ЛЬВІВСЬКИЙ ТОРГОВЕЛЬНО-ЕКОНОМІЧНИЙ УНІВЕРСИТЕТ

Кафедра комп'ютерних наук, прикладної та вищої математики

КВАЛІФІКАЦІЙНА РОБОТА

на тему:

РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ЗДІЙСНЕННЯ ОБЧИСЛЕНЬ

Виконав: здобувач першого (бакалаврського)
рівня вищої освіти
ОПП "Комп'ютерні науки"
спеціальності 122 "Комп'ютерні науки"
_____ Пйонтик Максим Тарасович
(підпис)

Науковий керівник: ст. викл
_____ Плеша Василь Іванович
(підпис)

Львів - 2026

ЛЬВІВСЬКИЙ ТОРГОВЕЛЬНО-ЕКОНОМІЧНИЙ УНІВЕРСИТЕТ

Факультет міжнародних економічних відносин та інформаційних технологій
Кафедра комп'ютерних наук, прикладної та вищої математики
Перший (бакалаврський) рівень вищої освіти
Освітньо-професійна програма "Комп'ютерні науки"
Спеціальність 122 "Комп'ютерні науки"

"**ЗАТВЕРДЖУЮ**"
Завідувач кафедри комп'ютерних наук,
прикладної та вищої математики
_____ Новосад З. Г.
(підпис) (прізвище та ініціали)

13.10.2025 р.

ЗАВДАННЯ
до кваліфікаційної роботи здобувачу
Пйонтик Максим Тарасович

1. Тема кваліфікаційної роботи «Розробка програмного забезпечення для здійснення обчислень»;
керівник кваліфікаційної роботи Плеша Василь Іванович, ст. викл;
затверджена наказом по Університету ***№ 991/03 від 06 жовтня 2025 р.***
2. Термін здачі здобувачем вищої освіти кваліфікаційної роботи 12 червня 2026 р.
3. Вхідні дані до роботи: література, документація, матеріали та сайти Internet, що пов'язані з розробкою програмного забезпечення для здійснення обчислень .
4. Зміст розрахунково-пояснювальної записки (перелік питань, які підлягають розробці)
   1) Опис предметної області та постановка задачі; 2) Проєктування та інструментальні засоби реалізації проєкту; 3) Реалізація програмного забезпечення для здійснення обчислень;
5. Перелік графічного матеріалу до 12 слайдів у форматі PowerPoint
6. Консультанти розділів кваліфікаційної роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання 13.10.2025 р.

Календарний план

№ з/п	Найменування етапів робіт	Термін виконання	Примітка
1.	Отримання завдання і аналіз інформаційних джерел за темою роботи	до 11.12.2025	
2.	Опис предметної області та постановка задачі	до 13.01.2026	
3.	Проєктування та інструментальні засоби реалізації проєкту	до 07.02.2026	
4.	Реалізація програмного забезпечення для здійснення обчислень	до 01.04.2026	
5.	Перевірка роботи програми	до 15.04.2026	
6.	Підготовка до оформлення кваліфікаційної роботи	до 22.04.2026	
7.	Оформлення кваліфікаційної роботи. Подання роботи на перевірку керівнику	до 16.05.2026	
8.	Доопрацювання кваліфікаційної роботи з врахуванням зауважень керівника. Подання роботи для перевірки на рівень запозичень	до 20.05.2026	
9.	Подання кваліфікаційної роботи керівнику для отримання відгуку	до 27.05.2026	
10.	Представлення кваліфікаційної роботи на зовнішнє рецензування. Подання кваліфікаційної роботи на кафедру комп'ютерних наук	до 5.06.2026	
11.	Попередній захист кваліфікаційної роботи	09.06.2026 - 10.06.2026	
12.	Захист кваліфікаційної роботи на засіданні екзаменаційної комісії	15.06.2026 - 19.06.2026 (згідно графіку роботи ЕК)	

Здобувач вищої освіти _____ Пйонтик М. Т.

Керівник роботи _____ Плеша В. І.

Реферат

Робота складається із вступу, трьох розділів, висновків, списку використаних джерел та додатків. Основний зміст роботи викладено на 41 сторінках і містить 3 рисунки, 2 таблиці та лістинги програмного коду.

Об'єктом дослідження є процес створення програмного забезпечення для здійснення математичних обчислень.

Предметом дослідження є методи та засоби розробки настільних застосунків із графічним інтерфейсом користувача з використанням фреймворку Qt та мови програмування C++.

Метою дослідження є розробка програмного застосунку «Калькулятор», що забезпечує коректне обчислення математичних виразів із підтримкою базових арифметичних операцій, дужок, степеня та константи π.

У першому розділі розглядаються теоретичні основи створення програмних застосунків для обчислень, аналізуються існуючі рішення та визначаються вимоги до функціональності майбутньої системи.

У другому розділі здійснено вибір технологій розробки, зокрема мови програмування C++17 та фреймворку Qt 6, а також описано архітектуру застосунку, структуру класів, механізм взаємодії елементів інтерфейсу та алгоритм обчислення математичних виразів на основі двох стеків.

У третьому розділі представлено реалізацію програмного забезпечення, включаючи розробку логічного ядра обчислень, створення графічного інтерфейсу користувача, а також результати тестування програми у різних сценаріях використання.

Перелік ключових слів: калькулятор, C++, Qt, графічний інтерфейс, обчислення, алгоритм двох стеків, Qt Creator, QSS.

Abstract

The paper consists of an introduction, three chapters, conclusions, a list of references, and appendices. The main content of the work is presented on 41 pages and includes 3 figures, 2 tables, and program code listings.

The object of the study is the process of developing software for performing mathematical computations.

The subject of the study is the methods and tools for developing desktop applications with a graphical user interface using the Qt framework and the C++ programming language.

The aim of the study is to develop a software application “Calculator” that ensures correct evaluation of mathematical expressions with support for basic arithmetic operations, parentheses, exponentiation, and the constant π.

The first chapter discusses the theoretical foundations of creating software applications for computations, analyzes existing solutions, and defines the functional requirements for the future system.

The second chapter describes the selection of development technologies, in particular the C++17 programming language and the Qt 6 framework, as well as the application architecture, class structure, interaction mechanisms between interface elements, and the algorithm for evaluating mathematical expressions based on two stacks.

The third chapter presents the implementation of the software, including the development of the computational core logic, the creation of the graphical user interface, and the results of testing the application under various usage scenarios.

Keywords: calculator, C++, Qt, graphical user interface, computations, two-stack algorithm, Qt Creator, QSS.

ЗМІСТ

ВСТУП 7
1. ОПИС ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ 9
1.1. Предметна область, основні поняття та сутності програмного забезпечення для здійснення обчислень 9
1.2. Задачі та призначення програмного забезпечення для здійснення обчислень 13
1.3. Аналіз існуючих рішень 15
2. ПРОЄКТУВАННЯ ТА ІНСТРУМЕНТАЛЬНІ ЗАСОБИ РЕАЛІЗАЦІЇ ПРОЄКТУ 19
2.1. Аналіз мов програмування та інструментів для реалізації проєкту 19
2.2. Інші допоміжні інструменти 22
2.3. Проєктування архітектури та основні користувацькі сценарії програмного забезпечення для здійснення обчислень 24
3. РЕАЛІЗАЦІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ЗДІЙСНЕННЯ ОБЧИСЛЕНЬ 28
3.1. Реалізація програми для здійснення обчислень 28
3.2. Створення інтерфейсу 31
3.3. Огляд роботи програми для здійснення обчислень 34
ВИСНОВКИ 38
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 40
ДОДАТКИ 42

ВСТУП

Сучасний світ важко уявити без комп'ютерної техніки та програмного забезпечення. Практично в кожній галузі людської діяльності — від освіти і науки до торгівлі та промисловості — широко застосовуються різноманітні програми для автоматизації обчислень і обробки даних. Програмне забезпечення для здійснення обчислень є одним із найпоширеніших типів прикладних програм: його використовують студенти під час навчання, інженери при розрахунках конструкцій, фінансисти для аналізу показників та звичайні користувачі в повсякденному житті.

Актуальність теми кваліфікаційної роботи обумовлена кількома чинниками. По-перше, незважаючи на широкий вибір готових рішень, більшість із них є або надто складними для пересічного користувача, або платними, або такими, що залежать від підключення до мережі Інтернет. По-друге, розробка власного калькулятора з відкритим вихідним кодом дає змогу повністю контролювати функціональність програми, адаптувати її під конкретні потреби та розширювати у майбутньому. По-третє, реалізація такого проєкту є чудовою практичною задачею для відпрацювання навичок програмування мовою C++ із застосуванням сучасного Qt Framework.

У рамках даної кваліфікаційної роботи розроблене програмне забезпечення для здійснення обчислень. Програма реалізована мовою C++ стандарту C++17 із використанням Qt 6 Framework та системи збірки qmake. Застосунок підтримує виконання стандартних арифметичних операцій (додавання, віднімання, множення, ділення), піднесення до степеня, роботу з дужками та підстановку математичної константи π. Обчислення виконуються за алгоритмом двох стеків із врахуванням пріоритетів операцій.

Метою кваліфікаційної роботи є проєктування, реалізація та тестування програмного забезпечення для здійснення математичних обчислень із зручним графічним інтерфейсом користувача, що відповідає сучасним вимогам до десктопних застосунків.

