ВІДОКРЕМЛЕНИЙ СТРУКТУРНИЙ ПІДРОЗДІЛ
«ТЕРНОПІЛЬСЬКИЙ ФАХОВИЙ КОЛЕДЖ
ТЕРНОПІЛЬСЬКОГО НАЦІОНАЛЬНОГО ТЕХНІЧНОГО УНІВЕРСИТЕТУ
ІМЕНІ ІВАНА ПУЛЮЯ»
ВІДДІЛЕННЯ ТЕЛЕКОМУНІКАЦІЙ ТА ЕЛЕКТРОННИХ СИСТЕМ
Циклова комісія комп'ютерних наук

КУРСОВА РОБОТА

з дисципліни:

«Об'єктно-орієнтоване програмування»

на тему: **«Матричний калькулятор»**

Студента 3 курсу групи КН-321
спеціальності 122 «Комп'ютерні науки»
Березовський В.О.
(прізвище та ініціали)

Керівник: викладач Лісовий В.М.

Національна шкала _____

Кількість балів: _____ Оцінка: ECTS _____

Члени комісії: _____ Володимир ЛІСОВИЙ
(підпис)

_____ Роман ДІЛАЙ
(підпис)

м. Тернопіль – 2023

ЗМІСТ

ВСТУП

В сучасних умовах розвитку інформаційних технологій та комп'ютерної індустрії об'єктно-орієнтоване програмування (ООП) стає невід'ємною складовою ефективної розробки програмного забезпечення. Програмісти використовують принципи ООП для створення гнучких та легко розширюваних програм, які задовольняють вимоги сучасного користувача.

У сучасній практиці програмування особливий акцент робиться на вдосконаленні та прискоренні процесу проектування програм, а також на психолого-фізіологічних аспектах сприймання інформації користувачем. Зростання приватного сектору економіки призводить до зменшення обсягів витрат на програмне забезпечення, спонукаючи розробників створювати ефективні та економічні програмні рішення.

У даній курсовій роботі розглядається розробка матричного калькулятора, реалізованого мовою програмування C++ з використанням ідеології об'єктно-орієнтованого програмування. Головна мета цього проекту — створення гнучкого програмного засобу, спрямованого на вирішення конкретних завдань у галузі обліку прибутково-видаткових операцій для домовласників багатоквартирного будинку.

Об'єктно-орієнтоване програмування дозволяє створювати програми у вигляді взаємодіючих об'єктів, що є екземплярами певних класів. Кожен клас утворює ієрархію наслідування, що спрощує розширення та модифікацію програмного коду. Такий підхід дозволяє швидко реалізувати функціональність та забезпечити ефективну роботу програмного забезпечення.

Розглядаючи цей проект, можна зрозуміти, як використання принципів об'єктно-орієнтованого програмування сприяє створенню потужних та ефективних програмних рішень, що відповідають вимогам сучасної індустрії та забезпечують економічну ефективність.

1 ТЕХНІЧНЕ ЗАВДАННЯ

1.1 Найменування та область застосування

Найменування програми – Програма «Matrix-Calculator-master».

Область застосування програми – "Matrix-Calculator-master" є матричним калькулятором, спрямованим на виконання різноманітних операцій над матрицями. Цей інструмент призначений для швидкого та ефективного вирішення матричних обчислень, які використовуються у різних областях науки та інженерії. Програма становить важливий інструмент для студентів, науковців, інженерів та всіх, хто працює з лінійною алгеброю та матричним аналізом.

1.2 Підстави для розробки

Підставами для проведення розробки являється індивідуальне завдання на курсову роботу з дисципліни «Об’єктно-орієнтоване програмування».

Найменування теми курсової роботи – «Матричний калькулятор».

Замовник – керівник курсової роботи, викладач Слободян Руслан Олесійович.

Виконавець – студент групи КН 321 Березовський Віталій Олександрович

1.3 Призначення розробки

"Matrix-Calculator-master" призначений для науковців, студентів, інженерів та всіх, хто займається науковими та технічними дослідженнями, що включають в себе матричні обчислення. Програма надає зручний та ефективний інтерфейс для виконання операцій над матрицями, що сприяє підвищенню продуктивності та точності обчислень.

Функціональне призначення: Основні функції "Matrix-Calculator-master" включають в себе оптимізацію та спрощення виконання матричних

операцій, реєстрацію та облік прибутково-видаткових операцій. Крім того, програма забезпечує можливість друкування звітів про фінансову діяльність, що дозволяє користувачам зручно аналізувати та документувати результати своєї роботи.

1.4 Вимоги до програми чи програмного виробу

1.4.1 Вимоги до функціональних характеристик

Операції з матрицями:

Додавання, віднімання та множення матриць: Забезпечення можливості виконання основних операцій лінійної алгебри для матриць будь-якого розміру.

Транспонування та обертання матриць: Реалізація функцій, що дозволяють змінювати орієнтацію та положення елементів матриць.

Робота з даними:

Введення матриць користувачем: Забезпечення зручного та інтуїтивно зрозумілого інтерфейсу для введення матриць з клавіатури.

Завантаження та збереження матриць: Можливість імпорту та експорту матриць у різних форматах для зручності обміну даними.

Вивід результатів:

Графічне відображення матриць: Забезпечення графічного відображення матриць для кращого сприйняття користувачем.

Подробний вивід обчислень: Виведення деталей процесу обчислень для кращого розуміння користувачем.

Обробка помилок:

Повідомлення про помилки введення: Система повідомлень, яка надає користувачеві інформацію про помилки при некоректному введенні даних.

Автоматична корекція помилок: Можливість програмної корекції невірних даних, якщо це можливо.

1.4.2 Вимоги до часових характеристик

Програма "Матричний Калькулятор" повинна дотримуватися вимог до часових характеристик, щоб забезпечити ефективність та комфорт користування:

Швидкодія операцій з матрицями:

Вимагається оптимізація операцій множення, додавання, віднімання, транспонування та обертання матриць для роботи з матрицями будь-якого розміру.

Забезпечення ефективної реалізації алгоритмів для швидкого виконання операцій лінійної алгебри.

Реакція на введення користувача:

Забезпечення мінімального часу очікування реакції програми на дії користувача.

Миттєва відповідь програми на клік, введення команд або зміни параметрів.

Робота з обсягами даних:

Забезпечення ефективної роботи програми при обчисленні з великими матрицями, враховуючи обмеження пам'яті та обчислювальних ресурсів.

Мінімізація часу завантаження та збереження матриць для оптимального використання файлової системи.

Ефективне використання ресурсів:

Підтримка операцій на багатьох ядрах для максимального використання ресурсів сучасних мультипроцесорних систем.

Адаптація програми під різні технічні характеристики пристроїв для забезпечення оптимальної швидкодії.

Асинхронні операції:

Можливість використання паралельних обчислень для підвищення ефективності виконання операцій.

Реалізація асинхронних процесів для уникнення блокування інтерфейсу під час виконання великих операцій.

1.4.3 Вимоги до надійності

Якщо дані введені не вірно, то виводиться причина помилки.

1.4.4 Умови до експлуатації

Розроблений програмний продукт по потребуватиме яких-небудь видів обслуговування, для його використання достатньо буде навиків користувача ПК

1.4.5 Вимоги до складу і параметрів технічних засобів

В склад технічних засобів повинні входити: монітор (діагоналлю не менше 15”), клавіатура, мишка, IBM-сумісний персональний комп’ютер із такими мінімальними характеристиками: – процесор – Intel Pentium 4; – оперативна пам’ять - 1 ГБ; – обсяг дискової пам’яті – 80 ГБ.

1.4.6 Вимоги до інформаційної і програмної сумісності

Вихідні коди програми повинні бути реалізовані мовою C++. В якості середовища розробки програми повинне використовуватись середовище Qt Creator не нижче 12 версії. Системні програмні засоби і утиліти, які буде використовувати програма, повинні забезпечуватись операційною системою сімейства Windows версії не нижче Windows 7. Вимоги до захисту інформації і

програми не пред’являються.

1.5 Вимоги до програмної документації

По закінченню розробки програмного забезпечення потрібно підготувати таку документацію:

- інструкція інсталяції програми;
- загальні відомості про можливості програми;
- інструкція з експлуатації.

1.6 Техніко-економічні показники

Розрахунок економічної ефективності і вартості розробки програмного продукту не проводиться. Приблизне число використань розробленої програми в рік – 100-130 раз.

1.7 Стадії та етапи розробки

Розробка матричного калькулятора на платформі C++ Qt, призначеного для виконання обчислень з матрицями без використання бази даних, передбачає послідовні стадії:

Аналіз:

Мета аналізу – докладний розгляд та опис задачі, пов'язаної із створенням матричного калькулятора. На цьому етапі проводиться об'єктна декомпозиція, визначаються особливості поведінки об'єктів та розробляється структурна схема програми. В результаті аналізу формується опис об'єктів та подій, а також визначається загальна архітектура програми.

Проектування:

Проектування розділяється на логічне та фізичне. На логічному етапі

розробляється структура класів, визначаються поля для збереження даних та алгоритми методів. Фізичне проектування включає об'єднання класів у модулі, вибір схеми підключення, визначення способів взаємодії з обладнанням та програмним забезпеченням. Результатом є ієрархія класів та їх детальний опис.

Еволюція:

Етап еволюції системи передбачає послідовну реалізацію та підключення класів до проекту. Починаючи із створення основного інтерфейсу та бібліотеки класів, на цьому етапі визначається тип полів класу, розробляються алгоритми методів та уточнюються вимоги до програми. Створюється прототип продукту, який піддається тестуванню та налагодженню.

Ці стадії дозволять систематично підходити до розробки матричного калькулятора, забезпечуючи високий рівень аналізу, структуризації, та ефективності програмного рішення.

1.8 Порядок контролю та прийому

Прийом розробленого програмного забезпечення повинен відбуватися на об’єкті Замовника в терміни, які зазначені в індивідуальному завданні.

Для прийому роботи Виконавець повинен представити:

– діючу програму, яка повністю відповідає даному технічному завданню;

– вихідний програмний код, записаний разом із програмою на оптичний носій інформації.

Прийом програмного забезпечення повинен відбуватися перед комісією з двох чоловік (один з яких – Замовник) у такій послідовності:

– доповідь Виконавця про виконану роботу;

– демонстрація Виконавцем роботи програми;

– контрольні випробовування роботи програми;

– відповіді на запитання і зауваження комісії.

2 РОЗРОБКА ТЕХНІЧНОГО ТА РОБОЧОГО ПРОЄКТУ

2.1 Розробка загальної структури і варіантів використання програми

В даному підрозділі пояснювальної записки описується такий етап розробки програми, як аналіз. Метою аналізу є максимально повний опис поставленої задачі, відповідно до технічного завдання. Під час проходження етапу аналізу спочатку було визначено такі питання:

– Хто буде діючими суб'єктами?

– Хто вводитиме інформацію?

– Хто запрошуватиме?

– Чи буде хто-небудь ще взаємодіяти з програмою?

– Чи буде сама програма взаємодіяти з іншими програмами?

В даному випадку з програмою "Matrix-Calculator-master" має працювати користувач, якому потрібні матричні обчислення. Таким чином, один і той самий користувач вводить інформацію, здійснює розрахунки і переглядає їх. Наступне, що потрібно було з'ясувати, це групу дій, які ініціюватиме діючий суб'єкт. Виходячи із технічного завдання, діючий суб'єкт – користувач буде виконувати такі дії:

– почати роботу з програмою;

– обрати тип матриці;

– обрати розмір матриці;

– ввести значення в матрицю;

– виконати потрібну дію з матрицею;

– отримати результат;

Таким чином, діаграму варіантів використання, яка отримується внаслідок наведеного переліку дій, можна подати так, як це показано на рисунку 2.1.

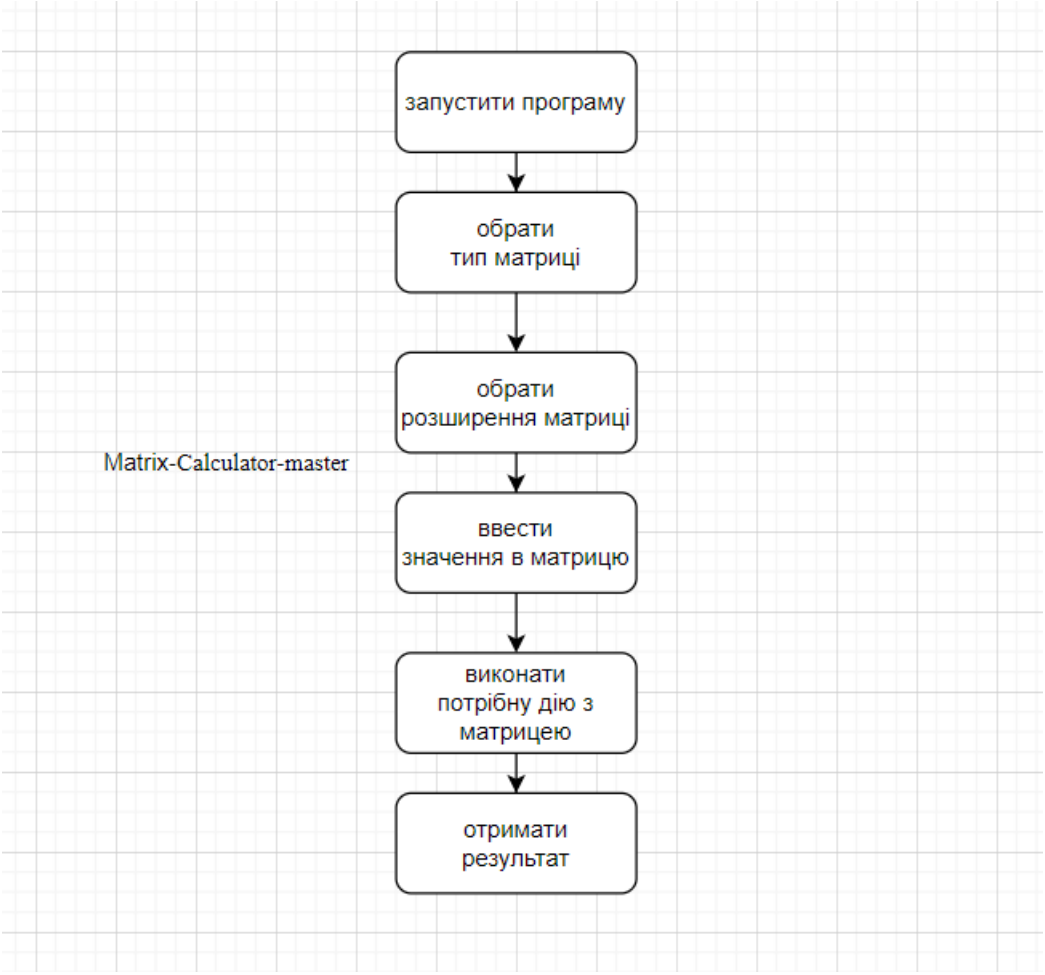

Рисунок 2.1 – Діаграма варіантів використання для програми Matrix-Calculator-master

Наступний крок етапу аналізу це опис варіантів використання, де необхідно детально описати всі варіанти використання.

