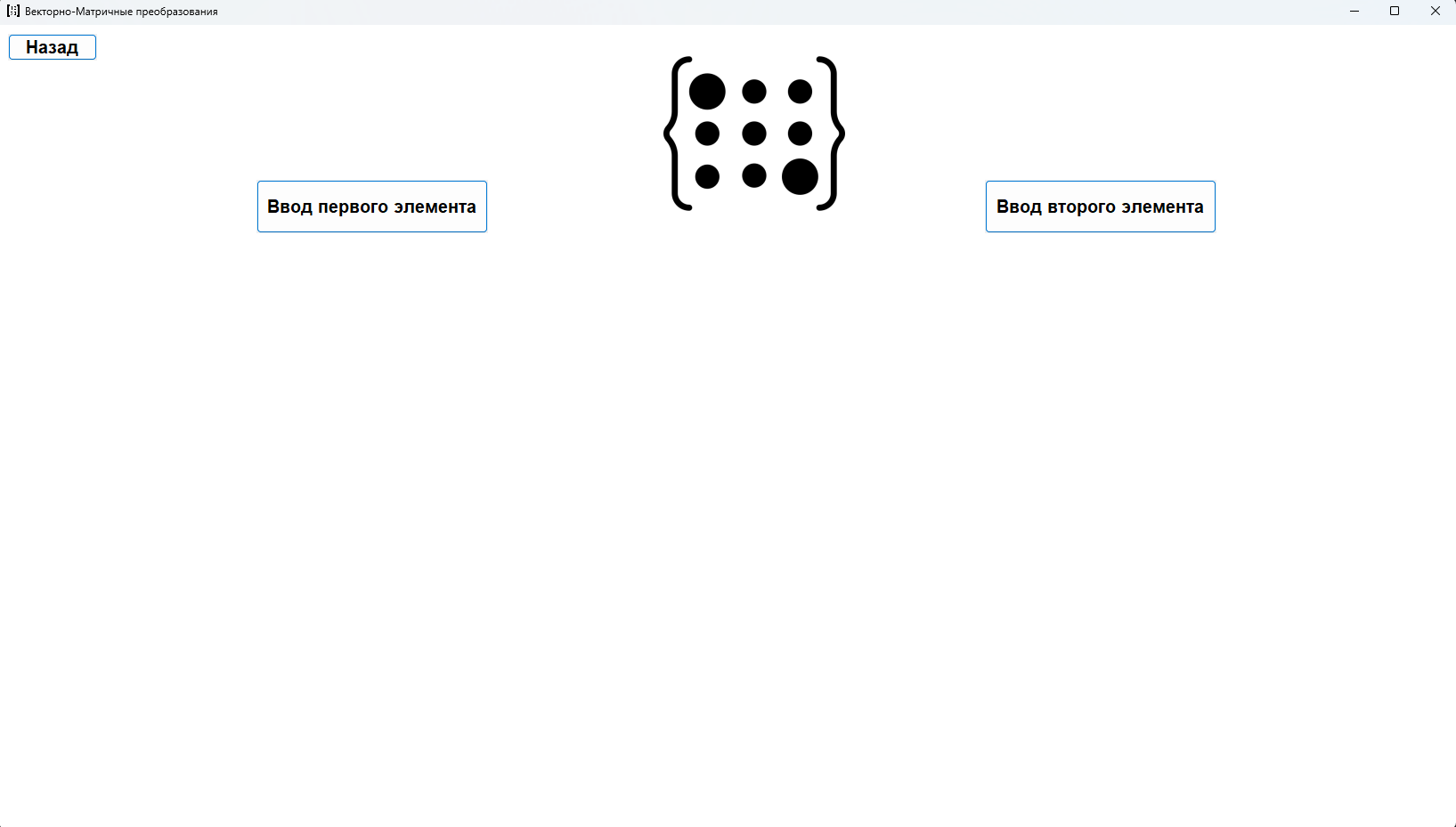


Рис. 2 – главный экран.

На главном экране отображаются три кнопки: "Ввод первого элемента", "Ввод второго элемента" и "Назад". Кнопка "Назад" возвращает пользователя на начальный экран. Кнопки "Ввод первого элемента" и "Ввод второго элемента" открывают соответствующие окна для ввода элементов. На экранах ввода элементов пользователю предоставляется возможность ввести матрицу или вектор. Интерфейс для ввода матрицы и вектора включает текстовые поля для каждого элемента, а также кнопки для подтверждения ввода и возврата на главный экран.

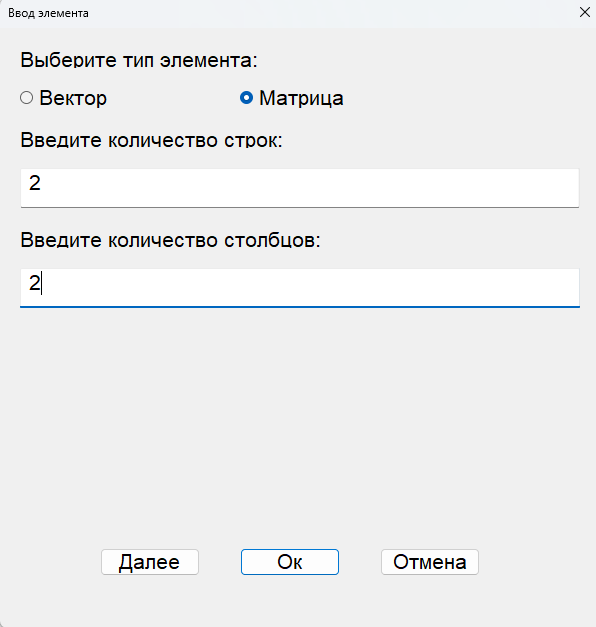
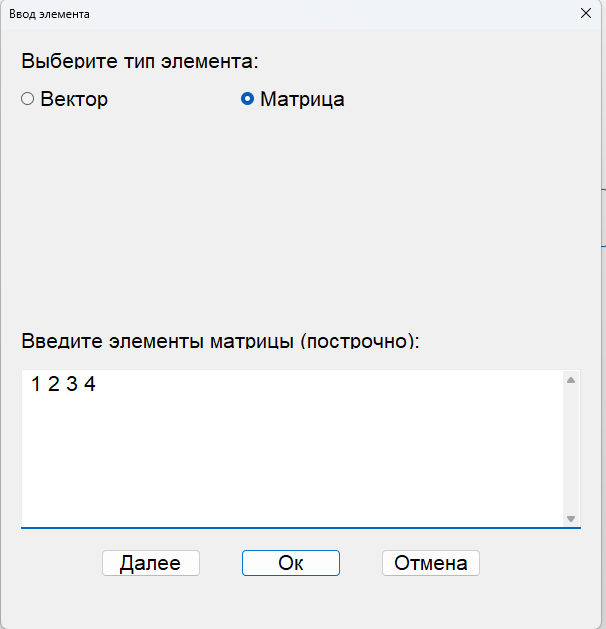
 

