МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ОБРАЗОВАНИЯ

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кафедра защиты информации



**ЛАБОРАТОРНАЯ РАБОТА №6**

**«Обработка двумерных массивов и контейнеров»**

**по дисциплине: «*Программирование*»**

|  |  |
| --- | --- |
| Выполнил:  Студент гр. «АБс-322», «АВТФ»  *Бурлаков И.Е.*  «7» июня 2024 г.  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись) | Проверил:  *Ассистент кафедры ЗИ*  *Исаев Г. А.*  «7» июня 2024 г.  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись) |

Новосибирск 2024

**Цели и задачи работы:** изучение алгоритмов формирования и обработки двумерных массивов, программирование и отладка программ формирования и обработки массивов (статических и динамических) и контейнеров STL.

**Задание к работе**: Самостоятельно решить задачи в соответствии с индивидуальным вариантом.

**Методика выполнения работы**:

1. Разработать алгоритм решения задачи по индивидуальному заданию.

2. Написать и отладить программу решения задачи на двух языках (С++ и второй, по выбору).

3. Протестировать работу программы на различных исходных данных.

**Задание №6.1 Обработка двумерных массивов.**

**Вариант 6**

1.Определите и инициализируйте матрицу размерности M × N случайными вещественными числами в диапазоне [0, 100]. Найдите и выведите средние арифметические значения элементов матрицы, а также каждой строки. Определите номер строки, у которой среднее арифметическое значение наибольшее.

2.Определите и инициализируйте квадратную матрицу порядка M (M > 5) случайными целыми числами в диапазоне [-100, 100]. Отсортируйте чётные столбцы в порядке возрастания, а нечётные – в порядке убывания. Определите, в какой половине матрицы относительно главной диагонали, включая саму диагональ, больше положительных элементов.

3.Реализуйте клеточный автомат Джона Конвея на ограниченной плоскости по классическим правилам. Продемонстрируйте работу клеточного автомата на нескольких примерах двигающихся фигур (фигур, у которых состояние повторяется, но со смещением) и на развитии колоний клеток, сгенерированных в случайном порядке. Реализовать работу графического отображения клеточного автомата можно таким образом, чтобы живые клетки изображались единицами, а мёртвые – нулями, либо живые – нулями, а мёртвые пробелами, либо другим возможным вариантом. Каждое новое поколение выводится на очищенное окно консоли через некоторый промежуток времени. Таким образом, получается непрерывная анимация.

**Язык программирования : С++**

**Задание 6.1.1**

**Код программы :**

```  
#include <iostream>

#include <random>

#include <string>

#include <iomanip>

using namespace std;

// Функция для создания и инициализации матрицы

double\*\* createMatrix(int m, int n) {

random\_device rd; // Создание объекта random\_device для генерации случайных чисел

mt19937 gen(rd()); // Инициализация генератора случайных чисел mt19937 с использованием random\_device

uniform\_real\_distribution<> dis(0.0, 100.0); // Создание равномерного распределения в диапазоне [0.0, 100.0)

double\*\* matrix = new double\* [m]; // Выделение памяти под массив указателей на массивы (матрицу)

for (int i = 0; i < m; ++i) { // Цикл по строкам матрицы

matrix[i] = new double[n]; // Выделение памяти под массив значений для текущей строки

for (int j = 0; j < n; ++j) { // Цикл по столбцам матрицы

matrix[i][j] = dis(gen); // Заполнение текущего элемента матрицы случайным числом из указанного диапазона

}

}

return matrix; // Возврат указателя на созданную матрицу

}

// Функция для вывода матрицы на экран

void printMatrix(double\*\* matrix, int m, int n) {

cout << "Матрица:" << endl; // Вывод заголовка для матрицы

for (int i = 0; i < m; ++i) { // Цикл по строкам матрицы

for (int j = 0; j < n; ++j) { // Цикл по столбцам матрицы

cout << fixed << setprecision(2) << matrix[i][j] << "\t"; // Вывод текущего элемента матрицы с точностью 2 знака после запятой и фиксированным форматом

}

cout << endl; // Переход на новую строку после вывода всех элементов текущей строки матрицы

}

}

// Функция для вычисления среднего арифметического значения всех элементов матрицы

double TotalAvg(double\*\* matrix, int m, int n) {

double totalSum = 0; // Общая сумма всех элементов матрицы

for (int i = 0; i < m; ++i) { // Цикл по строкам матрицы

for (int j = 0; j < n; ++j) { // Цикл по столбцам матрицы

totalSum += matrix[i][j]; // Суммирование текущего элемента матрицы

}

}

// Возвращаем среднее арифметическое, разделив общую сумму на количество элементов

return totalSum / (static\_cast<double>(m) \* n);

}

// Функция для вычисления среднего арифметического значения каждой строки и поиска строки с максимальным значением

void RowAverages(double\*\* matrix, int m, int n, double\* StrAverages, int& maxStroka) {

double maxRowAvg = 0; // Максимальное среднее значение строки

for (int i = 0; i < m; ++i) { // Цикл по строкам матрицы

double rowSum = 0; // Сумма элементов в текущей строке

for (int j = 0; j < n; ++j) { // Цикл по столбцам матрицы

rowSum += matrix[i][j]; // Суммирование элементов текущей строки

}

// Вычисляем среднее значение текущей строки

StrAverages[i] = rowSum / static\_cast<double>(n);

// Если среднее значение текущей строки больше максимального,

// обновляем максимальное значение и индекс строки

if (StrAverages[i] > maxRowAvg) {

maxRowAvg = StrAverages[i];

maxStroka = i;

}

}

}

// Функция для получения целочисленного значения от пользователя с обработкой исключений

int User(const string& prompt) {

while (true) {

try {

cout << prompt; // Выводим приглашение к вводу

string input; // Создаем строку для хранения ввода пользователя

getline(cin, input); // Считываем строку из стандартного ввода

size\_t pos; // Переменная для хранения позиции, на которой закончилось преобразование

int value = stoi(input, &pos); // Преобразуем строку в целое число

if (pos != input.size()) { // Проверяем, были ли преобразованы все символы строки

throw invalid\_argument("Неверный формат числа. Пожалуйста, введите целое число(без пробелов)."); // Генерируем исключение, если не все символы были преобразованы

}

return value; // Возвращаем преобразованное целочисленное значение

}

catch (const invalid\_argument& e) { // Обрабатываем исключение, если введены недопустимые символы

cerr << "Ошибка: " << e.what() << endl; // Выводим сообщение об ошибке в стандартный поток ошибок

}

}

}

void DisplayTask1() {

cout << "---Задание номер 1---" << endl;

int m = User("Введите количество строк: ");

int n = User("Введите количество столбцов: ");

double\*\* matrix = createMatrix(m, n);

printMatrix(matrix, m, n);

double totalAvg = TotalAvg(matrix, m, n);

cout << "\nСреднее арифметическое значение всех элементов матрицы: " << totalAvg << endl;

double\* StrAverages = new double[m];

int maxstroka = 0;

RowAverages(matrix, m, n, StrAverages, maxstroka);

cout << "\nСредние арифметические значения каждой строки:" << endl;

for (int i = 0; i < m; ++i) {

cout << "Строка " << i + 1 << ": " << StrAverages[i] << endl;

}

cout << "\nНомер строки с наибольшим средним арифметическим значением: " << maxstroka + 1 << endl;

}

// Основная функция

int main() {

setlocale(LC\_ALL, "ru");

DisplayTask1();

return 0;

}   
```

**Язык программирования : Python**

**Задание 6.1.1**

**Код программы :**

```

import random

# Функция для создания и инициализации матрицы

def create\_matrix(m, n):

matrix = [[random.uniform(0.0, 100.0) for \_ in range(n)] for \_ in range(m)]

return matrix

# Функция для вывода матрицы на экран

def print\_matrix(matrix):

print("Матрица:")

for row in matrix:

print("\t".join(f"{elem:.2f}" for elem in row))

# Функция для вычисления среднего арифметического значения всех элементов матрицы

def total\_avg(matrix):

total\_sum = sum(sum(row) for row in matrix)

return total\_sum / (len(matrix) \* len(matrix[0]))

# Функция для вычисления среднего арифметического значения каждой строки и поиска строки с максимальным значением

def row\_averages(matrix):

row\_averages = [sum(row) / len(row) for row in matrix]

max\_avg\_row = max(row\_averages)

max\_row\_index = row\_averages.index(max\_avg\_row) + 1

return row\_averages, max\_row\_index

# Основная функция

def main():

print("---Задание номер 1---")

m = int(input("Введите количество строк: "))

n = int(input("Введите количество столбцов: "))

matrix = create\_matrix(m, n)

print\_matrix(matrix)

total\_avg\_value = total\_avg(matrix)

print(f"\nСреднее арифметическое значение всех элементов матрицы: {total\_avg\_value:.2f}")

row\_avg\_values, max\_row\_index = row\_averages(matrix)

print("\nСредние арифметические значения каждой строки:")

for i, avg in enumerate(row\_avg\_values, 1):

print(f"Строка {i}: {avg:.2f}")

print(f"\nНомер строки с наибольшим средним арифметическим значением: {max\_row\_index}")

if \_\_name\_\_ == "\_\_main\_\_":

main()  
```

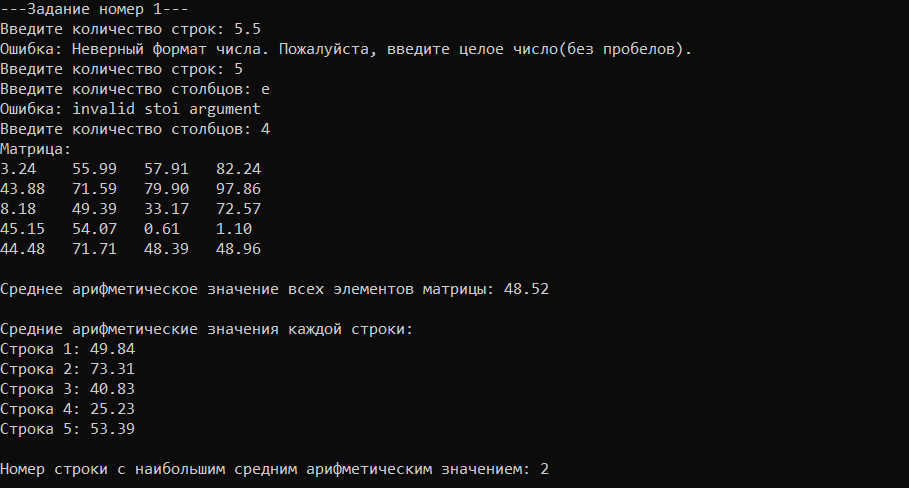


Рисунок 1 – Пример работы программы

**Язык программирования : С++**

**Задание 6.1.2**

**Код программы :**

**```  
#include** <iostream>

#include <algorithm>

#include <random>

#include <string>

using namespace std;

//создание матрицы

void createMatrix(int\*\*& matrix, int m) {

random\_device rd; // Создание объекта random\_device для генерации случайных чисел

mt19937 gen(rd()); // Инициализация генератора случайных чисел mt19937 с использованием random\_device

uniform\_int\_distribution<> dis(-100, 100); // Создание равномерного распределения в диапазоне

matrix = new int\* [m]; // Выделение памяти для массива указателей на строки

for (int i = 0; i < m; ++i) {

matrix[i] = new int[m]; // Выделение памяти для каждой строки

for (int j = 0; j < m; ++j) {

matrix[i][j] = dis(gen); // Инициализация случайным числом

}

}

}

// Функция для вывода матрицы на экран

void printMatrix(int\*\* matrix, int m) {

for (int i = 0; i < m; ++i) {

for (int j = 0; j < m; ++j) {

cout << matrix[i][j] << "\t";

}

cout << endl;

}

}

// Функция для сортировки столбцов матрицы

void sortColumns(int\*\* matrix, int m) {

// Итерация по каждому столбцу матрицы

for (int j = 0; j < m; ++j) {

// Создаем временный вектор для хранения текущего столбца

vector<int> column;

