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:	Проверил:
Студент гр. «АБс-323», «АВТФ»	*Ассистент кафедры ЗИ*
Пушкарев Виталий Иванович	*Исаев Глеб Андреевич*
«03» 06 2024г	«03» 06 2024г
_____	_____
(подпись)	(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 (C++ и второй, по выбору).

3. Протестировать работу программы на различных исходных данных.

Задача 1:

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.

Задача 2:

Вариант 1

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

б) систему шифрования AES128 (CFB) для преобразования зашифрованного текста в исходный.

Задача 3:

Вариант 6

Задание:

Из табл.6.3.1 выбрать данные для системы линейных уравнений. Найти решение этой системы прямым и приближенным методами с точностью до $\varepsilon=10^{-3}$. Варианты нечетные решить систему уравнений методом Гаусса с выбором главного элемента, четные варианты – методом LU-разложения.

Исходная система линейных уравнений:

$$\begin{cases} Mx_1 - 0.04x_2 + 0.21x_3 - 18x_4 = -1.24, \\ 0.25x_1 - 1.23x_2 + Nx_3 - 0.09x_4 = P, \\ -0.21x_1 + Nx_2 + 0.8x_3 - 0.13x_4 = 2.56, \\ 0.15x_1 - 1.31x_2 + 0.06x_3 + Px_4 = M. \end{cases}$$

Таблица 6.3.1 - Данные для исходной системы линейных уравнений по вариантам

Задача 1.

C++:

```
#include <iostream>
#include <vector>
#include <string>
#include<random>
#include <algorithm>
#include <iomanip>
#include <Windows.h>

using namespace std;

int GetRandomNumber(int min, int max)
{
    random_device rd;//random_device, который является источником
недетерминированных случайных чисел.
    //Затем мы используем это устройство для заполнения генератора
std::minstd_rand. Затем функция генератора() используется для генерации
случайного числа
    minstd_rand generator(rd());

    uniform_int_distribution<int> distribution(min, max);// функция destribition
для задания диапозона значений
    int random_number = distribution(generator);
    return random_number;
}

void  task1()
{
    cout << "Task 1" << endl;
    int quantityLine = 5, quantityColumns = 6;
    int maxSrednee = 0, numberLine = 0;

    vector<vector<int>> randomMatrix(quantityLine,
vector<int>(quantityColumns));;

    for (auto& i : randomMatrix)
    {
        for (auto& j : i)
```

```
        {
            j = GetRandomNumber(0, 100);
        }
    }
    for (auto& i : randomMatrix)
    {
        for (auto& j : i)
        {
            cout << setw(4) << j;
        }
        cout << endl;
    }

    for (int i = 0; i < quantityLine; i++)
    {
        float sredneeArithmetic = 0;

        for (int j = 0; j < quantityColumns; j++)
        {
            sredneeArithmetic += randomMatrix[i][j];// среднее арифетическое
строки
        }
        if (maxSrednee < sredneeArithmetic)// максимальное среднее арифметическое
        {
            maxSrednee = sredneeArithmetic;
            numberLine = i;
        }
        cout << "Номер строки " << i << " среднее арифметическое = " <<
sredneeArithmetic / quantityColumns << endl;
    }
    cout << "Наибольшее среднеее арифметическое в строке " << numberLine;
    cout << endl;
}


void task2()
{
    cout << "Task 2" << endl;

    int sizeSquareMatrix = 15;

    vector<vector<int>> squareMatrix(sizeSquareMatrix,
vector<int>(sizeSquareMatrix));

    for (auto& i : squareMatrix)
    {
        for (auto& j : i)
        {
            j = GetRandomNumber(-100, 100);
            cout << setw(4) <<j;
        }
```

```
        cout << endl;
    }

    for (int i = 0; i < sizeSquareMatrix; i++)// цикл наоборот. Берем первый
элемент строки
    {
        vector<int> columns(sizeSquareMatrix);// вектор для хранение стобцов
        for (int j = 0; j < sizeSquareMatrix; j++)// здесь рассматриваем строки
        {
           columns[j] = squareMatrix[j][i];// закидываем в вектор столбец все
элементы первого столбца где j номер строки

        }
        if (i % 2 == 0)// если строка четная то сортируем в порядке возрастания
        {
            sort(columns.begin(), columns.end());
        }
        else// если не четная то в порядке убывания
        {
            sort(columns.rbegin(), columns.rend());
        }
        for (int k = 0; k < sizeSquareMatrix; k++)
        {
            squareMatrix[k][i] = columns[k];
        }
    }
    cout << endl;

    for (auto& i : squareMatrix)
    {
        for (auto& j : i)
        {
            cout << setw(4) << j;
        }
        cout << endl;
    }

    int sumRigth = 0;

    for (int i = 0; i < squareMatrix.size(); i++)// высчитываем в правый угол
относительно диагонали
    {
        for (int j = i; j < sizeSquareMatrix; j++)
        {
            if (squareMatrix[i][j] > 0) sumRigth++;
        }

    }

    int sumLeft = 0;
```

```
    for (int i = sizeSquareMatrix; i > 0; i--)// высчитываем в левый угол
относительно диагонали
    {
        for (int j = i; j > 0; j--)
        {
            if (squareMatrix[i-1][j-1] > 0) sumLeft++;
        }

    }
    cout << "Количество положительных элементов правой половины = " << sumRigth
<< "\t количество положительных элементов левой половины = " << sumLeft << endl;
    if (sumRigth > sumLeft) cout << "Правая половина содержит больше
положительных элементов "<<endl;
    else cout << "Левая половина содержит больше положительных элементов" <<
endl;
}

int main()
{
    SetConsoleCP(1251);
    SetConsoleOutputCP(1251);

    task1();// доделать какието элементы матрицы
    cout << endl;
    task2();


}
```

Gamelife:

Header.h

```
#ifndef GAMELIFE_H
#define GAMELIFE_H


#include <iostream>
#include <cstdlib>
#include <ctime>
#include <vector>
#include <algorithm>
#include <numeric>
#include <string>
#include <ncurses.h>
#include <unistd.h>
#include <fstream>
#include <sstream>

using namespace std;
```

```
void generatePlayingField(vector<vector<int>> &matrix, int lines, int columns);
pair<int, int> readingPlayingField(vector<std::vector<int>> &matrix);
void fieldDisplay(const vector<std::vector<int>> &matrix, int lines, int
columns);
int cellCounting(vector<vector<int>> &matrix, int i, int j, int lines, int
columns);
void updateMatrix(vector<vector<int>> &matrix, int lines, int columns);
void playLife(vector<vector<int>> &matrix, int lines, int columns);









#endif
```

