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](#_Toc103235819) 8

[2. Розробка програми](#_Toc103235820) 9

[2.1. Загальні відомості](#_Toc103235821) 9

[2.2. Функціональне призначення](#_Toc103235822) 9

[2.3. Опис логічної структури](#_Toc103235823) 11

[2.4. Максимальне завантаження](#_Toc103235824) 14

[2.5. Вхідні та вихідні дані](#_Toc103235825) 15

[2.6. Керівництво користувачу](#_Toc103235826) 16

[2.7. Керівництво розробнику 2](#_Toc103235827)0

[3. Висновки](#_Toc103235828) 21

[3.1. Перевірка справності програми](#_Toc103235829) 21

[3.2. Висновок](#_Toc103235830) 24

[Додатки](#_Toc103235831) 25

[Література](#_Toc103235832) 25

[Лістинг програми](#_Toc103235833) 26

[Блок-схеми](#_Toc103235834) 48

**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.

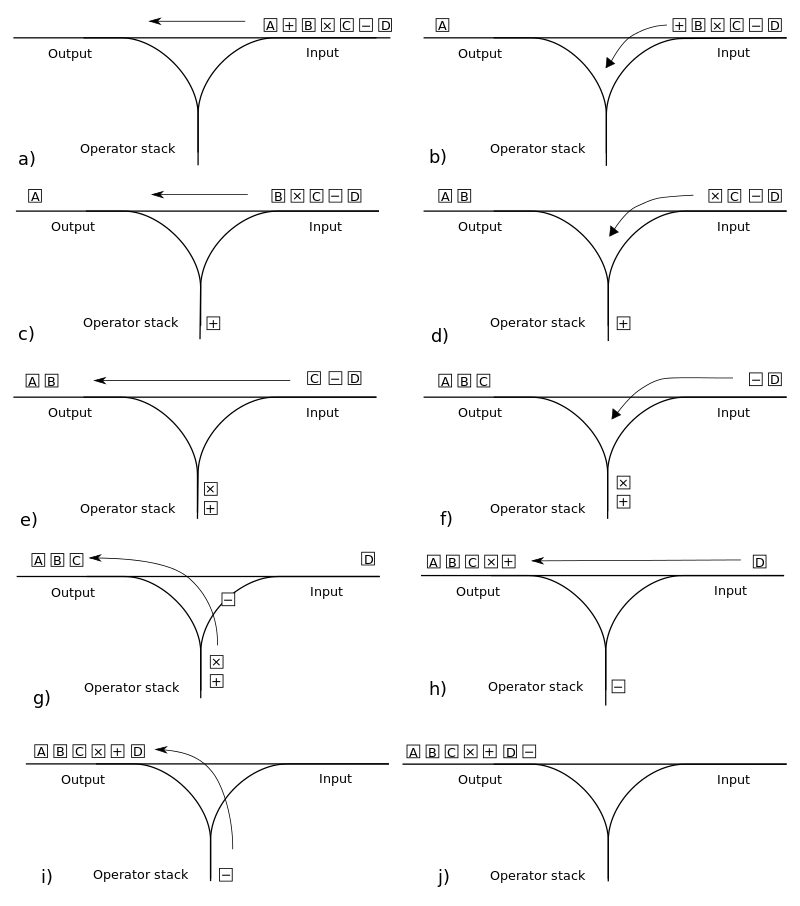


Рисунок 1 – Приклад роботи алгоритму сортвуальної станції

**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 – 20 Мб пам’яті
* Диск – 30 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](#Блок_схема_2) |
| 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](#Блок_схема_3) |
| int getPrecision() | const Token& symbol | Приймає оператор та по-вертає його пріоритетність  [Детальніше](#getPrecision)  [Блок-схема](#Блок_схема_1) |
| queue <Token>RPN() | const vector<Token>& tokens | Приймає набір токенів та перетворює їх з інфіксної у постфіксну запис.  [Детальніше](#RPN)  [Блок-схема](#Блок_схема_4) |
| 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](#Блок_схема_5) |
| 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.

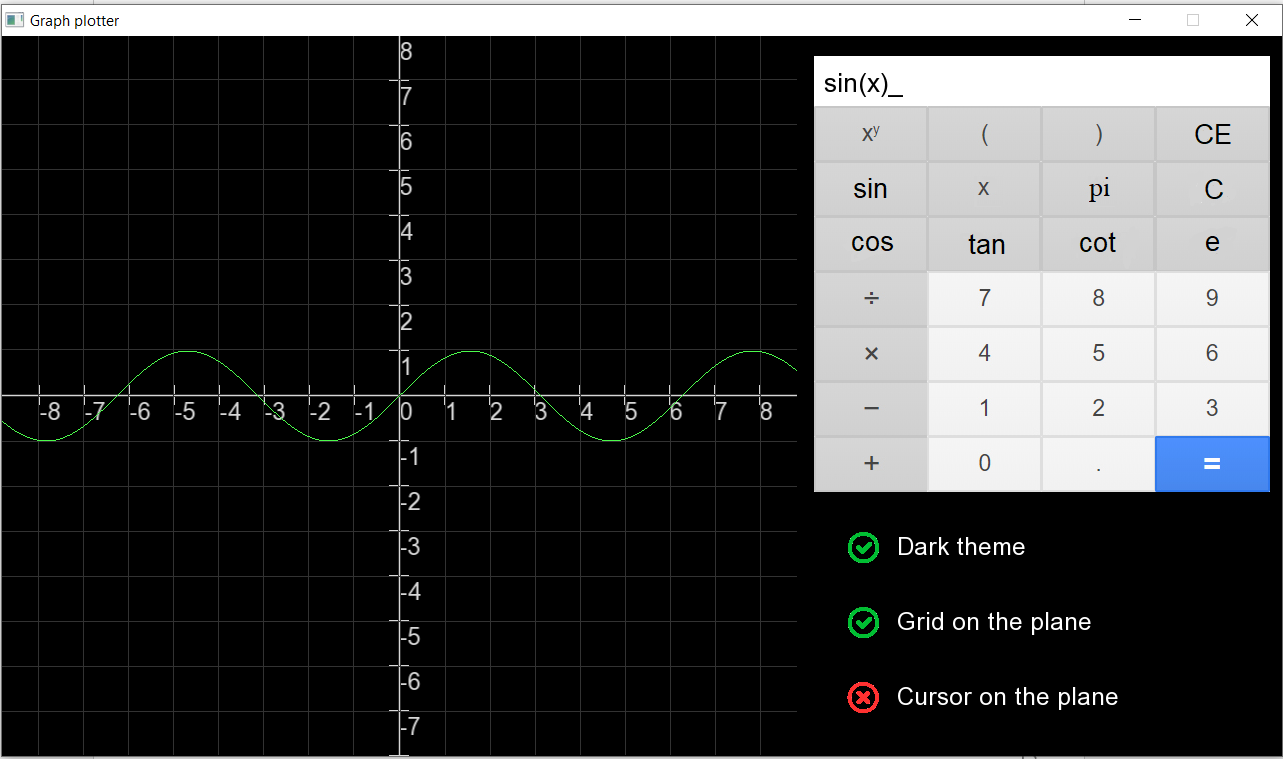


Рисунок 2 – Приклад роботи вікна програми з введеними даними у поле вводу

## **2.6. Керівництво користувачу**

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

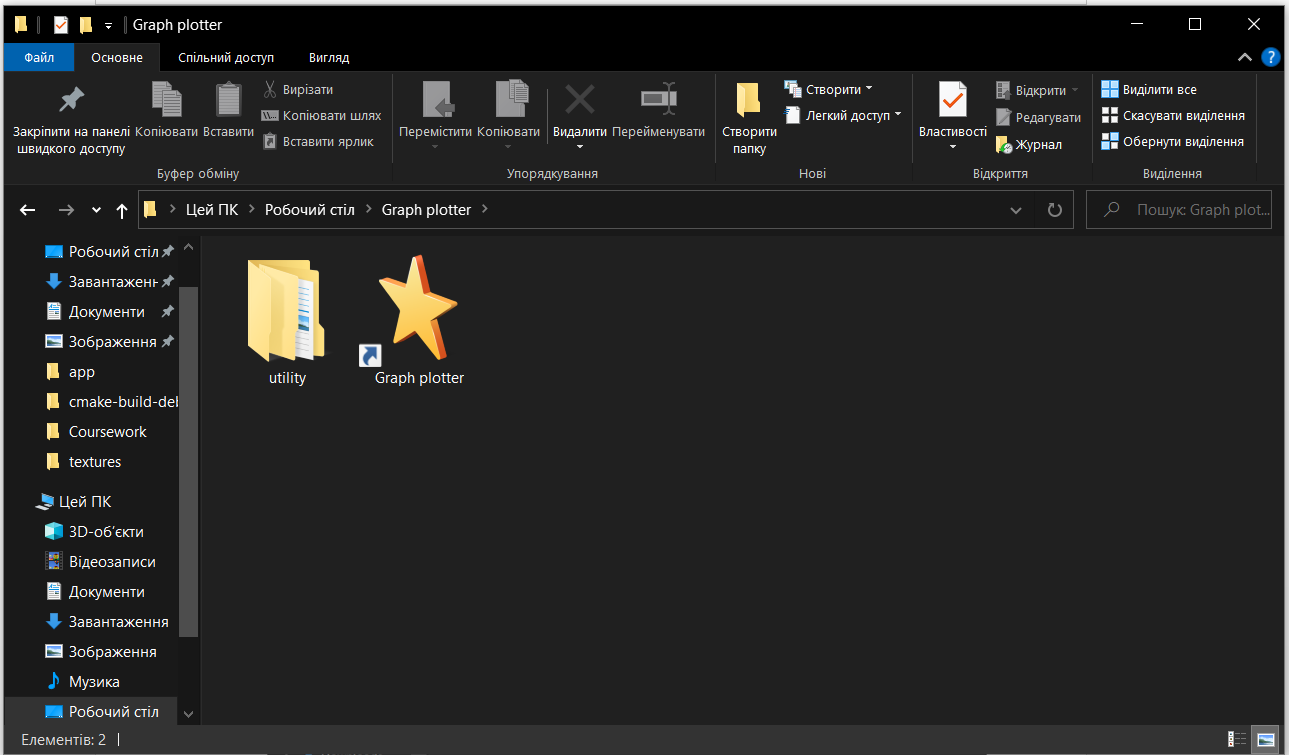
****

Рисунок 3 – 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.

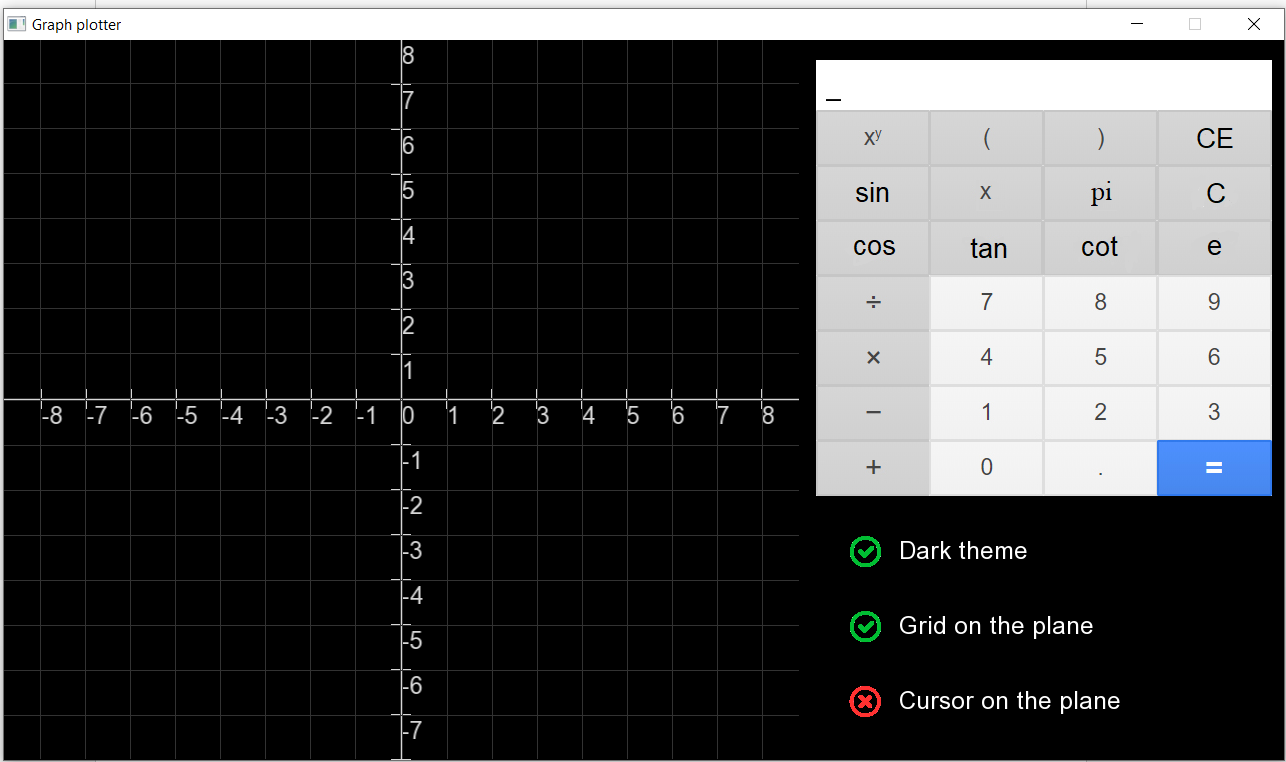


Рисунок 4 – Поле графіку та меню з додатковими кнопками.

