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.

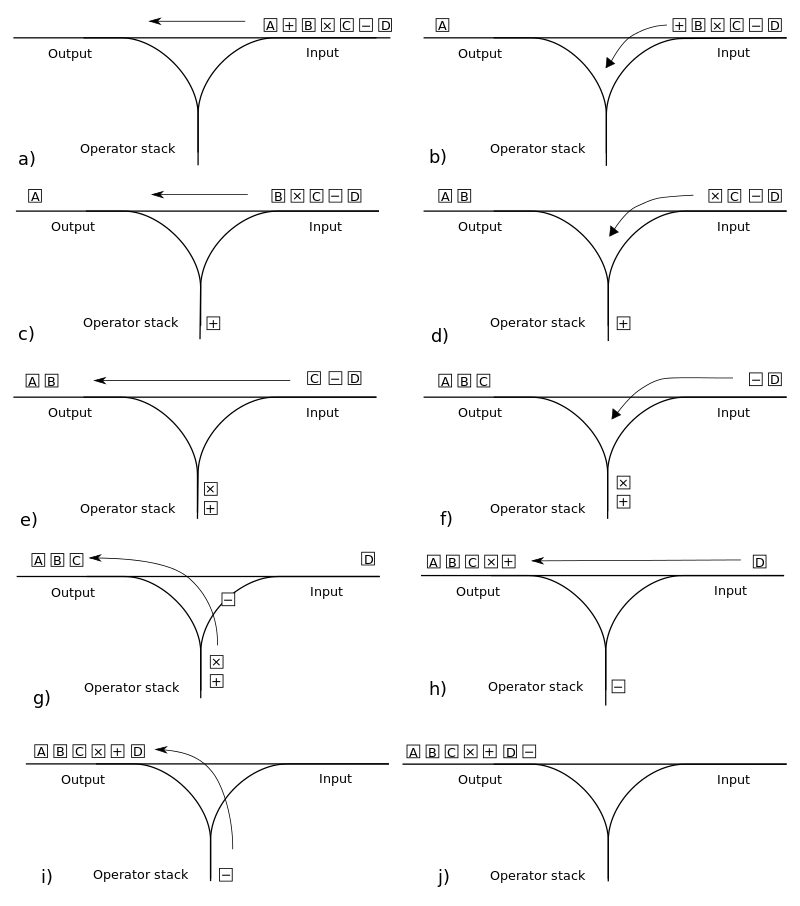


Рисунок 1 – Приклад роботи алгоритму сортвуальної станції

**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

* int getPrecision() – Отримує оператор та повертає його пріоритетність.
* queue <Token> RPN() – 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, принцип їх роботи та взаємозв’язок між ними.

|  |  |  |
| --- | --- | --- |
| Назва функції | Вхідні дані | Опис алгоритму |
| vector<Token> parse() | const string& expression | Приймає рядок, введений користувачем та робить пе-рший крок по конвертації виразу у набір токенів.  Задача цієї функції – це перетворення рядка у набір токенів, де тригонометричні функції будуть знаходитись в одній комірці.  [Детальніше](#parse)  Блок-схема |
| vector<Token> parseNegativeNumbers() | vector<Token>& tokens | Приймає набір токенів, з минулої функції та шукає мінуси перед токенами, які відповідають за знак токену та додає 0 перед ними. Таким чином, отримуємо готовий набір токенів, який можна конвертувати у інверсну польску нотацію.  [Детальніше](#parseNegativeNumbers)  Блок-схема |
| int getPrecision() | const Token& symbol | Приймає оператор та по-вертає його пріоритетність  [Детальніше](#getPrecision)  Блок-схема |
| queue <Token>RPN() | const vector<Token>& tokens | Приймає набір токенів та перетворює їх з інфіксної у постфіксну запис.  [Детальніше](#RPN)  Блок-схема |
| double stackСalc() | queue<Token>& rpn\_tokens, double x | Прийає чергу з токенів у форматі інверсного по-льского запису та значення змінної x, рахує значення всередині черги та повертає його як елемент на вершині стека  [Детальніше](#stackСalc)  Блок-схема |
| double evaluate() | string& expresision, double x | Прийає рядок, який ввів користувач та значення x з циклу для виводу зобра-ження графіка на єкран.  У функції викликаються попередні функції (1,2,4) та повертається функція 5. |

* **vector<Token> parse(const string& expression)**

1. На початку функції створюється: вектор типу Token, який буде зберігати в кожній комірці значенння та його тип; дві рядкові змінні для збеігання чисел та букв (тригонометричних функцій).
2. Починається цикл, в якому функція проходить по всім значенням рядка та оцінює їх.
3. Після проходження по всім елементам рядка, якщо в змінних чисел або букв залишились значення, то функція додає їх до фінального стеку.