Gamelife.cpp

```
#include "header.h"

void generatePlayingField(vector<vector<int>> &matrix, int lines, int columns)
{
    std::srand(time(nullptr));
    for (int i = 0; i < lines; i++)
    {
        for (int j = 0; j < columns; j++)
        {
            matrix[i][j] = rand() % 2; // 0 или 1
        }
    }
}

std::pair<int, int> readingPlayingField(std::vector<std::vector<int>> &matrix)
{
    std::vector<std::vector<int>> newMatrix;

    std::ifstream inputFile("train.txt");
    if (!inputFile)
    {
        std::cerr << "Не удалось открыть файл input.txt" << std::endl;
        return std::make_pair(0, 0);
    }

    std::string line;
    int rows = 0;
    int maxCols = 0;
    while (std::getline(inputFile, line))
```

```
    {
        std::vector<int> rowData;
        std::stringstream ss(line);
        int value;
        while (ss >> value)
        {
            rowData.push_back(value);
        }
        newMatrix.push_back(std::move(rowData));
        int currentCols = static_cast<int>(newMatrix.back().size());
        if (currentCols > maxCols)
        {
            maxCols = currentCols;
        }
        rows++;
    }

    inputFile.close();
    matrix = std::move(newMatrix);

    return std::make_pair(rows, maxCols);
}

void playLife(vector<vector<int>> &matrix, int lines, int columns)
{
    initscr();              // Инициализация ncurses
    curs_set(0);            // Скрываем курсор
    cbreak();               // Режим постоянного ввода
    noecho();               // Отключаем вывод введенных символов
    nodelay(stdscr, true); // Отключаем блокировку getch()

    bool running = true;

    char ch;
    int sleep = 200000;

    while (running)
    {
        clear();
        fieldDisplay(matrix, lines, columns);
        updateMatrix(matrix, lines, columns);

        ch = getch(); // Считываем нажатую клавишу

        switch (ch)
        {
        case '+': // Стрелка влево
            sleep += 200000;
            break;

        case '=': // Стрелка влево
```

```
            sleep += 200000;
            break;

        case '-': // Стрелка вправо

            if (sleep - 200000 > 0)
            {
                sleep -= 20000;
            }
            break;

        case 'q':            // Клавиша "q"
            running = false; // Выставляем флаг выхода
            break;

        default:
            break;
        }

        refresh();     // Обновляем экран
        usleep(sleep); // Задержка
    }

    endwin(); // Завершаем работу с ncurses
}

void updateMatrix(vector<vector<int>> &matrix, int lines, int columns)
{
    vector<vector<int>> newMatrix(lines, vector<int>(columns, 0)); //
Инициализируем newMatrix

    for (int i = 0; i < lines; i++)
    {
        for (int j = 0; j < columns; j++)
        {
            int numbeLivingCells = cellCounting(matrix, i, j, lines, columns);

            if (matrix[i][j] == 1)
            { // Если клетка живая
                if (numbeLivingCells < 2 || numbeLivingCells > 3)
                { // Умирает от одиночества или перенаселения
                    newMatrix[i][j] = 0;
                }
                else
                { // Остается живой
                    newMatrix[i][j] = 1;
                }
            }
            else if (matrix[i][j] == 0)
            { // Если клетка мертвая
                if (numbeLivingCells == 3)
```

```
                { // Оживает
                    newMatrix[i][j] = 1;
                }
            }
            else
            {
                continue;
            }
        }
    }

    matrix = move(newMatrix); // Заменяем содержимое matrix на newMatrix
}

int cellCounting(vector<vector<int>> &matrix, int i, int j, int lines, int
columns)
{
    int sum = 0;

    // Проверяем соседей клетки (i, j)
    for (int x = max(0, i - 1); x <= min(i + 1, lines - 1); x++)
    {
        for (int y = max(0, j - 1); y <= min(j + 1, columns - 1); y++)
        {
            if (!(x == i && y == j))
            { // Исключаем саму клетку (i, j)
                sum += matrix[x][y];
            }
        }
    }

    return sum;
}

void fieldDisplay(const std::vector<std::vector<int>> &matrix, int lines, int
columns)
{
    // clear();

    // Рисуем границу вокруг игрового поля
    for (int i = 0; i < lines + 2; i++)
    {
        for (int j = 0; j < columns + 2; j++)
        {
            if (i == 0 || i == lines + 1 || j == 0 || j == columns + 1)
            {
                mvaddch(i, j, ACS_BLOCK); // Отображаем стенку
            }
            else
            {
                if (matrix[i - 1][j - 1] == 1)
```

```
                {
                    mvaddch(i, j, '@'); // Отображаем живую клетку
                }
                else
                {
                    mvaddch(i, j, ' '); // Отображаем мертвую клетку
                }
            }
        }
    }

    refresh();
}
```

Main.cpp

```
#include "header.h"

using namespace std;


void task3() // X ИГРА В ЖИЗНЬ
{
    char action;
    int sizeMatrix;
    pair<int, int> lineRow;

    int line = 101;
    int column = 151;

    vector<vector<int>> playingField(line, vector<int>(column, 0));

    cout << "1-random 2-train q = exit" << endl;
    cout << "Enter action: ";
    cin >> action;

    switch (action)
    {
    case '1':
        generatePlayingField(playingField, line, column);

        playLife(playingField, line, column);
        break;

    case '2':
        lineRow = readingPlayingField(playingField);

        // cout << playingField << endl;
        // cout << lineRow.first << " " << lineRow.second;
        playLife(playingField, lineRow.first, lineRow.second);
        break;
```

```
    case 'q':
        return;
    default:
        cout << "unknown team\n";
    }
}
#include "header.h"

using namespace std;


void task3() // X ИГРА В ЖИЗНЬ
{
    char action;
    int sizeMatrix;
    pair<int, int> lineRow;

    int line = 101;
    int column = 151;

    vector<vector<int>> playingField(line, vector<int>(column, 0));

    cout << "1-random 2-train q = exit" << endl;
    cout << "Enter action: ";
    cin >> action;

    switch (action)
    {
    case '1':
        generatePlayingField(playingField, line, column);

        playLife(playingField, line, column);
        break;

    case '2':
        lineRow = readingPlayingField(playingField);

        // cout << playingField << endl;
        // cout << lineRow.first << " " << lineRow.second;
        playLife(playingField, lineRow.first, lineRow.second);
        break;

    case 'q':
        return;
    default:
        cout << "unknown team\n";
    }
}
```

GO:

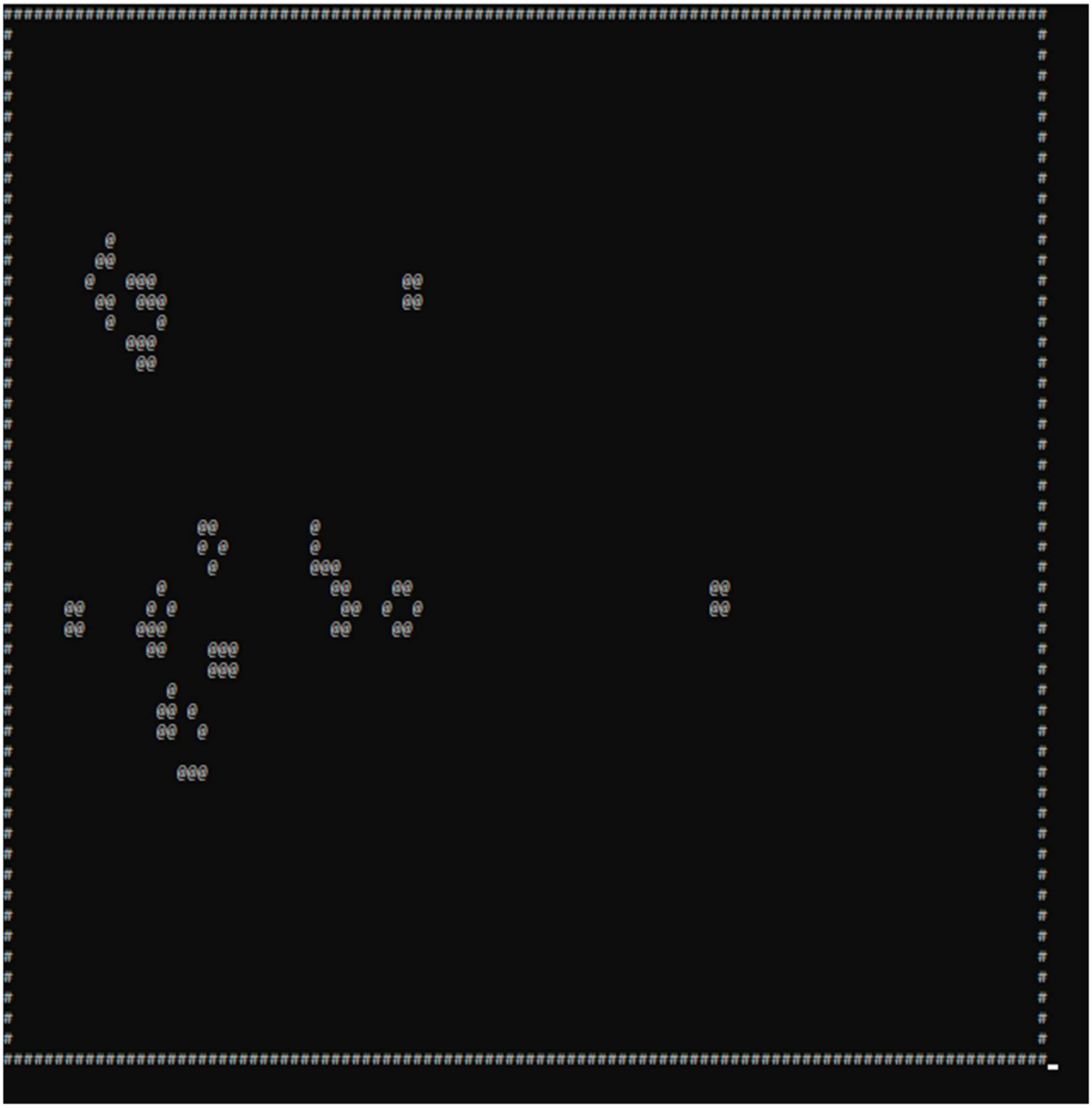
```go
package main

import (
    "fmt"
    "math/rand"
    "time"
)

func GetRandomNumber(min, max int) int {
    rand.Seed(time.Now().UnixNano())
    return min + rand.Intn(max-min+1)
}

func task1() {
    fmt.Println("Task 1")
    quantityLine, quantityColumns := 5, 6
    maxSrednee, numberLine := 0, 0

    randomMatrix := make([][]int, quantityLine)
    for i := range randomMatrix {
        randomMatrix[i] = make([]int, quantityColumns)
        for j := range randomMatrix[i] {
            randomMatrix[i][j] = GetRandomNumber(0, 100)
        }
    }

    for _, i := range randomMatrix {
        for _, j := range i {
            fmt.Printf("%4d", j)
        }
        fmt.Println()
    }

    for i := 0; i < quantityLine; i++ {
        sredneeArithmetic := 0
        for j := 0; j < quantityColumns; j++ {
            sredneeArithmetic += randomMatrix[i][j]
        }
        if maxSrednee < sredneeArithmetic {
            maxSrednee = sredneeArithmetic
            numberLine = i
        }
        fmt.Printf("Номер строки %d среднее арифметическое = %.2f\n", i,
float64(sredneeArithmetic)/float64(quantityColumns))
    }
    fmt.Printf("Наибольшее среднее арифметическое в строке %d\n", numberLine)
}

func task2() {
```

```
    fmt.Println("Task 2")

    sizeSquareMatrix := 15
    squareMatrix := make([][]int, sizeSquareMatrix)
    for i := range squareMatrix {
        squareMatrix[i] = make([]int, sizeSquareMatrix)
        for j := range squareMatrix[i] {
            squareMatrix[i][j] = GetRandomNumber(-100, 100)
            fmt.Printf("%4d", squareMatrix[i][j])
        }
        fmt.Println()
    }

    for i := 0; i < sizeSquareMatrix; i++ {
        columns := make([]int, sizeSquareMatrix)
        for j := 0; j < sizeSquareMatrix; j++ {
            columns[j] = squareMatrix[j][i]
        }
        if i%2 == 0 {
            Sort(columns, true)
        } else {
            Sort(columns, false)
        }
        for j := 0; j < sizeSquareMatrix; j++ {
            squareMatrix[j][i] = columns[j]
        }
    }

    fmt.Println()
    for _, i := range squareMatrix {
        for _, j := range i {
            fmt.Printf("%4d", j)
        }
        fmt.Println()
    }

    sumRight, sumLeft := 0, 0
    for i := 0; i < len(squareMatrix); i++ {
        for j := i; j < len(squareMatrix); j++ {
            if squareMatrix[i][j] > 0 {
                sumRight++
            }
        }
    }

    for i := len(squareMatrix) - 1; i >= 0; i-- {
        for j := i - 1; j >= 0; j-- {
            if squareMatrix[i][j] > 0 {
                sumLeft++
            }
        }
```

```
    }

    fmt.Printf("Количество положительных элементов правой половины =
%d\tколичество положительных элементов левой половины = %d\n", sumRight, sumLeft)
    if sumRight > sumLeft {
        fmt.Println("Правая половина содержит больше положительных элементов")
    } else {
        fmt.Println("Левая половина содержит больше положительных элементов")
    }
}

func Sort(arr []int, asc bool) {
    if asc {
        for i := 0; i < len(arr)-1; i++ {
            for j := i + 1; j < len(arr); j++ {
                if arr[i] > arr[j] {
                    arr[i], arr[j] = arr[j], arr[i]
                }
            }
        }
    } else {
        for i := 0; i < len(arr)-1; i++ {
            for j := i + 1; j < len(arr); j++ {
                if arr[i] < arr[j] {
                    arr[i], arr[j] = arr[j], arr[i]
                }
            }
        }
    }
}

func main() {
    task1()
    fmt.Println()
    task2()
}
```

