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»  *Тятюшкин М.О.*  «20» февраля 2025 г.  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись) | Проверил:  *Ассистент кафедры ЗИ*  *Исаев Г. А.*  «20» февраля 2025 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 целыми числами в диапазоне [-50, 50]. Вычислите и выведите сумму всех неотрицательных элементов в чётных строках, стоящих на нечётных местах. Инициализируйте новый массив этими числами.

Определите и инициализируйте квадратную матрицу порядка N случайными целыми числами в диапазоне [100, 200]. Найдите сумму элементов строк матрицы. Определите и инициализируйте массив строкой со второй максимальной суммой элементов среди строк матрицы.

Реализуйте клеточный автомат Джона Конвея на ограниченной плоскости по классическим правилам. Продемонстрируйте работу клеточного автомата на примере «паровоза» (двигающиеся фигуры, которая оставляет за собой следы в виде устойчивых или периодических фигур) 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.

1. Систему шифрования AES128 (CFB) для преобразования исходного текста, зашифрованного текста.
2. Из табл. выбрать данные для системы линейных уравнений. Найти решение этой системы прямым и приближенным методами с точностью до ε=10-3. Решить систему уравнений методом LU-разложения.

**Задание 1:**

1. **C++:**

// Вариант 10

#include <iostream>

#include <vector>

#include <random>

#include <algorithm>

using namespace std;

void Generate\_Matrix(int\*\* matrix, int m, int n) { // ф ия заполнения матрицы рандомными значениями [-50; 50] и её вывод

    for (int i = 0; i < m; ++i) {

        for (int j = 0; j < n; ++j) {

            matrix[i][j] = rand() % 49 + 50;

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

        }

        cout << endl;

    }

}

int find\_elements(vector<int>& nums, int\*\* matrix, int m, int n) { // ф-ия нахождения неотрицательных элементов матрицы, стоящих в чётных строках, на нечётных местах. И их суммы

    int sum = 0;

    for (int i = 0; i < m; i++) {

        for (int j = 0; j < n; j++) {

            if (((i+1) % 2 == 0) && ((j+1) % 2 != 0) && (matrix[i][j] >= 0)) {

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

                sum += matrix[i][j];

            }

        }

    }

    return sum;

}

template <typename T>

ostream& operator<<(ostream& os, const vector<T>& vec) { // перегрузка вывода вектора

    for (const auto& element : vec) {

        os << element << " ";

    }

    return os;

}

int main() {

    setlocale(LC\_ALL, "Rus");

    srand(time(0));

    int m, n;

    cout << "Введите размерность матрицы: ";

    cin >> m >> n;

    int \*\*matrix = new int\*[m];

    for(int i = 0; i < m; i++) {

        matrix[i] = new int[n];

    }

    Generate\_Matrix(matrix, m, n);

    vector<int> nums;

    int sum = find\_elements(nums, matrix, m, n);

    cout << "Массив неотрицательных элементов матрицы, стоящих в чётных строках, на нечётных местах: " << nums << endl;

    cout << "Сумма этих элементов: " << sum << endl;

    return 0;

}



Введите размерность матрицы: 6 5

74 93 58 85 88

88 92 74 56 95

58 68 79 91 84

86 51 61 89 92

69 67 87 90 81

71 92 79 70 76

Массив неотрицательных элементов матрицы, стоящих в чётных строках, на нечётных местах: 88 74 95 86 61 92 71 79 76

Сумма этих элементов: 722

**1.1 Python:**

import random

def generate\_matrix(m, n):

    # Функция заполнения матрицы случайными значениями [-50; 50] и её вывод

    matrix = [[random.randint(-50, 50) for \_ in range(n)] for \_ in range(m)]

    for row in matrix:

        print(&apos; &apos;.join(map(str, row)))

    return matrix

def find\_elements(matrix, m, n):

    # Функция нахождения неотрицательных элементов матрицы, стоящих в четных строках, на нечетных местах. И их суммы

    nums = []

    sum = 0

    for i in range(m):

        for j in range(n):

            if (i + 1) % 2 == 0 and (j + 1) % 2 != 0 and matrix[i][j] >= 0:

                nums.append(matrix[i][j])

                sum += matrix[i][j]

    return nums, sum

m = int(input("Введите размерность матрицы: "))

n = int(input())

matrix = generate\_matrix(m, n)

nums, sum = find\_elements(matrix, m, n)

print("Массив неотрицательных элементов матрицы, стоящих в четных строках, на нечетных местах:", nums)

print("Сумма этих элементов:", sum)