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:

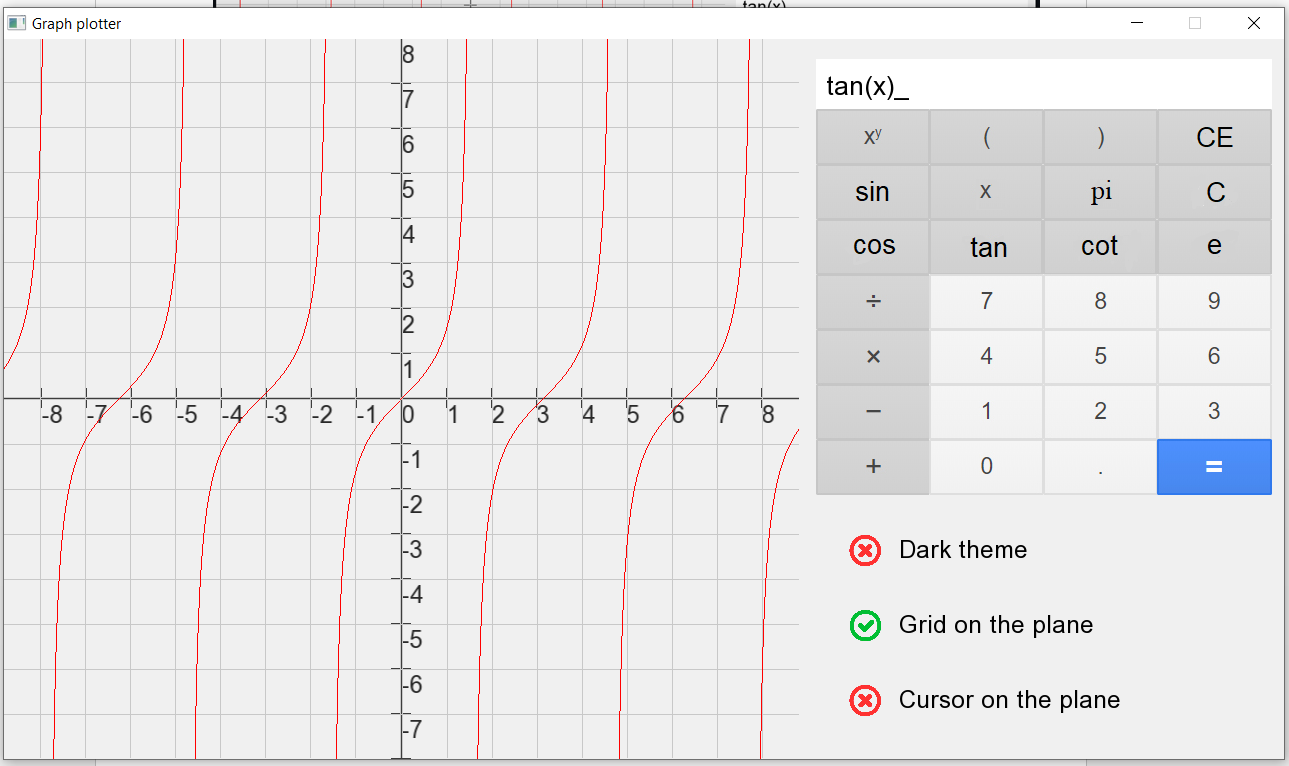


Рисунок 5 – Приклад використання програми №1

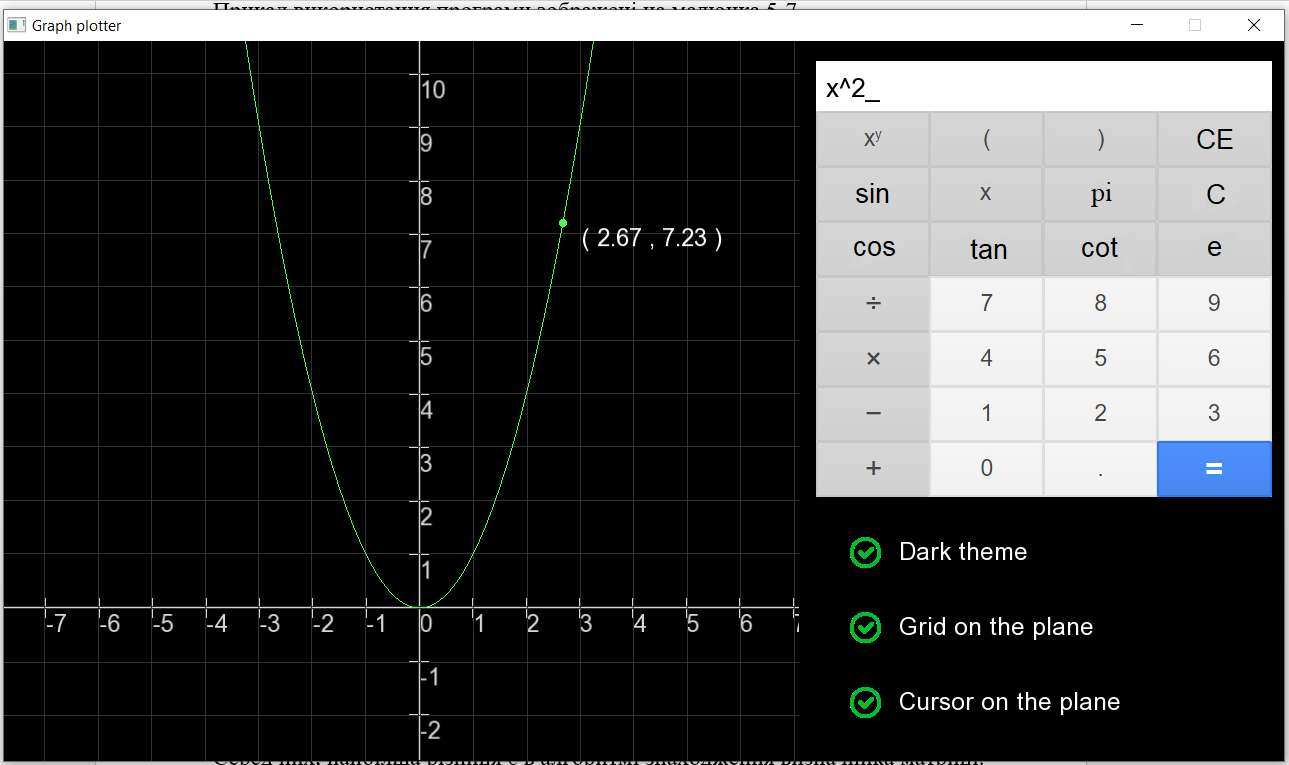


Рисунок 6 – Приклад використання програми №2

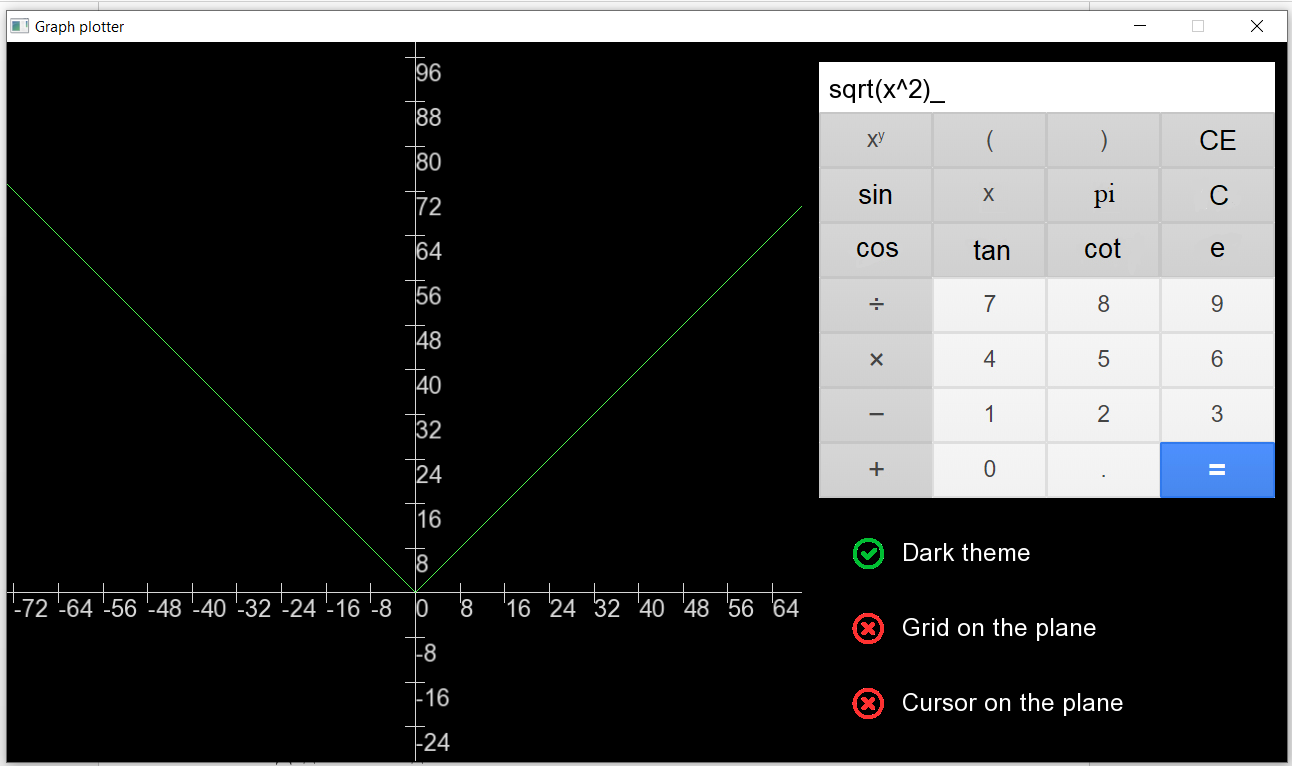


Рисунок 7 – Приклад використання програми №3

## **2.7. Керівництво розробнику**

Під час виконання задачі, мною було виявлено багато альтернативних методів вирішення різних завдань порівняно з тими, які є у фінальній версії програми, наприклад:

* Алгоритм сортувальної станції, а точніше парсер можна було написати найрізноманітнішими методами, але я зупнився на цьому, бо вважаю його найкращим з-поміж усіх
* Велику кількість умовних операторів if можна було б замынити завдяки unordered\_map. Однак, це додало б зайвих бібліотек, хоч і зробило б код охайнішим.
* Для малювання ліній графіку мною був використаний принцип “ланцюгів”.

За цим принципом, кожен елемент 1 поточного кроку у VertexArray дорівнює елементу 0 наступного кроку. Таким чином, не потрібно окремо створювати точки, а потім їх малювати. Я обрав саме цей спосіб, бо він оптимізує процес малювання графіку. Однак, це не єдина можлива реалізація та можливо не найкраща.

# **3. Висновки**

## **3.1. Перевірка справності програми**

В даному розіді буде перевірена справність алгоритмів програми на достатньо складних прикладах, з порівнянням вихідних даних з графічним калькулятором “Desmos”.

