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: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

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**

Краковича Павла Дмитровича

(прізвище, ім’я, по батькові)

1. Тема роботи

Програма побудови графіків по введеним функціям

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**

Графік — наочне зображення кількісної залежності показників різних явищ, процесів тощо. У математиці графік — це наочне зображення кількісної залежності показників різних явищ, процесів тощо. Графікі зображаються в двовимірному та трьохвимірному просторі.

Графікі застосовуються у всіх галузях бо можуть зображати найрізноманітніші залежності покизників різних явищ та процесів, наприклад: у науці, математиці, програмуванні, медецині, комерції, комп’ютерній графіці, соціології, економіці та хімії.

Курсова роботи присвячена наглядній побудові графіків математичних функцій у середовищі розробки C++.

В мережі існує велика кількість найрізноманітніших онлайн-програм для побудови математичних функцій по введеним графікам, як-от:

* [Desmos](https://www.desmos.com)
* [Mathway Graphing](https://www.mathway.com/Graph)
* Вбудований в пошукову сторінку [Google](https://www.google.com)

Однак, вони потребують постійного та стабільного доступу до інтернету.

Моє завдання полягає в створенні доступного оффлайн додатка з інтуїтивно зрозумілим інтерфейсом та керуванням, у якому можливо буде зрунчо будувати наглядні графіки математичних функцій. Для його виконання потрібно буде написати алгоритм для конвертації рядка типу string у математичну формулу та потім, у готову відповідь для будування графіка.

# **1. Обґрунтування та вибір алгоритмів**

Для роботи програми потрібно створити вікно з наступними об’єктами:

* Поле для відображення графіку
* Поле для вводу
* Меню для вводу даних за допомогою мишки
* Кнопки для вмикання \ вимикання спеціальних можливостей

Поле для вводу даних має отримувати формулу або дані та конвертувати їх у координату для побудови графіку. Тому, розробку програми можна умовно розділити на 2 пункти:

* Створення вікна, елементів та додавання логіки їх роботи
* Створення парсеру рядка

**1.1 Створення вікна, елементів та додавання логіки їх роботи**

Для створення додатка я використовую графічну бібліотеку SFML.

SFML (Simple and Fast Multimedia Library) – одна з найзручніших та найшвидших графічних бібліотек для C++. SFML використовується здебільшого невеликими стартапами та програмістами, для яких створення ігор – хобі. SFML є популярним серед невеликих команд завдяки тому, що розробка графічної частини програми не вимагає написання великих обсягів коду.

Ця бібліотека була обрана мною через доступність її документації та відносну простоту використання, порівнюючи с іншими бібліотеками.

Аналогами цієї бібліотеки є:

* OpenGL
* Qt
* GTK+

Однак, всі вони мають певні недоліки, порівняно з SFML.

OpenGL – хоча і являється надзвичайно потужним інструментом для розробки, але набагато складніша у використанні, потребує більше ресурсів та має менше доступної документації.

Qt – має свої складності та умовності з ліцензію, а додатки мають доволі велику вагу.

GTK+ - орієнтована на мову C

Тому, я зупинив свій вибір на SFML.

**1.2 Створення парсеру рядка**

Будь-який калькулятор чи програма побудови функцій використовує парсер рядка у вигляд інверсного польського запису для роботи з даними. Інверсного польського запис значно полегшує процес зчитування даних. Потреба у ній з’явилась тому, що програмі потрібно розуміти, що вона зчитує та залежно від цього робити певні арифметичні дії. Для отримання інверсного польського запису використовують алгоритм сортувальної станції.

Алгоритм сортувальної станції — метод синтаксичного розбору математичних виразів наданих в інфіксній нотації. Його можна використовувати для отримання інверсного польського запису. Алгоритм було винайдено Едсгером Дейкстрою і названо алгоритм «сортувальної станції», бо ця операція нагадує дію залізничної сортувальної станції.

Написання парсера, потребує перетворення рядка з даними таким чином, щоб кожен елемент знаходився в окремій комірці та мав індифікатор свого типу (число, функція, константа і тп.).

Для цього, доцільно використати структуру, яка буде зберагіти значення комірки і тип значення та перетворити введений рядок в масив токенів. Наступним кроком має бути перетворення масиву токенів з інфіксного запису (нормального для нас) у постфіксний формат запису (інверсна польська запис). Отримавши рядок у форматі зворотньої польского запису досить просто підрахувати його значення та повернути результат.