// Получаем j-ый столбец и добавляем его элементы в вектор

for (int i = 0; i < m; ++i) {

column.push\_back(matrix[i][j]);

}

// Если номер столбца четный

if (j % 2 == 0) {

// Сортировка четных столбцов по возрастанию

sort(column.begin(), column.end());

}else {

// Сортировка нечетных столбцов по убыванию

sort(column.begin(), column.end(), greater<int>());

}

// Заменяем элементы столбца в матрице отсортированными элементами из вектора

for (int i = 0; i < m; ++i) {

matrix[i][j] = column[i];

}

}

}

void countPositiveElements(int\*\* matrix, int m, int& upperHalfPositives, int& lowerHalfPositives) {

for (int i = 0; i < m; ++i) {

for (int j = 0; j < m; ++j) {

if (i < j) { // Верхняя половина

if (matrix[i][j] > 0) {

++upperHalfPositives;

}

}

else if (i == j) { // Главная диагональ

if (matrix[i][j] > 0) {

++upperHalfPositives;

++lowerHalfPositives;

}

}else { // Нижняя половина

if (matrix[i][j] > 0) {

++lowerHalfPositives;

}

}

}

}

if (lowerHalfPositives > upperHalfPositives) {

cout << "В нижней половине матрицы больше положительных элементов. " << lowerHalfPositives << ">" << upperHalfPositives << endl;

}

else if (lowerHalfPositives < upperHalfPositives) {

cout << "В верхней половине матрицы больше положительных элементов. " << lowerHalfPositives << "<" << upperHalfPositives << endl;

}

else {

cout << "Количество элементов в верхней и нижней половине одинаково. " << lowerHalfPositives << "=" << upperHalfPositives << endl;

}

}

int User(const string& prompt) {

while (true) {

try {

cout << prompt; // Выводим приглашение к вводу

string input; // Создаем строку для хранения ввода пользователя

getline(cin, input); // Считываем строку из стандартного ввода

size\_t pos; // Переменная для хранения позиции, на которой закончилось преобразование

int value = stoi(input, &pos); // Преобразуем строку в целое число

if (pos != input.size() || value <= 5) { // Проверяем, были ли преобразованы все символы строки

throw invalid\_argument("Неверный формат числа. Пожалуйста, введите целое число больше 5(без пробелов)."); // Генерируем исключение, если не все символы были преобразованы

}

return value; // Возвращаем преобразованное целочисленное значение

}

catch (const invalid\_argument& e) { // Обрабатываем исключение, если введены недопустимые символы

cerr << "Ошибка: " << e.what() << endl; // Выводим сообщение об ошибке в стандартный поток ошибок

}

}

}

int main() {

setlocale(LC\_ALL, "ru");

int M = User("Введите порядок квадратной матрицы: ");

int\*\* matrix;

createMatrix(matrix, M);

cout << "Исходная матрица:" << endl;

printMatrix(matrix, M);

// Сортировка столбцов

sortColumns(matrix, M);

cout << "Матрица после сортировки:" << endl;

printMatrix(matrix, M);

// Подсчет положительных элементов в каждой половине матрицы

int upperHalfPositives = 0, lowerHalfPositives = 0;

countPositiveElements(matrix, M, upperHalfPositives, lowerHalfPositives);

return 0;

}

**```**

**Язык программирования : Python**

**Задание 6.1.2**

**Код программы :**

**```**import random

# Функция для генерации случайного числа в заданном диапазоне

def get\_random\_number(min\_val, max\_val):

return random.randint(min\_val, max\_val)

# Функция для создания и инициализации матрицы

def create\_matrix(m):

matrix = []

for \_ in range(m):

row = [get\_random\_number(-100, 100) for \_ in range(m)]

matrix.append(row)

return matrix

# Функция для вывода матрицы на экран

def print\_matrix(matrix):

for row in matrix:

print("\t".join(map(str, row)))

# Функция для сортировки столбцов матрицы

def sort\_columns(matrix):

for j in range(len(matrix[0])):

column = [matrix[i][j] for i in range(len(matrix))]

if j % 2 == 0:

column.sort()

else:

column.sort(reverse=True)

for i in range(len(matrix)):

matrix[i][j] = column[i]

# Функция для подсчета положительных элементов в верхней и нижней половинах матрицы

def count\_positive\_elements(matrix):

m = len(matrix)

upper\_half\_positives = 0

lower\_half\_positives = 0

for i in range(m):

for j in range(m):

if i < j: # Верхняя половина

if matrix[i][j] > 0:

upper\_half\_positives += 1

elif i == j: # Главная диагональ

if matrix[i][j] > 0:

upper\_half\_positives += 1

lower\_half\_positives += 1

else: # Нижняя половина

if matrix[i][j] > 0:

lower\_half\_positives += 1

if lower\_half\_positives > upper\_half\_positives:

print(f"В нижней половине матрицы больше положительных элементов. {lower\_half\_positives} > {upper\_half\_positives}")

elif lower\_half\_positives < upper\_half\_positives:

print(f"В верхней половине матрицы больше положительных элементов. {lower\_half\_positives} < {upper\_half\_positives}")

else:

print(f"Количество элементов в верхней и нижней половине одинаково. {lower\_half\_positives} = {upper\_half\_positives}")

def user\_input(prompt):

while True:

try:

value = int(input(prompt))

if value <= 5:

raise ValueError("Порядок матрицы должен быть больше 5.")

return value

except ValueError as e:

print("Ошибка:", e)

def main():

m = user\_input("Введите порядок квадратной матрицы: ")

matrix = create\_matrix(m)

print("Исходная матрица:")

print\_matrix(matrix)

# Сортировка столбцов

sort\_columns(matrix)

print("Матрица после сортировки:")

print\_matrix(matrix)

# Подсчет положительных элементов в каждой половине матрицы

count\_positive\_elements(matrix)

if \_\_name\_\_ == "\_\_main\_\_":

main()  
```

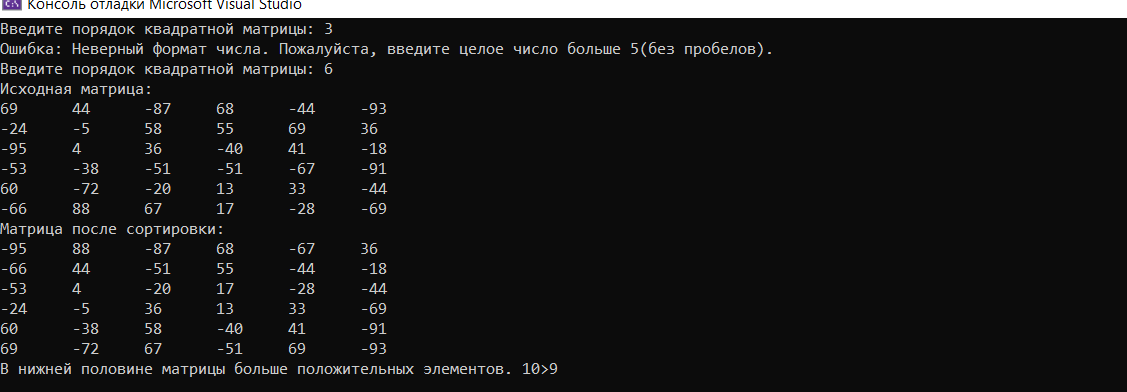


Рисунок 2 – Результат работы программы

**Язык программирования : C++**

**Задание 6.1.3**

**Код программы :  
```  
#include** <iostream>

#include <chrono>

#include <thread>

#include <random>

using namespace std;

// Размеры поля с учётом границ

const int width = 10;

const int height = 10;

// Функция для очистки экрана

static void clear\_screen() {

cout << "\033[2J\033[1;1H"; // ANSI escape код для очистки экрана

}

// Функция для отображения текущего состояния поля

void display\_field(int\*\* field) {

clear\_screen(); // Очищаем экран перед выводом нового состояния поля

// Перебираем все клетки поля и выводим их состояние

for (int i = 0; i < height; ++i) {

for (int j = 0; j < width; ++j) {

if (i == 0 || i == height - 1 || j == 0 || j == width - 1)

cout << "\*"; // Границы поля

else if (field[i][j] == 1)

cout << "#"; // Если клетка живая, выводим символ #

else

cout << " "; // Если клетка мертвая, выводим пробел

}

cout << endl; // Переходим на новую строку после вывода строки клеток

}

}

// Функция для вычисления следующего поколения

void next\_generation(int\*\*& field) {

int\*\* next\_field = new int\* [height]; // Создаём новое поле для следующего поколения

for (int i = 0; i < height; ++i) {

next\_field[i] = new int[width]; // Выделяем память под каждую строку нового поля

}

// Перебираем каждую клетку текущего поколения и вычисляем её состояние в следующем поколении

for (int i = 1; i < height - 1; ++i) { // не учитываем границы

for (int j = 1; j < width - 1; ++j) {

int neighbors = 0;

// Подсчитываем количество соседей для каждой клетки

for (int ni = -1; ni <= 1; ++ni) {

for (int nj = -1; nj <= 1; ++nj) {

if (ni == 0 && nj == 0) //проверка на текущую клетку

continue; // Пропускаем текущую клетку

neighbors += field[i + ni][j + nj]; // Увеличиваем счётчик соседей для живых клеток

}

}

// Применяем правила клеточного автомата "Жизнь"

if (field[i][j] == 1) {

if (neighbors < 2 || neighbors > 3) // если у живой клетки соседей меньше 2 или больше 3 она станоится мертвой

next\_field[i][j] = 0; // Смерть из-за перенаселённости или одиночества

else

next\_field[i][j] = 1; // Выживание

}else { // если у мертвой клетки соседей 3,то она рождается,в другом случает остается мертвой

if (neighbors == 3)

next\_field[i][j] = 1; // Рождение новой клетки

else

next\_field[i][j] = 0; // Клетка остаётся мёртвой

}

}

}

// Удаление старого поля

for (int i = 0; i < height; ++i) {

delete[] field[i]; // оочищаем память для каждой строки

}

delete[] field; // очищаем память массива указателей на строки

// Передача указателя на новое поле

field = next\_field;

}

// Функция для инициализации поля случайными значениями

int\*\* initialize\_field() {

int\*\* field = new int\* [height]; // Создание двумерного массива указателей для строк поля

for (int i = 0; i < height; ++i) {

field[i] = new int[width]; // Выделение памяти под каждую строку поля

}

// Заполнение поля нулями (все клетки мертвы)

for (int i = 0; i < height; ++i) {

for (int j = 0; j < width; ++j) {

field[i][j] = 0; // Установка значения 0 (мертвая клетка) для каждой клетки поля

}

}

return field; // Возвращаем указатель на инициализированное поле

}

// Функция для инициализации поля с шаблонами

void initialize\_field\_with\_patterns(int\*\* field) {

// Очистка поля перед установкой планера

for (int i = 0; i < height; ++i) {

for (int j = 0; j < width; ++j) {

field[i][j] = 0; // Установка всех клеток в начальное состояние (мертвые)

}

}

// Установка планера на поле

field[2][3] = field[3][4] = field[4][2] = field[4][3] = field[4][4] = 1; // Планер

}

int main() {

// Инициализация поля

int\*\* field = initialize\_field();

// Инициализация с определенными шаблонами

initialize\_field\_with\_patterns(field);

int generation\_count = 0; // Счётчик поколений

// Анимация

for (int generation = 0; generation < 100; ++generation) {

display\_field(field);

if (generation\_count == 13) {

initialize\_field\_with\_patterns(field); // Обновляем планер

generation\_count = 0; // Сбрасываем счётчик поколений

}

else {

next\_generation(field); // Вычисляем следующее поколение

generation\_count++; // Увеличиваем счётчик поколений

}

this\_thread::sleep\_for(chrono::milliseconds(100)); // Задержка в миллисекундах

}

// Освобождение памяти

for (int i = 0; i < height; ++i) {

delete[] field[i];

}

delete[] field;

return 0;

