|  |
| --- |
| [Название организации] |
| [Заголовок документа] |
| [Подзаголовок документа] |

|  |
| --- |
|  |

**Зміст**

1 ЗАГАЛЬНА ЧАСТИНА 2

1.1 Постановка задачі 2

1.2 Дослідження і аналіз об'єкту програмування С++ 3

1.2.1 Шифр «Магічний квадрат» 3

1.2.2 Квадрат Полібія 3

1.2.3 Порівняльний аналіз 4

1.3 Використані програмні засоби 4

1.3.1 Середовище розробки 4

1.3.2 Мова програмування 5

1.3.3 Додаткові бібліотеки 5

1.4 Вимоги до апаратного та програмного забезпечення 5

1.4.1 Апаратні вимоги 5

2 ПРАКТИЧНА ЧАСТИНА 7

2.1 Створення і налагодження програми 7

2.2 Опис першої програми та її алгоритмів 7

2.2.1 Математичні основи алгоритму 8

2.2.2 Алгоритм шифрування 8

2.2.3 Алгоритм дешифрування 8

2.2.4 Структура програми 9

2.2.5 Особливості реалізації 9

2.3 Опис другої програми та її алгоритмів 9

2.3.1 Математичні основи алгоритму 10

2.3.2 Алгоритм шифрування 10

2.3.3 Алгоритм дешифрування 10

2.3.4 Структура програми 11

2.3.5 Особливості реалізації 11

2.3.6 Криптографічний аналіз 12

2.4 Інструкція програміста і оператора до програми 12

ВИСНОВОК 14

ДОДАТОК А 15

# ЗАГАЛЬНА ЧАСТИНА

## Постановка задачі

Метою даної курсової роботи є дослідження та практична реалізація класичних криптографічних алгоритмів перестановки, а саме шифру «Магічний квадрат» та квадрату Полібія. Ці алгоритми представляють собою фундаментальні методи криптографії, які демонструють принципи перестановки символів та заміщення букв відповідно.

Основними завданнями роботи є:

Теоретичне дослідження принципів роботи обраних криптографічних алгоритмів, включаючи вивчення їх історичного розвитку та математичних основ. Важливо розуміти, що шифр «Магічний квадрат» базується на принципі перестановки символів повідомлення згідно з певною матрицею, де кожен елемент має унікальне числове значення. Квадрат Полібія, у свою чергу, є методом заміщення, який кодує кожну букву алфавіту парою координат у квадратній таблиці.

Аналіз переваг та недоліків кожного з алгоритмів з точки зору криптографічної стійкості є наступним важливим завданням. Необхідно оцінити рівень захисту, який забезпечують ці методи, та визначити їх уразливості до різних типів криптоаналітичних атак.

Практична реалізація обох алгоритмів у програмному середовищі дозволить продемонструвати їх функціонування та провести порівняльний аналіз ефективності. Це включає розробку програмного коду для шифрування та дешифрування повідомлень.

Проведення тестування розроблених програм на різних типах вхідних даних забезпечить перевірку коректності реалізації та виявлення можливих помилок у логіці алгоритмів.

Створення зручного користувацького інтерфейсу для взаємодії з реалізованими алгоритмами підвищить практичну цінність розробленого програмного продукту.

## Дослідження і аналіз об'єкту програмування С++

Об'єктом дослідження є два фундаментальні криптографічні алгоритми, кожен з яких представляє різний підхід до забезпечення конфіденційності інформації.

### Шифр «Магічний квадрат»

Шифр «Магічний квадрат» належить до класу перестановочних шифрів та базується на використанні магічного квадрата як ключа для перестановки символів повідомлення. Магічний квадрат являє собою квадратну матрицю розміром n×n, заповнену числами від 1 до n², де сума чисел у кожному рядку, стовпці та діагоналі є однаковою.

Принцип роботи алгоритму полягає у наступному. Спочатку повідомлення записується в таблицю згідно з порядком чисел у магічному квадраті. Потім символи зчитуються в іншому порядку, що й створює зашифроване повідомлення. Ключем шифрування служить сам магічний квадрат та спосіб зчитування символів.

Історично магічні квадрати використовувались у різних культурах для містичних та математичних цілей. У криптографії вони знайшли застосування завдяки своїй здатності створювати складні перестановки, які важко відновити без знання ключа.

Криптографічна стійкість даного методу залежить від розміру магічного квадрата та способу його заповнення. Більші квадрати забезпечують більшу кількість можливих перестановок, що ускладнює криптоаналіз. Однак цей шифр є вразливим до частотного аналізу, оскільки він не змінює частоту появи символів у тексті.

### Квадрат Полібія

Квадрат Полібія є одним з найдавніших відомих методів шифрування та належить до класу моноалфавітних шифрів заміщення. Цей метод був винайдений давньогрецьким істориком Полібієм у II столітті до нашої ери.