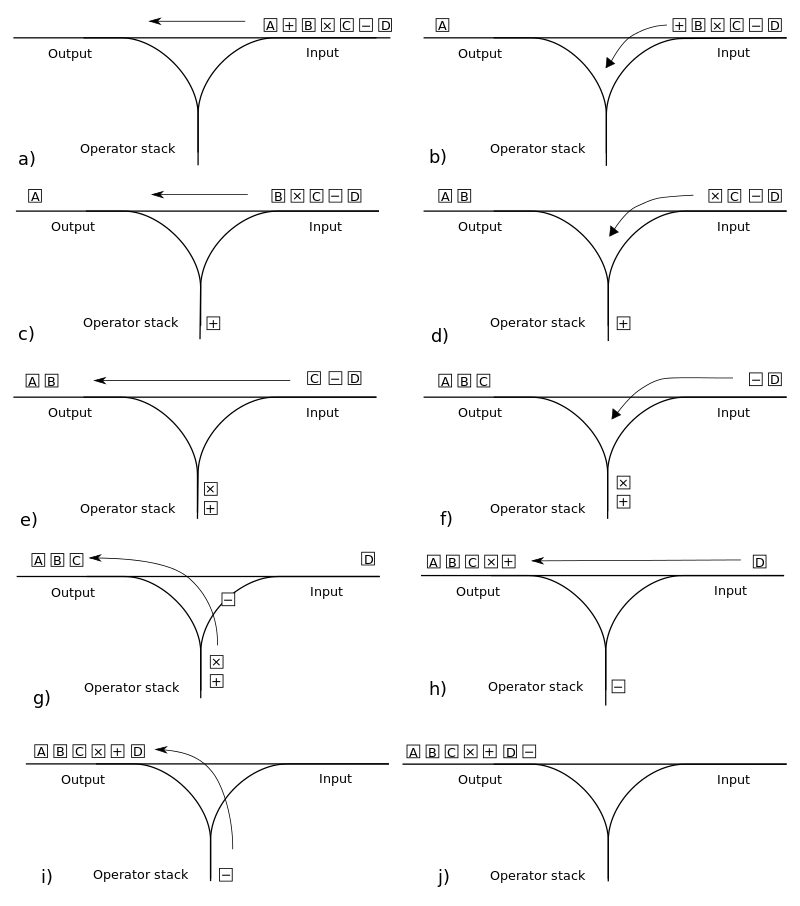


Рисунок 1 – Приклад роботи алгоритму сортвуальної станції

**2. Розробка програми**

## **2.1. Загальні відомості**

Створена програма має назву «Graph plotter». Вона була написана на мові С++ стандарту С++ 17, за допомогою IDE CLion та графічної бібліотеки SFML версії 2.5.1. Для того, щоб у користувача була можливість використовувати цю програму, його ПК має задовільняти наступним мінімальним вимогам:

* ОС – Windows 7,8,10,11 x64
* Процессор – будь-який сучасний
* Графічна карта – 100 Мб пам’яті
* Оперативна пам'ять – Не менше 50 Мб пам’яті
* Диск – 1 Мб вільної пам’яті

Тобто, програма буде працювати на будь-якій сучасній системі.

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

## **2.2. Функціональне призначення**

В программі були кивористані два класи:

* Textbox
* Button

Клас Textbox відповідає за поле вводу, його параметри та логіку роботи.

Цей клас містить в собі наступні функції (у форматі «Назва – Опис»):

* Textbox() – Конструктор класу, приймає та встановлює параметри поля
* setFont() – Приймає та встановлює шрифт для тексту
* setPosition() – Приймає та встановлює позицію поля для вводу
* getText() – Повертає рядок вводу
* drawTo() – Приймає показчик на вікно та малює поле вводу
* typedBurronMenu() – Приймає введений символ з меню та опрацьовує його
* typedBurronKeyboard() – Приймає введену клавішу та опрацьовує її
* deleteChar() – Видаляє останній символ в рядку вводу, якщо такий існує
* deleteString() – Очищає рядок вводу, якщо він не порожній
* inputLogic() – Функція логіки обробки
* deleteLastChar() – Функція логіки видалення останнього елемента рядка

Клас Button відповідає за кожну кнопку меню та за три спеціальні кнопки.

Цей клас містить в собі наступні функції (у форматі «Назва – Опис»):

* Button() - Конструктор класу, приймає та встановлює параметри кнопки
* setTexture() – Приймає та встановлює текстуру кнопки
* setScale() – Встановлює розмір для спеціальних кнопок
* drawButton() – Приймає вказівник на вікно та малює кнопку
* setColor() – Приймає та встановлює колір кнопки
* getGlobalBounds() – Повертає координати меж кнопки

Функції алгоритму маневрової станції:

* vector<Token> parse() – Перетворює вхідний рядок у набір токенів, групуючи функції.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| c | o | s | ( | x | ) | + | ( | - | x | ) |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| cos | ( | x | ) | + | ( | - | x | ) |