Результат работы программы

```
Task 1
 100  73  24   9  19   6
  60  88  26  20  99  84
  16  53  84  82  81  40
  80  53  60   4   4  63
  72  79  82  97  75  97
Номер строки 0 среднее арифметическое = 38.5
Номер строки 1 среднее арифметическое = 62.8333
Номер строки 2 среднее арифметическое = 59.3333
Номер строки 3 среднее арифметическое = 44
Номер строки 4 среднее арифметическое = 83.6667
Наибольшее среднеее арифметическое в строке 4

Task 2
 -54  56 -73  48  69 -94  74 -53 -84  59 -77  12 -72 -21  -8
 -59  90 -20  73   4  17  47 -65  57  44  57 -52  31  81  56
  57 -26  82  80  75  45  41 -24 -45  -6  59  30 -31  38 -18
   1 -25 -10 -36   6  -6  55  97 -84 -43  36  79  19  27 -45
 -57   9  61 -21 -89 -10  48 -53 -24 -67  79 -35 -17  54  88
  79  28 -39 -43  71  53 -66 -61  36  18 -88  72  49  15  44
  62  30  -6  -8  -2 -91 -51  31  42  92 -80 -52  28 -83  28
 -67 -71 -87   6 -33  63 -47 -71  -5 -37 -77 100  48  30  85
  67 -92  10 -69 -96 -52 -15  91  79 -76  47 -65 -88  90  99
 -46  17  16 -43  67  20  23  93 -24  92 -99  62 -70  77 -27
  85  83  34  24 -26  42  13  66  80  51 -53  76 -18  10  96
   6  42  -2  69 -95 -76 -49  15 -94  88 -93  97  49  84  39
 -94  50 -31 -94 -49  68  12 -70  -8  88  -6  20 -99 -15  99
 -52  15  29  91  39  32 -79   9  92  71 -33  64 -23 -12  72
  -8 -33 -69  56  97 -20 -44 -42  66 -38 -72  -4  42  -4 -37

 -94  90 -87  91 -96  68 -79  97 -94  92 -99 100 -99  90 -45
 -67  83 -73  80 -95  63 -66  93 -84  92 -93  97 -88  84 -37
 -59  56 -69  73 -89  53 -51  91 -84  88 -88  79 -72  81 -27
 -57  50 -39  69 -49  45 -49  66 -45  88 -80  76 -70  77 -18
 -54  42 -31  56 -33  42 -47  31 -24  71 -77  72 -31  54  -8
 -52  30 -20  48 -26  32 -44  15 -24  59 -77  64 -23  38  28
 -46  28 -10  24  -2  20 -15   9  -8  51 -72  62 -18  30  39
  -8  17  -6   6   4  17  12 -24  -5  44 -53  30 -17  27  44
   1  15  -2  -8   6  -6  13 -42  36  18 -33  20  19  15  56
   6   9  10 -21  39 -10  23 -53  42  -6  -6  12  28  10  72
  57 -25  16 -36  67 -20  41 -53  57 -37  36  -4  31  -4  85
  62 -26  29 -43  69 -52  47 -61  66 -38  47 -35  42 -12  88
  67 -33  34 -43  71 -76  48 -65  79 -43  57 -52  48 -15  96
  79 -71  61 -69  75 -91  55 -70  80 -67  59 -52  49 -21  99
  85 -92  82 -94  97 -94  74 -71  92 -76  79 -65  49 -83  99
Количество положительных элементов правой половины = 65  количество положительных элементов левой половины = 62
Правая половина содержит больше положительных элементов
```



Задание 2:

C++:

```cpp
#include <iostream>
#include <vector>
#include <string>
#include <windows.h>
#include <random>
#include <iomanip> // Добавляем заголовочный файл для использования setw, setfill и hex
using namespace std;

void generate_master_key(vector<unsigned char>& MASTER_KEY, mt19937_64& mt)
```

```
{
    uniform_int_distribution<int> letters('a', 'z');
    uniform_int_distribution<int> numbers(0, 9);
    for (int i = 1; i <= 8; i++) {
        MASTER_KEY.push_back(letters(mt));
        MASTER_KEY.push_back('0' + numbers(mt));
    }
}

void shift_rows(vector<unsigned char>& line)
{
    vector<unsigned char> v(line.size());
    for (int i = 1; i < line.size(); i++)
    {
        v[i - 1] = line[i];        //сдвиг элементов влево
    }
    v[line.size() - 1] = line[0]; //последний элемент становится первым

    line = v;
}

void inv_shift_rows(vector<unsigned char>& line)
{
    vector<unsigned char> v(line.size());

    for (int i = 0; i < line.size(); i++) {
        v[i] = line[(i + (i % 4) * (i % 4)) % line.size()];
    }
    line = v;
}

void sub_bytes(vector<unsigned char>& line)
{ //замена на соответствующие байты из S-BOX

    vector<unsigned char> 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
    };

    vector<unsigned char> v;
    for (auto i : line) {
        v.push_back(Sbox[i]);
    }
    line = v;
}

void inv_sub_bytes(vector<unsigned char>& line) { //замена на соответствующие
байты из обратной S-BOX

    vector<unsigned char> inv_sbox_data = {
        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,
        0x47, 0xf1, 0x1a, 0x71, 0x1d, 0x29, 0xc5, 0x89, 0x6f, 0xb7, 0x62, 0x0e,
0xaa, 0x18, 0xbe, 0x1b,
```

```
        0xfc, 0x56, 0x3e, 0x4b, 0xc6, 0xd2, 0x79, 0x20, 0x9a, 0xdb, 0xc0, 0xfe,
0x78, 0xcd, 0x5a, 0xf4,
        0x1f, 0xdd, 0xa8, 0x33, 0x88, 0x07, 0xc7, 0x31, 0xb1, 0x12, 0x10, 0x59,
0x27, 0x80, 0xec, 0x5f,
        0x60, 0x51, 0x7f, 0xa9, 0x19, 0xb5, 0x4a, 0x0d, 0x2d, 0xe5, 0x7a, 0x9f,
0x93, 0xc9, 0x9c, 0xef,
        0xa0, 0xe0, 0x3b, 0x4d, 0xae, 0x2a, 0xf5, 0xb0, 0xc8, 0xeb, 0xbb, 0x3c,
0x83, 0x53, 0x99, 0x61,
        0x17, 0x2b, 0x04, 0x7e, 0xba, 0x77, 0xd6, 0x26, 0xe1, 0x69, 0x14, 0x63,
0x55, 0x21, 0x0c, 0x7d
    };

    vector<unsigned char> v;
    for (auto i : line) {
        v.push_back(inv_sbox_data[i]);
    }
    line = v;
}

vector<unsigned char> add_round_key(vector<unsigned char>& FIRST, vector<unsigned
char>& SECOND) {
    vector<unsigned char> v;
    for (int i = 0; i < 4; ++i) {
        v.push_back(FIRST[i] ^ SECOND[i]);
    }
    return v;
}

void key_expansion(vector<unsigned char>& MASTER_KEY, vector<vector<unsigned
char>>& ROUND_KEYS) // расширение ключа
{

    const vector<unsigned char> Rcon = {
        0x00, 0x00, 0x00, 0x00,
        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
    };

    vector<unsigned char> v;
    int i = 0;
    int Nk = 4; //изначальные ключи из мастер_кей
    int Nb = 4; //количество столбцов в блоке данных
    int Nr = 10; //количество раундов
```

```
    ROUND_KEYS.resize(Nb * (Nr + 1), vector<unsigned char>(4)); //размер 44 по 4
строчки

    while (i < Nk) { //копирует первые 4 ключа из MASTER_KEY в ROUND_KEYS
        v = { MASTER_KEY[4 * i], MASTER_KEY[4 * i + 1], MASTER_KEY[4 * i + 2],
MASTER_KEY[4 * i + 3] };
        ROUND_KEYS[i] = v;
        i++;
    }

    i = Nk;
    while (i < (Nb * (Nr + 1))) { //расширяем ключ до 11
        v = ROUND_KEYS[i - 1]; //сохраняем предыдущий ключ
        if (i % Nk == 4) {
            shift_rows(v); //сдвиг байтов влево
            sub_bytes(v);  //замена на соответствующие байты из S-BOX

            for (int k = 0; k < v.size(); ++k) {
                v[k] = v[k] ^ Rcon[i / Nk]; //функция xor
            }
        } else if (Nk > 6 && (i % Nk == 4)) { //если индекс кратен Nk еще
поменять
            sub_bytes(v);
        }
        //новый ключ - XOR предыдущего из ROUND_KEYS[i - Nk] и полученной в
предыдущих
        for (int j = 0; j < v.size(); ++j) {
            ROUND_KEYS[i][j] = ROUND_KEYS[i - Nk][j] ^ v[j];
        }
        i++;
    }
}

unsigned char galois_multiply(unsigned char a, unsigned char b) // поле Галуа
{
    unsigned char result = 0; // Инициализация переменной для хранения результата
умножения
    unsigned char carry; // Переменная для хранения переноса

    for (int i = 0; i < 8; ++i) { // Цикл по битам переменной b (предполагается,
что char содержит 8 бит)
        if (b & 1) { // Если младший бит b равен 1
            result ^= a; // Выполняем XOR текущего результата с переменной a
        }

        carry = a & 0x80; // Проверяем старший бит переменной a перед сдвигом
        a <<= 1; // Сдвигаем переменную a влево на 1 бит

        if (carry) { // Если старший бит a был 1
            a ^= 0x1b; // Выполняем XOR переменной a с модулем AES
```

```
        }

        b >>= 1; // Сдвигаем переменную b вправо на 1 бит
    }

    return result; // Возвращаем результат умножения
}

vector<unsigned char> generate_mult_by_2()
{
    vector<unsigned char> mult_by_2(256);
    for (int i = 0; i < 256; ++i) {
        mult_by_2[i] = galois_multiply(i, 2);
    }
    return mult_by_2;
}

// Генерация таблицы mult_by_3
vector<unsigned char> generate_mult_by_3()
{
    vector<unsigned char> mult_by_3(256);
    for (int i = 0; i < 256; ++i) {
        mult_by_3[i] = galois_multiply(i, 3);
    }
    return mult_by_3;
}

vector<unsigned char> generate_mult_by_14()
{
    vector<unsigned char> mult_by_14(256);
    for (int i = 0; i < 256; ++i) {
        mult_by_14[i] = galois_multiply(i, 14);
    }
    return mult_by_14;
}

vector<unsigned char> generate_mult_by_9()
{
    vector<unsigned char> mult_by_9(256);
    for (int i = 0; i < 256; ++i) {
        mult_by_9[i] = galois_multiply(i, 9);
    }
    return mult_by_9;
}

vector<unsigned char> generate_mult_by_13()
{
    vector<unsigned char> mult_by_13(256);
    for (int i = 0; i < 256; ++i) {
        mult_by_13[i] = galois_multiply(i, 13);
    }
```

```
    return mult_by_13;
}

vector<unsigned char> generate_mult_by_11()
{
    vector<unsigned char> mult_by_11(256);
    for (int i = 0; i < 256; ++i) {
        mult_by_11[i] = galois_multiply(i, 11);
    }
    return mult_by_11;
}

void mix_column(vector<unsigned char>& line)
{
    vector<unsigned char> mult_by_2 = generate_mult_by_2(); //операции умножения
на 2 и 3 к столбцам матрицы данных

    vector<unsigned char> mult_by_3 = generate_mult_by_3();

    vector<unsigned char> v;
    v.push_back(mult_by_2[line[0]] ^ mult_by_3[line[1]] ^ line[2] ^ line[3]);
    v.push_back(mult_by_2[line[1]] ^ mult_by_3[line[2]] ^ line[0] ^ line[3]);
    v.push_back(mult_by_2[line[2]] ^ mult_by_3[line[3]] ^ line[0] ^ line[1]);
    v.push_back(mult_by_2[line[3]] ^ mult_by_3[line[0]] ^ line[1] ^ line[2]);
    line = v;
}

void inv_mix_columns(vector<unsigned char>& line)
{
    vector<unsigned char> mult_by_14 = generate_mult_by_14();
    vector<unsigned char> mult_by_9 = generate_mult_by_9();
    vector<unsigned char> mult_by_13 = generate_mult_by_13();
    vector<unsigned char> mult_by_11 = generate_mult_by_11();

    vector<unsigned char> v;
    v.push_back(mult_by_14[line[0]] ^ mult_by_9[line[1]] ^ mult_by_13[line[2]] ^
mult_by_11[line[3]]);
    v.push_back(mult_by_14[line[1]] ^ mult_by_9[line[2]] ^ mult_by_13[line[3]] ^
mult_by_11[line[0]]);
    v.push_back(mult_by_14[line[2]] ^ mult_by_9[line[3]] ^ mult_by_13[line[0]] ^
mult_by_11[line[1]]);
    v.push_back(mult_by_14[line[3]] ^ mult_by_9[line[0]] ^ mult_by_13[line[1]] ^
mult_by_11[line[2]]);
    line = v;
}

void block_generate(string& text, vector<vector<vector<unsigned char>>>& BLOCK) {
//генерируем изначальное представление сообщения

    while (text.size() % 16 != 0) { // добавление пробелов
        text += ' ';
```

```
    }

    vector<vector<vector<unsigned char>>> v; //временный BLOCK
    vector<vector<unsigned char>> sixteen(4, vector<unsigned char>(4)); //
матрица 4 на 4

    for (int i = 0; i < text.size(); ++i) {

        int a = (i % 16) % 4;       //по блокам выбирается строка
        int b = (i % 16) / 4;       //и столбец
        sixteen[a][b] = text[i];

        if ((i + 1) % 16 == 0) {   //если блок заполнился
            v.push_back(sixteen);
            sixteen = vector<vector<unsigned char>>(4, vector<unsigned char>(4));
        }
    }
    BLOCK = v;
}

vector<vector<unsigned char>> CIPHER(vector<vector<unsigned char>>& BLOCK,
vector<vector<unsigned char>>& ROUND_KEYS, vector<unsigned char> MASTER_KEY)
{
    vector<vector<unsigned char>> v(4, vector<unsigned char>(4, 0)); //первый
addroundkey
    for (int i = 0; i < 4; ++i) {
        v[i] = add_round_key(BLOCK[i], ROUND_KEYS[i]);
    }
    //ROUNDS_KEYS[0] - забрали
    for (int i = 1; i <= 9; i++) { // REPEAT 10 - 1
        for (int j = 0; j <= 3; j++) { //обрабатываем каждую строку BLOCK
            sub_bytes(v[j]);
            shift_rows(v[j]);
            mix_column(v[j]);
            v[j] = add_round_key(v[j], ROUND_KEYS[i]);
        }
    }
    //ROUNDS_KEYS[0] - ROUNDS_KEYS[9] - забрали

    for (int j = 0; j <= 3; j++) {
        sub_bytes(v[j]);
        shift_rows(v[j]);
        v[j] = add_round_key(v[j], ROUND_KEYS[10]);
    }//ROUNDS_KEYS[10] - забрали

    return v;
}

vector<vector<unsigned char>> DECIPHER(vector<vector<unsigned char>>& BLOCK,
vector<vector<unsigned char>>& ROUND_KEYS, vector<unsigned char> MASTER_KEY) {
        // Расширение ключа
```

```
    vector<vector<unsigned char>> round_keys(44, vector<unsigned char>(4));
    key_expansion(MASTER_KEY, round_keys);

    vector<vector<unsigned char>> v(4, vector<unsigned char>(4, 0));


    // Первый этап - AddRoundKey с последним раундовым ключом
    for (int j = 0; j <= 3; j++) {
        v[j] = add_round_key(BLOCK[j], ROUND_KEYS[10]);
    }

    // Циклический этап - InvShiftRows, InvSubBytes, AddRoundKey, InvMixColumns
    for (int i = 9; i >= 1; i--) {
        for (int j = 0; j <= 3; j++) {
            v[j] = add_round_key(v[j], ROUND_KEYS[i]);
            inv_mix_columns(v[j]);
            inv_shift_rows(v[j]);
            inv_sub_bytes(v[j]);
        }
    }

    // Последний этап - InvShiftRows, InvSubBytes, AddRoundKey
    for (int j = 0; j <= 3; j++) {
        v[j] = add_round_key(v[j], ROUND_KEYS[0]);
    }
    for (int j = 0; j <= 3; j++) {
        inv_shift_rows(v[j]);
        inv_sub_bytes(v[j]);
    }

    return v;
}

int main() {
    SetConsoleCP(1251);        //для корректного ввода и вывода русских символов
в консоли
    SetConsoleOutputCP(1251);

    mt19937_64 mt(random_device{}());
    string text;
    cout << "Input text for cipher >> ";
    getline(cin, text);

    vector<vector<vector<unsigned char>>> BLOCK;   //генерация матрицы для работы
    block_generate(text, BLOCK);
    cout << "-------------------------------------------";
    cout << "\nThe encryption block: \n";
    for (auto i : BLOCK) {
        for (auto j : i) {
            for (auto k : j) {
                cout << setw(4) << k << " ";
```

```
            }
            cout << endl;
        }
    }
    cout << "\n-----------------------------------------" << endl;

    vector<unsigned char> MASTER_KEY;
    generate_master_key(MASTER_KEY, mt);
    cout << "128-bit master key: "; //генерируем рандомный ключ с парами
буква+цифра размером 16 байт
    for (auto i : MASTER_KEY) {
        cout << i;
    }
    cout << "\n-----------------------------------------\n";

    vector<vector<unsigned char>> ROUND_KEYS;
    key_expansion(MASTER_KEY, ROUND_KEYS); //генерация ключей
    cout << "Generated keys: \n";
    for (auto i : ROUND_KEYS) {
        for (auto j : i) {
            cout << hex << setw(4) << static_cast<int>(j) << " ";

        }
        cout << endl;
    }
    cout << "\n-----------------------------------------" << endl;


    vector<vector<unsigned char>> PREV(4, vector<unsigned char>(4, 0));
    vector<vector<unsigned char>> DEFOLT = PREV;
    vector<vector<unsigned char>> TEK;
    vector<vector<vector<unsigned char>>> ECRYPT;

    for (int i = 0; i < BLOCK.size(); i++) {

        vector<vector<unsigned char>> res(4, vector<unsigned char>(4, 0));
        vector<vector<unsigned char>> B = BLOCK[i];
        TEK = CIPHER(PREV, ROUND_KEYS, MASTER_KEY);

        for (int i = 0; i < 4; ++i) {
            for (int j = 0; j < 4; ++j) {
                res[i][j] = (TEK[i][j] ^ B[i][j]);

            }
        }
        ECRYPT.push_back(res);

        PREV = TEK;
    }

    cout << "The final cipher after encryphion:\n" << endl;
```

```
    for (auto t : ECRYPT) {
        for (auto i : t) {
            for (auto j : i) {
                cout << setw(4) << j << " ";
            }
            cout << endl;
        }
    }

    cout << "\n-----------------------------------------------\n";

    PREV = DEFOLT;
    vector<vector<vector<unsigned char>>> DECRYPT;

    for (int i = 0; i < ECRYPT.size(); ++i) {
        vector<vector<unsigned char>> res(4, vector<unsigned char>(4, 0));
        vector<vector<unsigned char>> B = ECRYPT[i];

        TEK = CIPHER(PREV, ROUND_KEYS, MASTER_KEY);

        for (int i = 0; i < 4; ++i) {
            for (int j = 0; j < 4; ++j) {
                res[i][j] = (B[i][j] ^ TEK[i][j]);
            }
        }
        DECRYPT.push_back(res);
        PREV = TEK;
    }

    vector<vector<vector<unsigned char>>> DECRYPT_1;

    for (int i = 0; i < ECRYPT.size(); ++i) {
        vector<vector<unsigned char>> res(4, vector<unsigned char>(4, 0));
        vector<vector<unsigned char>> B = ECRYPT[i];

        TEK = DECIPHER(B, ROUND_KEYS, MASTER_KEY);

        for (int i = 0; i < 4; ++i) {
            for (int j = 0; j < 4; ++j) {
                res[i][j] = TEK[i][j];
            }
        }
        DECRYPT_1.push_back(res);
    }

    cout << "Message after decryption:\n" << endl;
    for (int i = 0; i < DECRYPT.size(); i++) {
        vector<vector<unsigned char>>& decrypt = DECRYPT[i];
        for (int j = 0; j < decrypt.size(); j++) {
            for (int k = 0; k < decrypt[j].size(); k++) {
                cout << decrypt[k][j];
```

```
            }
        }
    }
    cout << "\n-----------------------------------------------\n" << endl;

    return 0;
}
```