Алгоритм використовує квадратну таблицю розміром 5×5 або 6×6, у якій розміщені букви алфавіту. Кожна буква кодується парою чисел, що відповідають номеру рядка та стовпця, в яких вона розташована. Наприклад, у стандартному латинському варіанті буква A може бути закодована як (1,1), якщо вона знаходиться в першому рядку та першому стовпці.

Основною перевагою квадрата Полібія є його простота та можливість передачі закодованих повідомлень різними способами, включаючи візуальні сигнали та звукові коди. Це робило його особливо корисним для військових цілей у давнину.

Криптографічна стійкість квадрата Полібія є відносно низькою з сучасної точки зору, оскільки він залишається моноалфавітним шифром. Це означає, що кожна буква завжди кодується однаково, що робить його вразливим до частотного аналізу. Проте його можна посилити шляхом використання ключових слів для заповнення квадрата або комбінування з іншими криптографічними методами.

### Порівняльний аналіз

Обидва алгоритми представляють класичні підходи до криптографії, але мають принципово різні механізми роботи. Шифр «Магічний квадрат» змінює порядок символів, зберігаючи їх оригінальний вигляд, тоді як квадрат Полібія замінює кожен символ числовою парою.

З точки зору складності реалізації обидва алгоритми є відносно простими, що робить їх відмінними об'єктами для навчальних цілей. Однак їх криптографічна стійкість не відповідає сучасним стандартам безпеки, що обмежує їх практичне застосування в реальних системах захисту інформації.

## **Використані програмні засоби**

Для реалізації та дослідження обраних криптографічних алгоритмів було обрано комплекс сучасних програмних засобів, які забезпечують ефективну розробку та тестування програмного коду.

### Середовище розробки

Основним середовищем розробки обрано Visual Studio Code, який є сучасним, безкоштовним та потужним редактором коду з підтримкою широкого спектру мов програмування. Цей вибір обумовлений наступними факторами: наявність інтелектуального автодоповнення коду, що прискорює процес розробки та зменшує кількість помилок; вбудовані засоби для відлагодження програм, які дозволяють ефективно виявляти та виправляти помилки в коді; підтримка системи контролю версій Git, що забезпечує можливість відстеження змін у коді та співпраці з іншими розробниками; велика кількість розширень для різних мов програмування та фреймворків.

### Мова програмування

Для реалізації алгоритмів обрано мову програмування C++. Цей вибір обґрунтований наступними перевагами C++ у контексті криптографічних застосувань: висока швидкість виконання програм завдяки компілюванню в машинний код, що особливо важливо для криптографічних операцій; ефективне управління пам'яттю, що дозволяє оптимізувати використання ресурсів при роботі з великими обсягами даних; багата стандартна бібліотека STL (Standard Template Library), яка надає контейнери vector, array, map для ефективної роботи з матрицями та таблицями; строга типізація, що зменшує ймовірність помилок під час розробки криптографічних алгоритмів; можливість низькорівневого програмування для оптимізації критичних ділянок коду.

### Додаткові бібліотеки

Для розширення функціональності програми використовуються наступні C++ бібліотеки та компоненти. Стандартна бібліотека STL забезпечує ефективну роботу з контейнерами vector для створення двовимірних масивів та матриць, що особливо важливо для реалізації шифру «Магічний квадрат». Заголовок <iostream> використовується для введення-виведення даних, <string> , <algorithm> для алгоритмів сортування та пошуку.

## Вимоги до апаратного та програмного забезпечення

Для успішної розробки, тестування та експлуатації програмного продукту, що реалізує обрані криптографічні алгоритми, необхідно забезпечити відповідність певним мінімальним вимогам до апаратного та програмного забезпечення.

### Апаратні вимоги

Мінімальні вимоги до апаратного забезпечення є досить помірними, оскільки реалізовані алгоритми не потребують значних обчислювальних ресурсів. Процесор повинен мати тактову частоту не менше 1 ГГц, при цьому підтримуються як Intel, так і AMD архітектури. Рекомендується використання двоядерного процесора для більш комфортної роботи із середовищем розробки.

Оперативна пам'ять повинна становити не менше 4 ГБ для комфортної роботи операційної системи, компілятора C++ та середовища розробки.

Вільне місце на жорсткому диску має становити не менше 2 ГБ для встановлення компілятора C++, середовища розробки, необхідних бібліотек та зберігання файлів проекту. Рекомендується мати додатково 3-5 ГБ вільного місця для тимчасових файлів компіляції та комфортної роботи операційної системи.

Монітор з роздільною здатністю не менше 1024×768 пікселів забезпечить адекватне відображення графічного інтерфейсу програми. Рекомендується роздільна здатність 1920×1080 або вища для більш комфортної роботи з кодом.