Запустити програму. Ця дія, здавалося б, є дуже очевидною для того, щоб про неї зовсім не згадувати, але все ж таки... Коли запускається програма, на екран повинно виводитися меню, з якого користувач може вибрати потрібну дію. Це може називатися екраном інтерфейсу користувача.

Обрати тип матриці: сценарій 1. На екрані має відобразитися випадаючий список, у якому програма просить користувача обрати тип матриці. Є три варіанти: стандартна, бінарна та квадратна.

Обрати розмір матриці: сценарій 1. На екрані біля клітинок матриці мають відобразитися слайдери, перетягуючи котрі можна змінювати розмір

матриці. Користувачу потрібно обрати потрібний розмір матриці.

Ввести значення в матрицю: Вводимо потрібні дані у кожну клітинку матриці, які розміщені на екрані. Можна вводити лише числові значення.

Виконати потрібну дію з матрицею: У графічному інтерфейсі програми обираємо дію, яку ми будемо виконувати з матрицями. Є бінарні операції додавання, віднімання та множення матриць та унарні операції з матрицями: транспонування матриці, обернення матриці, заповнення матриці, очищення матриці, копіювання матриці, введення випадкових даних у матрицю. Також, якщо у типі матриці обрали квадратну матрицю, то з'являються функції магічного квадрату матриці та судоку матриці. Якщо ж ви обрали бінарну матрицю, тоді з'являється функція лабіринт матриці.

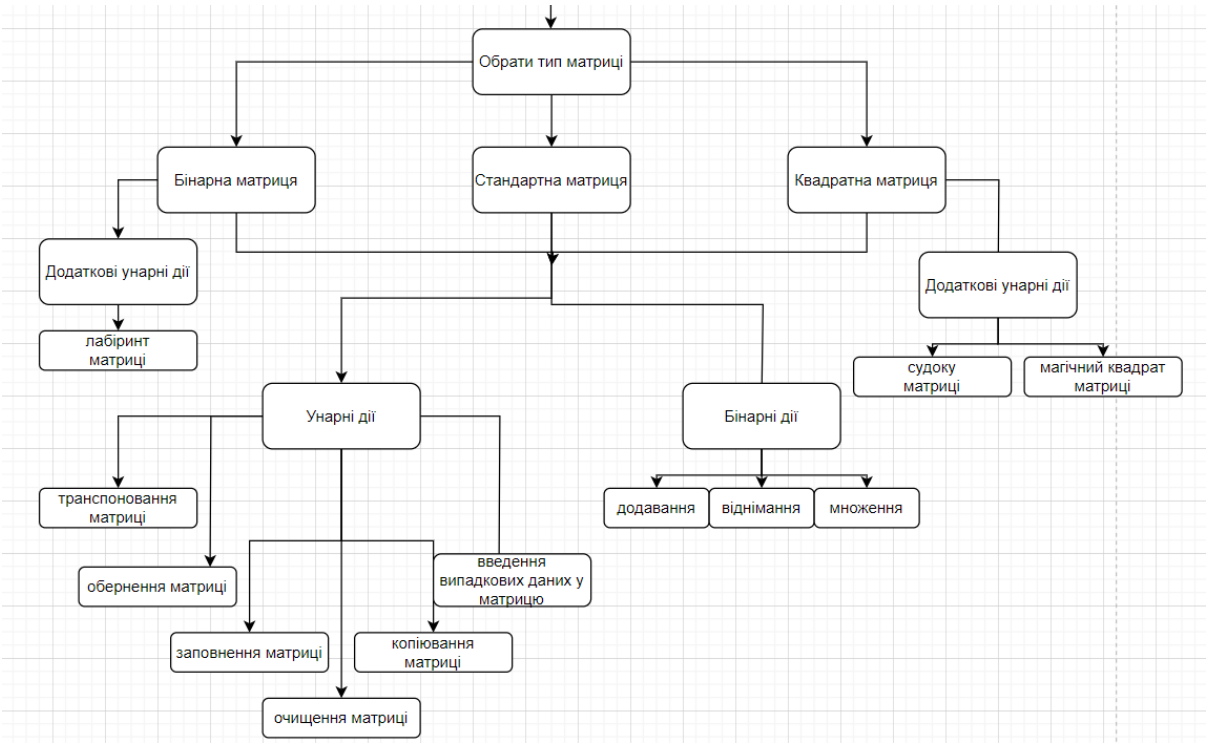

Рисунок 2.2 – Діаграма варіантів вибору дій для програми Matrix-Calculator-master

Отримати результат: Програма виводить результат у обрану матрицю, а помилки в спеціальне поле з текстом.

2.2 Розробка системи класів

Наступний етап розробки програмного забезпечення – етап проектування. Даний етап поділяється на логічне та фізичне проектування. Логічне полягає у розробці структури класів, коли визначаються поля для збереження складових об'єктів, алгоритми методів, що описують поведінку об'єктів. Насамперед необхідно передбачити класи, з яких буде складатися програма. У нас є три види матриць, тому давайте розіб’ємо їх на класи. Також розіб’ємо на класи дії з матрицями:

Отже, складемо перелік класів, які було тільки що уточнено:

1) Стандартна матриця.

2) Квадратна матриця.

3) Бінарна матриця.

4) Бінарні дії.

5) Унарні дії.

Розроблюючи програму для матричного калькулятора, ми визначимо класи для роботи з матрицями. Кожен клас буде відповідати конкретному типу матриці та виконувати відповідні операції. Давайте розглянемо основні класи програми:

1) **Стандартна матриця (StandardMatrix):** Цей клас буде відповідати за створення та операції зі звичайними матрицями.

2) **Квадратна матриця (SquareMatrix):** Він буде наслідувати властивості стандартної матриці, спрощуючи створення та операції для квадратних матриць.

3) **Бінарна матриця (BinaryMatrix):** Цей клас визначить бінарні матриці та пов'язані з ними операції.

4) **Бінарні дії (BinaryOperations):** Виконує бінарні операції над

матрицями, наприклад, додавання та множення.

5) **Унарні дії (UnaryOperations):** Містить унарні операції, такі як транспонування.

Ваша програма може використовувати ці класи для введення, операцій та виведення матриць. Наприклад, введення та виведення може здійснюватися через окна інтерфейсу користувача, а операції можуть бути викликані з відповідних кнопок чи інтерфейсів.

Ця структура допоможе вам організувати вашу програму для матричного калькулятора, роблячи код більш зрозумілим та підтримуваним.

StandardMatrix (Стандартна матриця):

Атрибути: rows (рядки), cols (стовпці), data (дані матриці).

Методи: createMatrix() (створення матриці), accessElement() (доступ до елементів матриці), basicOperation() (базові операції).

SquareMatrix (Квадратна матриця):

Наслідує властивості StandardMatrix.

Методи: createMatrix() (створення квадратної матриці), additionalMethod() (додатковий метод).

BinaryMatrix (Бінарна матриця):

Атрибути: rows (рядки), cols (стовпці), data (дані матриці).

Методи: createMatrix() (створення бінарної матриці), additionalMethod() (додатковий метод).

BinaryOperations (Бінарні операції):

Методи: add() (додавання матриць), multiply() (множення матриць).

UnaryOperations (Унарні операції):

Методи: transpose() (транспонування матриці), additionalMethod() (додатковий метод).

UserInterface (Інтерфейс користувача):

Методи: displayMatrix() (відображення матриці на екрані), getInput() (отримання введення від користувача), showOutput() (відображення виводу).

Кожен клас виконує конкретну функцію та взаємодіє з іншими класами для реалізації повного функціоналу матричного калькулятора.

2.3 Розробка методів

В рамках розробки програмного забезпечення Матричного калькулятора визначено низку методів, що спрямовані на реалізацію різноманітних операцій з матрицями через графічний інтерфейс програми.

Початковий етап еволюції включав у себе створення основного інтерфейсу та бібліотеки класів. Зокрема, для кожного типу матриці були визначені необхідні атрибути та методи. У класі Matrix реалізовано базові властивості, а класи для конкретних видів матриць (наприклад, квадратна та бінарна) розширюють функціонал відповідно до їхніх особливостей.

Однією з ключових частин програми є бінарні та унарні операції. Для реалізації бінарних операцій створено метод performBinaryOperation(matrix1, matrix2, operationType), який приймає дві матриці та тип операції, повертаючи результат відповідної операції. Унарні операції, такі як транспонування чи обертання матриці, реалізовані відповідними методами для кожного класу матриці.

Крім основних операцій, додаткові функції введено для роботи з квадратними та бінарними матрицями. Для квадратних матриць доступні функції створення магічного квадрата та головоломки судоку. Для бінарних матриць реалізовано функцію генерації лабіринту, що додає елемент гри в обробку даних.

Усі атрибути та методи класів взаємодіють між собою, утворюючи злагоджену систему для роботи з матрицями. Використання стандартних контейнерів STL, таких як вектори та множини, дозволяє ефективно організувати дані та виконувати операції з матрицями.

Конструктор MatrixCalculator()

Конструктор визначається для ініціалізації об'єкту класу MatrixCalculator при створенні. На цьому етапі можливі встановлення початкових значень чи ініціалізація деяких ресурсів.

Matrix performBinaryOperation(const Matrix& matrix1, const Matrix& matrix2, BinaryOperation operation)

Даний метод призначений для виконання бінарної операції над двома матрицями. Параметри методу включають обидві матриці, над якими проводиться операція, та тип бінарної операції (додавання, віднімання, множення).

Matrix performUnaryOperation(const Matrix& matrix, UnaryOperation operation)

Метод виконує унарну операцію над вхідною матрицею. Вказується сама матриця та тип унарної операції (транспонована, обернена, заповнення, очищення, копіювання, введення випадкових даних).

Matrix generateBinaryMatrixMaze(int rows, int cols)

Цей метод використовується для генерації бінарної матриці, яка служить лабіринтом. Розміри лабіринту задаються параметрами rows і cols.

Matrix generateMagicSquare(int order)

Метод генерує магічний квадрат для квадратної матриці з порядком order. Магічний квадрат – це квадратна матриця, у якої суми елементів по кожному рядку, стовпцю і діагоналі однакові.

Matrix generateSudoku(int order)

Даний метод створює матрицю для гри в судоку з порядком order. Генерується валідне початкове положення для гри.

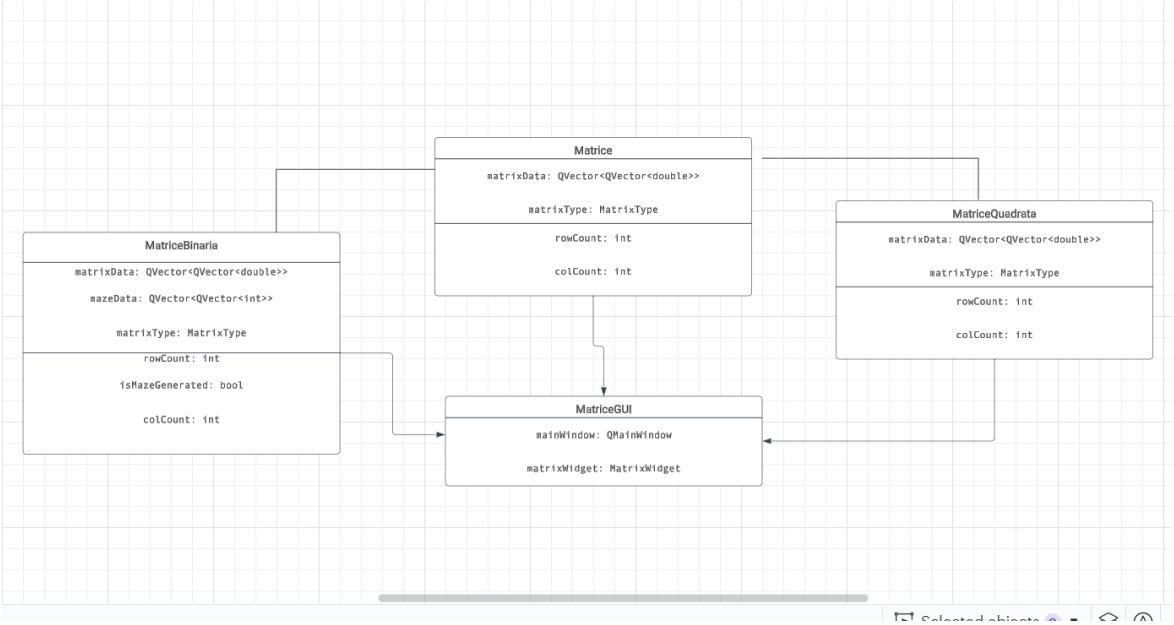

Рисунок 2.4 – UML-діаграма класів програми Matrix-Calculator-master

2.4 Проєктування і опис інтерфейсу користувача

У зв'язку з тим, що програма Матричного калькулятора складається лише з одного вікна, інтерфейс користувача має бути максимально зручним та функціональним. Наведемо основні елементи та їх функціонал:

Вибір типу матриці:

Користувач обирає тип матриці за допомогою випадаючого списку. Включає в себе опції: звичайна, квадратна, бінарна.

Додаткові параметри (магічний квадрат, судоку, лабіринт) можуть виводитися динамічно, залежно від обраного типу матриці.

Вибір дії з матрицею:

Відобразження доступних операцій для обраної матриці. Бінарні операції (додавання, віднімання, множення) та унарні операції (транспонування, обернення, заповнення, очищення, копіювання, введення випадкових даних).

Поле введення матриці:

Можливість введення матриці вручну через текстове поле.

Опція вибору матриці з попереднього сеансу або історії обчислень.

Виведення результатів:

Відображення результатів обчислень на інтерфейсі.

Чітка індикація помилок чи некоректних операцій.

Керування параметрами:

Відповідні кнопки для запуску обраної операції та очищення полів.

Відображення статусу обчислень та повідомлення про завершення операції.

Додаткові функції:

Можливість збереження та завантаження матриць або історії операцій.

Візуалізація матриць та їхніх властивостей.

Загальний принцип інтерфейсу полягає в тому, щоб забезпечити користувачеві легкий доступ до всіх функціональних можливостей програми, зберігаючи при цьому простоту та зрозумілість використання.

2.5 Опис файлової структури програми

Файлова структура програми Матричного калькулятора має наступний вигляд:

Source-файли:

main.cpp: Основний файл, який містить точку входу в програму.

widget.cpp: Реалізація класу "Widget" для графічного інтерфейсу.

matrice.cpp: Реалізація основного класу "Matrice" для роботи з матрицями.

matricequadrata.cpp: Реалізація класу "MatriceQuadrata" для роботи з квадратними матрицями.

matricebinaria.cpp: Реалізація класу "MatriceBinaria" для роботи з бінарними матрицями.

matricegui.cpp: Реалізація класу "MatriceGUI" для забезпечення графічного інтерфейсу користувача.