GO:

```
package main

import (
    "fmt"
    "math/rand"
    "time"
)

func generateMasterKey(masterKey *[]byte, mt *rand.Rand) {
    letters := "abcdefghijklmnopqrstuvwxyz"
    numbers := "0123456789"
    for i := 0; i < 8; i++ {
        *masterKey = append(*masterKey, byte(letters[mt.Intn(len(letters))]))
        *masterKey = append(*masterKey, byte(numbers[mt.Intn(len(numbers))]))
    }
}

func shiftRows(line []byte) {
    v := make([]byte, len(line))
    for i := 1; i < len(line); i++ {
        v[i-1] = line[i]
    }
    v[len(line)-1] = line[0]
    copy(line, v)
}

func invShiftRows(line []byte) {
    v := make([]byte, len(line))
    for i := range line {
        v[i] = line[(i+i%(i%4)*i%4)%len(line)]
    }
    copy(line, v)
}

func subBytes(line []byte) {
    sbox := [256]byte{
        0x63, 0x7c, 0x77, 0x7b, 0xf2, 0x6b, 0x6f, 0xc5, 0x30, 0x01, 0x67, 0x2b,
0xfe, 0xd7, 0xab, 0x76,
        // ...
    }
    for i, b := range line {
```

```
        line[i] = sbox[b]
    }
}

func invSubBytes(line []byte) {
    invSbox := [256]byte{
        0x52, 0x09, 0x6a, 0xd5, 0x30, 0x36, 0xa5, 0x38, 0xbf, 0x40, 0xa3, 0x9e,
0x81, 0xf3, 0xd7, 0xfb,
        // ...
    }
    for i, b := range line {
        line[i] = invSbox[b]
    }
}

func addRoundKey(first, second []byte) []byte {
    v := make([]byte, len(first))
    for i := range first {
        v[i] = first[i] ^ second[i]
    }
    return v
}

func keyExpansion(masterKey []byte, roundKeys *[][]byte) {
    Nk, Nb, Nr := 4, 4, 10
    *roundKeys = make([][]byte, Nb*(Nr+1))
    for i := range *roundKeys {
        (*roundKeys)[i] = make([]byte, 4)
    }

    i := 0
    for ; i < Nk; i++ {
        copy((*roundKeys)[i], masterKey[4*i:4*i+4])
    }

    rcon := []byte{
        0x00, 0x00, 0x00, 0x00,
        0x01, 0x00, 0x00, 0x00,
        // ...
    }

    for ; i < Nb*(Nr+1); i++ {
        tmp := make([]byte, 4)
        copy(tmp, (*roundKeys)[i-1])
        if i%Nk == 0 {
            shiftRows(tmp)
            subBytes(tmp)
            for j := range tmp {
                tmp[j] ^= rcon[i/Nk*4+j]
            }
        } else if Nk > 6 && i%Nk == 4 {
```

```
            subBytes(tmp)
        }
        for j := range tmp {
            (*roundKeys)[i][j] = (*roundKeys)[i-Nk][j] ^ tmp[j]
        }
    }
}

func galoisMultiply(a, b byte) byte {
    var result byte
    var carry byte

    for i := 0; i < 8; i++ {
        if b&1 == 1 {
            result ^= a
        }

        carry = a & 0x80
        a <<= 1
        if carry != 0 {
            a ^= 0x1b
        }

        b >>= 1
    }

    return result
}

func generateMultBy2() []byte {
    v := make([]byte, 256)
    for i := range v {
        v[i] = galoisMultiply(byte(i), 2)
    }
    return v
}

func generateMultBy3() []byte {
    v := make([]byte, 256)
    for i := range v {
        v[i] = galoisMultiply(byte(i), 3)
    }
    return v
}

func generateMultBy14() []byte {
    v := make([]byte, 256)
    for i := range v {
        v[i] = galoisMultiply(byte(i), 14)
    }
    return v
```

```
}

func generateMultBy9() []byte {
    v := make([]byte, 256)
    for i := range v {
        v[i] = galoisMultiply(byte(i), 9)
    }
    return v
}

func generateMultBy13() []byte {
    v := make([]byte, 256)
    for i := range v {
        v[i] = galoisMultiply(byte(i), 13)
    }
    return v
}

func generateMultBy11() []byte {
    v := make([]byte, 256)
    for i := range v {
        v[i] = galoisMultiply(byte(i), 11)
    }
    return v
}

func mixColumns(line []byte) {
    multBy2 := generateMultBy2()
    multBy3 := generateMultBy3()

    v := make([]byte, 4)
    v[0] = multBy2[line[0]] ^ multBy3[line[1]] ^ line[2] ^ line[3]
    v[1] = multBy2[line[1]] ^ multBy3[line[2]] ^ line[0] ^ line[3]
    v[2] = multBy2[line[2]] ^ multBy3[line[3]] ^ line[0] ^ line[1]
    v[3] = multBy2[line[3]] ^ multBy3[line[0]] ^ line[1] ^ line[2]
    copy(line, v)
}

func invMixColumns(line []byte) {
    multBy14 := generateMultBy14()
    multBy9 := generateMultBy9()
    multBy13 := generateMultBy13()
    multBy11 := generateMultBy11()

    v := make([]byte, 4)
    v[0] = multBy14[line[0]] ^ multBy9[line[1]] ^ multBy13[line[2]] ^
multBy11[line[3]]
    v[1] = multBy14[line[1]] ^ multBy9[line[2]] ^ multBy13[line[3]] ^
multBy11[line[0]]
    v[2] = multBy14[line[2]] ^ multBy9[line[3]] ^ multBy13[line[0]] ^
multBy11[line[1]]
```

```
    v[3] = multBy14[line[3]] ^ multBy9[line[0]] ^ multBy13[line[1]] ^
multBy11[line[2]]
    copy(line, v)
}

func blockGenerate(text string, block *[][][]byte) {
    for len(textfunc blockGenerate(text string, block *[][][]byte) {
    for len(text) % 16 != 0 {
        text += " " // Добавляем пробелы, если длина не кратна 16
    }

    *block = make([][][]byte, 0, len(text)/16)
    sixteen := make([][]byte, 4)
    for i := range sixteen {
        sixteen[i] = make([]byte, 4)
    }

    for i := range text {
        a := i % 16 % 4
        b := i % 16 / 4
        sixteen[a][b] = text[i]

        if (i+1)%16 == 0 {
            *block = append(*block, sixteen)
            sixteen = make([][]byte, 4)
            for i := range sixteen {
                sixteen[i] = make([]byte, 4)
            }
        }
    }
}

func cipher(block, roundKeys [][]byte, masterKey []byte) [][]byte {
    v := make([][]byte, 4)
    for i := range v {
        v[i] = addRoundKey(block[i], roundKeys[i])
    }

    for i := 1; i <= 9; i++ {
        for j := 0; j <= 3; j++ {
            subBytes(v[j])
            shiftRows(v[j])
            mixColumns(v[j])
            v[j] = addRoundKey(v[j], roundKeys[i])
        }
    }

    for j := 0; j <= 3; j++ {
        subBytes(v[j])
        shiftRows(v[j])
        v[j] = addRoundKey(v[j], roundKeys[10])
```

```
    }

    return v
}

func decipher(block, roundKeys [][]byte, masterKey []byte) [][]byte {
    roundKeys = make([][]byte, 44)
    keyExpansion(masterKey, &roundKeys)

    v := make([][]byte, 4)
    for j := 0; j <= 3; j++ {
        v[j] = addRoundKey(block[j], roundKeys[10])
    }

    for i := 9; i >= 1; i-- {
        for j := 0; j <= 3; j++ {
            v[j] = addRoundKey(v[j], roundKeys[i])
            invMixColumns(v[j])
            invShiftRows(v[j])
            invSubBytes(v[j])
        }
    }

    for j := 0; j <= 3; j++ {
        v[j] = addRoundKey(v[j], roundKeys[0])
        invShiftRows(v[j])
        invSubBytes(v[j])
    }

    return v
}

func main() {
    rand.Seed(time.Now().UnixNano())
    mt := rand.New(rand.NewSource(time.Now().UnixNano()))

    var text string
    fmt.Print("Input text for cipher >> ")
    fmt.Scanln(&text)

    var block [][][]byte
    blockGenerate(text, &block)
    fmt.Println("-------------------------------------------")
    fmt.Println("The encryption block:")
    for _, b := range block {
        for _, line := range b {
            for _, c := range line {
                fmt.Printf("%4d ", c)
            }
            fmt.Println()
        }
```

```
        fmt.Println()
    }
    fmt.Println("-----------------------------------------------")

    var masterKey []byte
    generateMasterKey(&masterKey, mt)
    fmt.Print("128-bit master key: ")
    for _, b := range masterKey {
        fmt.Printf("%c", b)
    }
    fmt.Println("\n-----------------------------------------------")

    var roundKeys [][]byte
    keyExpansion(masterKey, &roundKeys)
    fmt.Println("Generated keys:")
    for _, line := range roundKeys {
        for _, b := range line {
            fmt.Printf("%04x ", b)
        }
        fmt.Println()
    }
    fmt.Println("-----------------------------------------------")

    var prevState [][]byte
    defaultState := prevState
    var encrypted [][][]byte
    for _, b := range block {
        res := make([][]byte, 4)
        for i := range res {
            res[i] = make([]byte, 4)
        }
        cipherState := cipher(prevState, roundKeys, masterKey)
        for i := range b {
            for j := range b[i] {
                res[i][j] = cipherState[i][j] ^ b[i][j]
            }
        }
        encrypted = append(encrypted, res)
        prevState = cipherState
    }

    fmt.Println("The final cipher after encryption:")
    for _, b := range encrypted {
        for _, line := range b {
            for _, c := range line {
                fmt.Printf("%4d ", c)
            }
            fmt.Println()
        }
        fmt.Println()
    }
```

```
    fmt.Println("-----------------------------------------------")

    prevState = defaultState
    var decrypted [][][]byte
    for _, b := range encrypted {
        res := make([][]byte, 4)
        for i := range res {
            res[i] = make([]byte, 4)
        }
        cipherState := cipher(prevState, roundKeys, masterKey)
        for i := range b {
            for j := range b[i] {
                res[i][j] = b[i][j] ^ cipherState[i][j]
            }
        }
        decrypted = append(decrypted, res)
        prevState = cipherState
    }

    var decrypted2 [][][]byte
    for _, b := range encrypted {
        res := make([][]byte, 4)
        for i := range res {
            res[i] = make([]byte, 4)
        }
        cipherState := decipher(b, roundKeys, masterKey)
        for i := range cipherState {
            for j := range cipherState[i] {
                res[i][j] = cipherState[i][j]
            }
        }
        decrypted2 = append(decrypted2, res)
    }

    fmt.Println("Message after decryption:")
    for _, b := range decrypted {
        for _, line := range b {
            for _, c := range line {
                fmt.Printf("%c", c)
            }
        }
    }
    fmt.Println("\n-----------------------------------------------")
}
```