Рис. 3 – ввод матрицы.

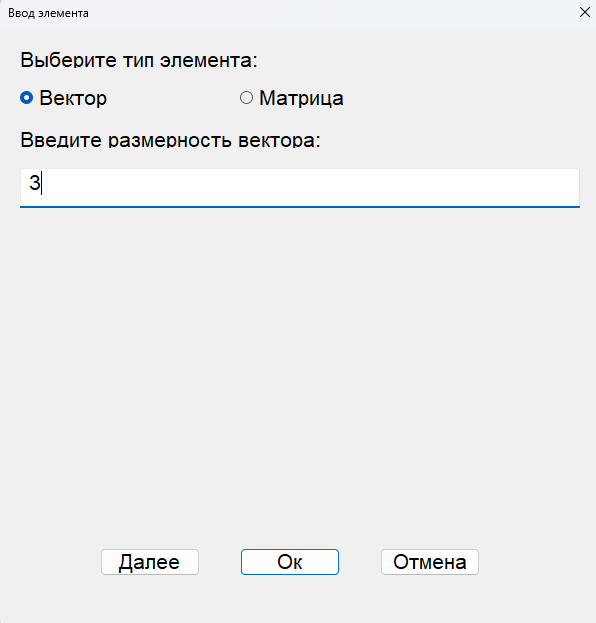
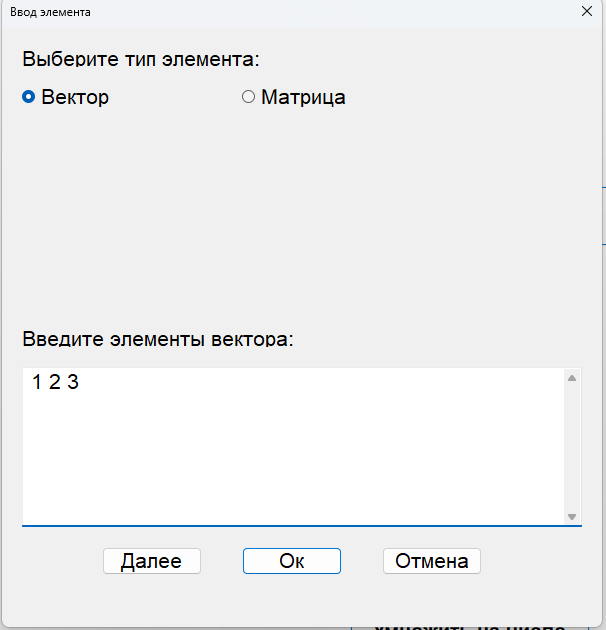
 

Рис. 4 – ввод вектора.

