МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ

імені ІГОРЯ СІКОРСЬКОГО»

ІНСТИТУТ ПРИКЛАДНОГО СИСТЕМНОГО АНАЛІЗУ

КАФЕДРА СИСТЕМНОГО ПРОЕКТУВАННЯ

**КУРСОВА РОБОТА**

з дисципліни «Алгоритмізація та програмування»

на тему: Програма побудови графіків по введеним функціям

Виконав: студент 1 курсу групи ДА-12

спеціальність 122 «Комп’ютерні науки»

Кракович Павло Дмитрович  
Керівник: Безносик О. Ю.   
Національна оцінка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кількість балів: \_\_\_\_\_\_Оцінка: ECTS \_\_\_\_

Прийняли:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
(підпис) (, прізвище та ініціали)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис) ( прізвище та ініціали)

Засвідчую, що у цій курсовій роботі немає запозичень з праць інших авторів без відповідних посилань

студент \_\_\_\_\_\_\_\_\_\_\_\_

Київ - 2022 рік

Національний технічний університет України “КПІ” ННК “ІПСА”

(назва вищого навчального закладу)

Кафедра системного проектування ⠀

Дисципліна алгоритмізація та програмування ⠀

Спеціальність 122 Комп’ютерні науки ⠀

Курс I ⠀Група ДА-12 Семестр 2 ⠀

**ЗАВДАННЯ**

**на курсову роботу студента**

Верхоли Олексія Андрійовича

(прізвище, ім’я, по батькові)

1. Тема роботи

Програма роботи з матрицями

2. Строк здачі студентом закінченого проекту (роботи) 10.06.22 ⠀

3. Вихідні дані до проекту (роботи)

Мова програмування С++

Програма для ОС Windows

Технічна література

4. Зміст розрахунково – пояснювальної записки (перелік питань, які підлягають розробці)

Розробка алгоритму програми

Архітектура програми, лістинг (код) програми

Інструкція користувачу

Інструкція розробнику

5. Перелік графічного матеріалу ( з точним зазначенням обов’язкових креслень)

Блок-схема алгоритму роботи програми (Формат А4)

Отримані результати. Графічні матеріали

6. Дата видачі завдання 08.02.2022 ⠀

**КАЛЕНДАРНИЙ ПЛАН**

|  |  |  |  |
| --- | --- | --- | --- |
| № п/п | Найменування етапів курсової роботи | Строк виконання  етапів роботи | Примітки |
| 1. | Вивчення та аналіз алгоритму | 18.02.2022 |  |
|  |  |  |  |
| 2. | Визначення структури програм- них модулів | 25.02.2022 |  |
|  |  |  |  |
| 3. | Розробка інтерфейсу | 21.03.2022 |  |
|  |  |  |  |
| 4. | Оптимізація програми | 08.04.2022 |  |
|  |  |  |  |
| 5. | Налагодження та тестування  програми | 12.04.2022 |  |
|  |  |  |  |
| 6. | Оформлення роботи | 22.04.2022 |  |
|  |  |  |  |
| 7. | Здача на перевірку (електронна версія | 24.04.2022 |  |
|  |  |  |  |
| 8. | 8. Виправлення недоліків та доробка програми | 29.04.2022 |  |
|  |  |  |  |
| 9. | Захист | 10.06.2022 |  |

Студент ⠀ Кракович П.Д ⠀   
 (підпис студента) (прізвище, ім‘я, по батькові студента)  
Керівник ⠀ Безносик О.Ю ⠀   
 (підпис викладача) (прізвище, ім‘я, по батькові викладача)

“10” червня 2022р.

**Зміст**

[Вступ 5](#_Toc103235816)

[1. Обґрунтування та вибір алгоритмів 6](#_Toc103235817)

[1.1. Прості операції 6](#_Toc103235818)

[1.2. Складні операції 9](#_Toc103235819)

[2. Розробка програми 12](#_Toc103235820)

[2.1. Загальні відомості 12](#_Toc103235821)

[2.2. Функціональне призначення 12](#_Toc103235822)

[2.3. Опис логічної структури 13](#_Toc103235823)

[2.4. Максимальне завантаження 16](#_Toc103235824)

[2.5. Вхідні та вихідні дані 17](#_Toc103235825)

[2.6. Керівництво користувачу 20](#_Toc103235826)

[2.7. Керівництво розробнику 22](#_Toc103235827)

[3. Висновки 23](#_Toc103235828)

[3.1. Перевірка справності програми 23](#_Toc103235829)

[3.2. Висновок 26](#_Toc103235830)

[Додатки 27](#_Toc103235831)

[Література 27](#_Toc103235832)

[Лістинг програми 28](#_Toc103235833)

[Блок-схеми 65](#_Toc103235834)

**Вступ**

Матриця – це математичний об'єкт, записаний у вигляді прямокутної таблиці чисел, він допускає операції (додавання, віднімання, множення та множення на скаляр). Зазвичай матриці представляються двовимірними (прямокутними) таблицями. Матрицею розміром називається множина елементів , розміщених у вигляді прямокутної таблиці з рядків і стовбців, де – елемент матриці,– номер рядка, – номер стовбця.

Загальний вигляд прямокутної матриці.

Матриці застосовуються у різних галузях: науці, комерції та соціології. Також вони використовуються у комп’ютерній графіці, оптиці, криптографії, економіці, хімії, математиці, бездротовому зв’язку тощо. Квадратні матриці можуть представляти лінійні перетворення об’єктів, вони використовуються для проекції тривимірних об’єктів на двовимірні поверхні в сфері графіки, цифрові зображення зберігаються як двовимірні матриці. В оптиці матриці використовуються для врахування відображення та заломлення. Також, у матриць є застосування поза технічними науками, адже вони можуть репрезентувати будь-який прямокутний масив чисел, наприклад, цін, витрат тощо.

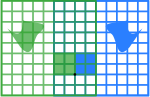


Рисунок 1 – Приклад використання матриці у графічних перетвореннях:  
Відбиття відносно вертикальної осі за допомогою матриці

Існує велика кількість онлайн матричних калькуляторів, як [Matrix Reshish](https://matrix.reshish.com/) або [Symbolab](https://www.symbolab.com/solver/matrix-calculator), але більшість із них мають проблеми, від довгого часу обрахунків, або сам факт того, що вони доступні лише за допомогою інтернету.

Отже, майже кожна сучасна сфера науки, пов’язана з математикою, використовує матриці. Тому вивчення матриць є важливою частиною вступу до цих сфер, для чого і потрібні матричні калькулятори. Моєю задачею стало створення доступного офлайн матричного калькулятора, з усіма базовими та кількома складними операціями, інтуїтивним інтерфейсом та функцією збереження даних для можливого аналізу результатів в майбутньому.

# **1. Обґрунтування та вибір алгоритмів**

Для початку потрібно розглянути специфіку визначення елементів матриці в програмі. Через нестачу елегантного динамічного двомірного масиву в мові програмування С++, прямокутні матриці з рядків і стовбців будуть визначені динамічним одномірним масивом розміром . Таким чином, елемент матриці буде записаний як , але для простоти запису, буде представлений як в подальшому тексті.

Частину операцій, що потрібно було імплементувати в програму, було дано в умові завдання, а саме: додавання, скалярне множення та ділення, обернення, алгоритм Штрасена для множення матриць, та обертання. До цього списку були додані ще декілька алгоритмів, тому, остаточним списком перетворень є:

* Додавання
* Віднімання
* Скалярне множення
* Скалярне ділення
* Транспонування
* Множення
* Знаходження детермінанту
* Обернення матриці
* Обертання матриці

Дані операції будуть поділені на прості (тривіальні) та складні.

## **1.1. Прості операції**

В цьому розділу буде розглянуто наступні операції:

* Додавання
* Віднімання
* Скалярне множення
* Скалярне ділення
* Транспонування
* Обертання матриць

Усі прості операції мають складність – .

***Додавання та віднімання:***

Сума двох матриць є матрицею , так що . З цього виходить, що всі три матриці мають однакові розміри на .

Приклад додавання двох матриць.

Аналогічно, різниця двох матриць є матрицею , так що . Усі три матриці також мають однакові розміри на .

Приклад віднімання двох матриць.

Алгоритм цих операцій є тривіальним: ітеративне проходження по всім елементам обох матриць, проведення відповідної операції та запис результату до нового масиву.

***Скалярне множення та скалярне ділення:***

Якщо дано матрицю і число , можемо означити множення на скаляр як матрицю , де .

Приклад скалярного множення матриці на число 2.

Аналогічно, означенням скалярного ділення матриці на число (за умови, що ) є матриця , де .

Приклад скалярного ділення матриці на число 2.

Алгоритм цих операцій є аналогічний алгоритму додавання і віднімання, з заміною на потрібну операцію.

***Транспонування***

Транспонування матриці розміром на утворює матрицю на що позначається , яка є результатом перевертання рядків у стовпці і навпаки .

Приклад транспонування матриці.

Алгоритм цієї операції був розділений на дві частини. Перша відповідає за квадратні матриці = , а друга, за інші випадки. Якщо матриця є квадратною, то потрібно замінити для усіх . Інакше, потрібно створити допоміжну матрицю розміром , скопіювати до неї початкову матрицю, провести транспонування на новій квадратній матриці, та перенести дані в початкову матрицю, та поміняти значення та .

***Обертання матриці***

Формально обертання не є матричною операцією, але вона була включена в список для виконання початкової умови завдання. Обертання матриці розміром на в матрицю (розміром на якщо обертання на 180, інакше на ) за допомогою геометричних перетворень.

Алгоритм поділено на 3 випадки, у кожному з яких проводиться послідовна ітерація по елементах матриці:

1. 90

2. 180

3. 270

Усі види обертання виконуються за часовою стрілкою.

## **1.2. Складні операції**

В цьому розділу буде розглянено наступні операції:

* Множення
* Знаходження детермінанту
* Обернення матриці

***Множення***

Нехай дано дві прямокутні матриці і розмірності і :

Тоді, матриця розмірності називається їх добутком, де:

Операція множення двох матриць здійсненна тільки в тому випадку, якщо число стовпців в першому співмножнику дорівнює числу рядків у другому. Ця операція є відносно довгою, маючи складність , тому в даній програмі був використаний алгоритм Штрассена.

***Алгоритм Штрассена***

Цей алгоритм дозволяє швидше за стандартний спосіб множити матриці. На практиці це відчутно на великих матрицях. На відміну від традиційного алгоритму множення матриць, алгоритм Штрассена множить матриці за час , що дає виграш на великих матрицях починаючи, приблизно, з матриць розміру . Варто зазначити, що даний алгоритм оптимальний лише для квадратих матриць розмірності . Інакше, відсутні рядки і стовпці заповнюються нулями. При цьому отримуються зручні для рекурсивного множення матриці, але втрачається ефективність за рахунок додаткових непотрібних множень.

Нехай — дві квадратні матриці розмірності . Ми можемо обчислити матрицю , як . Розділимо матриці на рівні за розміром блочні матриці

демають розмірність Далі, визначаємо нові матриці:

Тепер можливо виразити через :

Рекурсивний процес ділення повторюється раз поки розмір матриць не стане досить малим, далі використовується звичайний метод множення матриць.

***Знаходження детермінанту***

Алгоритм знаходження детермінанту (визначника) є рекурсивним, та розділений на 2 випадки:

1. Щоб знайти визначник матриці, множимо елементи головної діагоналі та віднімаємо добуток елементів побічної діагоналі:

2. Для матриць більш високих порядків () визначник можна обчислити, застосувавши таку рекурсивну формулу:

Де доповнювальний мінор до елементу . Ця формула називається розкладанням за рядком. Складність цього алгоритму – . Слід зауважити, що існують більш оптимальні версії алгоритмів, але для спрощення та кращого форматування коду було прийнято рішення використати саме цей алгоритм.

***Обернення матриці***

Обернена матриця – матриця якя існує для кожної квадратної матриці розмірності в якої , причому , де одинична матриця.

Приклад оберненої матриці розмірності та одиничної матриці.

Алгоритм знаходження оберненої матриці розділений на 4 частини:

1. Знаходження матриці мінорів.
2. Перетворити її в матрицю кофакторів.
3. Транспонувати матрицю кофакторів.
4. Помножити результуючу матрицю на

1. Щоб знайти матрицю мінорів будь-якої матриці , потрібно використати наступну формулу:

Де - доповнювальний мінор до елементу .

2. Для того щоб перетворити матрицю мінорів у матрицю кофакторів , потрібно використати наступну формулу:

3. Операцію транспонування було описано в попередніх розділах.  
4. Операції скалярного множення та знаходження детермінанту також було описано в попередніх розділах.

Приклад обернення матриці розмірності .

**2. Розробка програми**

## **2.1. Загальні відомості**

Створена програма була названа «Matrix Calculator». Вона була написана на мові С++ стандарту С++ 20, за допомогою IDE Microsoft Visual Studio та графічної бібліотеки SFML 2.5.1. Для того, щоб у користувача була можливість використовувати цю програму, їх ПК має задовільняти певним мінімальним вимогам:

* ОС – Windows 10 x64
* Процессор – 1 ГГц
* Графічна карта – 100 Мб пам’яті
* Оперативна пам'ять – 1 Гб
* Диск – 20 Мб вільної пам’яті

Ці вимоги були розроблені шляхом дослідження використаних ресурсів програми та базових вимог до будь яких програм скомпільованих на Windows 10 для 64-бітної ОС.

## **2.2. Функціональне призначення**

В цій програмі було використано 27 індивідуальних функцій. Не враховуючи головну функцію main, що відповідає за створення меню програми, та фунції DimChoose1, DimChoose2, DimChoose3, DimChoose4 та DimChoose5, що відповідають за допоміжні вікна вибору розміру матриць, залишається 21 функція. Вони будуть представлені в форматі «Назва – Опис». Також, у функціях що використовують вищевкзані алгоритми, буде вказана їх складність.

* OperationChooser() – Відповідає за вибір операцій та виклик потрібних функцій, залежно від вибраної операції.
* InputMatrix() – Функція поелементного введення матриці.
* OutputMatrix() – Функція виведення матриці та можливого збереження її до файлу.
* ZeroMatrix() – Функція приймає матрицю, та задає усі її елементи до 0 – .
* PadUp() – Збільшує розмір заданої матриці на будь-який, нові елементи заповнюються нулями – .
* PadDown() – Зменьшує розмір заданої матриці на будь-який інший, невикористані елементи знищуються – .
* NextPowerOfTwo() – Знаходить наступне число що є степенем двійки відносно заданого числа. – .
* IsPowerOfTwo() – Перевіряє, чи задане число є степенем двійки. –

Алгоритми наступних функцій пояснені у попередніх розділах.

* TransposeMatrix() – Транспонує задану матрицю – .
* AddMatrices() – Додає 2 задані матриці – .
* SubtractMatrices() – Віднімає 2 задані матриці – .
* ScalarMultiplyMatrix() – Скалярно множе матрицю на число – .
* DivideMatrix() – Скалярно ділить матрицю на число – .
* StandardMultiply() – Множить дві матриці простим (наївним) алгоритмом – .
* StrassenMultiply() – Множить дві матриці алгоритмом Штрассена. –
* GetSubMatrix() – Знаходить доповнювальний мінор матриці відносно заданого елемента шляхом викреслювання – .
* GetDeterminant() – Знаходить визначник матриці – .
* MatrixOfMinors() – Знаходить матрицю мінорів – .
* MatrixOfCofactors() – Знаходить матрицю кофакторів – .
* InverseMatrix() – Знаходить обернену матрицю до заданої матриці – .
* RotateMatrix() – Обертає матрицю на обраний кут – .

## **2.3. Опис логічної структури**

В цьому розділі будуть описані найголовніші функції програми, їх вхідні дані, та короткий опис з посиланням на блок-схеми в додатку.

|  |  |  |
| --- | --- | --- |
| Назва функції | Вхідні дані | Опис алгоритму |
| void ZeroMatrix() | int x, int y – Розміри матриці  double\* m – Матриця | Заповнює всі елементи матриці нулями. [Див. блок-схему 1.](#block1) |
| void PadUp() | int x, int y – Розміри початкової матриці  double\* m – Початкова матриця  int new\_x, int new\_y – Розміри вихідної матриці  double\* new\_m – Вихідна матриця | Копіює початкову матрицю до вихідної матриці, заповнює усі інші позиції нулями.  [Див. блок-схему 2.](#block2) |
| void PadDown() | int x, int y – Розміри початкової матриці  double\* m – Початкова матриця  int new\_x, int new\_y – Розміри вихідної матриці  double\* new\_m – Вихідна матриця | Копіює вхідну матрицю до вихідної, ігноруючи невикористані елементи.  [Див. блок-схему 3.](#block3) |
| int NextPowerOfTwo() | int a – Число, відносно якого знаходиться степінь двійки | Повертає першу степінь двійки, що білше за задане число.  [Див. блок-схему 4.](#block4) |
| bool IsPowerOfTwo() | int n – Число, що перевіряється | Перевіряє чи дане число є степенем двійки за допомогою біт-операції.  Див. лістинг програми. |
| void TransposeMatrix() | int& x, int& y – Розміри матриці  double\* m – Матриця | Транспонує матрицю, алгоритм розділений на дві частини: квадратні матриці та прямокутні матриці.  [Див. блок-схему 5.](#block5) |
| void AddMatrices() | int x, int y – Розміри вхідних та вихідної матриці  double\* a, double\* b – Вхідні матриці double\* c – Вихідна матриця | Додає дві вхідні матриці та записує дані в вихідну матрицю.  [Див блок-схему 6.](#block6) |
| void SubtractMatrices() | int x, int y – Розміри вхідних та вихідної матриці  double\* a, double\* b – Вхідні матриці double\* c – Вихідна матриця | Віднімає дві вхідні матриці та записує дані в вихідну матрицю.  [Алгоритм аналогічний блок-схемі 6.](#block6) |
| void ScalarMultiplyMatrix() | int x, int y – Розміри матриці  double\* m – Матриця  double mult – Множник | Множить вхідну матрицю на множник.  [Див блок-схему 7.](#block7) |
| void DivideMatrix() | int x, int y – Розміри матриці  double\* m – Матриця  double div – Дільник | Ділить вхідну матрицю на дільник. [Алгоритм аналогічний блок-схемі 7.](#block7) |
| void StandardMultiply() | int a\_x, int a\_y,  int b\_x, int b\_y – Розміри вхідних матриць  int c\_x, int c\_y – Розміри вихідної матриці  double\* a, double\* b – Вхідні матриці  double\* c – Вихідна матриця | Множить дві вхідні матриці та записує результат у вихідну матрицю за допомогою звичайного алгоритму множення.  [Див. блок-схему 8.](#block8) |
| void StrassenMultiply() | int a\_x, int a\_y,  int b\_x, int b\_y – Розміри вхідних матриць  int c\_x, int c\_y – Розміри вихідної матриці  double\* a, double\* b – Вхідні матриці  double\* c – Вихідна матриця | Множить дві вхідні матриці та записує результат у вихідну матрицю за допомогою алгоритму Штрассена.  [Див. блок-схему 9](#block9) та розділ 1. |
| void GetSubMatrix() | double\* mat – Вхідна матриця  double\* sub\_mat – Вихідна матриця  int curr\_row, int curr\_col – Позиція даного елемента  int mat\_size – Розмір вхідної матриці  int sub\_size – Розмір вихідної матриці | Записує обраний мінор вхідної матриці до вихідної матриці методом викреслення.  [Див. блок-схему 10.](#block10) |
| double GetDeterminant | double\* matrix – Матриця  int n – Розмір матриці | Знаходить визначник заданої матриці методом розкладанням за рядком.  [Див. блок-схему 11.](#block11) |
| void MatrixOfMinors() | double\* matrix – Вхідна матриця  double\* min\_matrix – Вихідна матриця  int n – Розмір матриць | Рахує матрицю мінорів відносно вхідної матриці.  [Див. блок-схему 12.](#block12) |
| void MatrixOfCofactors() | double\* matrix – Матриця  int n – Розмір матриці | Знаходить матрицю кофакторів.  [Див. блок-схему 13.](#block13) |
| void InverseMatrix() | double\* matrix – Вхідна матриця  double\* inv\_matrix – Вихідна матриця  int n – Розмір матриць | Знаходить обернену матрицю відносно вхідної матриці.  [Див. блок-схему 14.](#block14) |
| void RotateMatrix() | int a\_x, int a\_y – Розмір вхідної матриці  int b\_x, int b\_y – Розмір вихідної матриці  double\* a – Вхідна матриця  double\* b – Вихідна матриця  int rotate – Вибір кута обертання | Обертає вхідну матрицю на 90, 180 та 270 градусів, та записує її до вихідної.  [Див. блок-схему 15.](#block15) |

## **2.4. Максимальне завантаження**

Завдяки використанню динамічних масивів для означення матриць в програмі, розміри заданих матриць є обмежені тільки розміром оперативної пам’яті комп’ютера користувача. Але, для зменшення часу роботи програми, та для спрощення процесу введення розмірів матриць та даних матриць, розміри було обмежено до діапазону 1 – 99 на 1 – 99.