* f(x) = sin(cos(x^2))

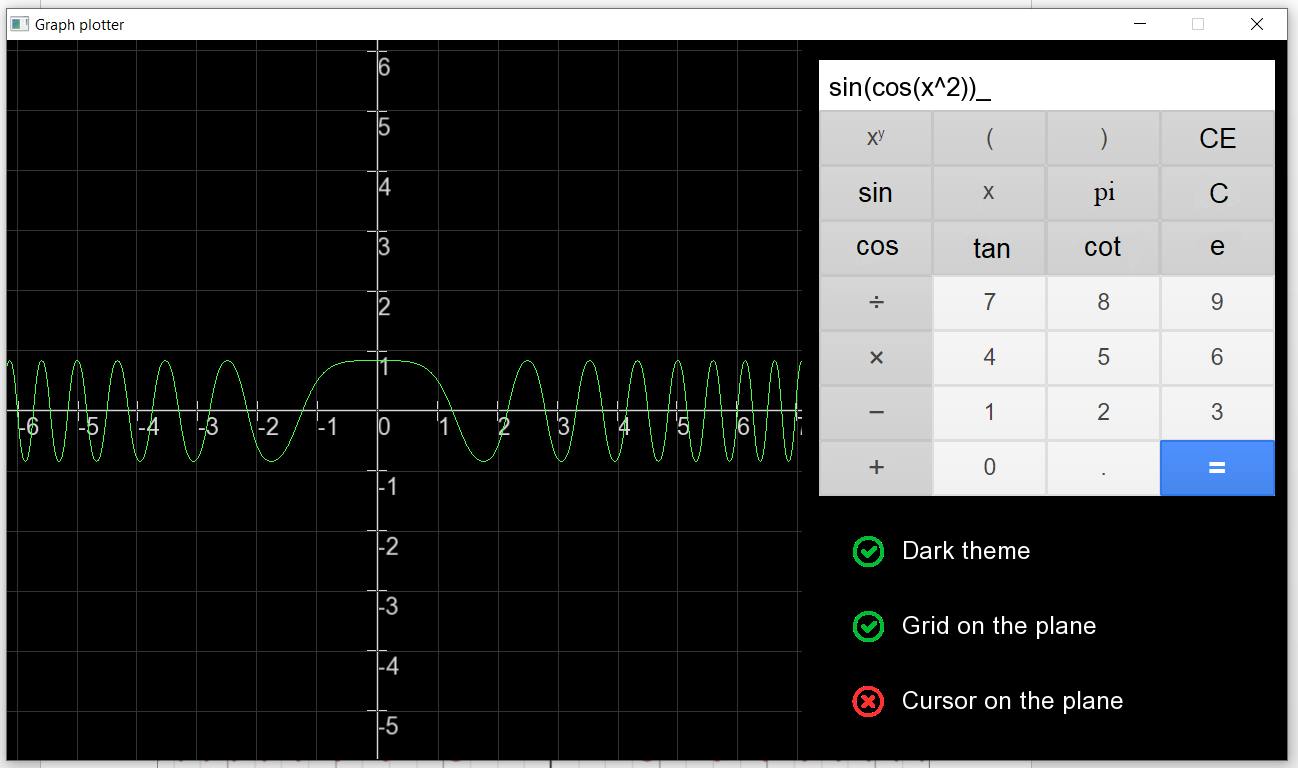


Рисунок 8 – Перевірка правильності виводу графіка.

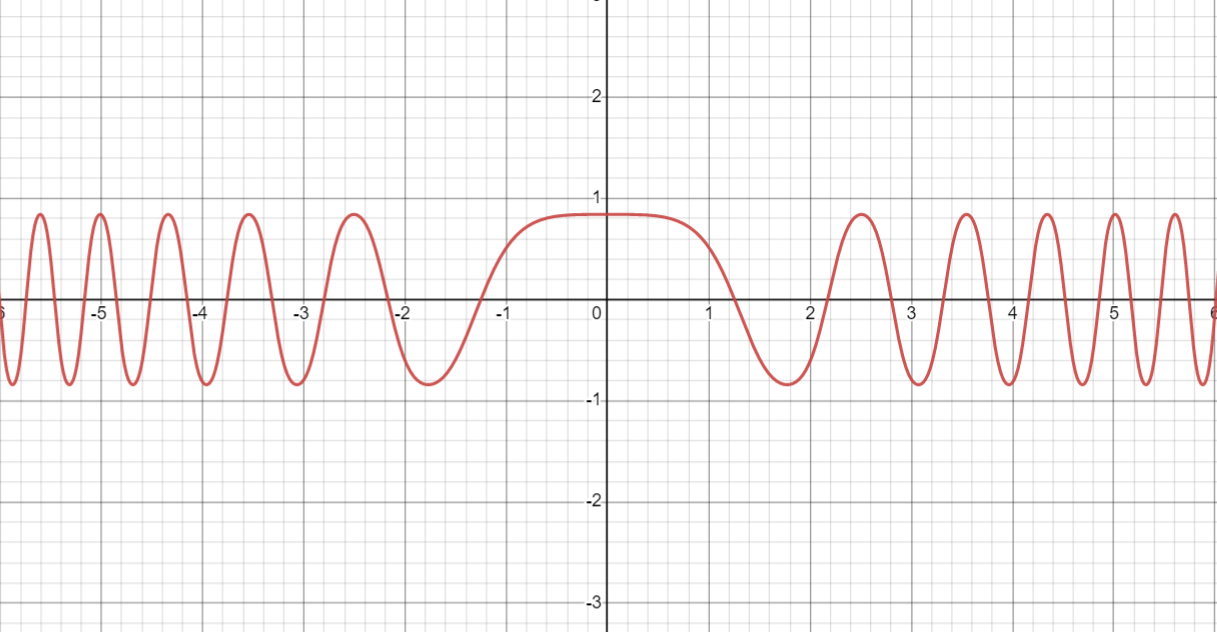


Рисунок 9 – Перевірка правильності виводу графіка.

* f(x) = sqrt(tan(x))\*sin(x)

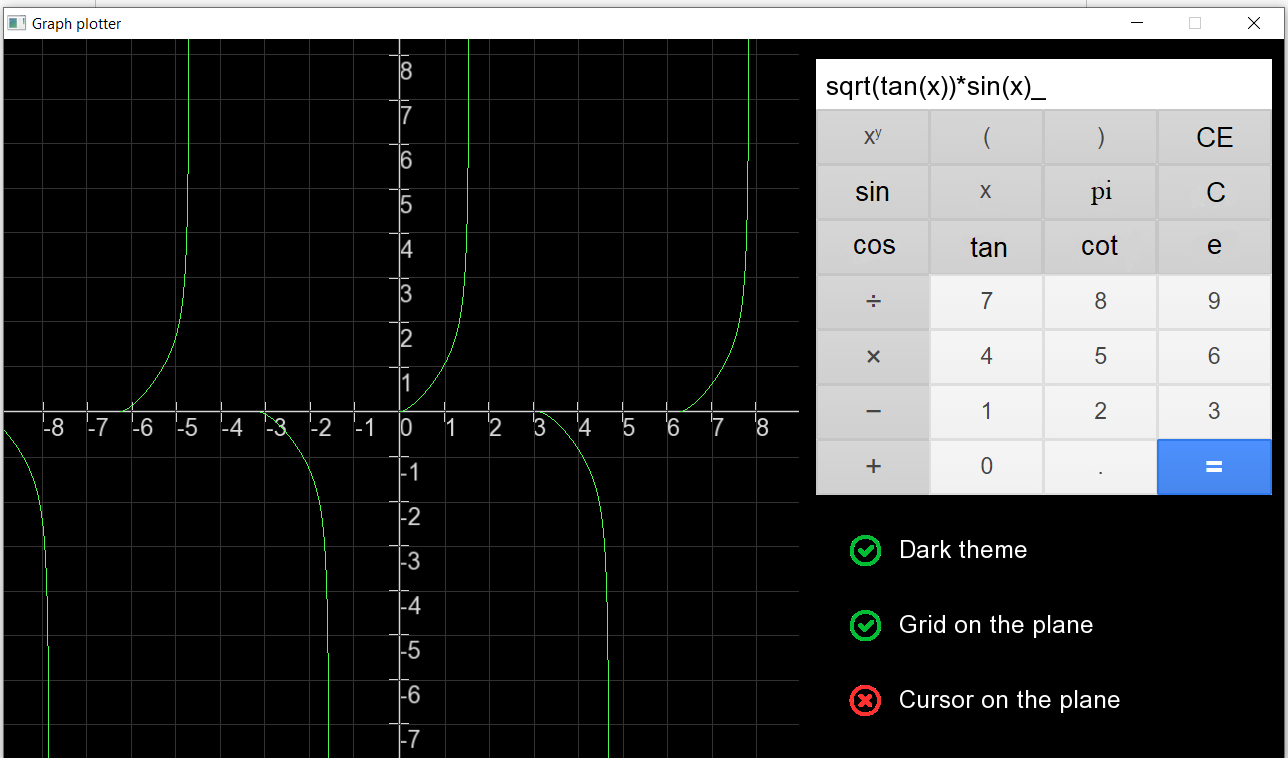


Рисунок 10 – Перевірка правильності виводу графіка.

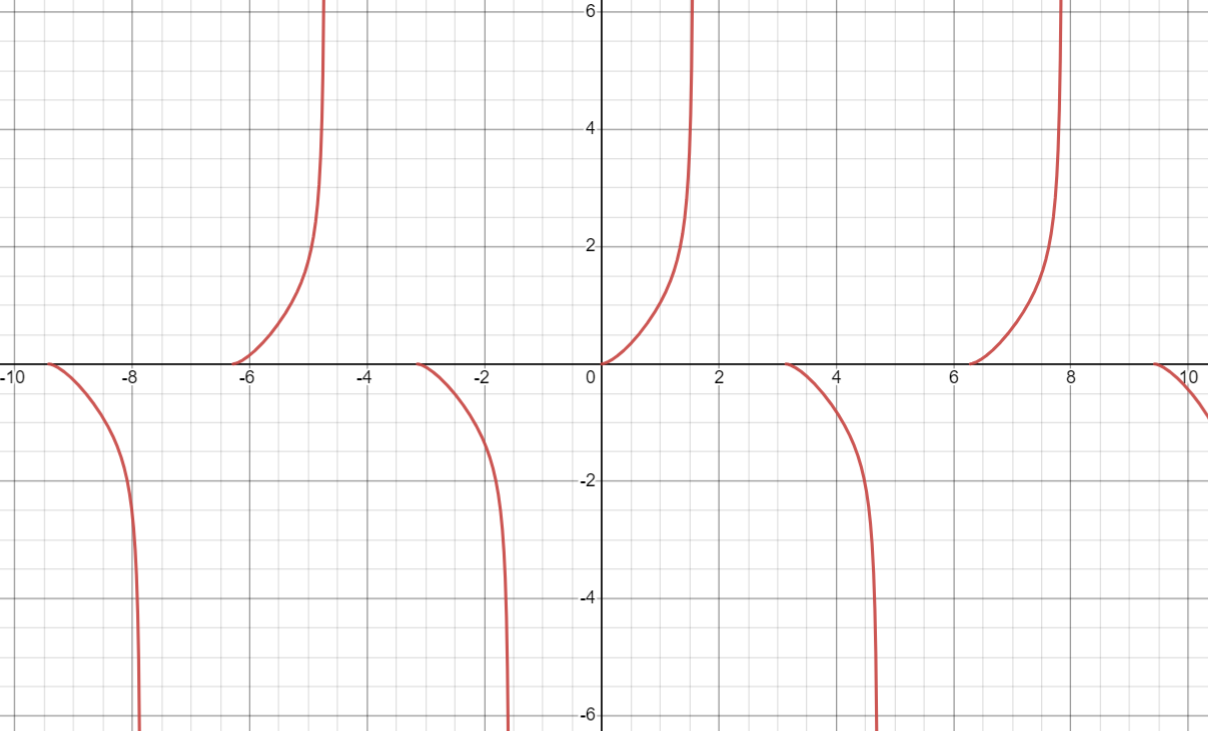


Рисунок 11 – Перевірка правильності виводу графіка.

* f(x) = ln(-2\*x)

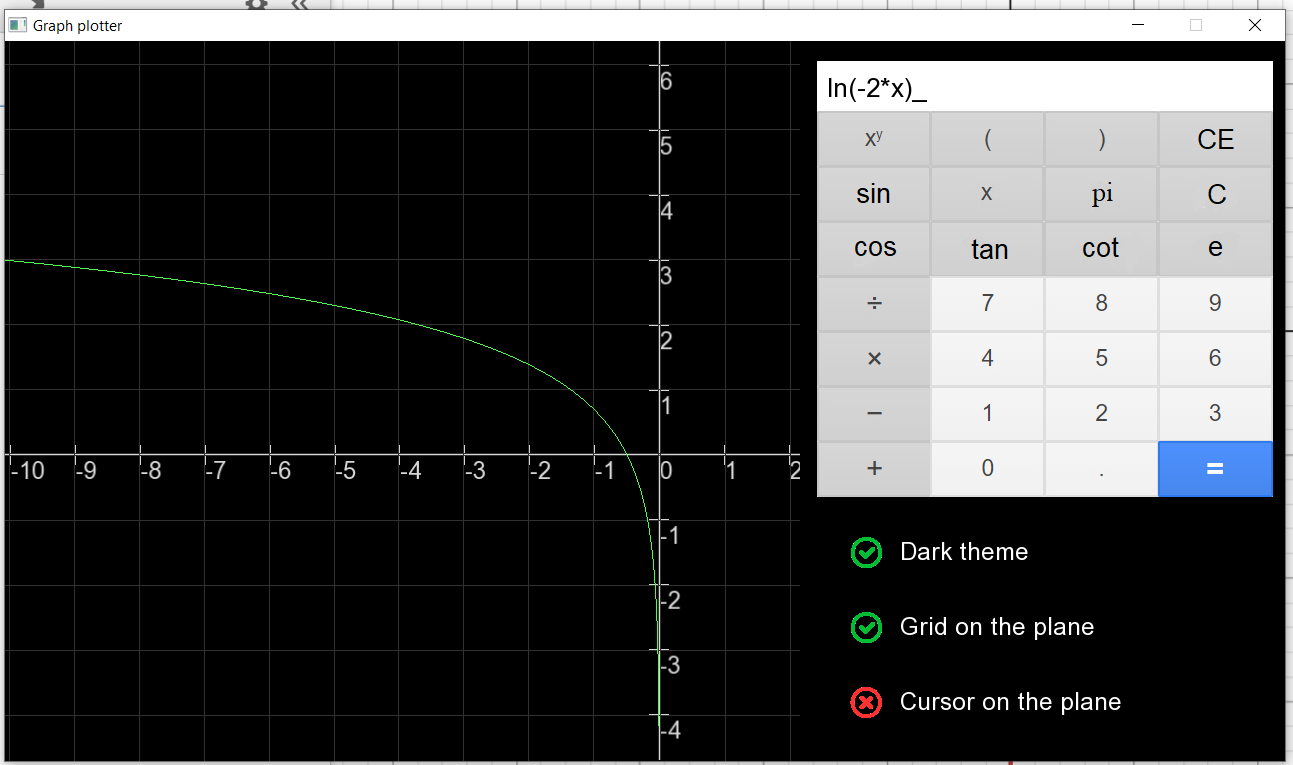


Рисунок 12 – Перевірка правильності виводу графіка.