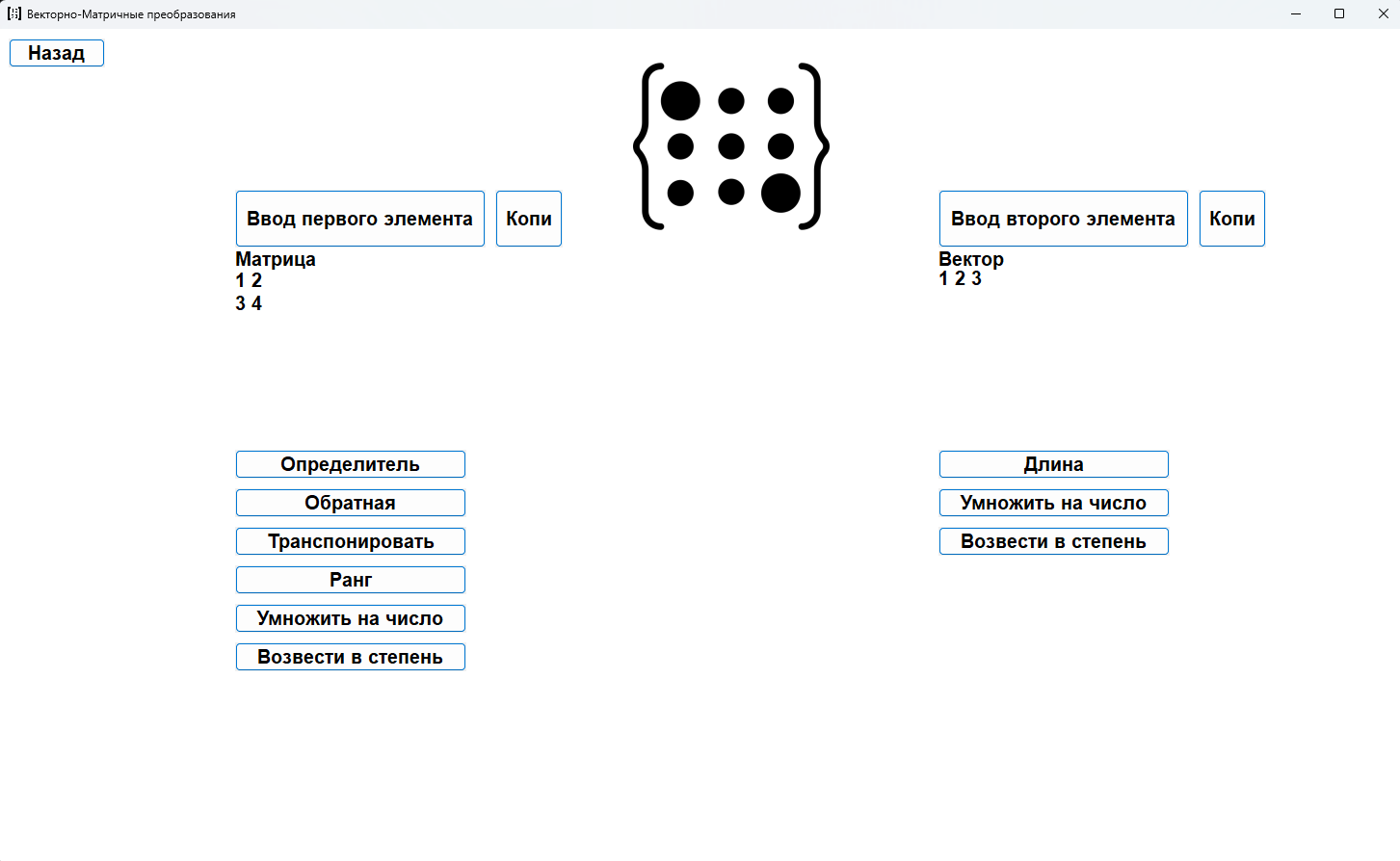


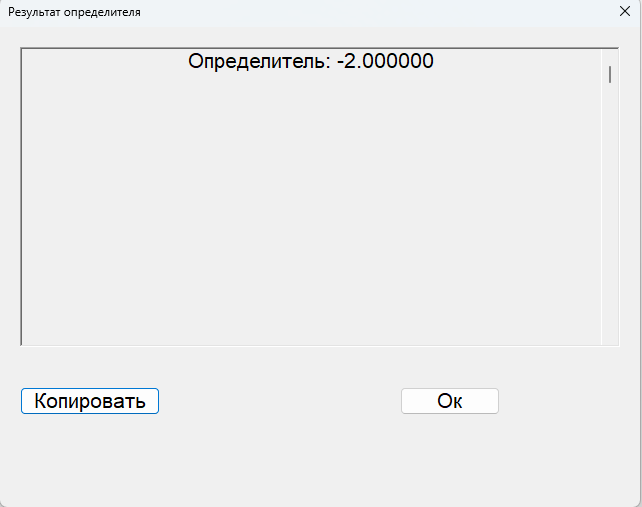
Рис. 5 – результат ввода элементов.

После ввода элементов они отображаются на экране. Также появляются дополнительные кнопки, связанные с введёнными элементами. Для матрицы это: "Определитель", "Обратная", "Транспонировать", "Ранг", "Умножить на число", "Возвести в степень". Для вектора это: "Длина", "Умножить на число", "Возвести в степень".

Операции над матрицой

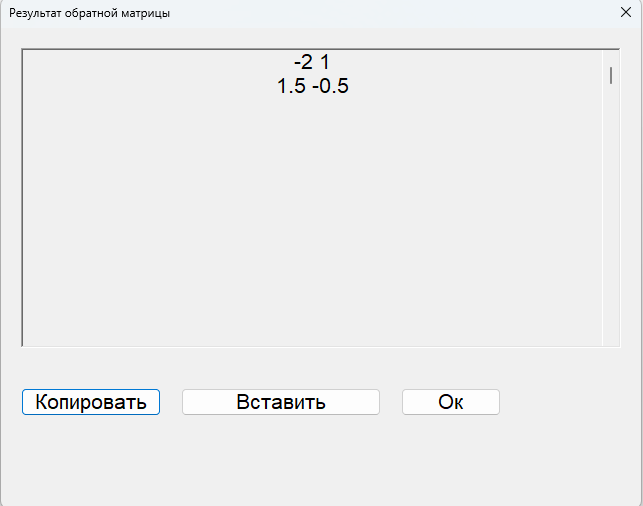
После ввода матрицы, пользователю доступны следующие операции:

1. **Определитель**:
   * Вычисление и отображение определителя введённой матрицы.

****

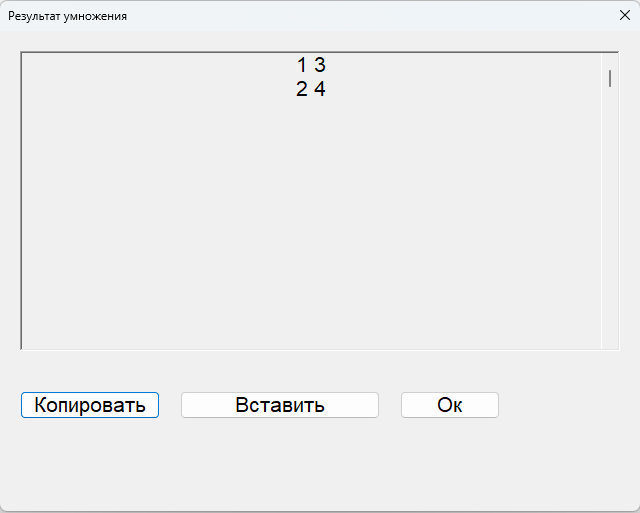
**Рис. 6 – результат вычисления определителя.**

1. **Обратная матрица**:
   * Вычисление и отображение обратной матрицы, если она существует.