} **```**

**Язык программирования : Python**

**Задание 6.1.3**

**Код программы :**

```

import os

import time

# Размеры поля с учётом границ

width = 10

height = 10

# Функция для очистки экрана

def clear\_screen():

os.system('cls' if os.name == 'nt' else 'clear')

# Функция для отображения текущего состояния поля

def display\_field(field):

clear\_screen()

for i in range(height):

for j in range(width):

if i == 0 or i == height - 1 or j == 0 or j == width - 1:

print("\*", end='') # Границы поля

elif field[i][j] == 1:

print("#", end='') # Живая клетка

else:

print(" ", end='') # Мертвая клетка

print()

# Функция для вычисления следующего поколения

def next\_generation(field):

next\_field = [[0] \* width for \_ in range(height)]

for i in range(1, height - 1):

for j in range(1, width - 1):

neighbors = sum(field[i + ni][j + nj] for ni in range(-1, 2) for nj in range(-1, 2) if not (ni == 0 and nj == 0))

if field[i][j] == 1:

if neighbors < 2 or neighbors > 3:

next\_field[i][j] = 0 # Смерть из-за перенаселённости или одиночества

else:

next\_field[i][j] = 1 # Выживание

else:

if neighbors == 3:

next\_field[i][j] = 1 # Рождение новой клетки

else:

next\_field[i][j] = 0 # Клетка остаётся мёртвой

return next\_field

# Функция для инициализации поля

def initialize\_field():

field = [[0] \* width for \_ in range(height)]

return field

# Функция для инициализации поля с шаблонами

def initialize\_field\_with\_patterns(field):

# Очистка поля перед установкой планера

for i in range(height):

for j in range(width):

field[i][j] = 0

# Установка планера на поле

field[2][3] = field[3][4] = field[4][2] = field[4][3] = field[4][4] = 1 # Планер

def main():

# Инициализация поля

field = initialize\_field()

# Инициализация с определенными шаблонами

initialize\_field\_with\_patterns(field)

generation\_count = 0 # Счётчик поколений

# Анимация

for generation in range(100):

display\_field(field)

if generation\_count == 13:

initialize\_field\_with\_patterns(field) # Обновляем планер

generation\_count = 0 # Сбрасываем счётчик поколений

else:

field = next\_generation(field) # Вычисляем следующее поколение

generation\_count += 1 # Увеличиваем счётчик поколений

time.sleep(0.1) # Задержка в секундах

if \_\_name\_\_ == "\_\_main\_\_":

main()  
```

Рисунок 3 — Результат работы программы(Изображен период жизни планера равный 4 поколениям)

**Задание №6.2 Многоалфавитное шифрование с использованием алгоритма AES. Дешифрование текста.**

**Вариант 2**

а) систему шифрования AES128 (OFB) для преобразования исходного текста;

б) систему шифрования AES128 (OFB) для преобразования зашифрованного текста в исходный. Для всех вариантов, вывести все сгенерированные ключи, промежуточные результаты State, а также вектор инициализации, добавить генерацию случайного ключа и запись ключа в файл. Реализация должна работать с любым языком, как русским, так и английским.

**Язык программирования : C++**

**Файл: AES.h**

**Код программы :**

**```**#pragma once

#include <vector>

// Определение sBox

extern const std::vector<std::vector<unsigned char>> sBox;

// Определение invsBox

extern const std::vector<std::vector<unsigned char>> invsBox;

// Определение r\_w

extern const std::vector<std::vector<unsigned char>> r\_w;

const std::vector<std::vector<unsigned char>> sBox =

{/\* 0 1 2 3 4 5 6 7 8 9 a b c d e f \*/

{0x63,0x7c,0x77,0x7b,0xf2,0x6b,0x6f,0xc5,0x30,0x01,0x67,0x2b,0xfe,0xd7,0xab,0x76}, /\*0\*/

{0xca,0x82,0xc9,0x7d,0xfa,0x59,0x47,0xf0,0xad,0xd4,0xa2,0xaf,0x9c,0xa4,0x72,0xc0}, /\*1\*/

{0xb7,0xfd,0x93,0x26,0x36,0x3f,0xf7,0xcc,0x34,0xa5,0xe5,0xf1,0x71,0xd8,0x31,0x15}, /\*2\*/

{0x04,0xc7,0x23,0xc3,0x18,0x96,0x05,0x9a,0x07,0x12,0x80,0xe2,0xeb,0x27,0xb2,0x75}, /\*3\*/

{0x09,0x83,0x2c,0x1a,0x1b,0x6e,0x5a,0xa0,0x52,0x3b,0xd6,0xb3,0x29,0xe3,0x2f,0x84}, /\*4\*/

{0x53,0xd1,0x00,0xed,0x20,0xfc,0xb1,0x5b,0x6a,0xcb,0xbe,0x39,0x4a,0x4c,0x58,0xcf}, /\*5\*/

{0xd0,0xef,0xaa,0xfb,0x43,0x4d,0x33,0x85,0x45,0xf9,0x02,0x7f,0x50,0x3c,0x9f,0xa8}, /\*6\*/

{0x51,0xa3,0x40,0x8f,0x92,0x9d,0x38,0xf5,0xbc,0xb6,0xda,0x21,0x10,0xff,0xf3,0xd2}, /\*7\*/

{0xcd,0x0c,0x13,0xec,0x5f,0x97,0x44,0x17,0xc4,0xa7,0x7e,0x3d,0x64,0x5d,0x19,0x73}, /\*8\*/

{0x60,0x81,0x4f,0xdc,0x22,0x2a,0x90,0x88,0x46,0xee,0xb8,0x14,0xde,0x5e,0x0b,0xdb}, /\*9\*/

{0xe0,0x32,0x3a,0x0a,0x49,0x06,0x24,0x5c,0xc2,0xd3,0xac,0x62,0x91,0x95,0xe4,0x79}, /\*a\*/

{0xe7,0xc8,0x37,0x6d,0x8d,0xd5,0x4e,0xa9,0x6c,0x56,0xf4,0xea,0x65,0x7a,0xae,0x08}, /\*b\*/

{0xba,0x78,0x25,0x2e,0x1c,0xa6,0xb4,0xc6,0xe8,0xdd,0x74,0x1f,0x4b,0xbd,0x8b,0x8a}, /\*c\*/

{0x70,0x3e,0xb5,0x66,0x48,0x03,0xf6,0x0e,0x61,0x35,0x57,0xb9,0x86,0xc1,0x1d,0x9e}, /\*d\*/

{0xe1,0xf8,0x98,0x11,0x69,0xd9,0x8e,0x94,0x9b,0x1e,0x87,0xe9,0xce,0x55,0x28,0xdf}, /\*e\*/

{0x8c,0xa1,0x89,0x0d,0xbf,0xe6,0x42,0x68,0x41,0x99,0x2d,0x0f,0xb0,0x54,0xbb,0x16} /\*f\*/

};

extern const std::vector<std::vector<unsigned char>> invsBox =

{/\* 0 1 2 3 4 5 6 7 8 9 a b c d e f \*/

{0x52,0x09,0x6a,0xd5,0x30,0x36,0xa5,0x38,0xbf,0x40,0xa3,0x9e,0x81,0xf3,0xd7,0xfb}, /\*0\*/

{0x7c,0xe3,0x39,0x82,0x9b,0x2f,0xff,0x87,0x34,0x8e,0x43,0x44,0xc4,0xde,0xe9,0xcb}, /\*1\*/

{0x54,0x7b,0x94,0x32,0xa6,0xc2,0x23,0x3d,0xee,0x4c,0x95,0x0b,0x42,0xfa,0xc3,0x4e}, /\*2\*/

{0x08,0x2e,0xa1,0x66,0x28,0xd9,0x24,0xb2,0x76,0x5b,0xa2,0x49,0x6d,0x8b,0xd1,0x25}, /\*3\*/

{0x72,0xf8,0xf6,0x64,0x86,0x68,0x98,0x16,0xd4,0xa4,0x5c,0xcc,0x5d,0x65,0xb6,0x92}, /\*4\*/

{0x6c,0x70,0x48,0x50,0xfd,0xed,0xb9,0xda,0x5e,0x15,0x46,0x57,0xa7,0x8d,0x9d,0x84}, /\*5\*/

{0x90,0xd8,0xab,0x00,0x8c,0xbc,0xd3,0x0a,0xf7,0xe4,0x58,0x05,0xb8,0xb3,0x45,0x06}, /\*6\*/

{0xd0,0x2c,0x1e,0x8f,0xca,0x3f,0x0f,0x02,0xc1,0xaf,0xbd,0x03,0x01,0x13,0x8a,0x6b}, /\*7\*/

{0x3a,0x91,0x11,0x41,0x4f,0x67,0xdc,0xea,0x97,0xf2,0xcf,0xce,0xf0,0xb4,0xe6,0x73}, /\*8\*/

{0x96,0xac,0x74,0x22,0xe7,0xad,0x35,0x85,0xe2,0xf9,0x37,0xe8,0x1c,0x75,0xdf,0x6e}, /\*9\*/

{0x47,0xf1,0x1a,0x71,0x1d,0x29,0xc5,0x89,0x6f,0xb7,0x62,0x0e,0xaa,0x18,0xbe,0x1b}, /\*a\*/

{0xfc,0x56,0x3e,0x4b,0xc6,0xd2,0x79,0x20,0x9a,0xdb,0xc0,0xfe,0x78,0xcd,0x5a,0xf4}, /\*b\*/

{0x1f,0xdd,0xa8,0x33,0x88,0x07,0xc7,0x31,0xb1,0x12,0x10,0x59,0x27,0x80,0xec,0x5f}, /\*c\*/

{0x60,0x51,0x7f,0xa9,0x19,0xb5,0x4a,0x0d,0x2d,0xe5,0x7a,0x9f,0x93,0xc9,0x9c,0xef}, /\*d\*/

{0xa0,0xe0,0x3b,0x4d,0xae,0x2a,0xf5,0xb0,0xc8,0xeb,0xbb,0x3c,0x83,0x53,0x99,0x61}, /\*e\*/

{0x17,0x2b,0x04,0x7e,0xba,0x77,0xd6,0x26,0xe1,0x69,0x14,0x63,0x55,0x21,0x0c,0x7d} /\*f\*/

};