Мінімальні вимоги до програмного забезпечення:

Операційна система – Microsoft Windows 7.

# ПРАКТИЧНА ЧАСТИНА

## Створення і налагодження програми

Практична реалізація криптографічних алгоритмів "Магічний квадрат" та квадрат Полібія здійснювалась поетапно з дотриманням принципів структурованого програмування та сучасних стандартів розробки програмного забезпечення.

Процес розробки розпочався з детального аналізу математичних основ кожного з алгоритмів та створення їх формальних описів. Для шифру "Магічний квадрат" було проаналізовано особливості побудови магічних квадратів різних розмірів та визначено оптимальні методи їх генерації. Особливу увагу приділено магічному квадрату розміром 3×3, який є найпростішим для реалізації та демонстрації принципів роботи алгоритму.

Архітектура програмного рішення базується на модульному підході, де кожен алгоритм реалізований як окрема програма з власним консольним інтерфейсом. Такий підхід забезпечує можливість незалежного тестування кожного компонента та спрощує процес налагодження.

Налагодження програм проводилось з використанням засобів вбудованого відлагоджувача Visual Studio Code. Особливу увагу приділено перевірці коректності обробки вхідних даних, включаючи валідацію довжини повідомлень для шифру "Магічний квадрат" та перевірку допустимих символів для квадрата Полібія.

Тестування програм здійснювалось на різноманітних наборах тестових даних, включаючи короткі та довгі повідомлення, тексти з різними типами символів, а також граничні випадки. Результати тестування підтвердили коректність реалізації обох алгоритмів та стабільну роботу програм у різних умовах.

## Опис першої програми та її алгоритмів

Перша програма реалізує алгоритм шифрування за допомогою "Магічного квадрата". Програма створена з використанням мови програмування C++ та надає користувачу консольний інтерфейс для виконання операцій шифрування та дешифрування.

### Математичні основи алгоритму

Магічний квадрат розміром 3×3, що використовується в програмі, має наступний вигляд:

2 7 6

9 5 1

4 3 8

Цей квадрат характеризується тим, що сума чисел у кожному рядку, стовпці та діагоналі дорівнює 15. Числа від 1 до 9 розташовані таким чином, що створюють унікальний ключ для перестановки символів повідомлення.

### Алгоритм шифрування

Процес шифрування складається з наступних етапів:

1. Підготовка повідомлення: Вхідне повідомлення доповнюється символами-заповнювачами (зазвичай пробілами) до довжини, що є кратною 9, оскільки магічний квадрат містить 9 позицій.

2. Розбиття на блоки: Підготовлене повідомлення розбивається на блоки по 9 символів кожен.

3. Розміщення в квадраті: Символи кожного блоку розміщуються в позиції квадрата 3×3 у порядку зростання чисел магічного квадрата. Тобто перший символ блоку розміщується в позицію з числом 1, другий символ - в позицію з числом 2, і так далі.

4. Зчитування: Символи зчитуються з квадрата по рядках зліва направо, зверху вниз, формуючи зашифрований блок.

5. Об'єднання результату: Всі зашифровані блоки об'єднуються в остаточне зашифроване повідомлення.

### Алгоритм дешифрування

Дешифрування виконується у зворотному порядку:

1. Розбиття на блоки: Зашифроване повідомлення розбивається на блоки по 9 символів.

2. Розміщення по рядках: Символи кожного блоку розміщуються в квадрат 3×3 по рядках.

3. Зчитування за магічним квадратом: Символи зчитуються з позицій квадрата у порядку зростання відповідних чисел магічного квадрата.

4. Формування результату: Отримані символи формують розшифрований блок, а всі блоки об'єднуються в остаточне повідомлення.

### Структура програми

Програма має модульну структуру з наступними основними компонентами:

- Функція generateMagicSquare(): Створює та повертає магічний квадрат 3×3.

- Функція encryptBlock(): Шифрує один блок з 9 символів.

- Функція decryptBlock(): Дешифрує один блок з 9 символів.

- Функція magicSquareEncrypt(): Основна функція шифрування, що обробляє весь текст.

- Функція magicSquareDecrypt(): Основна функція дешифрування.

- Функція main(): Забезпечує користувацький інтерфейс та керування програмою.

### Особливості реалізації

Програма включає механізми обробки помилок та валідації вхідних даних. Користувач може вибрати режим роботи (шифрування або дешифрування) через зрозумілий консольний інтерфейс. Програма автоматично обробляє тексти довільної довжини, додаючи необхідні символи-заповнювачі при шифруванні.

## Опис другої програми та її алгоритмів

Друга програма реалізує алгоритм шифрування за допомогою квадрата Полібія. Програма розроблена для роботи з українським алфавітом та забезпечує повний цикл операцій шифрування і дешифрування через консольний інтерфейс.