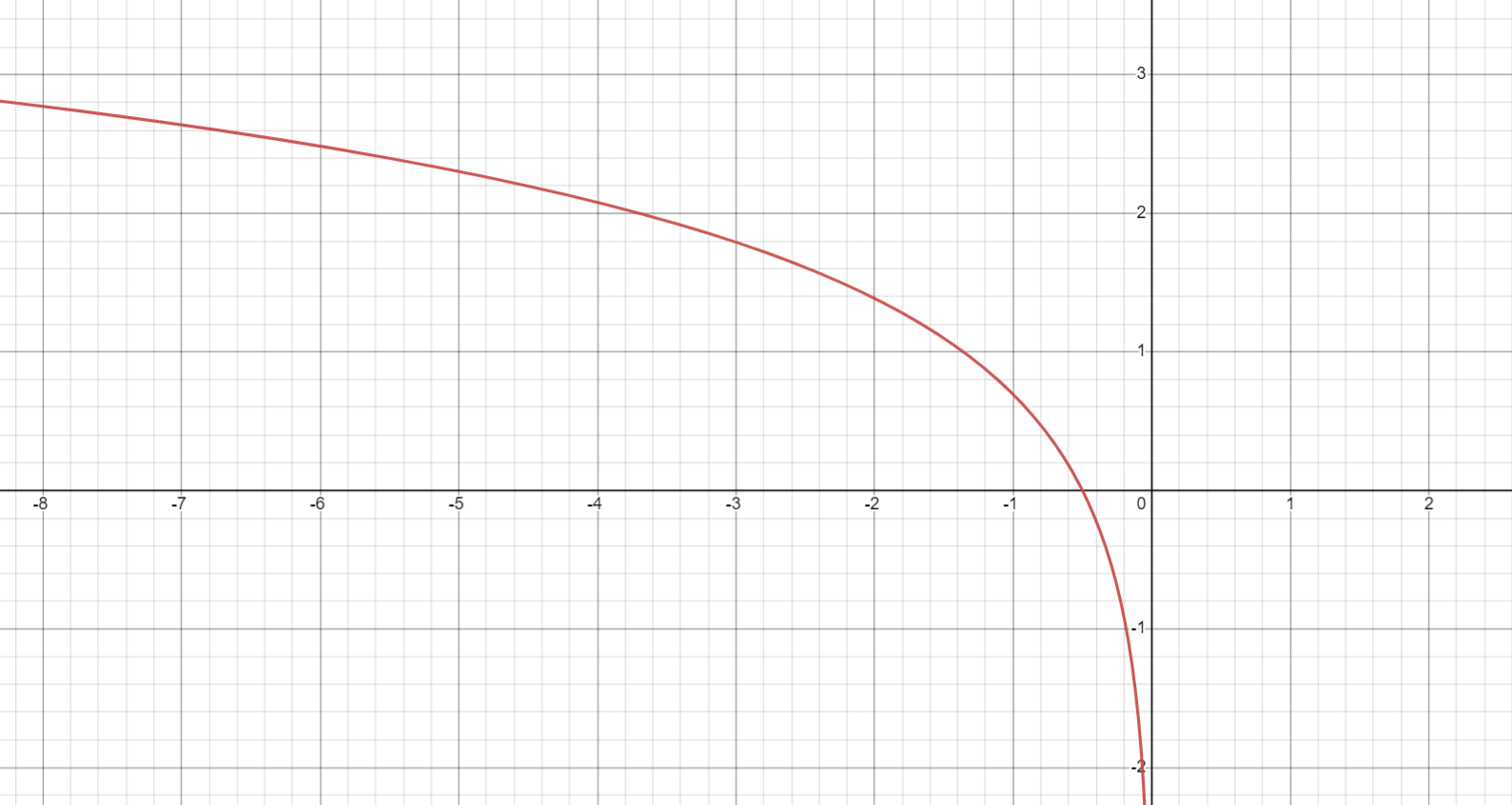


Рисунок 13 – Перевірка правильності виводу графіка.

## **3.2. Висновок**

В результаті виконання курсової роботи була розроблена програма, що дозволяє звичайному користувачу легко вивсети графік потрібної йому функції, дослідити його. Також, мною була вивчена графічна бібліотека SFML, її переваги, недоліки та обмеження.

Програма використовує мінімальну кількість оперативної та графічної пам’яті, що дозволяє запустити її на будь-якому сучасному персональному комп’ютері. Всі результати роботи програми були вірними

В першому розділі цієї записки було розглянено операції, що були імплементовані в програму. В другому, було описана логічна структура програми, використані бібліотеки, керівництва користувачу та розробнику. В третьому розділі було перевірено корректність роботи програми, та підбито підсумки.

# **Додатки**

## **Література**

1. Інтернет-енциклопедія “Вікіпедія”, Reverse Polish notation - <https://en.wikipedia.org/wiki/Reverse_Polish_notation>
2. Інтернет-енциклопедія “Вікіпедія”, Польський інверсний запис - https://uk.wikipedia.org/wiki/Польський\_інверсний\_запис
3. MoHPS, What is RPN? - https://www.hpmuseum.org/rpn.htm
4. Інтернет-енциклопедія “Вікіпедія”, Shunting yard algorithm -
5. Інтернет-енциклопедія “Вікіпедія”, Алгоритм сортувальної станції - https://uk.wikipedia.org/wiki/Алгоритм\_сортувальної\_станції
6. Brilliant.org, Shunting Yard Algorithm - https://brilliant.org/wiki/shunting-yard-algorithm/
7. J. Daniel Dodin, “Ввід: Зворотні польські позначення стали легкими в м’якій обкладинці ”
8. Інтернет-бібліотека “Wikibooks”, Reverse Polish - <https://en.wikibooks.org/wiki/A> level\_Computing/AQA/Paper\_1/Fundamentals\_of\_algorithms/Reverse\_polish
9. GeeksforGeeks, Java Program to Implement Shunting Yard Algorithm - <https://www.geeksforgeeks.org/java-program-to-implement-shunting-yard-algorithm/>
10. SFML, Documentation of SFML 2.5.1 -  [https://www.sfml-dev.org/documentation/2.5.1/l](http://www.graphics.stanford.edu/~seander/bithacks.html)
11. cppreference, Класи та функції - https://en.cppreference.com/w/

## **Лістинг програми**

* main.cpp

#include <SFML/Graphics.hpp>  
#include "textbox.h"  
#include "button.h"  
#include "shunting\_yard.h"  
  
int main() {  
 // defining the variables  
 string equation,cursorPosition;  
 float width = 795, height = 720, renderWidth = 1280, renderHeight = 720,  
 xOriginal = 0, yOriginal = 0, CenterX = (width/2), CenterY = (height/2),Scaler = 45, PanX, PanY;  
 bool isRightHold = false, isGridVisible = true, isThemeBlack = true, isCursorVisible = false;  
  
 // rendering window and apply framerate  
 RenderWindow window;  
 window.create(sf::VideoMode(renderWidth,renderHeight), "Graph plotter", Style::Close);  
 window.setFramerateLimit(60);  
 // creating buttons using class "Button"  
 Button button1("powX.png",{(renderWidth-468),70});  
 Button button2("leftBracket.png",{(renderWidth-355),70});  
 Button button3("rightBracket.png",{(renderWidth-241),70});  
 Button button4("clean.png",{(renderWidth-127),70});  
 Button button5("sin.png",{(renderWidth-468),125});  
 Button button6("x.png",{(renderWidth-355),125});  
 Button button7("pi.png",{(renderWidth-241),125});  
 Button button8("cleanLastChar.png",{(renderWidth-127),125});  
 Button button9("cos.png",{(renderWidth-468),180});  
 Button button10("tan.png",{(renderWidth-355),180});  
 Button button11("cot.png",{(renderWidth-241),180});  
 Button button12("exp.png",{(renderWidth-127),180});  
 Button button13("division.png",{(renderWidth-468),235});  
 Button button14("num7.png",{(renderWidth-355),235});  
 Button button15("num8.png",{(renderWidth-241),235});  
 Button button16("num9.png",{(renderWidth-127),235});  
 Button button17("multiplication.png",{(renderWidth-468),290});  
 Button button18("num4.png",{(renderWidth-355),290});  
 Button button19("num5.png",{(renderWidth-241),290});  
 Button button20("num6.png",{(renderWidth-127),290});  
 Button button21("minus.png",{(renderWidth-468),345});  
 Button button22("num1.png",{(renderWidth-355),345});  
 Button button23("num2.png",{(renderWidth-241),345});  
 Button button24("num3.png",{(renderWidth-127),345});  
 Button button25("plus.png",{(renderWidth-468),400});  
 Button button26("num0.png",{(renderWidth-355),400});  
 Button button27("dot.png",{(renderWidth-241),400});  
 Button button28("result.png",{(renderWidth-127),400});  
 Button button29("check.png",{width+50,height-225});  
 Button button30("check.png",{width+50,height-150});  
 Button button31("cross.png",{width+50,height-75});  
  
 // defining font and text  
 Font font;  
 font.loadFromFile("../fonts/arial.ttf");  
  
 Text text;  
 text.setFont(font);  
 text.setCharacterSize(24);  
  
 // creating objects that will be used for tasks  
 RectangleShape bounds(Vector2f(renderWidth-width, renderHeight));  
 bounds.setPosition({width,0});  
  
 CircleShape Cursor;  
 Cursor.setRadius(4);  
  
 RectangleShape CursorIntersection(Vector2f(1.f, 2\*height));  
 CursorIntersection.setFillColor(Color(255,255,255));  
  
 Textbox textbox(26,Color::Black);  
 textbox.setFont(font);  
 textbox.setPosition({(renderWidth-458),30});  
  
 RectangleShape inputField(Vector2f(456.f, 50.f));  
 inputField.setPosition({(renderWidth-468),20});  
 inputField.setFillColor(Color(255,255,255));  
  