****

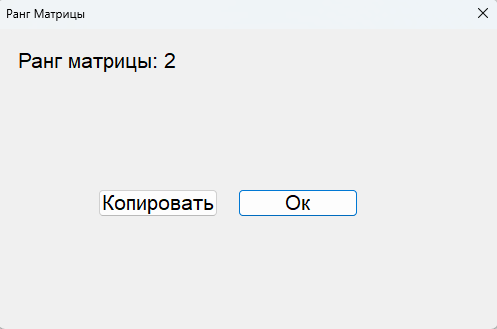
**Рис. 7 – результат вычисления обратной матрицы.**

1. **Транспонировать**:
   * Транспонирование матрицы и отображение результата.

****

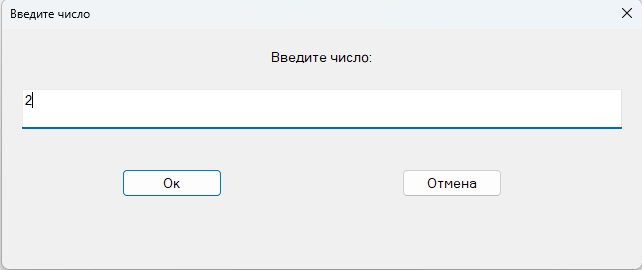
**Рис. 8 – результат транспонирования матрицы.**

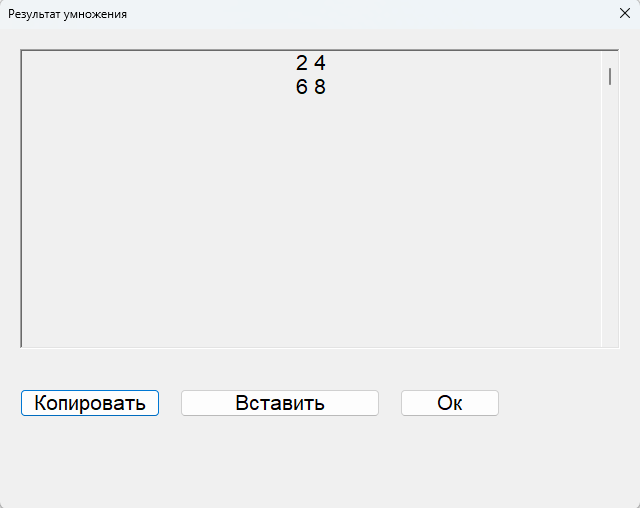
1. **Ранг**:
   * Вычисление и отображение ранга матрицы.



**Рис. 9 – результат вычисления ранга матрицы.**

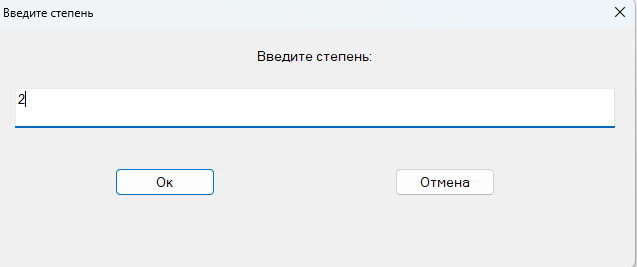
1. **Умножить на число**:
   * Ввод скалярного значения и умножение матрицы на это число. Отображение результата.

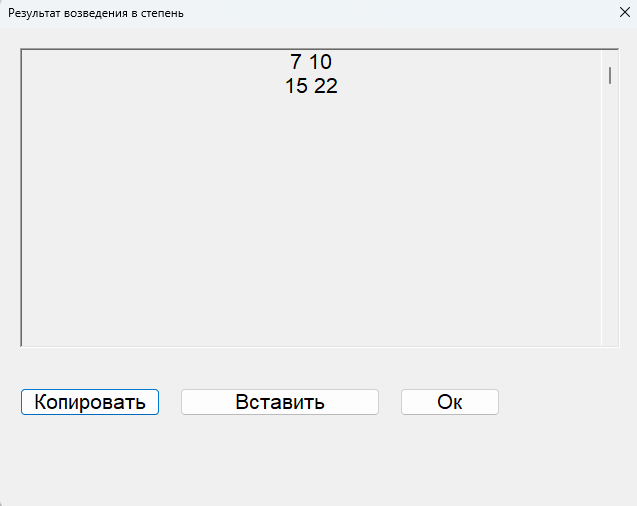




**Рис. 10 – результат умножения матрицы на число.**

1. **Возвести в степень**:
   * Ввод степени и возведение матрицы в эту степень. Отображение результата.



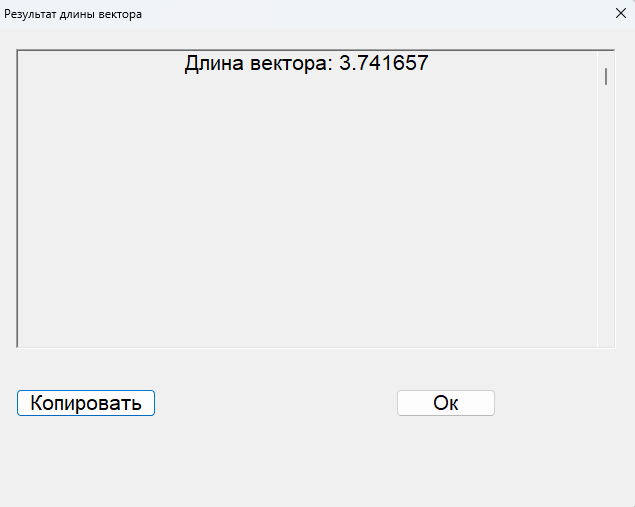


**Рис. 11 – результат возведения матрицы в степень.**

Операции над вектором

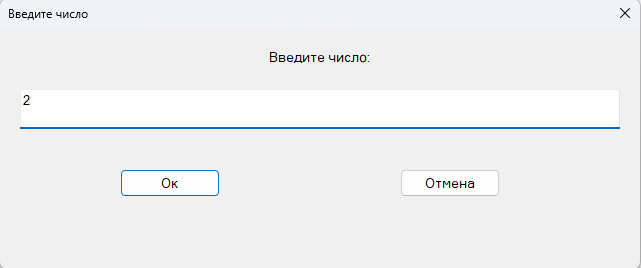
После ввода вектора, пользователю доступны следующие операции:

