**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра вычислительной техники**

отчет

**по лабораторной работе №1**

**по дисциплине «Программирование»**

Тема: Матрицы и их реализация

|  |  |  |
| --- | --- | --- |
| Студент гр. 4373 |  | Матвеев А.В. |
| Преподаватель |  | Глущенко |

Санкт-Петербург

2024

**Цель работы.**

Знакомство с матрицами.

**Текст программы**

**#include <iostream>**

**#include <iomanip> // Для std::setw**

**#include <Windows.h> // Для работы с Windows API**

**#include <thread> // Для использования std::this\_thread::sleep\_for**

**#include <random>**

**#include <algorithm>**

**#include <chrono>**

**#include <ctime>**

**#include <cstdlib>**

**using namespace std;**

**const int N = 4;**

**void setCursorPosition(HANDLE hStdout, int x, int y) {**

**COORD coord;**

**coord.X = x;**

**coord.Y = y;**

**SetConsoleCursorPosition(hStdout, coord);**

**}**

**void matrix\_zmeyka(HANDLE hStdout, int N) {**

**random\_device rd;**

**mt19937 gen(rd());**

**uniform\_int\_distribution<> dis(1, N \* N);**

**int value = 1;**

**int startstroka = 1;**

**setCursorPosition(hStdout, 0, startstroka);**

**cout << "Задание 1" << endl;**

**for (int j = 0; j < N; ++j) {**

**if (j % 2 == 0) { // Четные столбцы**

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

**setCursorPosition(hStdout, j \* 5, startstroka + i + 2); // Позиция для вывода с учётом отступа**

**cout << setw(4) << dis(gen);**

**cout.flush();**

**this\_thread::sleep\_for(chrono::milliseconds(100));**

**}**

**}**

**else { // Нечетные столбцы**

**for (int i = N - 1; i >= 0; --i) {**

**setCursorPosition(hStdout, j \* 5, startstroka + i + 2); // Позиция для вывода с учётом отступа**

**cout << setw(4) << dis(gen);**

**cout.flush();**

**this\_thread::sleep\_for(chrono::milliseconds(100));**

**}**

**}**

**}**

**}**

**void matrix\_spiral(HANDLE hStdout, int N) {**

**random\_device rd;**

**mt19937 gen(rd());**

**uniform\_int\_distribution<> dis(1, N \* N);**

**int value = 1;**

**int left = 0, right = N - 1, top = 0, bottom = N - 1;**

**while (left <= right && top <= bottom) {**

**// Заполняем верхнюю строку**

**for (int i = left; i <= right; ++i) {**

**setCursorPosition(hStdout, i \* 5, top + N + 4); // Позиция для вывода с отступом вниз**

**cout << setw(4) << dis(gen);**

**cout.flush();**

**this\_thread::sleep\_for(chrono::milliseconds(100));**

**}**

**top++;**

**// Заполняем правый столбец**

**for (int i = top; i <= bottom; ++i) {**

**setCursorPosition(hStdout, right \* 5, i + N + 4); // Позиция для вывода с отступом вниз**

**cout << setw(4) << dis(gen);**

**cout.flush();**

**this\_thread::sleep\_for(chrono::milliseconds(100));**

**}**

**right--;**

**// Заполняем нижнюю строку**

**if (top <= bottom) {**

**for (int i = right; i >= left; --i) {**

**setCursorPosition(hStdout, i \* 5, bottom + N + 4); // Позиция для вывода с отступом вниз**

**cout << setw(4) << dis(gen);**

**cout.flush();**

**this\_thread::sleep\_for(chrono::milliseconds(100));**

**}**

**bottom--;**

**}**

**// Заполняем левый столбец**

**if (left <= right) {**

**for (int i = bottom; i >= top; --i) {**

**setCursorPosition(hStdout, left \* 5, i + N + 4); // Позиция для вывода с отступом вниз**

**cout << setw(4) << dis(gen);**

**cout.flush();**

**this\_thread::sleep\_for(chrono::milliseconds(100));**

**}**

**left++;**

**}**

**}**

**}**

**void printQuadrantsA(int\* matrix, int N) {**

**int halfN = N / 2;**

**// Создаем массив для хранения квадрантов**

**int\* quadrants[4];**

**// Выделяем память для каждого квадранта**

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

**quadrants[i] = new int[halfN \* halfN];**

**}**

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

**for (int j = 0; j < halfN; ++j) {**

**quadrants[0][i \* halfN + j] = \*(matrix + i \* N + j);**

**quadrants[1][i \* halfN + j] = \*(matrix + i \* N + (j + halfN));**

**quadrants[2][i \* halfN + j] = \*(matrix + (i + halfN) \* N + j);**

**quadrants[3][i \* halfN + j] = \*(matrix + (i + halfN) \* N + (j + halfN));**

**}**

**}**

**cout << "Матрица в порядке 4-1-2-3 (A):" << endl;**

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