const std::vector<std::vector<unsigned char>> r\_w =

{

{0x01 , 0x00, 0x00, 0x00},

{0x02 , 0x00, 0x00, 0x00},

{0x04 , 0x00, 0x00, 0x00},

{0x08 , 0x00, 0x00, 0x00},

{0x10 , 0x00, 0x00, 0x00},

{0x20 , 0x00, 0x00, 0x00},

{0x40 , 0x00, 0x00, 0x00},

{0x80 , 0x00, 0x00, 0x00},

{0x1b , 0x00, 0x00, 0x00},

{0x36 , 0x00, 0x00, 0x00}

**};  
```  
Язык программирования: C++**

**Файл: lab6.2.cpp  
Код программы:**

**```  
#include** "AES.h"

#include <iostream>

#include <random>

#include <vector>

#include <string>

#include <algorithm>

#include <fstream>

#include <Windows.h>

using namespace std;

// Функция для генерации случайного ключа

void generate\_random\_key(vector<unsigned char>& key) {

key.clear();

key.resize(16); // 16 байт для AES-128

random\_device rd; // Создание объекта random\_device для генерации случайных чисел

mt19937 gen(rd()); // Инициализация генератора случайных чисел mt19937 с использованием random\_device

uniform\_int\_distribution<> dis(0, 255);

for (int i = 0; i < 16; ++i) {

key[i] = dis(gen); // генерируем ключ

}

}

// Функция для записи ключа в файл

void write\_key\_to\_file(const vector<unsigned char>& key, const string& filename) {

ofstream outputFile(filename, ios::binary); // в двоичной записи

if (outputFile.is\_open()) {

for (unsigned char ch : key) {

outputFile.write(reinterpret\_cast<const char\*>(&ch), sizeof(ch)); //преобразуем un char в char,для метода write,возвращаем кол-во байтов записанных в файл

}

outputFile.close();

cout << "Ключ успешно записан в файл " << filename << endl;

}else {

cout << "Не удалось открыть файл для записи ключа" << endl;

}

}

//функция для ввода строки

void init\_str(vector<unsigned char>& str) {

string in\_str;

cout << "(Строка должна быть не больше 16 символов): ";

while (true) {

getline(cin, in\_str);

if (in\_str.length() > 16) {

cout << "Строка больше чем 16 символов, попробуйте еще: ";

}

else {

str.clear(); // Очищаем вектор перед добавлением новых данных

str.reserve(in\_str.size()); // Резервируем место для элементов

for (char c : in\_str) {

str.push\_back(static\_cast<unsigned char>(c));

}

break;

}

}

}

//функция для заполнения матрицы

void init\_matrix(vector<unsigned char>& str, vector<vector<unsigned char>>& matrix) {

// Очищаем матрицу перед заполнением новыми данными

matrix.clear();

// Инициализируем матрицу размером 4x4 нулями

matrix.resize(4, vector<unsigned char>(4, 0));

// Проходим по каждой строке матрицы

for (int i = 0; i < 4; ++i) {

// Проходим по каждому столбцу матрицы

for (int j = 0; j < 4; ++j) {

// Вычисляем индекс элемента матрицы

int index = i \* 4 + j;

// Проверяем, что индекс не выходит за пределы размера вектора str

if (index < str.size()) {

// Присваиваем элементу матрицы значение из вектора str

matrix[i][j] = str[index];

}

}

}

}

//функция для транспонирование матрицы

void transpose\_matrix(vector<vector<unsigned char>>& matrix) {

// Получаем размерность квадратной матрицы

int n = matrix.size();

// Проходим по каждой строке матрицы

for (int i = 0; i < n; ++i) {

// Проходим по каждому столбцу, начиная с i+1, чтобы избежать повторной транспонирования уже транспонированных элементов

for (int j = i + 1; j < n; ++j) {

// Обмениваем значения элементов (i, j) и (j, i), чтобы выполнить транспонирование

swap(matrix[i][j], matrix[j][i]);

}

}

}

//функция для расширения ключа

vector<vector<unsigned char>> key\_expansion(vector<vector<unsigned char>>& matrix, int t) {

// Создаем новый ключ, инициализируем его значениями из исходной матрицы

vector<vector<unsigned char>> exp\_key(matrix);

// Создаем временный вектор для хранения последнего столбца матрицы

vector<unsigned char> temp(4);

// Копируем последний столбец матрицы во временный вектор

for (int i = 0; i < 4; ++i) {

temp[i] = matrix[i][3];

}

// Циклический сдвиг элементов вектора на одну позицию влево

rotate(temp.begin(), temp.begin() + 1, temp.end());

// Заменяем каждый элемент вектора с использованием S-box

for (int i = 0; i < 4; ++i) {

temp[i] = sBox[temp[i] / 16][temp[i] % 16];

}

// Выполняем XOR с раундовой константой rcon

temp[0] = temp[0] ^ r\_w[t][0];

// Генерируем первый столбец нового ключа путем XOR-а каждого элемента соответствующего столбца матрицы и временного вектора

for (int i = 0; i < 4; ++i) {

exp\_key[i][0] = matrix[i][0] ^ temp[i];

}

// Генерируем остальные столбцы нового ключа путем XOR-а предыдущего столбца нового ключа и соответствующего столбца матрицы

for (int i = 1; i < 4; ++i) {

for (int j = 0; j < 4; ++j) {

exp\_key[j][i] = exp\_key[j][i - 1] ^ matrix[j][i];

}

}

// Возвращаем новый расширенный ключ

return exp\_key;

}

//функция операции XOR для каждой пары элементов матрицы matrix1 и matrix2 и сохраняет результат в новой матрице, которая затем возвращается из функции

vector<vector<unsigned char>> xor\_matrices(const vector<vector<unsigned char>>& matrix1, const vector<vector<unsigned char>>& matrix2) {

// Получаем размерность матрицы

int n = matrix1.size();

// Создаем новую матрицу для хранения результата

vector<vector<unsigned char>> result(n, vector<unsigned char>(n));

// Проходим по каждой строке матрицы

for (int i = 0; i < n; ++i) {

// Проходим по каждому столбцу матрицы

for (int j = 0; j < n; ++j) {

// Вычисляем значение элемента новой матрицы как побитовое исключающее ИЛИ элементов соответствующих позиций в матрицах matrix1 и matrix2

result[i][j] = matrix1[i][j] ^ matrix2[i][j];

}

}

// Возвращаем результат

return result;

}

**//**функция применяется к каждому байту матрицы, заменяя его значением из S - блока.(Перестановка с помощью S-блока)

void sub\_bytes(vector<vector<unsigned char>>& matrix) {

// Проходим по каждой строке матрицы

for (int i = 0; i < 4; i++) {

// Проходим по каждому столбцу матрицы

for (int j = 0; j < 4; j++) {

// Получаем текущий байт и используем его значение в качестве индексов для S-блока

unsigned char byte = matrix[i][j]; //текущий байт

matrix[i][j] = sBox[byte / 16][byte % 16];

}

}

}

//функция Перестановка с помощью P-блока

void shiftRows(vector<vector<unsigned char>>& state) {

// Проходим по каждой строке матрицы, начиная с первой (индекс 1)

for (int i = 1; i < 4; ++i) {

// Сдвигаем элементы в текущей строке влево на количество позиций, равное индексу строки

rotate(state[i].begin(), state[i].begin() + i, state[i].end()); // middle становится началом

}

}

**//**функция для смешивания столбцов

void mixColumns(vector<vector<unsigned char>>& state) {

// Проходим по каждому столбцу матрицы состояния (state)

for (int i = 0; i < 4; ++i) {

// Создаем временные массивы для хранения промежуточных значений

unsigned char a[4]; // Для хранения значений текущего столбца

unsigned char b[4]; // Для хранения преобразованных значений текущего столбца

unsigned char h; // Для хранения старшего бита значения байта

// Заполняем массивы a и b и применяем необходимые преобразования

for (int c = 0; c < 4; ++c) {

// Копируем значения текущего столбца в массив a

a[c] = state[c][i];

// Вычисляем старший бит значения байта и сохраняем его в h

h = (unsigned char)((signed char)state[c][i] >> 7); //приводим state сначала к signed, потом к unsigned

// Применяем операцию смешивания (MixColumn), с учетом циклического сдвига и XOR с константой

b[c] = state[c][i] << 1; // Выполняем циклический сдвиг влево на 1 бит

b[c] ^= 0x1B & h; // Если старший бит (MSB) был установлен, выполняем XOR с константой 0x1B ( многочлен в конечном поле Галуа)

}

// Применяем операцию смешивания столбцов (MixColumn) для текущего столбца

// Результаты комбинируем через операцию XOR и записываем обратно в столбец state

state[0][i] = b[0] ^ a[1] ^ b[1] ^ a[2] ^ a[3];

state[1][i] = a[0] ^ b[1] ^ a[2] ^ b[2] ^ a[3];

state[2][i] = a[0] ^ a[1] ^ b[2] ^ a[3] ^ b[3];

state[3][i] = a[0] ^ b[0] ^ a[1] ^ a[2] ^ b[3];

}

}

**//**Функция для вывода матриц

void print\_matrix\_hex(const vector<vector<unsigned char>>& matrix) {

// Проходим по каждой строке матрицы

for (const auto& row : matrix) {

// Выводим значения каждого элемента строки в шестнадцатеричном формате

for (unsigned char value : row) {

// Выводим значение элемента в формате двузначного шестнадцатеричного числа

printf("%02X ", value);

}

// После окончания строки выводим символ перевода строки

cout << "\n";

}

// После вывода всех строк добавляем два символа перевода строки для отделения вывода от другого текста

cout << "\n\n";

}

//Функция для обратной перестановки с S-блоком

void invSubBytes(vector<vector<unsigned char>>& matrix) {

// Проходим по каждому элементу матрицы

for (int i = 0; i < 4; i++) {

for (int j = 0; j < 4; j++) {

// Используем значение байта как индекс для обратной S-бокс (invsBox)

matrix[i][j] = invsBox[matrix[i][j] / 16][matrix[i][j] % 16];

}

}

}

//функция для обратной перестановки с P-блоком

void invShiftRows(vector<vector<unsigned char>>& state) {

// Проходим по каждой строке матрицы состояния, начиная со второй

for (int i = 1; i < 4; ++i) {

// Выполняем обратный циклический сдвиг вправо на i позиций для каждой строки

rotate(state[i].rbegin(), state[i].rbegin() + i, state[i].rend());

}

}

//функция для умножения в поле Галуа

unsigned char gmul(unsigned char a, unsigned char b) {

unsigned char p = 0; // Результат умножения

unsigned char counter; // Счетчик для выполнения операции умножения в цикле

unsigned char highBit; // для проверки старшего бита

// Выполняем умножение в конечном поле Галуа

for (counter = 0; counter < 8; counter++) {

// Если младший бит b равен 1, выполняем операцию XOR между p и a

if (b & 1) {

p ^= a;

}

// Проверяем старший бит a

highBit = (a & 0x80);

// Сдвигаем a на один бит влево

a <<= 1;

// Если старший бит установлен, выполняем операцию XOR с 0x1b

if (highBit) {

a ^= 0x1b;

}

// Сдвигаем b на один бит вправо

b >>= 1;

}

// Возвращаем результат умножения

return p;

}

**//**функция для обратного смешивания столбцов

void invMixColumns(vector<vector<unsigned char>>& state) {

for (int i = 0; i < 4; ++i) {

unsigned char a[4]; //текущее значение

unsigned char b[4]; //преобразованное

// Копируем текущий столбец в массив a

for (int c = 0; c < 4; ++c) {

a[c] = state[c][i];

}

// Вычисляем новые значения столбца b

b[0] = gmul(a[0], 0x0e) ^ gmul(a[1], 0x0b) ^ gmul(a[2], 0x0d) ^ gmul(a[3], 0x09);

b[1] = gmul(a[0], 0x09) ^ gmul(a[1], 0x0e) ^ gmul(a[2], 0x0b) ^ gmul(a[3], 0x0d);

b[2] = gmul(a[0], 0x0d) ^ gmul(a[1], 0x09) ^ gmul(a[2], 0x0e) ^ gmul(a[3], 0x0b);

b[3] = gmul(a[0], 0x0b) ^ gmul(a[1], 0x0d) ^ gmul(a[2], 0x09) ^ gmul(a[3], 0x0e);

// Заменяем значения столбца state новыми значениями из b

for (int c = 0; c < 4; ++c) {

state[c][i] = b[c];

}

}

}

//**Функция для расшифрования (использование других функция для шифровки)**

void encrypted() {

vector<unsigned char> key; // Вектор для хранения ключа

vector<unsigned char> message; // Вектор для хранения сообщения

generate\_random\_key(key); // Генерация случайного ключа

write\_key\_to\_file(key, "key.txt"); // Запись ключа в файл

init\_str(message); // Инициализация сообщения

vector<vector<unsigned char>> key\_matrix;

vector<vector<unsigned char>> message\_matrix;

init\_matrix(key, key\_matrix); // Преобразование ключа в матрицу

init\_matrix(message, message\_matrix); // Преобразование сообщения в матрицу

vector<vector<unsigned char>> key\_exp = key\_expansion(key\_matrix, 0); // Расширение ключа