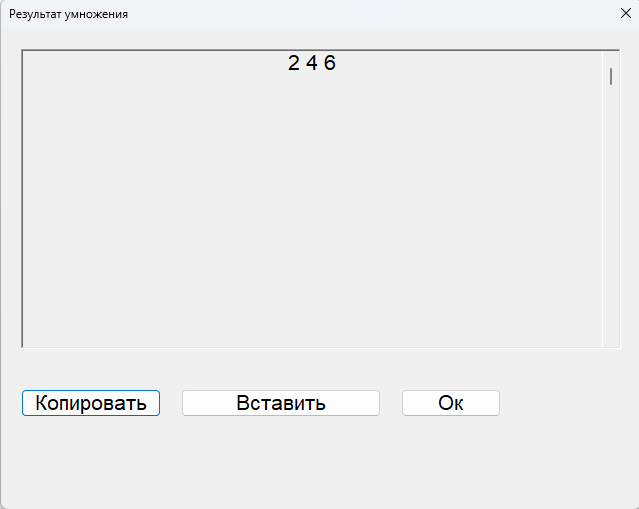
1. **Длина**:
   * Вычисление и отображение длины введённого вектора.



**Рис. 12 – результат вычисления длины вектора.**

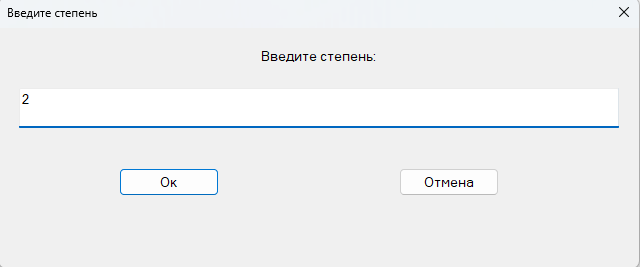
1. **Умножить на число**:
   * Ввод скалярного значения и умножение вектора на это число. Отображение результата.

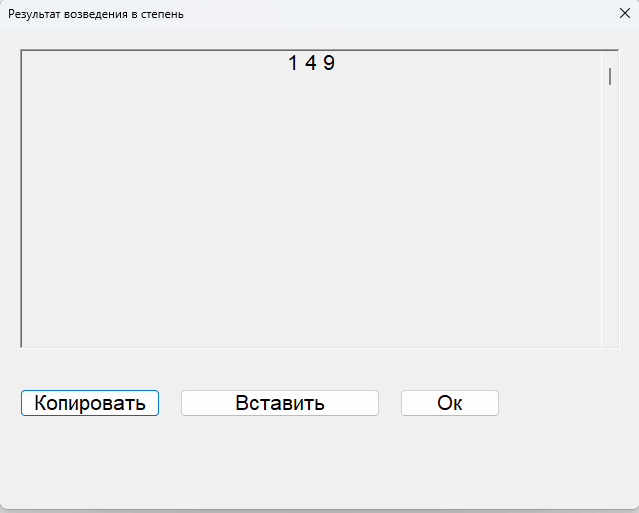




**Рис. 13 – результат умножения вектора на число.**

1. **Возвести в степень**:
   * Ввод степени и возведение вектора в эту степень. Отображение результата.