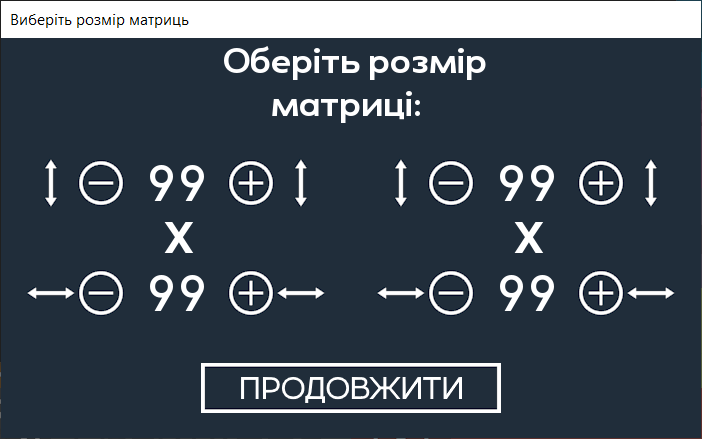


Рисунок 2 – Максимальний розмір двох матриць при множенні.

Винятком до цього є операції знаходження визначника та обернення матриць. Через обмеження використаних алгоритмів, максимальний можливий розмір матриць для цих операцій був обмежений до 1 – 12 на 1 – 12.



Рисунок 3 – Максимальний розмір двох матриць при знаходженні визначника та обернення матриці.

## **2.5. Вхідні та вихідні дані**

Частина вхідних даних, а саме вибір операції, вибір розмірів матриць та вихід з програми, виконується за допомогою курсора. Введення значень елементів матриць, та вибір зберігання результатів до файлу виконується в консолі, за допомогою клавіатури.

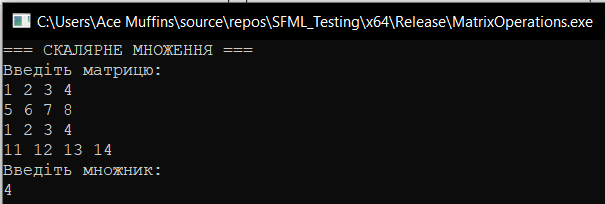


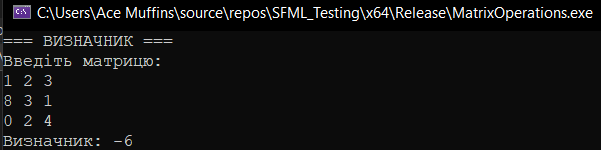
Рисунок 4 – Приклад введення матриці та множника при операції

скалярного множення.

В даній програмі є 4 типи вихідних даних:

* Дійсне число
* Матриця дійсних чисел
* Помилка
* Текстовий файл

Дійсне число виводиться лише при обрахунку визначника.

  
Рисунок 5 – Приклад виведення визначника.

Матриця дійсних чисел виводиться при всіх операціях, за винятком знаходження визначника та помилки знаходження оберненої матриці.

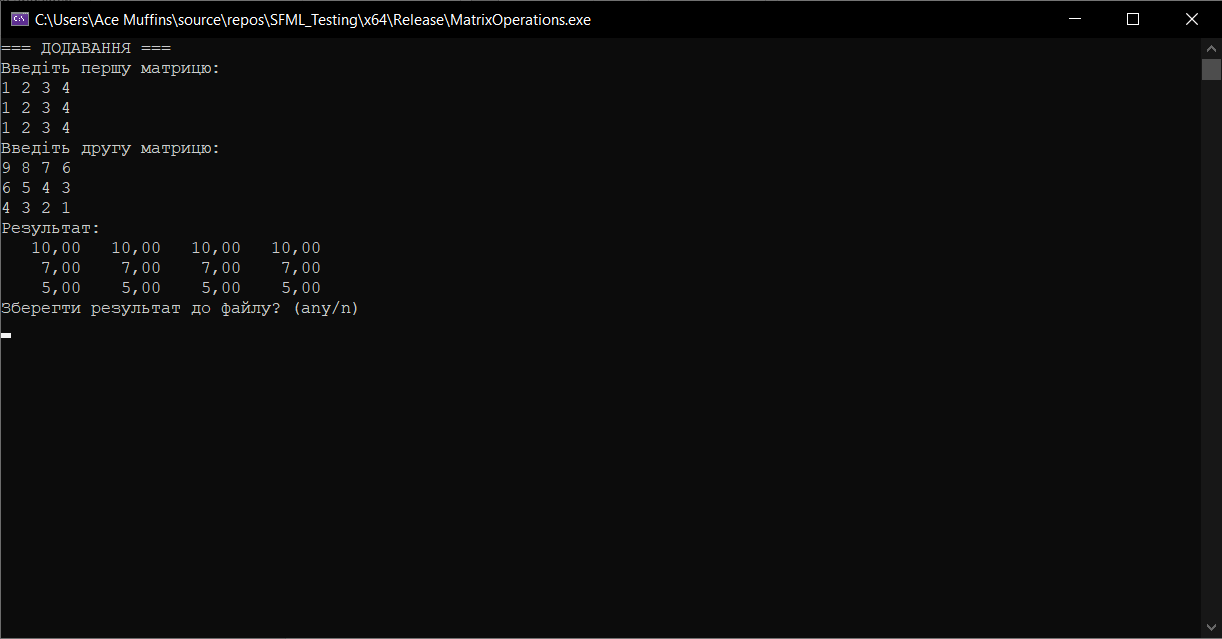


Рисунок 6 – Приклад виведення матриці дійсних чисел після операції додавання.

Помилка виводиться лише за операції обернення матриці, якщо визначник даної матриці є нулем. Після виведення помилки, програма запитує користувача ввести матрицю знову.

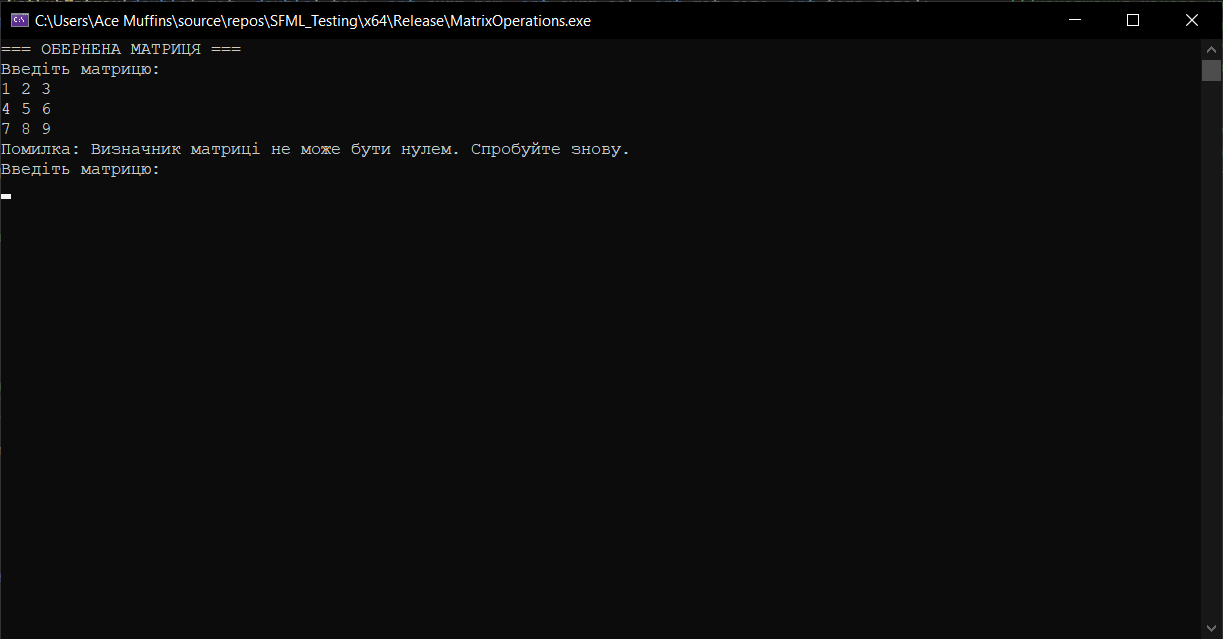
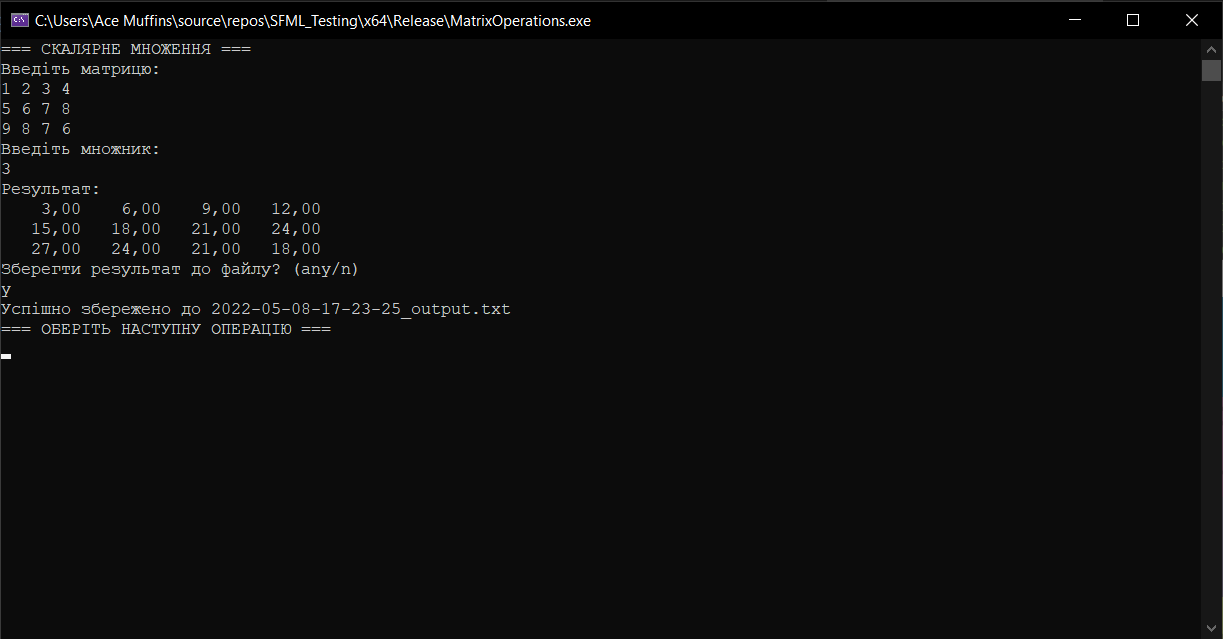
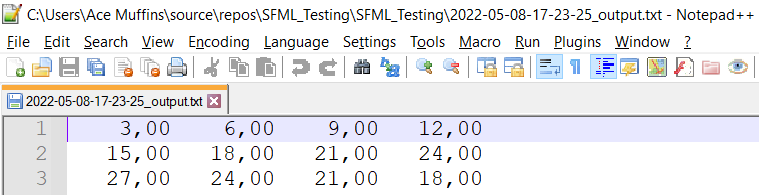


Рисунок 7 – Приклад виведення помилки після операції обернення.

Виведену матрицю можливо зберегти до текстового файлу. Для цього, після виведення матриці, потрібно ввести будь-який символ, окрім літери n.





Рисунки 8 і 9 – Приклад збереження виведеної матриці до файлу.

## **2.6. Керівництво користувачу**

Для того щоб встановити дану програму, потрібно розархівувати архів “Matrix Calculator.zip” у будь-яку папку комп’ютера.

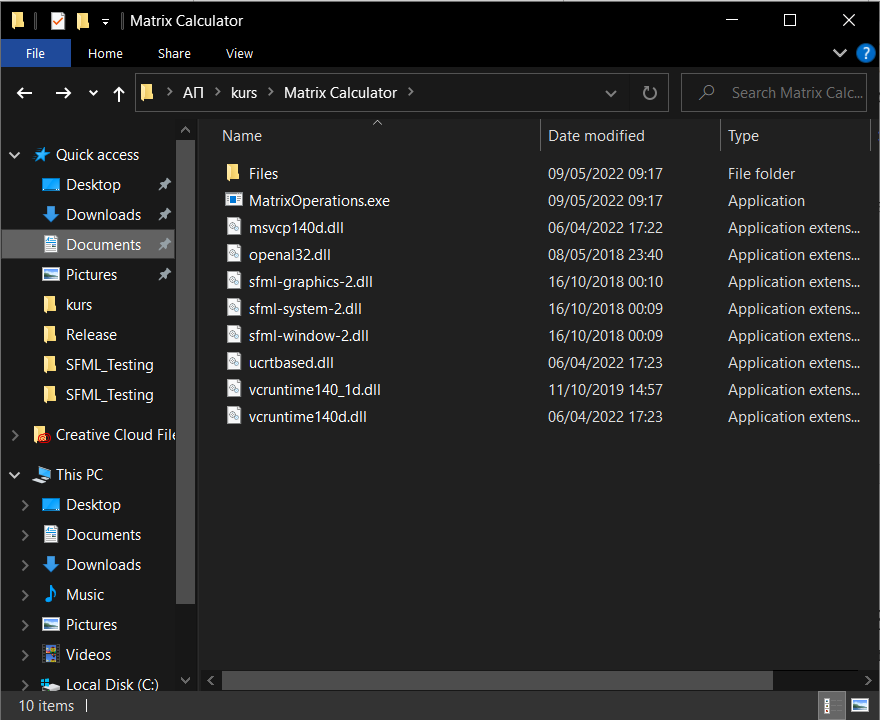
****

Рисунок 10 – Файли, що знаходяться в архіві “Matrix Calculator.zip”.

Для того, щоб запустити програму, потрібно відкрити файл “MatrixCalculator.exe”. Після відкриття, на екран висвітиться 2 вікна: консоль, яка буди приймати вхідні дані, та меню вибору операції.

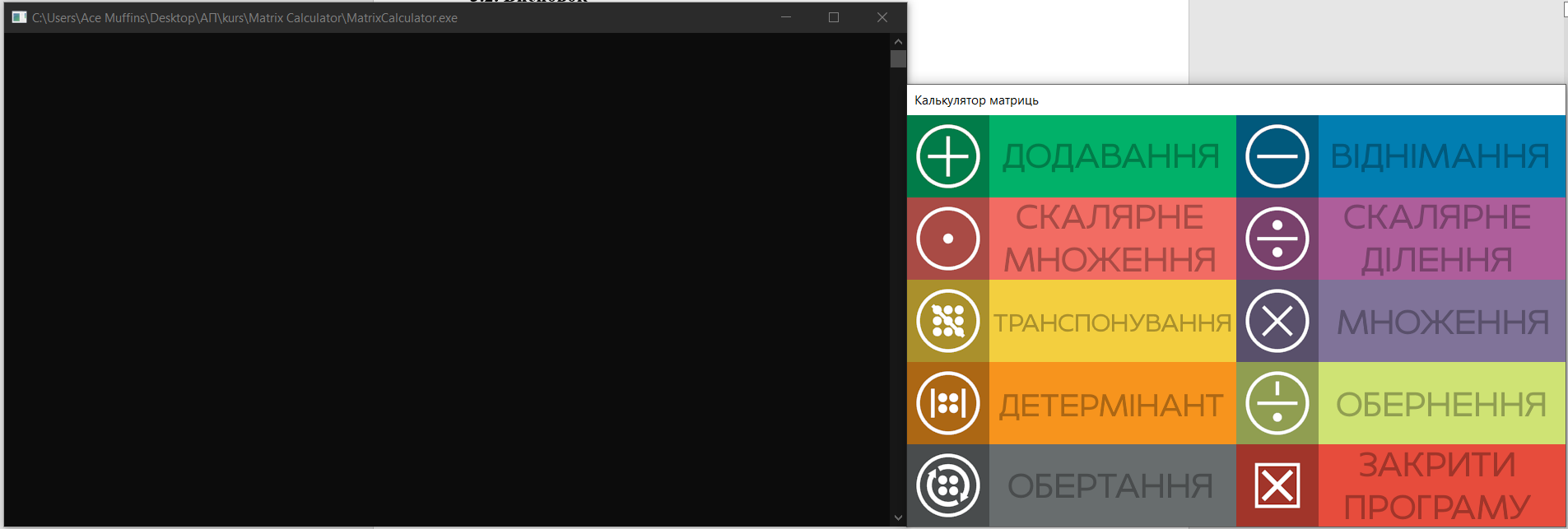


Рисунок 11 – Меню вибору операцій та консоль.

Після вибору операції, висвітиться вікно, в якому потрібно вказати розмір введеної матриці. В залежності від операції, можливо буде ввести:

* Квадратну матрицю – Детермінант, обернення.
* Прямокутну матрицю – Додавання, віднімання, скалярне множення, скалярне ділення, транспонування
* Дві матриці – Множення.
* Прямокутну матрицю з вибором кута – Обертання.



Рисунок 12 – Вікно введення розміру матриць.

Далі, натисніть кнопку “Продовжити” та перейдіть до консолі. В ній, введіть вибрану вами матрицю (або декілька) та будь-які потрібні допоміжні дані, такі як множник або дільник. Після введення усіх потрібних даних, в консоль виведеться результуюча матриця, з можливістю збереження її до файлу. Дл того щоб зберегти матрицю в текстовий файл, потрібно ввести будь-який символ, окрім літери “n”. Файл буде збережено до папки, в які знаходиться програма.