До перетворення Після перетворення

* vector<Token> parseNegativeNumbers() – Перетворює набір токенів, додаючи перед від’ємни токенами значення 0, для корерктної роботи парсера в інверсну польську запис.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| cos | ( | x | ) | + | ( | - | x | ) |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| cos | ( | x | ) | + | ( | 0 | - | x | ) |

До перетворення Після перетворення

* queue <Token> rpn\_tokens() – Конвертує запис з інфіксу в постфікс (робить з вектора чергу у форматі інверсного польського запису.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| cos | ( | x | ) | + | ( | 0 | - | x | ) |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| x | cos | 0 | x | - | + |

До перетворення Після перетворення

* float stack\_calc() – Конвертує постфіксну запис у готовий результат.

Додаткові функції:

* roundValue() – Приймає значення координат та округлює їх до сотих
* intToString() – Приймає значення типу float і конвертує у string

## **2.3. Опис логічної структури**

В цьому розділі детально описані функції алгоритму сортувальної станції

|  |  |  |
| --- | --- | --- |
| Назва функції | Вхідні дані | Опис алгоритму |
| vector<Token> parse() | int x, int y – Розміри матриці  double\* m – Матриця | Заповнює всі елементи матриці нулями. [Див. блок-схему 1.](#block1) |
| vector<Token> parseNegativeNumbers() | 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) |
| queue <Token> rpn\_tokens() | 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) |
| float stack\_calc() | 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.

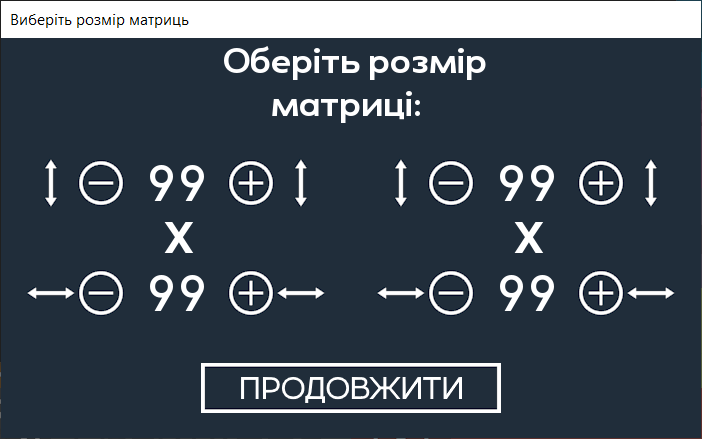


Рисунок 2 – Максимальний розмір двох матриць при множенні.

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.

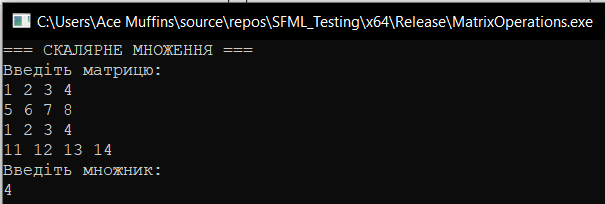


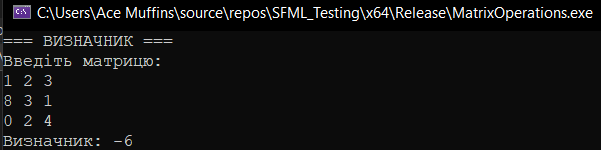
Рисунок 4 – Приклад введення матриці та множника при операції

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.