### Математичні основи алгоритму

Квадрат Полібія для української мови представляє собою таблицю розміром 6×6, що містить 33 літери українського алфавіту та додаткові символи. Кожна літера алфавіту має унікальні координати у вигляді пари чисел (рядок, стовпець).

Структура квадрата Полібія для української мови:

1 2 3 4 5 6

1 А Б В Г Ґ Д

2 Е Є Ж З И І

3 Ї Й К Л М Н

4 О П Р С Т У

5 Ф Х Ц Ч Ш Щ

6 Ь Ю Я . , !

### Алгоритм шифрування

Процес шифрування включає наступні кроки:

1. Підготовка тексту: Вхідний текст перетворюється до верхнього регістру для уніфікації обробки. Символи, що не входять до квадрата Полібія, замінюються найближчими аналогами або ігноруються.

2. Пошук координат: Для кожного символу вхідного тексту визначаються його координати в квадраті Полібія. Кожна літера замінюється парою чисел: номером рядка та номером стовпця.

3. Формування результату: Координати записуються послідовно, формуючи числову послідовність, яка представляє зашифроване повідомлення.

### Алгоритм дешифрування

Дешифрування здійснюється наступним чином:

1. Валідація вхідних даних: Перевіряється, чи складається зашифроване повідомлення з парної кількості цифр, оскільки кожна літера кодується двома цифрами.

2. Розбиття на пари: Числова послідовність розбивається на пари цифр, де кожна пара представляє координати символу в квадраті.

3. Відновлення символів: За кожною парою координат знаходиться відповідний символ у квадраті Полібія.

4. Формування тексту: Відновлені символи об'єднуються в остаточний розшифрований текст.

### Структура програми

Програма організована за модульним принципом:

- Функція initializePolybiusSquare(): Ініціалізує квадрат Полібія з українським алфавітом.

- Функція findCoordinates(): Знаходить координати заданого символу в квадраті.

- Функція findCharacter(): Знаходить символ за заданими координатами.

- Функція polybiusEncrypt(): Основна функція шифрування тексту.

- Функція polybiusDecrypt(): Основна функція дешифрування.

- Функція isValidEncryptedText(): Перевіряє коректність формату зашифрованого тексту.

- Функція main(): Забезпечує користувацький інтерфейс.

### Особливості реалізації

Програма включає розширену систему обробки помилок, що дозволяє коректно обробляти некоректні вхідні дані. Реалізовано підтримку повного українського алфавіту, включаючи специфічні літери. Користувацький інтерфейс забезпечує зручну навігацію між різними режимами роботи.

Програма оптимізована для роботи з текстами різної довжини та забезпечує високу швидкість обробки завдяки ефективним алгоритмам пошуку в двовимірних масивах.

### Криптографічний аналіз

Реалізований варіант квадрата Полібія зберігає класичні властивості цього шифру: простоту використання та відносно низьку криптографічну стійкість. Шифр залишається вразливим до частотного аналізу, оскільки кожна літера завжди кодується однаковою парою чисел.

Для підвищення стійкості в програмі передбачено можливість модифікації квадрата шляхом перестановки літер, що може служити додатковим ключем шифрування.

## Інструкція програміста і оператора до програми

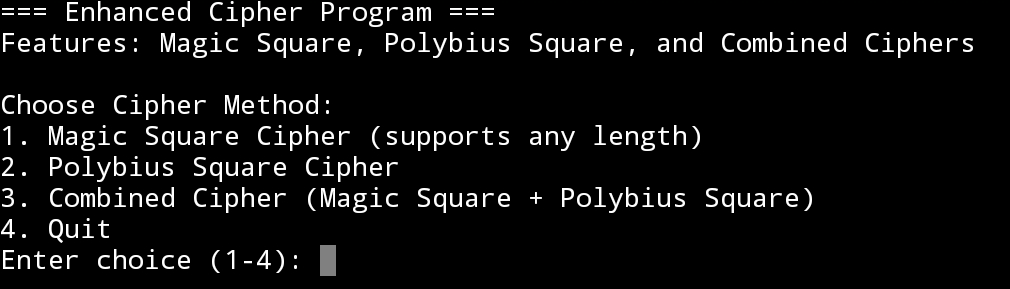
Після того, як ми запустили програму на екрані з'явиться головне меню в якому буду можливих 4 дії:

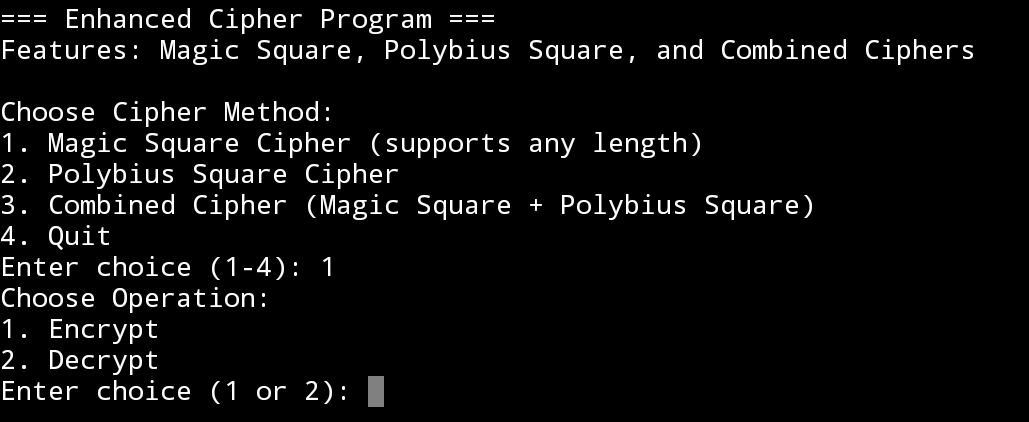
1 – Шифрування «Магічний квадрат»

2 – Шифрування «Квадрат Полібія»