* **vector<Token> parseNegativeNumbers(vector<Token>& tokens)**

1. На початку функції створюється новий вектор типу Token, в який будуть переписані значення та він буде повернутий в кінці функції.
2. Починається цикл, в якому функція проходить по токенам та оцінює їх.
3. Якщо функція знохить знак “-“, який описує знак токена то вона додає перед цим токеном 0, ніби віднімаючи токен від 0 та додає їх до стеку. Таким чином, ми отримуємо набір токенів, який буде корректно опрацьований у наступній функції.

* **int getPrecision(const Token& symbol)**

Функція приймає токен, звертається до значення його значення у комірці та повертає пріоритетність за наступною схемою:

* додавання та віднімання мають найнижчу пріоритетність – 1;
* множення та ділення мають пріоритетність – 2;
* піднесення в степінь має найвищу пріоритетність – 3.
* **queue <Token>RPN(const vector<Token>& tokens)**

1. На початку функції створюється новий вектор та черга типу Token.
2. Починається цикл, в якому функція проходить по токенам та оцінює їх.
3. Після циклу очищає стек операторів та додає їх до черги зі всіма значеннями у форматі інверсного польського запису.

* **double stackСalc(queue<Token>& rpn\_tokens, double x)**

1. На початку функції створюється новий вектор та черга типу double та змінна у якій буде повернено резултат.
2. Починається цикл, який працює доки черга містить елементи. Цей цикл рахує вміст черги та додає його до стеку.
3. Змінній result присвоюється значення верхнього елементу стеку та повертається функцією.

## 

## **2.4. Максимальне завантаження**

Завдяки використанню бібліотеки cmath, список можливих функцій легко модифікується та можна додати будь-які функції, але для використання дозволені наступні:

* sin()
* cos()
* tan()
* cot()
* ln()
* sqrt()

Довжина функції, яку можна ввести обмежена довжиною поля вводу. Це було зроблено для уникнення моментів, коли функція могла вийти за видиму межу поля програми.

Всі буквенні значення окрім змінної “x” мають бути введені маленькими літерами. Однак, це можливо змінити незначною модифікацією коду парсерса.

Додавання можливості введення тільки змінної “x” великою літерою – це демонстрація можливостей аналогічної модифікації для інших змінних та функцій.

Також, можливе віддалення та наближення графіка було обмежено. Це обмеження було зроблено тому, що завелике віддалення або наближення не мають сенсу.

## **2.5. Вхідні та вихідні дані**

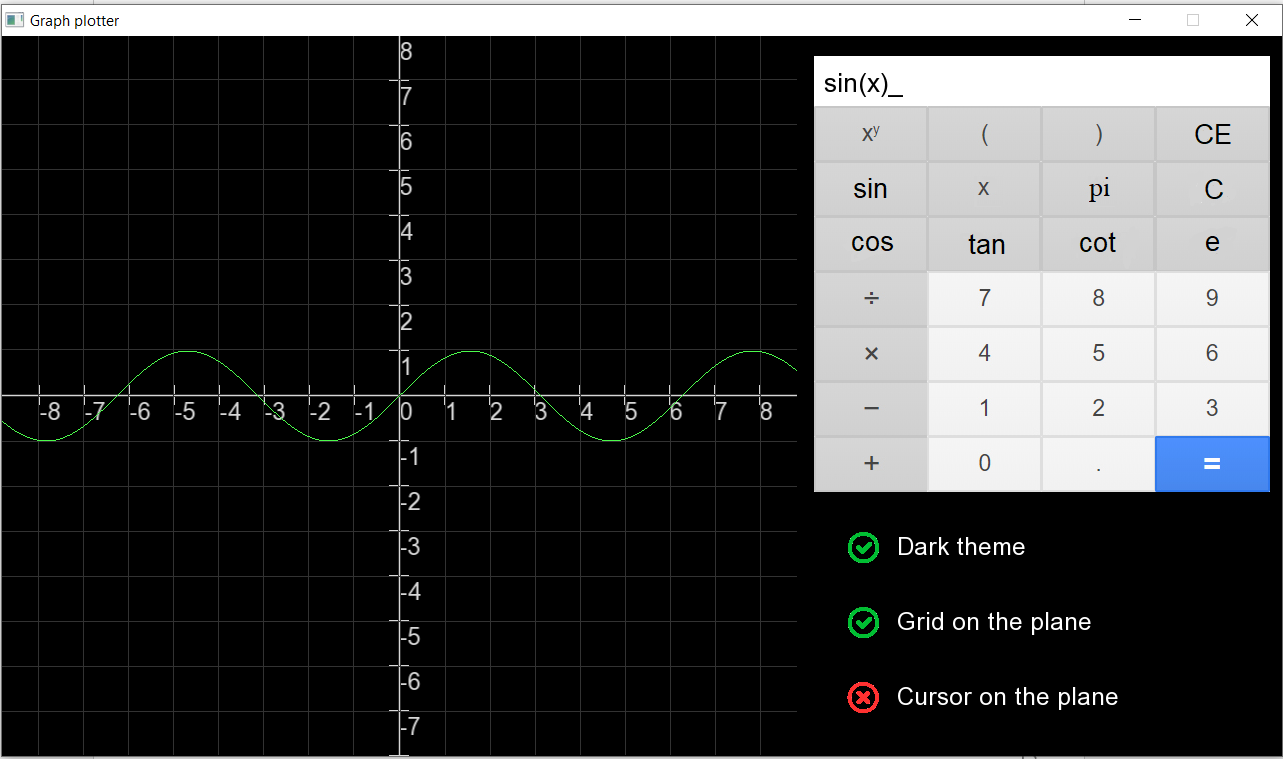
Дані в програму вводяться двома способами:

* За допомогою клавіатури
* За допомогою кнопок меню

З клавіатури буквені дані можливо вводити лише англійськими маленькими літерами (окрім “x”, його можна вводити з великої літери).

При введені даних, вони додаються до поля вводу, яке являє собою рядок типу string. Дані обробляються за допомогою алгоритма сортувальної станції, який конвертує рядок з введеними даними в інверсну польску нотацію, рахує значення постфіксу та виводить на єкран графік по заданим точкам.

Графік виводиться у головному вікні програми. З графіком можна взаємодіяти. Його можна переміщати, наближати та віддаляти. Також, за допомогою функціональних кнопок можна користуватися додатковими функціями програми.



Приклад роботи вікна програми з введеними даними у поле вводу

## **2.6. Керівництво користувачу**

Для того щоб встановити дану програму, потрібно розархівувати архів “Graph plotter.zip” у зручну директорію.

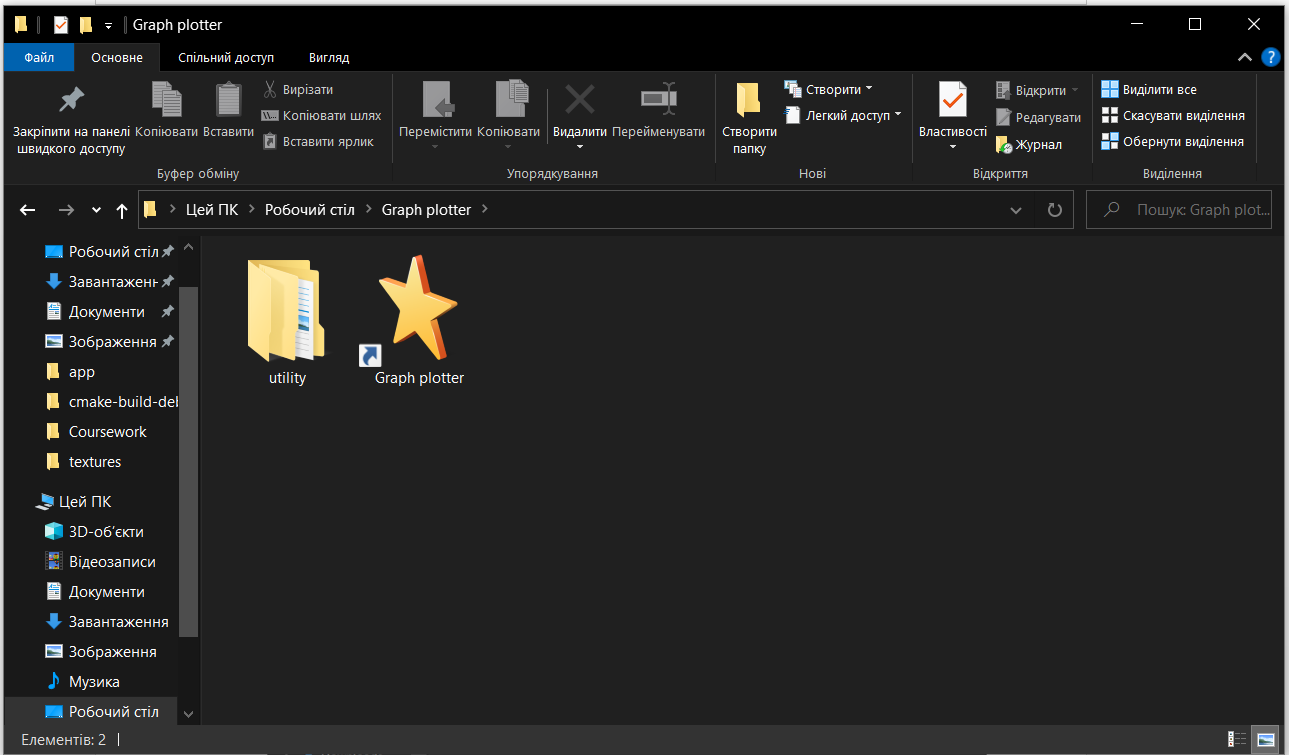
****

Рисунок 10 – Файли, що знаходяться в архіві “ Graph plotter.zip”.