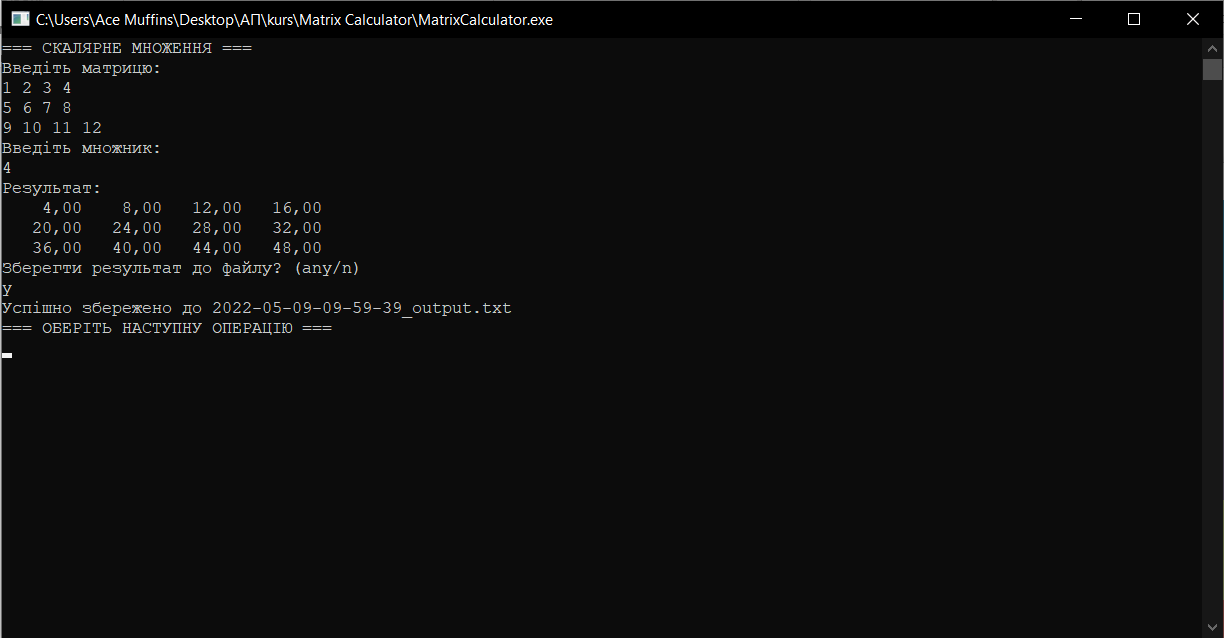


Рисунок 13 – Приклад операції скалярного множення, та збереження результату

до файлу.

Даний процес може бути повторений стільки, скільки потрібно користувачу. Після цього, програму можна закрити за допомогою програмної панелі, або кнопки “Закрити Програму”.

## **2.7. Керівництво розробнику**

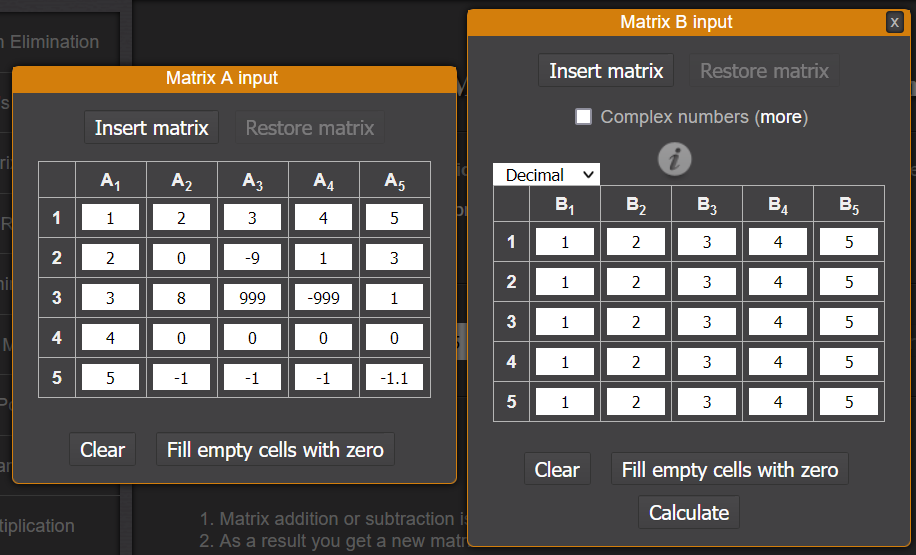
За вибору алгоритмів для виконання поставленої задачі, було знайдено кілька альтернатив до тих, що були використані в остаточній версії програми. Серед них, найбілша різниця є в алгоритмі знаходження визначника матриці. Даний алгоритм є рекурсивним, та має складність , що не є оптимальним, але будь-які інші алгоритми використовують операції, що не оптимальними за мого відображення динамічного двомірного масиву. Також слід зазначити, що даний алгоритм працює до матриць розміром , що, на мою думку, є достатнім для будь-яких практичних задач, що будуть розв’язані на такому матричному калькуляторі.

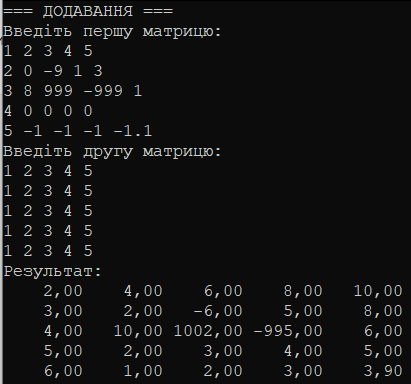
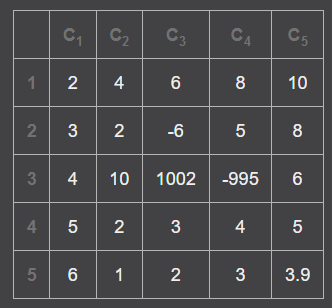
# **3. Висновки**

## **3.1. Перевірка справності програми**

В даному розіді буде перевірена справність алгоритмів програми на достатньо складних прикладах, з порівнянням вихідних даних з іншими матричними калькуляторами. Для спрощення цієї задачі, деякі операції будуть згрупповані.

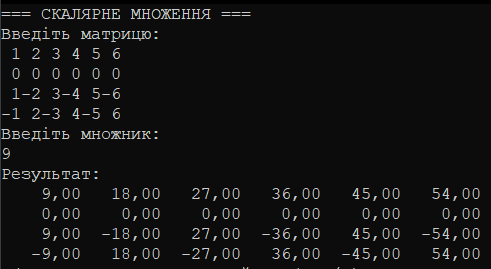
* Додавання\Віднімання

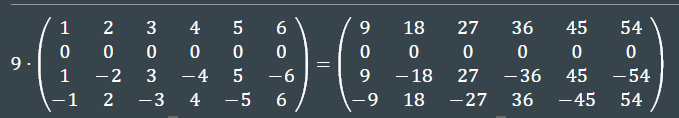




Рисунки 14-16 – Перевірка операції додавання.

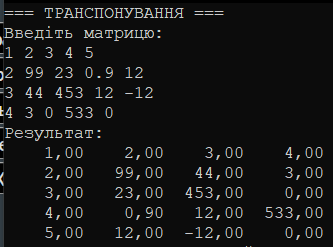
* Скалярне множення\Скалярне ділення

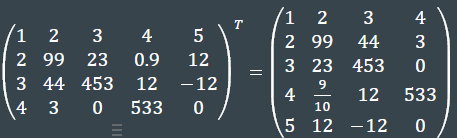




Рисунки 17-18 – Перевірка операції скалярного множення.

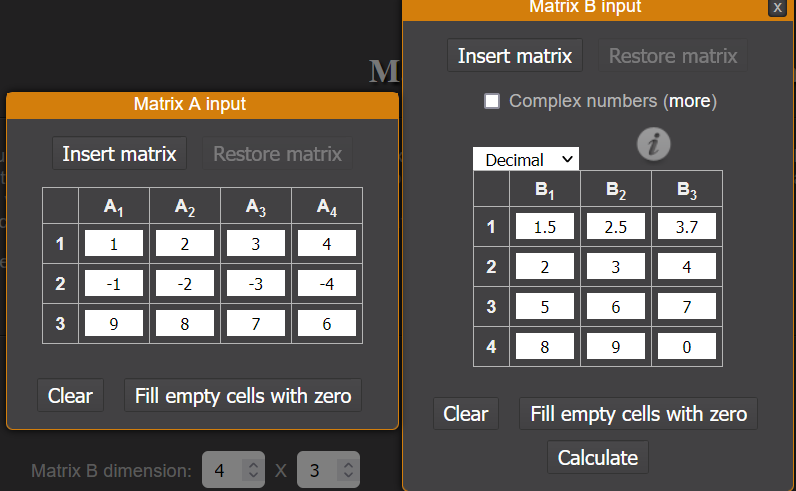
* Транспонування

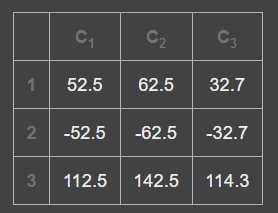
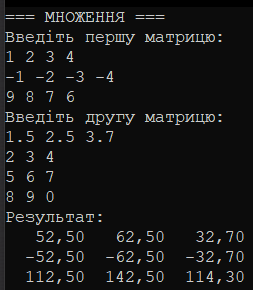




Рисунки 19-20 – Перевірка операції транспонування.

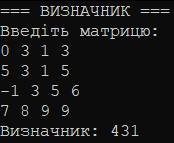
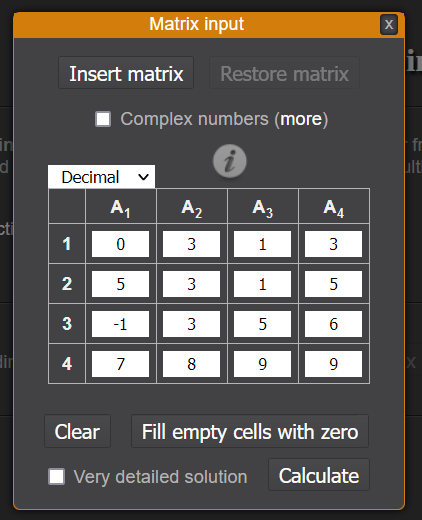
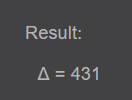
* Множення



‘ 

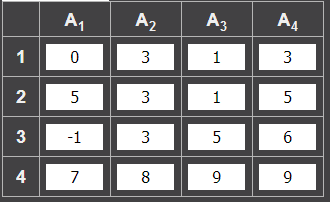
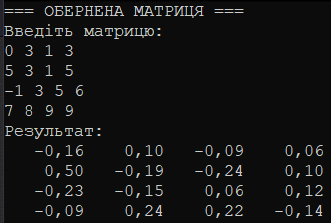
Рисунки 21-23 – Перевірка операції множення.

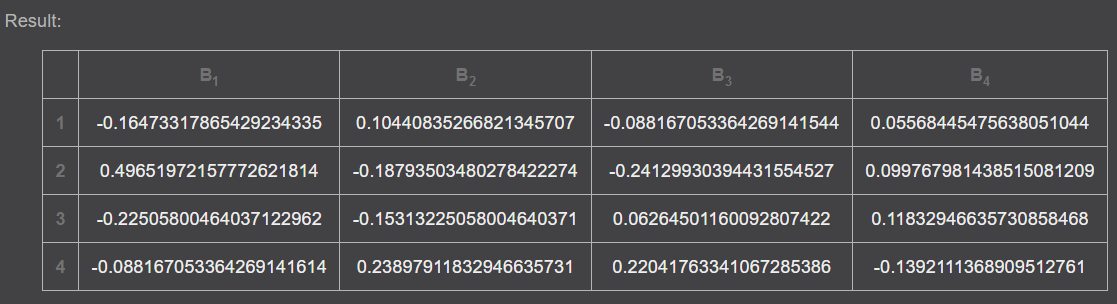
* Знаходження детермінанту

Рисунки 24-26 – Перевірка операції знаходження визначника.

* Обернення матриці





Рисунки 27-29 – Перевірка операції обернення матриці.

## **3.2. Висновок**

В результаті виконання курсової роботи була розроблена програма, що дозволяє звичайному користувачу легко оперувати матрицями та виконувати матричні операції. Також була вивчена графічна бібліотека SFML, її переваги та обмеження. Програма використовує мінімальну кількість оперативної пам’яті, доходячи до максимуму в 100 МБ при виконанні усіх операцій.

В першому розділі було розглянено операції, що були імплементовані в програму. В другому, було описана логічна структура програми, використані бібліотеки та IDE, її призначення, та керівництва користувачу та розробнику. В третьому – було перевірено справність результуючої програми, та підбито висновок.

# **Додатки**

## **Література**

1. Інтернет-енциклопедія “Вікіпедія”, Матриця - <https://uk.wikipedia.org/wiki/Матриця_(математика)>
2. Vedantu, Application of Matrices in Science, Commerce and Social Science Fields - <https://www.vedantu.com/maths/application-of-matrices>
3. Інтернет-енциклопедія “Вікіпедія”, Множення матриць - [https://uk.wikipedia.org/wiki/Множення\_матриць](https://uk.wikipedia.org/wiki/%D0%9C%D0%BD%D0%BE%D0%B6%D0%B5%D0%BD%D0%BD%D1%8F_%D0%BC%D0%B0%D1%82%D1%80%D0%B8%D1%86%D1%8C)
4. Інтернет-енциклопедія “Вікіпедія”, Алгоритм Штрассена - [https://uk.wikipedia.org/wiki/Алгоритм\_Штрассена](https://uk.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D0%A8%D1%82%D1%80%D0%B0%D1%81%D1%81%D0%B5%D0%BD%D0%B0)
5. Mathisfun, Inverse of a Matrix using Minors, Cofactors and Adjugate - <https://www.mathsisfun.com/algebra/matrix-inverse-minors-cofactors-adjugate.html>
6. Інтернет-енциклопедія “Вікіпедія”, Визначник - [https://uk.wikipedia.org/wiki/Визначник](https://uk.wikipedia.org/wiki/%D0%92%D0%B8%D0%B7%D0%BD%D0%B0%D1%87%D0%BD%D0%B8%D0%BA)
7. Інтернет-енциклопедія “Вікіпедія”, Невироджена матриця - [https://uk.wikipedia.org/wiki/Невироджена\_матриця](https://uk.wikipedia.org/wiki/%D0%9D%D0%B5%D0%B2%D0%B8%D1%80%D0%BE%D0%B4%D0%B6%D0%B5%D0%BD%D0%B0_%D0%BC%D0%B0%D1%82%D1%80%D0%B8%D1%86%D1%8F)
8. Reshish, Калькулятор Матриць - <https://matrix.reshish.com/>
9. Шон Ерон Андерсон, Bit Twiddling Hacks - <http://www.graphics.stanford.edu/~seander/bithacks.html>
10. Воєводин, Кузнєцов, Матриці та обчислення - <http://mathscinet.ru/files/VoevodinKuznecov.pdf>

## **Лістинг програми**

/\*

ENG:

Developed by Oleksii Verhola, all code and graphics by me!

Matrix Calculator 1.1.0-11.04.22

A simple graphic program to demonstrate some basic (and less basic)

algorithms and operations one can perform on matrices!

NOTE: The determinant and inverse operations take an obscene amount

of time if you put a matrix of size 12 or bigger in them. Too bad!

UKR:

Розроблено Верхолою Олексієм, весь код і графіка зроблені мною!

Matrix Calculator 1.1.0-11.04.22

Проста програма для демонстрації простих та складних операцій, що можливо

застосовувати на матрицях.

\*/

#include <SFML/Graphics.hpp>

#include <iostream>

#include <locale>

#include <stdio.h>

#include <windows.h>

#include <string>

#include <cmath>

#include <stdlib.h>

#include <iomanip>

#include <ctime>

#include <sstream>

void InputMatrix(int x, int y, double\* m); //Функція введення матриці

void OutputMatrix(int x, int y, double\* m); //Функція виведення матриці та збереження до файлу

void ZeroMatrix(int x, int y, double\* m); //Функція обнулення матриці (усі елементи = 0)

void TransposeMatrix(int& x, int& y, double\* m); //Транспонування матриці

void AddMatrices(int x, int y, double\* a, double\* b, double\* c); //Додавання матриць

void SubtractMatricies(int x, int y, double\* a, double\* b, double\* c); //Віднімання матриць

void ScalarMultiplyMatrix(int x, int y, double\* a, double mult); //Скалярне множення матриць

void DivideMatrix(int x, int y, double\* a, double div); //Скалярне ділення матриць

void StandardMultiply(int a\_x, int a\_y, int b\_x, int b\_y, int c\_x, int c\_y, double\* a, double\* b, double\* c); //Простий алгоритм множення матриць

void StrassenMultiply(int a\_x, int a\_y, int b\_x, int b\_y, int c\_x, int c\_y, double\* a, double\* b, double\* c); //Алгоритм Штрассена множення матриць

void GetSubMatrix(double\* mat, double\* temp, int curr\_row, int curr\_col, int mat\_size, int temp\_size); //Знаходження мінора матриці шляхом викреслювання

double GetDeterminant(double\* matrix, int n); //Обчислення визначника за схемою розкладу визначника за рядком

void MatrixOfMinors(double\* matrix, double\* min\_matrix, int n); //Знаходження матриці мінорів

void MatrixOfCofactors(double\* matrix, int n); //Знаходження матриці кофакторів

void InverseMatrix(double\* matrix, double\* inv\_matrix, int n); //Обернення матриці

void RotateMatrix(int a\_x, int a\_y, int b\_x, int b\_y, double\* a, double\* b, int rotate); //Обертання матриці

void PadUp(int x, int y, double\* m, int new\_x, int new\_y, double\* new\_m); //Збільшення розмірів матриці до квадратної з розміром стпені 2х

void PadDown(int x, int y, double\* m, int new\_x, int new\_y, double\* new\_m); //Зменьшення до потрібних розмірів

int NextPowerOfTwo(int a); //Знаходження наступного степеня 2

bool IsPowerOfTwo(int num); //Перевірка чи число є степенем 2

/\*Блок виборів розміру матриць для різних випадків

Усі функції створюють додаткові вікна та заповнюють їх відповідними елементами (Текст, кнопки, тд)

DimChoose1 - Вибір розмірів для прямокутних матриць (2 матриці одного розміру)

DimChoose2 - Вибір розмірів для прямокутної матриці (1 матриця)

DimChoose3 - Вибір розміру для квадратної матриці

DimChoose4 - Вибір розмірів для двох прямокутних матриць для операції множення (розміри автоматично обмежуються)