Результат работы программы:

```
Input text for cipher >> Привет меня зовут боб
-----------------------------------------------
The encryption block:
   П    е    е    з
   р    т    н    о
   и         я    в
   в    м         у
   т    б

   б
   о

-----------------------------------------------
128-bit master key: i0v0n7z6x3l8z2m4
-----------------------------------------------
Generated keys:
  69   30   76   30
  6e   37   7a   36
  78   33   6c   38
  7a   32   6d   34
  13    2   1b    4
  7d   35   61   32
   5    6    d    a
  7f   34   60   3e
  6c   36   7b   3a
  11    3   1a    8
  14    5   17    2
  6b   31   77   3c
   7    7    c    6
  16    4   16    e
   2    1    1    c
  69   30   76   30
  6e   37   7a   36
```

```
  69   30   76   30
  6e   37   7a   36
  78   33   6c   38
  7a   32   6d   34
  13    2   1b    4
  7d   35   61   32
   5    6    d    a
  7f   34   60   3e
  6c   36   7b   3a
  11    3   1a    8
  14    5   17    2
  6b   31   77   3c
   7    7    c    6
  16    4   16    e

------------------------------------------------
The final cipher after encryphion:

       K    ч    m
   Ц    #    )    P
   Ъ    ™    `
   Ц         Ґ    ь
   “    ‹    7    .
   Н    Ю    l    ж
   Ц    4    w    \
   я    x    i    Є

------------------------------------------------
Message after decryption:

Привет меня зовут боб
------------------------------------------------
```