Для того, щоб запустити програму, потрібно два рази натиснути лівою кнопокою миші по ярлику. Після відкриття, на екран висвітиться вікно з програмою.

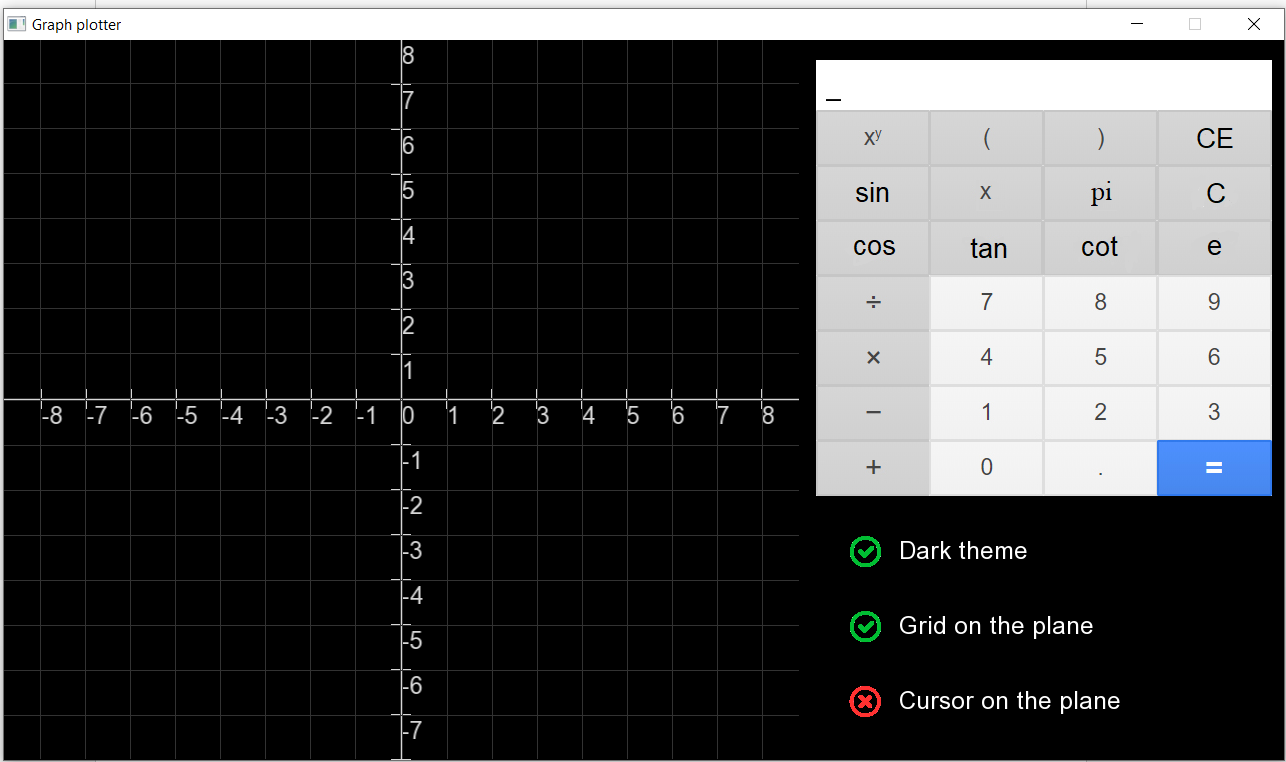


Рисунок 11 – Поле графіку та меню з додатковими кнопками.

Можливі дії з програмою: (натисніть на посилання, щоб дізнатись детальніше)

* [Введення даних за допомогою меню](#Введення_даних_за_допомогою_меню)
* [Введення та опрацювання даних за допомогою клавіатури](#Введення_даних_за_допомогою_клавіатури)
* [Маштабування графіка](#Маштабування)
* [Переміщення по площині](#Переміщення)
* Увімкнення/вимкнення додаткових функцій програми

**Детальна інструкція по керуванням програмою:**

Щоб ввести дані за допомогою меню, необхідно лівою кнопкою миші натискати необіхдні кнопки. Меню має класичний та інтутивний вигляд.

Дані з клавіатури вводяться лише англійськими літерами. Щоб вивести графік на єкран – натисність клавішу «Enter». Для очищення поля вводу використовується клавіша «Backspace».

Щоб приблизити графік, потрібно використовувати колесико мишки або стрілочки вверх/вниз на клавіатурі.

Щоб рухати графік, потрібно затиснути ліву кнопку миші та рухати нею в протилежну сторону, від бажаного зміщення.