DimChoose5 - Вибір розмірів для прямокутної матриці та кута повороту

\*/

void DimChoose1(int &x, int &y) {

int winH = 400;

int winW = 350;

sf::RenderWindow windowChoose(sf::VideoMode(winW, winH), L"Виберіть розмір матриць", sf::Style::Titlebar);

windowChoose.setFramerateLimit(60);

int a\_x = 1;

int a\_y = 1;

sf::Font fontNumbers;

sf::Font fontText;

fontNumbers.loadFromFile("Files//apercu.ttf");

fontText.loadFromFile("Files//moula.ttf");

sf::Text textX;

sf::Text textY;

sf::Text textNa;

sf::Text textChoose;

sf::Texture buttonPlusUnfocused;

sf::Texture buttonMinusUnfocused;

sf::Texture buttonPlusFocused;

sf::Texture buttonMinusFocused;

sf::Texture buttonCloseUnfocused;

sf::Texture buttonCloseFocused;

sf::Texture buttonUpDown;

sf::Texture buttonLeftRight;

buttonPlusUnfocused.loadFromFile("Files\\plus.png");

buttonMinusUnfocused.loadFromFile("Files\\minus.png");

buttonPlusFocused.loadFromFile("Files\\plus\_f.png");

buttonMinusFocused.loadFromFile("Files\\minus\_f.png");

buttonCloseUnfocused.loadFromFile("Files\\continue.png");

buttonCloseFocused.loadFromFile("Files\\continue\_f.png");

buttonUpDown.loadFromFile("Files\\arrow\_up.png");

buttonLeftRight.loadFromFile("Files\\arrow\_side.png");

textNa.setFont(fontText);

textNa.setCharacterSize(50);

textNa.setFillColor(sf::Color::White);

textNa.setString(L"X");

textNa.setPosition(163, 165);

textChoose.setFont(fontText);

textChoose.setCharacterSize(40);

textChoose.setFillColor(sf::Color::White);

textChoose.setString(L"Оберіть розмір\n матриць:");

textChoose.setPosition(47, -5);

textX.setFont(fontNumbers);

textX.setCharacterSize(50);

textX.setFillColor(sf::Color::White);

textX.setString('0' + std::to\_string(a\_x));

textX.setPosition(145, 113);

textY.setFont(fontNumbers);

textY.setCharacterSize(50);

textY.setFillColor(sf::Color::White);

textY.setString('0' + std::to\_string(a\_y));

textY.setPosition(145, 223);

sf::RectangleShape close(sf::Vector2f(300.f, 50.f));

close.setFillColor(sf::Color::White);

close.setPosition(25, 325);

close.setTexture(&buttonCloseUnfocused);

sf::RectangleShape plusX(sf::Vector2f(50.f, 50.f));

sf::RectangleShape minusX(sf::Vector2f(50.f, 50.f));

plusX.setPosition(225, 120);

minusX.setPosition(75, 120);

plusX.setTexture(&buttonPlusUnfocused);

minusX.setTexture(&buttonMinusUnfocused);

sf::RectangleShape plusY(sf::Vector2f(50.f, 50.f));

sf::RectangleShape minusY(sf::Vector2f(50.f, 50.f));

plusY.setPosition(225, 230);

minusY.setPosition(75, 230);

plusY.setTexture(&buttonPlusUnfocused);

minusY.setTexture(&buttonMinusUnfocused);

sf::RectangleShape arrowUpDown(sf::Vector2f(50.f, 50.f));

arrowUpDown.setPosition(25, 120);

arrowUpDown.setTexture(&buttonUpDown);

sf::RectangleShape arrowLeftRight(sf::Vector2f(50.f, 50.f));

arrowLeftRight.setPosition(25, 230);

arrowLeftRight.setTexture(&buttonLeftRight);

sf::RectangleShape arrowUpDown1(sf::Vector2f(50.f, 50.f));

arrowUpDown1.setPosition(275, 120);

arrowUpDown1.setTexture(&buttonUpDown);

sf::RectangleShape arrowLeftRight1(sf::Vector2f(50.f, 50.f));

arrowLeftRight1.setPosition(275, 230);

arrowLeftRight1.setTexture(&buttonLeftRight);

windowChoose.setFramerateLimit(60);

bool pressed = false;

while (windowChoose.isOpen()) {

sf::Event event;

while (windowChoose.pollEvent(event))

{

if (event.type == sf::Event::Closed)

windowChoose.close();

}

if (sf::Mouse::isButtonPressed(sf::Mouse::Left)) {

if (pressed) continue;

else {

if (plusX.getGlobalBounds().contains(windowChoose.mapPixelToCoords(sf::Mouse::getPosition(windowChoose)))) {

a\_x++;

}

if (minusX.getGlobalBounds().contains(windowChoose.mapPixelToCoords(sf::Mouse::getPosition(windowChoose)))) {

a\_x--;

}

if (plusY.getGlobalBounds().contains(windowChoose.mapPixelToCoords(sf::Mouse::getPosition(windowChoose)))) {

a\_y++;

}

if (minusY.getGlobalBounds().contains(windowChoose.mapPixelToCoords(sf::Mouse::getPosition(windowChoose)))) {

a\_y--;

}

if (close.getGlobalBounds().contains(windowChoose.mapPixelToCoords(sf::Mouse::getPosition(windowChoose)))) {

windowChoose.close();

}

pressed = true;

}

continue;

}

pressed = false;

if (a\_x < 1) {

a\_x = 1;

}

if (a\_x > 99) {

a\_x = 99;

}

if (a\_x < 10) {

textX.setString('0' + std::to\_string(a\_x));

}

else {

textX.setString(std::to\_string(a\_x));

}

if (a\_y < 1) {

a\_y = 1;

}

if (a\_y > 99) {

a\_y = 99;

}

if (a\_y < 10) {

textY.setString('0' + std::to\_string(a\_y));

}

else {

textY.setString(std::to\_string(a\_y));

}

if (plusX.getGlobalBounds().contains(windowChoose.mapPixelToCoords(sf::Mouse::getPosition(windowChoose)))) {

plusX.setTexture(&buttonPlusFocused);

}

else {

plusX.setTexture(&buttonPlusUnfocused);

}

if (plusY.getGlobalBounds().contains(windowChoose.mapPixelToCoords(sf::Mouse::getPosition(windowChoose)))) {

plusY.setTexture(&buttonPlusFocused);

}

else {

plusY.setTexture(&buttonPlusUnfocused);

}

if (minusX.getGlobalBounds().contains(windowChoose.mapPixelToCoords(sf::Mouse::getPosition(windowChoose)))) {

minusX.setTexture(&buttonMinusFocused);

}

else {

minusX.setTexture(&buttonMinusUnfocused);

}

if (minusY.getGlobalBounds().contains(windowChoose.mapPixelToCoords(sf::Mouse::getPosition(windowChoose)))) {

minusY.setTexture(&buttonMinusFocused);

}

else {

minusY.setTexture(&buttonMinusUnfocused);

}

if (close.getGlobalBounds().contains(windowChoose.mapPixelToCoords(sf::Mouse::getPosition(windowChoose)))) {

close.setTexture(&buttonCloseFocused);

}

else {

close.setTexture(&buttonCloseUnfocused);

}

windowChoose.clear(sf::Color(32, 45, 58, 255));

windowChoose.draw(plusX);

windowChoose.draw(minusX);

windowChoose.draw(plusY);

windowChoose.draw(minusY);

windowChoose.draw(close);

windowChoose.draw(textX);

windowChoose.draw(textY);

windowChoose.draw(textNa);

windowChoose.draw(arrowUpDown);

windowChoose.draw(arrowLeftRight);

windowChoose.draw(arrowUpDown1);

windowChoose.draw(arrowLeftRight1);

windowChoose.draw(textChoose);

windowChoose.display();

x = a\_x;

y = a\_y;

}

}

void DimChoose2(int &x, int &y) {

int winH = 400;

int winW = 350;

sf::RenderWindow windowChoose(sf::VideoMode(winW, winH), L"Виберіть розмір матриці", sf::Style::Titlebar);

windowChoose.setFramerateLimit(60);

int a\_x = 1;

int a\_y = 1;

sf::Font fontNumbers;

sf::Font fontText;

fontNumbers.loadFromFile("Files//apercu.ttf");

fontText.loadFromFile("Files//moula.ttf");

sf::Text textX;

sf::Text textY;

sf::Text textNa;

sf::Text textChoose;

sf::Texture buttonPlusUnfocused;

sf::Texture buttonMinusUnfocused;

sf::Texture buttonPlusFocused;

sf::Texture buttonMinusFocused;

sf::Texture buttonCloseUnfocused;

sf::Texture buttonCloseFocused;

sf::Texture buttonUpDown;

sf::Texture buttonLeftRight;

buttonPlusUnfocused.loadFromFile("Files\\plus.png");

buttonMinusUnfocused.loadFromFile("Files\\minus.png");

buttonPlusFocused.loadFromFile("Files\\plus\_f.png");

buttonMinusFocused.loadFromFile("Files\\minus\_f.png");

buttonCloseUnfocused.loadFromFile("Files\\continue.png");

buttonCloseFocused.loadFromFile("Files\\continue\_f.png");

buttonUpDown.loadFromFile("Files\\arrow\_up.png");

buttonLeftRight.loadFromFile("Files\\arrow\_side.png");

textNa.setFont(fontText);

textNa.setCharacterSize(50);

textNa.setFillColor(sf::Color::White);

textNa.setString(L"X");

textNa.setPosition(163, 165);

textChoose.setFont(fontText);

textChoose.setCharacterSize(40);

textChoose.setFillColor(sf::Color::White);

textChoose.setString(L"Оберіть розмір\n матриці:");

textChoose.setPosition(47, -5);

textX.setFont(fontNumbers);

textX.setCharacterSize(50);

textX.setFillColor(sf::Color::White);

textX.setString('0' + std::to\_string(a\_x));

textX.setPosition(145, 113);

textY.setFont(fontNumbers);

textY.setCharacterSize(50);

textY.setFillColor(sf::Color::White);

textY.setString('0' + std::to\_string(a\_y));

textY.setPosition(145, 223);

sf::RectangleShape close(sf::Vector2f(300.f, 50.f));

close.setFillColor(sf::Color::White);

close.setPosition(25, 325);

close.setTexture(&buttonCloseUnfocused);

sf::RectangleShape plusX(sf::Vector2f(50.f, 50.f));

sf::RectangleShape minusX(sf::Vector2f(50.f, 50.f));

plusX.setPosition(225, 120);

minusX.setPosition(75, 120);

plusX.setTexture(&buttonPlusUnfocused);

minusX.setTexture(&buttonMinusUnfocused);

sf::RectangleShape plusY(sf::Vector2f(50.f, 50.f));

sf::RectangleShape minusY(sf::Vector2f(50.f, 50.f));

plusY.setPosition(225, 230);

minusY.setPosition(75, 230);

plusY.setTexture(&buttonPlusUnfocused);

minusY.setTexture(&buttonMinusUnfocused);

sf::RectangleShape arrowUpDown(sf::Vector2f(50.f, 50.f));

arrowUpDown.setPosition(25, 120);

arrowUpDown.setTexture(&buttonUpDown);

sf::RectangleShape arrowLeftRight(sf::Vector2f(50.f, 50.f));

arrowLeftRight.setPosition(25, 230);

arrowLeftRight.setTexture(&buttonLeftRight);

sf::RectangleShape arrowUpDown1(sf::Vector2f(50.f, 50.f));

arrowUpDown1.setPosition(275, 120);

arrowUpDown1.setTexture(&buttonUpDown);

sf::RectangleShape arrowLeftRight1(sf::Vector2f(50.f, 50.f));

arrowLeftRight1.setPosition(275, 230);

arrowLeftRight1.setTexture(&buttonLeftRight);

windowChoose.setFramerateLimit(60);

bool pressed = false;

while (windowChoose.isOpen()) {

sf::Event event;

while (windowChoose.pollEvent(event))

{

if (event.type == sf::Event::Closed)

windowChoose.close();

}

if (sf::Mouse::isButtonPressed(sf::Mouse::Left)) {

if (pressed) continue;

else {

if (plusX.getGlobalBounds().contains(windowChoose.mapPixelToCoords(sf::Mouse::getPosition(windowChoose)))) {

a\_x++;

}

if (minusX.getGlobalBounds().contains(windowChoose.mapPixelToCoords(sf::Mouse::getPosition(windowChoose)))) {

a\_x--;

}

if (plusY.getGlobalBounds().contains(windowChoose.mapPixelToCoords(sf::Mouse::getPosition(windowChoose)))) {

a\_y++;

}

if (minusY.getGlobalBounds().contains(windowChoose.mapPixelToCoords(sf::Mouse::getPosition(windowChoose)))) {

a\_y--;

}

if (close.getGlobalBounds().contains(windowChoose.mapPixelToCoords(sf::Mouse::getPosition(windowChoose)))) {

windowChoose.close();

}

pressed = true;

}

continue;

}

pressed = false;

if (a\_x < 1) {

a\_x = 1;

}

if (a\_x > 99) {

a\_x = 99;

}

if (a\_x < 10) {

textX.setString('0' + std::to\_string(a\_x));

}

else {

textX.setString(std::to\_string(a\_x));

}

if (a\_y < 1) {

a\_y = 1;

}

if (a\_y > 99) {

a\_y = 99;

}

if (a\_y < 10) {

textY.setString('0' + std::to\_string(a\_y));

}

else {

textY.setString(std::to\_string(a\_y));

}

if (plusX.getGlobalBounds().contains(windowChoose.mapPixelToCoords(sf::Mouse::getPosition(windowChoose)))) {

plusX.setTexture(&buttonPlusFocused);

}

else {

plusX.setTexture(&buttonPlusUnfocused);

}

if (plusY.getGlobalBounds().contains(windowChoose.mapPixelToCoords(sf::Mouse::getPosition(windowChoose)))) {

plusY.setTexture(&buttonPlusFocused);

}

else {

plusY.setTexture(&buttonPlusUnfocused);

}

if (minusX.getGlobalBounds().contains(windowChoose.mapPixelToCoords(sf::Mouse::getPosition(windowChoose)))) {

minusX.setTexture(&buttonMinusFocused);

}

else {

minusX.setTexture(&buttonMinusUnfocused);

}

if (minusY.getGlobalBounds().contains(windowChoose.mapPixelToCoords(sf::Mouse::getPosition(windowChoose)))) {

minusY.setTexture(&buttonMinusFocused);

}

else {

minusY.setTexture(&buttonMinusUnfocused);

}

if (close.getGlobalBounds().contains(windowChoose.mapPixelToCoords(sf::Mouse::getPosition(windowChoose)))) {

close.setTexture(&buttonCloseFocused);

}

else {

close.setTexture(&buttonCloseUnfocused);

}

windowChoose.clear(sf::Color(32, 45, 58, 255));

windowChoose.draw(plusX);

windowChoose.draw(minusX);

windowChoose.draw(plusY);

windowChoose.draw(minusY);

windowChoose.draw(close);

windowChoose.draw(textX);

windowChoose.draw(textY);

windowChoose.draw(textNa);

windowChoose.draw(arrowUpDown);

windowChoose.draw(arrowLeftRight);

windowChoose.draw(arrowUpDown1);

windowChoose.draw(arrowLeftRight1);

windowChoose.draw(textChoose);

windowChoose.display();

x = a\_x;

y = a\_y;

}

}

void DimChoose3(int &x) {

int winH = 400;

int winW = 350;

sf::RenderWindow windowChoose(sf::VideoMode(winW, winH), L"Виберіть розмір матриці", sf::Style::Titlebar);

windowChoose.setFramerateLimit(60);

int a\_x = 1;

sf::Font fontNumbers;

sf::Font fontText;

fontNumbers.loadFromFile("Files//apercu.ttf");

fontText.loadFromFile("Files//moula.ttf");

sf::Text textX;

sf::Text textY;

sf::Text textNa;

sf::Text textChoose;

sf::Texture buttonPlusUnfocused;

sf::Texture buttonMinusUnfocused;

sf::Texture buttonPlusFocused;

sf::Texture buttonMinusFocused;

sf::Texture buttonCloseUnfocused;

sf::Texture buttonCloseFocused;

sf::Texture buttonUpDown;

sf::Texture buttonLeftRight;

buttonPlusUnfocused.loadFromFile("Files\\plus.png");

buttonMinusUnfocused.loadFromFile("Files\\minus.png");

buttonPlusFocused.loadFromFile("Files\\plus\_f.png");

buttonMinusFocused.loadFromFile("Files\\minus\_f.png");

buttonCloseUnfocused.loadFromFile("Files\\continue.png");

buttonCloseFocused.loadFromFile("Files\\continue\_f.png");

buttonUpDown.loadFromFile("Files\\arrow\_up.png");

buttonLeftRight.loadFromFile("Files\\arrow\_side.png");

textNa.setFont(fontText);

textNa.setCharacterSize(50);

textNa.setFillColor(sf::Color::White);

textNa.setString(L"X");

textNa.setPosition(163, 165);

textChoose.setFont(fontText);

textChoose.setCharacterSize(40);

textChoose.setFillColor(sf::Color::White);

textChoose.setString(L"Оберіть розмір\n матриці:");

textChoose.setPosition(47, -5);

textX.setFont(fontNumbers);

textX.setCharacterSize(50);

textX.setFillColor(sf::Color::White);

textX.setString('0' + std::to\_string(a\_x));

textX.setPosition(145, 113);

textY.setFont(fontNumbers);

textY.setCharacterSize(50);

textY.setFillColor(sf::Color::White);

textY.setString('0' + std::to\_string(a\_x));

textY.setPosition(145, 223);

sf::RectangleShape close(sf::Vector2f(300.f, 50.f));

close.setFillColor(sf::Color::White);

close.setPosition(25, 325);

close.setTexture(&buttonCloseUnfocused);

sf::RectangleShape plusX(sf::Vector2f(50.f, 50.f));

sf::RectangleShape minusX(sf::Vector2f(50.f, 50.f));

plusX.setPosition(225, 120);

minusX.setPosition(75, 120);

plusX.setTexture(&buttonPlusUnfocused);

minusX.setTexture(&buttonMinusUnfocused);

sf::RectangleShape arrowUpDown(sf::Vector2f(50.f, 50.f));

arrowUpDown.setPosition(25, 120);

arrowUpDown.setTexture(&buttonUpDown);

sf::RectangleShape arrowLeftRight(sf::Vector2f(50.f, 50.f));

arrowLeftRight.setPosition(75, 230);

arrowLeftRight.setTexture(&buttonLeftRight);

sf::RectangleShape arrowUpDown1(sf::Vector2f(50.f, 50.f));

arrowUpDown1.setPosition(275, 120);

arrowUpDown1.setTexture(&buttonUpDown);

sf::RectangleShape arrowLeftRight1(sf::Vector2f(50.f, 50.f));

arrowLeftRight1.setPosition(225, 230);

arrowLeftRight1.setTexture(&buttonLeftRight);

windowChoose.setFramerateLimit(60);

bool pressed = false;

while (windowChoose.isOpen()) {

sf::Event event;

while (windowChoose.pollEvent(event))

{

if (event.type == sf::Event::Closed)

windowChoose.close();

}

if (sf::Mouse::isButtonPressed(sf::Mouse::Left)) {

if (pressed) continue;

else {

if (plusX.getGlobalBounds().contains(windowChoose.mapPixelToCoords(sf::Mouse::getPosition(windowChoose)))) {

a\_x++;

}

if (minusX.getGlobalBounds().contains(windowChoose.mapPixelToCoords(sf::Mouse::getPosition(windowChoose)))) {

a\_x--;

}

if (close.getGlobalBounds().contains(windowChoose.mapPixelToCoords(sf::Mouse::getPosition(windowChoose)))) {

windowChoose.close();

}

pressed = true;

}

continue;

}

pressed = false;

if (a\_x < 1) {

a\_x = 1;

}

if (a\_x > 12) {

a\_x = 12;

}

if (a\_x < 10) {

textX.setString('0' + std::to\_string(a\_x));

textY.setString('0' + std::to\_string(a\_x));

}

else {

textX.setString(std::to\_string(a\_x));

textY.setString(std::to\_string(a\_x));

}

if (plusX.getGlobalBounds().contains(windowChoose.mapPixelToCoords(sf::Mouse::getPosition(windowChoose)))) {

plusX.setTexture(&buttonPlusFocused);

}

else {

plusX.setTexture(&buttonPlusUnfocused);

}

if (minusX.getGlobalBounds().contains(windowChoose.mapPixelToCoords(sf::Mouse::getPosition(windowChoose)))) {

minusX.setTexture(&buttonMinusFocused);

}

else {

minusX.setTexture(&buttonMinusUnfocused);

}

if (close.getGlobalBounds().contains(windowChoose.mapPixelToCoords(sf::Mouse::getPosition(windowChoose)))) {

close.setTexture(&buttonCloseFocused);

}

else {

close.setTexture(&buttonCloseUnfocused);

}

windowChoose.clear(sf::Color(32, 45, 58, 255));

windowChoose.draw(plusX);

windowChoose.draw(minusX);

windowChoose.draw(close);

windowChoose.draw(textX);

windowChoose.draw(textY);

windowChoose.draw(textNa);

windowChoose.draw(arrowUpDown);

windowChoose.draw(arrowLeftRight);

windowChoose.draw(arrowUpDown1);

windowChoose.draw(arrowLeftRight1);

windowChoose.draw(textChoose);

windowChoose.display();

x = a\_x;