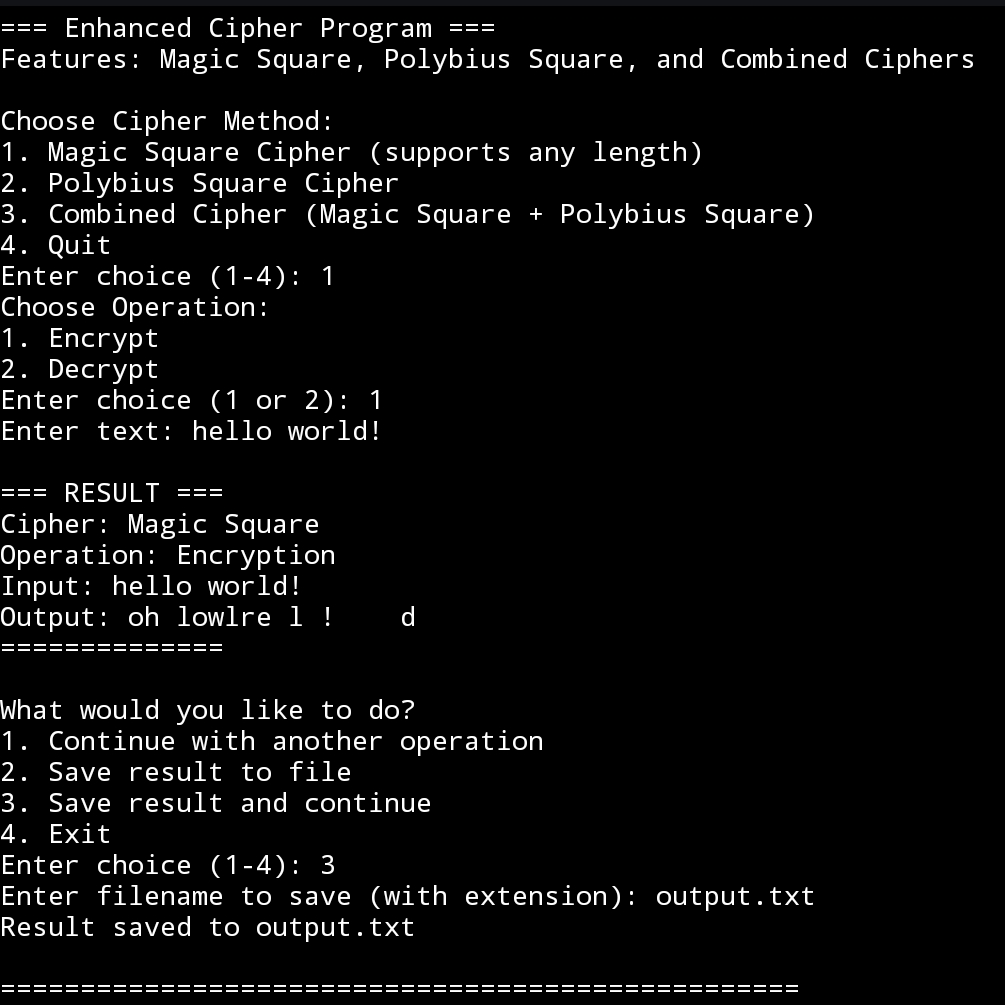
3 – Комбіноване шифрування

4 – Вийти

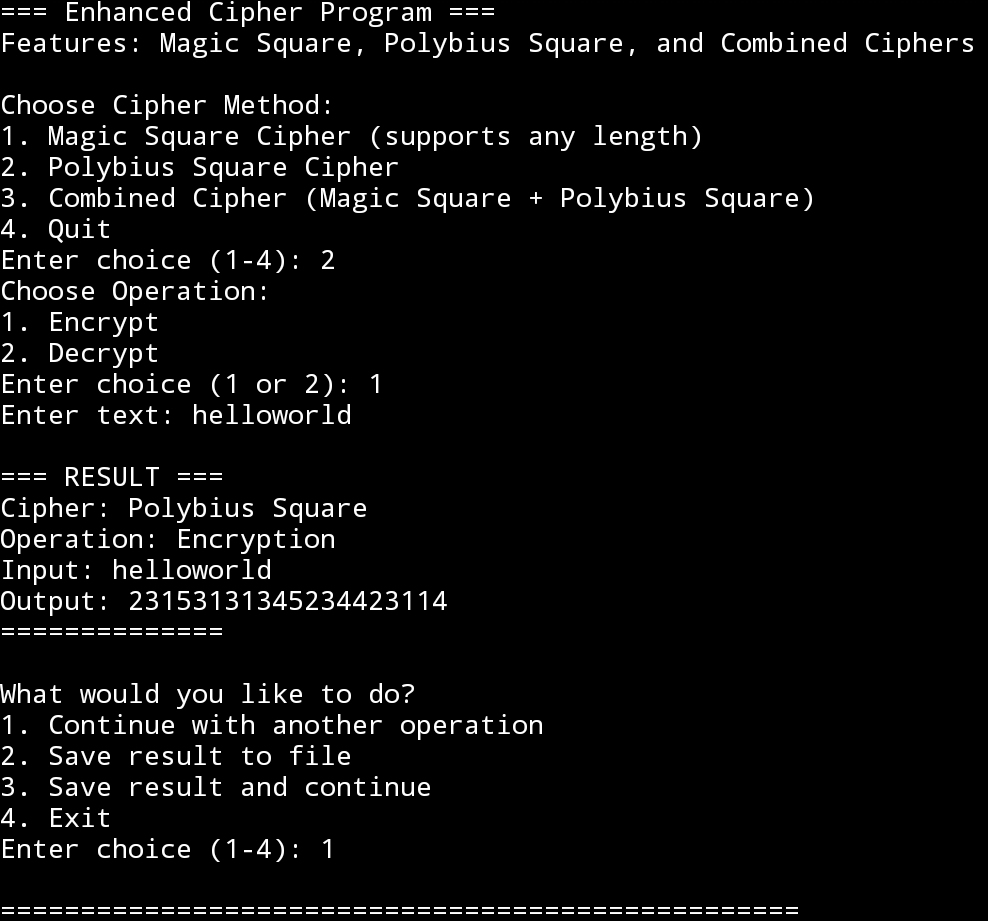


Після вибору метода шифрування користувач обирає операцію (Зашифрувати або розшифрувати) 

Шифрування «Магічний квадрат»



Шифрування «Квадрат Полібія»



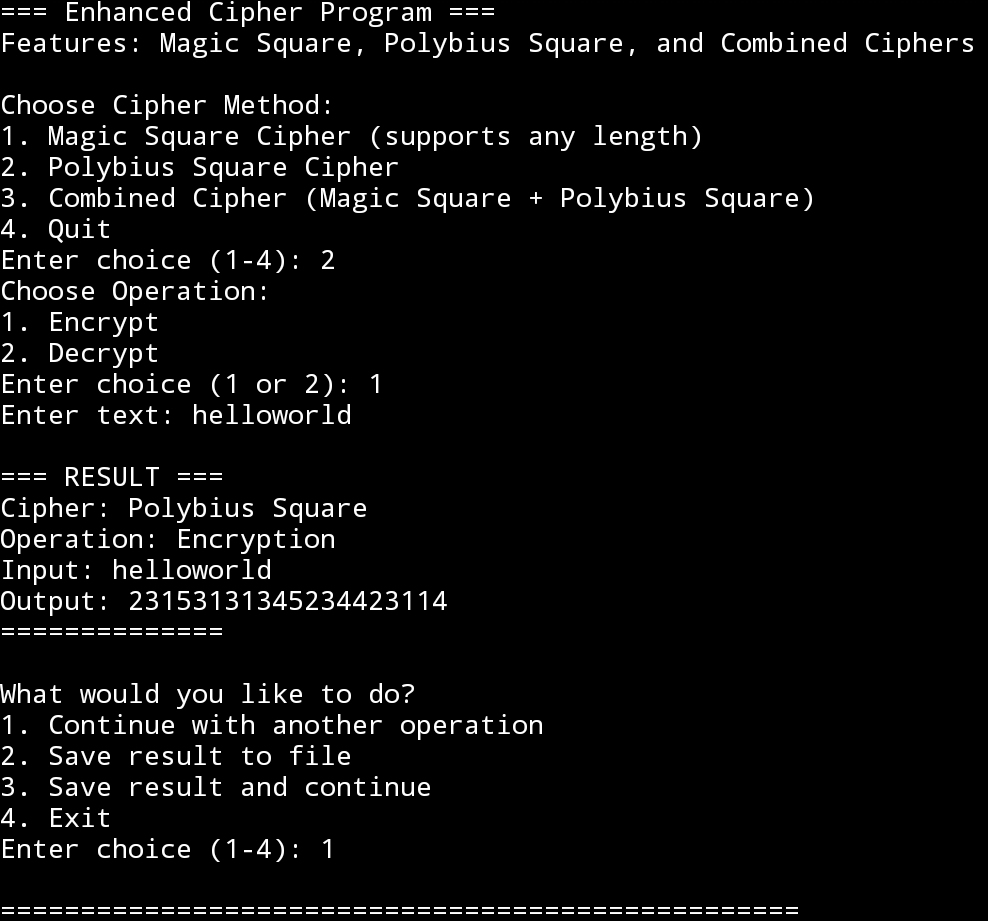
Після завершення шифрування/розшифрування користувач може обрати, що він хоче зробити з результатом операції:

1 – Продовжити з іншою операцією

2 – Зберегти результат до файлу

3 – Зберегти результат до файлу та продовжити з іншою операцією

4 – Вийти



# ВИСНОВОК

Відповідно до завдання були розроблені програми, що мають зручний інтерфейс, і надають можливість шифрування та дешифрування слів методами «Магічний квадрат» і квадратом Полібія.

Під час написання курсового проєкту було закріплено знання здобуті з предмету «Основи програмування та алгоритмічні мови».

# ДОДАТОК А

#include <iostream>

#include <string>

#include <map>

#include <cctype>

#include <algorithm>

#include <fstream>

#include <vector>

using namespace std;

// ================= MAGIC SQUARE =================

int magicSquare[3][3] = {

{8, 1, 6},

{3, 5, 7},

{4, 9, 2}

};

string encryptMagicSquare(const string &msg) {

string result = "";

string text = msg;

// Process text in chunks of 9 characters

for (size\_t i = 0; i < text.length(); i += 9) {

string chunk = text.substr(i, 9);

// Pad chunk to 9 characters if needed

while (chunk.size() < 9) chunk += ' ';

char square[3][3];

for (int j = 1; j <= 9; ++j) {

for (int r = 0; r < 3; ++r) {

for (int c = 0; c < 3; ++c) {

if (magicSquare[r][c] == j) {

square[r][c] = chunk[j - 1];

}

}

}

}

// Read encrypted chunk row by row

for (int r = 0; r < 3; ++r) {

for (int c = 0; c < 3; ++c) {

result += square[r][c];

}

}

}

return result;

}

string decryptMagicSquare(const string &cipher) {

string result = "";

// Process cipher in chunks of 9 characters

for (size\_t i = 0; i < cipher.length(); i += 9) {

string chunk = cipher.substr(i, 9);

if (chunk.size() < 9) {

result += "Incomplete chunk in Magic Square decryption.";

break;

}

char square[3][3];

int idx = 0;

for (int r = 0; r < 3; ++r) {

for (int c = 0; c < 3; ++c) {

square[r][c] = chunk[idx++];

}

}

char decrypted[9];

for (int j = 1; j <= 9; ++j) {

for (int r = 0; r < 3; ++r) {

for (int c = 0; c < 3; ++c) {

if (magicSquare[r][c] == j) {

decrypted[j - 1] = square[r][c];

}

}

}

}

result += string(decrypted, 9);

}

return result;

}

// ================= POLYBIUS SQUARE =================

char polybiusSquare[5][5] = {

{'A','B','C','D','E'},

{'F','G','H','I','K'},

{'L','M','N','O','P'},

{'Q','R','S','T','U'},

{'V','W','X','Y','Z'}

};

map<char, string> createPolybiusMap() {

map<char, string> polyMap;

for (int r = 0; r < 5; ++r)

for (int c = 0; c < 5; ++c)

polyMap[polybiusSquare[r][c]] = to\_string(r + 1) + to\_string(c + 1);

polyMap['J'] = polyMap['I']; // Treat J as I

return polyMap;

}

string encryptPolybius(const string &text) {

map<char, string> polyMap = createPolybiusMap();

string cipher = "";

for (char ch : text) {

ch = toupper(ch);

if (ch >= 'A' && ch <= 'Z') {

cipher += polyMap[ch];

}

}

return cipher;

}

string decryptPolybius(const string &cipher) {

string text = "";

for (size\_t i = 0; i + 1 < cipher.size(); i += 2) {

int row = cipher[i] - '1';

int col = cipher[i + 1] - '1';

if (row >= 0 && row < 5 && col >= 0 && col < 5)

text += polybiusSquare[row][col];

}

return text;

}

// ================= COMBINED CIPHER =================

string encryptCombined(const string &text) {

string magicResult = encryptMagicSquare(text);

return encryptPolybius(magicResult);