 // creating a vertex array to store the coordinates of points  
 VertexArray Graph(LinesStrip, 2);  
 // main cycle, while application is open  
 while (window.isOpen())  
 {  
 Event event;  
 while (window.pollEvent(event))  
 {  
 switch (event.type) {  
 case Event::Closed: // check for window status  
 window.close();  
 break;  
 case Event::MouseWheelMoved: // check for mouse scrolling  
 if (event.mouseWheel.delta < 0 && Scaler > 6) {  
 Scaler -= Scaler \* 0.1;  
 }  
 if (event.mouseWheel.delta > 0 && Scaler < 300) {  
 Scaler += Scaler \* 0.1;  
 }  
 break;  
 case::Event::KeyPressed: // check for keyboard status  
 if (event.key.code == Keyboard::Enter) {  
 equation = textbox.getText();  
 }  
 if (event.key.code == Keyboard::Escape) {  
 Scaler = 45;  
 CenterX = width/2;  
 CenterY = height/2;  
 }  
 if (event.key.code == sf::Keyboard::Up) {  
 if(Scaler == -1 && event.mouseWheel.delta >=1)  
 Scaler = 1;  
 Scaler+=Scaler\*0.1f;  
 }  
 if (event.key.code == sf::Keyboard::Down) {  
 if(Scaler == 1 && event.mouseWheel.delta <=-1)  
 Scaler = -1;  
 Scaler-=Scaler\*0.1f;  
 }  
 case Event::MouseButtonPressed: // check for mouse buttons status  
 if (event.mouseButton.button == Mouse::Right) {  
 xOriginal = event.mouseButton.x;  
 yOriginal = event.mouseButton.y;  
 isRightHold = true;  
 }  
 if (event.mouseButton.button == Mouse::Left) {  
 Vector2i mousePos = Mouse::getPosition( window );  
 Vector2f mousePosF( static\_cast<float>( mousePos.x ), static\_cast<float>( mousePos.y ) );  
 if ( button1.getGlobalBounds().contains( mousePosF )) {  
 textbox.typedButtonMenu("^");  
 }  
 if ( button2.getGlobalBounds().contains( mousePosF )) {  
 textbox.typedButtonMenu("(");  
 }  
 if ( button3.getGlobalBounds().contains( mousePosF )) {  
 textbox.typedButtonMenu(")");  
 }  
 if ( button4.getGlobalBounds().contains( mousePosF )) {  
 textbox.deleteString();  
 }  
 if ( button5.getGlobalBounds().contains( mousePosF )) {  
 textbox.typedButtonMenu("sin");  
 }  
 if ( button6.getGlobalBounds().contains( mousePosF )) {  
 textbox.typedButtonMenu("x");  
 }  
 if ( button7.getGlobalBounds().contains( mousePosF )) {  
 textbox.typedButtonMenu("pi");  
 }  
 if ( button8.getGlobalBounds().contains( mousePosF )) {  
 textbox.deleteChar();  
 }  
 if ( button9.getGlobalBounds().contains( mousePosF )) {  
 textbox.typedButtonMenu("cos");  
 }  
 if ( button10.getGlobalBounds().contains( mousePosF )) {  
 textbox.typedButtonMenu("tan");  
 }  
 if ( button11.getGlobalBounds().contains( mousePosF )) {  
 textbox.typedButtonMenu("cot");  
 }  
 if ( button12.getGlobalBounds().contains( mousePosF )) {  
 textbox.typedButtonMenu("e");  
 }  
 if ( button13.getGlobalBounds().contains( mousePosF )) {  
 textbox.typedButtonMenu("/");  
 }  
 if ( button14.getGlobalBounds().contains( mousePosF )) {  
 textbox.typedButtonMenu("7");  
 }  
 if ( button15.getGlobalBounds().contains( mousePosF )) {  
 textbox.typedButtonMenu("8");  
 }  
 if ( button16.getGlobalBounds().contains( mousePosF )) {  
 textbox.typedButtonMenu("9");  
 }  
 if ( button17.getGlobalBounds().contains( mousePosF )) {  
 textbox.typedButtonMenu("\*");  
 }  
 if ( button18.getGlobalBounds().contains( mousePosF )) {  
 textbox.typedButtonMenu("4");  
 }  
 if ( button19.getGlobalBounds().contains( mousePosF )) {  
 textbox.typedButtonMenu("5");  
 }  
 if ( button20.getGlobalBounds().contains( mousePosF )) {  
 textbox.typedButtonMenu("6");  
 }  
 if ( button21.getGlobalBounds().contains( mousePosF )) {  
 textbox.typedButtonMenu("-");  
 }  
 if ( button22.getGlobalBounds().contains( mousePosF )) {  
 textbox.typedButtonMenu("1");  
 }  
 if ( button23.getGlobalBounds().contains( mousePosF )) {  
 textbox.typedButtonMenu("2");  
 }  
 if ( button24.getGlobalBounds().contains( mousePosF )) {  
 textbox.typedButtonMenu("3");  
 }  
 if ( button25.getGlobalBounds().contains( mousePosF )) {  
 textbox.typedButtonMenu("+");  
 }  
 if ( button26.getGlobalBounds().contains( mousePosF )) {  
 textbox.typedButtonMenu("0");  
 }  
 if ( button27.getGlobalBounds().contains( mousePosF )) {  
 textbox.typedButtonMenu(".");  
 }  
 if ( button28.getGlobalBounds().contains( mousePosF )) {  
 equation = textbox.getText();  
 }  
  
 if ( button29.getGlobalBounds().contains( mousePosF )) {  
 if (isThemeBlack) {  
 button29.setTexture("cross.png");  
 isThemeBlack = false;  
 }  
 else {  
 button29.setTexture("check.png");  
 isThemeBlack = true;  
 }  
 }  
 if ( button30.getGlobalBounds().contains( mousePosF )) {  
 if (isGridVisible) {  
 button30.setTexture("cross.png");  
 isGridVisible = false;  
 }  
 else {  
 button30.setTexture("check.png");  
 isGridVisible = true;  
 }  
 }  
 if ( button31.getGlobalBounds().contains( mousePosF )) {  
 if (isCursorVisible) {  
 button31.setTexture("cross.png");  
 isCursorVisible = false;  
 }  
 else {  
 button31.setTexture("check.png");  
 isCursorVisible = true;  
 }  
 }  
 }  
 break;  
 {case Event::MouseMoved: // check for mouse position  
  
 if(isRightHold) {  
 PanX = event.mouseMove.x - xOriginal;  
 PanY = event.mouseMove.y - yOriginal;  
 CenterX+=PanX;  
 CenterY+=PanY;  
 xOriginal = event.mouseMove.x;  
 yOriginal = event.mouseMove.y;  
 }  
 Vector2i mousePos = Mouse::getPosition( window );  
 Vector2f mousePosF( static\_cast<float>( mousePos.x ), static\_cast<float>( mousePos.y ) );  
  
 Vector2f CursorIntersectionPos( static\_cast<float>( mousePos.x ), static\_cast<float>( mousePos.y - height) );  
 CursorIntersection.setPosition(CursorIntersectionPos);  
  
 if ( button1.getGlobalBounds().contains( mousePosF )) {  
 button1.setColor( sf::Color( 183, 183, 185 ) );  
 } else button1.setColor( sf::Color( 255, 255, 255 ) );  
 if ( button2.getGlobalBounds().contains( mousePosF )) {  
 button2.setColor( sf::Color( 183, 183, 185 ) );  
 } else button2.setColor( sf::Color( 255, 255, 255 ) );  
 if ( button3.getGlobalBounds().contains( mousePosF )) {  
 button3.setColor( sf::Color( 183, 183, 185 ) );  
 } else button3.setColor( sf::Color( 255, 255, 255 ) );  
 if ( button4.getGlobalBounds().contains( mousePosF )) {  
 button4.setColor( sf::Color( 248, 125, 125 ) );  
 } else button4.setColor( sf::Color( 255, 255, 255 ) );  
 if ( button5.getGlobalBounds().contains( mousePosF )) {  
 button5.setColor( sf::Color( 183, 183, 185 ) );  
 } else button5.setColor( sf::Color( 255, 255, 255 ) );  
 if ( button6.getGlobalBounds().contains( mousePosF )) {  
 button6.setColor( sf::Color( 183, 183, 185 ) );  
 } else button6.setColor( sf::Color( 255, 255, 255 ) );  
 if ( button7.getGlobalBounds().contains( mousePosF )) {  
 button7.setColor( sf::Color( 183, 183, 185 ) );  
 } else button7.setColor( sf::Color( 255, 255, 255 ) );  
 if ( button8.getGlobalBounds().contains( mousePosF )) {  
 button8.setColor( sf::Color( 248, 125, 125 ) );  
 } else button8.setColor( sf::Color( 255, 255, 255 ) );  
 if ( button9.getGlobalBounds().contains( mousePosF )) {  
 button9.setColor( sf::Color( 183, 183, 185 ) );  
 } else button9.setColor( sf::Color( 255, 255, 255 ) );  
 if ( button10.getGlobalBounds().contains( mousePosF )) {  
 button10.setColor( sf::Color( 183, 183, 185 ) );  
 } else button10.setColor( sf::Color( 255, 255, 255 ) );  
 if ( button11.getGlobalBounds().contains( mousePosF )) {  
 button11.setColor( sf::Color( 183, 183, 185 ) );  
 } else button11.setColor( sf::Color( 255, 255, 255 ) );  
 if ( button12.getGlobalBounds().contains( mousePosF )) {  
 button12.setColor( sf::Color( 183, 183, 185 ) );  
 } else button12.setColor( sf::Color( 255, 255, 255 ) );  
 if ( button13.getGlobalBounds().contains( mousePosF )) {  
 button13.setColor( sf::Color( 183, 183, 185 ) );  
 } else button13.setColor( sf::Color( 255, 255, 255 ) );  
 if ( button14.getGlobalBounds().contains( mousePosF )) {  
 button14.setColor( sf::Color( 183, 183, 185 ) );  
 } else button14.setColor( sf::Color( 255, 255, 255 ) );  
 if ( button15.getGlobalBounds().contains( mousePosF )) {  
 button15.setColor( sf::Color( 183, 183, 185 ) );  
 } else button15.setColor( sf::Color( 255, 255, 255 ) );  
 if ( button16.getGlobalBounds().contains( mousePosF )) {  
 button16.setColor( sf::Color( 183, 183, 185 ) );  
 } else button16.setColor( sf::Color( 255, 255, 255 ) );  
 if ( button17.getGlobalBounds().contains( mousePosF )) {  
 button17.setColor( sf::Color( 183, 183, 185 ) );  
 } else button17.setColor( sf::Color( 255, 255, 255 ) );  
 if ( button18.getGlobalBounds().contains( mousePosF )) {  
 button18.setColor( sf::Color( 183, 183, 185 ) );  
 } else button18.setColor( sf::Color( 255, 255, 255 ) );  
 if ( button19.getGlobalBounds().contains( mousePosF )) {  
 button19.setColor( sf::Color( 183, 183, 185 ) );  
 } else button19.setColor( sf::Color( 255, 255, 255 ) );  
 if ( button20.getGlobalBounds().contains( mousePosF )) {  
 button20.setColor( sf::Color( 183, 183, 185 ) );  
 } else button20.setColor( sf::Color( 255, 255, 255 ) );  
 if ( button21.getGlobalBounds().contains( mousePosF )) {  
 button21.setColor( sf::Color( 183, 183, 185 ) );  
 } else button21.setColor( sf::Color( 255, 255, 255 ) );  
 if ( button22.getGlobalBounds().contains( mousePosF )) {  
 button22.setColor( sf::Color( 183, 183, 185 ) );  
 } else button22.setColor( sf::Color( 255, 255, 255 ) );  
 if ( button23.getGlobalBounds().contains( mousePosF )) {  
 button23.setColor( sf::Color( 183, 183, 185 ) );  
 } else button23.setColor( sf::Color( 255, 255, 255 ) );  
 if ( button24.getGlobalBounds().contains( mousePosF )) {  
 button24.setColor( sf::Color( 183, 183, 185 ) );  
 } else button24.setColor( sf::Color( 255, 255, 255 ) );  
 if ( button25.getGlobalBounds().contains( mousePosF )) {  
 button25.setColor( sf::Color( 183, 183, 185 ) );  
 } else button25.setColor( sf::Color( 255, 255, 255 ) );  
 if ( button26.getGlobalBounds().contains( mousePosF )) {  
 button26.setColor( sf::Color( 183, 183, 185 ) );  
 } else button26.setColor( sf::Color( 255, 255, 255 ) );  
 if ( button27.getGlobalBounds().contains( mousePosF )) {  
 button27.setColor( sf::Color( 183, 183, 185 ) );  
 } else button27.setColor( sf::Color( 255, 255, 255 ) );  
 if ( button28.getGlobalBounds().contains( mousePosF )) {  
 button28.setColor( sf::Color( 64, 127, 246 ) );  
 } else button28.setColor( sf::Color( 255, 255, 255 ) );  
 break;}  
 case Event::MouseButtonReleased: // check if mouse released (right click)  
 if (event.mouseButton.button == Mouse::Right) isRightHold = false;  
 break;  
 case Event::TextEntered:// watching for the keyboard input  
 textbox.typedButtonKeyboard(event);  
 }  
 }  
  
 // checks bool value and apply colors for background after clearing it  
 if (!isThemeBlack) window.clear(Color(240,240,240));  
 else window.clear(Color::Black);  
  
 // creates additional scaler for scaling numbers and lines on the  
 float FScaler = Scaler;  
 while ( width / FScaler > 20) {  
 FScaler \*= 2;  
 }  
  
  
 VertexArray Line(LinesStrip, 2);  
  
 // checks bool value and apply colors for lines of the grid  
 if (isThemeBlack) {  
 Line[0].color = {215, 215, 215};  
 Line[1].color = {215, 215, 215};  
 text.setFillColor({215, 215, 215});  
 }  
 else {  
 Line[0].color = {60, 60, 60};  
 Line[1].color = {60, 60, 60};  
 text.setFillColor({40, 40, 40});  
 }  
  
 // creates x and y axis  
 Line[0].position = Vector2f(CenterX, (0));  
 Line[1].position = Vector2f(CenterX, (height));  
 window.draw(Line);  
 Line[0].position = Vector2f((0), CenterY);  
 Line[1].position = Vector2f((width), CenterY);  
 window.draw(Line);  
  