}

}

void DimChoose4(int& x1, int& y1, int& x2, int& y2) {

int winH = 400;

int winW = 700;

sf::RenderWindow windowChoose(sf::VideoMode(winW, winH), L"Виберіть розмір матриць", sf::Style::Titlebar);

windowChoose.setFramerateLimit(60);

int a\_x = 1;

int a\_y = 1;

int b\_x = 1;

int b\_y = 1;

//a\_y == b\_x

sf::Font fontNumbers;

sf::Font fontText;

fontNumbers.loadFromFile("Files//apercu.ttf");

fontText.loadFromFile("Files//moula.ttf");

sf::Text textX1;

sf::Text textY1;

sf::Text textX2;

sf::Text textY2;

sf::Text textNa1;

sf::Text textNa2;

sf::Text textChoose;

sf::Texture buttonPlusUnfocused;

sf::Texture buttonMinusUnfocused;

sf::Texture buttonPlusFocused;

sf::Texture buttonMinusFocused;

sf::Texture buttonCloseUnfocused;

sf::Texture buttonCloseFocused;

sf::Texture buttonUpDown;

sf::Texture buttonLeftRight;

buttonPlusUnfocused.loadFromFile("Files\\plus.png");

buttonMinusUnfocused.loadFromFile("Files\\minus.png");

buttonPlusFocused.loadFromFile("Files\\plus\_f.png");

buttonMinusFocused.loadFromFile("Files\\minus\_f.png");

buttonCloseUnfocused.loadFromFile("Files\\continue.png");

buttonCloseFocused.loadFromFile("Files\\continue\_f.png");

buttonUpDown.loadFromFile("Files\\arrow\_up.png");

buttonLeftRight.loadFromFile("Files\\arrow\_side.png");

textNa1.setFont(fontText);

textNa1.setCharacterSize(50);

textNa1.setFillColor(sf::Color::White);

textNa1.setString(L"X");

textNa1.setPosition(163, 165);

textNa2.setFont(fontText);

textNa2.setCharacterSize(50);

textNa2.setFillColor(sf::Color::White);

textNa2.setString(L"X");

textNa2.setPosition(163 + 350, 165);

textChoose.setFont(fontText);

textChoose.setCharacterSize(40);

textChoose.setFillColor(sf::Color::White);

textChoose.setString(L"Оберіть розмір\n матриці:");

textChoose.setPosition(47+175, -5);

textX1.setFont(fontNumbers);

textX1.setCharacterSize(50);

textX1.setFillColor(sf::Color::White);

textX1.setString('0' + std::to\_string(a\_x));

textX1.setPosition(145, 113);

textY1.setFont(fontNumbers);

textY1.setCharacterSize(50);

textY1.setFillColor(sf::Color::White);

textY1.setString('0' + std::to\_string(a\_y));

textY1.setPosition(145, 223);

textX2.setFont(fontNumbers);

textX2.setCharacterSize(50);

textX2.setFillColor(sf::Color::White);

textX2.setString('0' + std::to\_string(b\_x));

textX2.setPosition(145+350, 113);

textY2.setFont(fontNumbers);

textY2.setCharacterSize(50);

textY2.setFillColor(sf::Color::White);

textY2.setString('0' + std::to\_string(b\_y));

textY2.setPosition(145+350, 223);

sf::RectangleShape close(sf::Vector2f(300.f, 50.f));

close.setFillColor(sf::Color::White);

close.setPosition(25 + 175, 325);

close.setTexture(&buttonCloseUnfocused);

sf::RectangleShape plusX1(sf::Vector2f(50.f, 50.f));

sf::RectangleShape minusX1(sf::Vector2f(50.f, 50.f));

plusX1.setPosition(225, 120);

minusX1.setPosition(75, 120);

plusX1.setTexture(&buttonPlusUnfocused);

minusX1.setTexture(&buttonMinusUnfocused);

sf::RectangleShape plusY1(sf::Vector2f(50.f, 50.f));

sf::RectangleShape minusY1(sf::Vector2f(50.f, 50.f));

plusY1.setPosition(225, 230);

minusY1.setPosition(75, 230);

plusY1.setTexture(&buttonPlusUnfocused);

minusY1.setTexture(&buttonMinusUnfocused);

sf::RectangleShape plusX2(sf::Vector2f(50.f, 50.f));

sf::RectangleShape minusX2(sf::Vector2f(50.f, 50.f));

plusX2.setPosition(225 + 350, 120);

minusX2.setPosition(75 + 350, 120);

plusX2.setTexture(&buttonPlusUnfocused);

minusX2.setTexture(&buttonMinusUnfocused);

sf::RectangleShape plusY2(sf::Vector2f(50.f, 50.f));

sf::RectangleShape minusY2(sf::Vector2f(50.f, 50.f));

plusY2.setPosition(225 + 350, 230);

minusY2.setPosition(75 + 350, 230);

plusY2.setTexture(&buttonPlusUnfocused);

minusY2.setTexture(&buttonMinusUnfocused);

sf::RectangleShape arrowUpDown1(sf::Vector2f(50.f, 50.f));

arrowUpDown1.setPosition(25, 120);

arrowUpDown1.setTexture(&buttonUpDown);

sf::RectangleShape arrowLeftRight1(sf::Vector2f(50.f, 50.f));

arrowLeftRight1.setPosition(25, 230);

arrowLeftRight1.setTexture(&buttonLeftRight);

sf::RectangleShape arrowUpDown2(sf::Vector2f(50.f, 50.f));

arrowUpDown2.setPosition(275, 120);

arrowUpDown2.setTexture(&buttonUpDown);

sf::RectangleShape arrowLeftRight2(sf::Vector2f(50.f, 50.f));

arrowLeftRight2.setPosition(275, 230);

arrowLeftRight2.setTexture(&buttonLeftRight);

sf::RectangleShape arrowUpDown3(sf::Vector2f(50.f, 50.f));

arrowUpDown3.setPosition(25 + 350, 120);

arrowUpDown3.setTexture(&buttonUpDown);

sf::RectangleShape arrowLeftRight3(sf::Vector2f(50.f, 50.f));

arrowLeftRight3.setPosition(25 + 350, 230);

arrowLeftRight3.setTexture(&buttonLeftRight);

sf::RectangleShape arrowUpDown4(sf::Vector2f(50.f, 50.f));

arrowUpDown4.setPosition(275 + 350, 120);

arrowUpDown4.setTexture(&buttonUpDown);

sf::RectangleShape arrowLeftRight4(sf::Vector2f(50.f, 50.f));

arrowLeftRight4.setPosition(275 + 350, 230);

arrowLeftRight4.setTexture(&buttonLeftRight);

windowChoose.setFramerateLimit(60);

bool pressed = false;

while (windowChoose.isOpen()) {

sf::Event event;

while (windowChoose.pollEvent(event))

{

if (event.type == sf::Event::Closed)

windowChoose.close();

}

if (sf::Mouse::isButtonPressed(sf::Mouse::Left)) {

if (pressed) continue;

else {

if (plusX1.getGlobalBounds().contains(windowChoose.mapPixelToCoords(sf::Mouse::getPosition(windowChoose)))) {

a\_x++;

}

if (minusX1.getGlobalBounds().contains(windowChoose.mapPixelToCoords(sf::Mouse::getPosition(windowChoose)))) {

a\_x--;

}

if (plusY1.getGlobalBounds().contains(windowChoose.mapPixelToCoords(sf::Mouse::getPosition(windowChoose)))) {

a\_y++;

b\_x = a\_y;

}

if (minusY1.getGlobalBounds().contains(windowChoose.mapPixelToCoords(sf::Mouse::getPosition(windowChoose)))) {

a\_y--;

b\_x = a\_y;

}

if (plusX2.getGlobalBounds().contains(windowChoose.mapPixelToCoords(sf::Mouse::getPosition(windowChoose)))) {

b\_x++;

a\_y = b\_x;

}

if (minusX2.getGlobalBounds().contains(windowChoose.mapPixelToCoords(sf::Mouse::getPosition(windowChoose)))) {

b\_x--;

a\_y = b\_x;

}

if (plusY2.getGlobalBounds().contains(windowChoose.mapPixelToCoords(sf::Mouse::getPosition(windowChoose)))) {

b\_y++;

}

if (minusY2.getGlobalBounds().contains(windowChoose.mapPixelToCoords(sf::Mouse::getPosition(windowChoose)))) {

b\_y--;

}

if (close.getGlobalBounds().contains(windowChoose.mapPixelToCoords(sf::Mouse::getPosition(windowChoose)))) {

windowChoose.close();

}

pressed = true;

}

continue;

}

pressed = false;

if (a\_x < 1) {

a\_x = 1;

}

if (a\_x > 99) {

a\_x = 99;

}

if (a\_x < 10) {

textX1.setString('0' + std::to\_string(a\_x));

}

else {

textX1.setString(std::to\_string(a\_x));

}

if (a\_y < 1) {

a\_y = 1;

b\_x = 1;

}

if (a\_y > 99) {

a\_y = 99;

b\_x = 99;

}

if (a\_y < 10) {

textY1.setString('0' + std::to\_string(a\_y));

textX2.setString('0' + std::to\_string(b\_x));

}

else {

textY1.setString(std::to\_string(a\_y));

textX2.setString(std::to\_string(b\_x));

}

if (b\_x < 1) {

a\_y = 1;

b\_x = 1;

}

if (b\_x > 99) {

a\_y = 99;

b\_x = 99;

}

if (b\_x < 10) {

textY1.setString('0' + std::to\_string(a\_y));

textX2.setString('0' + std::to\_string(b\_x));

}

else {

textY1.setString(std::to\_string(a\_y));

textX2.setString(std::to\_string(b\_x));

}

if (b\_y < 1) {

b\_y = 1;

}

if (b\_y > 99) {

b\_y = 99;

}

if (b\_y < 10) {

textY2.setString('0' + std::to\_string(b\_y));

}

else {

textY2.setString(std::to\_string(b\_y));

}

if (plusX1.getGlobalBounds().contains(windowChoose.mapPixelToCoords(sf::Mouse::getPosition(windowChoose)))) {

plusX1.setTexture(&buttonPlusFocused);

}

else {

plusX1.setTexture(&buttonPlusUnfocused);

}

if (plusY1.getGlobalBounds().contains(windowChoose.mapPixelToCoords(sf::Mouse::getPosition(windowChoose)))) {

plusY1.setTexture(&buttonPlusFocused);

}

else {

plusY1.setTexture(&buttonPlusUnfocused);

}

if (minusX1.getGlobalBounds().contains(windowChoose.mapPixelToCoords(sf::Mouse::getPosition(windowChoose)))) {

minusX1.setTexture(&buttonMinusFocused);

}

else {

minusX1.setTexture(&buttonMinusUnfocused);

}

if (minusY1.getGlobalBounds().contains(windowChoose.mapPixelToCoords(sf::Mouse::getPosition(windowChoose)))) {

minusY1.setTexture(&buttonMinusFocused);

}

else {

minusY1.setTexture(&buttonMinusUnfocused);

}

if (plusX2.getGlobalBounds().contains(windowChoose.mapPixelToCoords(sf::Mouse::getPosition(windowChoose)))) {

plusX2.setTexture(&buttonPlusFocused);

}

else {

plusX2.setTexture(&buttonPlusUnfocused);

}

if (plusY2.getGlobalBounds().contains(windowChoose.mapPixelToCoords(sf::Mouse::getPosition(windowChoose)))) {

plusY2.setTexture(&buttonPlusFocused);

}

else {

plusY2.setTexture(&buttonPlusUnfocused);

}

if (minusX2.getGlobalBounds().contains(windowChoose.mapPixelToCoords(sf::Mouse::getPosition(windowChoose)))) {

minusX2.setTexture(&buttonMinusFocused);

}

else {

minusX2.setTexture(&buttonMinusUnfocused);

}

if (minusY2.getGlobalBounds().contains(windowChoose.mapPixelToCoords(sf::Mouse::getPosition(windowChoose)))) {

minusY2.setTexture(&buttonMinusFocused);

}

else {

minusY2.setTexture(&buttonMinusUnfocused);

}

if (close.getGlobalBounds().contains(windowChoose.mapPixelToCoords(sf::Mouse::getPosition(windowChoose)))) {

close.setTexture(&buttonCloseFocused);

}

else {

close.setTexture(&buttonCloseUnfocused);

}

windowChoose.clear(sf::Color(32, 45, 58, 255));

windowChoose.draw(plusX1);

windowChoose.draw(minusX1);

windowChoose.draw(plusY1);

windowChoose.draw(minusY1);

windowChoose.draw(plusX2);

windowChoose.draw(minusX2);

windowChoose.draw(plusY2);

windowChoose.draw(minusY2);

windowChoose.draw(close);

windowChoose.draw(textX1);

windowChoose.draw(textY1);

windowChoose.draw(textX2);

windowChoose.draw(textY2);

windowChoose.draw(textNa1);

windowChoose.draw(textNa2);

windowChoose.draw(arrowUpDown1);

windowChoose.draw(arrowLeftRight1);

windowChoose.draw(arrowUpDown2);

windowChoose.draw(arrowLeftRight2);

windowChoose.draw(arrowUpDown3);

windowChoose.draw(arrowLeftRight3);

windowChoose.draw(arrowUpDown4);

windowChoose.draw(arrowLeftRight4);

windowChoose.draw(textChoose);

windowChoose.display();

x1 = a\_x;

y1 = a\_y;

x2 = b\_x;

y2 = b\_y;

}

}

void DimChoose5(int& x, int& y, int &angle) {

int winH = 400;

int winW = 520;

sf::RenderWindow windowChoose(sf::VideoMode(winW, winH), L"Виберіть розмір матриці", sf::Style::Titlebar);

windowChoose.setFramerateLimit(60);

int a\_x = 1;

int a\_y = 1;

int a\_angle = 1;

sf::Font fontNumbers;

sf::Font fontText;

fontNumbers.loadFromFile("Files//apercu.ttf");

fontText.loadFromFile("Files//moula.ttf");

sf::Text textX;

sf::Text textY;

sf::Text textNa;

sf::Text textChoose;

sf::Text txt90;

sf::Text txt180;

sf::Text txt270;

sf::Texture buttonPlusUnfocused;

sf::Texture buttonMinusUnfocused;

sf::Texture buttonPlusFocused;

sf::Texture buttonMinusFocused;

sf::Texture buttonCloseUnfocused;

sf::Texture buttonCloseFocused;

sf::Texture buttonUpDown;

sf::Texture buttonLeftRight;

sf::Texture tickbox\_a;

sf::Texture tickbox\_b;

buttonPlusUnfocused.loadFromFile("Files\\plus.png");

buttonMinusUnfocused.loadFromFile("Files\\minus.png");

buttonPlusFocused.loadFromFile("Files\\plus\_f.png");

buttonMinusFocused.loadFromFile("Files\\minus\_f.png");

buttonCloseUnfocused.loadFromFile("Files\\continue.png");

buttonCloseFocused.loadFromFile("Files\\continue\_f.png");

buttonUpDown.loadFromFile("Files\\arrow\_up.png");

buttonLeftRight.loadFromFile("Files\\arrow\_side.png");

tickbox\_a.loadFromFile("Files\\tickbox\_a.png");

tickbox\_b.loadFromFile("Files\\tickbox\_b.png");

textNa.setFont(fontText);

textNa.setCharacterSize(50);

textNa.setFillColor(sf::Color::White);

textNa.setString(L"X");

textNa.setPosition(163, 165);

textChoose.setFont(fontText);

textChoose.setCharacterSize(40);

textChoose.setFillColor(sf::Color::White);

textChoose.setString(L"Оберіть розмір\nматриці та кут:");

textChoose.setPosition(47 + 85, -5);

textX.setFont(fontNumbers);

textX.setCharacterSize(50);

textX.setFillColor(sf::Color::White);

textX.setString('0' + std::to\_string(a\_x));

textX.setPosition(145, 113);

textY.setFont(fontNumbers);

textY.setCharacterSize(50);

textY.setFillColor(sf::Color::White);

textY.setString('0' + std::to\_string(a\_y));

textY.setPosition(145, 223);

sf::RectangleShape close(sf::Vector2f(300.f, 50.f));

close.setFillColor(sf::Color::White);

close.setPosition(110, 325);

close.setTexture(&buttonCloseUnfocused);

sf::RectangleShape plusX(sf::Vector2f(50.f, 50.f));

sf::RectangleShape minusX(sf::Vector2f(50.f, 50.f));

plusX.setPosition(225, 120);

minusX.setPosition(75, 120);

plusX.setTexture(&buttonPlusUnfocused);

minusX.setTexture(&buttonMinusUnfocused);

sf::RectangleShape plusY(sf::Vector2f(50.f, 50.f));

sf::RectangleShape minusY(sf::Vector2f(50.f, 50.f));

plusY.setPosition(225, 230);

minusY.setPosition(75, 230);

plusY.setTexture(&buttonPlusUnfocused);

minusY.setTexture(&buttonMinusUnfocused);

sf::RectangleShape arrowUpDown(sf::Vector2f(50.f, 50.f));

arrowUpDown.setPosition(25, 120);

arrowUpDown.setTexture(&buttonUpDown);

sf::RectangleShape arrowLeftRight(sf::Vector2f(50.f, 50.f));

arrowLeftRight.setPosition(25, 230);

arrowLeftRight.setTexture(&buttonLeftRight);

sf::RectangleShape arrowUpDown1(sf::Vector2f(50.f, 50.f));

arrowUpDown1.setPosition(275, 120);

arrowUpDown1.setTexture(&buttonUpDown);

sf::RectangleShape arrowLeftRight1(sf::Vector2f(50.f, 50.f));

arrowLeftRight1.setPosition(275, 230);

arrowLeftRight1.setTexture(&buttonLeftRight);

sf::RectangleShape tickbox90(sf::Vector2f(50.f, 50.f));

sf::RectangleShape tickbox180(sf::Vector2f(50.f, 50.f));

sf::RectangleShape tickbox270(sf::Vector2f(50.f, 50.f));

tickbox90.setPosition(325 + 15, 120 - 10);

tickbox180.setPosition(325 + 15, 170);

tickbox270.setPosition(325 + 15, 220 + 10);

tickbox90.setTexture(&tickbox\_a);

tickbox180.setTexture(&tickbox\_a);

tickbox270.setTexture(&tickbox\_a);

txt90.setFont(fontNumbers);

txt90.setCharacterSize(45);

txt90.setFillColor(sf::Color::White);

txt90.setString("90°");

txt90.setPosition(325 + 70, 120 - 13);

txt180.setFont(fontNumbers);

txt180.setCharacterSize(45);

txt180.setFillColor(sf::Color::White);

txt180.setString("180°");

txt180.setPosition(325 + 70, 170);

txt270.setFont(fontNumbers);

txt270.setCharacterSize(45);

txt270.setFillColor(sf::Color::White);

txt270.setString("270°");

txt270.setPosition(325 + 70, 220 + 10);

windowChoose.setFramerateLimit(60);

bool pressed = false;

while (windowChoose.isOpen()) {

sf::Event event;

while (windowChoose.pollEvent(event))

{

if (event.type == sf::Event::Closed)

windowChoose.close();

}

if (sf::Mouse::isButtonPressed(sf::Mouse::Left)) {

if (pressed) continue;

else {

if (plusX.getGlobalBounds().contains(windowChoose.mapPixelToCoords(sf::Mouse::getPosition(windowChoose)))) {

a\_x++;

}

if (minusX.getGlobalBounds().contains(windowChoose.mapPixelToCoords(sf::Mouse::getPosition(windowChoose)))) {

a\_x--;

}

if (plusY.getGlobalBounds().contains(windowChoose.mapPixelToCoords(sf::Mouse::getPosition(windowChoose)))) {

a\_y++;

}

if (minusY.getGlobalBounds().contains(windowChoose.mapPixelToCoords(sf::Mouse::getPosition(windowChoose)))) {

a\_y--;