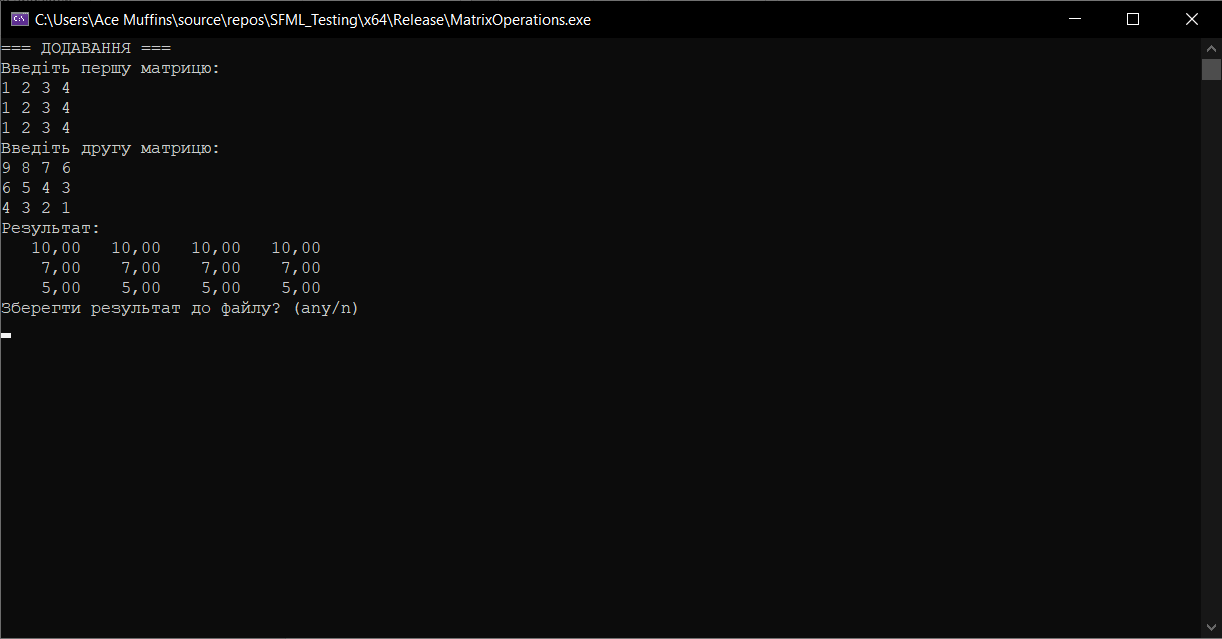


Рисунок 6 – Приклад виведення матриці дійсних чисел після операції додавання.

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

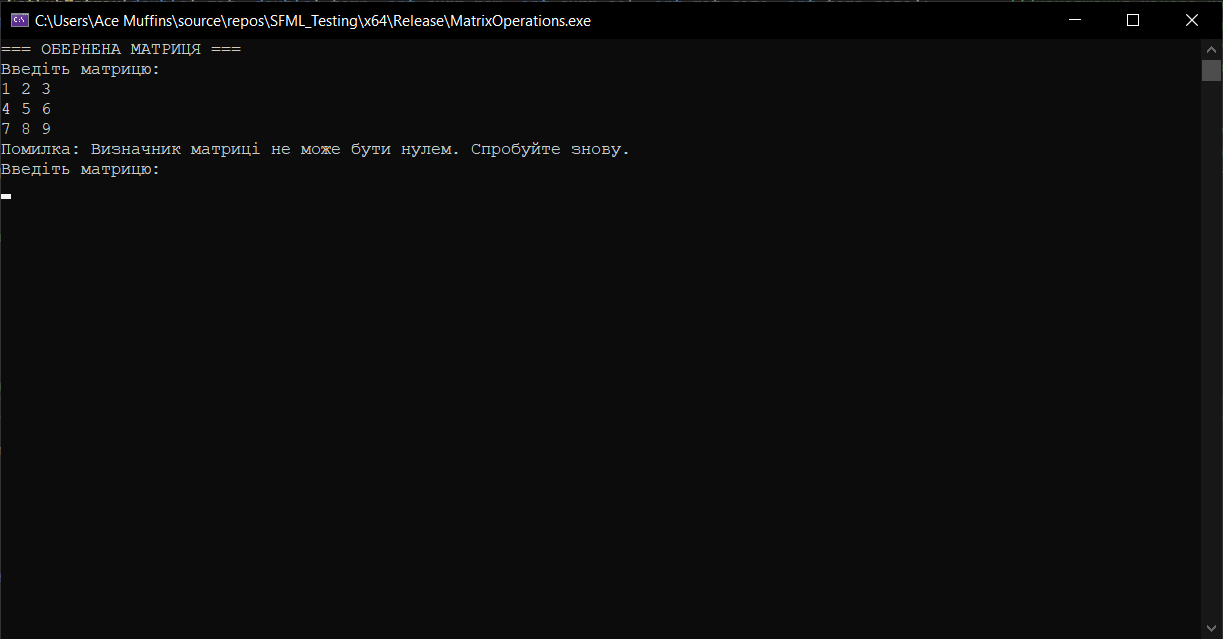
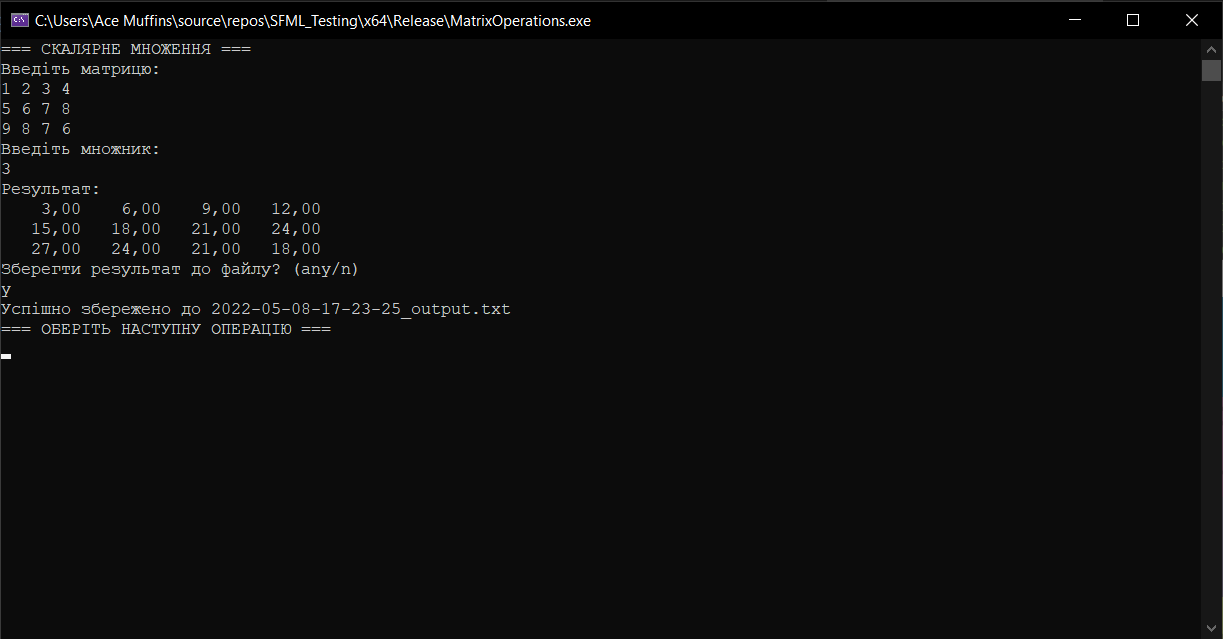
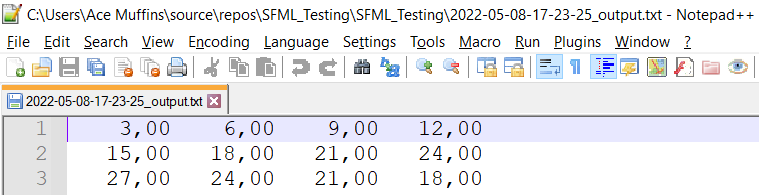