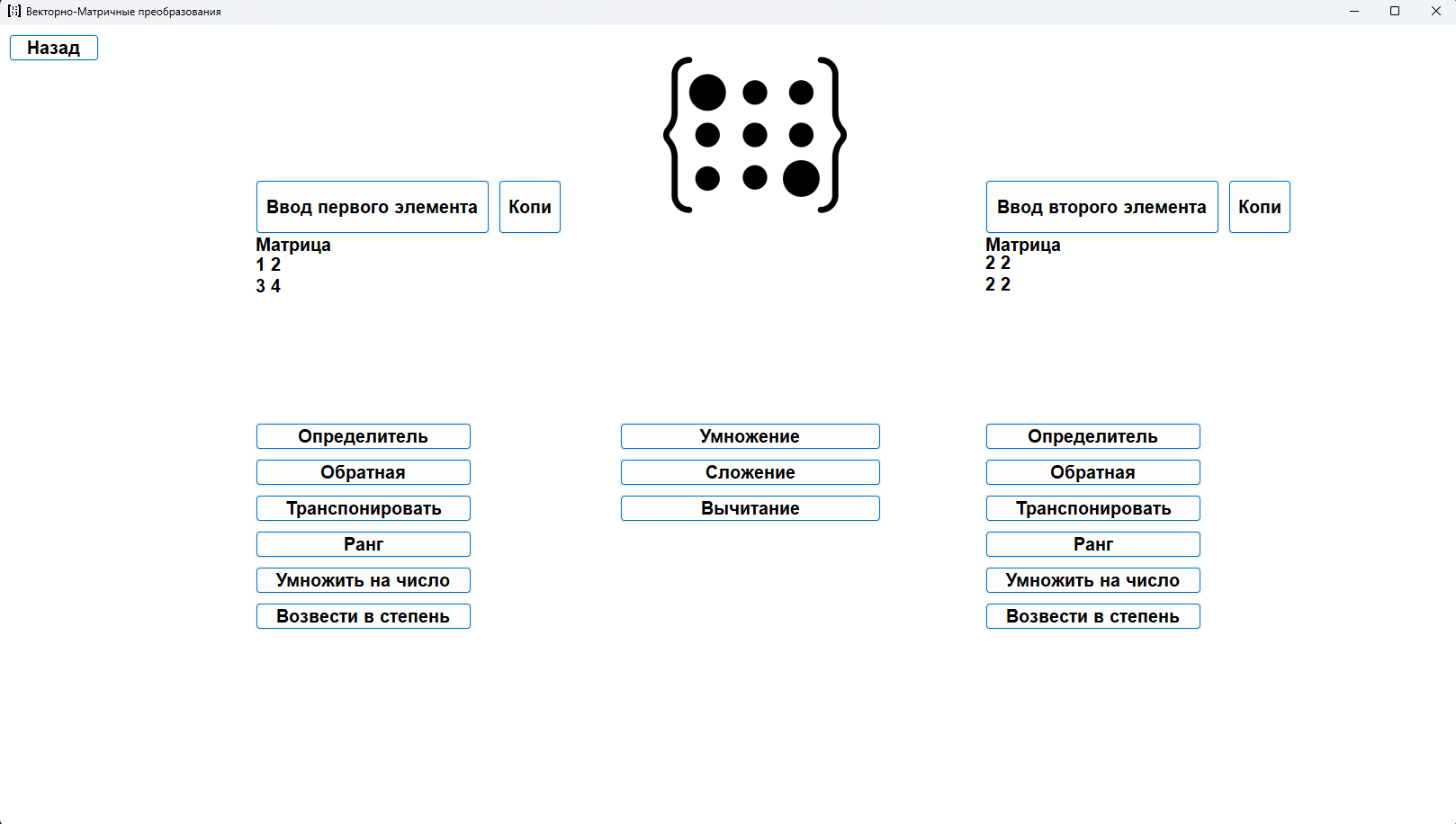




**Рис. 14 – результат возведения вектора в степень.**

Операции матрица-матрица

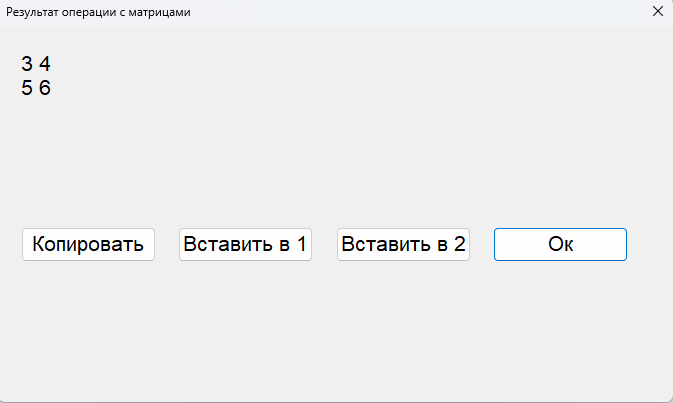
Введем во второй элемент матрицу 2 на 2:  
2 2  
2 2

****

**Рис. 14 – кнопки при двух матрицах.**

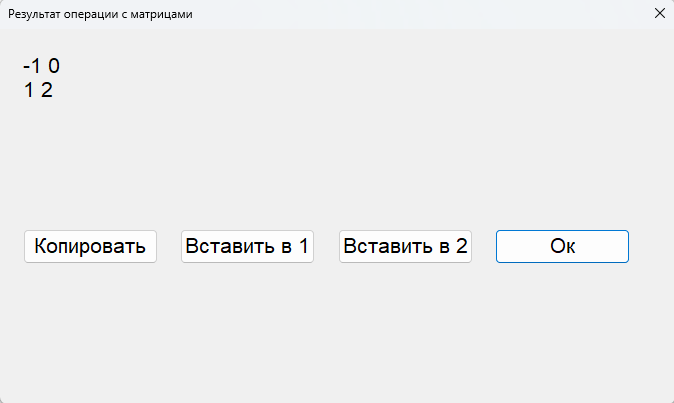
После ввода двух матриц пользователю доступны следующие операции:

1. **Сложение матриц**:
   * Пользователь может выбрать операцию сложения двух введенных матриц. Результатом будет матрица, где каждый элемент равен сумме соответствующих элементов двух исходных матриц.



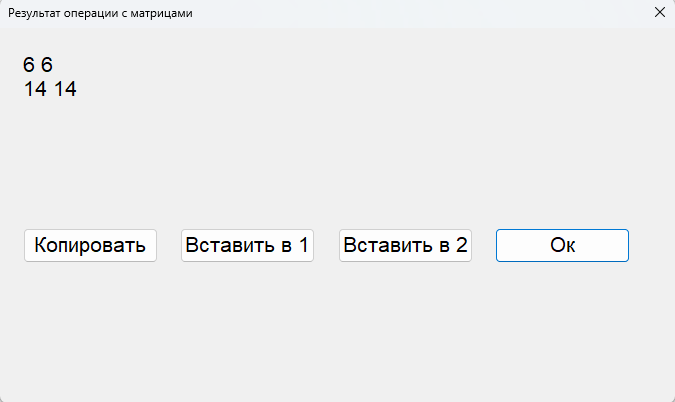
**Рис. 16 – результат сложения матриц.**

1. **Вычитание матриц**:
   * Пользователь может выбрать операцию вычитания одной матрицы из другой. Результатом будет матрица, где каждый элемент равен разности соответствующих элементов двух исходных матриц.