Header-файли:

widget.h: Заголовковий файл класу "Widget" для графічного інтерфейсу.

matrice.h: Заголовковий файл класу "Matrice" для роботи з матрицями.

matricequadrata.h: Заголовковий файл класу "MatriceQuadrata" для роботи з квадратними матрицями.

matricebinaria.h: Заголовковий файл класу "MatriceBinaria" для роботи з бінарними матрицями.

matricegui.h: Заголовковий файл класу "MatriceGUI" для графічного інтерфейсу користувача.

eccezioni.h: Заголовковий файл, що містить визначення виключень (eccezioni).

Form-файли:

widget.ui: Форма користувальницького інтерфейсу для класу "Widget".

Ця структура спрощує розподіл та управління різними частинами програми, забезпечуючи чистоту та організованість кодової бази.

2.6 Опис структури бази даних програми

Програма не використовує базу даних.

3 ТЕСТУВАННЯ ПРОГРАМИ І РЕЗУЛЬТАТИ ЇЇ ВИКОНАННЯ

В ході тестування програми було виявлено і виправлено незначні помилки. Запустивши програму на екрані монітора відображається головний екран програми, яке представлене на рисунку 3.1.

З його допомогою домовласник може виконати такі дії:

– Введення даних;

– Виведення даних;

– Вихід із програми.

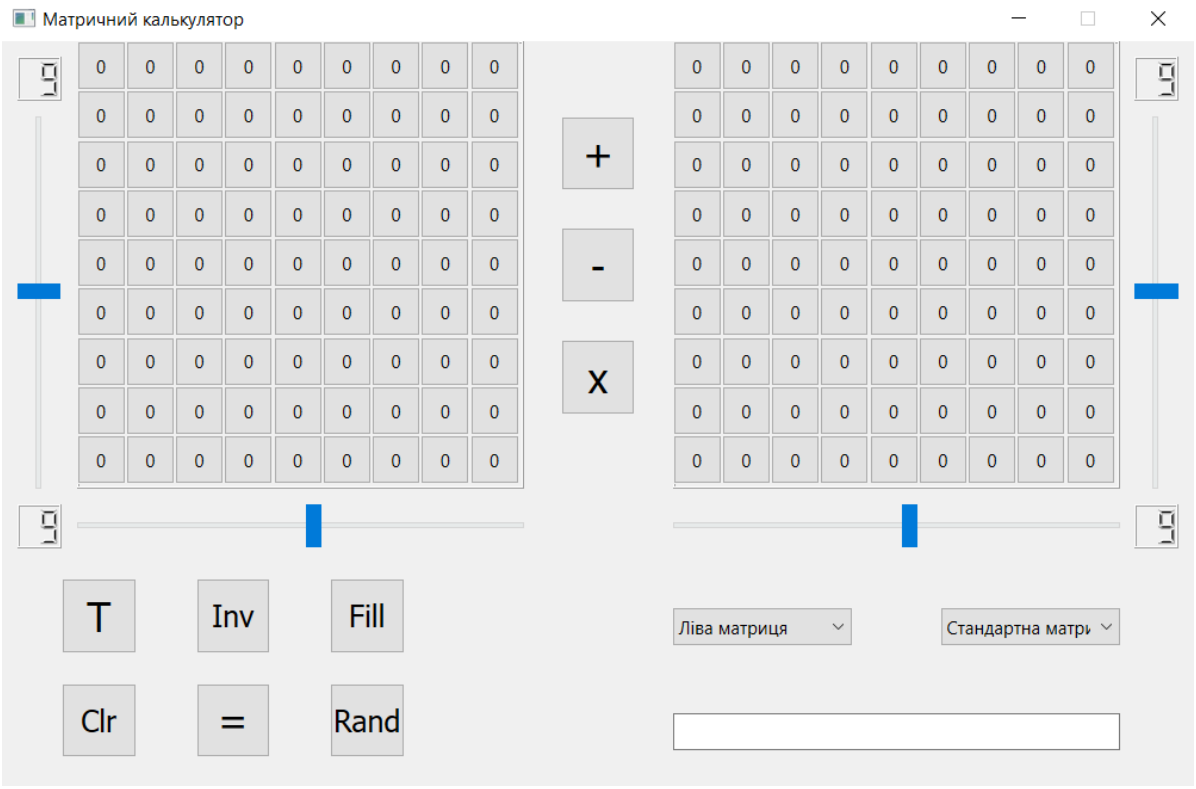

Рисунок 3.1 – Головний екран програми

На головному екрані ви бачите дві матриці, слайдери для змінення розміру матриці, кнопки для бінарних та унарних дій, випадаючі списки для викору типу матриці та для вибору матриці для проведення дій, поле для виводу помилок.

Далі нажмемо на випадаючий список з типами матриць та вибираємо стандартну (див. рис. 3.2). Далі у другому випадаючому списку обираємо роботу з

лівою матрицею (див. рис. 3.3).

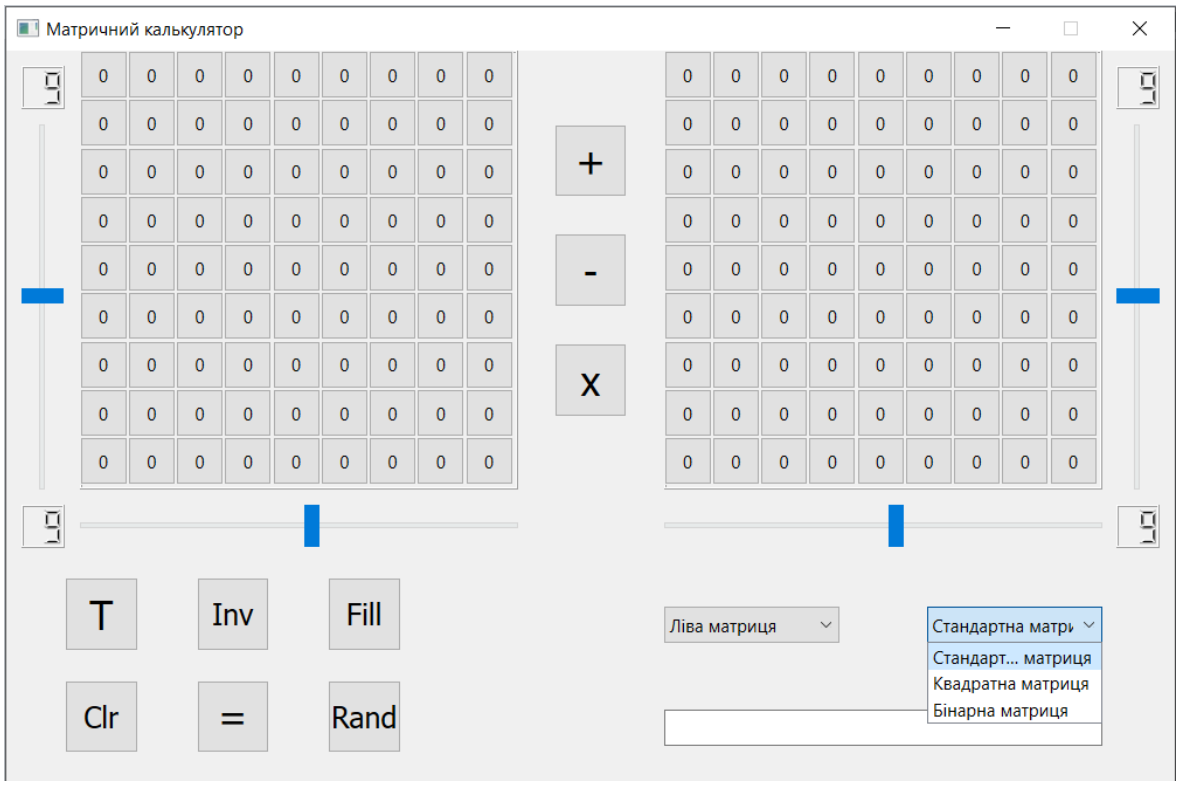

Рисунок 3.2 – Головний

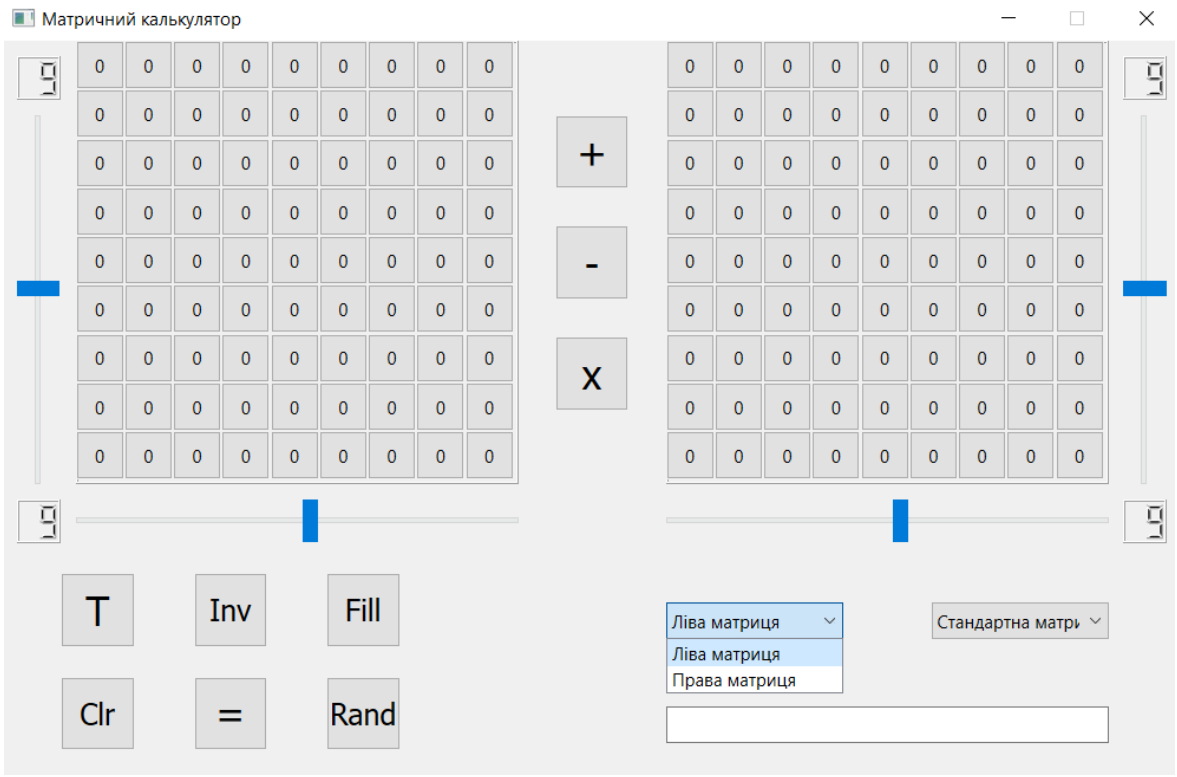

Рисунок 3.3 – Головний

Далі за допомогою слайдерів змінюємо розмір матриці з 9x9 до 3x3(див. рис. 3.4), але якщо ми спробуємо зробити дію, то побачимо помилку (див. рис. 3.5).

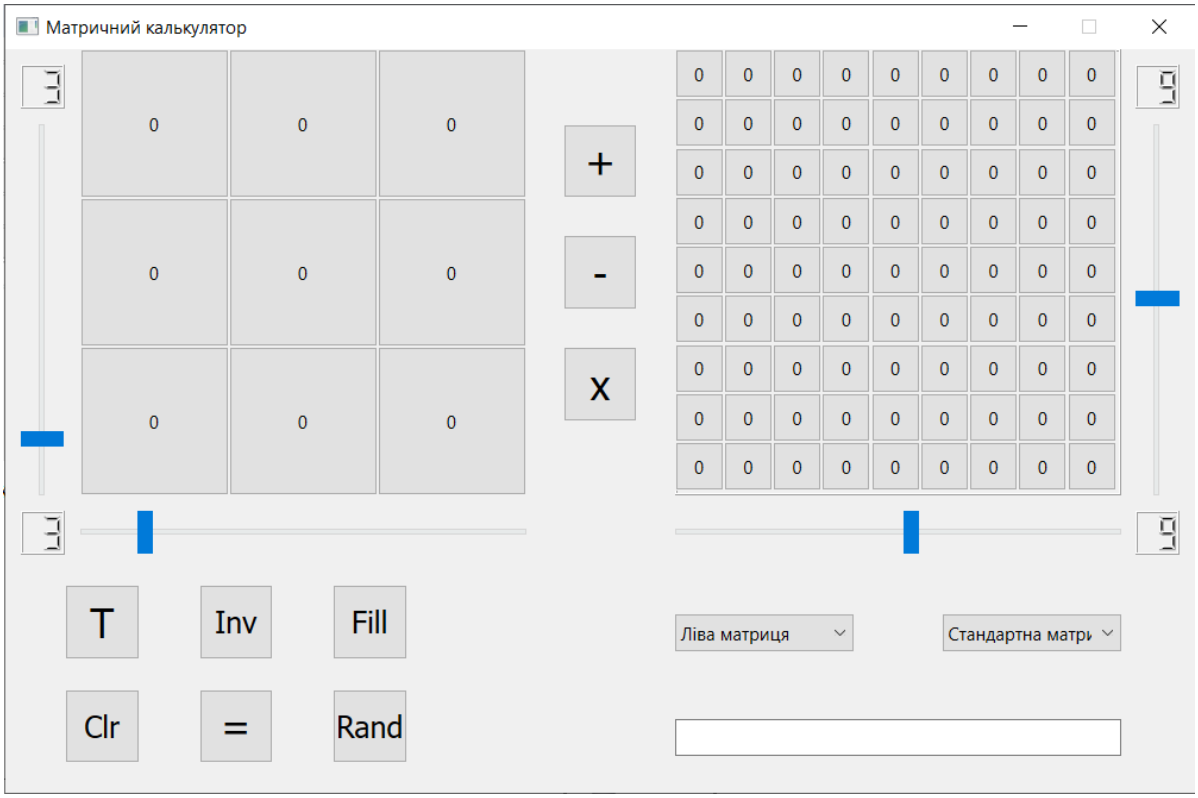

Рисунок 3.4 – Головний

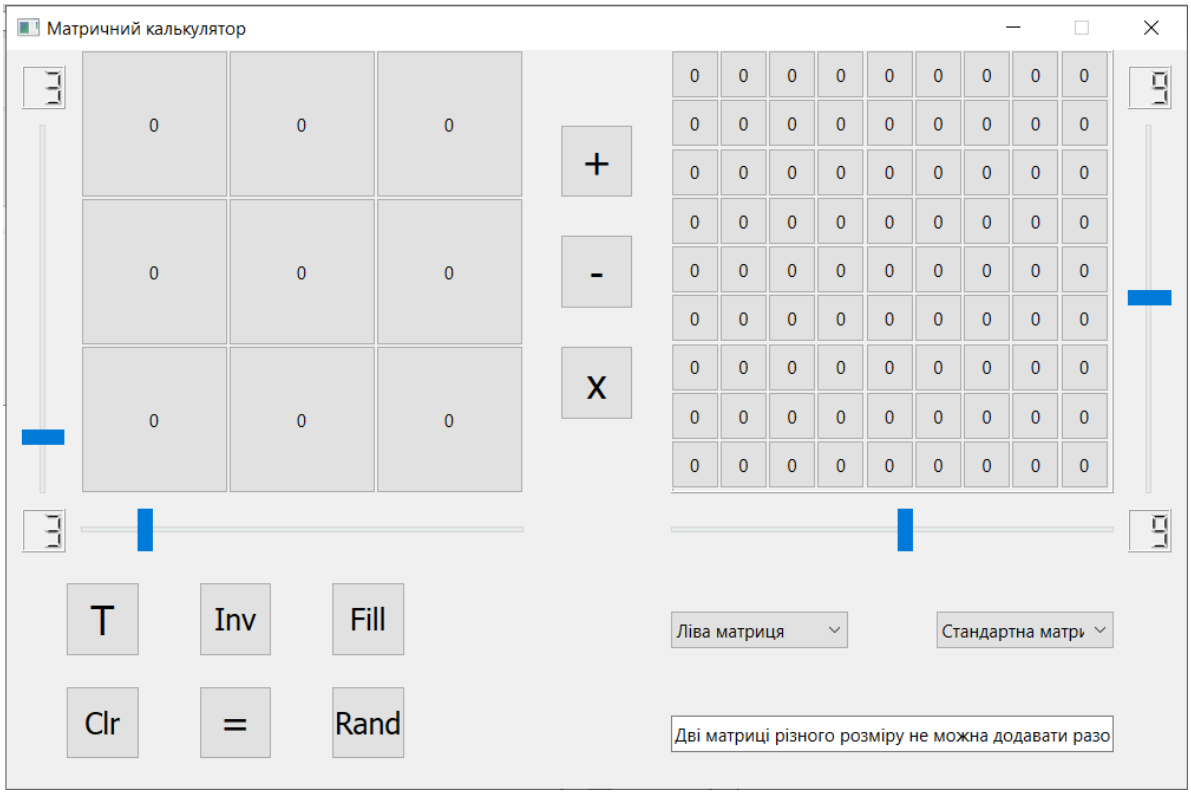

Рисунок 3.5 – Головний

Помилку ми отримуємо через те, що матриці різного розміру, тому давайте змінемо розмір правої матриці(див. рис. 3.6). Тепер заповнюємо кожну клітинку числами. Для цього клікаємо на кожну клітинку матриці та з клавіатури заповнюємо їх числами(див. рис. 3.7).