Для досягнення поставленої мети визначено такі завдання дослідження:

– проаналізувати предметну область програмного забезпечення для обчислень та визначити основні поняття і вимоги;
– дослідити та порівняти наявні аналогічні програмні рішення;
– обґрунтувати вибір технологічного стеку та інструментів розробки;
– спроєктувати архітектуру застосунку, описати основні компоненти та користувацькі сценарії;
– реалізувати алгоритм обчислення математичних виразів;
– розробити графічний інтерфейс користувача у стилі сучасного темного оформлення;
– провести функціональне тестування готового застосунку та описати результати.

Об'єктом дослідження є процес розробки прикладного програмного забезпечення для виконання математичних обчислень з графічним інтерфейсом.

Предметом дослідження є методи, алгоритми та інструменти проєктування і реалізації десктопних застосунків на базі мови C++ та Qt Framework.

Практична значущість роботи полягає у створенні повноцінного функціонального застосунку-калькулятора, що може бути використаний як самостійний інструмент або як основа для подальшого розширення функціональності, зокрема додавання тригонометричних функцій, побудови графіків та підтримки складних виразів.

1. ОПИС ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1. Предметна область, основні поняття та сутності програмного забезпечення для здійснення обчислень

Програмне забезпечення для здійснення обчислень є одним із найстаріших і найбільш поширених класів прикладних програм. Ще до появи персональних комп'ютерів у 1960-х роках на мейнфреймах IBM існували програми для виконання математичних розрахунків. З появою мікропроцесорів і персональних комп'ютерів у 1970–1980-х роках виникли перші калькулятори — спочатку апаратні, а потім і програмні. Сьогодні калькулятори присутні на кожному смартфоні, планшеті та комп'ютері як стандартний системний інструмент.

Предметна область програмного забезпечення для здійснення обчислень охоплює сукупність методів, засобів і технологій, що забезпечують виконання математичних, логічних, статистичних та інших видів обчислень із використанням комп’ютерних систем. Основною метою такого програмного забезпечення є автоматизація процесу отримання результатів на основі вхідних даних, підвищення точності розрахунків, зменшення часу їх виконання та мінімізація впливу людського фактору.

У межах цієї предметної області ключовим є поняття обчислювального процесу, який являє собою послідовність операцій над даними, що виконуються відповідно до заданого алгоритму. Алгоритм виступає формалізованим описом кроків розв’язання задачі та є базовою одиницею будь-якої обчислювальної системи. Його реалізація здійснюється за допомогою програмного коду, який інтерпретується або компілюється середовищем виконання.

Важливим поняттям є дані, які можуть бути представлені у різних формах: числовій, символьній, структурованій або неструктурованій. У процесі обчислень дані проходять етапи введення, обробки, зберігання та виведення результатів. Відповідно, програмне забезпечення для обчислень включає механізми роботи з пам’яттю, засоби введення/виведення та модулі математичної обробки.

До основних сутностей програмного забезпечення для обчислень належать:

користувач — суб’єкт, який ініціює обчислювальний процес і задає параметри задачі;

обчислювальний модуль — програмна компонента, що реалізує конкретні алгоритми розрахунків;

інтерфейс користувача — засіб взаємодії між користувачем і системою, який забезпечує введення даних та отримання результатів;

алгоритмічні структури — логічні конструкції (послідовність, розгалуження, цикли), що визначають порядок виконання операцій;

система керування пам’яттю та ресурсами — компоненти, які забезпечують ефективне використання обчислювальних ресурсів;

математичні бібліотеки та функції — набір готових реалізацій стандартних обчислювальних операцій.

Сучасне програмне забезпечення для обчислень часто реалізується у вигляді інтегрованих систем або спеціалізованих застосунків, що підтримують як чисельні, так і символьні обчислення. До таких систем належать наукові обчислювальні середовища, математичні пакети, а також спеціалізовані калькулятори та мобільні додатки.

Особливістю предметної області є тісний зв’язок із такими напрямами, як математичне моделювання, теорія алгоритмів, об’єктно-орієнтоване програмування та системне програмування. Це зумовлює необхідність комплексного підходу до проєктування програмного забезпечення, яке повинно забезпечувати не лише коректність обчислень, але й масштабованість, швидкодію та зручність використання.

Предметна область даної роботи охоплює розробку прикладного програмного забезпечення, призначеного для обчислення математичних виразів. Розглянемо основні поняття та терміни цієї предметної області.

Математичний вираз — це запис, що складається з чисел (операндів), знаків арифметичних або інших операцій (операторів) і, за потреби, дужок, який може бути обчислений за встановленими математичними правилами та повертає числове значення. Прикладами виразів є: «2 + 3», «(4 − 1) × 7», «3 ^ 2 + π», «(10 / 2) − (3 × 4)». Вирази можуть бути простими (одна операція) та складеними (кілька операцій з різними пріоритетами).

Операнд — числове значення або ідентифікатор (наприклад, константа π), над яким виконується математична операція. У виразі «5 + 3 × 2» операндами є числа 5, 3 і 2. Операнди можуть бути цілими числами (наприклад, 7, −4) або числами з плаваючою комою (наприклад, 3.14, −0.5).

Оператор — символ або позначення математичної дії, що визначає операцію, яку слід виконати над операндами. У розробленому застосунку підтримуються такі оператори: «+» (додавання), «−» (віднімання), «×» або «*» (множення), «÷» або «/» (ділення), «^» (піднесення до степеня). Кожен оператор має визначений пріоритет виконання.

Пріоритет операцій — це порядок, у якому виконуються операції у виразі за відсутності дужок. Загальноприйняте правило: піднесення до степеня виконується першим, потім множення і ділення (зліва направо), і в останню чергу — додавання та віднімання (зліва направо). Дужки мають найвищий пріоритет і дозволяють змінити стандартний порядок обчислень.

Математична константа π (пі) — ірраціональне число, що приблизно дорівнює 3.14159265358979..., яке визначається як відношення довжини кола до його діаметра. π широко використовується в геометрії, тригонометрії та фізиці. У розробленому застосунку π підтримується як спеціальна кнопка, при натисканні якої у вираз підставляється точне значення π, обчислене засобами C++ як acos(-1.0).

Парсинг виразу (синтаксичний розбір) — процес аналізу рядка символів, що являє собою математичний вираз, з метою визначення його структури та порядку виконання операцій. Парсер виділяє з вхідного рядка лексеми —

мінімальні смислові одиниці (числа, оператори, дужки) — та будує з них внутрішнє представлення для подальшого обчислення.

Лексема (токен) — мінімальна смислова одиниця мови виразів. У контексті даного проєкту лексема визначена як структура Leksema із двома полями: type (символ, що позначає тип: '0' для числа, '+', '-', '*', '/', '^', '(' для відповідних операторів та дужки) та value (числове значення для лексем-чисел). Така структура є спрощеним аналогом токенів у повноцінних лексичних аналізаторах.

Стек (stack) — абстрактна структура даних, що працює за принципом LIFO (Last In — First Out: останній прийшов — перший вийшов). У алгоритмі обчислення виразів використовуються два стеки: стек операндів (Stack_n) для зберігання числових значень і стек операторів (Stack_o) для зберігання операцій і дужок. Стек реалізований стандартним класом std::stack із бібліотеки STL.

Зворотний польський запис (RPN, Reverse Polish Notation або постфіксний запис) — це форма запису математичного виразу, у якій оператор розміщується після своїх операндів. Наприклад, вираз «3 + 4» у RPN записується як «3 4 +», а «(3 + 4) × 5» — як «3 4 + 5 ×». Перевага RPN полягає в тому, що для її обчислення не потрібні дужки і не потрібно знати пріоритети операцій — вираз обчислюється лінійно за допомогою одного стека.

Алгоритм сортувальної станції (Shunting Yard Algorithm) — алгоритм, запропонований Едсгером Дейкстрою, що перетворює математичний вираз зі стандартного інфіксного запису у постфіксний (RPN) з урахуванням пріоритетів операцій та дужок. Саме цей алгоритм (у спрощеній формі — алгоритм двох стеків) лежить в основі математичного ядра розробленого калькулятора.

Графічний інтерфейс користувача (GUI — Graphical User Interface) — сукупність візуальних елементів (вікон, кнопок, полів введення, міток тощо), що забезпечують взаємодію між людиною та програмою у зручній та інтуїтивно зрозумілій формі. У Qt GUI-компоненти реалізовані у вигляді класів-нащадків від базового класу QWidget.

Кросплатформність — здатність програмного забезпечення запускатись і коректно функціонувати на різних операційних системах без суттєвої зміни вихідного коду. Qt Framework забезпечує кросплатформність завдяки власному рівню абстракції над API різних ОС, що дозволяє компілювати один і той самий Qt-код для Windows, Linux і macOS.

1.2. Задачі та призначення програмного забезпечення для здійснення обчислень

Програмне забезпечення для здійснення обчислень вирішує широкий спектр задач у різних сферах людської діяльності. Залежно від галузі застосування функціональні вимоги до такого ПЗ суттєво відрізняються, проте існує базовий набір завдань, актуальних для будь-якого калькулятора.

Основним призначенням розробленого застосунку є надання зручного та швидкого інструменту для обчислення довільних арифметичних виразів без необхідності виконувати розрахунки вручну. Цільова аудиторія програми — студенти, учні, фахівці технічних спеціальностей та всі, хто регулярно стикається з потребою виконувати обчислення у повсякденній роботі.