cout << "Начальный ключ(1):\n";

print\_matrix\_hex(key\_matrix); // Вывод начального ключа в шестнадцатеричном формате

//преобразования

transpose\_matrix(key\_exp); // Транспонирование матрицы расширенного ключа

transpose\_matrix(key\_matrix); // Транспонирование матрицы ключа

transpose\_matrix(message\_matrix); // Транспонирование матрицы сообщения

vector<vector<unsigned char>> result\_matrix = xor\_matrices(key\_matrix, message\_matrix); // Применение операции XOR между ключом и сообщением,дл

for (int i = 1; i < 10; i++) {

sub\_bytes(result\_matrix); // Применение операции SubBytes

shiftRows(result\_matrix); // Применение операции ShiftRows

mixColumns(result\_matrix); // Применение операции MixColumns

cout << "State после раунда " << i << ":\n";

print\_matrix\_hex(result\_matrix); // Вывод промежуточного состояния после каждого раунда

//Генерируем новый раундовйы ключ

result\_matrix = xor\_matrices(result\_matrix, key\_exp); // Применение операции XOR между промежуточным состоянием и раундовым ключом

transpose\_matrix(key\_exp); // Транспонирование матрицы раундового ключа

key\_exp = key\_expansion(key\_exp, i); // Расширение раундового ключа

transpose\_matrix(key\_exp); // Транспонирование матрицы раундового ключа

cout << "Раундовый ключ " << i + 1 << ":\n";

print\_matrix\_hex(key\_exp); // Вывод раундового ключа в шестнадцатеричном формате

}

sub\_bytes(result\_matrix); // Применение операции SubBytes

shiftRows(result\_matrix); // Применение операции ShiftRows

result\_matrix = xor\_matrices(result\_matrix, key\_exp); // Применение операции XOR между промежуточным состоянием и последним раундовым ключом

transpose\_matrix(result\_matrix); // Транспонирование заключительного состояния

cout << "Заключительный State:\n";

print\_matrix\_hex(result\_matrix); // Вывод заключительного состояния в шестнадцатеричном формате

vector<unsigned char> ciphertext; // зашифрвоанный текст

for (const auto& row : result\_matrix) {

for (const unsigned char ch : row) {

ciphertext.push\_back(ch);

}

}

ofstream outputFile("ciphertext.txt", ios::binary);

if (outputFile.is\_open()) {

for (unsigned char ch : ciphertext) {

outputFile.write(reinterpret\_cast<const char\*>(&ch), sizeof(ch));

}

outputFile.close();

cout << "Зашифрованный текст успешно записан в файл ciphertext.txt" << endl;

}

else {

cout << "Не удалось открыть файл для записи" << endl;

}

}

**void** decrypt() {

vector<unsigned char> key; // Вектор для хранения ключа

vector<unsigned char> ciphertext; // Вектор для хранения зашифрованного текста

// Чтение ключа из файла

ifstream keyFile("key.txt", ios::binary);

if (keyFile.is\_open()) {

unsigned char ch;

while (keyFile.read(reinterpret\_cast<char\*>(&ch), sizeof(ch))) {

key.push\_back(ch);

}

keyFile.close();

}else {

cout << "Не удалось открыть файл с ключом" << endl;

return;

}

// Чтение зашифрованного текста из файла

ifstream inputFile("ciphertext.txt", ios::binary);

if (inputFile.is\_open()) {

unsigned char ch;

while (inputFile.read(reinterpret\_cast<char\*>(&ch), sizeof(ch))) {

ciphertext.push\_back(ch);

}

inputFile.close();

}else {

cout << "Не удалось открыть файл для чтения" << endl;

return;

}

vector<vector<unsigned char>> key\_matrix; // Матрица для ключа

vector<vector<unsigned char>> ciphertext\_matrix; // Матрица для зашифрованного текста

init\_matrix(key, key\_matrix); // Преобразование ключа в матрицу

init\_matrix(ciphertext, ciphertext\_matrix); // Преобразование зашифрованного текста в матрицу

cout << "Зашифрованное сообщение (State):\n";

print\_matrix\_hex(ciphertext\_matrix); // Вывод зашифрованного текста в шестнадцатеричном формате

vector<vector<vector<unsigned char>>> round\_key\_exp; // Вектор для хранения расширенных раундовых ключей

vector<vector<unsigned char>> first\_key\_matrix = key\_matrix;

transpose\_matrix(first\_key\_matrix);

round\_key\_exp.push\_back(first\_key\_matrix);

transpose\_matrix(first\_key\_matrix);

// Создание и сохранение расширенных раундовых ключей

for (int i = 0; i < 10; i++) {

vector<vector<unsigned char>> round = key\_expansion(first\_key\_matrix, i);

transpose\_matrix(round);

round\_key\_exp.push\_back(round);

transpose\_matrix(round);

first\_key\_matrix = round;

}

transpose\_matrix(ciphertext\_matrix); // Транспонирование матрицы зашифрованного текста

vector<vector<unsigned char>> result\_matrix = xor\_matrices(ciphertext\_matrix, round\_key\_exp[10]); // Применение операции XOR к зашифрованному тексту и последнему раундовому ключу

// Обратные шаги для расшифровки

for (int i = 9; i > 0; i--) {

invShiftRows(result\_matrix); // Обратная операция ShiftRows

invSubBytes(result\_matrix); // Обратная операция SubBytes

result\_matrix = xor\_matrices(result\_matrix, round\_key\_exp[i]); // Применение операции XOR к промежуточному результату и текущему раундовому ключу

invMixColumns(result\_matrix); // Обратная операция MixColumns

}

invShiftRows(result\_matrix); // Обратная операция ShiftRows

invSubBytes(result\_matrix); // Обратная операция SubBytes

result\_matrix = xor\_matrices(result\_matrix, round\_key\_exp[0]); // Применение операции XOR к промежуточному результату и первому раундовому ключу

transpose\_matrix(result\_matrix); // Транспонирование результата

vector<unsigned char> plaintext; // Вектор для расшифрованного текста

for (const auto& row : result\_matrix) {

for (const unsigned char ch : row) {

plaintext.push\_back(ch);

}

}

// Запись расшифрованного текста в файл

ofstream outputFile("decrypted\_message.txt", ios::binary);

if (outputFile.is\_open()) {

for (unsigned char ch : plaintext) {

outputFile.write(reinterpret\_cast<const char\*>(&ch), sizeof(ch));

}

outputFile.close();

cout << "Расшифрованный текст успешно записан в файл decrypted\_message.txt" << endl;

cout << "Содержимое файла: " << endl;

for (unsigned char ch : plaintext) {

cout << ch;

}

cout << endl;

}

else {

cout << "Не удалось открыть файл для записи" << endl;

}

}

**int** main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

setlocale(LC\_ALL, "rus");

encrypted();

decrypt();

} **```**

**Язык программирования: Python**

**Файл: task6.2.py  
Код программы:**

**```**import os

# Определение r\_w

r\_w = [

[0x01, 0x00, 0x00, 0x00],

[0x02, 0x00, 0x00, 0x00],

[0x04, 0x00, 0x00, 0x00],

[0x08, 0x00, 0x00, 0x00],

[0x10, 0x00, 0x00, 0x00],

[0x20, 0x00, 0x00, 0x00],

[0x40, 0x00, 0x00, 0x00],

[0x80, 0x00, 0x00, 0x00],

[0x1b, 0x00, 0x00, 0x00],

[0x36, 0x00, 0x00, 0x00]

]

# Определение sBox (надо вернуть его определение)

sBox = [

[0x63, 0x7c, 0x77, 0x7b, 0xf2, 0x6b, 0x6f, 0xc5, 0x30, 0x01, 0x67, 0x2b, 0xfe, 0xd7, 0xab, 0x76],

[0xca, 0x82, 0xc9, 0x7d, 0xfa, 0x59, 0x47, 0xf0, 0xad, 0xd4, 0xa2, 0xaf, 0x9c, 0xa4, 0x72, 0xc0],

[0xb7, 0xfd, 0x93, 0x26, 0x36, 0x3f, 0xf7, 0xcc, 0x34, 0xa5, 0xe5, 0xf1, 0x71, 0xd8, 0x31, 0x15],

[0x04, 0xc7, 0x23, 0xc3, 0x18, 0x96, 0x05, 0x9a, 0x07, 0x12, 0x80, 0xe2, 0xeb, 0x27, 0xb2, 0x75],

[0x09, 0x83, 0x2c, 0x1a, 0x1b, 0x6e, 0x5a, 0xa0, 0x52, 0x3b, 0xd6, 0xb3, 0x29, 0xe3, 0x2f, 0x84],

[0x53, 0xd1, 0x00, 0xed, 0x20, 0xfc, 0xb1, 0x5b, 0x6a, 0xcb, 0xbe, 0x39, 0x4a, 0x4c, 0x58, 0xcf],

[0xd0, 0xef, 0xaa, 0xfb, 0x43, 0x4d, 0x33, 0x85, 0x45, 0xf9, 0x02, 0x7f, 0x50, 0x3c, 0x9f, 0xa8],

[0x51, 0xa3, 0x40, 0x8f, 0x92, 0x9d, 0x38, 0xf5, 0xbc, 0xb6, 0xda, 0x21, 0x10, 0xff, 0xf3, 0xd2],

[0xcd, 0x0c, 0x13, 0xec, 0x5f, 0x97, 0x44, 0x17, 0xc4, 0xa7, 0x7e, 0x3d, 0x64, 0x5d, 0x19, 0x73],

[0x60, 0x81, 0x4f, 0xdc, 0x22, 0x2a, 0x90, 0x88, 0x46, 0xee, 0xb8, 0x14, 0xde, 0x5e, 0x0b, 0xdb],

[0xe0, 0x32, 0x3a, 0x0a, 0x49, 0x06, 0x24, 0x5c, 0xc2, 0xd3, 0xac, 0x62, 0x91, 0x95, 0xe4, 0x79],

[0xe7, 0xc8, 0x37, 0x6d, 0x8d, 0xd5, 0x4e, 0xa9, 0x6c, 0x56, 0xf4, 0xea, 0x65, 0x7a, 0xae, 0x08],

[0xba, 0x78, 0x25, 0x2e, 0x1c, 0xa6, 0xb4, 0xc6, 0xe8, 0xdd, 0x74, 0x1f, 0x4b, 0xbd, 0x8b, 0x8a],

[0x70, 0x3e, 0xb5, 0x66, 0x48, 0x03, 0xf6, 0x0e, 0x61, 0x35, 0x57, 0xb9, 0x86, 0xc1, 0x1d, 0x9e],

[0xe1, 0xf8, 0x98, 0x11, 0x69, 0xd9, 0x8e, 0x94, 0x9b, 0x1e, 0x87, 0xe9, 0xce, 0x55, 0x28, 0xdf],

[0x8c, 0xa1, 0x89, 0x0d, 0xbf, 0xe6, 0x42, 0x68, 0x41, 0x99, 0x2d, 0x0f, 0xb0, 0x54, 0xbb, 0x16]

]

def init\_str(str):

in\_str = input("(Строка должна быть не больше 16 символов): ")

while len(in\_str) > 16:

print("Строка больше чем 16 символов, попробуйте еще: ")

in\_str = input()

str.clear()

str.extend(in\_str.encode())

def init\_matrix(str, matrix):

matrix.clear()

matrix.extend([[0]\*4 for \_ in range(4)])

for i in range(4):

for j in range(4):

index = i \* 4 + j

if index < len(str):

matrix[i][j] = str[index]

def transpose\_matrix(matrix):

matrix[:] = list(map(list, zip(\*matrix)))

def key\_expansion(matrix, t):

exp\_key = [row[:] for row in matrix]

state = [row[:] for row in matrix]

state[3] = state[3][1:] + state[3][:1]

for i in range(4):

state[3][i] = sBox[state[3][i] >> 4][state[3][i] & 0xF]

state[3][i] ^= r\_w[t][i]

for j in range(4):

exp\_key[j][i] = exp\_key[j][i] ^ state[3][i]

return exp\_key

def xor\_matrices(matrix1, matrix2):

return [[a ^ b for a, b in zip(row1, row2)] for row1, row2 in zip(matrix1, matrix2)]

def sub\_bytes(matrix):

for i in range(4):

for j in range(4):

matrix[i][j] = sBox[matrix[i][j] >> 4][matrix[i][j] & 0xF]

def shiftRows(state):

for i in range(1, 4):

state[i] = state[i][i:] + state[i][:i]

def mixColumns(state):

for i in range(4):

a = [state[c][i] for c in range(4)]

b = [0]\*4

for c in range(4):

h = (state[c][i] >> 7) & 1

b[c] = (state[c][i] << 1) ^ (0x1B if h else 0)

state[0][i] = b[0] ^ a[1] ^ b[1] ^ a[2] ^ a[3]

state[1][i] = a[0] ^ b[1] ^ a[2] ^ b[2] ^ a[3]

state[2][i] = a[0] ^ a[1] ^ b[2] ^ a[3] ^ b[3]

state[3][i] = a[0] ^ b[0] ^ a[1] ^ a[2] ^ b[3]

def print\_matrix\_hex(matrix):

for row in matrix:

for value in row:

print(f"{value:02X}", end=" ")

print()

print()

def encrypted():

key = []

message = []

init\_str(key)

init\_str(message)

key\_matrix = [[0]\*4 for \_ in range(4)]

message\_matrix = [[0]\*4 for \_ in range(4)]

init\_matrix(key, key\_matrix)

init\_matrix(message, message\_matrix)

key\_exp = key\_expansion(key\_matrix, 0)

print("Ключ :")

print\_matrix\_hex(key\_exp)

transpose\_matrix(key\_exp)

transpose\_matrix(key\_matrix)

transpose\_matrix(message\_matrix)

result\_matrix = xor\_matrices(key\_matrix, message\_matrix)

print("Матрица состояния: ")

print\_matrix\_hex(result\_matrix)

for i in range(1, 10):

sub\_bytes(result\_matrix)

shiftRows(result\_matrix)

mixColumns(result\_matrix)

result\_matrix = xor\_matrices(result\_matrix, key\_exp)

print("Матрица состояния: ")

print\_matrix\_hex(result\_matrix)

transpose\_matrix(key\_exp)

key\_exp = key\_expansion(key\_exp, i)

print("Ключ :")

print\_matrix\_hex(key\_exp)

transpose\_matrix(key\_exp)

sub\_bytes(result\_matrix)

shiftRows(result\_matrix)

result\_matrix = xor\_matrices(result\_matrix, key\_exp)

print("Матрица состояния: ")

print\_matrix\_hex(result\_matrix)

transpose\_matrix(result\_matrix)

ciphertext = [ch for row in result\_matrix for ch in row]

print("Зашифрованное сообщение: ")

for ch in ciphertext:

print(f"{ch:02X}", end=" ")

print()

with open("ciphertext.txt", "wb") as output\_file:

for ch in ciphertext:

output\_file.write(bytes([ch]))

print("Зашифрованный текст успешно записан в файл ciphertext.txt")

if \_\_name\_\_ == "\_\_main\_\_":

encrypted()  
```