Рисунок 7 – Приклад виведення помилки після операції обернення.

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.

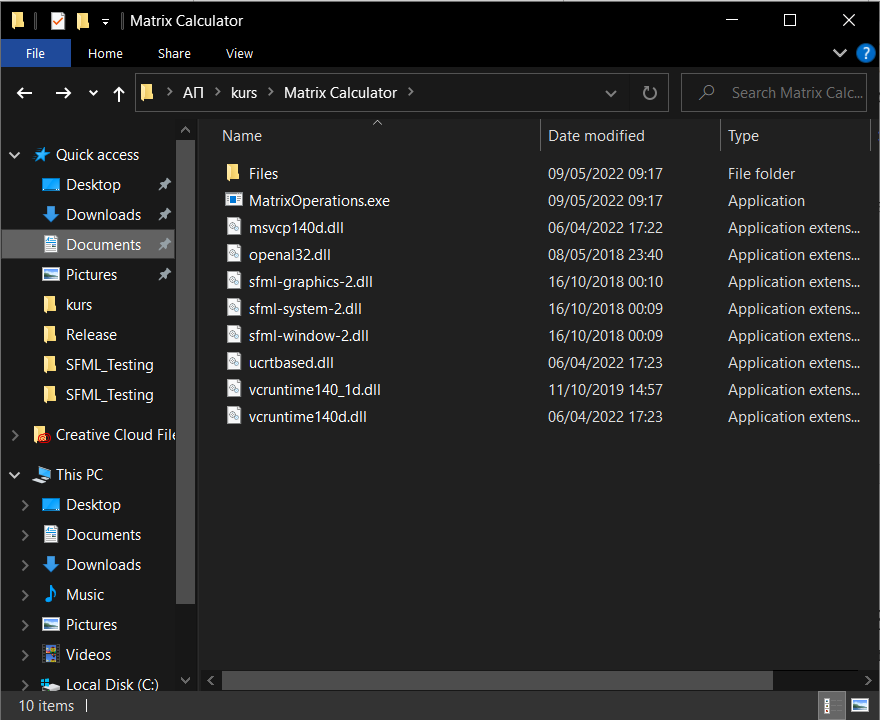
****

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.