}

if (tickbox90.getGlobalBounds().contains(windowChoose.mapPixelToCoords(sf::Mouse::getPosition(windowChoose)))) {

a\_angle = 1;

}

if (tickbox180.getGlobalBounds().contains(windowChoose.mapPixelToCoords(sf::Mouse::getPosition(windowChoose)))) {

a\_angle = 2;

}

if (tickbox270.getGlobalBounds().contains(windowChoose.mapPixelToCoords(sf::Mouse::getPosition(windowChoose)))) {

a\_angle = 3;

}

if (close.getGlobalBounds().contains(windowChoose.mapPixelToCoords(sf::Mouse::getPosition(windowChoose)))) {

windowChoose.close();

}

pressed = true;

}

continue;

}

pressed = false;

if (a\_x < 1) {

a\_x = 1;

}

if (a\_x > 99) {

a\_x = 99;

}

if (a\_x < 10) {

textX.setString('0' + std::to\_string(a\_x));

}

else {

textX.setString(std::to\_string(a\_x));

}

if (a\_y < 1) {

a\_y = 1;

}

if (a\_y > 99) {

a\_y = 99;

}

if (a\_y < 10) {

textY.setString('0' + std::to\_string(a\_y));

}

else {

textY.setString(std::to\_string(a\_y));

}

if (plusX.getGlobalBounds().contains(windowChoose.mapPixelToCoords(sf::Mouse::getPosition(windowChoose)))) {

plusX.setTexture(&buttonPlusFocused);

}

else {

plusX.setTexture(&buttonPlusUnfocused);

}

if (plusY.getGlobalBounds().contains(windowChoose.mapPixelToCoords(sf::Mouse::getPosition(windowChoose)))) {

plusY.setTexture(&buttonPlusFocused);

}

else {

plusY.setTexture(&buttonPlusUnfocused);

}

if (minusX.getGlobalBounds().contains(windowChoose.mapPixelToCoords(sf::Mouse::getPosition(windowChoose)))) {

minusX.setTexture(&buttonMinusFocused);

}

else {

minusX.setTexture(&buttonMinusUnfocused);

}

if (minusY.getGlobalBounds().contains(windowChoose.mapPixelToCoords(sf::Mouse::getPosition(windowChoose)))) {

minusY.setTexture(&buttonMinusFocused);

}

else {

minusY.setTexture(&buttonMinusUnfocused);

}

if (close.getGlobalBounds().contains(windowChoose.mapPixelToCoords(sf::Mouse::getPosition(windowChoose)))) {

close.setTexture(&buttonCloseFocused);

}

else {

close.setTexture(&buttonCloseUnfocused);

}

if (a\_angle == 1) {

tickbox90.setTexture(&tickbox\_b);

tickbox180.setTexture(&tickbox\_a);

tickbox270.setTexture(&tickbox\_a);

}

if (a\_angle == 2) {

tickbox90.setTexture(&tickbox\_a);

tickbox180.setTexture(&tickbox\_b);

tickbox270.setTexture(&tickbox\_a);

}

if (a\_angle == 3) {

tickbox90.setTexture(&tickbox\_a);

tickbox180.setTexture(&tickbox\_a);

tickbox270.setTexture(&tickbox\_b);

}

windowChoose.clear(sf::Color(32, 45, 58, 255));

windowChoose.draw(plusX);

windowChoose.draw(minusX);

windowChoose.draw(plusY);

windowChoose.draw(minusY);

windowChoose.draw(close);

windowChoose.draw(textX);

windowChoose.draw(textY);

windowChoose.draw(textNa);

windowChoose.draw(arrowUpDown);

windowChoose.draw(arrowLeftRight);

windowChoose.draw(arrowUpDown1);

windowChoose.draw(arrowLeftRight1);

windowChoose.draw(textChoose);

windowChoose.draw(tickbox90);

windowChoose.draw(tickbox180);

windowChoose.draw(tickbox270);

windowChoose.draw(txt90);

windowChoose.draw(txt180);

windowChoose.draw(txt270);

windowChoose.display();

x = a\_x;

y = a\_y;

angle = a\_angle;

}

}

/\*Функція вибору операцій

Вона відповідає за введення та виведення даних на екран, створення допоміжних матриць та передання їх в відповідні функції.

Також, вона перевіряє, чи введені дані є допустимими, та повідомляє про помилки (ділення на 0, тд)

\*/

void OperationChooser(int opr) {

int a\_y = 0, a\_x = 0, b\_y = 0, b\_x = 0, toContinue = 0;

if (opr == 1)

{

std::wcout << L"=== ДОДАВАННЯ ===" << std::endl;

DimChoose1(a\_x, a\_y);

b\_x = a\_x;

b\_y = a\_y;

double\* a = new double[a\_x \* a\_y];

double\* b = new double[b\_x \* b\_y];

double\* c = new double[b\_x \* b\_y];

std::wcout << L"Введіть першу матрицю: " << std::endl;

InputMatrix(a\_x, a\_y, a);

std::wcout << L"Введіть другу матрицю: " << std::endl;

InputMatrix(b\_x, b\_y, b);

AddMatrices(a\_x, a\_y, a, b, c);

OutputMatrix(a\_x, a\_y, c);

}

else if (opr == 2)

{

std::wcout << L"=== ВІДНІМАННЯ ===" << std::endl;

DimChoose1(a\_x, a\_y);

b\_x = a\_x;

b\_y = a\_y;

double\* a = new double[a\_x \* a\_y];

double\* b = new double[b\_x \* b\_y];

double\* c = new double[b\_x \* b\_y];

std::wcout << L"Введіть першу матрицю: " << std::endl;

InputMatrix(a\_x, a\_y, a);

std::wcout << L"Введіть другу матрицю: " << std::endl;

InputMatrix(b\_x, b\_y, b);

SubtractMatricies(a\_x, a\_y, a, b, c);

OutputMatrix(a\_x, a\_y, c);

}

else if (opr == 3)

{

std::wcout << L"=== СКАЛЯРНЕ МНОЖЕННЯ ===" << std::endl;

DimChoose2(a\_x, a\_y);

double\* a = new double[a\_x \* a\_y];

double mult;

std::wcout << L"Введіть матрицю: " << std::endl;

InputMatrix(a\_x, a\_y, a);

std::wcout << L"Введіть множник: " << std::endl;

std::cin >> mult;

ScalarMultiplyMatrix(a\_x, a\_y, a, mult);

OutputMatrix(a\_x, a\_y, a);

}

else if (opr == 4)

{

std::wcout << L"=== СКАЛЯРНЕ ДІЛЕННЯ ===" << std::endl;

DimChoose2(a\_x, a\_y);

double\* a = new double[a\_x \* a\_y];

double div;

std::wcout << L"Введіть матрицю: " << std::endl;

InputMatrix(a\_x, a\_y, a);

std::wcout << L"Введіть дільник: " << std::endl;

std::cin >> div;

while (div == 0) {

std::wcout << L"Помилка: Дільник не може бути нулем! Спробуйте знову: " << std::endl;

std::cin >> div;

}

DivideMatrix(a\_x, a\_y, a, div);

OutputMatrix(a\_x, a\_y, a);

}

else if (opr == 5)

{

std::wcout << L"=== ТРАНСПОНУВАННЯ ===" << std::endl;

DimChoose2(a\_x, a\_y);

double\* a = new double[a\_x \* a\_y];

std::wcout << L"Введіть матрицю: " << std::endl;

InputMatrix(a\_x, a\_y, a);

TransposeMatrix(a\_x, a\_y, a);

OutputMatrix(a\_x, a\_y, a);

}

else if (opr == 6) {

std::wcout << L"=== МНОЖЕННЯ ===" << std::endl;

DimChoose4(a\_x, a\_y, b\_x, b\_y);

double\* a = new double[a\_x \* a\_y];

double\* b = new double[b\_x \* b\_y];

std::wcout << L"Введіть першу матрицю: " << std::endl;

InputMatrix(a\_x, a\_y, a);

std::wcout << L"Введіть другу матрицю: " << std::endl;

InputMatrix(b\_x, b\_y, b);

int c\_x = a\_x;

int c\_y = b\_y;

double\* c = new double[c\_x \* c\_y];

if (a\_x == a\_y && b\_x == b\_y && a\_x == b\_x && IsPowerOfTwo(a\_x)) { //якщо матриця квадратна і розмір = степінь 2ки, алгоритм Штрассена

StrassenMultiply(a\_x, a\_y, b\_x, b\_y, c\_x, c\_y, a, b, c);

OutputMatrix(c\_x, c\_y, c);

}

else { //Інакше, збільшити матриці

/\*StandardMultiply(a\_x, a\_y, b\_x, b\_y, c\_x, c\_y, a, b, c);

OutputMatrix(c\_x, c\_y, c);\*/

int new\_x, new\_y, temp\_a;

temp\_a = std::max(a\_x, a\_y);

temp\_a = std::max(temp\_a, b\_x);

temp\_a = std::max(temp\_a, b\_y);

new\_x = NextPowerOfTwo(temp\_a);

new\_y = new\_x;

double\* new\_a = new double[new\_x \* new\_y];

double\* new\_b = new double[new\_x \* new\_y];

double\* new\_c = new double[new\_x \* new\_y];

PadUp(a\_x, a\_y, a, new\_x, new\_y, new\_a);

PadUp(b\_x, b\_y, b, new\_x, new\_y, new\_b);

StrassenMultiply(new\_x, new\_y, new\_x, new\_y, new\_x, new\_y, new\_a, new\_b, new\_c);

PadDown(c\_x, c\_y, c, new\_x, new\_y, new\_c);

OutputMatrix(c\_x, c\_y, c);

}

}

else if (opr == 7) {

std::wcout << L"=== ВИЗНАЧНИК ===" << std::endl;

DimChoose3(a\_x);

double\* a = new double[a\_x \* a\_x];

std::wcout << L"Введіть матрицю: " << std::endl;

InputMatrix(a\_x, a\_x, a);

int determinant = GetDeterminant(a, a\_x);

std::wcout << L"Визначник: " << determinant << std::endl;

}

else if (opr == 8) {

std::wcout << L"=== ОБЕРНЕНА МАТРИЦЯ ===" << std::endl;

DimChoose3(a\_x);

double\* a = new double[a\_x \* a\_x];

double\* inv\_a = new double[a\_x \* a\_x];

std::wcout << L"Введіть матрицю: " << std::endl;

InputMatrix(a\_x, a\_x, a);

while (GetDeterminant(a, a\_x) == 0) {

std::wcout << L"Помилка: Визначник матриці не може бути нулем. Спробуйте знову." << std::endl;

std::wcout << L"Введіть матрицю: " << std::endl;

InputMatrix(a\_x, a\_x, a);

}

InverseMatrix(a, inv\_a, a\_x);

OutputMatrix(a\_x, a\_x, inv\_a);

}

else if (opr == 9) {

std::wcout << L"=== ОБЕРТАННЯ ===" << std::endl;

int choice;

DimChoose5(a\_x, a\_y, choice);

double\* a = new double[a\_x \* a\_y];

std::wcout << L"Введіть матрицю:" << std::endl;

InputMatrix(a\_x, a\_y, a);

if (choice == 1 || choice == 3) {

double\* b = new double[a\_y \* a\_x];

RotateMatrix(a\_x, a\_y, a\_y, a\_x, a, b, choice);

OutputMatrix(a\_y, a\_x, b);

}

else if (choice == 2) {

double\* b = new double[a\_x \* a\_y];

RotateMatrix(a\_x, a\_y, a\_x, a\_y, a, b, choice);

OutputMatrix(a\_x, a\_y, b);

}

}

std::wcout << L"=== ОБЕРІТЬ НАСТУПНУ ОПЕРАЦІЮ ===" << std::endl;

}

int main()

{

//Встановити українську мову

setlocale(LC\_ALL, "Ukrainian");

SetConsoleOutputCP(1251);

SetConsoleCP(1251);

int winH = 500;

int winW = 800;

sf::RenderWindow window(sf::VideoMode(winW, winH), L"Калькулятор матриць", sf::Style::Titlebar); //Створення вікна

sf::Image icon;

icon.loadFromFile("Files\\ico.png");

window.setFramerateLimit(60); //Для запобігання перевантаження відеокарти

window.setIcon(icon.getSize().x, icon.getSize().y, icon.getPixelsPtr()); //Задання іконки вікна

//Ініціалізація кнопок вікна

sf::RectangleShape buttonAddition(sf::Vector2f(400.f, 100.f));

sf::RectangleShape buttonSubtraction(sf::Vector2f(400.f, 100.f));

sf::RectangleShape buttonMultiplication(sf::Vector2f(400.f, 100.f));

sf::RectangleShape buttonDivision(sf::Vector2f(400.f, 100.f));

sf::RectangleShape buttonTransposition(sf::Vector2f(400.f, 100.f));

sf::RectangleShape buttonMatrixMultiplication(sf::Vector2f(400.f, 100.f));

sf::RectangleShape buttonDeterminant(sf::Vector2f(400.f, 100.f));

sf::RectangleShape buttonInversion(sf::Vector2f(400.f, 100.f));

sf::RectangleShape buttonRotation(sf::Vector2f(400.f, 100.f));

sf::RectangleShape buttonExit(sf::Vector2f(400.f, 100.f));

//Ініціалізація текстур кнопок

sf::Texture buttonAdditionTexture;

sf::Texture buttonSubtractionTexture;

sf::Texture buttonMultiplicationTexture;

sf::Texture buttonDivisionTexture;

sf::Texture buttonTranspositionTexture;

sf::Texture buttonMatrixMultiplicationTexture;

sf::Texture buttonDeterminantTexture;

sf::Texture buttonInversionTexture;

sf::Texture buttonRotationTexture;

sf::Texture buttonExitTexture;

//Ініціалізація альтернативних текстур кнопок

sf::Texture buttonAdditionTexture1;

sf::Texture buttonSubtractionTexture1;

sf::Texture buttonMultiplicationTexture1;

sf::Texture buttonDivisionTexture1;

sf::Texture buttonTranspositionTexture1;

sf::Texture buttonMatrixMultiplicationTexture1;

sf::Texture buttonDeterminantTexture1;

sf::Texture buttonInversionTexture1;

sf::Texture buttonRotationTexture1;

sf::Texture buttonExitTexture1;

//Відкриття текстур кнопок

buttonAdditionTexture.loadFromFile("Files\\addition.png");

buttonSubtractionTexture.loadFromFile("Files\\subtraction.png");

buttonMultiplicationTexture.loadFromFile("Files\\multiplication.png");

buttonDivisionTexture.loadFromFile("Files\\division.png");

buttonTranspositionTexture.loadFromFile("Files\\transposition.png");

buttonMatrixMultiplicationTexture.loadFromFile("Files\\matrix\_multiplication.png");

buttonDeterminantTexture.loadFromFile("Files\\determinant.png");

buttonInversionTexture.loadFromFile("Files\\inversion.png");

buttonRotationTexture.loadFromFile("Files\\rotation.png");

buttonExitTexture.loadFromFile("Files\\exit.png");

//Відкриття альтернативних текстур кнопок

buttonAdditionTexture1.loadFromFile("Files\\addition\_s.png");

buttonSubtractionTexture1.loadFromFile("Files\\subtraction\_s.png");

buttonMultiplicationTexture1.loadFromFile("Files\\multiplication\_s.png");

buttonDivisionTexture1.loadFromFile("Files\\division\_s.png");

buttonTranspositionTexture1.loadFromFile("Files\\transposition\_s.png");

buttonMatrixMultiplicationTexture1.loadFromFile("Files\\matrix\_multiplication\_s.png");

buttonDeterminantTexture1.loadFromFile("Files\\determinant\_s.png");

buttonInversionTexture1.loadFromFile("Files\\inversion\_s.png");

buttonRotationTexture1.loadFromFile("Files\\rotation\_s.png");

buttonExitTexture1.loadFromFile("Files\\exit\_s.png");

//Призначення початкових текстур до кнопок

buttonAddition.setTexture(&buttonAdditionTexture);

buttonSubtraction.setTexture(&buttonSubtractionTexture);

buttonMultiplication.setTexture(&buttonMultiplicationTexture);

buttonDivision.setTexture(&buttonDivisionTexture);

buttonTransposition.setTexture(&buttonTranspositionTexture);

buttonMatrixMultiplication.setTexture(&buttonMatrixMultiplicationTexture);

buttonDeterminant.setTexture(&buttonDeterminantTexture);

buttonInversion.setTexture(&buttonInversionTexture);

buttonRotation.setTexture(&buttonRotationTexture);

buttonExit.setTexture(&buttonExitTexture);

buttonAddition.setTextureRect(sf::IntRect(0, 0, 400, 100));

buttonSubtraction.setTextureRect(sf::IntRect(0, 0, 400, 100));

buttonMultiplication.setTextureRect(sf::IntRect(0, 0, 400, 100));

buttonDivision.setTextureRect(sf::IntRect(0, 0, 400, 100));

buttonTransposition.setTextureRect(sf::IntRect(0, 0, 400, 100));

buttonMatrixMultiplication.setTextureRect(sf::IntRect(0, 0, 400, 100));

buttonDeterminant.setTextureRect(sf::IntRect(0, 0, 400, 100));

buttonInversion.setTextureRect(sf::IntRect(0, 0, 400, 100));

buttonRotation.setTextureRect(sf::IntRect(0, 0, 400, 100));

buttonExit.setTextureRect(sf::IntRect(0, 0, 400, 100));

//Призначення позицій кнопок

buttonAddition.setPosition(0, 0);

buttonSubtraction.setPosition(400, 0);

buttonMultiplication.setPosition(0, 100);

buttonDivision.setPosition(400, 100);

buttonTransposition.setPosition(0, 200);

buttonMatrixMultiplication.setPosition(400, 200);

buttonDeterminant.setPosition(0, 300);

buttonInversion.setPosition(400, 300);

buttonRotation.setPosition(0, 400);

buttonExit.setPosition(400, 400);

bool pressed = false;

while (window.isOpen())

{

sf::Event event;

while (window.pollEvent(event))

{

if (event.type == sf::Event::Closed)

window.close();

}

//Перевірка розташування курсора та натиснення на вибрану кнопку

if (sf::Mouse::isButtonPressed(sf::Mouse::Left))

{

if (pressed) continue;

else {

if (buttonAddition.getGlobalBounds().contains(window.mapPixelToCoords(sf::Mouse::getPosition(window))) && window.hasFocus()) {

OperationChooser(1);

}

if (buttonSubtraction.getGlobalBounds().contains(window.mapPixelToCoords(sf::Mouse::getPosition(window))) && window.hasFocus()) {

OperationChooser(2);

}

if (buttonMultiplication.getGlobalBounds().contains(window.mapPixelToCoords(sf::Mouse::getPosition(window))) && window.hasFocus()) {

OperationChooser(3);

}

if (buttonDivision.getGlobalBounds().contains(window.mapPixelToCoords(sf::Mouse::getPosition(window))) && window.hasFocus()) {

OperationChooser(4);

}

if (buttonTransposition.getGlobalBounds().contains(window.mapPixelToCoords(sf::Mouse::getPosition(window))) && window.hasFocus()) {

OperationChooser(5);

}

if (buttonMatrixMultiplication.getGlobalBounds().contains(window.mapPixelToCoords(sf::Mouse::getPosition(window))) && window.hasFocus()) {

OperationChooser(6);

}

if (buttonDeterminant.getGlobalBounds().contains(window.mapPixelToCoords(sf::Mouse::getPosition(window))) && window.hasFocus()) {

OperationChooser(7);

}

if (buttonInversion.getGlobalBounds().contains(window.mapPixelToCoords(sf::Mouse::getPosition(window))) && window.hasFocus()) {

OperationChooser(8);