Тепер давайте спробуємо виконати бінарну функцію віднімання(див. рис. 3.8) та унарну функцію очищення матриці (див. рис. 3.9).

Заповнюємо очищену матрицю знову(див. рис. 3.10). Тепер на зановозаповненій матриці випробуємо унарну функцію обернення матриці(див. рис. 3.11). Тепер давайте закриємо програму, для цього нажміть на хрестик у верхньому правому куті вікна(див. рис. 3.12).

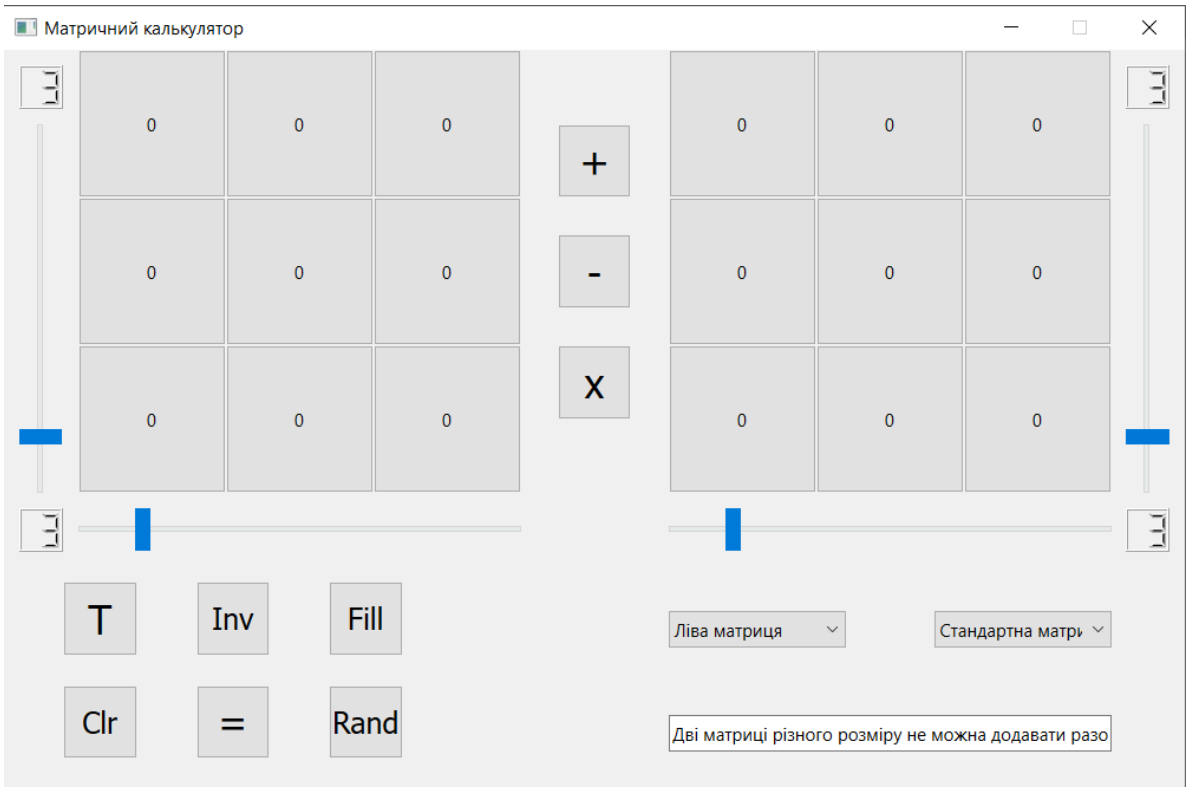

Рисунок 3.6 – Головний

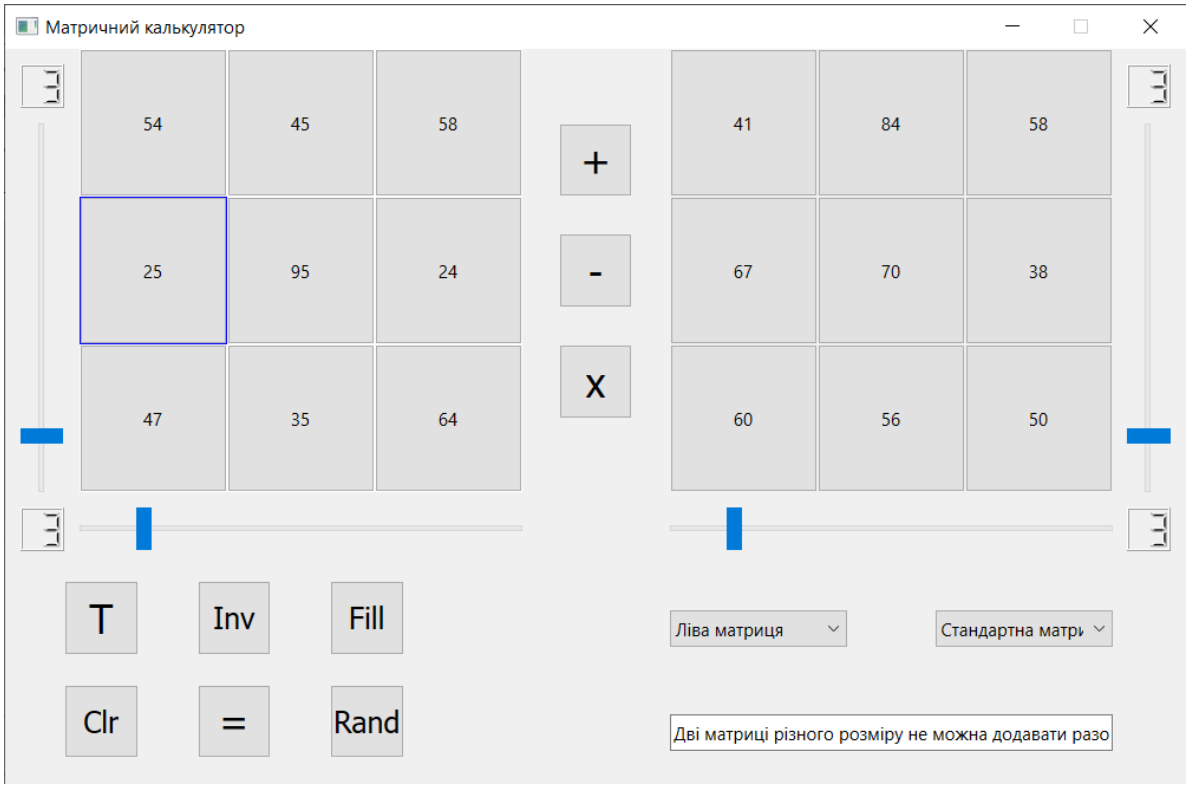

Рисунок 3.7 – Головний

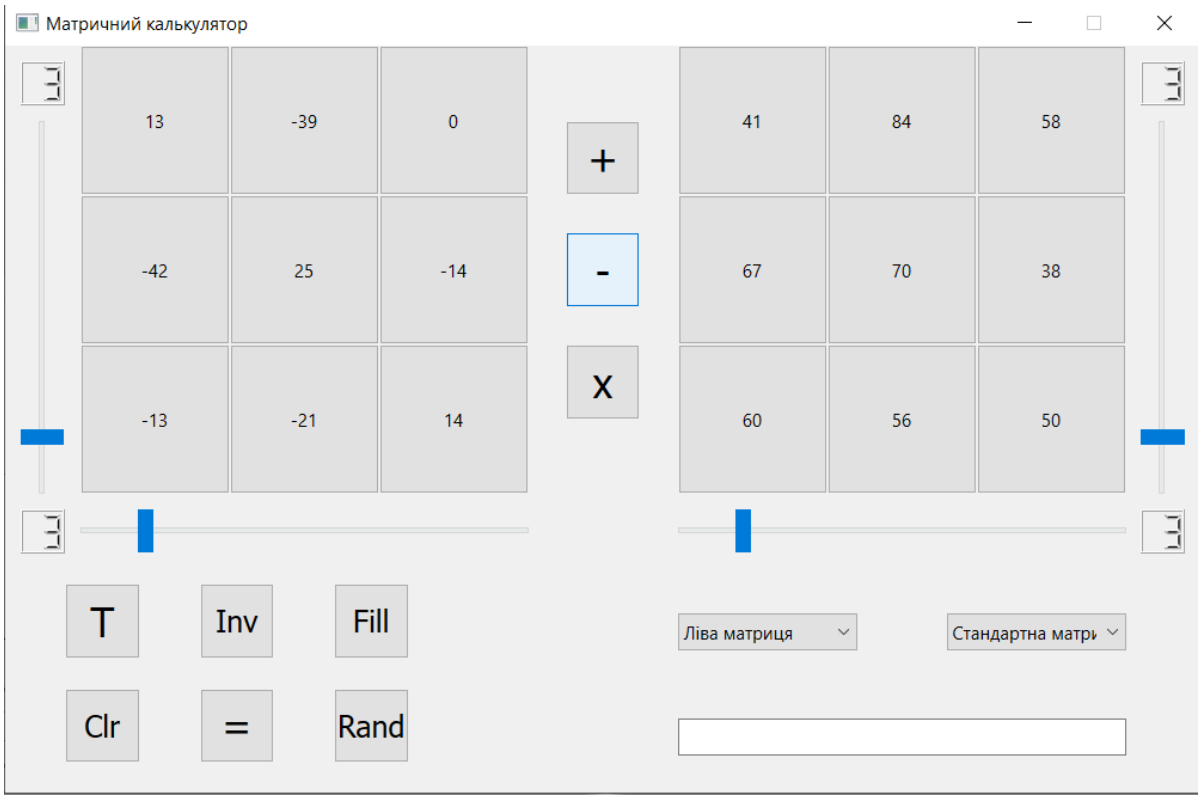

Рисунок 3.8 – Головний

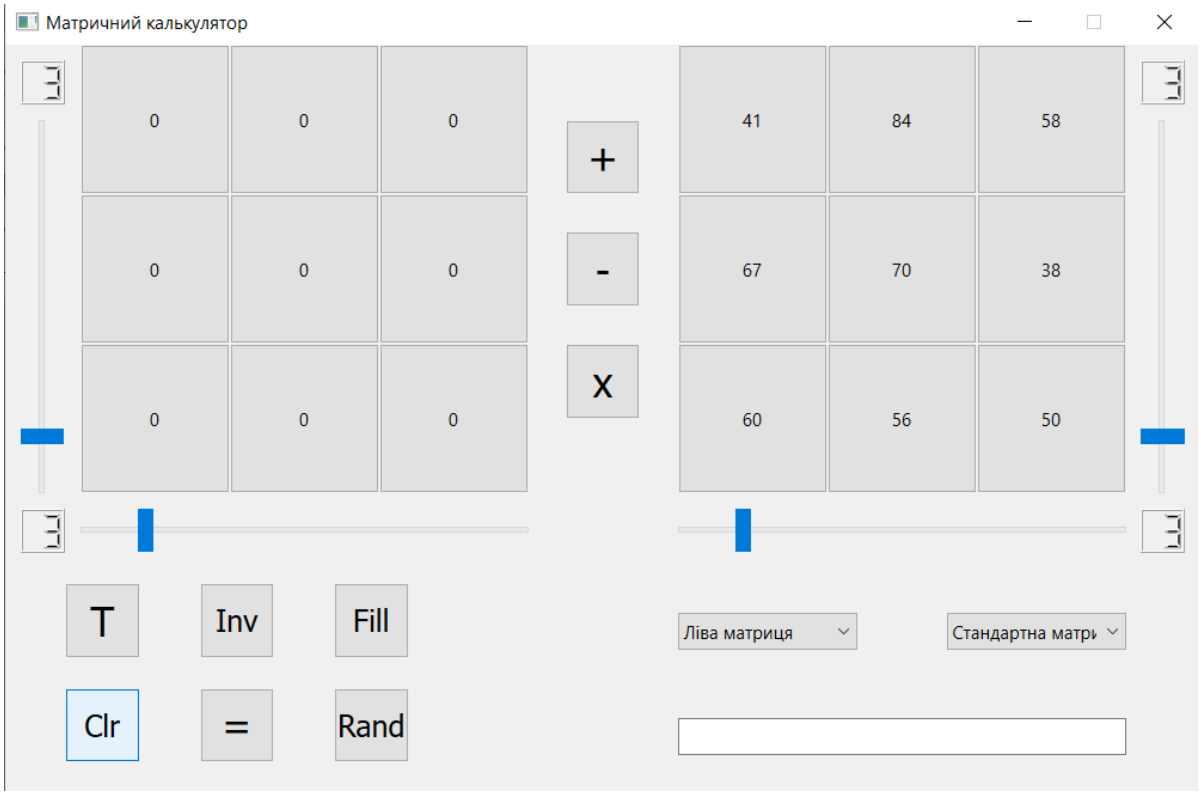

Рисунок 3.9 – Головний

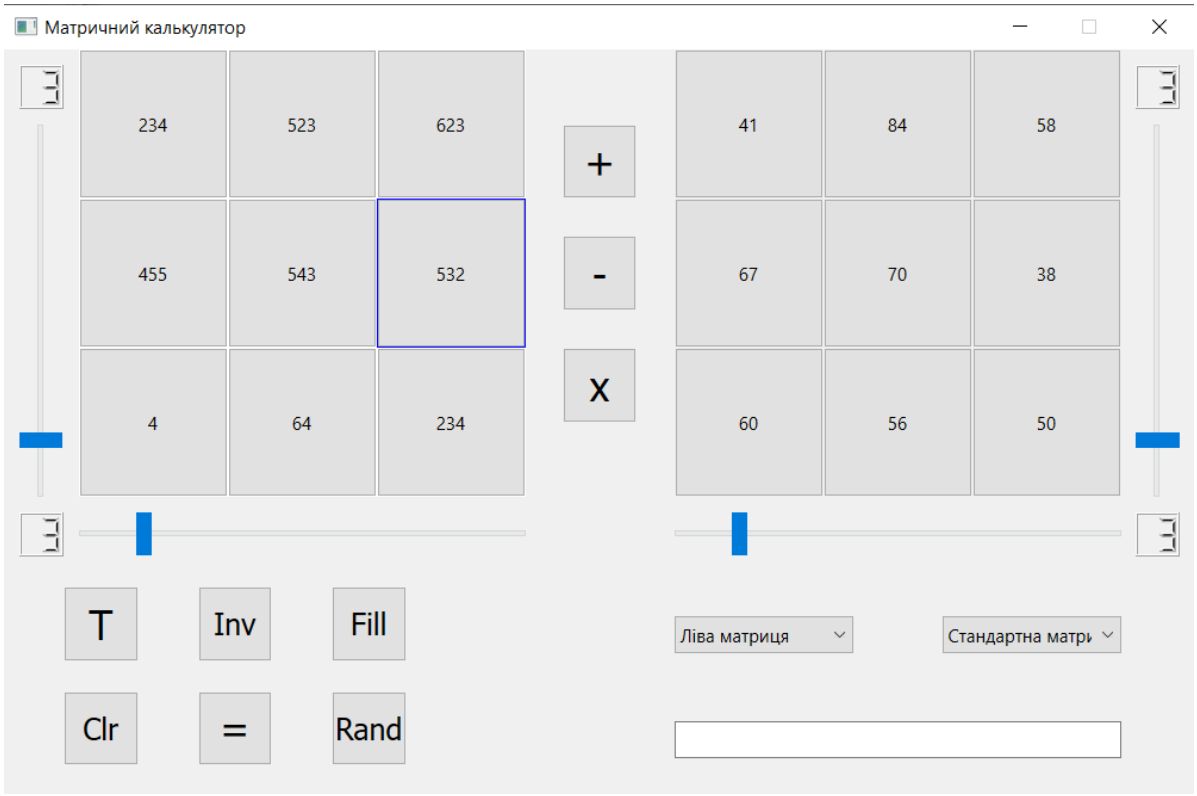

Рисунок 3.10 – Головний

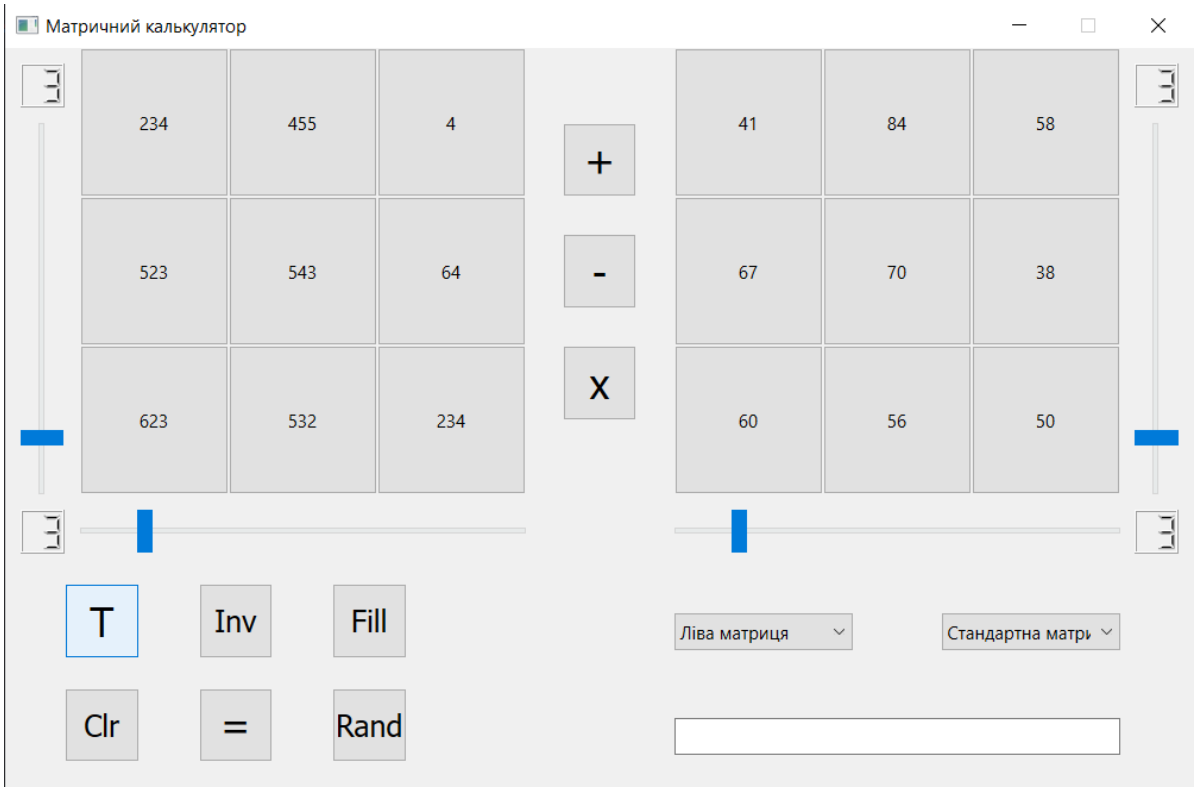

Рисунок 3.11 – Головний

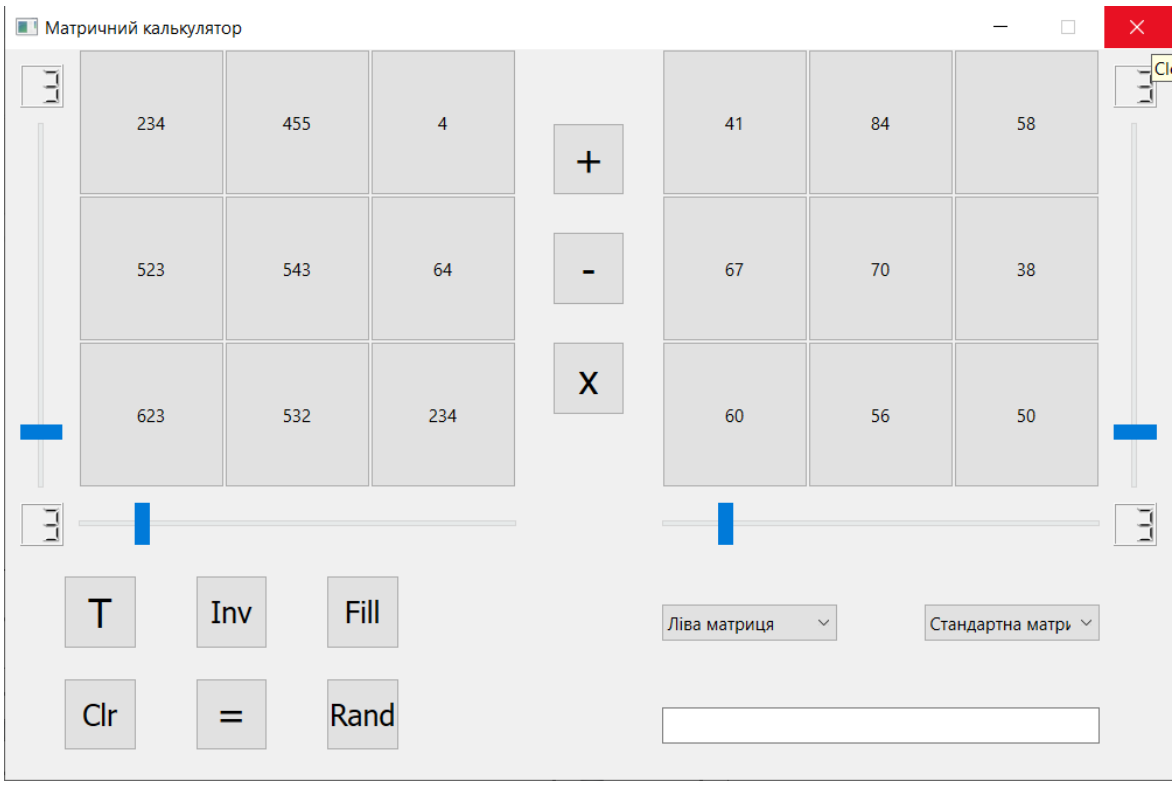

Рисунок 3.12 – Головний

ВИСНОВКИ

У ході розробки матричного калькулятора на платформі C++ Qt з використанням об’єктно-орієнтованого програмування (ООП), були вдосконалені та застосовані принципи роботи з мовою програмування C++. Програма має графічний інтерфейс, що полегшує взаємодію з користувачем.

Під час розробки використовувалися ключові концепції ООП, такі як інкапсуляція, наслідування та поліморфізм. Це сприяло створенню гнучкої та розширюваної архітектури програми.

Застосування бібліотеки Qt дозволило швидко та ефективно реалізувати графічний інтерфейс, забезпечивши зручність користування програмою. Використання класів для роботи з матрицями, таких як "Matrice," "MatriceQuadrata," і "MatriceBinaria," дозволяє легко виконувати різноманітні операції над матрицями.

Організація файлової структури дозволяє легко управляти та розширювати кодову базу програми. Використання ООП та багатошарової структури дозволяє зберігати код чистим та зрозумілим для інших розробників.

У результаті, розроблений матричний калькулятор відповідає вимогам завдання, і впроваджені техніки ООП та бібліотеки Qt сприяли в створенні функціонального та ефективного програмного продукту.

ПЕРЕЛІК ПОСИЛАНЬ

1) Грицюк Ю.І. Рак Т.Є. Об’єктно-орієнтоване програмування мовою С++ : навч. посіб. Львів : Вид-во Львівського ДУ БЖД, 2011. 404 с.

2) Build with Qt. URL: https://www1.qt.io/built-with-qt (дата звернення: 15.12.2019).

Додаток А

Лістинг файлу «kalk2.pro»

```
QT       += core gui

greaterThan(QT_MAJOR_VERSION, 4): QT += widgets

TARGET = kalk2
TEMPLATE = app

# The following define makes your compiler emit warnings if you use
# any feature of Qt which has been marked as deprecated (the exact warnings
# depend on your compiler). Please consult the documentation of the
# deprecated API in order to know how to port your code away from it.
DEFINES += QT_DEPRECATED_WARNINGS