Функціональні вимоги до розробленого ПЗ визначають перелік дій, які має виконувати система. До функціональних вимог даного проєкту належать:

– приймання від користувача математичного виразу у вигляді рядка символів, введеного з клавіатури або сформованого через кнопки інтерфейсу;
– підтримка арифметичних операцій: додавання (+), віднімання (−), множення (×), ділення (÷);
– підтримка операції піднесення до степеня (^);
– правильна обробка дужок для зміни порядку обчислення операцій;
– підтримка унарного мінуса (від'ємних чисел та від'ємних виразів у дужках);
– підтримка десяткових дробів з крапкою як роздільником;
– підстановка математичної константи π (кнопка «π»);

– відображення поточного виразу в полі дисплея з можливістю його редагування;

– відображення попереднього виразу у рядку «історії» після обчислення;

– виведення результату обчислення або повідомлення про помилку;

– очищення поля дисплея (кнопка «C»);

– видалення останнього введеного символу (кнопка «⌫»);

– підтримка введення виразу з фізичної клавіатури комп'ютера.

Нефункціональні вимоги описують якісні характеристики системи. До ключових нефункціональних вимог даного проєкту належать:

– Швидкодія: обчислення результату має виконуватись миттєво (менш ніж 0.1 секунди) для будь-якого коректного виразу розумної довжини;

– Надійність: програма не повинна аварійно завершуватись при введенні будь-яких некоректних даних — натомість відображати повідомлення «Помилка»;

– Кросплатформність: вихідний код має компілюватись та коректно виконуватись на Windows, Linux та macOS без модифікацій;

– Зручність використання: інтерфейс має бути інтуїтивно зрозумілим, не вимагати інструкцій для початку роботи;

– Підтримуваність: код має бути чітко структурованим, добре коментованим та легко розширюваним для додавання нових функцій;

– Мінімальність залежностей: застосунок не повинен вимагати встановлення додаткових бібліотек поза межами стандартної Qt-поставки.

Типові сценарії використання застосунку охоплюють широке коло задач: побутові обчислення (підрахунок здачі, розрахунок відсотків за кредитом, визначення площі приміщення), навчальні задачі (перевірка результатів домашніх завдань, розрахунок математичних виразів під час самопідготовки), технічні розрахунки (перевірка проміжних результатів інженерних обчислень), наукові обчислення (вирази з константою π, степеневі функції).

Варто відзначити, що програма спроєктована з урахуванням принципу мінімалізму: вона реалізує рівно той функціонал, який необхідний для зручного та надійного виконання арифметичних і степеневих обчислень, не перевантажуючи користувача зайвими опціями. Водночас архітектурне рішення дозволяє легко розширити функціонал у майбутніх версіях.

1.3. Аналіз існуючих рішень

Перед розробкою власного рішення проведено аналіз існуючих програм для здійснення обчислень. Це дозволило визначити сильні та слабкі сторони наявних продуктів, виявити незаповнені ніші та обґрунтувати доцільність розробки власного застосунку.

Калькулятор Windows. Вбудований калькулятор Windows (додаток «Калькулятор») є одним із найпоширеніших програмних калькуляторів у світі завдяки тому, що входить до стандартної поставки Windows. Програма пропонує кілька режимів: стандартний (базові чотири дії), науковий (тригонометричні та логарифмічні функції, факторіал), інженерний (побітові операції), а також конвертор одиниць виміру та розрахунок іпотеки. Серед переваг — безкоштовність, наявність наукового режиму, підтримка клавіатурного введення, висока надійність. Недоліки: доступний лише для Windows, не є відкритим програмним забезпеченням, обмежені можливості налаштування зовнішнього вигляду.

Калькулятор macOS (Calculator). Системний калькулятор Apple пропонує три режими: базовий, науковий (з тригонометричними функціями, логарифмами, константами) і програмістський (побітові операції, різні системи числення). Відрізняється стримним дизайном у стилі macOS та підтримкою Paper Tape — стрічки обчислень, що зберігає повну історію виразів. Недолік — платформна прив'язаність до macOS.

GNOME Calculator / KCalc (Linux). На платформі Linux популярні системні калькулятори GNOME Calculator та KCalc. GNOME Calculator — частина

робочого середовища GNOME, підтримує базовий та науковий режими, має сучасний GTK-дизайн. KCalc — калькулятор середовища KDE, що відрізняється широкою функціональністю, підтримкою тригонометрії, статистичних функцій та можливістю налаштування кольорової теми. Обидва є безкоштовним відкритим ПЗ, але прив'язані до конкретного Linux-середовища.

SpeedCrunch. Це один із найпотужніших безкоштовних калькуляторів з відкритим вихідним кодом, реалізований також на C++ та Qt. SpeedCrunch підтримує сотні математичних функцій, змінні, автодоповнення, підсвічування синтаксису та побудову графіків. Він є кросплатформним (Windows, Linux, macOS) і позиціонується як альтернатива Wolfram Alpha для складних обчислень. Однак для пересічного користувача інтерфейс SpeedCrunch може здатися надто складним — він орієнтований на досвідчених користувачів.

Десмос (Desmos). Популярний онлайновий графічний калькулятор, орієнтований передусім на побудову та дослідження графіків математичних функцій. Використовується переважно у шкільній та університетській освіті. Вимагає підключення до мережі та не є інструментом для швидких щоденних обчислень.

Мобільні калькулятори (Android, iOS). Системні калькулятори смартфонів є зручними для мобільного використання: оптимізований під сенсорне введення інтерфейс, компактність, завжди під рукою. Проте вони не підходять для роботи на десктопі та, як правило, пропонують лише базовий набір операцій.

Порівняльний аналіз виявив таку ситуацію: існуючі рішення охоплюють два полюси — або надто прості (системні калькулятори з мінімальним набором операцій), або надто складні для звичайного користувача (SpeedCrunch, Wolfram Alpha). При цьому більшість простих рішень є платформо-залежними, а потужні — орієнтованими на фахівців. Розроблений у рамках кваліфікаційної роботи застосунок займає проміжну позицію: він підтримує розширений набір операцій (включно зі степенями та константою π), має сучасний і привабливий темний інтерфейс, є кросплатформним та реалізований на основі відкритого вихідного коду.

Таблиця 1.1 узагальнює результати порівняльного аналізу за ключовими критеріями.

Таблиця 1.1 — Порівняльний аналіз існуючих рішень

Критерій	**Windows Calc**	**SpeedCrunch**	**Wolfram Alpha**	**Розроблений застосунок**
Кросплатформність	Ні	Так	Так (web)	Так
Відкритий код	Ні	Так	Ні	Так
Ділення, множення	Так	Так	Так	Так
Піднесення до степеня	Так	Так	Так	Так
Константа π	Частково	Так	Так	Так
Темна тема	Частково	Ні	Ні	Так
Без Інтернету	Так	Так	Ні	Так

Додатково варто відзначити, що сучасні програмні рішення для обчислень суттєво відрізняються за рівнем орієнтації на користувача. Частина з них спрямована на масового користувача, забезпечуючи максимально простий інтерфейс та мінімальний набір функцій. Інша частина орієнтована на професійне середовище (інженери, математики, програмісти), де ключовими є розширена функціональність, підтримка складних виразів, робота зі змінними та можливість аналітичних обчислень. Такий поділ формує суттєвий розрив між простими та професійними інструментами, що ускладнює вибір оптимального рішення для користувачів із середнім рівнем потреб.

Окремої уваги заслуговує аспект користувацького досвіду (UX/UI). У більшості системних калькуляторів інтерфейс є стандартизованим і не змінюється роками, що забезпечує звичність, але обмежує можливості адаптації під індивідуальні потреби. Натомість більш складні системи, такі як SpeedCrunch або онлайн-сервіси, часто перевантажують інтерфейс додатковими функціями, що знижує зручність для нових користувачів. Таким чином, спостерігається проблема балансу між функціональністю та простотою використання.

Важливим критерієм є також кросплатформність і незалежність від екосистеми. Більшість вбудованих рішень (Windows Calculator, macOS Calculator) жорстко прив’язані до відповідних операційних систем, що обмежує їх використання в гетерогенних середовищах. У той же час відкриті та веборієнтовані рішення частково вирішують цю проблему, однак можуть мати інші обмеження, зокрема залежність від інтернет-з’єднання або складність розгортання.

Ще одним важливим аспектом є можливість розширення функціональності. У ряді розглянутих рішень розширення або відсутнє, або реалізоване через складні механізми плагінів чи сторонніх бібліотек. Це ускладнює адаптацію програмного забезпечення під специфічні потреби користувачів або освітні завдання, де часто необхідно додавати нові математичні операції або режими обчислень.

Отже, проведений аналіз показує, що на ринку програмних калькуляторів відсутнє універсальне рішення, яке б одночасно поєднувало простоту використання, розширену функціональність, кросплатформність та сучасний інтерфейс. Саме ця невідповідність між існуючими рішеннями та потребами користувачів і стала основним обґрунтуванням для розробки власного програмного продукту в межах кваліфікаційної роботи.

2. ПРОЄКТУВАННЯ ТА ІНСТРУМЕНТАЛЬНІ ЗАСОБИ РЕАЛІЗАЦІЇ ПРОЄКТУ

2.1. Аналіз мов програмування та інструментів для реалізації проєкту

Вибір технологічного стеку є одним із найважливіших рішень на етапі проєктування будь-якого програмного забезпечення. Від цього рішення залежать продуктивність готового застосунку, складність розробки, кросплатформність, доступна екосистема бібліотек і фреймворків, а також довгостроковий потенціал розвитку проєкту. У даному розділі розглянуто основні альтернативи та обґрунтовано зроблений вибір.

Мова програмування C++. C++ — об'єктно-орієнтована мова програмування загального призначення, що є розширенням мови C. Стандартизована ISO (остання актуальна версія — C++23), мова постійно розвивається і залишається одним із лідерів у промисловій розробці. У даному проєкті використовується стандарт C++17, що забезпечує доступ до сучасних можливостей мови: auto, structured bindings, std::optional, lambda-виразів та інших.