**for (int j = 0; j < halfN; ++j) {**

**cout << setw(4) << quadrants[2][i \* halfN + j];**

**}**

**for (int j = 0; j < halfN; ++j) {**

**cout << setw(4) << quadrants[0][i \* halfN + j];**

**}**

**cout << endl;**

**}**

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

**for (int j = 0; j < halfN; ++j) {**

**cout << setw(4) << quadrants[3][i \* halfN + j];**

**}**

**for (int j = 0; j < halfN; ++j) {**

**cout << setw(4) << quadrants[1][i \* halfN + j];**

**}**

**cout << endl;**

**}**

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

**delete[] quadrants[i];**

**}**

**}**

**void printQuadrantsB(int\* matrix, int N) {**

**int halfN = N / 2;**

**// Создаем массив для хранения квадрантов**

**int\* quadrants[4];**

**// Выделяем память для каждого квадранта**

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

**quadrants[i] = new int[halfN \* halfN];**

**}**

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

**for (int j = 0; j < halfN; ++j) {**

**quadrants[0][i \* halfN + j] = \*(matrix + i \* N + j);**

**quadrants[1][i \* halfN + j] = \*(matrix + i \* N + (j + halfN));**

**quadrants[2][i \* halfN + j] = \*(matrix + (i + halfN) \* N + j);**

**quadrants[3][i \* halfN + j] = \*(matrix + (i + halfN) \* N + (j + halfN));**

**}**

**}**

**cout << "Матрица в порядке 3-4-1-2 (B):" << endl;**

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

**for (int j = 0; j < halfN; ++j) {**

**cout << setw(4) << quadrants[3][i \* halfN + j];**

**}**

**for (int j = 0; j < halfN; ++j) {**

**cout << setw(4) << quadrants[2][i \* halfN + j];**

**}**

**cout << endl;**

**}**

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

**for (int j = 0; j < halfN; ++j) {**

**cout << setw(4) << quadrants[1][i \* halfN + j];**

**}**

**for (int j = 0; j < halfN; ++j) {**

**cout << setw(4) << quadrants[0][i \* halfN + j];**

**}**

**cout << endl;**

**}**

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

**delete[] quadrants[i];**

**}**

**}**

**void printQuadrantsC(int\* matrix, int N) {**

**int halfN = N / 2;**

**int\* quadrants[4];**

**// Выделяем память для каждого квадранта**

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

**quadrants[i] = new int[halfN \* halfN];**

**}**

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

**for (int j = 0; j < halfN; ++j) {**

**quadrants[0][i \* halfN + j] = \*(matrix + i \* N + j);**

**quadrants[1][i \* halfN + j] = \*(matrix + i \* N + (j + halfN));**

**quadrants[2][i \* halfN + j] = \*(matrix + (i + halfN) \* N + j);**

**quadrants[3][i \* halfN + j] = \*(matrix + (i + halfN) \* N + (j + halfN));**

**}**

**}**

**cout << "Матрица в порядке 4-3-2-1 (C):" << endl;**

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

**for (int j = 0; j < halfN; ++j) {**

**cout << setw(4) << quadrants[2][i \* halfN + j];**

**}**

**for (int j = 0; j < halfN; ++j) {**

**cout << setw(4) << quadrants[3][i \* halfN + j];**

**}**

**cout << endl;**

**}**

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

**for (int j = 0; j < halfN; ++j) {**

**cout << setw(4) << quadrants[0][i \* halfN + j];**

**}**

**for (int j = 0; j < halfN; ++j) {**

**cout << setw(4) << quadrants[1][i \* halfN + j];**

**}**

**cout << endl;**

**}**

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

**delete[] quadrants[i];**

**}**

**}**

**void printQuadrantsD(int\* matrix, int N) {**

**int halfN = N / 2;**

**// Создаем массив для хранения квадрантов**

**int\* quadrants[4];**

**// Выделяем память для каждого квадранта**

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

**quadrants[i] = new int[halfN \* halfN];**

**}**

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