**Язык программирования: Python**

**Файл: task6.2(decrypted).py  
Код программы:**  
```  
import os

# Определение sBox

sBox = [

[0x63,0x7c,0x77,0x7b,0xf2,0x6b,0x6f,0xc5,0x30,0x01,0x67,0x2b,0xfe,0xd7,0xab,0x76],

[0xca,0x82,0xc9,0x7d,0xfa,0x59,0x47,0xf0,0xad,0xd4,0xa2,0xaf,0x9c,0xa4,0x72,0xc0],

[0xb7,0xfd,0x93,0x26,0x36,0x3f,0xf7,0xcc,0x34,0xa5,0xe5,0xf1,0x71,0xd8,0x31,0x15],

[0x04,0xc7,0x23,0xc3,0x18,0x96,0x05,0x9a,0x07,0x12,0x80,0xe2,0xeb,0x27,0xb2,0x75],

[0x09,0x83,0x2c,0x1a,0x1b,0x6e,0x5a,0xa0,0x52,0x3b,0xd6,0xb3,0x29,0xe3,0x2f,0x84],

[0x53,0xd1,0x00,0xed,0x20,0xfc,0xb1,0x5b,0x6a,0xcb,0xbe,0x39,0x4a,0x4c,0x58,0xcf],

[0xd0,0xef,0xaa,0xfb,0x43,0x4d,0x33,0x85,0x45,0xf9,0x02,0x7f,0x50,0x3c,0x9f,0xa8],

[0x51,0xa3,0x40,0x8f,0x92,0x9d,0x38,0xf5,0xbc,0xb6,0xda,0x21,0x10,0xff,0xf3,0xd2],

[0xcd,0x0c,0x13,0xec,0x5f,0x97,0x44,0x17,0xc4,0xa7,0x7e,0x3d,0x64,0x5d,0x19,0x73],

[0x60,0x81,0x4f,0xdc,0x22,0x2a,0x90,0x88,0x46,0xee,0xb8,0x14,0xde,0x5e,0x0b,0xdb],

[0xe0,0x32,0x3a,0x0a,0x49,0x06,0x24,0x5c,0xc2,0xd3,0xac,0x62,0x91,0x95,0xe4,0x79],

[0xe7,0xc8,0x37,0x6d,0x8d,0xd5,0x4e,0xa9,0x6c,0x56,0xf4,0xea,0x65,0x7a,0xae,0x08],

[0xba,0x78,0x25,0x2e,0x1c,0xa6,0xb4,0xc6,0xe8,0xdd,0x74,0x1f,0x4b,0xbd,0x8b,0x8a],

[0x70,0x3e,0xb5,0x66,0x48,0x03,0xf6,0x0e,0x61,0x35,0x57,0xb9,0x86,0xc1,0x1d,0x9e],

[0xe1,0xf8,0x98,0x11,0x69,0xd9,0x8e,0x94,0x9b,0x1e,0x87,0xe9,0xce,0x55,0x28,0xdf],

[0x8c,0xa1,0x89,0x0d,0xbf,0xe6,0x42,0x68,0x41,0x99,0x2d,0x0f,0xb0,0x54,0xbb,0x16]

]

# Определение invsBox

invsBox = [

[0x52,0x09,0x6a,0xd5,0x30,0x36,0xa5,0x38,0xbf,0x40,0xa3,0x9e,0x81,0xf3,0xd7,0xfb],

[0x7c,0xe3,0x39,0x82,0x9b,0x2f,0xff,0x87,0x34,0x8e,0x43,0x44,0xc4,0xde,0xe9,0xcb],

[0x54,0x7b,0x94,0x32,0xa6,0xc2,0x23,0x3d,0xee,0x4c,0x95,0x0b,0x42,0xfa,0xc3,0x4e],

[0x08,0x2e,0xa1,0x66,0x28,0xd9,0x24,0xb2,0x76,0x5b,0xa2,0x49,0x6d,0x8b,0xd1,0x25],

[0x72,0xf8,0xf6,0x64,0x86,0x68,0x98,0x16,0xd4,0xa4,0x5c,0xcc,0x5d,0x65,0xb6,0x92],

[0x6c,0x70,0x48,0x50,0xfd,0xed,0xb9,0xda,0x5e,0x15,0x46,0x57,0xa7,0x8d,0x9d,0x84],

[0x90,0xd8,0xab,0x00,0x8c,0xbc,0xd3,0x0a,0xf7,0xe4,0x58,0x05,0xb8,0xb3,0x45,0x06],

[0xd0,0x2c,0x1e,0x8f,0xca,0x3f,0x0f,0x02,0xc1,0xaf,0xbd,0x03,0x01,0x13,0x8a,0x6b],

[0x3a,0x91,0x11,0x41,0x4f,0x67,0xdc,0xea,0x97,0xf2,0xcf,0xce,0xf0,0xb4,0xe6,0x73],

[0x96,0xac,0x74,0x22,0xe7,0xad,0x35,0x85,0xe2,0xf9,0x37,0xe8,0x1c,0x75,0xdf,0x6e],

]

r\_w = [

[0x01, 0x00, 0x00, 0x00],

[0x02, 0x00, 0x00, 0x00],

[0x04, 0x00, 0x00, 0x00],

[0x08, 0x00, 0x00, 0x00],

[0x10, 0x00, 0x00, 0x00],

[0x20, 0x00, 0x00, 0x00],

[0x40, 0x00, 0x00, 0x00],

[0x80, 0x00, 0x00, 0x00],

[0x1b, 0x00, 0x00, 0x00],

[0x36, 0x00, 0x00, 0x00]

]

def init\_str(str):

in\_str = input("(Строка должна быть не больше 16 символов): ")

while len(in\_str) > 16:

print("Строка больше чем 16 символов, попробуйте еще: ")

in\_str = input()

str.clear()

str.extend(in\_str.encode())

def init\_matrix(str, matrix):

matrix.clear()

matrix.extend([[0]\*4 for \_ in range(4)])

for i in range(4):

for j in range(4):

index = i \* 4 + j

if index < len(str):

matrix[i][j] = str[index]

def transpose\_matrix(matrix):

matrix[:] = list(map(list, zip(\*matrix)))

def key\_expansion(matrix, t):

exp\_key = [row[:] for row in matrix]

state = [row[:] for row in matrix]

state[3] = state[3][1:] + state[3][:1]

for i in range(4):

state[3][i] = sBox[state[3][i] >> 4][state[3][i] & 0xF]

state[3][i] ^= r\_w[t][i]

for j in range(4):

exp\_key[j][i] = exp\_key[j][i] ^ state[3][i]

return exp\_key

def xor\_matrices(matrix1, matrix2):

return [[a ^ b for a, b in zip(row1, row2)] for row1, row2 in zip(matrix1, matrix2)]

def invSubBytes(matrix):

for i in range(4):

for j in range(4):

matrix[i][j] = invsBox[matrix[i][j] >> 4][matrix[i][j] & 0xF]

def invShiftRows(state):

for i in range(1, 4):

state[i] = state[i][-i:] + state[i][:-i]

def gmul(a, b):

p = 0

counter = 0

while counter < 8:

if b & 1:

p ^= a

hi\_bit\_set = a & 0x80

a <<= 1

if hi\_bit\_set:

a ^= 0x1b

b >>= 1

counter += 1

return p

def invMixColumns(state):

for i in range(4):

a = [state[c][i] for c in range(4)]

b = [0]\*4

b[0] = gmul(a[0], 0x0e) ^ gmul(a[1], 0x0b) ^ gmul(a[2], 0x0d) ^ gmul(a[3], 0x09)

b[1] = gmul(a[0], 0x09) ^ gmul(a[1], 0x0e) ^ gmul(a[2], 0x0b) ^ gmul(a[3], 0x0d)

b[2] = gmul(a[0], 0x0d) ^ gmul(a[1], 0x09) ^ gmul(a[2], 0x0e) ^ gmul(a[3], 0x0b)

b[3] = gmul(a[0], 0x0b) ^ gmul(a[1], 0x0d) ^ gmul(a[2], 0x09) ^ gmul(a[3], 0x0e)

for c in range(4):

state[c][i] = b[c]

def print\_matrix\_hex(matrix):

for row in matrix:

for value in row:

print(f"{value:02X}", end=" ")

print()

print()

def decrypt():

key = []

ciphertext = []

init\_str(key)

with open("ciphertext.txt", "rb") as input\_file:

while True:

ch = input\_file.read(1)

if not ch:

break

ciphertext.append(ord(ch))

key\_matrix = [[0]\*4 for \_ in range(4)]

ciphertext\_matrix = [[0]\*4 for \_ in range(4)]

init\_matrix(key, key\_matrix)

init\_matrix(ciphertext, ciphertext\_matrix)

print("Зашифрованное сообщение")

print\_matrix\_hex(ciphertext\_matrix)

round\_key\_exp = []

firs\_key\_matrix = [row[:] for row in key\_matrix]

transpose\_matrix(firs\_key\_matrix)

round\_key\_exp.append(firs\_key\_matrix)

transpose\_matrix(firs\_key\_matrix)

for i in range(10):

round = key\_expansion(firs\_key\_matrix, i)

transpose\_matrix(round)

round\_key\_exp.append(round)

transpose\_matrix(round)

firs\_key\_matrix = round

matrix in round\_key\_exp

print("Ключ:")

print\_matrix\_hex(matrix)

transpose\_matrix(ciphertext\_matrix)

result\_matrix = xor\_matrices(ciphertext\_matrix, round\_key\_exp[10])

print("Матрица состояния:")

print\_matrix\_hex(result\_matrix)

for i in range(9, 0, -1):

invShiftRows(result\_matrix)

invSubBytes(result\_matrix)

result\_matrix = xor\_matrices(result\_matrix, round\_key\_exp[i])

print("Матрица состояния:")

print\_matrix\_hex(result\_matrix)

invMixColumns(result\_matrix)

invShiftRows(result\_matrix)

invSubBytes(result\_matrix)

result\_matrix = xor\_matrices(result\_matrix, round\_key\_exp[0])

print("Матрица состояния:")

print\_matrix\_hex(result\_matrix)

transpose\_matrix(result\_matrix)

plaintext = [ch for row in result\_matrix for ch in row]

print("Расшифрованный текст сообщения:")

for ch in plaintext:

print(f"{ch:02X}", end=" ")

print()

with open("decrypted\_message.txt", "wb") as output\_file:

for ch in plaintext:

output\_file.write(bytes([ch]))

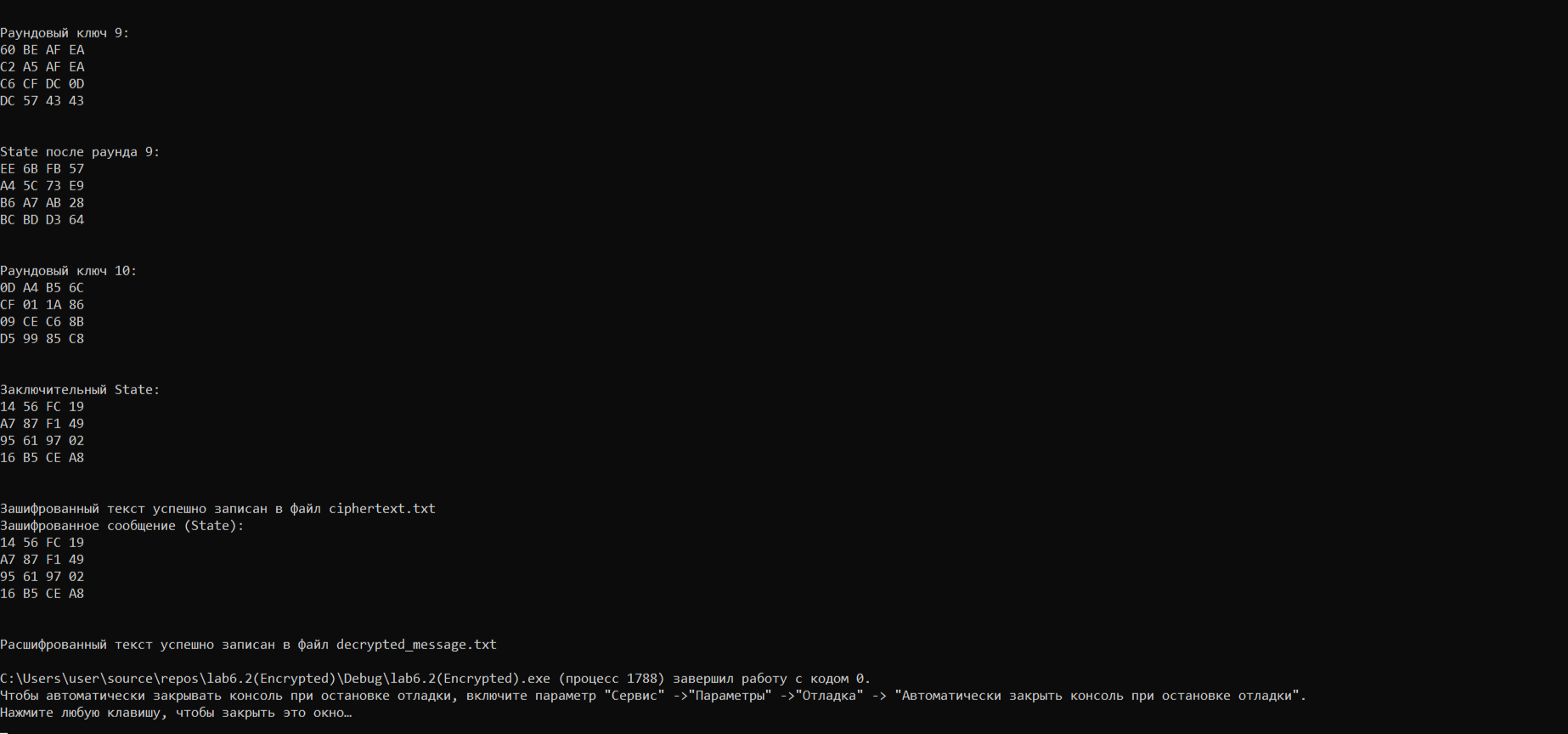
print("Расшифрованный текст успешно записан в файл decrypted\_message.txt")

if \_\_name\_\_ == "\_\_main\_\_":

decrypt()  
```  
Р**езультат работы программы:**