Ключові переваги C++ для даного проєкту [16]:

– висока продуктивність: C++ компілюється безпосередньо в машинний код без проміжного інтерпретатора або JVM, що забезпечує максимальну швидкість виконання обчислень;

– пряма інтеграція з Qt: Qt Framework написаний мовою C++ і надає найбільш повний та зрілий API саме для C++;

– контроль над пам'яттю: C++ дозволяє точно керувати виділенням і звільненням пам'яті, хоча Qt також надає власну систему управління пам'яттю через механізм батьківських об'єктів;

– широка екосистема: STL (Standard Template Library) надає готові реалізації стеків, черг, векторів та інших структур даних, використаних у проєкті;

– кросплатформність: C++ код компілюється на всіх основних платформах стандартними компіляторами GCC, Clang та MSVC.

Альтернативні мови програмування та причини їх відхилення:

Python + PyQt/PySide. Python є надзвичайно популярною мовою завдяки простоті синтаксису та швидкості розробки. PyQt та PySide надають Python-обгортки для Qt API. Проте для десктопного застосунку, де важлива продуктивність та відсутність залежностей від інтерпретатора, C++ є кращим вибором. Python-застосунки потребують встановленого інтерпретатора Python, мають помітно більші витрати пам'яті та нижчу швидкість виконання у порівнянні з нативним C++.

Java + JavaFX. Java забезпечує кросплатформність через JVM, проте потребує встановленого JRE (Java Runtime Environment), що є суттєвою залежністю. JavaFX поступово зменшує свою присутність у ринку, а сама платформа асоціюється з корпоративними застосунками, а не з легкими десктопними утилітами. Крім того, Qt є більш зрілим рішенням для розробки GUI-застосунків на C++.

C# + WPF/WinForms. C# та технології Microsoft (WPF, WinForms, MAUI) є потужним стеком для Windows-розробки, проте він обмежений платформою Windows (або вимагає Mono/Wine для Linux/macOS). Оскільки кросплатформність є однією з ключових вимог проєкту, цей стек відхилено.

Qt Framework — вибір та обґрунтування. Qt Framework є зрілим і потужним інструментом для розробки кросплатформних GUI-застосунків. Розроблявся з 1991 року компанією Trolltech, пізніше Nokia, а зараз підтримується The Qt Company. Qt підтримує операційні системи Windows, Linux, macOS, Android, iOS, а також embedded-системи. Ліцензується як комерційно, так і за відкритими ліцензіями LGPL v3 та GPL v3 [12].

Ключові можливості Qt, що використані у проєкті:

– QMainWindow — базовий клас для головних вікон з підтримкою меню, панелей інструментів і рядка стану;

– QLineEdit — поле однорядкового введення тексту для дисплея калькулятора;

– QPushButton — кнопки інтерфейсу з підтримкою кастомних стилів CSS;

– QGridLayout — сіткове розміщення кнопок у регулярній матриці 4×6;

– QVBoxLayout — вертикальне розміщення основних секцій інтерфейсу;

– QLabel — мітка для відображення рядка «історії» обчислень;

– Механізм сигналів і слотів — для з'єднання кнопок з обробниками подій;

– QSS (Qt Style Sheets) — CSS-подібна мова для стилізації UI-елементів;

– keyPressEvent — перевизначення обробника клавіатурних подій.

У проєкті використано Qt 6 — найновіша мажорна версія фреймворку. Qt 6 запропонував значно покращену систему типів, нативну підтримку HiDPI-дисплеїв, оновлений рушій QML та ряд інших покращень порівняно з Qt 5.

Тут треба згадати про meta-object систему Qt. Це одна з головних фішок Qt. Утиліта moc (Meta-Object Compiler) бере заголовки з макросом Q_OBJECT і сама створює C++ код. Цей C++ код запускає сигнали і слоти, додає динамічні властивості, дає рефлексію і забезпечує переклад інтерфейсу. Саме завдяки цьому механізму з’єднання кнопок з обробниками подій проєкт виглядає простим і зрозумілим. Рядок connect(button, &QPushButton::clicked, this, &MainWindow::onDigitClicked) описує зв’язок між подією і реакцією без шаблонного коду.

Підбиваючи підсумок вибору технологій, C++17 дає швидку роботу коду, а C++17 разом з Qt 6 працює найкраще для нашого проєкту. Qt 6 вже давно стабільний, Qt 6 підтримує кросплатформність, QSS дає багато способів оформити інтерфейс, і QSS легко налаштовується. Підтримка на різних системах дозволяє запускати програму скрізь. Все це разом створює міцну основу. Ця основа допомагає зараз працювати застосунку і дозволяє легко додавати нові функції в майбутньому.

2.2. Інші допоміжні інструменти

Поряд з основним технологічним стеком (C++ та Qt), у процесі розробки проєкту було задіяно ряд допоміжних інструментів, що забезпечують ефективну організацію робочого процесу та управління збіркою.

Qt Creator — основне середовище розробки. Qt Creator є інтегрованим середовищем розробки (IDE), спеціально розробленим компанією The Qt Company для роботи з Qt-проєктами. Qt Creator надає всі необхідні інструменти для комфортної роботи над C++/Qt-проєктами в єдиному середовищі.

Основні можливості Qt Creator, що використовувались у даному проєкті [11]:

– Редактор коду з підсвіченням синтаксису C++, автодоповненням на основі clangd, навігацією за кодом (переходом до визначення, пошуком усіх використань) та вбудованою документацією Qt;

– Вбудована підтримка системи збірки qmake, що дозволяє редагувати файл calculator.pro та збирати проєкт одним натисканням кнопки;

– Вбудований відладчик (GDB на Linux/macOS, CDB/MSVC на Windows) для покрокового виконання коду та огляду значень змінних;

– Менеджер комплектів (Kit Manager), що автоматично знаходить встановлені версії Qt та компіляторів і дозволяє перемикатись між ними;

– Вбудований профайлер Valgrind Memcheck для виявлення витоків пам'яті (на Linux);

– Підтримка системи контролю версій Git безпосередньо з інтерфейсу IDE.

qmake — система збірки. У проєкті використовується qmake — традиційна система збірки Qt-проєктів, конфігурація якої описується у файлі calculator.pro. На відміну від CMake, qmake є більш простим та лаконічним рішенням для невеликих Qt-проєктів: файл .pro автоматично налаштовує всі необхідні параметри збірки, підключає бібліотеки Qt та генерує нативні Makefile для компілятора.

У файлі calculator.pro визначено такі параметри:

– QT += core gui widgets — підключення модулів Qt, необхідних для GUI-застосунку (core — базовий модуль, gui — графічний рівень, widgets — класи QMainWindow, QPushButton тощо);

– CONFIG += c++17 — використання стандарту C++17;

– SOURCES, HEADERS — перелік файлів вихідного коду та заголовних файлів;

– DEFINES += QT_DEPRECATED_WARNINGS — вмикання попереджень при використанні застарілого Qt API.

MinGW (Minimalist GNU for Windows). На платформі Windows проєкт збирається компілятором MinGW (GCC-порт для Windows), включеним до складу Qt-дистрибутива. Версія, використана при розробці: Desktop Qt 6.11.0 MinGW 64-bit (про що свідчить назва директорії збірки Desktop_Qt_6_11_0_MinGW_64_bit-Debug у складі архіву). MinGW забезпечує повну підтримку C++17 та стандартної бібліотеки STL.

STL (Standard Template Library). Стандартна бібліотека шаблонів C++ використовується у математичному ядрі застосунку. Зокрема, застосовуються:

– std::stack<Leksema> — стек для реалізації алгоритму двох стеків (стек операндів та стек операторів);

– std::string — рядковий тип для роботи з виразом у методі evaluate;

– std::stringstream (із заголовного файлу <sstream>) — потік рядка для посимвольного читання виразу та парсингу чисел;

– std::pow (із <cmath>) — стандартна функція піднесення до степеня;

– acos(-1.0) (із <cmath>) — обчислення значення константи π з максимальною точністю double.

Git — система контролю версій. Для відстеження змін у вихідному коді та організації версіонування проєкту використовується Git. Система контролю версій є стандартом де-факто у сучасній розробці програмного забезпечення та дозволяє повертатись до попередніх версій коду, відслідковувати всі зміни та організовувати командну роботу.

QSS (Qt Style Sheets) — мова стилізації інтерфейсу. Qt Style Sheets є підмножиною CSS, адаптованою для стилізації Qt-віджетів. У проєкті QSS активно використовується для реалізації темного дизайну інтерфейсу: кожна кнопка отримує індивідуальні або групові стилі, що визначають колір фону, колір тексту, радіус заокруглення, а також стилі для стану hover (при наведенні курсора) та pressed (при натисканні). Це дозволяє досягти привабливого зовнішнього вигляду без використання зовнішніх ресурсів або зображень.

2.3. Проєктування архітектури та основні користувацькі сценарії програмного забезпечення для здійснення обчислень

Архітектура розробленого застосунку базується на стандартній Qt-моделі «головне вікно — центральний віджет — компоновники», адаптованій під потреби калькулятора. Оскільки застосунок є відносно невеликим і однофункціональним, архітектура залишається простою і легко зрозумілою.