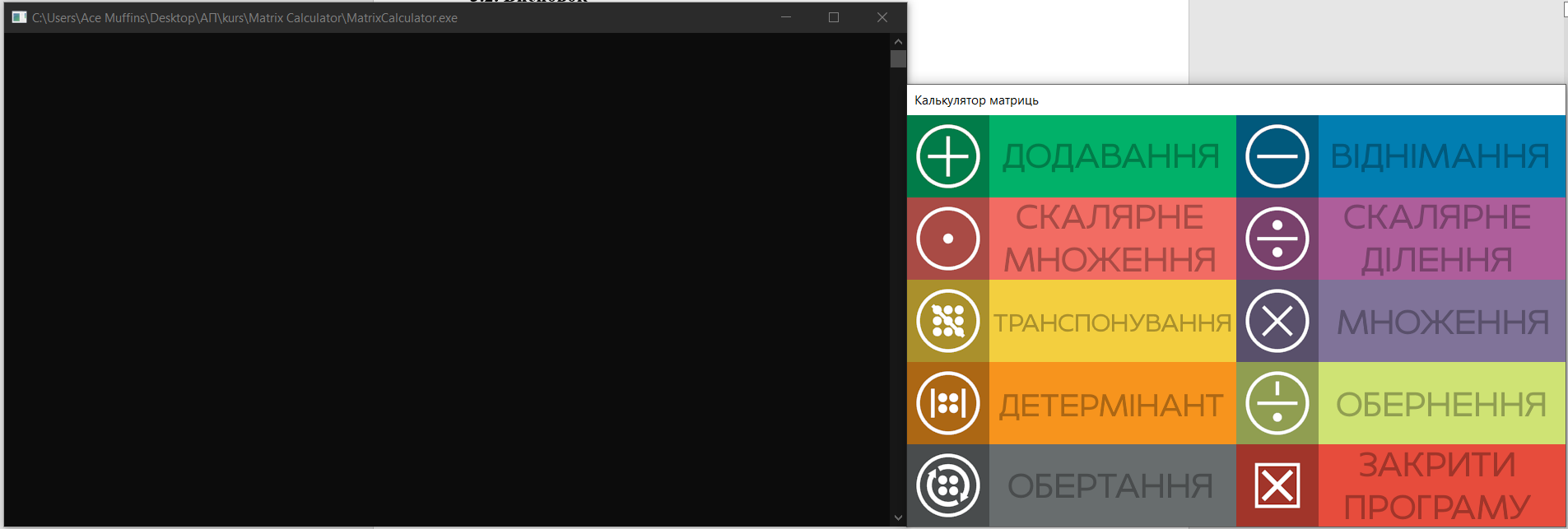


Рисунок 11 – Меню вибору операцій та консоль.

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.