# You can also make your code fail to compile if you use deprecated APIs.
# In order to do so, uncomment the following line.
# You can also select to disable deprecated APIs only up to a certain version of Qt.
#DEFINES += QT_DISABLE_DEPRECATED_BEFORE=0x060000     # disables all the APIs deprecated before Qt 6.0.0


SOURCES += \
        main.cpp \
        widget.cpp \
    matrice.cpp \
    matricequadrata.cpp \
    matricebinaria.cpp \
    matricegui.cpp

HEADERS += \
        widget.h \
```

```
    matrice.h \
    matricequadrata.h \
    matricebinaria.h \
    matricegui.h \
    eccezioni.h

FORMS += \
        widget.ui
```

Додаток Б

Лістинг файлу «eccezioni.h»

```
#ifndef ECCEZIONI_H
#define ECCEZIONI_H

class errSize {};
class errSizeProd {};
class errLabirinto {};
class errSudoku {};
class errSudokuInit {};
class errQuadratoMagico {};

#endif // ECCEZIONI_H
```

Додаток В

Лістинг файлу «main.cpp»

```
#include "widget.h"
#include "matrice.h"
#include "matricequadrata.h"
#include "matricebinaria.h"
#include "matriceGui.h"
#include <QApplication>


//9 luglio
int main(int argc, char *argv[])
{
  QApplication a(argc, argv);
  Widget w;
  w.show();
  return a.exec();
}
```

Додаток Г

Лістинг файлу «matrice.h»

```
#ifndef MATRICE_H
#define MATRICE_H
#include <iostream>
#include <vector>
#include <time.h> //usata per generare numeri casuali
#include <math.h>
#include <eccezioni.h>

using std::vector;
using std::cout;
using std::endl;

class matrice{
private:
  int r;
  int c;
  vector<int> pos;
public:
  matrice(int y=0, int x=0, int d=0);
  virtual matrice& operator=(const matrice&);//ridefinito per matrice
binaria
  virtual ~matrice() {}
  int read(int y=0, int x=0) const;//ritorna il valore in posizione (x,y)
  int& ref(int y=0,int x=0);//ritorna la posizione (x,y)
  virtual void stampa() const;
  bool sizeEqual(const matrice&) const;//ritorna true se le due matrici
hanno le stesse dimensioni
  virtual matrice& operator+(const matrice&);
  virtual matrice& operator-(const matrice&);
  virtual matrice& operator*(const matrice&);
  bool operator==(const matrice&) const;
```

```
  bool operator!=(const matrice&) const;
  void writeAll(int n=0);
  virtual void writeAllRandom(int min=0,int max=9);
  void trasposta();
  virtual void reverse();//non calcola l'inversa della matrice ma equivale
a moltiplicare per -1
  int getColumns() const;
  int getRows() const;
  matrice copy(matrice t, int yd=0, int xd=0, int yt=0, int xt=0,int
sizey=0, int sizex=0);//copia d in *this a partire da specifiche coordinate
  void swap(int y1=0, int x1=0, int y2=0, int x2=0, int sizey=1,int
sizex=1);//scambia due aree di una matrice di una certa dimensione
  virtual matrice* clone() const;
  int max() const;//ritornano il numero massimo e minimo nella matrice
  int min() const;
  int mostRepeated() const;//ritorna il valore che compare più volte
all'interno della matrice escludendo lo 0
};

#endif // MATRICE_H
```