Задание 3:
C++:

```cpp
#include <iostream>
#include <vector>
#include <cmath>
#include <iomanip>
#include <algorithm>

using namespace std;

// Функция для выбора главного элемента в текущем столбце
```

```
int findPivot(const vector<vector<double>>& matrix, int col, int start_row) {
    double max_val = 0.0;
    int pivot_row = start_row;

    for (int i = start_row; i < matrix.size(); i++) { //проходимся по текущей
строке
        if (fabs(matrix[i][col]) > max_val) {      //находим максимальный по
модулю элемент в столбце проходясь по всем строкам
            max_val = fabs(matrix[i][col]);
            pivot_row = i;                         //проходимся по строке
        }
    }
    return pivot_row + 1;              //возвращаем номер строки, начиная с 1
}

// Функция для решения системы линейных уравнений
vector<double> solveLinearSystem(const vector<vector<double>>& matrix) {
    int n = matrix.size();
    int m = matrix[0].size() - 1;
    vector<vector<double>> working_matrix = matrix; //копия матрицы

    // Прямой ход
    cout << "Прямой ход:" << endl;
    for (int col = 0; col < m; col++) {
        int pivot_row = findPivot(working_matrix, col, col); //находим номер
строки
        double pivot_value = working_matrix[pivot_row - 1][col]; //находим
главный элемент
        cout << "Выбран главный элемент " << pivot_value << " в строке " <<
pivot_row << endl;

        if (pivot_row != col + 1) {
            //меняем местами строки, если необходимо, так как галвные на
диагонали
            swap(working_matrix[col], working_matrix[pivot_row - 1]);
//используем индекс, начиная с 0
            cout << "Меняем местами строки " << col + 1 << " и " << pivot_row <<
endl;
        }

        //элементарные преобразования для получения нулей под главным элементом
        for (int i = col + 1; i < n; i++) {
            double factor = working_matrix[i][col] / working_matrix[col][col];
            cout << "Вычитаем из строки " << i + 1 << " " << factor << endl;

            for (int j = col; j < m + 1; j++) {
                working_matrix[i][j] -= factor * working_matrix[col][j];
            }
        }

        //выводим промежуточную матрицу
```

```
        cout << "Промежуточная матрица после " << col + 1 << "-го столбца:" <<
endl;
        for (const auto& row : working_matrix) {
            for (double elem : row) {
                if (fabs(elem) < 3e-17){
                    cout << 0 << " ";
                } else {
                    cout << elem << " ";
                }
            }
            cout << endl;
        }
        cout << endl;
    }

    // Обратный ход
    cout << "Обратный ход:" << endl;
    vector<double> solution(m, 0.0);
    for (int i = n - 1; i >= 0; i--) {
        double sum = 0.0;
        for (int j = i + 1; j < m; j++) {
            sum += working_matrix[i][j] * solution[j];
        }
        solution[i] = (working_matrix[i][m] - sum) / working_matrix[i][i];
        cout << "x" << i + 1 << " = " << solution[i] << endl;
    }

    return solution;
}

void checkSpectralRadius(vector<vector<double>>& C){
    double spectralRadius = 0.0;
    for (int i = 0; i < 4; i++) {
        double sum = 0.0;
        for (int j = 0; j < 4; j++) {
            sum += fabs(C[i][j]);
        }
        if (sum > spectralRadius) {
            spectralRadius = sum;
        }
    }

    cout << "\nСпектральный радиус матрицы C: " << spectralRadius << endl;

    if (spectralRadius >= 1.0) {
        cout << "Условие сходимости метода простых итераций не выполняется." <<
endl;
    } else {
        cout << "Условие сходимости метода простых итераций выполняется." <<
endl;
    }
```

```
}

// Преобразование системы Ax=b к виду x=Cx+f
void formCanonicalSystem(const vector<vector<double>>& A, const vector<double>&
b, vector<vector<double>>& C, vector<double>& f, int n) {
    for (int i = 0; i < n; i++) {
        f[i] = b[i] / A[i][i];
        for (int j = 0; j < n; j++) {
            if (j != i) {
                C[i][j] = -A[i][j] / A[i][i];
            } else {
                C[i][j] = 0.0;
            }
        }
    }
}

//метод простых итераций
bool simpleIteration(const vector<vector<double>>& C, const vector<double>& f,
vector<double>& x, int n, double epsilon) {
    vector<double> xNew(n);
    int k = 0;
    double maxDiff = 0.0;
    int maxIteration = 10;

    // Задаем начальное приближение
    for (int i = 0; i < n; i++) {
        x[i] = 0.0;
    }

    cout << "N" << setw(10) << "x1" << setw(10) << "x2" << setw(10) << "x3" <<
setw(10) << "x4" << setw(10) << "εn" << endl;

    do {
        // Вычисляем новое приближение
        for (int i = 0; i < n; i++) {
            double sum = f[i];
            for (int j = 0; j < n; j++) {
                sum += C[i][j] * x[j];
            }
            xNew[i] = sum;
        }

        // Проверяем условие остановки
        maxDiff = 0.0;
        for (int i = 0; i < n; i++) {
            if (fabs(xNew[i] - x[i]) > maxDiff) {
                maxDiff = fabs(xNew[i] - x[i]);
            }
            x[i] = xNew[i];
        }
```

```
        k++;

        // Вывод результатов в таблицу
        cout << k << fixed << setprecision(3) << setw(10) << x[0] << setw(10) <<
x[1] << setw(10) << x[2] << setw(10) << x[3] << setw(10) << maxDiff << endl;
    } while (maxDiff > epsilon && k < maxIteration);

    cout << "Число итераций: " << k;
    if (k = 10){
        cout << "(максимум)";
    }

    // Вывод сообщения о сходимости или расходимости
    if (maxDiff <= epsilon) {
        cout << "\nМетод сходится." << endl;
    } else {
        cout << "\nМетод расходится." << endl;
        return 1;
    }
    return 0;
}

int main() {
    system("chcp 65001");

    vector<vector<double>> matrix = {
        {-1.21, -0.04,  0.21, -18.0, -1.24},
        { 0.25, -1.23, 0.2, -0.09,  0.88},
        {-0.21, 0.2,  0.80, -0.13,  2.56},
        { 0.15, -1.31,  0.06,  0.88, -1.21}
    };

    cout << "\tРешение СЛАУ с выбором главного элемента:\n" << endl;
    vector<double> solution = solveLinearSystem(matrix);
    cout << "Матрица решений СЛАУ: " << endl;
    for (double x : solution) {
        cout << x << " ";
    }
    cout << endl;

    vector<vector<double>> A = {
        {-1.14, -0.04,  0.21, -18.0},
        { 0.25, -1.23, -0.17, -0.09},
        {-0.21, -0.17,  0.80, -0.13},
        { 0.15, -1.31,  0.06,  0.95}
    };
    vector<double> b = {-1.24, 0.95, 2.56, -1.14};
    vector<double> x(4, 0.0);
    vector<vector<double>> C = {
        { 0.0, 0.0, 0.0, 0.0},
        { 0.0, 0.0, 0.0, 0.0},
```

```
        { 0.0, 0.0, 0.0, 0.0},
        { 0.0, 0.0, 0.0, 0.0}
    };
    vector<double> f(4, 0.0);

    formCanonicalSystem(A, b, C, f, 4);
    checkSpectralRadius(C);

    cout << "\nРешение методом простых итераций:" << endl;
    simpleIteration(C, f, x, 4, 0.001);
    if (!simpleIteration) {
        cout << "x1 = " << fixed << setprecision(3) << x[0] << endl;
        cout << "x2 = " << fixed << setprecision(3) << x[1] << endl;
        cout << "x3 = " << fixed << setprecision(3) << x[2] << endl;
        cout << "x4 = " << fixed << setprecision(3) << x[3] << endl;
    }

    return 0;
}
```