Структура класів. Застосунок містить один основний клас — MainWindow, що успадковує QMainWindow. Усередині класу визначені:

– Поля-вказівники на UI-елементи: display (QLineEdit*) для відображення виразу та результату, historyLabel (QLabel*) для відображення попереднього виразу;
– Приватні структура Leksema (char type; double value) — тип токена для математичного ядра;
– Приватні методи-слоти: onDigitClicked(), onOperatorClicked(), onEqualsClicked(), onClearAll(), onBackspace(), onDotClicked(), onPiClicked();
– Допоміжні методи: appendToDisplay(QString), getRang(char), maths(Stack&, Stack&, Leksema&), evaluate(QString, bool&);
– Перевизначений обробник keyPressEvent(QKeyEvent*) для підтримки клавіатурного введення.

Функція main (файл main.cpp) є точкою входу. Вона: ініціалізує об'єкт QApplication (що керує подіями та ресурсами застосунку), встановлює ім'я програми (setApplicationName("Калькулятор")), створює об'єкт MainWindow та відображає його (show()), і запускає цикл обробки подій (a.exec()).

Компоновники (Layouts). Інтерфейс організований за допомогою двох вкладених компоновників:

– QVBoxLayout (mainLayout) — вертикальний компоновник верхнього рівня, що розміщує по вертикалі: рядок historyLabel, поле display, та сітку кнопок;

– QGridLayout (grid) — сіткова матриця 6 рядків × 4 стовпці для розміщення 19 кнопок (кнопка «0» займає 2 стовпці, тому загальна кількість клітинок — 23, але фактично використано 22).

Механізм сигналів і слотів. Qt використовує механізм сигналів і слотів як основний спосіб реакції на події та комунікації між об'єктами. У проєкті:

– Кожна цифрова кнопка (btn0..btn9) з'єднана сигналом clicked() зі слотом onDigitClicked(). Метод qobject_cast<QPushButton*>(sender()) дозволяє отримати вказівник на кнопку, що надіслала сигнал, та зчитати її відображуваний текст.

– Кнопки операторів (+, −, ×, ÷, ^, (,)) з'єднані зі слотом onOperatorClicked(). Замість відображуваних символів (наприклад, «×», «÷», «−») у вираз додається ASCII-символ із objectName кнопки («*», «/», «-»), що спрощує подальший парсинг.

– Кнопки C, ⌫, «.», «π» та «=» з'єднані з відповідними індивідуальними слотами.

Алгоритм двох стеків (evaluate). Математичне ядро реалізує спрощений варіант алгоритму сортувальної станції Дейкстри без явного перетворення у RPN: обчислення виконується «на льоту» за допомогою двох стеків — Stack_n (операнди) та Stack_o (оператори). Алгоритм обробляє вхідний рядок посимвольно через std::stringstream:

– Пробіли ігноруються;

– Символ 'p' (підстановка π) → число PI_VAL штовхається у Stack_n;

– Число або унарний мінус → парситься через оператор >> та штовхається у Stack_n;

– Оператор (+, -, *, /, ^) → якщо Stack_o порожній або пріоритет оператора вищий за вершину Stack_o, оператор штовхається у Stack_o; інакше спочатку виконуються операції з вершини Stack_o (метод maths), поки пріоритет не стане нижчим;

– «(» → штовхається у Stack_o;

– «)» → виконуються операції з Stack_o до «(», потім «(» видаляється;

– Невідомий символ → повертається 0, ok=false;

– Після читання всього рядка виконуються всі операції, що залишились у Stack_o;

– Результат — значення на вершині Stack_n.

Метод maths виконує одну бінарну операцію: знімає два операнди з Stack_n та один оператор з Stack_o, обчислює результат і кладе його назад у Stack_n. Перевіряється ділення на нуль: якщо ділник дорівнює 0.0, метод повертає false, що призводить до відображення «Помилка».

Функція getRang визначає пріоритет оператора: '^' → 3, '*' і '/' → 2, '+' і '-' → 1, решта → 0. Це стандартна математична ієрархія операцій.

Основні користувацькі сценарії (Use Cases):

UC-1: Обчислення простого виразу. Передумова: запущений застосунок, поле дисплея містить «0». Кроки: 1) Користувач натискає «7»; 2) Натискає «+»; 3) Натискає «3»; 4) Натискає «=». Результат: дисплей показує «10», рядок історії — «7+3 =".

UC-2: Обчислення складного виразу з дужками. Кроки: 1) Вводить «(2+3)*4»; 2) Натискає «=». Результат: дисплей показує «20».

UC-3: Використання константи π. Кроки: 1) Вводить вираз; 2) Натискає «π»; 3) продовжує вираз; 4) Натискає «=». Результат: π підставляється як значення ≈ 3.14159265359.

UC-4: Введення некоректного виразу. Кроки: 1) Вводить «2++3»; 2) Натискає «=». Результат: дисплей показує «Помилка».

UC-5: Ділення на нуль. Кроки: 1) Вводить «5/0»; 2) Натискає «=». Результат: дисплей показує «Помилка» (перевірка у методі maths).

UC-6: Клавіатурне введення. Кроки: 1) Вводить вираз з клавіатури; 2) Натискає Enter. Результат: викликається onEqualsClicked(), аналогічно натисканню кнопки «=».

3. РЕАЛІЗАЦІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ЗДІЙСНЕННЯ ОБЧИСЛЕНЬ

3.1. Реалізація програми для здійснення обчислень

У даному підрозділі детально описана реалізація логічного ядра застосунку — тих частин коду, що безпосередньо відповідають за обчислення математичних виразів та обробку введення. Розглянемо кожен файл проєкту.

Файл main.cpp є точкою входу застосунку. Він є лаконічним і виконує мінімально необхідні дії для запуску Qt-застосунку:

```
#include "mainwindow.h"
#include <QApplication>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    a.setApplicationName("Калькулятор");
    MainWindow w;
    w.show();
    return a.exec();
}
```

Лістинг 3.1 — Файл main.cpp

Об'єкт QApplication є центральним об'єктом будь-якого Qt-застосунку: він ініціалізує підсистеми Qt, налаштовує параметри локалі, керує чергою подій та забезпечує коректне завершення програми. Метод a.exec() запускає нескінченний цикл обробки подій, з якого програма виходить лише після закриття вікна.

Файл mainwindow.h визначає структуру класу MainWindow. Розглянемо ключові елементи оголошення:

```
#include <QMainWindow>
#include <QLineEdit>
#include <QPushButton>
#include <QGridLayout>
#include <QVBoxLayout>
#include <QLabel>
#include <stack>
#include <string>
```

```
class MainWindow : public QMainWindow
{
    Q_OBJECT
public:
    MainWindow(QWidget *parent = nullptr);
    ~MainWindow();
protected:
    void keyPressEvent(QKeyEvent *event) override;
private slots:
    void onDigitClicked();    void onOperatorClicked();
    void onEqualsClicked();   void onClearAll();
    void onBackspace();       void onDotClicked();
    void onPiClicked();
private:
    QLineEdit *display;
    QLabel    *historyLabel;
    struct Leksema { char type; double value; };
    bool   maths(std::stack<Leksema>&, std::stack<Leksema>&, Leksema&);
    int    getRang(char ch);
    double evaluate(const QString &expr, bool &ok);
    void   appendToDisplay(const QString &text);
};
```

Лістинг 3.2 — Оголошення класу MainWindow (mainwindow.h)

Макрос Q_OBJECT є обов'язковим для будь-якого класу Qt, що оголошує сигнали або слоти. Він активує механізм Meta-Object System, що є основою для роботи сигналів/слотів, rtti (runtime type information в Qt) та ряду інших механізмів фреймворку. При компіляції система AUTOMOC автоматично запускає утиліту moc (Meta-Object Compiler), що генерує відповідний C++-код на основі Q_OBJECT-класів.

Реалізація математичного ядра — метод evaluate. Метод evaluate є серцем застосунку: він отримує рядок виразу (expr), парсить його та обчислює результат. Розглянемо ключові фрагменти реалізації:

```
double MainWindow::evaluate(const QString &expr, bool &ok)
{
    ok = false;
    std::string str = expr.toStdString();
    std::stringstream sstr(str);
```

```
    char ch; double value; bool flag = true;
    std::stack<Leksema> Stack_n;  // стек операндів
    std::stack<Leksema> Stack_o;  // стек операторів
    Leksema item;
    while (true) {
        ch = (char)sstr.peek();
        if (ch == '\377' || sstr.eof()) break;
        if (ch == ' ') { sstr.ignore(); continue; }
        if (ch == 'p') { /* вставляємо п */ ... }
        if ((ch >= '0' && ch <= '9') || (ch == '-' && flag) || ch == '.')
        { sstr >> value; Stack_n.push({'0', value}); flag = false;
continue; }
    }
    while (!Stack_o.empty()) { if (!maths(Stack_n, Stack_o, item)) return
0; }
    if (Stack_n.empty()) return 0;
    ok = true;
    return Stack_n.top().value;
}
```

Лістинг 3.3 — Спрощений вигляд методу evaluate

Прапорець flag відіграє важливу роль: він дозволяє відрізняти унарний мінус (знак числа, наприклад «-5») від бінарного мінуса (операція віднімання, наприклад «3-5»). Якщо flag=true (очікується нове число), то «-» трактується як частина числа і читається оператором >> разом із цифрами. Якщо flag=false (щойно прочитано число), «-» є оператором.