Щоб повернутись до початкового положення, потрібно натиснути клавішу «Esc».

Для перемикання додаткових кнопок достатньо один раз натиснути лівою клавішою миші по кнопці.

Прикади використання програми зображені на наступних рисунках:

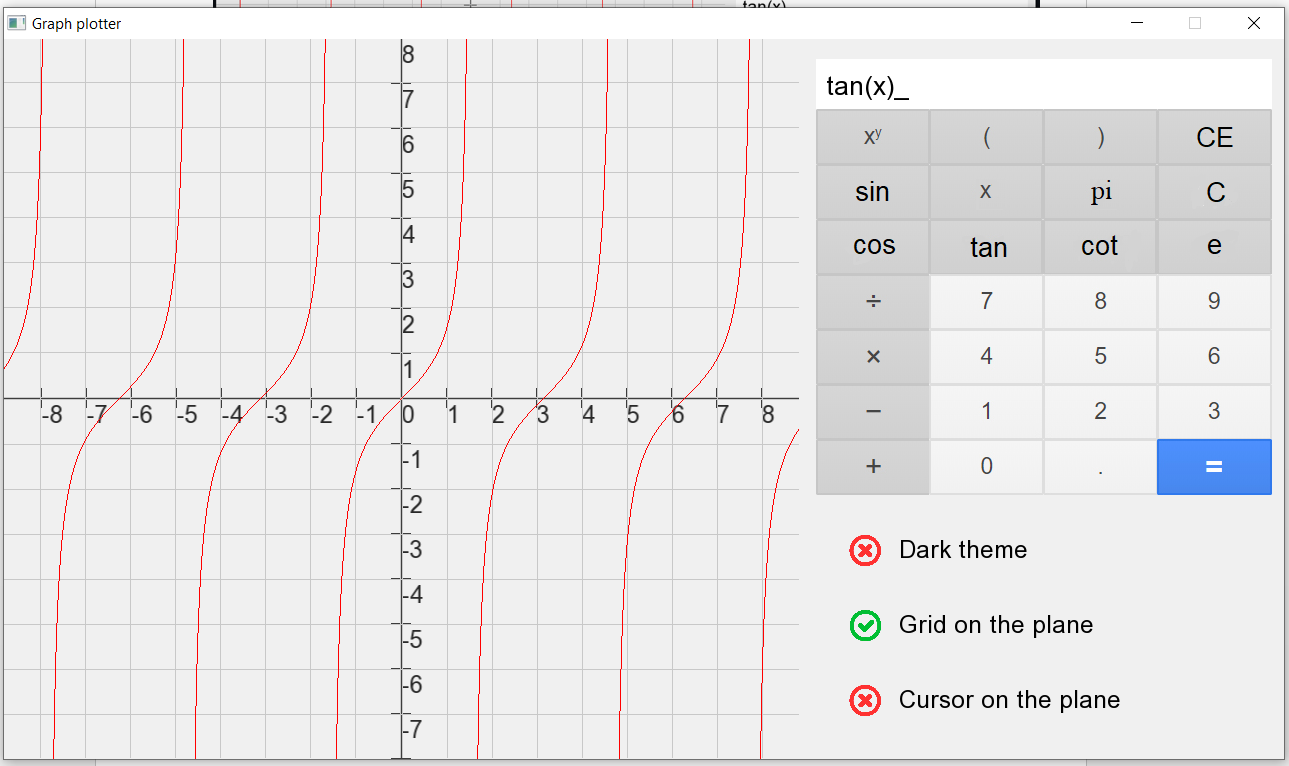


Рисунок 7 – Приклад використання програми №1

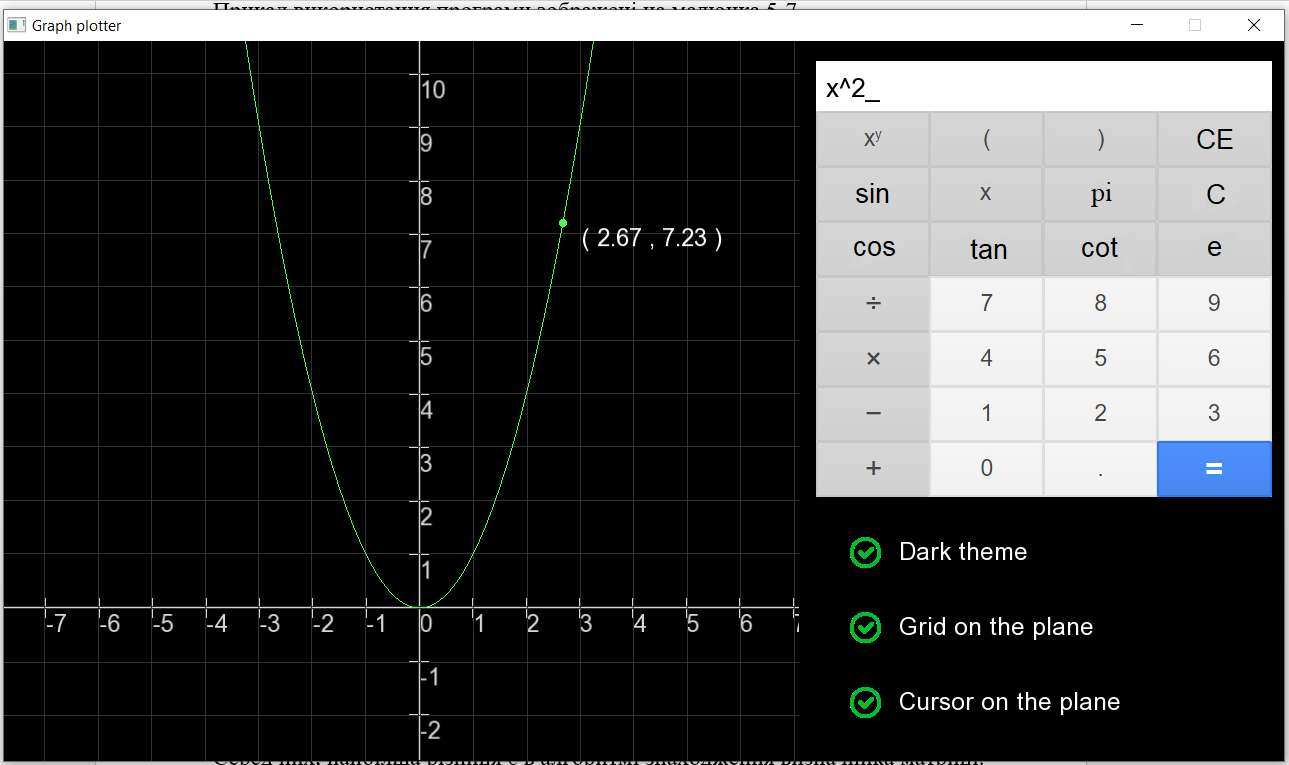


Рисунок 8 – Приклад використання програми №2

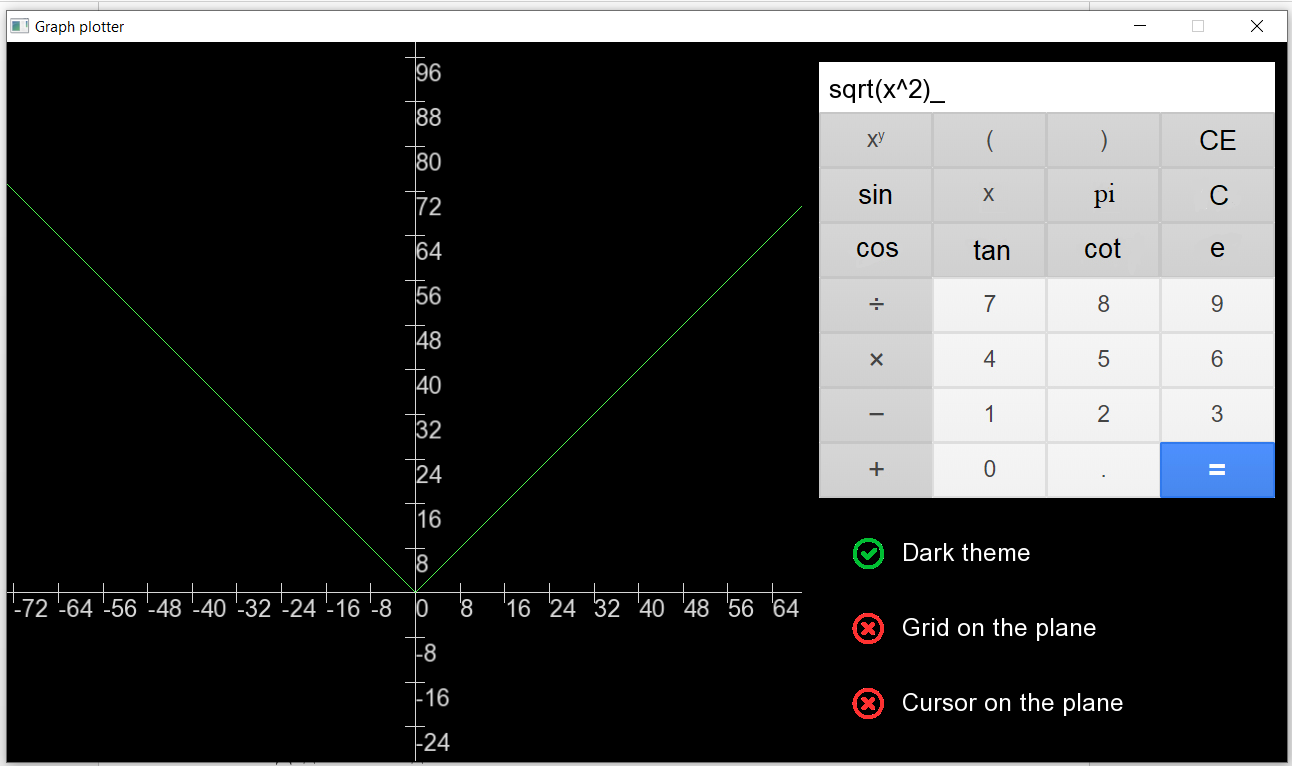