Додаток Г

Лістинг файлу «matrice.cpp»

```
#include "matrice.h"

//MATRICE
matrice::matrice(int y, int x, int d): r(y),c(x) {
  for(int i=0;i<x*y;i++)
    pos.push_back(d);
}

int matrice::read(int y, int x) const{
  return pos[y*c+x];
}

int& matrice::ref(int y,int x){
  return pos[y*c+x];
}

void matrice::stampa() const{
  for(int i=0;i<r;i++){
    for(int j=0;j<c;j++)
        cout<<read(i,j)<<"  ";
    cout<<endl;
  }
}

bool matrice::sizeEqual(const matrice& m) const{
  return c==m.c && r==m.r;
}

matrice& matrice::operator=(const matrice& m){
  if(this!=&m){
    c=m.c;
```

```
    r=m.r;
    pos=m.pos;
  }
  return *this;
}

matrice& matrice::operator+(const matrice& m){
  matrice* x=new matrice(*this);
  if(!sizeEqual(m))
    throw errSize();
  for(int i=0;i<getRows();i++)
    for(int j=0;j<getColumns();j++)
      x->ref(i,j)+=m.read(i,j);
  return *x;
}

matrice& matrice::operator-(const matrice& m){
    matrice* x=new matrice(*this);
    if(!sizeEqual(m))
      throw errSize();
    for(int i=0;i<getRows();i++)
      for(int j=0;j<getColumns();j++)
        x->ref(i,j)-=m.read(i,j);
    return *x;
}

matrice& matrice::operator*(const matrice& m){
  matrice* ris=new matrice(getRows(),m.getColumns());
  if(!(getColumns()==m.getRows() && getRows()==m.getColumns()))
    throw errSizeProd();
  for(int i=0;i<ris->getRows();i++)
    for(int j=0;j<ris->getColumns();j++){
      int somma=0;
      for(int k=0;k<getColumns();k++)
```

```
        somma+=read(i,k)*m.read(k,j);
      ris->ref(i,j)=somma;
    }
  return *ris;
}

bool matrice::operator==(const matrice& m) const{
  if(!sizeEqual(m))
    return false;
  for(int i=0;i<r;i++)
    for(int j=0;j<c;j++)
      if(read(i,j)!=m.read(i,j))
        return false;
  return true;
}

bool matrice::operator!=(const matrice& m) const{
  return !(*this==m);
}

void matrice::writeAll(int n){
  for(int i=0;i<r;i++)
    for(int j=0;j<c;j++)
      ref(i,j)=n;
}

void matrice::writeAllRandom(int min,int max) {
  srand(time(NULL));
  for(int i=0;i<r;i++)
    for(int j=0;j<c;j++)
      ref(i,j)=rand()%(max+1-min)+min;
}

void matrice::trasposta(){
```

```
  matrice t(*this);
  int temp=r;
  r=c;
  c=temp;
  for(int i=0;i<r;i++)
    for(int j=0;j<c;j++)
      ref(i,j)=t.read(j,i);
}

void matrice::reverse(){
  for(int i=0;i<r;i++)
    for(int j=0;j<c;j++)
      ref(i,j)=-read(i,j);
}

int matrice::getColumns() const{
  return c;
}
int matrice::getRows() const{
  return r;
}

void matrice::swap(int y1, int x1, int y2, int x2, int sizey, int sizex){
  if(y1<0  ||  y1>=getRows()  ||  x1<0  ||  x1>=getColumns()  ||  y2<0  ||
y2>=getRows() || x2<0 || x2>=getRows() || sizey<0 || sizex<0)
    return;
  matrice temp(*this);
  copy(temp,y1,x1,y2,x2,sizey,sizex);
  copy(temp,y2,x2,y1,x1,sizey,sizex);
}

matrice matrice::copy(matrice t, int yd, int xd, int yt, int xt, int sizey,
int sizex){
  if(yd<0  ||  yd>=getRows()  ||  xd<0  ||  xd>=getColumns()  ||  yt<0  ||
```

```
yt>=t.getRows() || xt<0 || xt>=t.getRows() || sizey<0 || sizex<0)
    return *this;
  if(sizey==0)sizey=t.getRows();
  if(sizex==0)sizex=t.getColumns();
  for(int i=0;i+yd<getRows() && i<sizey;i++)
    for(int j=0;j+xd<getColumns() && j<sizex;j++)
      ref(yd+i,xd+j)=t.read(yt+i,xt+j);
  return *this;
}

matrice* matrice::clone() const {
  return new matrice(*this);
}

int matrice::max() const{
  int nMax=read(0,0);
  for(int i=0;i<r;i++)
    for(int j=0;j<c;j++)
      if(read(i,j)>nMax)
        nMax=read(i,j);
  return nMax;
}

int matrice::min() const{
  int nMin=read(0,0);
  for(int i=0;i<r;i++)
    for(int j=0;j<c;j++)
      if(read(i,j)<nMin)
        nMin=read(i,j);
  return nMin;
}

int matrice::mostRepeated() const{
  int mr=read(0,0);
```

```
  int tr=0;
  for(int i=0;i<r;i++)
    for(int j=0;j<c;j++)
      if(read(i,j)!=0){
        int n=0;
        for(int k=0;k<r;k++)
          for(int h=0;h<c;h++)
            if(read(i,j)==read(k,h))
              n++;
        if(n>tr){tr=n;mr=read(i,j);}
        }
  return mr;
}
```

Додаток Д

Лістинг файлу «matricebinaria.h»

```
#ifndef MATRICEBINARIA_H
#define MATRICEBINARIA_H
#include "matrice.h"

class matriceBinaria: public matrice{
public:
  class casella{ //questa classe serve solo per memorizzare il percorso migliore del labirinto
  public:
    int y;
    int x;
    casella(int r=0,int c=0): y(r),x(c) {}
  };
private:
  //QUESTE FUNZIONI SONO USATE PER RISOLVERE IL LABIRINTO
  static bool fine_lab(int y,int x,int ye,int xe);
  static bool bloccato(const matrice& m,int y,int x);
  static bool is_best_path(const vector<matriceBinaria::casella>& p,int y,int x);
  static bool risolvi_lab(matrice& m,int y,int x,int ye,int xe,int &minmosse,vector<matriceBinaria::casella>& p,int mosse=0);
public:
  matriceBinaria(int y=0,int x=0, int d=0);
  matriceBinaria(const matrice& m): matrice(m) {aggiusta();}
  matriceBinaria& operator=(const matrice&);
  matriceBinaria& operator+(const matrice&);
  matriceBinaria& operator-(const matrice&);
  matriceBinaria& operator*(const matrice&);
  void writeAllRandom();//non ereditata da matrice ma funzione nuova perchà cambiati i parametri
  void reverse();
```

```
  virtual matriceBinaria aggiusta();
  vector<casella>   labirinto(int   ys=0,int   xs=0,int   ye=0,int   xe=0)
const;//trova un percorso per muoversi dal punto s(y,x) al punto e(y,x),
ritorna false se il percorso non esiste
  matriceBinaria* clone() const;
};

#endif // MATRICEBINARIA_H
```

Додаток Е

Лістинг файлу «matricebinaria.cpp»

```
#include "matricebinaria.h"

//MATRICE BINARIA
matriceBinaria::matriceBinaria(int y,int x, int d): matrice(y,x,d) {
  aggiusta();
}

matriceBinaria matriceBinaria::aggiusta(){
  for(int i=0;i<getRows();i++)
    for(int j=0;j<getColumns();j++){
      if(read(i,j)<0)ref(i,j)=0;
      if(read(i,j)>1)ref(i,j)=1;
    }
  return *this;
}

matriceBinaria& matriceBinaria::operator+(const matrice& m){
    matriceBinaria* x=new matriceBinaria(*this);
    if(!sizeEqual(m))
      throw errSize();
    for(int i=0;i<getRows();i++)
      for(int j=0;j<getColumns();j++)
        x->ref(i,j)+=m.read(i,j);
    x->aggiusta();
    return *x;
}

matriceBinaria& matriceBinaria::operator-(const matrice& m){
    matriceBinaria* x=new matriceBinaria(*this);
    if(!sizeEqual(m))
      throw errSize();
```

```
    for(int i=0;i<getRows();i++)
      for(int j=0;j<getColumns();j++)
        x->ref(i,j)-=m.read(i,j);
    x->aggiusta();
    return *x;
}

matriceBinaria& matriceBinaria::operator*(const matrice& m){
    matriceBinaria* ris=new matriceBinaria(getRows(),m.getColumns());
    if(!(getColumns()==m.getRows() && getRows()==m.getColumns()))
      throw errSizeProd();
    for(int i=0;i<ris->getRows();i++)
      for(int j=0;j<ris->getColumns();j++){
        int somma=0;
        for(int k=0;k<getColumns();k++)
          somma+=read(i,k)*m.read(k,j);
        if(somma>1)somma=1;
        if(somma<0)somma=0;
        ris->ref(i,j)=somma;
      }
    return *ris;
}

void matriceBinaria::writeAllRandom(){
  matrice::writeAllRandom(0,1);
}

void matriceBinaria::reverse(){
  for(int i=0;i<getRows();i++)
    for(int j=0;j<getColumns();j++)
      if(read(i,j)==1)ref(i,j)=0;
      else ref(i,j)=1;
}
```

```
matriceBinaria& matriceBinaria::operator=(const matrice& m){
  matrice::operator=(m);
  aggiusta();
  return *this;
}

vector<matriceBinaria::casella> matriceBinaria::labirinto(int ys,int xs,int ye,int xe) const{
  if(ye==0)ye=getRows()-1;
  if(xe==0)xe=getColumns()-1;
  vector<casella> p;//questo vector memorizza le caselle del percorso migliore al termine dell'esecuzione di risolvi_lab
  int minmosse=getColumns()*getRows()+1;//al termine dell'esecuzione di risolvi_lab memorizzerà la distanza del percorso più breve
  int old_minmosse=minmosse;
  matrice m(*this);//siccome la funzione risolvi_lab ha bisogno di utilizzare altri valori all'interno della matrice allora uso una matrice normale
  risolvi_lab(m,ys,xs,ye,xe,minmosse,p);
  if(old_minmosse==minmosse){
    throw errLabirinto();
  }
  p.push_back(casella(ys,xs));//aggiungo la posizione di partenza al vettore del percorso
  return p;
}

matriceBinaria* matriceBinaria::clone() const {
  return new matriceBinaria(*this);
}

//QUESTE FUNZIONI SONO USATE PER RISOLVERE IL LABIRINTO
bool matriceBinaria::fine_lab(int y,int x,int ye,int xe){
  if(y==ye && x==xe)
```

```
    return true;
  return false;
}

bool matriceBinaria::bloccato(const matrice& m,int y,int x){
  int n=0;
  if(m.read(y+1,x)==1 || m.read(y+1,x)==3 || y+1>=m.getRows())n++;
  if(m.read(y-1,x)==1 || m.read(y-1,x)==3 || y-1<0)n++;
  if(m.read(y,x+1)==1 || m.read(y,x+1)==3 || x+1>=m.getColumns())n++;
  if(m.read(y,x-1)==1 || m.read(y,x-1)==3 || x-1<0)n++;
  if(n==4)return true;
  return false;
}

bool  matriceBinaria::is_best_path(const  vector<matriceBinaria::casella>&
p,int y,int x){
  for(unsigned int i=0;i<p.size();i++)
    if(y==p[i].y && x==p[i].x)
      return true;//ritorna true se le coordinate (y,x) si trovano su una
casella che appartiene al percorso più breve finora trovato
  return false;
}

bool matriceBinaria::risolvi_lab(matrice& m,int y,int x,int ye,int xe,int
&minmosse,vector<matriceBinaria::casella>& p,int mosse){
  int r=m.getRows();
  int c=m.getColumns();
  if(fine_lab(y,x,ye,xe) && mosse<minmosse){
    p.clear();
    minmosse=mosse;//salva il migliore percorso nell'array p
    for(int i=0;i<r;i++)
      for(int j=0;j<c;j++)
        if(m.read(i,j)==2)
          p.push_back(matriceBinaria::casella(i,j));
```

```
    return true;
  }
  if(bloccato(m,y,x) || m.read(y,x)==1)
      return false;
  for(int i=0;i<4;i++){
    int k1,k2;
    if(i==0){k1=1;k2=0;}//basso
    if(i==1){k1=0;k2=1;}//destra
    if(i==2){k1=-1;k2=0;}//alto
    if(i==3){k1=0;k2=-1;}//sinistra
    if(m.read(y+k1,x+k2)==0 && y+k1<r && x+k2<c && y+k1>=0 && x+k2>=0){
      m.ref(y+k1,x+k2)=2;
      if(!risolvi_lab(m,y+k1,x+k2,ye,xe,minmosse,p,mosse+1)              &&
!is_best_path(p,y+k1,x+k2))
          m.ref(y+k1,x+k2)=3;
      else
          m.ref(y+k1,x+k2)=0;
    }
  }
  return false;
}
```

Додаток Е

Лістинг файлу «matricegui.h»

```
#ifndef MATRICEGUI_H
#define MATRICEGUI_H

#include <QColor>
#include <QWidget>
#include <QtWidgets>
#include <QAbstractButton>
#include <QPushButton>
#include <QString>
#include <QLabel>
#include <QFrame>
#include "matrice.h"
#include "matricequadrata.h"
#include "matricebinaria.h"

class matriceGui : public QWidget
{
  Q_OBJECT
private:
  class cell: public QPushButton{
    public:
      cell(int v=0, QWidget *parent = 0);
      void setNumText(int n=0);
  };
  int sel_y;
  int sel_x;
  QVector<cell*> cel;
  int size;
  int cellSize;
  int rows;
```

```
  int columns;
  void keyPressEvent(QKeyEvent *event) override;
  matrice* matrix;

public:
  int x,y;//posizione rispetto al widget parent
  matriceGui(const matrice&, int s=300, QWidget *parent = 0);
  ~matriceGui();
  int getSize() const;
  int getColumns() const;
  int getRows() const;
  void setDim(int r,int c);//cambia le dimensioni della matrice grafica
  void update();//aggiorna la matrice grafica quando vengono cambiati i valori della matrice interna
  void markCells(const vector<matriceBinaria::casella>& c);//dato un insieme di caselle, evidenzia le celle corrispondenti nella matrice grafica
  matrice* getMatrix() const;
  void writeMatrix(matrice* m);
public slots:
  void selChanged();
};

#endif // MATRICEGUI_H
```