 // creates the dimension (numbers) on the axes  
 for(float i = 0; i<=width/2+(-1\*(CenterX-width/2)); i+=FScaler)  
 {  
 text.setString(intToString(i/Scaler));  
 text.setPosition((i+CenterX),CenterY);  
 window.draw(text);  
 }  
 for(float i = 0; i>=-width/2+(-1\*(CenterX-width/2)); i-=FScaler)  
 {  
 text.setString(intToString(i/Scaler));  
 text.setPosition((i+CenterX),CenterY);  
 if (i != 0) {  
 window.draw(text);  
 }  
 }  
 for(float i = 0; i<=height/2+(CenterY-height/2); i+=FScaler)  
 {  
 text.setString(intToString(i/Scaler));  
 text.setPosition(CenterX,CenterY-i);  
 if (i != 0) {  
 window.draw(text);  
 }  
 }  
 for(float i = 0; i>=-height/2+(CenterY-height/2); i-=FScaler)  
 {  
 text.setString(intToString(i/Scaler));  
 text.setPosition(CenterX,CenterY-i);  
 if (i != 0) {  
 window.draw(text);  
 }  
 }  
  
 // create lines on the axes  
 for(float i = 0; i <= width/2+(CenterX-width/2); i+=FScaler)  
 {  
 Line[0].position = Vector2f(CenterX-i, CenterY-(10));  
 Line[1].position = Vector2f(CenterX-i, CenterY+(10));  
 window.draw(Line);  
 }  
 for(float i = 0; i <= width/2+(-1\*(CenterX-width/2)); i+=FScaler)  
 {  
 Line[0].position = Vector2f(CenterX+i, CenterY-(10));  
 Line[1].position = Vector2f(CenterX+i, CenterY+(10));  
 window.draw(Line);  
 }  
 for(float i = 0; i <= height/2+(-1\*(CenterY-height/2)); i+=FScaler)  
 {  
 Line[0].position = Vector2f(CenterX-(10),CenterY+i);  
 Line[1].position = Vector2f(CenterX+(10),CenterY+i);  
 window.draw(Line);  
 }  
 for(float i = 0; i <= height/2+((CenterY-height/2)); i+=FScaler)  
 {  
 Line[0].position = Vector2f(CenterX-(10),CenterY-i);  
 Line[1].position = Vector2f(CenterX+(10),CenterY-i);  
 window.draw(Line);  
 }  
  
 if (isGridVisible) {  
 // checks bool value and apply colors for lines of the grid  
 if (isThemeBlack) {  
 Line[0].color = {50, 50, 50};  
 Line[1].color = {50, 50, 50};  
 }  
 else {  
 Line[0].color = {200, 200, 200};  
 Line[1].color = {200, 200, 200};  
 };  
  
 // creates the horizontal and vertical lines of the grid  
 for(float i = 1; i <= width/2+(CenterX-width/2); i+=FScaler)  
 {  
 Line[0].position = Vector2f(CenterX-i, 0);  
 Line[1].position = Vector2f(CenterX-i, height);  
 window.draw(Line);  
 }  
 for(float i = 1; i <= width/2+(-1\*(CenterX-width/2)); i+=FScaler)  
 {  
 Line[0].position = Vector2f(CenterX+i, 0);  
 Line[1].position = Vector2f(CenterX+i, height);  
 window.draw(Line);  
 }  
 for(float i = 1; i <= height/2+(-1\*(CenterY-height/2)); i+=FScaler)  
 {  
 Line[0].position = Vector2f(0,CenterY+i);  
 Line[1].position = Vector2f(width,CenterY+i);  
 window.draw(Line);  
 }  
 for(float i = 1; i <= height/2+((CenterY-height/2)); i+=FScaler)  
 {  
 Line[0].position = Vector2f(0,CenterY-i);  
 Line[1].position = Vector2f(width,CenterY-i);  
 window.draw(Line);  
 }  
 }  
  
 // checks bool value and apply colors for graph and dot with coordinates (x,y)  
 if (isThemeBlack) {  
 Graph[0].color = {75, 251, 75};  
 Graph[1].color = {75, 251, 75};  
 Cursor.setFillColor({75, 251, 75});  
 text.setFillColor({255, 255, 255});  
 }  
 else {  
 Graph[0].color = Color::Red;  
 Graph[1].color = Color::Red;  
 Cursor.setFillColor(Color::Red);  
 text.setFillColor({0, 0, 0});  
 }  
  
 float lastPositive = (width/2+(-1\*(CenterX-width/2)))/Scaler,  
 lastNegative = (-width/2+(-1\*(CenterX-width/2)))/Scaler;  
  
 // if equation is not empty - continue  
 if (!equation.empty()) {  
 for(float x = lastNegative; x < lastPositive; x += 1/Scaler) // 1/Scaler is responsible for the number of lines per unit  
 {  
 // Creating points that will be connected in each cycle step and draw them  
 Graph[0].position = Vector2f((x\*Scaler+CenterX),(evaluate(equation,x)\*Scaler\*-1+CenterY));  
 Graph[1].position = Vector2f(((x+1/Scaler)\*Scaler+CenterX),(evaluate(equation,x+1/Scaler)\*Scaler\*-1+CenterY));  
 if (fabs(Graph[1].position.y) < 1000) {  
 window.draw(Graph);  
 }  
  
 // checks bool value of drawing coordinates on the plot based on mouse position  
 if (isCursorVisible) {  
 if ( CursorIntersection.getGlobalBounds().contains(Graph[0].position) ) {  
 Vector2i mousePos = Mouse::getPosition( window );  
 Vector2f textPos( static\_cast<float>( mousePos.x + 20 ), static\_cast<float>( (Graph[0].position.y) ) );  
 Vector2f cursorPos( static\_cast<float>( mousePos.x - 3 ), static\_cast<float>( (Graph[0].position.y) - 3) );  
  
 text.setPosition(textPos);  
 Cursor.setPosition(cursorPos);  
  
 cursorPosition = "( ";  
 cursorPosition += intToString(roundValue((mousePos.x-width/2+(-1\*(CenterX-width/2)))/Scaler));  
 cursorPosition += " , ";  
 cursorPosition += intToString(roundValue((-(Graph[0].position.y)+height/2-(-1\*(CenterY-height/2)))/Scaler));  
 cursorPosition += " )";  
  
 text.setString(cursorPosition);  
 window.draw(text);  
 window.draw(Cursor);  
 }  
 }  
 }  
 }  
 // Checks bool and apply color for menu field background  
 if (isThemeBlack) bounds.setFillColor(Color::Black);  
 else bounds.setFillColor(Color(240,240,240));  
  
 // draw bounds for menu, input field for users input and background field for it  
 window.draw(bounds);  
 window.draw(inputField);  
 textbox.drawTo(window);  
  
 // draw text for menu  
 text.setString("Dark theme");  
 text.setPosition({width+100,height-225});  
 window.draw(text);  
 text.setString("Grid on the plane");  
 text.setPosition({width+100,height-150});  
 window.draw(text);  
 text.setString("Cursor on the plane");  
 text.setPosition({width+100,height-75});  
 window.draw(text);  
  
 // Draw all buttons using class  
 button1.drawButton(window);  
 button2.drawButton(window);  
 button3.drawButton(window);  
 button4.drawButton(window);  
 button5.drawButton(window);  
 button6.drawButton(window);  
 button7.drawButton(window);  
 button8.drawButton(window);  
 button9.drawButton(window);  
 button10.drawButton(window);  
 button11.drawButton(window);  
 button12.drawButton(window);  
 button13.drawButton(window);  
 button14.drawButton(window);  
 button15.drawButton(window);  
 button16.drawButton(window);  
 button17.drawButton(window);  
 button18.drawButton(window);  
 button19.drawButton(window);  
 button20.drawButton(window);  
 button21.drawButton(window);  
 button22.drawButton(window);  
 button23.drawButton(window);  
 button24.drawButton(window);  
 button25.drawButton(window);  
 button26.drawButton(window);  
 button27.drawButton(window);  
 button28.drawButton(window);  
 button29.setScale();  
 button30.setScale();  
 button31.setScale();  
 button29.drawButton(window);  
 button30.drawButton(window);  
 button31.drawButton(window);  
  
  
 // displays everything drawn  
 window.display();  
  
 }  
}

* shunting\_yard.h

//  
// Created by User on 29.04.2022.  
//  
  
#ifndef MAIN\_CPP\_SHUNTING\_YARD\_H  
#define MAIN\_CPP\_SHUNTING\_YARD\_H  
#include <iostream>  
#include <string>  
#include <stack>  
#include <queue>  
#include <cmath>  
  
using namespace std;  
  
enum Token\_type // Defining token types  
{  
 NUMBER,  
 OPERATOR,  
 BRACKET,  
 FUNCTION,  
 CONSTANT  
};  
  
struct Token // Creating struct of token, which consists of token type and value  
{  
 Token\_type type;  
 string value;  
};  
  
vector<Token> parse(const string& expression)  
{  
 vector <Token> tokens; // creating dynamic array with type of structure "Token"  
 string digits; // creating digits  
 string letters;  
 int i = 0;  
 while (i < expression.size()) { // while is not end of the string  
 char character = expression[i];  
  
 if (isdigit(character) || character == '.') { // for decimals  
 digits.push\_back(character);  
 }  
  
 else if (isalpha(character)) { // if char is a latter. for ex. a, c, d (for functions)  
 letters.push\_back(character); // separate array to store letters  
 }  
  
 else if ((character == '+') || (character == '-') || (character == '\*') || (character == '/') || (character == '^')) {  
 Token sign;  
  
 if (!digits.empty()) { //if diigts present - push all and clear  
 sign.type = Token\_type::NUMBER;  
 sign.value = digits;  
 tokens.push\_back(sign);  
 digits = "";  
 }  
  
 if (!letters.empty()) { // if letters present - push all and clear  
 sign.type = Token\_type::CONSTANT;  
 sign.value = letters;  
 tokens.push\_back(sign);  
 letters = "";  
 }  
  
 sign.type = Token\_type::OPERATOR; // if operator (e, pi)  
 sign.value = character;  
 tokens.push\_back(sign);  
 }  
 else if (character == '(') {  
 Token bracket;  
  
 if (!letters.empty()) { // if open bracket, firstly push function and then push bracket  
 bracket.type = Token\_type::FUNCTION; // that is for case like that cos(, sin( , etc.  
 bracket.value = letters;  
 tokens.push\_back(bracket);  
 letters = "";  
 }  
 // else push only bracket to the stack  
 bracket.type = Token\_type::BRACKET;  
 bracket.value = character;  
 tokens.push\_back(bracket);  
 }  
 else if (character == ')') {  
 Token sign;  
  
 if (!digits.empty()) { // if digits present - push all and clear  
 sign.type = Token\_type::NUMBER;  
 sign.value = digits;  
 tokens.push\_back(sign);  
 digits = "";  
 }  
  
 if (!letters.empty()) { // if letters present - push all and clear  
 sign.type = Token\_type::CONSTANT;  
 sign.value = letters;  
 tokens.push\_back(sign);  
 letters = "";  
 }  
 // else push only bracket  
 sign.type = Token\_type::BRACKET;  
 sign.value = character;  
 tokens.push\_back(sign);  
 }  
 i++;  
 }  
 // if digits still present - push all to the stack of tokens  
 Token token;  
 if (!digits.empty()) {  
 token.type = Token\_type::NUMBER;  
 token.value = digits;  
 tokens.push\_back(token);  
 }  
 // if letters still present - push all to the stack of tokens  
 if (!letters.empty()) {  
 token.type = Token\_type::CONSTANT;  
 token.value = letters;  
 tokens.push\_back(token);  
 }  
 // return stack with tokens (they are still locking familiar to 1+cos(x), not like rpn  
 return tokens;  
}  
  
vector<Token> parseNegativeNumbers(vector <Token>& tokens)  
{  
 vector <Token> tokens\_;  
 int i = 0;  
 while (i < tokens.size()) {  
 Token symbol = tokens[i];  
  
 if (symbol.type == Token\_type::NUMBER) {  
 tokens\_.push\_back(symbol);  
 }  
  
 if (symbol.type == Token\_type::BRACKET) {  
 tokens\_.push\_back(symbol);  
 }  
  
 if (symbol.type == Token\_type::FUNCTION) {  
 tokens\_.push\_back(symbol);  
 }  
  
 // the whole point is that if the function finds a "-" sign without values  
 // in front of it (like numbers, constants, and brackets), it adds 0 before that number and subtracts it.  
 else if ((symbol.type == Token\_type::OPERATOR) || (symbol.type == Token\_type::CONSTANT)) {  
 if (((tokens\_.empty()) || (tokens\_.back().value == "(")) && symbol.value == "-") {  
 Token token;  
 token.type = Token\_type::NUMBER;  
 token.value = "0";  
  
 tokens\_.push\_back(token);  
 tokens\_.push\_back(symbol);  
 }  
 else {  
 tokens\_.push\_back(symbol);  
 }  
 }  
 i++;  
 }  
 // return stack with tokens but if in the previous step there were something like that: -x\*2-(-x)  
 // now it returns expression like that: 0-x\*2-(0-x). so now we can finally convert this to rpn  
 return tokens\_;  
}  
  