Рисунок 9 – Приклад використання програми №3

## **2.7. Керівництво розробнику**

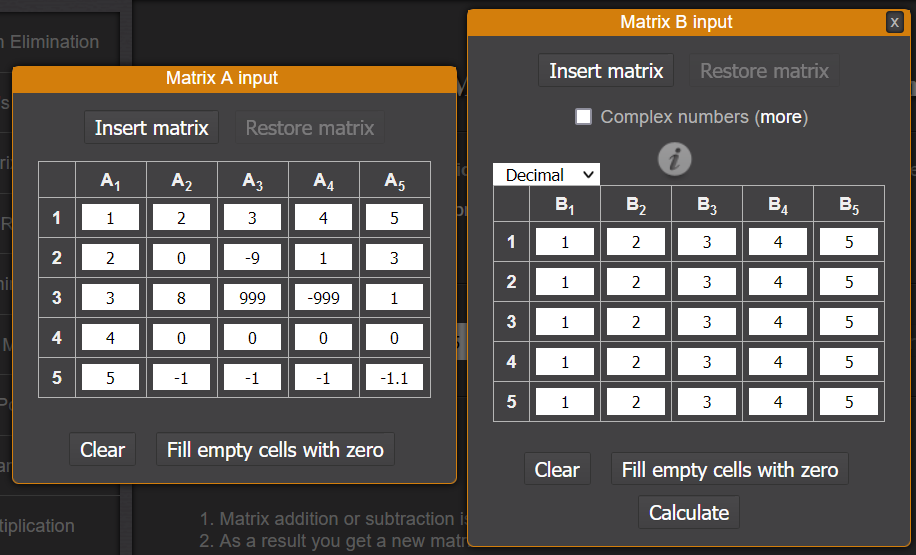
За вибору алгоритмів для виконання поставленої задачі, було знайдено кілька альтернатив до тих, що були використані в остаточній версії програми. Серед них, найбілша різниця є в алгоритмі знаходження визначника матриці. Даний алгоритм є рекурсивним, та має складність , що не є оптимальним, але будь-які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

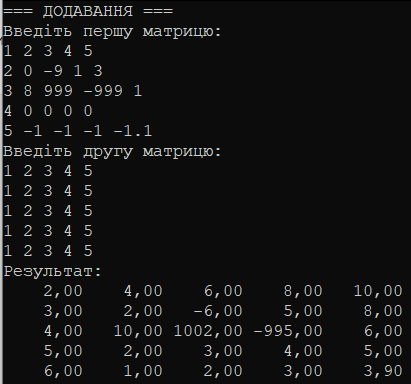