Додаток Є

Лістинг файлу «matricegui.cpp»

```
#include <QtWidgets>
#include "matricegui.h"

matriceGui::cell::cell(int v, QWidget *parent): QPushButton(parent){
  setNumText(v);
}

void matriceGui::cell::setNumText(int n){
  QString s;
  s.setNum(n);
  setText(s);
}

matriceGui::matriceGui(const matrice& m, int s, QWidget *parent):
QWidget(parent),sel_y(0),sel_x(0),size(s),x(0),y(0),matrix(m.clone()){
  columns=m.getColumns();
  rows=m.getRows();
  resize(size,size);
  if(columns>=rows)
    cellSize=size/columns;
  else
    cellSize=size/rows;
  for(int i=0;i<rows;i++)
    for(int j=0;j<columns;j++){
      cel.push_back(new cell(matrix->read(i,j),this));
      cel[i*columns+j]-
>setGeometry(j*cellSize,i*cellSize,cellSize,cellSize);
      cel[i*columns+j]->setAutoFillBackground(1);//If enabled, this
property will cause Qt to fill the background of the widget before invoking
the paint event. The color used is defined by the QPalette::Window color
```

```
role from the widget's palette
      connect(cel[i*columns+j],SIGNAL(pressed()),this,SLOT(selChanged()));
    }
}
matriceGui::~matriceGui() {delete matrix;}

void matriceGui::selChanged(){
  for(int i=0;i<rows;i++)
    for(int j=0;j<columns;j++){
       cel[i*columns+j]->setPalette(QPalette(QColor(255,255,255)));
       if(cel[i*columns+j]->isDown()){
          sel_y=i;
          sel_x=j;
          cel[i*columns+j]->setPalette(QPalette(QColor(0,0,255)));
       }
    }
  if(dynamic_cast<matriceBinaria*>(matrix)){//se  la  matrice  è  binaria
l'input viene dato solo col mouse (un click cambia il valore della casella)
    if(matrix->read(sel_y,sel_x)==0)matrix->ref(sel_y,sel_x)=1;
    else matrix->ref(sel_y,sel_x)=0;
    cel[sel_y*columns+sel_x]->setNumText(matrix->read(sel_y,sel_x));
  }
}

void matriceGui::keyPressEvent(QKeyEvent *event){
  if(dynamic_cast<matriceBinaria*>(matrix))
    return;
  int digit=-1;
  switch (event->key()) {
    case Qt::Key_0: digit=0;break;
    case Qt::Key_1: digit=1;break;
    case Qt::Key_2: digit=2;break;
    case Qt::Key_3: digit=3;break;
    case Qt::Key_4: digit=4;break;
```

```
    case Qt::Key_5: digit=5;break;
    case Qt::Key_6: digit=6;break;
    case Qt::Key_7: digit=7;break;
    case Qt::Key_8: digit=8;break;
    case Qt::Key_9: digit=9;break;
    case Qt::Key_Minus: if(matrix->read(sel_y,sel_x)>0){
        matrix->ref(sel_y,sel_x)=-matrix->read(sel_y,sel_x);
        cel[sel_y*columns+sel_x]->setNumText(matrix->read(sel_y,sel_x));
      }
      break;
    case Qt::Key_Plus: if(matrix->read(sel_y,sel_x)<0){
        matrix->ref(sel_y,sel_x)=-matrix->read(sel_y,sel_x);
        cel[sel_y*columns+sel_x]->setNumText(matrix->read(sel_y,sel_x));
      }
      break;
    case Qt::Key_Backspace:
      if(abs(matrix->read(sel_y,sel_x))<10){
        matrix->ref(sel_y,sel_x)=0;
        cel[sel_y*columns+sel_x]->setNumText(matrix->read(sel_y,sel_x));
      }
      else{
        matrix->ref(sel_y,sel_x)=matrix->read(sel_y,sel_x)/10;
        cel[sel_y*columns+sel_x]->setNumText(matrix->read(sel_y,sel_x));
      }
  }
  if(digit>=0){
    if(matrix->read(sel_y,sel_x)==0){
      matrix->ref(sel_y,sel_x)=digit;
      cel[sel_y*columns+sel_x]->setNumText(matrix->read(sel_y,sel_x));
    }
    else{
      int num;
      if(matrix->read(sel_y,sel_x)<0)
        num=-(abs(matrix->read(sel_y,sel_x))*10+digit);
```

```
      else num=matrix->read(sel_y,sel_x)*10+digit;
      matrix->ref(sel_y,sel_x)=num;
      cel[sel_y*columns+sel_x]->setNumText(matrix->read(sel_y,sel_x));
    }
  }
}

int matriceGui::getSize() const{
  return size;
}

int matriceGui::getColumns() const{
  return columns;
}

int matriceGui::getRows() const{
  return rows;
}

void matriceGui::update(){
  for(int i=0;i<columns*rows;i++)
    delete cel[i];
  sel_y=0;sel_x=0;
  columns=matrix->getColumns();
  rows=matrix->getRows();
  if(columns>=rows)cellSize=size/columns;
  else cellSize=size/rows;
  cel.clear();//rimetto le celle con il valore aggiornato
  for(int i=0;i<rows;i++)
    for(int j=0;j<columns;j++){
      cel.push_back(new cell(matrix->read(i,j),this));
      cel[i*columns+j]-
>setGeometry(j*cellSize,i*cellSize,cellSize,cellSize);
      cel[i*columns+j]->setAutoFillBackground(1);
```

```
      connect(cel[i*columns+j],SIGNAL(pressed()),this,SLOT(selChanged()));
      cel[i*columns+j]->show();
    }
}

void matriceGui::setDim(int r,int c){
  matrice* t;
  if(dynamic_cast<matriceQuadrata*>(matrix)){
    t=new matriceQuadrata(r);
  }
  if(dynamic_cast<matriceBinaria*>(matrix)){
    t=new matriceBinaria(r,c);
  }
  if(dynamic_cast<matrice*>(matrix)){
    t=new  matrice(r,c);//viene  creata  una  nuova  matrice  con  le  nuove
dimensioni e mantendo gli stessi valori se possibile
  }
  t->copy(*matrix);
  *matrix=*t;
  update();
  delete t;
}

void matriceGui::markCells(const vector<matriceBinaria::casella> &c){
  for(int i=0;i<rows;i++)
    for(int j=0;j<columns;j++)
       cel[i*columns+j]->setPalette(QPalette(QColor(255,255,255)));
  for(unsigned int i=0;i<c.size();i++){
    cel[c[i].y*columns+c[i].x]->setPalette(QPalette(QColor(0,0,255)));
  }
}

matrice* matriceGui::getMatrix() const{
  return matrix;
```

```
}
void matriceGui::writeMatrix(matrice* m){
  matrix=m;
}
```

Додаток З

Лістинг файлу «matricequadrata.h»

```
#ifndef MATRICEQUADRATA_H
#define MATRICEQUADRATA_H
#include "matrice.h"

class matriceQuadrata: public matrice{
private:
  static bool check_full(const matriceQuadrata&);
  static bool check_sudoku(const matriceQuadrata&);
  static bool risolvi_sudoku(matriceQuadrata&);
  static matriceQuadrata quadratoMagicoDispari(matriceQuadrata);
  static matriceQuadrata quadratoMagicoPari(matriceQuadrata);
  static matriceQuadrata quadratoMagicoPariDiv4(matriceQuadrata);
public:
  matriceQuadrata(int x=0, int d=0);
  matriceQuadrata& operator+(const matrice&);
  matriceQuadrata& operator-(const matrice&);
  matriceQuadrata& operator*(const matrice&);
  matriceQuadrata sudoku();//riempie tutti gli spazi con le regole del
sudoku
  matriceQuadrata quadratoMagico();//riempie tutti gli spazi con le regole
del quadrato magico (solo per grandezza >2)
  matriceQuadrata* clone() const;
};

#endif // MATRICEQUADRATA_H
```

Додаток И

Лістинг файлу «matricequadrata.cpp»

```
#include "matricequadrata.h"

//MATRICE QUADRATA
matriceQuadrata::matriceQuadrata(int x, int d): matrice(x,x,d) {}

matriceQuadrata& matriceQuadrata::operator+(const matrice& m){
    matriceQuadrata* x=new matriceQuadrata(*this);
    if(!sizeEqual(m))
      throw errSize();
    for(int i=0;i<getRows();i++)
      for(int j=0;j<getRows();j++)
        x->ref(i,j)+=m.read(i,j);
    return *x;
}

matriceQuadrata& matriceQuadrata::operator-(const matrice& m){
    matriceQuadrata* x=new matriceQuadrata(*this);
    if(!sizeEqual(m))
      throw errSize();
    for(int i=0;i<getRows();i++)
      for(int j=0;j<getRows();j++)
        x->ref(i,j)+=m.read(i,j);
    return *x;
}

matriceQuadrata& matriceQuadrata::operator*(const matrice& m){
    matriceQuadrata* ris=new matriceQuadrata(getRows(),m.getRows());
    if(!sizeEqual(m))
      throw errSizeProd();
    for(int i=0;i<ris->getRows();i++)
      for(int j=0;j<ris->getRows();j++){
```

```
        int somma=0;
        for(int k=0;k<getRows();k++)
          somma+=read(i,k)*m.read(k,j);
        ris->ref(i,j)=somma;
      }
    return *ris;
}

matriceQuadrata matriceQuadrata::quadratoMagicoDispari(matriceQuadrata m){
  int dim=m.getColumns();
  m.writeAll(0);
  int x=(dim-1)/2,y=0;
  for(int i=1;i<=dim*dim;i++){
    m.ref(y,x)=i;
    int old_x=x,old_y=y;
    y--;
    if(y<0)y=dim-1;
    x++;
    if(x>=dim)x=0;
    if(m.read(y,x)!=0){
      x=old_x;
      y=old_y+1;
    }
  }
  return m;
}

matriceQuadrata matriceQuadrata::quadratoMagicoPari(matriceQuadrata m){
  int hdim=m.getColumns()/2;
  matriceQuadrata t(hdim);
  matriceQuadrata a(quadratoMagicoDispari(t));
  matriceQuadrata b(quadratoMagicoDispari(t));
  b+matriceQuadrata(b.getColumns(),hdim*hdim);
  matriceQuadrata c(quadratoMagicoDispari(t));
```

```
  c+matriceQuadrata(c.getColumns(),hdim*hdim*2);
  matriceQuadrata d(quadratoMagicoDispari(t));
  d+matriceQuadrata(d.getColumns(),hdim*hdim*3);
  m.copy(a,0,0);
  m.copy(b,hdim,hdim);
  m.copy(c,0,hdim);
  m.copy(d,hdim,0);
  int hhdim=(hdim-1)/2;
  m.swap(0,0,hdim,0,hhdim,hhdim);
  m.swap(hhdim,1,hhdim+hdim,1,1,hhdim*hhdim);
  m.swap(hhdim+1,0,hhdim+hdim+1,0,hhdim,hhdim);
  return m;
}

matriceQuadrata matriceQuadrata::quadratoMagicoPariDiv4(matriceQuadrata m){
  int hhdim=m.getColumns()/4;
  int dim=m.getColumns();
  for(int i=0;i<m.getColumns();i++)
    for(int j=0;j<m.getColumns();j++)
      if((i<hhdim  &&  (j<hhdim  ||  j>=dim-hhdim))  ||  (i>=dim-hhdim  &&
(j<hhdim  ||  j>=dim-hhdim))  ||  (i>=hhdim  &&  i<dim-hhdim  &&  j>=hhdim  &&
j<dim-hhdim))
        m.ref(i,j)=i*dim+j+1;
      else
        m.ref(i,j)=dim*dim-(i*dim+j);
  return m;
 }

matriceQuadrata matriceQuadrata::quadratoMagico(){
  if(getColumns()<3){
    throw errQuadratoMagico();
  }
  if(getColumns()%2==1)
    *this=quadratoMagicoDispari(*this);
```

```
  else
    if(getColumns()%4==0)
      *this=quadratoMagicoPariDiv4(*this);
    else
      *this=quadratoMagicoPari(*this);
  return *this;
}

matriceQuadrata matriceQuadrata::sudoku(){
  int rad=sqrt(getColumns());
  if(rad<2){
    throw errSudoku();
  }
  if(rad*rad!=getColumns()){
    throw errSudoku();
  }
  for(int i=0;i<getColumns();i++)
    for(int j=0;j<getColumns();j++)
      if(read(i,j)<0 || read(i,j)>getColumns()){
        throw errSudokuInit();
      }
  matriceQuadrata m(*this);
  if(risolvi_sudoku(m))
    return *this=m;
  throw errSudokuInit();
}

//QUESTE FUNZIONI SONO USATE PER RISOLVERE IL SUDOKU
bool matriceQuadrata::check_full(const matriceQuadrata& m){
  for(int i=0;i<m.getColumns();i++)
    for(int j=0;j<m.getColumns();j++)
      if(m.read(i,j)==0)
        return false;
  return true;
```

```
}

bool matriceQuadrata::check_sudoku(const matriceQuadrata& m){
  int dim=sqrt(m.getColumns());
  for(int i=0;i<m.getColumns();i++)
    for(int j=0;j<m.getColumns();j++)
      if(m.read(i,j)!=0)
        for(int k=0;k<m.getColumns();k++){
          if((m.read(i,j)==m.read(i,k)         &&         k!=j)         ||
(m.read(i,j)==m.read(k,j) && k!=i))//ci sono 2 numeri uguali nella stessa
riga o colonna
            return false;
          for(int h=0;h<m.getColumns();h++)
            if(m.read(i,j)==m.read(k,h) && k/dim==i/dim && h/dim==j/dim &&
(k!=i && h!=j))
              return false;
      }
  return true;
}

bool matriceQuadrata::risolvi_sudoku(matriceQuadrata& m){
  if(check_sudoku(m) && check_full(m))
    return true;
  if(!check_sudoku(m))
    return false;
  for(int i=0;i<m.getColumns();i++)
    for(int j=0;j<m.getColumns();j++)
      if(m.read(i,j)==0)
        for(int k=1;k<=m.getColumns();k++){
          m.ref(i,j)=k;
          if(risolvi_sudoku(m))
            return true;
          m.ref(i,j)=0;
          if(m.read(i,j)==0 && k==m.getColumns())
```

```
            return false;
    }
  return check_sudoku(m);
}

matriceQuadrata* matriceQuadrata::clone() const {
  return new matriceQuadrata(*this);
}
```

Додаток І

Лістинг файлу «widget.h»

```
#ifndef WIDGET_H
#define WIDGET_H

#include <QWidget>
#include <QSlider>
#include <QLCDNumber>
#include "matricegui.h"

class Widget : public QWidget
{
  Q_OBJECT

public:
  explicit Widget(QWidget *parent = 0);
  ~Widget();

private:
  //VARIABILI
  int matriceSelect;
  //MATRICI
  matriceGui *mat[2];
  //slider per modificare le dimensioni delle matrici
  QSlider *Smat1_y;
  QSlider *Smat1_x;
  QSlider *Smat2_y;
  QSlider *Smat2_x;
  //display per visualizzare le dimensioni delle matrici
  QLCDNumber *Dmat1_y;
  QLCDNumber *Dmat1_x;
  QLCDNumber *Dmat2_y;
  QLCDNumber *Dmat2_x;
```

```
  //OPERATORI BINARI
  QPushButton *somma;
  QPushButton *sottrazione;
  QPushButton *prodotto;
  //OPERATORI UNARI
  QPushButton *trasposta;
  QPushButton *copia;
  QPushButton *inversa;
  QPushButton *fill;
  QPushButton *random;
  QPushButton *clear;
  //FINESTRE PER SELEZIONI
  QComboBox *activeMatComboBox;
  QComboBox *typeMatComboBox;
  //OPERATORI SPECIALI
  QPushButton *quadratoMagico;
  QPushButton *sudoku;
  QPushButton *labirinto;
  //FINESTRA PER MESSAGGI DI ERRORE
  QLineEdit *display;
  //METODI
  int notActiveMat() const;//ritorna il numero associato alla matrice non
selezionata
  void updateDim();//mostra le giuste dimensioni delle matrici

private slots:
  void sizeYmat1changed(int);
  void sizeXmat1changed(int);
  void sizeYmat2changed(int);
  void sizeXmat2changed(int);

  void activeMatChanged();
  void typeMatChanged();
```

```
    void opSomma();
    void opSottrazione();
    void opProdotto();

    void opTrasposta();
    void opCopia();
    void opInversa();
    void opFill();
    void opRandom();
    void opClear();

    void opQuadratoMagico();
    void opSudoku();
    void opLabirinto();
};

#endif // WIDGET_H
```