int getPrecision(const Token& symbol) { // here we get precision for the operators  
 if ((symbol.value == "+") || (symbol.value == "-")) return 1;  
 if ((symbol.value == "\*") || (symbol.value == "/")) return 2;  
 if (symbol.value == "^") return 3;  
 return 0;  
}  
  
queue <Token> RPN(const vector <Token>& tokens) { // main part of the Shunting-Yard  
 queue <Token> output;  
 stack <Token> stack\_;  
 int i = 0;  
 while (i < tokens.size()) {  
 Token symbol = tokens[i];  
  
 if ((symbol.type == Token\_type::NUMBER) || (symbol.type == Token\_type::CONSTANT)) {  
 output.push(symbol);  
 }  
  
 if (symbol.type == Token\_type::FUNCTION) {  
 stack\_.push(symbol);  
 }  
  
 // this part is responsible for moving the higher-priority operators  
 // on the stack and the lower-priority ones to the queue  
 if (symbol.type == Token\_type::OPERATOR) {  
 while ((!stack\_.empty()) && (stack\_.top().value != "(") && ((stack\_.top().type == Token\_type::FUNCTION) || (getPrecision(stack\_.top()) >= getPrecision(symbol))))  
 {  
 output.push(stack\_.top());  
 stack\_.pop();  
 }  
 stack\_.push(symbol);  
 }  
  
 if (symbol.value == "(") {  
 stack\_.push(symbol);  
 }  
  
 if (symbol.value == ")") { // while function can not find "(" it will devastate stack and fill queue  
 while (stack\_.top().value != "(") {  
 output.push(stack\_.top());  
 stack\_.pop();  
 }  
 if (stack\_.top().value == "(") {  
 stack\_.pop();  
 }  
 }  
 i++;  
 }  
 // drags all the remaining contents of the stack to the queue and clears the stack  
 while(!stack\_.empty()) {  
 output.push(stack\_.top());  
 stack\_.pop();  
 }  
 return output;  
}  
  
double stack\_calc(queue<Token>& rpn\_tokens, double x) { // easiest part. here the values are taken out one by one,  
 double result; // and the solution stays on the stack at the top  
 stack <double> stack\_;  
  
 while (!rpn\_tokens.empty()) {  
 Token token = rpn\_tokens.front();  
  
 if (token.type == Token\_type::NUMBER) {  
 double number = stof(token.value);  
 stack\_.push(number);  
 }  
 else if (token.type == Token\_type::CONSTANT) {  
 double constant;  
 if (token.value == "e") {  
 constant = M\_E;  
 stack\_.push(constant);  
 }  
  
 if (token.value == "x" || token.value == "X") { // additional ability to using uppercase X  
 constant = x; // This can also be done for all constants and  
 stack\_.push(constant); // functions, but I don't see the point.  
 }  
  
 if (token.value == "pi") {  
 constant = M\_PI;  
 stack\_.push(constant);  
 }  
 }  
 else if (token.type == Token\_type::OPERATOR) {  
 double a = stack\_.top();  
 stack\_.pop();  
  
 double b = stack\_.top();  
 stack\_.pop();  
  
 if (token.value == "+") {  
 stack\_.push(a + b);  
 }  
  
 if (token.value == "-") {  
 stack\_.push(b - a);  
 }  
  
 if (token.value == "\*") {  
 stack\_.push(a \* b);  
 }  
  
 if (token.value == "/") {  
 stack\_.push(b / a);  
 }  
  
 if (token.value == "^") {  
 stack\_.push(pow(b, a));  
 }  
 }  
 else if (token.type == Token\_type::FUNCTION) {  
 double c = stack\_.top();  
 stack\_.pop();  
  
 if (token.value == "sqrt") {  
 stack\_.push(sqrt(c));  
 }  
  
 if (token.value == "sin") {  
 stack\_.push(sin(c));  
 }  
  
 if (token.value == "cos") {  
 stack\_.push(cos(c));  
 }  
  
 if (token.value == "tan") {  
 stack\_.push((sin(c) / cos(c)));  
 }  
  
 if (token.value == "cot") {  
 stack\_.push((cos(c) / sin(c)));  
 }  
  
 if (token.value == "ln") {  
 stack\_.push((log(c)));  
 }  
 }  
 rpn\_tokens.pop();  
 }  
 result = stack\_.top();  
 return result;  
}  
  
float evaluate(string& expression,double x) {  
 vector <Token> tokens = parse(expression);  
 vector <Token> tokens\_ = parseNegativeNumbers(tokens);  
 queue <Token> rpn\_tokens = RPN(tokens\_);  
 return stack\_calc(rpn\_tokens,x);  
}  
  
#endif //MAIN\_CPP\_SHUNTING\_YARD\_H

* **textbox.h**

//  
// Created by User on 29.04.2022.  
//  
  
#ifndef MAIN\_CPP\_TEXTBOX\_H  
#define MAIN\_CPP\_TEXTBOX\_H  
#define DELETE\_KEY 8  
#define ENTER\_KEY 13  
#define ESCAPE\_KEY 27  
  
#include <iostream>  
#include <sstream>  
using namespace sf;  
using namespace std;  
  
// class with user input processing and correct operation of the input field  
class Textbox {  
public:  
 Textbox(int size, Color color) { // Constructor for input field  
 textbox.setCharacterSize(size);  
 textbox.setFillColor(color);  
 textbox.setString("\_");  
 }  
  
 void setFont(Font &font) { // function for defining font of text field  
 textbox.setFont(font);  
 }  
  
 void setPosition(Vector2f pos) { // function for draw input field  
 textbox.setPosition(pos);  
 }  
  
 string getText() { // function for get text  
 return text.str();  
 }  
  
 void drawTo (RenderWindow &window) { // function for draw input field  
 window.draw(textbox);  
 }  
  
 void typedButtonKeyboard(Event input) {  
 int charTyped = input.text.unicode; // Converting text for check is it from latin alphabet  
 if (charTyped < 128) {  
 if (text.str().length() <= limit) {  
 inputLogic(charTyped);  
 }  
 else if(text.str().length() > limit && charTyped && DELETE\_KEY) {  
 deleteLastChar();  
 }  
 }  
 }  
  
 void typedButtonMenu(const string& inputButton) {  
 int i = 0;  
 while (i<inputButton.length()) {  
 if (text.str().length() <= limit) {  
 inputLogic(inputButton[i]);  
 }  
 else if(text.str().length() > limit && inputButton[i] && DELETE\_KEY) {  
 deleteLastChar();  
 }  
 i++;  
 }  
 }  
  
 void deleteChar() {  
 if (text.str().length() != 0) {  
 deleteLastChar();  
 }  
 textbox.setString(text.str() + "\_");  
 }  
  
 void deleteString() {  
 if (text.str().length() != 0) {  
 while (text.str().length() > 0) {  
 deleteLastChar();  
 }  
 textbox.setString("\_");  
 }  
 }  
private:  
 Text textbox;  
 ostringstream text;  
 int limit = 30;  
  
 void inputLogic(int charTyped) {  
 if(charTyped != DELETE\_KEY && charTyped != ENTER\_KEY && charTyped != ESCAPE\_KEY) {  
 text << static\_cast<char>(charTyped);  
 }  
 else if (charTyped == DELETE\_KEY) {  
 if (text.str().length() > 0) {  
 deleteLastChar();  
 }  
 }  
 textbox.setString(text.str() + "\_");  
 }  
 void deleteLastChar() {  
 string str = text.str();  
 string newStr;  
 for (int i = 0; i < str.length()-1; i++) {  
 newStr += str[i];  
 }  
 text.str("");  
 text << newStr;  
 }  
};  
  
#endif //MAIN\_CPP\_TEXTBOX\_H

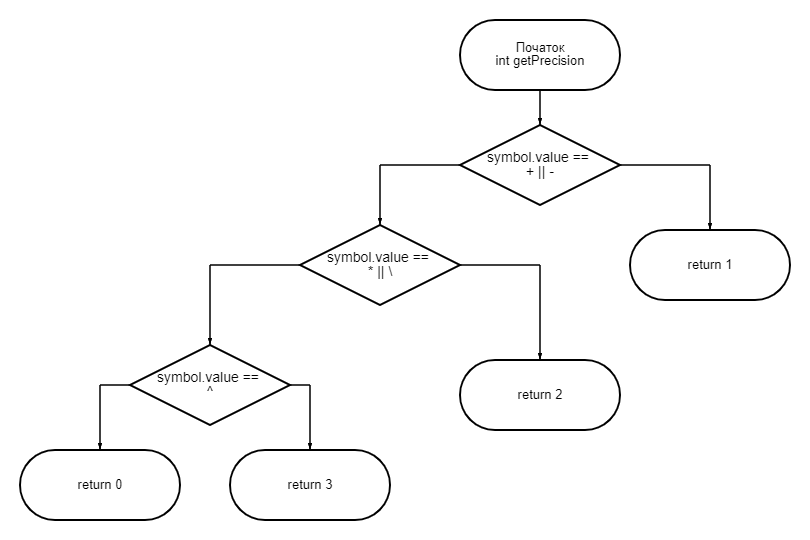
* **button.h**

//  
// Created by User on 29.04.2022.  
//  
  
#ifndef MAIN\_CPP\_BUTTON\_H  
#define MAIN\_CPP\_BUTTON\_H  
#include "utility.h"  
  
class Button {  
public:  
 Button(const string& name, Vector2f position) { // Constructor for each button  
 button.setPosition(position);  
 buttonT.loadFromFile(texturesDirectory+name);  
 button.setTexture(buttonT);  
 }  
 void setTexture(const string& name) {  
 buttonT.loadFromFile(texturesDirectory+name);  
 button.setTexture(buttonT);  
 }  
  
 void setScale() {  
 button.setScale(0.12,0.12);  
 }  
  
 void drawButton (RenderWindow &window) {  
 window.draw(button);  
 }  
  
 void setColor(Color color) {  
 button.setColor(color);  
 }  
  
 FloatRect getGlobalBounds() {  
 return button.getGlobalBounds();  
 }  
private:  
 Sprite button;  
 Texture buttonT;  
 string texturesDirectory = "../textures/";  
};  
  
#endif //MAIN\_CPP\_BUTTON\_H

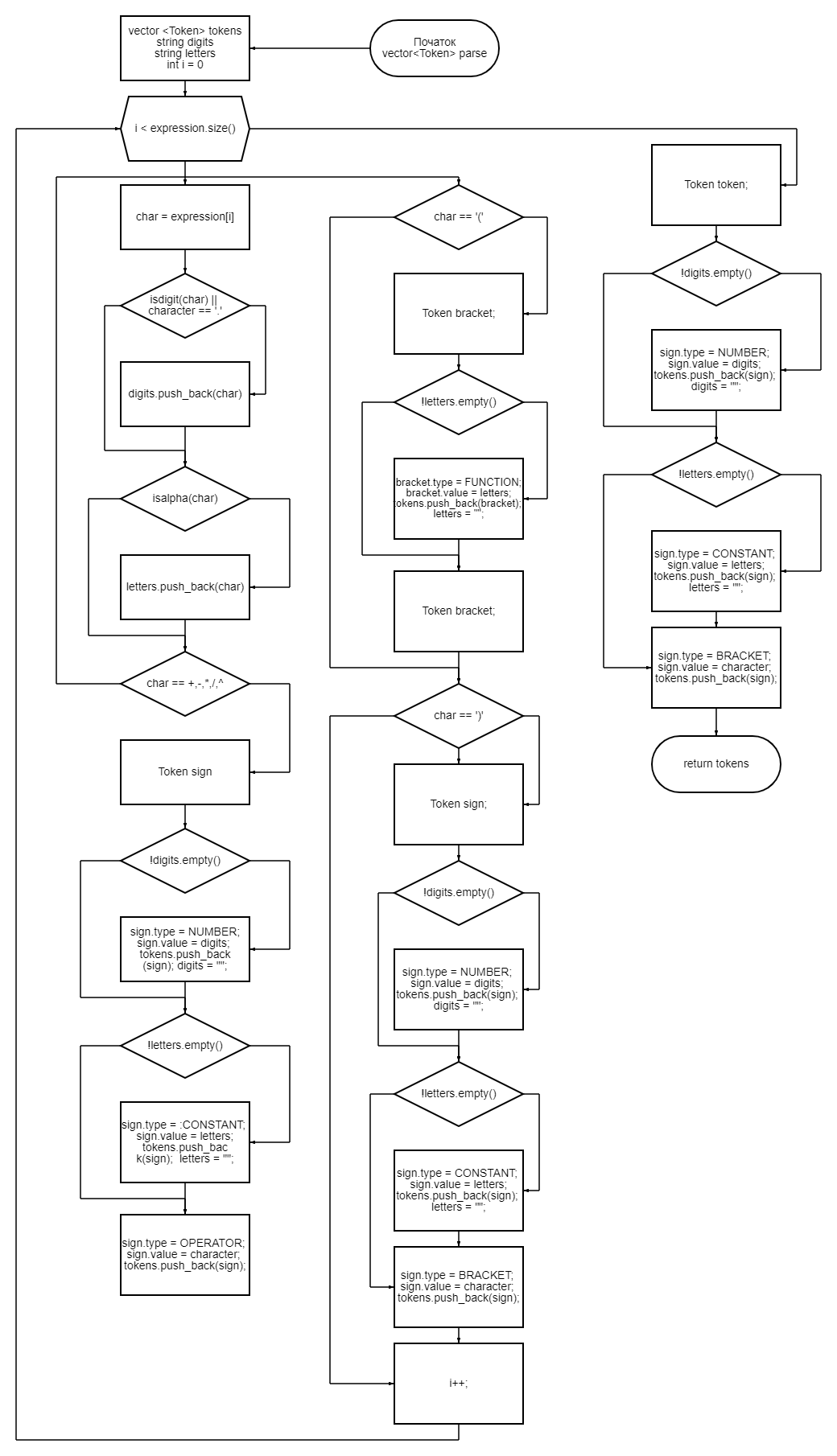
* utility.h

//  
// Created by User on 09.06.2022.  
//  
  
#ifndef MAIN\_CPP\_UTILITY\_H  
#define MAIN\_CPP\_UTILITY\_H  
#include <sstream>  
  
float roundValue(float var) { // round the number down to a hundredth  
 float value = (int)(var \* 100 + .5);  
 return value / 100;  
}  
  
string intToString(float number) {  
 stringstream ss; // create a stringstream  
 string str; // create string  
 ss << number; // add number to the stream  
 ss >> str; // push the contents onto a string  
 return str; // return the string  
}  
  