}

if (buttonRotation.getGlobalBounds().contains(window.mapPixelToCoords(sf::Mouse::getPosition(window))) && window.hasFocus()) {

OperationChooser(9);

}

if (buttonExit.getGlobalBounds().contains(window.mapPixelToCoords(sf::Mouse::getPosition(window))) && window.hasFocus()) {

window.close();

}

pressed = true;

}

continue;

}

pressed = false;

//ПОЧАТОК Перевірка розташування курсора та замінення на підсвічену текстуру

if (buttonAddition.getGlobalBounds().contains(window.mapPixelToCoords(sf::Mouse::getPosition(window))) && window.hasFocus()) {

buttonAddition.setTexture(&buttonAdditionTexture1);

}

else {

buttonAddition.setTexture(&buttonAdditionTexture);

}

if (buttonSubtraction.getGlobalBounds().contains(window.mapPixelToCoords(sf::Mouse::getPosition(window))) && window.hasFocus()) {

buttonSubtraction.setTexture(&buttonSubtractionTexture1);

}

else {

buttonSubtraction.setTexture(&buttonSubtractionTexture);

}

if (buttonMultiplication.getGlobalBounds().contains(window.mapPixelToCoords(sf::Mouse::getPosition(window))) && window.hasFocus()) {

buttonMultiplication.setTexture(&buttonMultiplicationTexture1);

}

else {

buttonMultiplication.setTexture(&buttonMultiplicationTexture);

}

if (buttonDivision.getGlobalBounds().contains(window.mapPixelToCoords(sf::Mouse::getPosition(window))) && window.hasFocus()) {

buttonDivision.setTexture(&buttonDivisionTexture1);

}

else {

buttonDivision.setTexture(&buttonDivisionTexture);

}

if (buttonTransposition.getGlobalBounds().contains(window.mapPixelToCoords(sf::Mouse::getPosition(window))) && window.hasFocus()) {

buttonTransposition.setTexture(&buttonTranspositionTexture1);

}

else {

buttonTransposition.setTexture(&buttonTranspositionTexture);

}

if (buttonMatrixMultiplication.getGlobalBounds().contains(window.mapPixelToCoords(sf::Mouse::getPosition(window))) && window.hasFocus()) {

buttonMatrixMultiplication.setTexture(&buttonMatrixMultiplicationTexture1);

}

else {

buttonMatrixMultiplication.setTexture(&buttonMatrixMultiplicationTexture);

}

if (buttonDeterminant.getGlobalBounds().contains(window.mapPixelToCoords(sf::Mouse::getPosition(window))) && window.hasFocus()) {

buttonDeterminant.setTexture(&buttonDeterminantTexture1);

}

else {

buttonDeterminant.setTexture(&buttonDeterminantTexture);

}

if (buttonInversion.getGlobalBounds().contains(window.mapPixelToCoords(sf::Mouse::getPosition(window))) && window.hasFocus()) {

buttonInversion.setTexture(&buttonInversionTexture1);

}

else {

buttonInversion.setTexture(&buttonInversionTexture);

}

if (buttonRotation.getGlobalBounds().contains(window.mapPixelToCoords(sf::Mouse::getPosition(window))) && window.hasFocus()) {

buttonRotation.setTexture(&buttonRotationTexture1);

}

else {

buttonRotation.setTexture(&buttonRotationTexture);

}

if (buttonExit.getGlobalBounds().contains(window.mapPixelToCoords(sf::Mouse::getPosition(window))) && window.hasFocus()) {

buttonExit.setTexture(&buttonExitTexture1);

}

else {

buttonExit.setTexture(&buttonExitTexture);

}

//КІНЕЦЬ

window.clear(sf::Color(32, 45, 58, 255));

window.draw(buttonAddition);

window.draw(buttonSubtraction);

window.draw(buttonMultiplication);

window.draw(buttonDivision);

window.draw(buttonTransposition);

window.draw(buttonMatrixMultiplication);

window.draw(buttonDeterminant);

window.draw(buttonInversion);

window.draw(buttonRotation);

window.draw(buttonExit);

window.display();

}

return 0;

}

void InputMatrix(int x, int y, double\* m) {

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

for (int j = 0; j < y; j++) {

std::cin >> m[i \* y + j];

}

}

}

void OutputMatrix(int x, int y, double\* m) {

char ch;

std::wcout << L"Результат:" << std::endl;

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

for (int j = 0; j < y; j++) {

printf("%8.2f", m[i \* y + j]);

}

std::cout << std::endl;

}

std::wcout << L"Зберегти результат до файлу? (any/n)" << std::endl;

std::cin >> ch;

if (ch != 'n') {

long long int t = std::time(nullptr);

tm tm = \*std::localtime(&t);

std::stringstream textStream;

textStream << std::put\_time(&tm, "%Y-%m-%d-%H-%M-%S");

std::string tempStr = textStream.str() + "\_output.txt";

char st[] = "";

strcpy(st, tempStr.c\_str());

FILE\* fp;

fp = fopen(st, "w");

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

for (int j = 0; j < y; j++) {

fprintf(fp, "%8.2f", m[i \* y + j]);

}

fprintf(fp, "\n");

}

fclose(fp);

std::wcout << L"Успішно збережено до " << st << std::endl;

}

}

void ZeroMatrix(int x, int y, double\* m) {

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

for (int j = 0; j < y; j++) {

m[i \* y + j] = 0;

}

}

}

void PadUp(int x, int y, double\* m, int new\_x, int new\_y, double\* new\_m) {

for (int i = 0; i < new\_x; i++) {

for (int j = 0; j < new\_y; j++) {

if (i < x && j < y) {

new\_m[i \* new\_y + j] = m[i \* y + j];

}

else {

new\_m[i \* new\_y + j] = 0;

}

}

}

}

void PadDown(int x, int y, double\* m, int new\_x, int new\_y, double\* new\_m) {

for (int i = 0; i < new\_x; i++) {

for (int j = 0; j < new\_y; j++) {

if (i < x && j < y) {

m[i \* y + j] = new\_m[i \* new\_y + j];

}

}

}

}

int NextPowerOfTwo(int a) {

int pw = 0;

int i = 0;

while (pw < a) {

pw = pow(2, i);

i++;

}

return pw;

}

bool IsPowerOfTwo(int n) {

return (n > 0 && ((n & (n - 1)) == 0));

}

void TransposeMatrix(int& x, int& y, double\* m) {

if (x == y) { //Trivial for a square matrix

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

for (int j = 0; j < y; j++) {

if (i > j) {

std::swap(m[i \* y + j], m[j \* y + i]);

}

}

}

}

else {

int new\_size = std::max(x, y);

double\* new\_m = new double[new\_size \* new\_size];

ZeroMatrix(new\_size, new\_size, new\_m);

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

for (int j = 0; j < y; j++) {

new\_m[i \* new\_size + j] = m[i \* y + j];

}

}

for (int i = 0; i < new\_size; i++) {

for (int j = 0; j < new\_size; j++) {

if (i > j) {

std::swap(new\_m[i \* new\_size + j], new\_m[j \* new\_size + i]);

}

}

}

std::swap(x, y);

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

for (int j = 0; j < y; j++) {

m[i \* y + j] = new\_m[i \* new\_size + j];

}

}

}

}

void AddMatrices(int x, int y, double\* a, double\* b, double\* c) {

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

for (int j = 0; j < y; j++) {

c[i \* y + j] = a[i \* y + j] + b[i \* y + j];

}

}

}

void SubtractMatricies(int x, int y, double\* a, double\* b, double\* c) {

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

for (int j = 0; j < y; j++) {

c[i \* y + j] = a[i \* y + j] - b[i \* y + j];

}

}

}

void ScalarMultiplyMatrix(int x, int y, double\* a, double mult) {

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

for (int j = 0; j < y; j++) {

a[i \* y + j] \*= mult;

}

}

}

void DivideMatrix(int x, int y, double\* a, double div) {

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

for (int j = 0; j < y; j++) {

a[i \* y + j] /= div;

}

}

}

void StandardMultiply(int a\_x, int a\_y, int b\_x, int b\_y, int c\_x, int c\_y, double\* a, double\* b, double\* c) {

int n = a\_y;

double c\_ij;

for (int i = 0; i < c\_x; i++) {

for (int j = 0; j < c\_y; j++) {

c\_ij = 0;

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

c\_ij += a[i \* a\_y + k] \* b[k \* b\_y + j];

}

c[i \* c\_y + j] = c\_ij;

}

}

}

void StrassenMultiply(int a\_x, int a\_y, int b\_x, int b\_y, int c\_x, int c\_y, double\* a, double\* b, double\* c) {

int block = a\_x / 2;

double\* a11 = new double[block \* block];

double\* a12 = new double[block \* block];

double\* a21 = new double[block \* block];

double\* a22 = new double[block \* block];

double\* b11 = new double[block \* block];

double\* b12 = new double[block \* block];

double\* b21 = new double[block \* block];

double\* b22 = new double[block \* block];

double\* c11 = new double[block \* block];

double\* c12 = new double[block \* block];

double\* c21 = new double[block \* block];

double\* c22 = new double[block \* block];

double\* m1 = new double[block \* block];

double\* m2 = new double[block \* block];

double\* m3 = new double[block \* block];

double\* m4 = new double[block \* block];

double\* m5 = new double[block \* block];

double\* m6 = new double[block \* block];

double\* m7 = new double[block \* block];

double\* temp1 = new double[block \* block];

double\* temp2 = new double[block \* block];

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

for (int j = 0; j < block; j++) {

//a[i][j] is then rewritten as

//a[i\*sizeY+j]

a11[i \* block + j] = a[i \* a\_y + j];

}

}

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

for (int j = block; j < block \* 2; j++) {

a12[i \* block + (j - block)] = a[i \* a\_y + j];

}

}

for (int i = block; i < block \* 2; i++) {

for (int j = 0; j < block; j++) {

a21[(i - block) \* block + j] = a[i \* a\_y + j];

}

}

for (int i = block; i < block \* 2; i++) {

for (int j = block; j < block \* 2; j++) {

a22[(i - block) \* block + (j - block)] = a[i \* a\_y + j];

}

}

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

for (int j = 0; j < block; j++) {

//a[i][j] is then rewritten as

//a[i\*sizeY+j]

b11[i \* block + j] = b[i \* b\_y + j];

}

}

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

for (int j = block; j < block \* 2; j++) {

b12[i \* block + (j - block)] = b[i \* b\_y + j];

}

}

for (int i = block; i < block \* 2; i++) {

for (int j = 0; j < block; j++) {

b21[(i - block) \* block + j] = b[i \* b\_y + j];

}

}

for (int i = block; i < block \* 2; i++) {

for (int j = block; j < block \* 2; j++) {

b22[(i - block) \* block + (j - block)] = b[i \* b\_y + j];

}

}

if (block == 1) {

//M1

AddMatrices(block, block, a11, a22, temp1);

AddMatrices(block, block, b11, b22, temp2);

StandardMultiply(block, block, block, block, block, block, temp1, temp2, m1);

//M2

AddMatrices(block, block, a21, a22, temp1);

StandardMultiply(block, block, block, block, block, block, temp1, b11, m2);

//M3

SubtractMatricies(block, block, b12, b22, temp1);

StandardMultiply(block, block, block, block, block, block, a11, temp1, m3);

//M4

SubtractMatricies(block, block, b21, b11, temp1);

StandardMultiply(block, block, block, block, block, block, a22, temp1, m4);

//M5

AddMatrices(block, block, a11, a12, temp1);

StandardMultiply(block, block, block, block, block, block, temp1, b22, m5);

//M6

SubtractMatricies(block, block, a21, a11, temp1);

AddMatrices(block, block, b11, b12, temp2);

StandardMultiply(block, block, block, block, block, block, temp1, temp2, m6);

//M7

SubtractMatricies(block, block, a12, a22, temp1);

AddMatrices(block, block, b21, b22, temp2);

StandardMultiply(block, block, block, block, block, block, temp1, temp2, m7);

}

else {

//M1

AddMatrices(block, block, a11, a22, temp1);

AddMatrices(block, block, b11, b22, temp2);

StrassenMultiply(block, block, block, block, block, block, temp1, temp2, m1);

//M2

AddMatrices(block, block, a21, a22, temp1);

StrassenMultiply(block, block, block, block, block, block, temp1, b11, m2);

//M3

SubtractMatricies(block, block, b12, b22, temp1);

StrassenMultiply(block, block, block, block, block, block, a11, temp1, m3);

//M4

SubtractMatricies(block, block, b21, b11, temp1);

StrassenMultiply(block, block, block, block, block, block, a22, temp1, m4);

//M5

AddMatrices(block, block, a11, a12, temp1);

StrassenMultiply(block, block, block, block, block, block, temp1, b22, m5);

//M6

SubtractMatricies(block, block, a21, a11, temp1);

AddMatrices(block, block, b11, b12, temp2);

StrassenMultiply(block, block, block, block, block, block, temp1, temp2, m6);

//M7

SubtractMatricies(block, block, a12, a22, temp1);

AddMatrices(block, block, b21, b22, temp2);

StrassenMultiply(block, block, block, block, block, block, temp1, temp2, m7);

}

//C11

AddMatrices(block, block, m1, m4, temp1);

SubtractMatricies(block, block, temp1, m5, temp2);

AddMatrices(block, block, temp2, m7, c11);

//C12

AddMatrices(block, block, m3, m5, c12);

//C21

AddMatrices(block, block, m2, m4, c21);

//C22

SubtractMatricies(block, block, m1, m2, temp1);

AddMatrices(block, block, m3, m6, temp2);

AddMatrices(block, block, temp1, temp2, c22);

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

for (int j = 0; j < block; j++) {

//a[i][j] is then rewritten as

//a[i\*sizeY+j]

c[i \* a\_y + j] = c11[i \* block + j];

}

}

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

for (int j = block; j < block \* 2; j++) {

c[i \* a\_y + j] = c12[i \* block + (j - block)];

}

}

for (int i = block; i < block \* 2; i++) {

for (int j = 0; j < block; j++) {

c[i \* a\_y + j] = c21[(i - block) \* block + j];

}

}

for (int i = block; i < block \* 2; i++) {

for (int j = block; j < block \* 2; j++) {

c[i \* a\_y + j] = c22[(i - block) \* block + (j - block)];

}

}

delete[] temp1;

delete[] temp2;

delete[] c11;

delete[] c12;

delete[] c21;

delete[] c22;

delete[] a11;

delete[] a12;

delete[] a21;

delete[] a22;

delete[] b11;

delete[] b12;

delete[] b21;

delete[] b22;

delete[] m1;

delete[] m2;

delete[] m3;

delete[] m4;

delete[] m5;

delete[] m6;

delete[] m7;

}

void GetSubMatrix(double\* mat, double\* sub\_mat , int curr\_row, int curr\_col, int mat\_size, int sub\_size) {

int i = 0, j = 0;

for (int row = 0; row < mat\_size; row++) {

for (int col = 0; col < mat\_size; col++) {

if (row != curr\_row && col != curr\_col) {

sub\_mat[i \* sub\_size + j] = mat[row \* mat\_size + col];

j++;

if (j == mat\_size - 1) {

j = 0;

i++;

}

}

}

}

}

double GetDeterminant(double\* matrix, int n) {

int determinant = 0;

if (n == 1) {

return matrix[0];

}

if (n == 2) {

return (matrix[0] \* matrix[3]) - (matrix[1] \* matrix[2]);

}

int sign = 1;

double\* temp = new double[(n - 1) \* (n - 1)];

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

GetSubMatrix(matrix, temp, 0, i, n, n - 1);

determinant += sign \* matrix[i] \* GetDeterminant(temp, n - 1);

sign = -sign;

}

delete[] temp;

return determinant;

}

void MatrixOfMinors(double\* matrix, double\* min\_matrix, int n) {

double\* subMatrix = new double[(n - 1) \* (n - 1)];

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

for (int j = 0; j < n; j++) {

GetSubMatrix(matrix, subMatrix, i, j, n, n - 1);

min\_matrix[i \* n + j] = GetDeterminant(subMatrix, n - 1);

}

}

}

void MatrixOfCofactors(double\* matrix, int n) {

int check = -1;

for (int i = 0; i < n \* n; i++) {

if (i % n == 0) {

check \*= -1;

}

if (i % 2 != 0 && check == 1) {

matrix[i] \*= -1;

}

if (i % 2 == 0 && check == -1) {

matrix[i] \*= -1;

}

}

}

void InverseMatrix(double\* matrix, double\* inv\_matrix, int n) {

MatrixOfMinors(matrix, inv\_matrix, n);

MatrixOfCofactors(inv\_matrix, n);

TransposeMatrix(n, n, inv\_matrix);

ScalarMultiplyMatrix(n, n, inv\_matrix, (1.0 / GetDeterminant(matrix, n)));

}

void RotateMatrix(int a\_x, int a\_y, int b\_x, int b\_y, double\* a, double\* b, int rotate) {

int a\_row, a\_col, b\_row, b\_col;

switch (rotate) {

case 1: //90 degrees

for (int i = 0; i < b\_x; i++) {

for (int j = 0; j < b\_y; j++) {

b\_row = i;

b\_col = j;

a\_row = (a\_x - 1) - b\_col;

a\_col = b\_row;

b[b\_row \* b\_y + b\_col] = a[a\_row \* a\_y + a\_col];

}

}

break;

case 2: //180 degrees

for (int i = 0; i < b\_x; i++) {

for (int j = 0; j < b\_y; j++) {

b\_row = i;

b\_col = j;

a\_row = (a\_x - 1) - b\_row;

a\_col = (a\_y - 1) - b\_col;

b[b\_row \* b\_y + b\_col] = a[a\_row \* a\_y + a\_col];

}

}

break;

case 3: //270 degrees

for (int i = 0; i < b\_x; i++) {

for (int j = 0; j < b\_y; j++) {

b\_row = i;

b\_col = j;

a\_row = b\_col;

a\_col = (b\_x - 1) - b\_row;

b[b\_row \* b\_y + b\_col] = a[a\_row \* a\_y + a\_col];

}

}

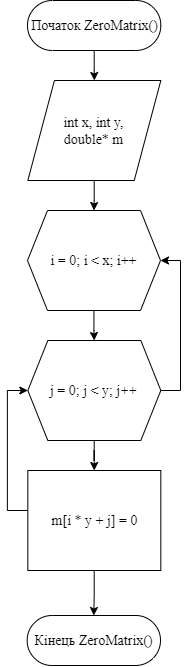
break;