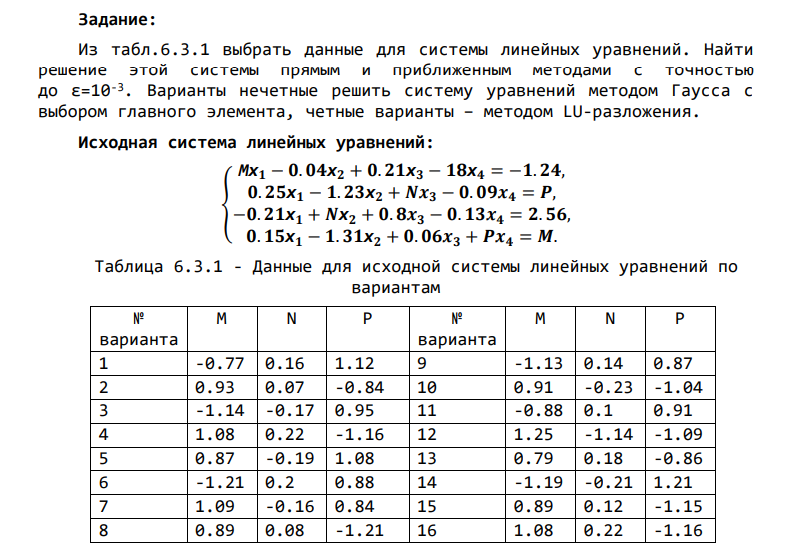




Рисунок 4 — Результат работы программы(Aes)

**Задание №6.3 Методы решения систем линейных алгебраических уравнений**

**Вариант 6**

 **Язык программирования : С++**

**Задание 6.3**

**Код программы :**

**```  
#include** <iostream>

#include <iomanip>

#include <cmath>

using namespace std;

const int SizeMatrix = 4; // Количество уравнений и переменных

// Функция для вычисления нового значения переменной xi на k-й итерации

double calculateNewValue(int k, int i, double\*\* matrix, double\* reply, double\* x) {

double sum1 = 0, sum2 = 0;

for (int j = 0; j < i; ++j) {

sum1 += matrix[i][j] \* x[j]; // Сумма соседних переменных с меньшим индексом aij \* xj(k+1)

}

for (int j = i + 1; j < SizeMatrix; ++j) {

sum2 += matrix[i][j] \* x[j]; // Сумма соседних переменных с большим индексом

}

return (reply[i] - sum1 - sum2) / matrix[i][i]; // Новое значение переменной xi ; основная формула

}

// Функция для решения системы уравнений методом Зейделя

void solveSystem(double\*\* matrix, double\* reply, double\* x, double epsilon, int maxIterations) {

double\* xNew = new double[SizeMatrix](); // Новое приближение к решению

int iterations = 0; // Счетчик итераций

cout << "|-----|----------|----------|----------|----------|-------------|" << endl;

cout << "| N | x1 | x2 | x3 | x4 | Epsilon |" << endl;

cout << "|-----|----------|----------|----------|----------|-------------|" << endl;

while (iterations < maxIterations) {

double maxDiff = 0; // Максимальное изменение переменной на текущей итерации

for (int i = 0; i < SizeMatrix; ++i) {

double oldValue = xNew[i]; //обновляем старое значение

xNew[i] = calculateNewValue(iterations, i, matrix, reply, xNew); // Вычисление нового значения переменной xi

maxDiff = max(maxDiff, abs(xNew[i] - oldValue)); // Обновление максимального изменения для сравнения с epsilon

}

cout << "| " << iterations << " | " << setw(8) << xNew[0] << " | " << setw(8) << xNew[1] << " | " << setw(8) << xNew[2] << " | " << setw(8) << xNew[3] << " | " << setw(11) << epsilon << " |" << endl;

if (maxDiff < epsilon) { // Проверка условия сходимости

cout << "|-----|----------|----------|----------|----------|-------------|" << endl << endl;

cout << "Количество итераций: " << iterations << endl;

break;

}

iterations++;

}

// Вывод решения

cout << "Решение:" << endl;

for (int i = 0; i < SizeMatrix; ++i) {

cout << "x" << i + 1 << " = " << xNew[i] << endl;

}

}

void canonicalForm(double\*\* matrix, double\* reply) { //вывод канонического вида

cout << "Канонический вид: " << endl;

string variables[] = { "x1", "x2", "x3", "x4"};

for (int i = 0; i < SizeMatrix; ++i) {

for (int j = 0; j < SizeMatrix; ++j) {

cout << "(" << setw(4) << fixed << setprecision(3) << matrix[i][j] << "\*" << variables[j] << ")";

if (j < SizeMatrix - 1) {

cout << " + ";

}

}

cout << " = " << reply[i] << endl;

}

cout << endl;

}

// Функция для выполнения LU-разложения

void luDecomposition(double\*\* A, double\*\* L, double\*\* U, int n) {

for (int i = 0; i < n; i++) { //// Проходим по каждой строке матрицы A (размерность n)

// Вычисление элементов верхнетреугольной матрицы U

for (int j = i; j < n; j++) { // Внутренний цикл этого цикла проходится по столбцам матрицы A, начиная с i-го столбца.

//Переменная j здесь используется для индексации столбцов.

double sum = 0;

// Вычисление суммы произведений элементов нижнетреугольной матрицы L и верхнетреугольной матрицы U

for (int k = 0; k < i; k++) {

sum += L[i][k] \* U[k][j];

}

// Вычисление элемента верхнетреугольной матрицы U

U[i][j] = A[i][j] - sum;

}

// Вычисление элементов нижнетреугольной матрицы L

for (int i1 = i; i1 < n; i1++) {

int j = i; // столбец для L

// Если индекс строки равен индексу столбца, устанавливаем соответствующий элемент матрицы L в 1

if (i == i1) { //диагонали

L[i][i1] = 1;

}else {

// Инициализация суммы нулем

double sum = 0;

// Вычисление суммы произведений элементов нижнетреугольной матрицы L и верхнетреугольной матрицы U

for (int k = 0; k < j; k++) {

sum += L[i1][k] \* U[k][j];

}

// Вычисление элемента нижнетреугольной матрицы L

L[i1][j] = (A[i1][j] - sum) / U[i][j];

}

}

}

// Вывод матриц U и L

cout << "Матрица U:" << endl;

for (int i = 0; i < SizeMatrix; ++i) {

for (int j = 0; j < SizeMatrix; ++j) {

cout << fixed << setprecision(4) << setw(8) << U[i][j] << " ";

}

cout << endl;

}

cout << "Матрица L:" << endl;

for (int i = 0; i < SizeMatrix; ++i) {

for (int j = 0; j < SizeMatrix; ++j) {

cout << fixed << setprecision(4) << setw(8) << L[i][j] << " ";

}

cout << endl;

}

}

// Функция для выполнения прямой подстановки

double\* forwardSubstitution(double\*\* L, double\* b, int n) { //Решаем Ly = b

// Создаем массив y для хранения результатов прямой подстановки

double\* y = new double[n]();

// Проходимся по каждой строке матрицы L

for (int k = 0; k < n; k++) {

// Инициализируем y[k] значением b[k]

y[k] = b[k];

// Проходимся по каждому столбцу до диагонального элемента

for (int j = 0; j < k; j++) {

// Вычитаем сумму произведений из y[k]

y[k] -= L[k][j] \* y[j];

}

}

// Возвращаем массив y

return y;

}

// Функция для выполнения обратной подстановки

double\* backwardSubstitution(double\*\* U, double\* y, int n) { // Решаем Ux = y