Введите размерность матрицы: 6 5

74 93 58 85 88

88 92 74 56 95

58 68 79 91 84

86 51 61 89 92

69 67 87 90 81

71 92 79 70 76

Массив неотрицательных элементов матрицы, стоящих в чётных строках, на нечётных местах: 88 74 95 86 61 92 71 79 76

Сумма этих элементов: 722

**1.2 С++:**

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

template <typename T>

ostream& operator<<(ostream& os, const vector<T>& vec) { // перегрузка вывода вектора

    for (const auto& element : vec) {

        os << element << " ";

    }

    return os;

}

void Generate\_Matrix(int\*\* matrix, int m) { // ф-ия заполнения квадратной матрицы рандомными значениями [100:200] и её вывод

    for (int i = 0; i < m; ++i) {

        for (int j = 0; j < m; ++j) {

            matrix[i][j] = rand() % 99 + 100;

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

        }

        cout << endl;

    }

}

void find\_maxstr\_inmatrix(int\*\* matrix, int m) { // ф-ия нахождения строки со второй максимальной суммой элементов строк квадратной матрицы

    int sum = 0;

    vector<int> sums;

    for (int i = 0; i < m; i++ ) { // заполняем массив сумм элементов строк кв. матрицы

        sum = 0;

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

            sum += matrix[i][j];

        }

        sums.push\_back(sum);

    }

    cout << "Массив сумм элементов строк квадратной матрицы: " << sums << endl;

    int maxsumindex = distance(sums.begin(), max\_element(sums.begin(), sums.end())); // находим индекс строки с максимальной суммой элементов

    sums[maxsumindex] = 0; // обнуляем максимальную сумму

    maxsumindex = distance(sums.begin(), max\_element(sums.begin(), sums.end())); // ещё раз находим индекс макс. суммы элементов

    cout << "Строка со второй максимальной суммой элементов строк квадратной матрицы: ";

    for (int i = 0; i < m; ++i) {

        cout << matrix[maxsumindex][i] << " ";

    }

}

int main() {

    setlocale(LC\_ALL, "Rus");

    srand(time(0));

    int m;

    cout << "Введите размерность квадратной матрицы: ";

    cin >> m;

    int \*\*matrix = new int\*[m];

    for(int i = 0; i < m; i++) {

        matrix[i] = new int[m];

    }

    Generate\_Matrix(matrix, m);

    find\_maxstr\_inmatrix(matrix, m);

    return 0;

}



Введите размерность квадратной матрицы: 5

106 155 125 171 190

127 165 166 105 136

137 104 166 124 163

143 167 171 193 176

144 141 152 178 197

Массив сумм элементов строк квадратной матрицы: 747 699 694 850 812

Строка со второй максимальной суммой элементов строк квадратной матрицы: 144 141 152 178 197

**1.2 Python:**

import random

def print\_matrix(matrix):

    # Функция вывода матрицы

    for row in matrix:

        print(&apos; &apos;.join(map(str, row)))

def generate\_matrix(m):

    # Функция заполнения квадратной матрицы случайными значениями [100; 200] и её вывод

    matrix = [[random.randint(100, 200) for \_ in range(m)] for \_ in range(m)]

    print\_matrix(matrix)

    return matrix

def find\_maxstr\_inmatrix(matrix):

    # Функция нахождения строки со второй максимальной суммой элементов строк квадратной матрицы

    sums = [sum(row) for row in matrix]

    print("Массив сумм элементов строк квадратной матрицы:", sums)

    max\_sum = max(sums)

    sums[sums.index(max\_sum)] = -1  # Обнуляем максимальную сумму

    second\_max\_sum = max(sums)

    second\_max\_index = sums.index(second\_max\_sum)

    print("Строка со второй максимальной суммой элементов строк квадратной матрицы:", matrix[second\_max\_index])

m = int(input("Введите размерность квадратной матрицы: "))

matrix = generate\_matrix(m)

find\_maxstr\_inmatrix(matrix)



Введите размерность квадратной матрицы: 5

106 155 125 171 190

127 165 166 105 136

137 104 166 124 163

143 167 171 193 176

144 141 152 178 197

Массив сумм элементов строк квадратной матрицы: 747 699 694 850 812

Строка со второй максимальной суммой элементов строк квадратной матрицы: 144 141 152 178 197