#endif //MAIN\_CPP\_UTILITY\_H

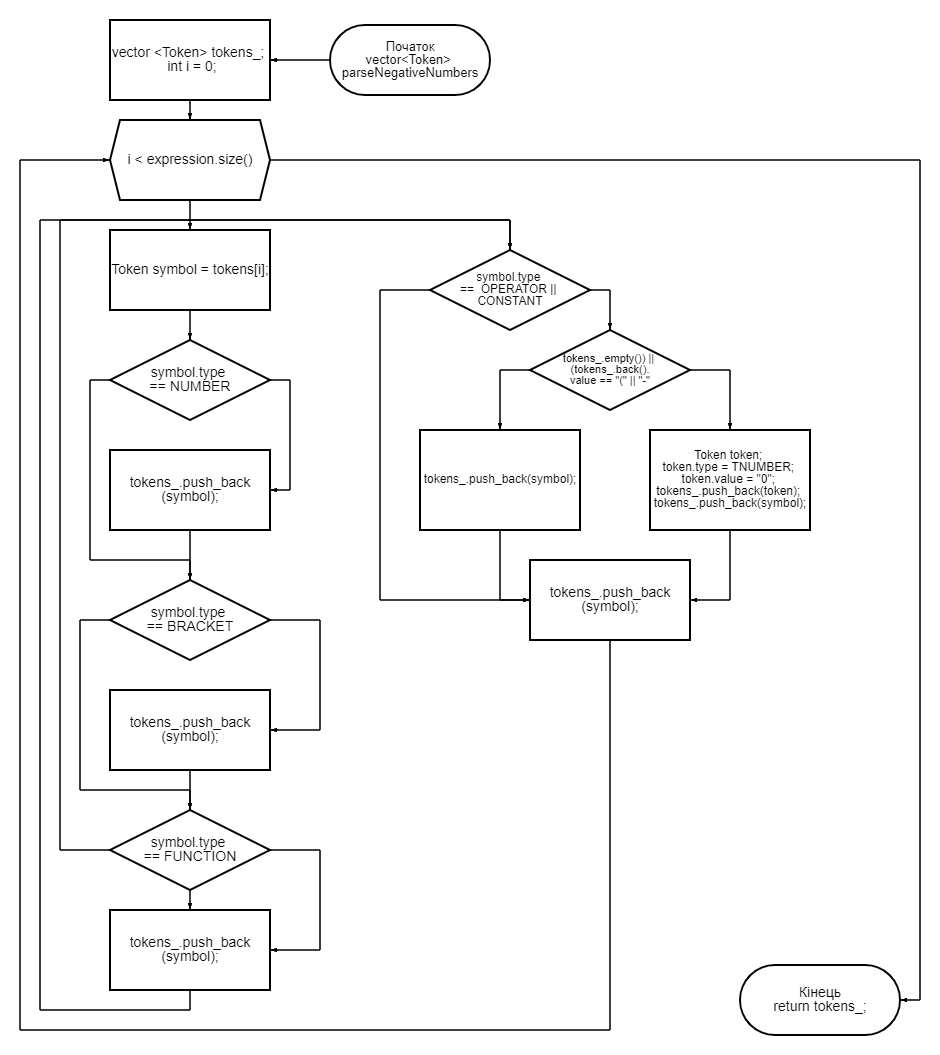
## **Б****лок-схеми**



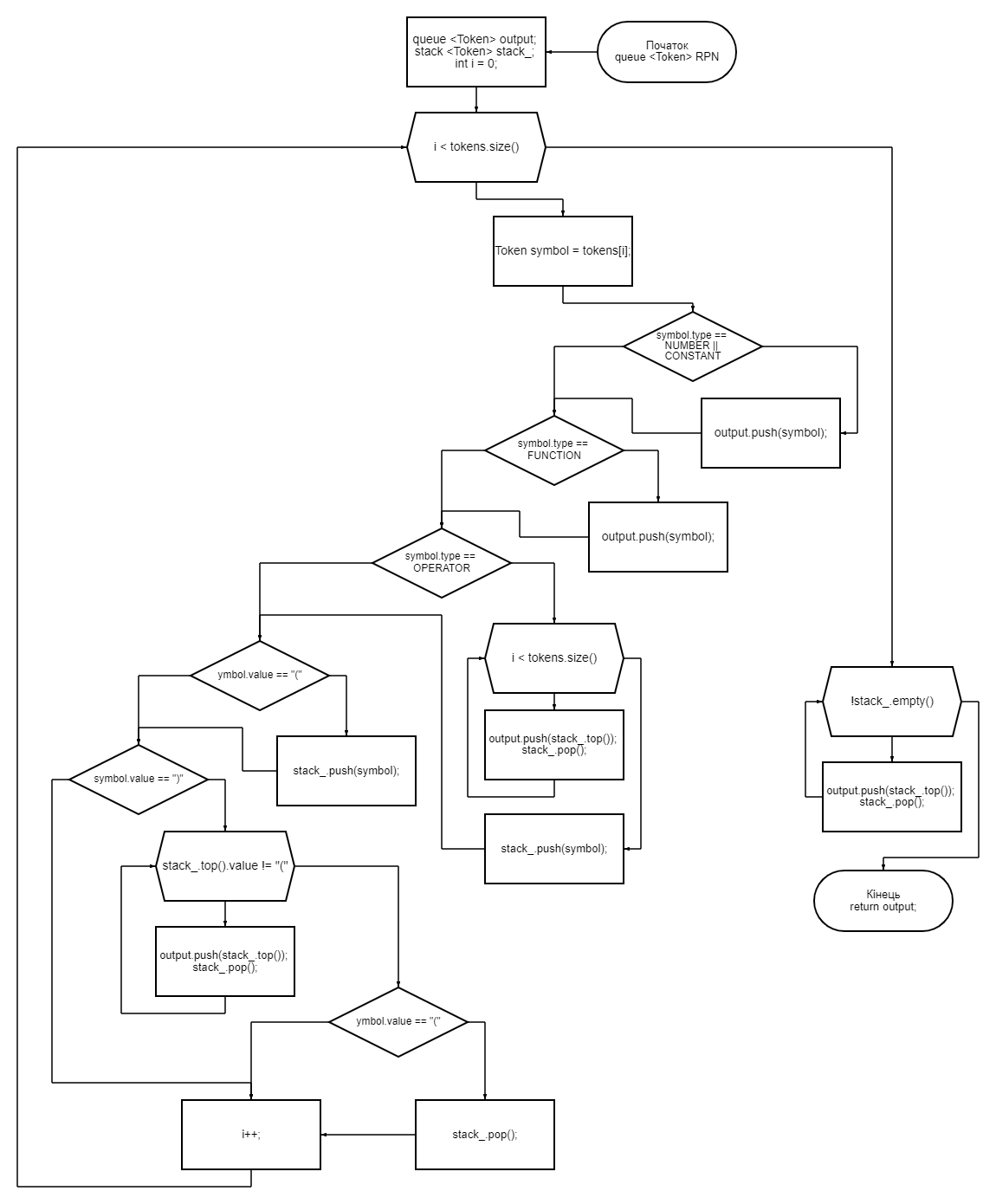
Блок-схема 1 – getPrecision()



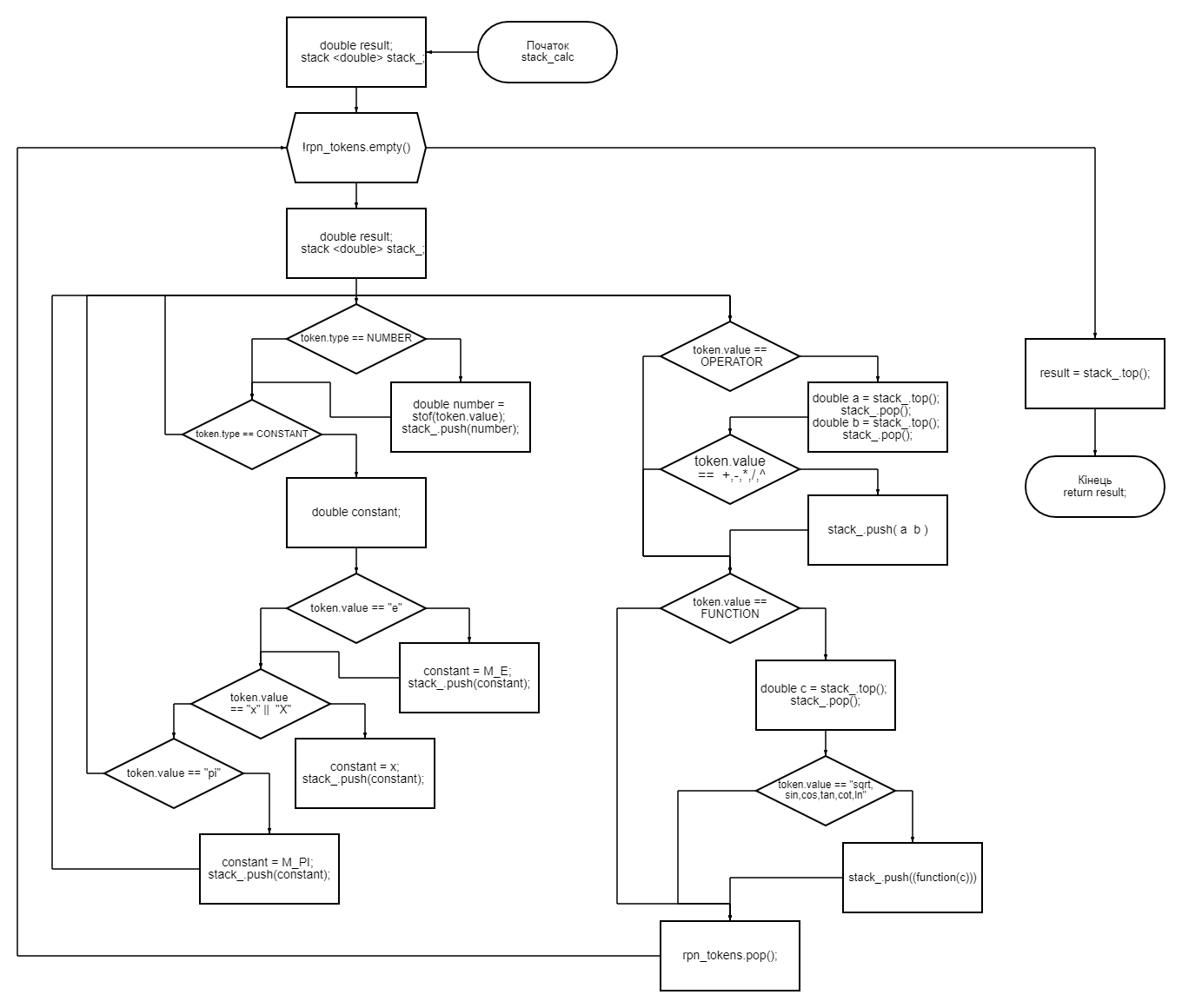
Блок-схема 2 – vector<Token> parse()



Блок-схема 3 – vector<Token> parseNegativeNumbers()



Блок-схема 4 – queue<Token> RPN()



Блок-схема 5 – double stack\_calc()