Метод maths реалізує виконання однієї бінарної операції:

```
bool MainWindow::maths(std::stack<Leksema> &Stack_n,
                       std::stack<Leksema> &Stack_o,
                       Leksema             &item)
{
    if (Stack_n.empty() || Stack_o.empty()) return false;
    double a = Stack_n.top().value; Stack_n.pop();
    char op  = Stack_o.top().type;  Stack_o.pop();
    if (Stack_n.empty()) return false;
    double b = Stack_n.top().value; Stack_n.pop();
    double c = 0;
    switch (op) {
    case '+': c = b + a; break;
    case '-': c = b - a; break;
```

```
        case '*': c = b * a; break;
        case '^': c = std::pow(b, a); break;
        case '/':
            if (a == 0.0) return false;  // ділення на нуль
            c = b / a; break;
        default: return false;
        }
        Stack_n.push({'0', c});
        return true;
    }
```

Лістинг 3.4 — Метод maths

Важливо зауважити порядок зняття операндів зі стека: спочатку знімається «a» (правий операнд), потім «b» (лівий операнд). Це критично для некомутативних операцій — віднімання (b − a), ділення (b / a) та піднесення до степеня (b ^ a). Неправильний порядок призвів би до хибних результатів.

3.2. Створення інтерфейсу

Графічний інтерфейс розробленого застосунку реалізовано програмно у конструкторі класу MainWindow, без використання Qt Designer та .ui-файлів. Такий підхід надає повний контроль над кожним аспектом інтерфейсу та дозволяє використовувати лямбда-функції та інші можливості C++ для генерації елементів.

Загальна структура компоновки. Усі елементи розміщуються за допомогою двох вкладених компоновників. QVBoxLayout (mainLayout) є зовнішнім вертикальним компоновником, що містить рядок historyLabel, поле display та сітку кнопок (QGridLayout). QGridLayout (grid) являє собою матрицю розміром 6×4, у якій розміщено 19 кнопок.

Налаштування дисплея. Поле QLineEdit виконує подвійну функцію: воно є і полем введення (користувач може редагувати вираз безпосередньо), і полем відображення результату:

```
display = new QLineEdit("0", this);
display->setAlignment(Qt::AlignRight);
display->setReadOnly(false);
display->setFont(QFont("Consolas", 28, QFont::Bold));
```

```
display->setFixedHeight(70);
display->setStyleSheet(
    "QLineEdit {"
    "  background: #1e1e2e; color: #cdd6f4;"
    "  border: 2px solid #313244;"
    "  border-radius: 10px; padding: 4px 12px;"
    "}"
);
```

Лістинг 3.5 — Налаштування поля дисплея

Великий розмір шрифту Consolas 28pt забезпечує гарну читабельність введеного виразу та результату. Вирівнювання по правому краю є традиційним для калькуляторів і відповідає звичним очікуванням користувачів. Початкове значення «0» — також стандартне для калькуляторів.

Лямбда-функція makeBtn. Для уникнення дублювання коду при створенні кнопок визначено лямбда-функцію makeBtn, що приймає текст кнопки та кольори фону і тексту:

```
auto makeBtn = [&](const QString &text,
                   const QString &bg,
                   const QString &fg = "#cdd6f4") -> QPushButton*
{
    QPushButton *btn = new QPushButton(text, this);
    btn->setFixedHeight(62);
    btn->setSizePolicy(QSizePolicy::Expanding, QSizePolicy::Fixed);
    btn->setFont(QFont("Consolas", 18, QFont::Bold));
    QString style = QString(
        "QPushButton { background: %1; color: %2;"
        " border: none; border-radius: 10px; }"
        "QPushButton:hover { background: #585b70; }"
        "QPushButton:pressed { background: #313244; }"
    ).arg(bg, fg);
    btn->setStyleSheet(style);
    return btn;
};
```

Лістинг 3.6 — Лямбда-функція makeBtn

Колірна схема інтерфейсу. Інтерфейс реалізовано у стилі темної теми на основі кольорової палітри Catppuccin Mocha. Ця палітра була обрана завдяки м'яким контрастам та приємним для ока відтінкам:

– Фон вікна: #181825 (глибокий темно-синій).

– Фон дисплея: #1e1e2e (трохи світліший темно-синій).

– Цифрові кнопки: #313244 (темно-сірий).

– Оператори (+, −, ×, ÷): #89b4fa (м'який синій).

– Кнопка «=»: #a6e3a1 з темним текстом (м'який зелений).

– Кнопка «C»: #f38ba8 з темним текстом (м'який рожевий/червоний).

– Кнопка «⌫»: #fab387 (м'який помаранчевий).

– Кнопка «π»: #cba6f7 (м'який фіолетовий).

– Стан hover всіх кнопок: #585b70 (трохи світліший від основного фону).

Такий колірний кодинг дозволяє користувачу миттєво орієнтуватися в інтерфейсі: цифри, оператори, спеціальні кнопки та кнопка результату візуально відрізняються одна від одної, що зменшує ймовірність помилкового натискання.

Розміщення кнопок у сітці. Матриця кнопок організована у 6 рядків і 4 стовпці відповідно до традиційного компактного розкладу калькулятора:

– Рядок 0 (функціональний): C, ⌫, (,).

– Рядок 1 (розширені операції): π, ^, ÷, ×.

– Рядки 2–4 (цифри і арифметика): 7-8-9-−, 4-5-6-+, 1-2-3-=.

– Рядок 5 (нуль та крапка): 0 (займає 2 клітинки), ., (четверта клітинка — порожня).

Підтримка клавіатурного введення. Метод keyPressEvent перевизначений для обробки клавіш Enter/Return (виклик onEqualsClicked), Escape (виклик onClearAll) та Backspace (виклик onBackspace). Введення цифр та операторів безпосередньо з клавіатури також підтримується через поле display, яке є редагованим (ReadOnly = false).

Рядок «Історії» (historyLabel). Мітка QLabel розміщена над полем дисплея та відображає вираз поточного обчислення після натискання «=». Наприклад, якщо користувач обчислив «7+3», після натискання «=» у рядку з'явиться «7+3 =», а на дисплеї — «10». Це дозволяє перевірити правильність введеного виразу. Рядок історії очищується при натисканні «C».

3.3. Огляд роботи програми для здійснення обчислень

У цьому підрозділі наведено результати функціонального тестування застосунку та загальний огляд його роботи в різних сценаріях використання.

Запуск застосунку. При запуску відкривається головне вікно з мінімальним розміром 320×480 пікселів. Вікно є масштабованим: при збільшенні розміру кнопки та поле дисплея пропорційно збільшуються завдяки використанню компоновників та SizePolicy::Expanding. Фон вікна — глибокий темно-синій (#181825). Поле дисплея відображає початкове значення «0».



Рисунок 3.1. - Поле дисплея та клавіатура

Введення виразу. При натисканні цифрових кнопок символи додаються до поля дисплея. Якщо поточне значення — «0» і натиснуто цифру (не 0), значення «0» замінюється введеною цифрою (метод appendToDisplay). Це відповідає стандартній поведінці калькуляторів. Оператори (+, −, ×, ÷, ^) також додаються до рядка виразу.

Результати тестових обчислень. Проведено тестування на наборі тестових виразів для перевірки коректності алгоритму.

Таблиця 3.1 — Результати тестування обчислень

Введений вираз	**Очікуваний результат**	**Результат програми**
2+3	5	5
10-4*2	2	2
(10-4)*2	12	12
2^10	1024	1024
p	3.14159265359	3.14159265359
2*p	6.28318530718	6.28318530718
10/4	2.5	2.5
5/0	Помилка	Помилка
2++3	Помилка	Помилка
(3+4	Помилка	Помилка



Рисунок 3.2. - Результати тестових обчислень

Як видно з таблиці 3.1, усі тестові випадки пройдено успішно: коректні вирази обчислюються правильно, некоректні вирази (подвійний оператор, незачинена дужка) та ділення на нуль призводять до відображення «Помилка» без аварійного завершення програми.

Обробка крайових випадків. Розглянемо кілька важливих крайових випадків:

– Від'ємні числа: вираз «-5+3» коректно обчислюється як −2 завдяки обробці унарного мінуса (flag=true на початку виразу).

– Від'ємні числа в дужках: «(-5)*2» = −10 — унарний мінус розпізнається після «(», оскільки flag=true після відкриваючої дужки.

– Десяткові дроби: «1.5+2.7» = 4.2 — коректно зчитується через оператор >>.

– Піднесення у від'ємний степінь: «2^-1» = 0.5.

– Вираз із π: «p*2» ≈ 6.283 — значення π підставляється як константа acos(-1.0) з точністю double (~15 значущих цифр).

Виведення результату. Результат відображається у полі дисплея у форматі 'g' з 12 значущими цифрами (QString::number(result, 'g', 12)). Формат 'g' автоматично обирає між звичайним (3.5) та науковим (1.5e+10) записом залежно від величини числа, що є зручним для загального використання.

Адаптивний розмір вікна. Завдяки використанню компоновників та setSizePolicy(QSizePolicy::Expanding, QSizePolicy::Fixed) для кнопок, інтерфейс



коректно масштабується: при розтягуванні вікна по горизонталі всі чотири стовпці кнопок рівномірно розширюються, заповнюючи доступний простір.

Рисунок 3.3. – Масштабування інтерфейсу при розтягуванні вікна

Клавіатурна навігація. Підтримка клавіатурного введення значно підвищує зручність використання для досвідчених користувачів. Вираз можна ввести повністю з клавіатури і натиснути Enter для отримання результату, не використовуючи мишу. Це робить застосунок зручним у поєднанні з іншими програмами — наприклад, для швидкої перевірки обчислень під час роботи в текстовому редакторі.

Кросплатформне тестування. Проєкт успішно збирається та виконується на Windows 10/11 з використанням Qt 6.11.0 MinGW 64-bit. Оскільки код використовує виключно стандартні класи Qt та STL без платформо-специфічних викликів, збірка та виконання на Linux та macOS (з відповідно встановленими версіями Qt 6) також не мають потребувати жодних змін у вихідному коді.