**C++:**

#include <iostream>

#include <vector>

#include <chrono>

#include <thread>

using namespace std;

// Размеры игрового поля

const int WIDTH = 50;

const int HEIGHT = 20;

// Символы для отображения живых и мертвых клеток

const char ALIVE\_CELL = &apos;\*&apos;;

const char DEAD\_CELL = &apos;.&apos;;

// Задержка между кадрами (в миллисекундах)

const int FRAME\_DELAY = 100;

// Функция для отрисовки текущего состояния игрового поля

void draw(const vector<vector<bool>>& grid) {

    system("cls"); // Очистка консоли

    for (int y = 0; y < HEIGHT; ++y) {

        for (int x = 0; x < WIDTH; ++x) {

            cout << (grid[y][x] ? ALIVE\_CELL : DEAD\_CELL); // Если true - живая кл, false - мертвая

        }

        cout << endl;

    }

}

// Функция для обновления состояния игрового поля на один шаг

void update(vector<vector<bool>>& grid) {

    vector<vector<bool>> newGrid = grid;

    // Применяем правила клеточного автомата

    for (int y = 0; y < HEIGHT; ++y) {

        for (int x = 0; x < WIDTH; ++x) {

            // Считаем количество живых соседей

            int aliveNeighbors = 0;

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

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

                    if (dx == 0 && dy == 0) continue; // Пропускаем текущую клетку

                    int ny = (y + dy + HEIGHT) % HEIGHT;

                    int nx = (x + dx + WIDTH) % WIDTH;

                    if (grid[ny][nx]) aliveNeighbors++;

                }

            }

            // Применяем правила

            if (grid[y][x]) {

                if (aliveNeighbors < 2 || aliveNeighbors > 3) newGrid[y][x] = false; // Умирает от одиночества или перенаселения

            } else {

                if (aliveNeighbors == 3) newGrid[y][x] = true; // Воскрешается от трех соседей

            }

        }

    }

    grid = newGrid;

}

int main() {

    srand(time(0));

    // Инициализация игрового поля

    vector<vector<bool>> grid(HEIGHT, vector<bool>(WIDTH, false));

    // Пример «паровоза»

    grid[5][5] = true;

    grid[5][6] = true;

    grid[5][7] = true;

    grid[4][7] = true;

    grid[3][6] = true;

    // Основной цикл игры

    while (true) {

        draw(grid);

        update(grid);

        this\_thread::sleep\_for(chrono::milliseconds(FRAME\_DELAY)); // Задержка между кадрами

    }

    return 0;

}

**Python:**

import os

import time

import random

# Размеры игрового поля

WIDTH = 50

HEIGHT = 20

# Символы для отображения живых и мертвых клеток

ALIVE\_CELL = &apos;\*&apos;

DEAD\_CELL = &apos;.&apos;

# Задержка между кадрами (в секундах)

FRAME\_DELAY = 0.1

# Функция для отрисовки текущего состояния игрового поля

def draw(grid):

    os.system(&apos;cls&apos; if os.name == &apos;nt&apos; else &apos;clear&apos;)  # Очистка консоли

    for y in range(HEIGHT):

        for x in range(WIDTH):

            print(ALIVE\_CELL if grid[y][x] else DEAD\_CELL, end=&apos;&apos;)  # Если true - живая кл, false - мертвая

        print()

# Функция для обновления состояния игрового поля на один шаг

def update(grid):

    new\_grid = [[False for \_ in range(WIDTH)] for \_ in range(HEIGHT)]

    # Применяем правила клеточного автомата

    for y in range(HEIGHT):

        for x in range(WIDTH):

            # Считаем количество живых соседей

            alive\_neighbors = 0

            for dy in range(-1, 2):

                for dx in range(-1, 2):

                    if dx == 0 and dy == 0:

                        continue  # Пропускаем текущую клетку

                    ny = (y + dy + HEIGHT) % HEIGHT

                    nx = (x + dx + WIDTH) % WIDTH

                    if grid[ny][nx]:

                        alive\_neighbors += 1

            # Применяем правила

            if grid[y][x]:

                if alive\_neighbors < 2 or alive\_neighbors > 3:

                    new\_grid[y][x] = False  # Умирает от одиночества или перенаселения

            else:

                if alive\_neighbors == 3:

                    new\_grid[y][x] = True  # Воскрешается от трех соседей

    return new\_grid

# Инициализация игрового поля

grid = [[False for \_ in range(WIDTH)] for \_ in range(HEIGHT)]