}

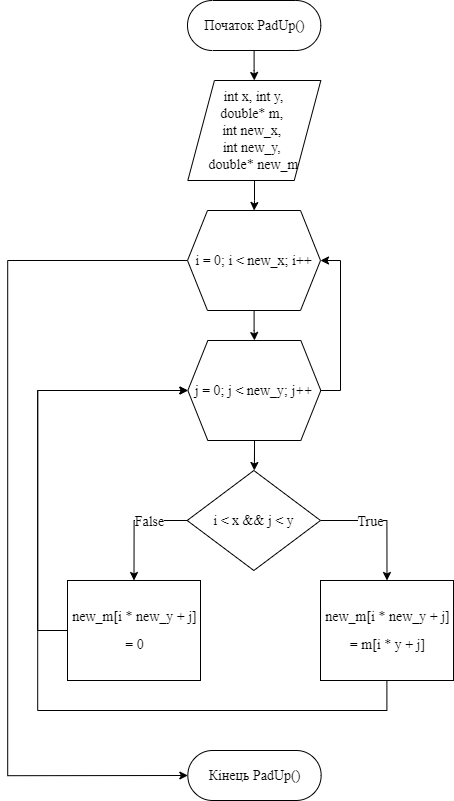
}

## **Б****лок-схеми**

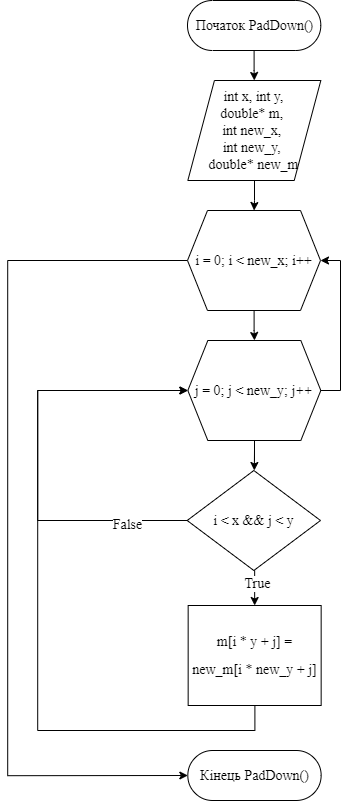
*Блок-схема 1 – ZeroMatrix().*



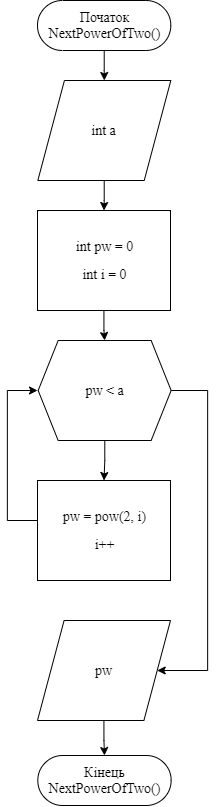
*Блок-схема 2 – PadUp().*

**

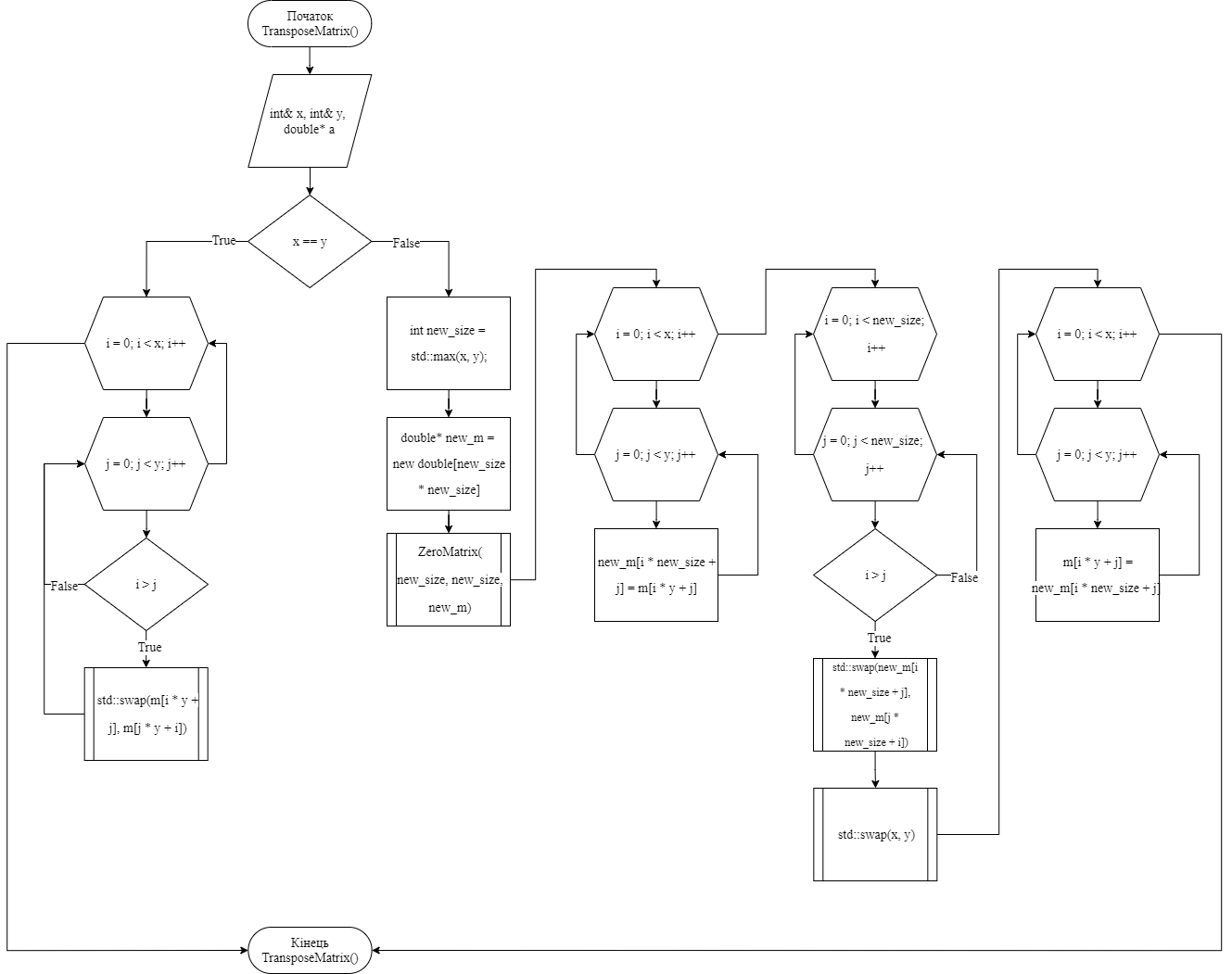
*Блок-схема 3 – PadDown().*

**

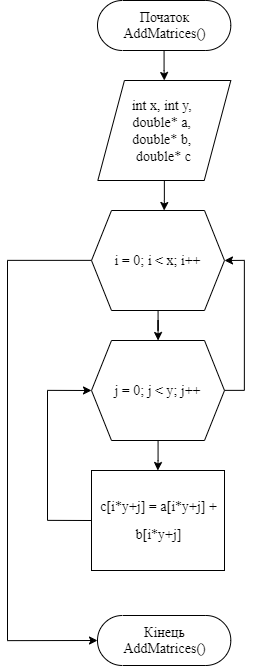
*Блок-схема 4 – NextPowerOfTwo().*

**

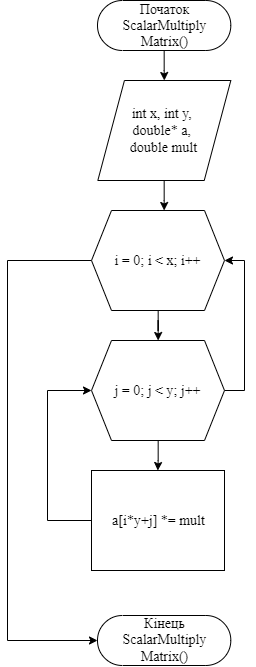
*Блок-схема 5 – TransposeMatrix().*

**

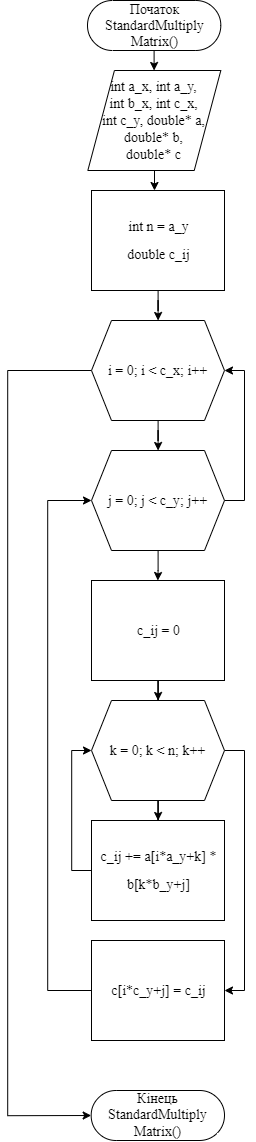
*Блок-схема 6 – AddMatrices().*

**

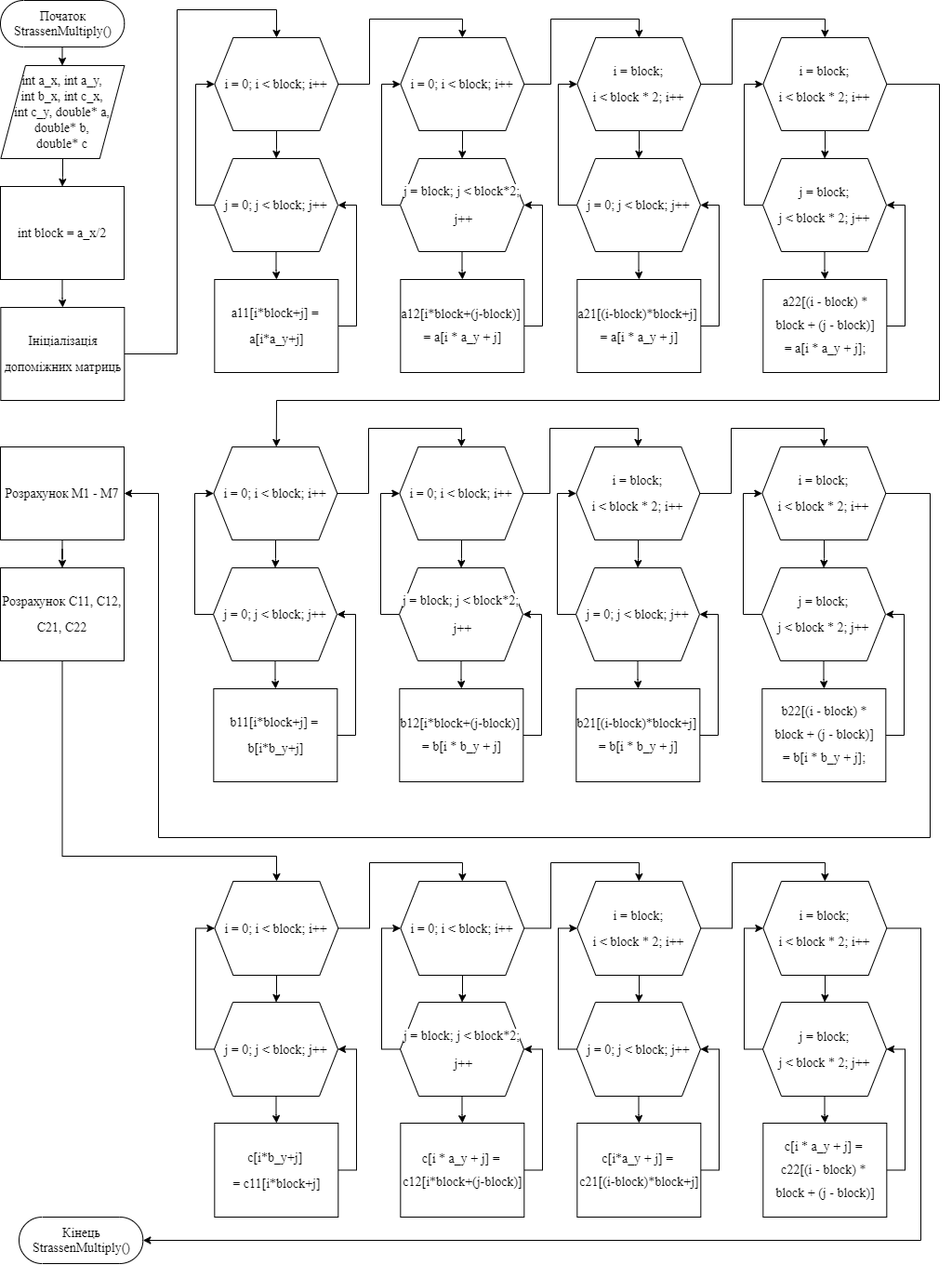
*Блок-схема 7 – ScalarMultiplyMatrix().*

**

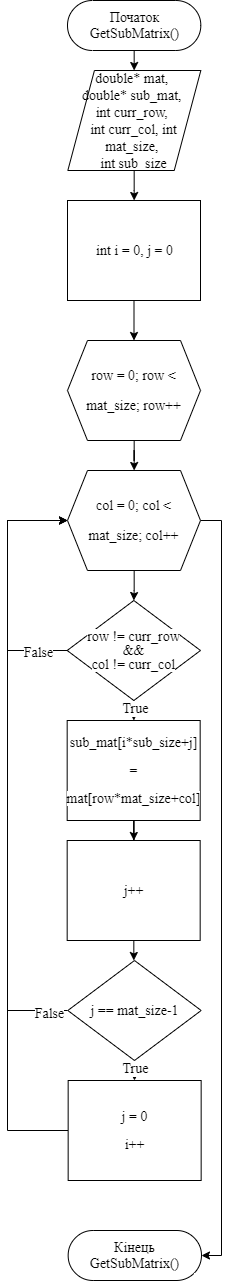
*Блок-схема 8 – StandardMultiplyMatrix().*

**

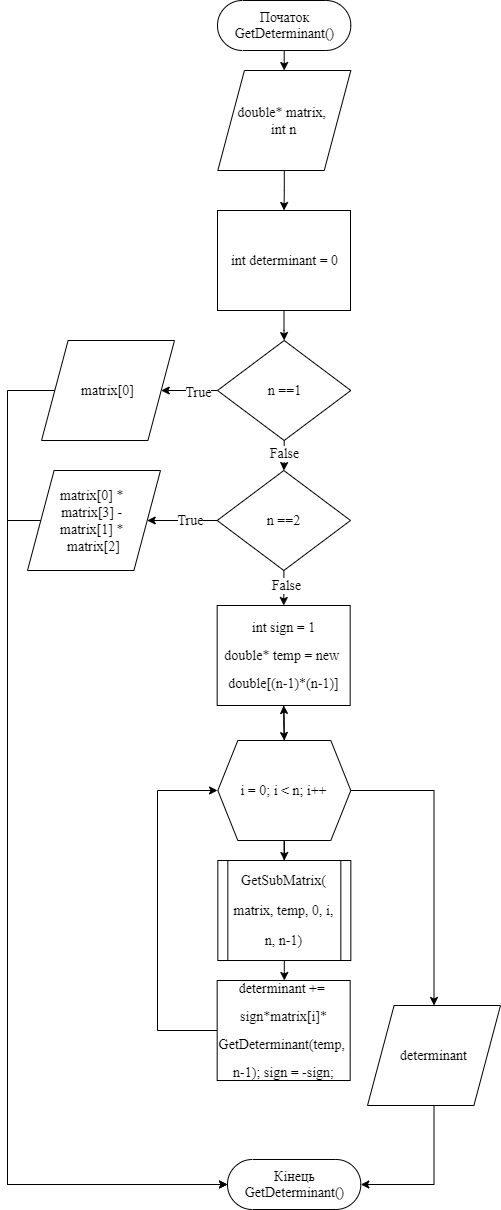
*Блок-схема 9 – StrassenMultiply().*

**

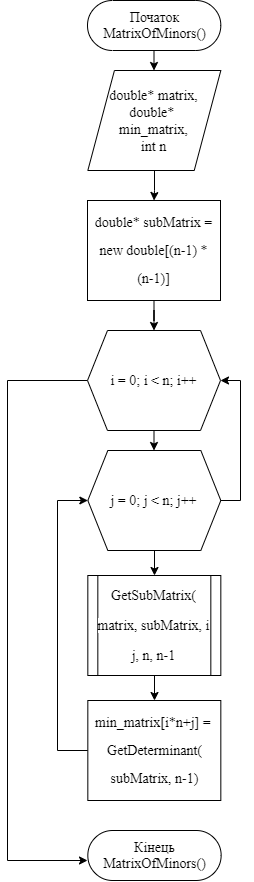
*Блок-схема 10 – GetSubMatrix().*

**

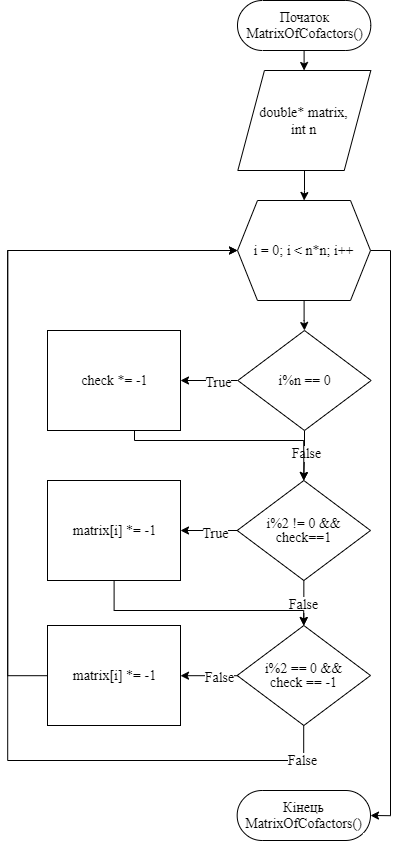
*Блок-схема 11 – GetDeterminant().*

**

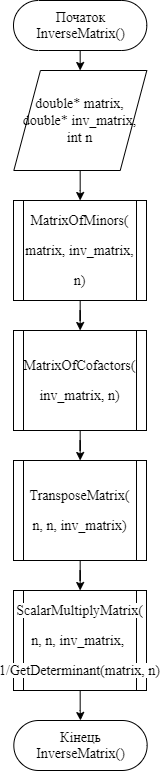
*Блок-схема 12 – MatrixOfMinors().*

**

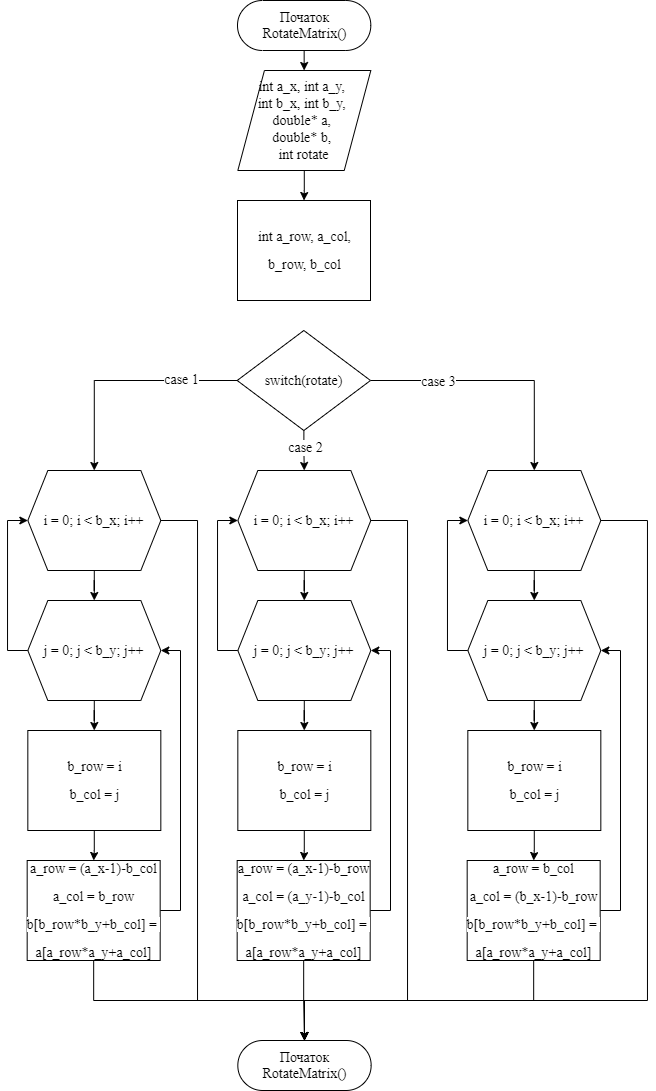
*Блок-схема 13 – MatrixOfCofactors().*

**

*Блок-схема 14 – InverseMatrix().*

**

*Блок-схема 15 – RotateMatrix().*

**