Додаток Ї

Лістинг файлу «widget.cpp»

```
#include "widget.h"

Widget::Widget(QWidget *parent): QWidget(parent){
  setFixedSize(800,500);
  setWindowTitle("Матричний калькулятор");
  //BOXES FOR MATRICES
  QLabel* l1 = new QLabel(this);
  l1->setGeometry(50,0,300,300);
  l1->setFrameStyle(QFrame::Box | QFrame::Raised);
  QLabel* l2 = new QLabel(this);
  l2->setGeometry(450,0,300,300);
  l2->setFrameStyle(QFrame::Box | QFrame::Raised);
  //MATRICS
  matriceSelect=0;
  mat[0]=new matriceGui(matrice(9,9),300,this);
  mat[1]=new matriceGui(matrice(9,9),300,this);
  mat[0]->x=50;
  mat[1]->x=450;
  mat[0]->setGeometry(mat[0]->x,0,300,300);
  mat[1]->setGeometry(mat[1]->x,0,300,300);
  //SLIDER Y.1
  Smat1_y=new QSlider(Qt::Vertical,this);
  Smat1_y->setValue(9);
  Smat1_y->setRange(1,16);
  Smat1_y->setGeometry(10,50,30,250);
  Dmat1_y=new QLCDNumber(2,this);
  Dmat1_y->setGeometry(10,10,30,30);
  connect(Smat1_y,SIGNAL(valueChanged(int)),Dmat1_y,SLOT(display(int)));

connect(Smat1_y,SIGNAL(valueChanged(int)),this,SLOT(sizeYmat1changed(int)))
;
```

```
  //SLIDER X.1
  Smat1_x=new QSlider(Qt::Horizontal,this);
  Smat1_x->setValue(9);
  Smat1_x->setRange(1,16);
  Smat1_x->setGeometry(50,310,300,30);
  Dmat1_x=new QLCDNumber(2,this);
  Dmat1_x->setGeometry(10,310,30,30);
  connect(Smat1_x,SIGNAL(valueChanged(int)),Dmat1_x,SLOT(display(int)));

connect(Smat1_x,SIGNAL(valueChanged(int)),this,SLOT(sizeXmat1changed(int)))
;
  //SLIDER Y.2
  Smat2_y=new QSlider(Qt::Vertical,this);
  Smat2_y->setValue(9);
  Smat2_y->setRange(1,16);
  Smat2_y->setGeometry(760,50,30,250);
  Dmat2_y=new QLCDNumber(2,this);
  Dmat2_y->setGeometry(760,10,30,30);
  connect(Smat2_y,SIGNAL(valueChanged(int)),Dmat2_y,SLOT(display(int)));

connect(Smat2_y,SIGNAL(valueChanged(int)),this,SLOT(sizeYmat2changed(int)))
;
  //SLIDER X.2
  Smat2_x=new QSlider(Qt::Horizontal,this);
  Smat2_x->setValue(9);
  Smat2_x->setRange(1,16);
  Smat2_x->setGeometry(450,310,300,30);
  Dmat2_x=new QLCDNumber(2,this);
  Dmat2_x->setGeometry(760,310,30,30);
  connect(Smat2_x,SIGNAL(valueChanged(int)),Dmat2_x,SLOT(display(int)));

connect(Smat2_x,SIGNAL(valueChanged(int)),this,SLOT(sizeXmat2changed(int)))
;
  emit Smat1_y->valueChanged(9);
```

```
  emit Smat1_x->valueChanged(9);
  emit Smat2_y->valueChanged(9);
  emit Smat2_x->valueChanged(9);
  //BINARY OPERATORS (ADDITION, SUBTRACTION, PRODUCT)
  somma=new QPushButton(tr("+"),this);
  somma->setGeometry(375,50,50,50);
  somma->setFont(QFont("Times",20));
  connect(somma,SIGNAL(pressed()),this,SLOT(opSomma()));
  sottrazione=new QPushButton(tr("-"),this);
  sottrazione->setGeometry(375,125,50,50);
  sottrazione->setFont(QFont("Times",20));
  connect(sottrazione,SIGNAL(pressed()),this,SLOT(opSottrazione()));
  prodotto=new QPushButton(tr("x"),this);
  prodotto->setGeometry(375,200,50,50);
  prodotto->setFont(QFont("Times",20));
  connect(prodotto,SIGNAL(pressed()),this,SLOT(opProdotto()));
  //WINDOWS FOR SELECTIONS
  activeMatComboBox=new QComboBox(this);
  activeMatComboBox->addItem(tr("Ліва матриця"));
  activeMatComboBox->addItem(tr("Права матриця"));
  activeMatComboBox->setGeometry(450,350+30,120,25);

connect(activeMatComboBox,SIGNAL(activated(int)),this,SLOT(activeMatChanged
()));
  typeMatComboBox=new QComboBox(this);
  typeMatComboBox->addItem(tr("Стандартна матриця"));
  typeMatComboBox->addItem(tr("Квадратна матриця"));
  typeMatComboBox->addItem(tr("Бінарна матриця"));
  typeMatComboBox->setGeometry(630,350+30,120,25);

connect(typeMatComboBox,SIGNAL(activated(int)),this,SLOT(typeMatChanged()))
;
  //UNARY OPERATORS
  trasposta=new QPushButton(tr("T"),this);
```

```
trasposta->setGeometry(40,10+350,50,50);
trasposta->setFont(QFont("Times",20));
connect(trasposta,SIGNAL(pressed()),this,SLOT(opTrasposta()));
inversa=new QPushButton(tr("Inv"),this);
inversa->setGeometry(130,10+350,50,50);
inversa->setFont(QFont("Times",15));
connect(inversa,SIGNAL(pressed()),this,SLOT(opInversa()));
fill=new QPushButton(tr("Fill"),this);
fill->setGeometry(220,10+350,50,50);
fill->setFont(QFont("Times",15));
connect(fill,SIGNAL(pressed()),this,SLOT(opFill()));
clear=new QPushButton(tr("Clr"),this);
clear->setGeometry(40,80+350,50,50);
clear->setFont(QFont("Times",15));
connect(clear,SIGNAL(pressed()),this,SLOT(opClear()));
copia=new QPushButton(tr("="),this);
copia->setGeometry(130,80+350,50,50);
copia->setFont(QFont("Times",20));
connect(copia,SIGNAL(pressed()),this,SLOT(opCopia()));
random=new QPushButton(tr("Rand"),this);
random->setGeometry(220,80+350,50,50);
random->setFont(QFont("Times",15));
connect(random,SIGNAL(pressed()),this,SLOT(opRandom()));
//SPECIAL OPERATORS
quadratoMagico=new QPushButton(tr("M"),this);
quadratoMagico->setGeometry(310,10+350,50,50);
quadratoMagico->setFont(QFont("Times",20));
quadratoMagico->setEnabled(0);
quadratoMagico->hide();
connect(quadratoMagico,SIGNAL(pressed()),this,SLOT(opQuadratoMagico()));
sudoku=new QPushButton(tr("S"),this);
sudoku->setGeometry(310,80+350,50,50);
sudoku->setFont(QFont("Times",20));
sudoku->setEnabled(0);
```

```
  sudoku->hide();
  connect(sudoku,SIGNAL(pressed()),this,SLOT(opSudoku()));
  labirinto=new QPushButton(tr("L"),this);
  labirinto->setGeometry(310,10+350,50,50);
  labirinto->setFont(QFont("Times",20));
  labirinto->setEnabled(0);
  labirinto->hide();
  connect(labirinto,SIGNAL(pressed()),this,SLOT(opLabirinto()));
  //WINDOW FOR ERROR MESSAGES
  display=new QLineEdit(" ",this);
  display->setGeometry(450,350+100,300,25);
  display->setReadOnly(true);
  display->setAlignment(Qt::AlignLeft);
}

Widget::~Widget(){
}

void Widget::sizeYmat1changed(int n){
  mat[0]->setDim(n,mat[0]->getColumns());
  if(dynamic_cast<matriceQuadrata*>(mat[0]->getMatrix()))
    Smat1_x->setValue(Smat1_y->value());
}

void Widget::sizeXmat1changed(int n){
  mat[0]->setDim(mat[0]->getRows(),n);
  if(dynamic_cast<matriceQuadrata*>(mat[0]->getMatrix()))
    Smat1_y->setValue(Smat1_x->value());
}

void Widget::sizeYmat2changed(int n){
  mat[1]->setDim(n,mat[1]->getColumns());
  if(dynamic_cast<matriceQuadrata*>(mat[1]->getMatrix()))
    Smat2_x->setValue(Smat2_y->value());
```

```
}

void Widget::sizeXmat2changed(int n){
  mat[1]->setDim(mat[1]->getRows(),n);
  if(dynamic_cast<matriceQuadrata*>(mat[1]->getMatrix()))
    Smat2_y->setValue(Smat2_x->value());
}

void Widget::activeMatChanged(){
  matriceSelect=activeMatComboBox->currentIndex();
}

void Widget::opSomma(){
  display->clear();
  try{
    mat[matriceSelect]->writeMatrix(&(*mat[matriceSelect]-
>getMatrix()+*mat[notActiveMat()]->getMatrix()));
  }
  catch (errSize){
      display->setText("Дві матриці різного розміру не можна додавати
разом");}
  emit mat[matriceSelect]->update();
}

void Widget::opSottrazione(){
  display->clear();
  try{
    mat[matriceSelect]->writeMatrix(&(*mat[matriceSelect]->getMatrix()-
*mat[notActiveMat()]->getMatrix()));
  }
  catch (errSize){
      display->setText("Не можна віднімати дві матриці різних розмірів");}
  emit mat[matriceSelect]->update();
}
```

```
void Widget::opProdotto(){
  display->clear();
  try{
    mat[matriceSelect]->writeMatrix(&(*mat[matriceSelect]-
>getMatrix()*(*mat[notActiveMat()]->getMatrix())));
  }
  catch (errSizeProd){
      display->setText("Розміри матриці не відносяться до виробу");}
  emit mat[matriceSelect]->update();
  updateDim();
}

void Widget::opTrasposta(){
  mat[matriceSelect]->getMatrix()->trasposta();
  emit mat[matriceSelect]->update();
  updateDim();
}

void Widget::opInversa(){
  mat[matriceSelect]->getMatrix()->reverse();
  emit mat[matriceSelect]->update();
}

void Widget::opFill(){
  mat[matriceSelect]->getMatrix()->writeAll(mat[matriceSelect]-
>getMatrix()->mostRepeated());
  emit mat[matriceSelect]->update();
}

void Widget::opClear(){
  mat[matriceSelect]->getMatrix()->writeAll(0);
  emit mat[matriceSelect]->update();
}
```

```
void Widget::opCopia(){
  delete mat[matriceSelect]->getMatrix();
  mat[matriceSelect]->writeMatrix(mat[notActiveMat()]->getMatrix()-
>clone());
  emit mat[matriceSelect]->update();
}

void Widget::opRandom(){
  if(matriceBinaria*    b=dynamic_cast<matriceBinaria*>(mat[matriceSelect]-
>getMatrix())){
    b->writeAllRandom();
    mat[matriceSelect]->writeMatrix(b);
  }
  else
    mat[matriceSelect]->getMatrix()->writeAllRandom(mat[matriceSelect]-
>getMatrix()->min(),mat[matriceSelect]->getMatrix()->max());
  emit mat[matriceSelect]->update();
}

void Widget::opQuadratoMagico(){
  display->clear();
  matriceQuadrata*     m=dynamic_cast<matriceQuadrata*>(mat[matriceSelect]-
>getMatrix());
  try{
    m->quadratoMagico();}
  catch(errQuadratoMagico){
    display->setText("Матриця повинна мати щонайменше 3 рядки та 3
стовпці");}
  mat[matriceSelect]->writeMatrix(m);
  emit mat[matriceSelect]->update();
}

void Widget::opSudoku(){
```

```
  display->clear();
  matriceQuadrata*     m=dynamic_cast<matriceQuadrata*>(mat[matriceSelect]-
>getMatrix());
  try{
    m->sudoku();}
  catch (errSudoku){
    display->setText("Приймаються матриці розміром 4x4, 9x9, 16x16");}
  catch (errSudokuInit){
    display->setText("Початкове судоку неправильне");}
  mat[matriceSelect]->writeMatrix(m);
  emit mat[matriceSelect]->update();
}

void Widget::opLabirinto(){
  display->clear();
  matriceBinaria*       m=dynamic_cast<matriceBinaria*>(mat[matriceSelect]-
>getMatrix());
  vector<matriceBinaria::casella> p;
  try{
    p=m->labirinto();}
  catch(errLabirinto){
    display->setText("Лабіринт неможливий");}
  mat[matriceSelect]->writeMatrix(m);
  mat[matriceSelect]->markCells(p);
}

void Widget::typeMatChanged(){
  quadratoMagico->setEnabled(0);
  quadratoMagico->hide();
  sudoku->setEnabled(0);
  sudoku->hide();
  labirinto->setEnabled(0);
  labirinto->hide();
  for(int i=0;i<2;i++){
```

```
    if(typeMatComboBox->currentIndex()==0){//matrice standard
      matrice newMat(mat[i]->getRows(),mat[i]->getColumns());
      newMat.copy(*mat[i]->getMatrix());
      delete mat[i];
      mat[i]=new matriceGui(newMat,300,this);
    }
    if(typeMatComboBox->currentIndex()==1){//matrice quadrata
      quadratoMagico->setEnabled(1);
      quadratoMagico->show();
      sudoku->setEnabled(1);
      sudoku->show();
      int dim=mat[i]->getColumns();
      if(mat[i]->getRows()>mat[i]->getColumns()) dim=mat[i]->getRows();
      matriceQuadrata newMat(dim);
      newMat.copy(*mat[i]->getMatrix());
      delete mat[i];
      mat[i]=new matriceGui(newMat,300,this);
      updateDim();
    }
    if(typeMatComboBox->currentIndex()==2){//matrice binaria
      labirinto->setEnabled(1);
      labirinto->show();
      matriceBinaria newMat(mat[i]->getRows(),mat[i]->getColumns());
      delete mat[i];
      mat[i]=new matriceGui(newMat,300,this);
    }
    if(i==0)mat[i]->x=50;
    else mat[i]->x=450;
    mat[i]->setGeometry(mat[i]->x,0,300,300);
    mat[i]->show();
  }
}

int Widget::notActiveMat() const{
```

```
  if(matriceSelect==0)return 1;
  else return 0;
}

void Widget::updateDim(){
  emit Smat1_y->valueChanged(mat[0]->getRows());
  emit Smat1_x->valueChanged(mat[0]->getColumns());
  emit Smat2_y->valueChanged(mat[1]->getRows());
  emit Smat2_x->valueChanged(mat[1]->getColumns());
}
```

Додаток Й

Компакт-диск із програмним продуктом