# Пример «паровоза»

grid[5][5] = True

grid[5][6] = True

grid[5][7] = True

grid[4][7] = True

grid[3][6] = True

# Основной цикл игры

try:

    while True:

        draw(grid)

        grid = update(grid)

        time.sleep(FRAME\_DELAY)  # Задержка между кадрами

except KeyboardInterrupt:

    pass  # Выход из цикла при нажатии Ctrl+C

**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++**

**function.cpp**

**#include "header.h"**

**unsigned char Sbox[] = { // Таблица замен (S-box) для шифрования**

**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**

**};**

**unsigned char InvSbox[] = { // Обратная таблица замен (InvSbox) для расшифрования**

**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**

**};**

**// Генерация ключей для каждого раунда**

**vector<string> KEY(const string& key, const int& rounds) {**

**vector<string> keys(rounds);**

**keys[0] = key; // Начальный ключ**

**for (int i = 1; i < rounds; i++) {**

**keys[i] = keys[i - 1]; // Копируем предыдущий ключ**

**for (int j = 0; j < key.size(); j++) {**

**keys[i][j] = Sbox[static\_cast<unsigned char>(keys[i - 1][j])] ^ (j \* i); // Применяем S-box и XOR**

**}**

**}**

**return keys;**

**}**

**// Добавление раундового ключа (XOR с ключом)**

**string AddRoundKey(const string& text, const string& key) {**

**string result = text;**

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

**result[i] = text[i] ^ key[i];**

**}**

**return result;**

**}**

**// Замена байтов (S-box)**

**string SubBytes(const string& text) {**

**string result = text;**

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

**result[i] = Sbox[static\_cast<unsigned char>(text[i])];**

**}**

**return result;**

**}**

**// Циклический сдвиг строк (ShiftRows)**

**string ShiftRows(const string& text, const bool& crypt) {**

**string result = text;**

**int shift[4] = {0, 1, 2, 3}; // Смещения для каждой строки**

**if (!crypt) { // Если расшифрование, сдвигаем в обратную сторону**

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

**shift[i] = 4 - shift[i];**

**}**

**}**

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

**string row = result.substr(i \* 4, 4);**

**rotate(row.begin(), row.begin() + shift[i], row.end());**

**result.replace(i \* 4, 4, row);**

**}**

**return result;**

**}**

**// Умножение в поле GF(2^8)**

**unsigned char GF\_mul(unsigned char a, unsigned char b) {**

**unsigned char p = 0;**

**for (int i = 0; i < 8; i++) {**

**if (b & 1) p ^= a;**

**bool hi\_bit\_set = (a & 0x80);**

**a <<= 1;**

**if (hi\_bit\_set) a ^= 0x1B;**

**b >>= 1;**

**}**

**return p;**

**}**

**// Перемешивание столбцов (MixColumns)**

**string MixColumns(const string& text, const vector<int>& initialization) {**

**unsigned char state[4][4];**

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

**state[i % 4][i / 4] = static\_cast<unsigned char>(text[i]);**

**}**

**// Матрица для перемешивания столбцов**

**unsigned char matrix[4][4] = {**

**{2, 3, 1, 1},**

**{1, 2, 3, 1},**

**{1, 1, 2, 3},**

**{3, 1, 1, 2}**

**};**

**unsigned char temp[4][4];**

**for (int col = 0; col < 4; col++) {**

**for (int row = 0; row < 4; row++) {**

**temp[row][col] = GF\_mul(matrix[row][0], state[0][col]) ^**

**GF\_mul(matrix[row][1], state[1][col]) ^**

**GF\_mul(matrix[row][2], state[2][col]) ^**

**GF\_mul(matrix[row][3], state[3][col]);**

**}**

**}**

**string result;**

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

**result += temp[i % 4][i / 4];**

**}**

**return result;**

**}**

**AES.cpp**

#include "function.cpp"

// Функция для выполнения одного блока AES

string DeblockAES(string text, string key, bool crypt, vector<int> initialization) {

int rounds = 10;

vector<string> keys = KEY(key, rounds);

if (crypt == true) {

cout << endl << "Зашифрованное:" << endl << endl;

} else {

cout << endl << "Расшифровка:" << endl << endl;