GO:

```
package main

import (
    "fmt"
    "math"
)

// Функция для выбора главного элемента в текущем столбце
func findPivot(matrix [][]float64, col, startRow int) int {
    maxVal := 0.0
    pivotRow := startRow

    for i := startRow; i < len(matrix); i++ {
        if math.Abs(matrix[i][col]) > maxVal {
            maxVal = math.Abs(matrix[i][col])
            pivotRow = i
        }
    }
    return pivotRow + 1 // возвращаем номер строки, начиная с 1
}

// Функция для решения системы линейных уравнений
func solveLinearSystem(matrix [][]float64) []float64 {
    n, m := len(matrix), len(matrix[0])-1
    workingMatrix := make([][]float64, n)
    for i := range workingMatrix {
        workingMatrix[i] = make([]float64, len(matrix[i]))
        copy(workingMatrix[i], matrix[i])
    }
```

```
    // Прямой ход
    fmt.Println("Прямой ход:")
    for col := 0; col < m; col++ {
        pivotRow := findPivot(workingMatrix, col, col)
        pivotValue := workingMatrix[pivotRow-1][col]
        fmt.Printf("Выбран главный элемент %.2f в строке %d\n", pivotValue,
pivotRow)

        if pivotRow != col+1 {
            workingMatrix[col], workingMatrix[pivotRow-1] =
workingMatrix[pivotRow-1], workingMatrix[col]
            fmt.Printf("Меняем местами строки %d и %d\n", col+1, pivotRow)
        }

        for i := col + 1; i < n; i++ {
            factor := workingMatrix[i][col] / workingMatrix[col][col]
            fmt.Printf("Вычитаем из строки %d %.2f\n", i+1, factor)

            for j := col; j < m+1; j++ {
                workingMatrix[i][j] -= factor * workingMatrix[col][j]
            }
        }

        fmt.Println("Промежуточная матрица после", col+1, "-го столбца:")
        for _, row := range workingMatrix {
            for _, elem := range row {
                if math.Abs(elem) < 3e-17 {
                    fmt.Print("0 ")
                } else {
                    fmt.Printf("%.2f ", elem)
                }
            }
            fmt.Println()
        }
        fmt.Println()
    }

    // Обратный ход
    fmt.Println("Обратный ход:")
    solution := make([]float64, m)
    for i := n - 1; i >= 0; i-- {
        sum := 0.0
        for j := i + 1; j < m; j++ {
            sum += workingMatrix[i][j] * solution[j]
        }
        solution[i] = (workingMatrix[i][m] - sum) / workingMatrix[i][i]
        fmt.Printf("x%d = %.2f\n", i+1, solution[i])
    }

    return solution
```

```
}

func checkSpectralRadius(C [][]float64) {
    spectralRadius := 0.0
    for i := 0; i < 4; i++ {
        sum := 0.0
        for j := 0; j < 4; j++ {
            sum += math.Abs(C[i][j])
        }
        if sum > spectralRadius {
            spectralRadius = sum
        }
    }

    fmt.Printf("\nСпектральный радиус матрицы C: %.2f\n", spectralRadius)

    if spectralRadius >= 1.0 {
        fmt.Println("Условие сходимости метода простых итераций не выполняется.")
    } else {
        fmt.Println("Условие сходимости метода простых итераций выполняется.")
    }
}

// Преобразование системы Ax=b к виду x=Cx+f
func formCanonicalSystem(A [][]float64, b []float64, C [][]float64, f []float64,
n int) {
    for i := 0; i < n; i++ {
        f[i] = b[i] / A[i][i]
        for j := 0; j < n; j++ {
            if j != i {
                C[i][j] = -A[i][j] / A[i][i]
            } else {
                C[i][j] = 0.0
            }
        }
    }
}

// Метод простых итераций
func simpleIteration(C [][]float64, f []float64, x []float64, n int, epsilon
float64) bool {
    xNew := make([]float64, n)
    k := 0
    maxDiff := 0.0
    maxIteration := 10

    // Задаем начальное приближение
    for i := range x {
        x[i] = 0.0
    }
```

```
    fmt.Printf("N%10sx1%10sx2%10sx3%10sx4%10sεn\n", "", "", "", "", "")

    for {
        // Вычисляем новое приближение
        for i := 0; i < n; i++ {
            sum := f[i]
            for j := 0; j < n; j++ {
                sum += C[i][j] * x[j]
            }
            xNew[i] = sum
        }

        // Проверяем условие остановки
        maxDiff = 0.0
        for i := 0; i < n; i++ {
            if math.Abs(xNew[i]-x[i]) > maxDiff {
                maxDiff = math.Abs(xNew[i] - x[i])
            }
            x[i] = xNew[i]
        }
        k++

        // Вывод результатов в таблицу
        fmt.Printf("%d%10.3f%10.3f%10.3f%10.3f%10.3f\n", k, x[0], x[1], x[2],
x[3], maxDiff)

        if maxDiff <= epsilon {
            fmt.Println("Число итераций:", k)
            fmt.Println("Метод сходится.")
            return false
        }

        if k >= maxIteration {
            fmt.Println("Число итераций:", k, "(максимум)")
            fmt.Println("Метод расходится.")
            return true
        }
    }
}

func main() {
    matrix := [][]float64{
        {-1.21, -0.04, 0.21, -18.0, -1.24},
        {0.25, -1.23, 0.2, -0.09, 0.88},
        {-0.21, 0.2, 0.80, -0.13, 2.56},
        {0.15, -1.31, 0.06, 0.88, -1.21},
    }

    fmt.Println("\tРешение СЛАУ с выбором главного элемента:\n")
    solution := solveLinearSystem(matrix)
    fmt.Println("Матрица решений СЛАУ:")
```

```
    for _, x := range solution {
        fmt.Printf("%.2f ", x)
    }
}
```

Результат работы программы:

```
Выбран главный элемент -1.77666 в строке 4
Промежуточная матрица после 4-го столбца:
-1.14 -0.04 0.21 -18 -1.24
0 -1.31526 0.0876316 -1.41842 -1.30316
0 0 0.75048 3.36118 2.94956
0 0 0 -1.77666 2.71696

Обратный ход:
x4 = -1.52925
x3 = 10.7793
x2 = 3.35817
x1 = 27.1016
Матрица решений СЛАУ:
27.1016 3.35817 10.7793 -1.52925

Спектральный радиус матрицы C: 16.0088
Условие сходимости метода простых итераций не выполняется.

Решение методом простых итераций:
N        x1        x2        x3        x4       εn
1     1.088    -0.772     3.200    -1.200     3.200
2    20.652    -0.906     3.126    -2.639    19.564
3    43.362     3.186     8.000    -5.907    22.710
4    95.722     7.368    14.300    -4.158    52.359
5    69.122    17.011    29.217    -7.057    26.599
6   117.306     9.755    23.813     9.498    48.184
7  -144.838    19.084    37.609    -7.774   262.145
8   130.095   -34.840   -32.028    45.610   274.933
9  -723.748    26.759    37.358   -67.761   853.843
10 1076.946  -148.081  -192.109   147.616  1800.694
Число итераций: 10(максимум)
Метод расходится.
```

```
        Решение СЛАУ с выбором главного элемента:

Прямой ход:
Выбран главный элемент -1.14 в строке 1
Вычитаем из строки 2 -0.219298
Вычитаем из строки 3 0.184211
Вычитаем из строки 4 -0.131579
Промежуточная матрица после 1-го столбца:
-1.14 -0.04 0.21 -18 -1.24
0 -1.23877 -0.123947 -4.03737 0.67807
0 -0.162632 0.761316 3.18579 2.78842
0 -1.31526 0.0876316 -1.41842 -1.30316

Выбран главный элемент -1.31526 в строке 4
Меняем местами строки 2 и 4
Вычитаем из строки 3 0.123649
Вычитаем из строки 4 0.941843
Промежуточная матрица после 2-го столбца:
-1.14 -0.04 0.21 -18 -1.24
0 -1.31526 0.0876316 -1.41842 -1.30316
0 0 0.75048 3.36118 2.94956
0 0 -0.206483 -2.70144 1.90544

Выбран главный элемент 0.75048 в строке 3
Вычитаем из строки 4 -0.275134
Промежуточная матрица после 3-го столбца:
-1.14 -0.04 0.21 -18 -1.24
0 -1.31526 0.0876316 -1.41842 -1.30316
0 0 0.75048 3.36118 2.94956
0 0 0 -1.77666 2.71696
```