**Рис. 17 – результат вычитания матриц.**

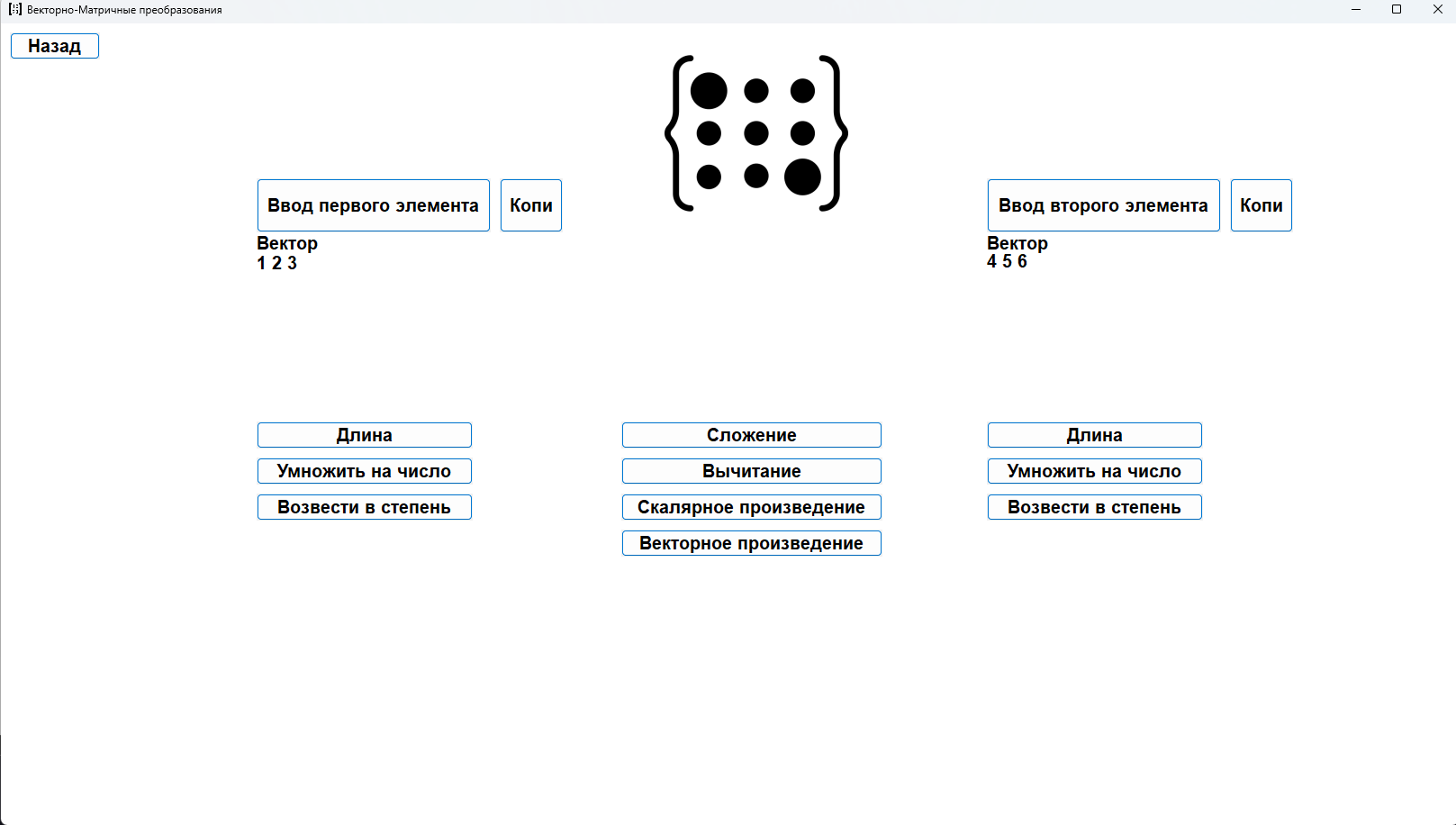
1. **Умножение матриц**:
   * Пользователь может выбрать операцию умножения двух введенных матриц. Результатом будет новая матрица, где каждый элемент вычисляется как сумма произведений элементов строк первой матрицы и столбцов второй матрицы.

****

**Рис. 18 – результат умножения матриц.**

Операции вектор-вектор

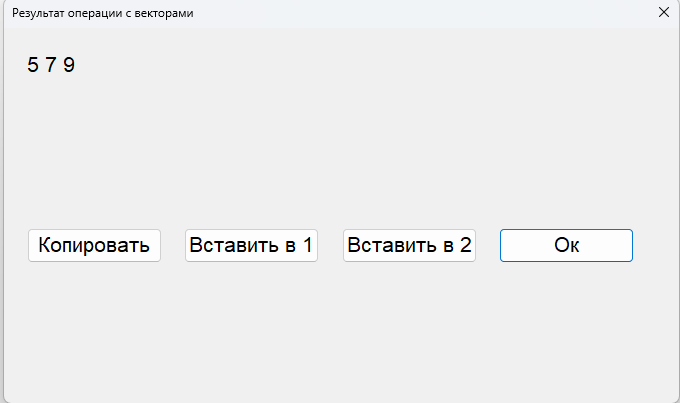
Введем в 1 элемент: вектор размерностью 3 – 1 2 3  
Введем во 2 элемент: вектор размерностью 3 – 4 5 6



**Рис. 19 – кнопки при двух векторах.**

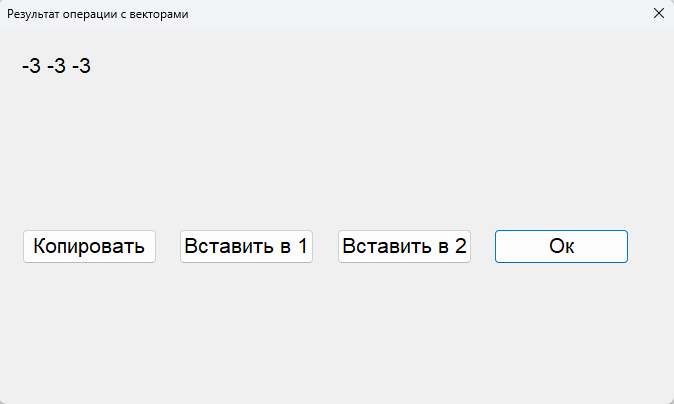
После ввода двух векторов пользователю доступны следующие операции:

1. **Сложение векторов**:
   * Пользователь может выбрать операцию сложения двух введенных векторов. Результатом будет новый вектор, где каждый элемент равен сумме соответствующих элементов двух исходных векторов.