}

cout << "Раунд 1:" << endl << "AddRoundKey: ";

string retention = AddRoundKey(text, key);

cout << retention << endl << endl;

for (int i = 1; i < rounds - 1; i++) {

cout << "Раунд " << i + 1 << ":" << endl;

if (crypt == true) {

cout << "SubBytes: ";

retention = SubBytes(retention);

cout << retention << endl;

cout << "ShiftRows: ";

retention = ShiftRows(retention, crypt);

cout << retention << endl;

cout << "MixColumns: ";

retention = MixColumns(retention, initialization);

cout << retention << endl;

cout << "AddRoundKey: ";

retention = AddRoundKey(retention, keys[i]);

cout << retention << endl << endl;

} else {

cout << "InvShiftRows: ";

retention = ShiftRows(retention, crypt);

cout << retention << endl;

cout << "InvSubBytes: ";

retention = SubBytes(retention);

cout << retention << endl;

cout << "AddRoundKey: ";

retention = AddRoundKey(retention, keys[i]);

cout << retention << endl << endl;

cout << "InvMixColumns: ";

retention = MixColumns(retention, initialization);

cout << retention << endl;

}

}

cout << "Последний раунд:" << endl;

if (crypt == true) {

cout << "SubBytes: ";

retention = SubBytes(retention);

cout << retention << endl;

cout << "ShiftRows: ";

retention = ShiftRows(retention, crypt);

} else {

cout << "InvShiftRows: ";

retention = ShiftRows(retention, crypt);

cout << retention << endl;

cout << "InvSubBytes: ";

retention = SubBytes(retention);

}

cout << retention << endl;

cout << "AddRoundKey: ";

retention = AddRoundKey(retention, keys[rounds - 1]);

cout << retention << endl;

cout << "Финальный результат: " << retention << endl;

return retention;

}

string EncryptDecipherCFB(string text, string key, bool crypt, vector<int> initialization) {

string result;

string iv = key;

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

string block = text.substr(i, 16);

string cipher = DeblockAES(iv, key, true, initialization);

string outputBlock = AddRoundKey(block, cipher);

result += outputBlock;

iv = crypt ? outputBlock : block; // Правильное обновление IV при расшифровке

}

return result;

}

int main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

string text = "Определите и инициализируйте матрицу размера M х N случайными целыми числами в диапазоне [10, 50]. Поменяйте в каждой строке местами максимальный и минимальный элементы. Определите и инициализируйте новую матрицу, первая строка которой содержит минимальные элементы, а вторая строка – максимальные.";

while (text.size() % 16 != 0) text += ' ';

random\_device ran;

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

string key(16, '\0');

for (char& c : key) c = dist(ran);

vector<int> initialization{3, 2, 4, 5};

string encryptedText = EncryptDecipherCFB(text, key, true, initialization);

cout << "Зашифрованный текст: " << encryptedText << endl;

string decryptedText = EncryptDecipherCFB(encryptedText, key, false, initialization);

while (!decryptedText.empty() && decryptedText.back() == ' ') decryptedText.pop\_back();

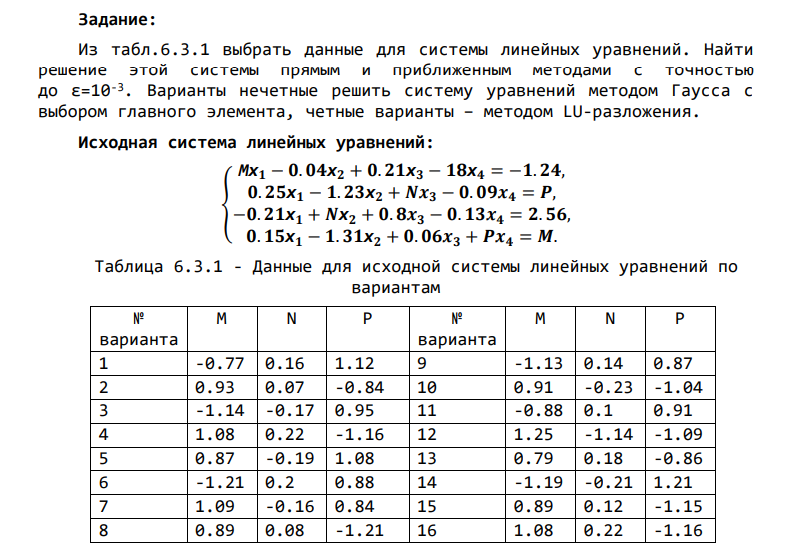
cout << "Расшифрованный текст: " << decryptedText << endl;

return 0;

}

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

**Вариант 11**

 **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;

}

**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.