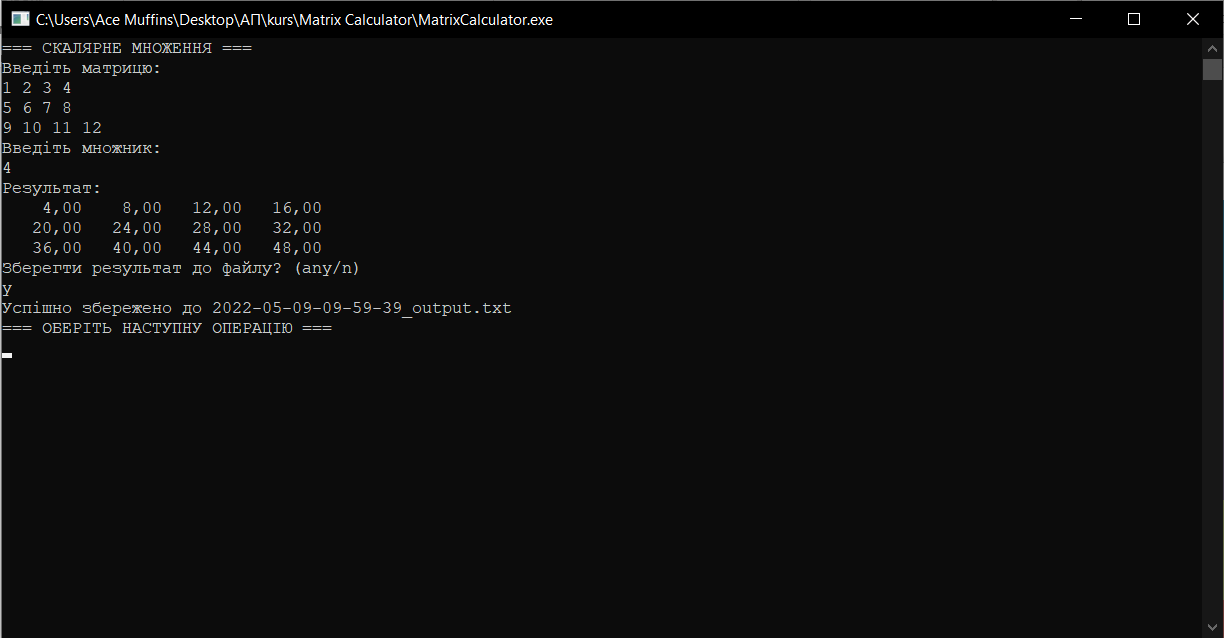


Рисунок 13 – Приклад операції скалярного множення, та збереження результату

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**

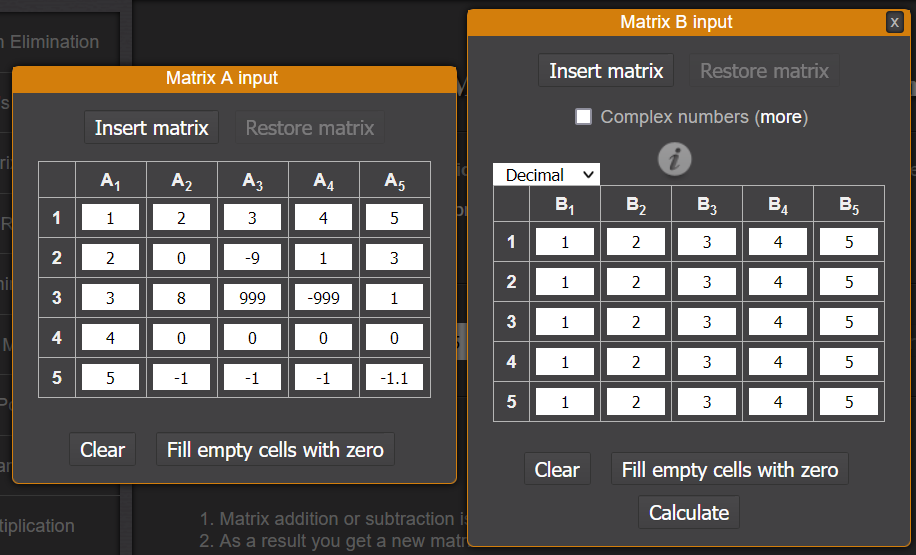
За вибору алгоритмів для виконання поставленої задачі, було знайдено кілька альтернатив до тих, що були використані в остаточній версії програми. Серед них, найбілша різниця є в алгоритмі знаходження визначника матриці. Даний алгоритм є рекурсивним, та має складність , що не є оптимальним, але будь-які 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.

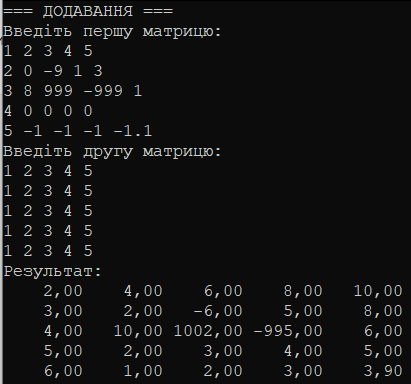
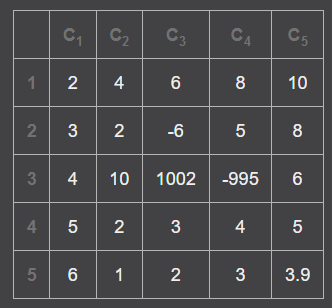
# **3. Висновки**

## **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.

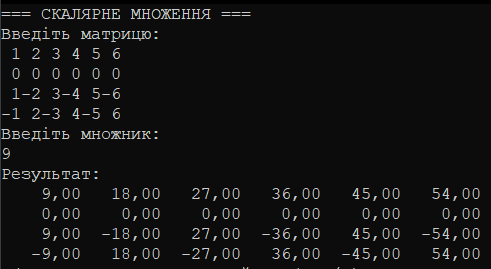
* Додавання\Віднімання

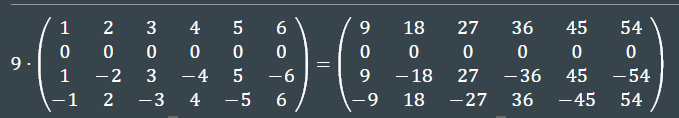




Рисунки 14-16 – Перевірка операції додавання.

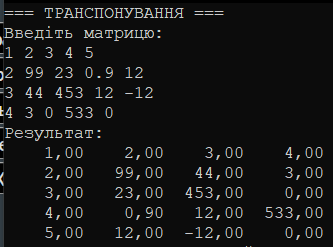
* Скалярне множення\Скалярне ділення

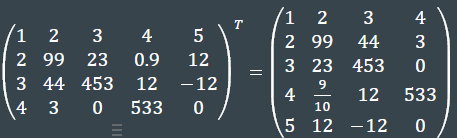




Рисунки 17-18 – Перевірка операції скалярного множення.