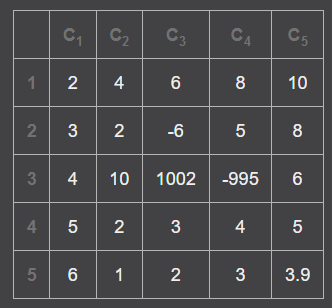
# **3. Висновки**

## **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.

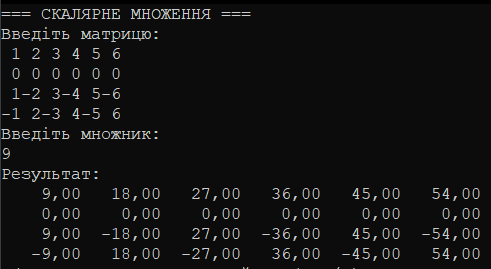
* Додавання\Віднімання

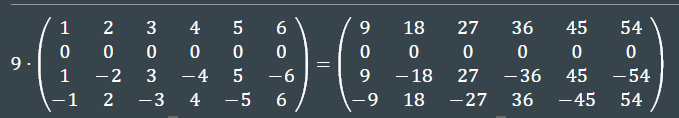




Рисунки 14-16 – Перевірка операції додавання.

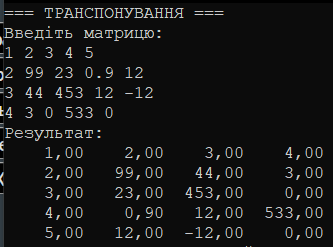
* Скалярне множення\Скалярне ділення

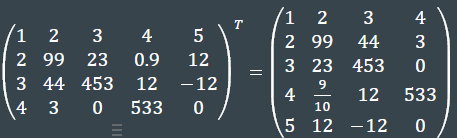




Рисунки 17-18 – Перевірка операції скалярного множення.