****

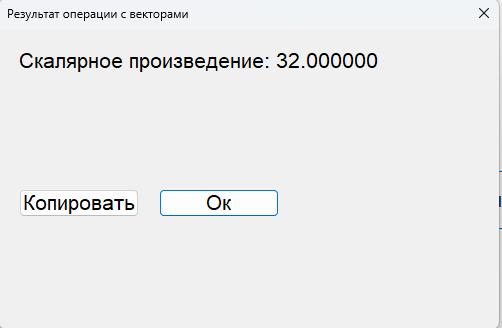
**Рис. 20 – результат сложения векторов.**

1. **Вычитание векторов**:
   * Пользователь может выбрать операцию вычитания одного вектора из другого. Результатом будет новый вектор, где каждый элемент равен разности соответствующих элементов двух исходных векторов.

****

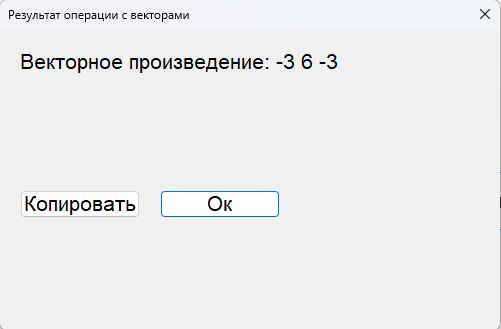
**Рис. 21 – результат вычитания векторов.**

1. **Скалярное произведение**:
   * Пользователь может выбрать операцию скалярного произведения двух векторов. Результатом будет число, равное сумме произведений соответствующих элементов двух векторов.

****

**Рис. 22 – результат скалярного произведения векторов.**

1. **Векторное произведение**:
   * Пользователь может выбрать операцию векторного произведения двух векторов. Это операция доступна только для трёхмерных векторов. Результатом будет новый вектор, перпендикулярный к исходным векторам.

****

**Рис. 23 – результат векторного произведения векторов.**

Все результаты вычислений, полученные в программе, могут быть скопированы пользователем для дальнейшего использования. В окнах с результатами есть кнопка "Скопировать", которая позволяет быстро и легко поместить результат в буфер обмена. Пользователь может нажать эту кнопку, чтобы скопировать текст результата и использовать его в других приложениях или для дальнейших вычислений.

Кроме того, некоторые результаты могут быть непосредственно вставлены в другие элементы приложения. Например, результат умножения матрицы на число или возведения вектора в степень может быть использован как новый входной элемент для дальнейших операций. Это позволяет удобно и быстро проводить последовательные вычисления, не вводя данные повторно.

Вывод

В ходе работы над курсовым проектом было разработано классическое приложение Windows для выполнения различных операций над матрицами и векторами. Программа предоставляет удобный интерфейс, позволяющий пользователю легко вводить элементы, выполнять операции и просматривать результаты.

Основные функциональные возможности приложения включают:

* Ввод и отображение матриц и векторов.
* Выполнение операций матрица-матрица (сложение, вычитание, умножение).
* Выполнение операций вектор-вектор (сложение, вычитание, скалярное и векторное произведения).
* Выполнение операций над отдельными матрицами (определитель, обратная матрица, транспонирование, ранг, умножение на число, возведение в степень).
* Выполнение операций над отдельными векторами (длина, умножение на число, возведение в степень).

Программа также предоставляет возможность копирования всех результатов вычислений для дальнейшего использования. В окнах с результатами есть кнопка "Скопировать", которая позволяет быстро и легко поместить результат в буфер обмена. Некоторые результаты могут быть непосредственно вставлены в другие элементы приложения, что упрощает последовательные вычисления и делает работу с программой более эффективной.

Благодаря использованию Windows API, приложение обеспечивает высокую производительность и гибкость, предлагая пользователям интуитивно понятный графический интерфейс и широкий спектр функциональных возможностей для работы с матрицами и векторами.

Список источников

1. <https://learn.microsoft.com/ru-ru/windows/win32/api/>. На этом сайте представлена официальная документация по функциям и методам Windows API, включая подробные объяснения и примеры использования таких функций, как CreateWindowEx, RegisterClass, и другие.
2. <https://learn.microsoft.com/ru-ru/>. Этот сайт предоставляет доступ к широкой документации по различным технологиям и инструментам Microsoft, включая язык программирования C++, среду разработки Visual Studio.
3. <https://metanit.com/cpp/>. Сайт можно назвать онлайн-учебником программирования, где собраны самые различные материалы, в том числе и по C++.
4. <https://stackoverflow.com/>: Вопросы и ответы по конкретным проблемам, возникающим при разработке приложений на C++ и использовании Windows API.
5. Безруков В.А. Win32 API. Программирование /учебное пособие. – СПб: СПбГУ ИТМО, 2009. – 90 с.
6. Вестяк А. В., Вестяк В. А., Тарлаковский Д. В. Алгебра и аналитическая геометрия. Ч. 1. Учебное пособие. Москва, издательство МАИ 2002 г., 460 с.
7. Вестяк А. В., Вестяк В. А., Тарлаковский Д. В. Алгебра и аналитическая геометрия. Ч. 2. Учебное пособие. Москва, издательство МАИ 2007 г., 520 с.
8. Линейная алгебра. Аналитическая геометрия : учеб. пособие / А.М. Ивлева, П.И. Прилуцкая, И.Д. Черных. – 4-е изд-е, испр. и доп. – Новосибирск : Изд-во НГТУ, 2014. – 180 с.