ВИСНОВКИ

У кваліфікаційній роботі виконано повний цикл розробки програмного забезпечення для здійснення обчислень — від аналізу предметної галузі та постановки задачі до проєктування, реалізації та тестування готового застосунку. Результати роботи дозволяють зробити такі висновки.

1. Проведено аналіз предметної галузі програмного забезпечення для обчислень. Визначено та охарактеризовано основні поняття: математичний вираз, операнд, оператор, пріоритет операцій, парсинг, лексема, стек, зворотний польський запис. Описано класифікацію ПЗ для обчислень за типом операцій та платформою розміщення.

2. Виконано порівняльний аналіз існуючих рішень: вбудованих калькуляторів Windows/macOS/Linux, SpeedCrunch, Wolfram Alpha, Desmos та мобільних застосунків. Виявлено, що більшість простих рішень є платформо-залежними, а потужні — орієнтованими на фахівців. Обґрунтовано доцільність розробки власного кросплатформного застосунку із сучасним темним інтерфейсом.

3. Обґрунтовано вибір технологічного стеку: мова C++ стандарту C++17, Qt 6 Framework, система збірки qmake, IDE Qt Creator та компілятор MinGW. Проаналізовано альтернативи (Python+PyQt, Java+JavaFX, C#+WPF) та встановлено переваги обраного стеку за критеріями продуктивності, кросплатформності та доступності Qt API.

4. Спроєктовано архітектуру застосунку з центральним класом MainWindow, визначено структуру компоновників (QVBoxLayout + QGridLayout), описано механізм сигналів і слотів для з'єднання кнопок з обробниками. Розроблено шість основних користувацьких сценаріїв використання.

5. Реалізовано математичне ядро застосунку на основі алгоритму двох стеків — спрощеного варіанту алгоритму сортувальної станції Дейкстри. Алгоритм коректно обробляє: арифметичні операції (+, −, ×, ÷), піднесення до

степеня (^), дужки, унарний мінус, десяткові дроби, константу π та всі крайові випадки (ділення на нуль, некоректні вирази).

6. Розроблено графічний інтерфейс у сучасному темному стилі з використанням кольорової палітри Catppuccin Mocha. Реалізовано 19 кнопок у матриці 6×4, поле дисплея з великим шрифтом, рядок «Історії» та підтримку клавіатурного введення. Колірний кодинг кнопок (цифри, оператори, результат, функції) підвищує інтуїтивність інтерфейсу.

7. Проведено функціональне тестування на 10 тестових виразах, що охоплюють базові операції, пріоритети, дужки, степені, π, десяткові дроби та крайові випадки. Всі тести пройдено успішно: коректні вирази обчислюються правильно, некоректні — не призводять до аварійного завершення програми.

8. Застосунок є кросплатформним: вихідний код не містить платформо-специфічних викликів і може бути скомпільований для Windows, Linux та macOS без змін.

Перспективами подальшого розвитку проєкту є: розширення підтримуваних операцій (sin, cos, tan, log, sqrt, факторіал), додавання можливості збереження й завантаження виразів, реалізація повноцінної «Историї» з прокруткою, розробка режиму програмування (побітові операції, системи числення), а також оптимізація для мобільних платформ на базі того самого Qt-коду.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. C++ reference. URL: https://en.cppreference.com/w/

2. Deitel P., Deitel H. C++20 for Programmers: An Objects-Natural Approach. 3rd ed. Boston : Pearson, 2021. 1012 p.

3. GeeksforGeeks. Expression Evaluation. URL: https://www.geeksforgeeks.org/expression-evaluation/

4. ISO/IEC 14882:2020 Programming Languages — C++. URL: https://isocpp.org/std/the-standard

5. Meyers S. Effective Modern C++. Sebastopol : O'Reilly Media, 2014. 334 p. URL: https://www.oreilly.com/library/view/effective-modern-c/9781491908419/

6. Microsoft C++ Docs. URL: https://learn.microsoft.com/en-us/cpp/

7. Open Source Guide. URL: https://opensource.guide/

8. Oracle C++ Documentation. URL: https://docs.oracle.com/en/cpp/

9. Об'єктно-Орієнтоване Програмування (ООП). Пояснюємо чотири основні принципи [Електронний ресурс] — Режим доступу : https://career.softserveinc.com/uk-ua/stories/what-is-object-oriented-programming-oop-explaining-four-major-principles

10. QStack Class Documentation. URL: https://doc.qt.io/qt-6/qstack.html

11. Qt Creator Manual. URL: https://doc.qt.io/qtcreator/

12. Qt Documentation. Qt 6 Documentation. URL: https://doc.qt.io/qt-6/

13. Qt Style Sheets Reference. URL: https://doc.qt.io/qt-6/stylesheet-reference.html

14. Software Engineering Institute. URL: https://www.sei.cmu.edu/

15. TutorialsPoint C++ Tutorial. URL: https://www.tutorialspoint.com/cplusplus/index.htm

16. Жуковський С. С., Вакалюк Т. А. Об'єктно-орієнтоване програмування мовою C++ : навч. посіб. Житомир : Вид-во ЖДУ ім. І. Франка, 2020. 118 с. [Електронний ресурс] — Режим доступу : https://duikt.edu.ua/ua/lib/1/category/2228/view/2138

17. Основи програмування на C++ : навч. посіб. / В. М. Ткачов та ін. Харків : НТУ «ХПІ», 2022. 194 с. [Електронний ресурс] — Режим доступу : https://repository.kpi.kharkov.ua/entities/publication/83caee9f-e294-4f64-b79d-79f20824076c

18. Основи програмування на C/C++ в прикладах. Частина 1: навч.-метод. посібник / Соболь М.О., Любченко Н.Ю, Паржин Ю.В., Пугачов Р.В. – Харків : НТУ "ХПІ", 2021. – 113 с

19. Про Програмування | ABout IT And Programming. Коротко про Gradle [Електронний ресурс] — Режим доступу : https://abitap.com/16-4-korotko-pro-gradle/

20. Ришковець Ю. В., Висоцька В. А. Алгоритмізація та програмування. Частина 1: навчальний посібник. Львів : Видавництво «Новий Світ-2000», 2020. 320 с.

21. Ришковець Ю. В., Висоцька В. А. Алгоритмізація та програмування. Частина 2: навчальний посібник. Львів : Видавництво «Новий Світ-2000», 2020. 337 с

ДОДАТКИ

Додаток А. Повний лістинг програмного коду

А.1. Файл calculator.pro

```
QT       += core gui widgets
greaterThan(QT_MAJOR_VERSION, 4): QT += widgets
CONFIG += c++17
SOURCES += \
    main.cpp \
    mainwindow.cpp
HEADERS += \
    mainwindow.h
DEFINES += QT_DEPRECATED_WARNINGS
```

А.2. Файл main.cpp

```
#include "mainwindow.h"
#include <QApplication>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    a.setApplicationName("Калькулятор");
    MainWindow w;
    w.show();
    return a.exec();
}
```

А.3. Файл mainwindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QLineEdit>
#include <QPushButton>
#include <QGridLayout>
#include <QVBoxLayout>
#include <QLabel>
#include <stack>
#include <string>

class MainWindow : public QMainWindow
{
    Q_OBJECT
public:
    MainWindow(QWidget *parent = nullptr);
    ~MainWindow();
protected:
    void keyPressEvent(QKeyEvent *event) override;
private slots:
    void onDigitClicked();
```

```
    void onOperatorClicked();
    void onEqualsClicked();
    void onClearAll();
    void onBackspace();
    void onDotClicked();
    void onPiClicked();
private:
    QLineEdit *display;
    QLabel    *historyLabel;
    struct Leksema { char type; double value; };
    bool   maths(std::stack<Leksema>&, std::stack<Leksema>&, Leksema&);
    int    getRang(char ch);
    double evaluate(const QString &expr, bool &ok);
    void   appendToDisplay(const QString &text);
};
#endif // MAINWINDOW_H
```

A.4. Файл mainwindow.cpp (повний лістинг)

```
#include "mainwindow.h"
#include <QKeyEvent>
#include <QFont>
#include <QSizePolicy>
#include <cmath>
#include <sstream>

const double PI_VAL = acos(-1.0);

MainWindow::MainWindow(QWidget *parent) : QMainWindow(parent)
{
    setWindowTitle("Калькулятор");
    setMinimumSize(320, 480);
    QWidget *central = new QWidget(this);
    setCentralWidget(central);
    QVBoxLayout *mainLayout = new QVBoxLayout(central);
    mainLayout->setSpacing(8);
    mainLayout->setContentsMargins(12, 12, 12, 12);

    historyLabel = new QLabel("", this);
    historyLabel->setAlignment(Qt::AlignRight | Qt::AlignVCenter);
    historyLabel->setStyleSheet("color: #888; font-size: 13px;");
    historyLabel->setFixedHeight(20);
    mainLayout->addWidget(historyLabel);

    display = new QLineEdit("0", this);
    display->setAlignment(Qt::AlignRight);
    display->setReadOnly(false);
    display->setFont(QFont("Consolas", 28, QFont::Bold));
    display->setFixedHeight(70);
    display->setStyleSheet("QLineEdit { background:#1e1e2e; color:#cdd6f4;"
        " border:2px solid #313244; border-radius:10px; padding:4px
12px;}");
    mainLayout->addWidget(display);