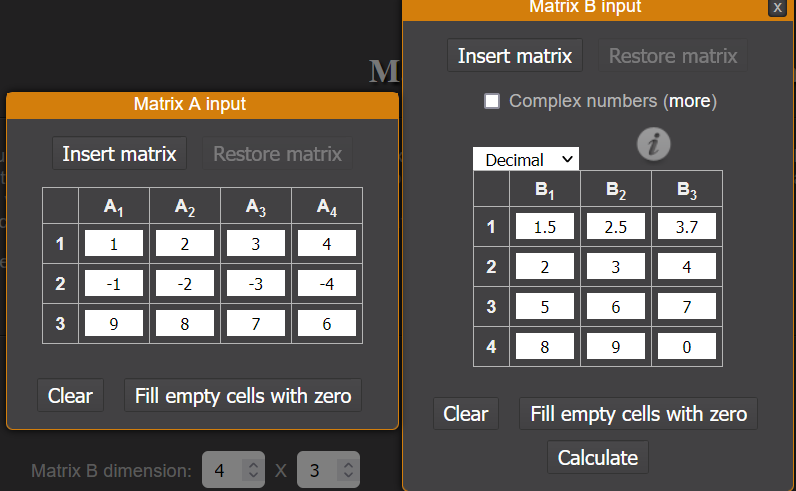
* Транспонування

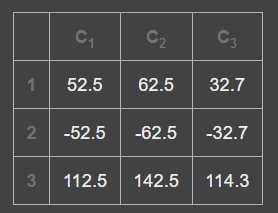
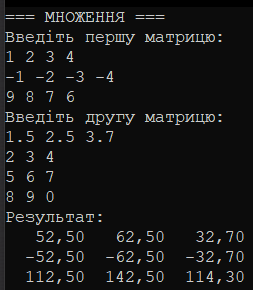




Рисунки 19-20 – Перевірка операції транспонування.

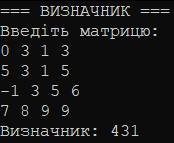
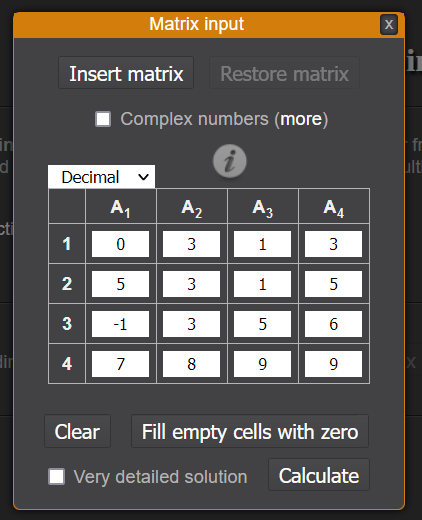
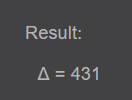
* Множення



‘ 

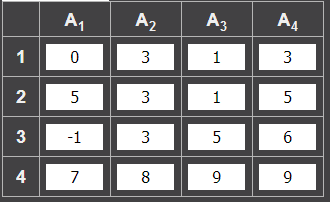
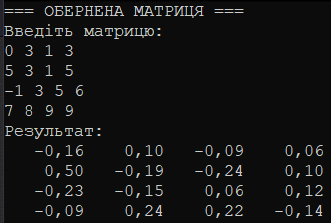
Рисунки 21-23 – Перевірка операції множення.

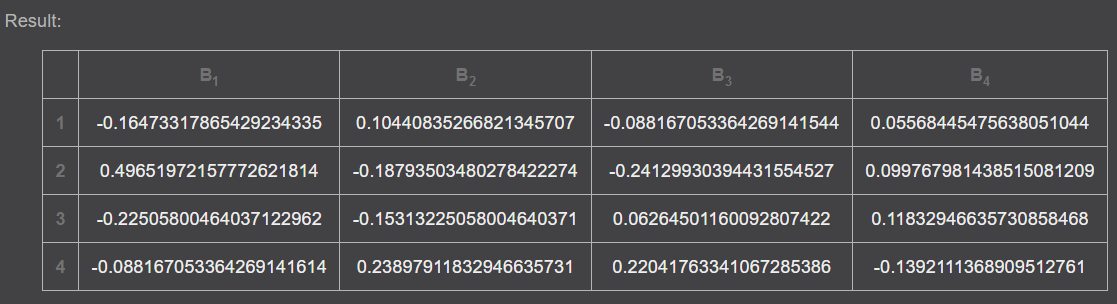
* Знаходження детермінанту

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**