// Создаем массив x для хранения результатов обратной подстановки

double\* x = new double[n]();

// Проходимся по каждой строке матрицы U, начиная с последней

for (int k = n - 1; k >= 0; k--) {

// Инициализируем x[k] значением y[k]

x[k] = y[k];

// Проходимся по каждому столбцу после диагонального элемента

for (int j = k + 1; j < n; j++) {

// Вычитаем сумму произведений из x[k]

x[k] -= U[k][j] \* x[j];

}

// Делим x[k] на соответствующий диагональный элемент U[k][k]

x[k] /= U[k][k];

}

// Возвращаем массив x

return x;

}

int main() {

setlocale(LC\_ALL, "ru");

double M = -1.21, N = 0.2, P = 0.88; // переменные по варианту

double epsilon = 0.001; // точность

int maxIterations = 1000; // макс. кол-во итераций

int n = 4; // размер матрицы

// Создание матрицы A и вектора b

double\*\* A = new double\* [n];

for (int i = 0; i < n; i++) {

A[i] = new double[n];

}

A[0][0] = M; A[0][1] = -0.04; A[0][2] = 0.21; A[0][3] = -18;

A[1][0] = 0.25; A[1][1] = -1.23; A[1][2] = N; A[1][3] = -0.09;

A[2][0] = -0.21; A[2][1] = N; A[2][2] = 0.8; A[2][3] = -0.13;

A[3][0] = 0.15; A[3][1] = -1.31; A[3][2] = 0.06; A[3][3] = P;

double\* b = new double[n]; //вектор b

b[0] = -1.24; b[1] = P; b[2] = 2.56; b[3] = M;

// Создание матрицы L и U

double\*\* L = new double\* [n]; //выделяем память под массив указателей

double\*\* U = new double\* [n];

for (int i = 0; i < n; i++) {

L[i] = new double[n](); //выделяем память для каждой строки

U[i] = new double[n]();

}

// Выполнение LU-разложения

luDecomposition(A, L, U, n);

// Прямая подстановка для нахождения y

double\* y = forwardSubstitution(L, b, n);

// Обратная подстановка для нахождения x

double\* x = backwardSubstitution(U, y, n);

// Вывод результатов прямой подстановки

cout << "Результаты прямой подстановки (нахождение y):" << endl;

for (int i = 0; i < n; i++) {

cout << "y" << i + 1 << " = " << y[i] << endl;

}

// Вывод результатов обратной подстановки

cout << "Результаты обратной подстановки (нахождение x):" << endl;

for (int i = 0; i < n; i++) {

cout << "x" << i + 1 << " = " << x[i] << endl;

}

//2 Способ

// Матрица коэффициентов системы уравнений с наибольшими элементами по диагонали

double\*\* matrix = new double\* [SizeMatrix];

for (int i = 0; i < SizeMatrix; ++i) {

matrix[i] = new double[SizeMatrix];

}

matrix[0][0] = 0.25; matrix[0][1] = -1.23; matrix[0][2] = 0.2; matrix[0][3] = -0.09;

matrix[1][0] = -0.15; matrix[1][1] = 1.31; matrix[1][2] = -0.06; matrix[1][3] = -0.88;

matrix[2][0] = -0.21; matrix[2][1] = 0.2; matrix[2][2] = 0.8; matrix[2][3] = -0.13;

matrix[3][0] = 1.21; matrix[3][1] = 0.04; matrix[3][2] = -0.21; matrix[3][3] = 18;

// Вектор свободных членов

double\* reply = new double[SizeMatrix];

reply[0] = 0.88; reply[1] = 1.21; reply[2] = 2.56; reply[3] = 1.24;

canonicalForm(matrix, reply);

// Начальное приближение к решению (все переменные равны 0)

double\* x1 = new double[SizeMatrix]();

// Решение системы уравнений

solveSystem(matrix, reply, x1, epsilon, maxIterations);

return 0;

}

```

**Язык программирования : Python**

**Задание 6.3**

**Код программы :**

**```**

import numpy as np

def calculate\_new\_value(i, matrix, reply, x):

sum1 = sum(matrix[i][j] \* x[j] for j in range(i))

sum2 = sum(matrix[i][j] \* x[j] for j in range(i + 1, len(matrix)))

return (reply[i] - sum1 - sum2) / matrix[i][i]

def solve\_system(matrix, reply, x, epsilon, max\_iterations):

n = len(matrix)

x\_new = np.zeros(n)

iterations = 0

print("|-----|----------|----------|----------|----------|-------------|")

print("| N | x1 | x2 | x3 | x4 | Epsilon |")

print("|-----|----------|----------|----------|----------|-------------|")

while iterations < max\_iterations:

max\_diff = 0

for i in range(n):

old\_value = x\_new[i]

x\_new[i] = calculate\_new\_value(i, matrix, reply, x\_new)

max\_diff = max(max\_diff, abs(x\_new[i] - old\_value))

print(f"| {iterations:3d} | {x\_new[0]:8.4f} | {x\_new[1]:8.4f} | {x\_new[2]:8.4f} | {x\_new[3]:8.4f} | {epsilon:11.4f} |")

if max\_diff < epsilon:

print("|-----|----------|----------|----------|----------|-------------|")

print(f"\nКоличество итераций: {iterations}")

break

iterations += 1

print("Решение:")

for i in range(n):

print(f"x{i + 1} = {x\_new[i]:.4f}")

def output(matrix, reply):

print("Канонический вид:")

variables = ['a', 'b', 'c', 'd']

for i in range(len(matrix)):

row = " + ".join(f"{matrix[i][j]:8.3f}\*{variables[j]}" for j in range(len(matrix[i])))

print(f"{row} = {reply[i]:.3f}")

print()

def lu\_decomposition(A):

n = len(A)

L = np.zeros((n, n))

U = np.zeros((n, n))

for i in range(n):

for j in range(i, n):

sum\_value = sum(L[i][k] \* U[k][j] for k in range(i))

U[i][j] = A[i][j] - sum\_value

for j in range(i, n):

if i == j:

L[i][j] = 1

else:

sum\_value = sum(L[j][k] \* U[k][i] for k in range(i))

L[j][i] = (A[j][i] - sum\_value) / U[i][i]

print("Матрица U:")

print(U)

print("Матрица L:")

print(L)

return L, U

def forward\_substitution(L, b):

n = len(L)

y = np.zeros(n)

for i in range(n):

y[i] = b[i] - sum(L[i][j] \* y[j] for j in range(i))

return y

def backward\_substitution(U, y):

n = len(U)

x = np.zeros(n)

for i in range(n - 1, -1, -1):

x[i] = (y[i] - sum(U[i][j] \* x[j] for j in range(i + 1, n))) / U[i][i]

return x

def main():

M = -1.21

N = 0.2

P = 0.88

epsilon = 0.001

max\_iterations = 1000

A = np.array([

[M, -0.04, 0.21, -18],

[0.25, -1.23, N, -0.09],

[-0.21, N, 0.8, -0.13],

[0.15, -1.31, 0.06, P]

])

b = np.array([-1.24, P, 2.56, M])

L, U = lu\_decomposition(A)

y = forward\_substitution(L, b)

x = backward\_substitution(U, y)

print("Результаты прямой подстановки (нахождение y):")

for i in range(len(y)):

print(f"y{i + 1} = {y[i]:.4f}")

print("Результаты обратной подстановки (нахождение x):")

for i in range(len(x)):

print(f"x{i + 1} = {x[i]:.4f}")

matrix = np.array([

[0.25, -1.23, 0.2, -0.09],

[-0.15, 1.31, -0.06, -0.88],

[-0.21, 0.2, 0.8, -0.13],

[1.21, 0.04, -0.21, 18]

])

reply = np.array([0.88, 1.21, 2.56, 1.24])

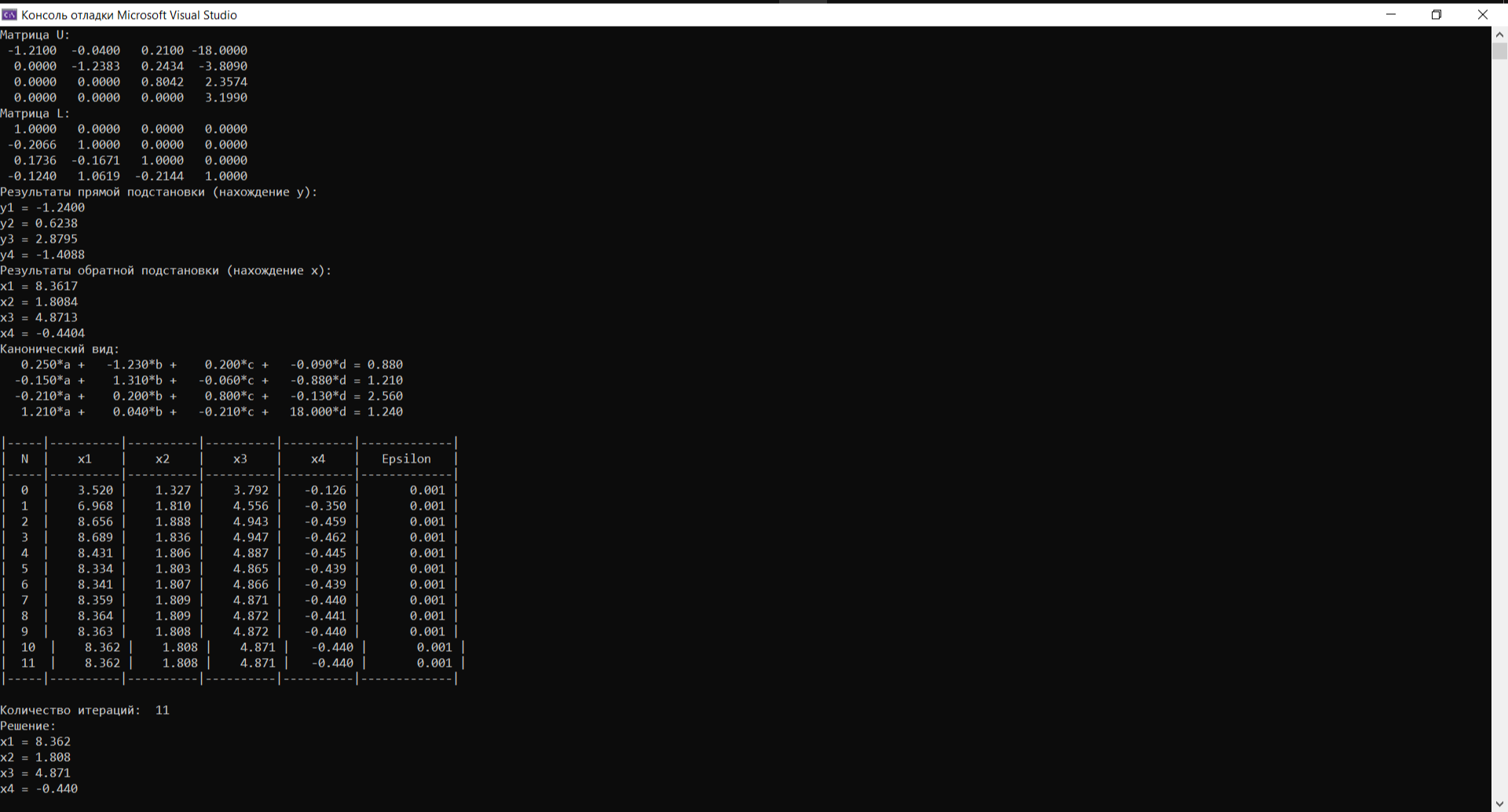
output(matrix, reply)

x1 = np.zeros(len(matrix))

solve\_system(matrix, reply, x1, epsilon, max\_iterations)

if \_\_name\_\_ == "\_\_main\_\_":

main()  
```

Рисунок 5 – Результат работы программы

**Вывод**

В ходе лабораторной работы были выполнены все поставленные задачи, которые были направлены на изучение алгоритмов формирования и обработки двумерных массивов, программирование и отладка программ формирования и обработки массивов (статических и динамических) и контейнеров STL, на двух языках программирования: C++ и Python.