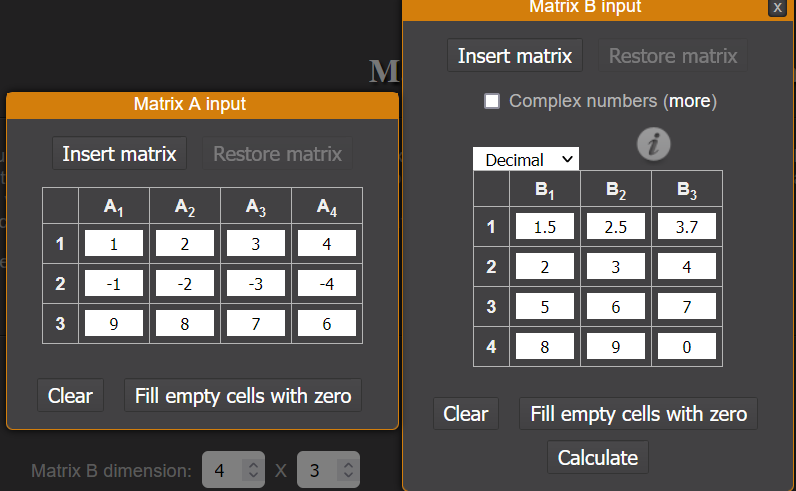
* Транспонування

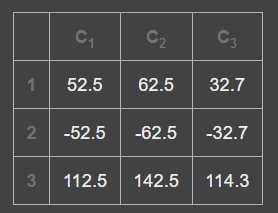
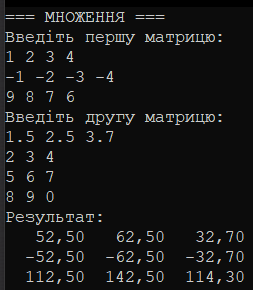




Рисунки 19-20 – Перевірка операції транспонування.

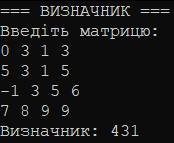
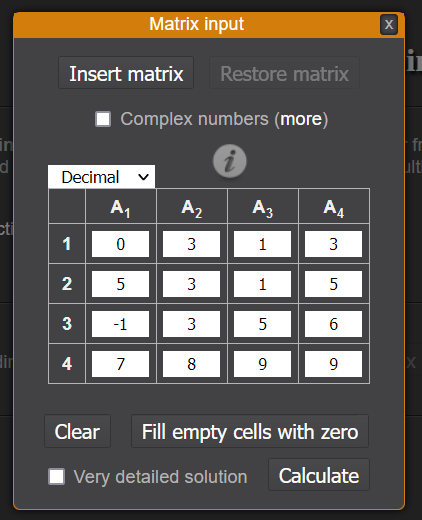
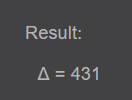
* Множення



‘ 

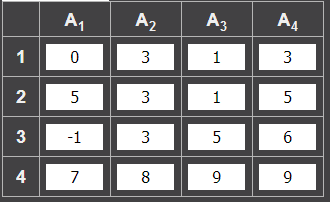
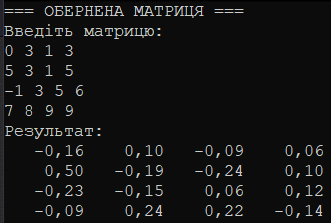
Рисунки 21-23 – Перевірка операції множення.

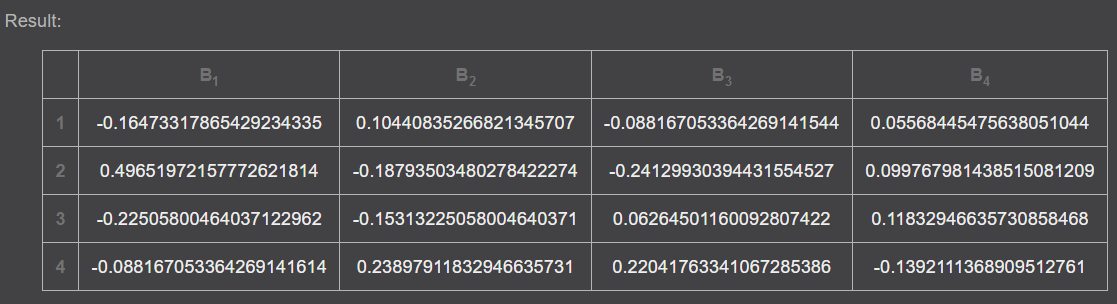
* Знаходження детермінанту

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**

## **Б****лок-схеми**