}

string decryptCombined(const string &cipher) {

string polybiusResult = decryptPolybius(cipher);

return decryptMagicSquare(polybiusResult);

}

// ================= UTILITY FUNCTIONS =================

string cleanInput(const string &input) {

string cleaned = input;

// Remove control characters like ^C, ^V, ^@, etc.

cleaned.erase(remove\_if(cleaned.begin(), cleaned.end(), [](char ch) {

return ch < 32 && ch != '\t' && ch != '\n' && ch != '\r';

}), cleaned.end());

return cleaned;

}

void saveToFile(const string &result, const string &operation) {

string filename;

cout << "Enter filename to save (with extension): ";

getline(cin, filename);

filename = cleanInput(filename);

if (filename.empty()) {

cout << "Invalid filename. Result not saved.\n";

return;

}

ofstream file(filename);

if (file.is\_open()) {

file << operation << " Result:\n";

file << result << endl;

file.close();

cout << "Result saved to " << filename << endl;

} else {

cout << "Error: Could not save to file.\n";

}

}

bool askContinue() {

char choice;

cout << "\nWhat would you like to do?\n";

cout << "1. Continue with another operation\n";

cout << "2. Save result to file\n";

cout << "3. Save result and continue\n";

cout << "4. Exit\n";

cout << "Enter choice (1-4): ";

cin >> choice;

cin.ignore(); // flush newline

return choice == '1' || choice == '3';

}

// ================= MAIN INTERFACE =================

int main() {

cout << "=== Enhanced Cipher Program ===\n";

cout << "Features: Magic Square, Polybius Square, and Combined Ciphers\n\n";

while (true) {

cout << "Choose Cipher Method:\n";

cout << "1. Magic Square Cipher (supports any length)\n";

cout << "2. Polybius Square Cipher\n";

cout << "3. Combined Cipher (Magic Square + Polybius Square)\n";

cout << "4. Quit\n";

cout << "Enter choice (1-4): ";

int cipherChoice;

cin >> cipherChoice;

if (cipherChoice == 4) {

cout << "Goodbye!\n";

break;

}

if (cipherChoice < 1 || cipherChoice > 4) {

cout << "Invalid cipher selection. Please try again.\n\n";

continue;

}

cout << "Choose Operation:\n";

cout << "1. Encrypt\n";

cout << "2. Decrypt\n";

cout << "Enter choice (1 or 2): ";

int operation;

cin >> operation;

if (operation != 1 && operation != 2) {

cout << "Invalid operation. Please try again.\n\n";

continue;

}

cin.ignore(); // flush newline

string inputText;

cout << "Enter text: ";

getline(cin, inputText);

// Clean input to remove control characters

inputText = cleanInput(inputText);

if (inputText.empty()) {

cout << "No valid input provided. Please try again.\n\n";

continue;

}

string result;

string cipherName;

switch (cipherChoice) {

case 1: // Magic Square

cipherName = "Magic Square";

if (operation == 1) {

result = encryptMagicSquare(inputText);

} else {

result = decryptMagicSquare(inputText);

}

break;

case 2: // Polybius Square

cipherName = "Polybius Square";

if (operation == 1) {

result = encryptPolybius(inputText);

} else {

result = decryptPolybius(inputText);

}

break;

case 3: // Combined

cipherName = "Combined (Magic + Polybius)";

if (operation == 1) {

result = encryptCombined(inputText);

} else {

result = decryptCombined(inputText);

}

break;

}

cout << "\n=== RESULT ===\n";

cout << "Cipher: " << cipherName << endl;

cout << "Operation: " << (operation == 1 ? "Encryption" : "Decryption") << endl;

cout << "Input: " << inputText << endl;

cout << "Output: " << result << endl;

cout << "==============\n";

// Ask what to do next

char nextChoice;

cout << "\nWhat would you like to do?\n";

cout << "1. Continue with another operation\n";

cout << "2. Save result to file\n";

cout << "3. Save result and continue\n";

cout << "4. Exit\n";

cout << "Enter choice (1-4): ";

cin >> nextChoice;

cin.ignore(); // flush newline

bool shouldContinue = false;

switch (nextChoice) {

case '1':

shouldContinue = true;

break;

case '2':

saveToFile(result, cipherName + " " + (operation == 1 ? "Encryption" : "Decryption"));

shouldContinue = false;

break;

case '3':

saveToFile(result, cipherName + " " + (operation == 1 ? "Encryption" : "Decryption"));

shouldContinue = true;

break;

case '4':

shouldContinue = false;

break;

default:

cout << "Invalid choice. Continuing...\n";

shouldContinue = true;

break;

}

if (!shouldContinue) {

cout << "Goodbye!\n";

break;

}

cout << "\n" << string(50, '=') << "\n\n";

}

return 0;

}