    QGridLayout *grid = new QGridLayout();
```

```
    grid->setSpacing(8);

    auto makeBtn = [&](const QString &text, const QString &bg,
                       const QString &fg = "#cdd6f4") -> QPushButton* {
        QPushButton *btn = new QPushButton(text, this);
        btn->setFixedHeight(62);
        btn->setSizePolicy(QSizePolicy::Expanding, QSizePolicy::Fixed);
        btn->setFont(QFont("Consolas", 18, QFont::Bold));
        btn->setStyleSheet(QString(
            "QPushButton{background:%1;color:%2;border:none;border-
radius:10px;}"
            "QPushButton:hover{background:#585b70;}"
            "QPushButton:pressed{background:#313244;}").arg(bg,fg));
        return btn; };

    QPushButton *btnClear=makeBtn("C","#f38ba8","#1e1e2e");
    QPushButton *btnBack =makeBtn("□","#fab387","#1e1e2e");
    QPushButton *btnLPar =makeBtn("(","#45475a");
    QPushButton *btnRPar =makeBtn(")","#45475a");
    QPushButton *btnPi   =makeBtn("π","#cba6f7");
    QPushButton *btnPow  =makeBtn("^","#45475a");
    QPushButton *btnDiv  =makeBtn("÷","#89b4fa");
    QPushButton *btnMul  =makeBtn("×","#89b4fa");
    QPushButton *btn7=makeBtn("7","#313244"), *btn8=makeBtn("8","#313244"),
                *btn9=makeBtn("9","#313244"),
*btnSub=makeBtn("−","#89b4fa");
    QPushButton *btn4=makeBtn("4","#313244"), *btn5=makeBtn("5","#313244"),
                *btn6=makeBtn("6","#313244"),
*btnAdd=makeBtn("+","#89b4fa");
    QPushButton *btn1=makeBtn("1","#313244"), *btn2=makeBtn("2","#313244"),
                *btn3=makeBtn("3","#313244"),
*btnEq=makeBtn("=","#a6e3a1","#1e1e2e");
    QPushButton *btn0=makeBtn("0","#313244"),
*btnDot=makeBtn(".","#313244");

    grid->addWidget(btnClear,0,0); grid->addWidget(btnBack,0,1);
    grid->addWidget(btnLPar,0,2);  grid->addWidget(btnRPar,0,3);
    grid->addWidget(btnPi,1,0);    grid->addWidget(btnPow,1,1);
    grid->addWidget(btnDiv,1,2);   grid->addWidget(btnMul,1,3);
    grid->addWidget(btn7,2,0);  grid->addWidget(btn8,2,1);
    grid->addWidget(btn9,2,2);  grid->addWidget(btnSub,2,3);
    grid->addWidget(btn4,3,0);  grid->addWidget(btn5,3,1);
    grid->addWidget(btn6,3,2);  grid->addWidget(btnAdd,3,3);
    grid->addWidget(btn1,4,0);  grid->addWidget(btn2,4,1);
    grid->addWidget(btn3,4,2);  grid->addWidget(btnEq,4,3);
    grid->addWidget(btn0,5,0,1,2); grid->addWidget(btnDot,5,2);
    mainLayout->addLayout(grid);

    for(auto *b:{btn0,btn1,btn2,btn3,btn4,btn5,btn6,btn7,btn8,btn9})
        connect(b,&QPushButton::clicked,this,&MainWindow::onDigitClicked);
    btnAdd->setObjectName("+"); btnSub->setObjectName("-");
    btnMul->setObjectName("*"); btnDiv->setObjectName("/");
    btnPow->setObjectName("^"); btnLPar->setObjectName("(");
    btnRPar->setObjectName(")");
    for(auto *b:{btnAdd,btnSub,btnMul,btnDiv,btnPow,btnLPar,btnRPar})
```

```
connect(b,&QPushButton::clicked,this,&MainWindow::onOperatorClicked);
    connect(btnEq,
&QPushButton::clicked,this,&MainWindow::onEqualsClicked);
    connect(btnClear,&QPushButton::clicked,this,&MainWindow::onClearAll);
    connect(btnBack, &QPushButton::clicked,this,&MainWindow::onBackspace);
    connect(btnDot,  &QPushButton::clicked,this,&MainWindow::onDotClicked);
    connect(btnPi,   &QPushButton::clicked,this,&MainWindow::onPiClicked);
    central->setStyleSheet("background:#181825;");
}

MainWindow::~MainWindow() {}

void MainWindow::keyPressEvent(QKeyEvent *e) {
    switch(e->key()) {
    case Qt::Key_Return: case Qt::Key_Enter: onEqualsClicked(); break;
    case Qt::Key_Escape:    onClearAll();    break;
    case Qt::Key_Backspace: onBackspace();   break;
    default: QMainWindow::keyPressEvent(e);
    }
}

void MainWindow::appendToDisplay(const QString &text) {
    QString cur = display->text();
    if (cur=="0" && text>="0" && text<="9") display->setText(text);
    else display->setText(cur+text);
}
void MainWindow::onDigitClicked() {
    if(auto *b=qobject_cast<QPushButton*>(sender())) appendToDisplay(b-
>text());
}
void MainWindow::onOperatorClicked() {
    if(auto *b=qobject_cast<QPushButton*>(sender())) appendToDisplay(b-
>objectName());
}
void MainWindow::onDotClicked()  { appendToDisplay("."); }
void MainWindow::onPiClicked() {
    QString cur=display->text();
    display->setText(cur=="0" ? "p" : cur+"p");
}
void MainWindow::onBackspace() {
    QString cur=display->text();
    display->setText(cur.length()<=1 ? "0" : cur.left(cur.length()-1));
}
void MainWindow::onClearAll() { display->setText("0"); historyLabel-
>setText(""); }
void MainWindow::onEqualsClicked() {
    QString expr=display->text().trimmed();
    if(expr.isEmpty()) return;
    bool ok=false;
    double result=evaluate(expr,ok);
    if(ok) { historyLabel->setText(expr+" =");
             display->setText(QString::number(result,'g',12)); }
    else   { display->setText("Помилка"); }
}
```

```
int MainWindow::getRang(char ch) {
    if(ch=='^') return 3;
    if(ch=='*'||ch=='/') return 2;
    if(ch=='+'||ch=='-') return 1;
    return 0;
}
bool MainWindow::maths(std::stack<Leksema> &Stack_n,
                       std::stack<Leksema> &Stack_o,
                       Leksema             &item)
{
    if (Stack_n.empty() || Stack_o.empty()) return false;

    double a = Stack_n.top().value;
    Stack_n.pop();

    char op = Stack_o.top().type;
    Stack_o.pop();

    if (Stack_n.empty()) return false;
    double b = Stack_n.top().value;
    Stack_n.pop();

    double c = 0;
    switch (op) {
    case '+': c = b + a; break;
    case '-': c = b - a; break;
    case '*': c = b * a; break;
    case '^': c = std::pow(b, a); break;
    case '/':
        if (a == 0.0) return false;   // ділення на 0
        c = b / a;
        break;
    default:
        return false;
    }

    item.type  = '0';
    item.value = c;
    Stack_n.push(item);
    return true;
}

double MainWindow::evaluate(const QString &expr, bool &ok)
```

```
{
    ok = false;
    std::string str = expr.toStdString();
    std::stringstream sstr(str);

    char   ch;
    double value;
    bool   flag = true;

    std::stack<Leksema> Stack_n;
    std::stack<Leksema> Stack_o;
    Leksema item;

    while (true) {
        ch = (char)sstr.peek();
        if (ch == '\377' || sstr.eof()) break;

        if (ch == ' ') { sstr.ignore(); continue; }

        // π
        if (ch == 'p') {
            item.type  = '0';
            item.value = PI_VAL;
            Stack_n.push(item);
            flag = false;
            sstr.ignore();
            continue;
        }
        if ((ch >= '0' && ch <= '9') ||
            (ch == '-' && flag) ||
            (ch == '.'))
        {
            sstr >> value;
            if (sstr.fail()) return 0;
            item.type  = '0';
            item.value = value;
            Stack_n.push(item);
            flag = false;
            continue;
        }
        if (ch == '+' || (ch == '-' && !flag) ||
            ch == '*' || ch == '/' || ch == '^')
```

```
        {
            sstr.ignore();
            if (Stack_o.empty() || getRang(ch) >
getRang(Stack_o.top().type)) {
                item.type  = ch;
                item.value = 0;
                Stack_o.push(item);
            } else {
                while (!Stack_o.empty() &&
                       Stack_o.top().type != '(' &&
                       getRang(ch) <= getRang(Stack_o.top().type))
                {
                    if (!maths(Stack_n, Stack_o, item)) return 0;
                }
                item.type  = ch;
                item.value = 0;
                Stack_o.push(item);
            }
            flag = true;
            continue;
        }
        if (ch == '(') {
            item.type  = '(';
            item.value = 0;
            Stack_o.push(item);
            sstr.ignore();
            flag = true;
            continue;
        }
        if (ch == ')') {
            sstr.ignore();
            while (!Stack_o.empty() && Stack_o.top().type != '(') {
                if (!maths(Stack_n, Stack_o, item)) return 0;
            }
            if (Stack_o.empty()) return 0;   // немає відповідної '('
            Stack_o.pop();
            flag = false;
            continue;
        }

        // Невідомий символ
        return 0;
    }
```

```
    while (!Stack_o.empty()) {
        if (Stack_o.top().type == '(') return 0;
        if (!maths(Stack_n, Stack_o, item)) return 0;
    }

    if (Stack_n.empty()) return 0;
    ok = true;
    return Stack_n.top().value;
}
```