**for (int j = 0; j < halfN; ++j) {**

**quadrants[0][i \* halfN + j] = \*(matrix + i \* N + j);**

**quadrants[1][i \* halfN + j] = \*(matrix + i \* N + (j + halfN));**

**quadrants[2][i \* halfN + j] = \*(matrix + (i + halfN) \* N + j);**

**quadrants[3][i \* halfN + j] = \*(matrix + (i + halfN) \* N + (j + halfN));**

**}**

**}**

**cout << "Матрица в порядке 2-1-4-3 (D):" << endl;**

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

**for (int j = 0; j < halfN; ++j) {**

**cout << setw(4) << quadrants[1][i \* halfN + j];**

**}**

**for (int j = 0; j < halfN; ++j) {**

**cout << setw(4) << quadrants[0][i \* halfN + j];**

**}**

**cout << endl;**

**}**

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

**for (int j = 0; j < halfN; ++j) {**

**cout << setw(4) << quadrants[3][i \* halfN + j];**

**}**

**for (int j = 0; j < halfN; ++j) {**

**cout << setw(4) << quadrants[2][i \* halfN + j];**

**}**

**cout << endl;**

**}**

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

**delete[] quadrants[i];**

**}**

**}**

**void quickSort(int\* arr, int left, int right) {**

**int i = left, j = right;**

**int pivot = \*(arr + (left + right) / 2);**

**while (i <= j) {**

**while (\*(arr + i) < pivot) {**

**i++;**

**}**

**while (\*(arr + j) > pivot) {**

**j--;**

**}**

**if (i <= j) {**

**swap(\*(arr + i), \*(arr + j));**

**i++;**

**j--;**

**}**

**}**

**if (left < j) {**

**quickSort(arr, left, j);**

**}**

**if (i < right) {**

**quickSort(arr, i, right);**

**}**

**}**

**void manipulateMatrix(int\* matrix, int N, char operation, int number) {**

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

**switch (operation) {**

**case '+':**

**\*(matrix + i) += number;**

**break;**

**case '-':**

**\*(matrix + i) -= number;**

**break;**

**case '\*':**

**\*(matrix + i) \*= number;**

**break;**

**case '/':**

**if (number != 0) {**

**\*(matrix + i) /= number;**

**break;**

**default:**

**cout << "Неверное значение" << endl;**

**return;**

**}**

**}**

**}**

**cout << "Измененная матрица:" << endl;**

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

**for (int j = 0; j < N; ++j) {**

**cout << setw(4) << \*(matrix + i \* N + j);**

**}**

**cout << endl;**

**}**

**}**

**void idz(){**

**const int z = 3;**

**int matrixid[z][z];**

**int determinant = 0;**

**srand(time(0));**

**// Заполнение матрицы случайными числами от -30 до 30**

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

**for (int j = 0; j < z; j++) {**

**matrixid[i][j] = rand() % 61 - 30; // Генерация числа от -30 до 30**

**}**

**}**

**// Вывод матрицы**

**cout << "Matrix:" << endl;**

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

**for (int j = 0; j < z; j++) {**

**cout << setw(4) << matrixid[i][j];**

**}**

**cout << endl;**

**}**

**// Вычисление определителя матрицы**

**determinant = matrixid[0][0] \* (matrixid[1][1] \* matrixid[2][2] - matrixid[1][2] \* matrixid[2][1])**

**- matrixid[0][1] \* (matrixid[1][0] \* matrixid[2][2] - matrixid[1][2] \* matrixid[2][0])**

**+ matrixid[0][2] \* (matrixid[1][0] \* matrixid[2][1] - matrixid[1][1] \* matrixid[2][0]);**

**cout << "Определитель: " << determinant << endl;**

**}**

**int main() {**

**HANDLE hStdout = GetStdHandle(STD\_OUTPUT\_HANDLE);**

**setlocale(0, "");**

**int N;**

**cout << "Какой порядок матрицы (6,8,10): ";**

**cin >> N;**

**if (N != 6 && N != 8 && N != 10) {**

**cout << "Неверный порядок матрицы!" << endl;**

**return 1;**

**}**

**matrix\_zmeyka(hStdout, N);**

**cout << endl;**

**matrix\_spiral(hStdout, N);**

**cout << endl << endl << endl << endl << endl;**

**random\_device rd;**

**mt19937 gen(rd());**

**uniform\_int\_distribution<> dis(1, N \* N);**

**int value = 1;**

**int\* matrix = new int[N \* N];**

**// Заполнение матрицы по схеме змейки перед разделением на квадранты**

**for (int j = 0; j < N; ++j) {**

**if (j % 2 == 0) { // Четные столбцы**

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

**\*(matrix + i \* N + j) = dis(gen);**

**}**

**}**

**else { // Нечетные столбцы**

**for (int i = N - 1; i >= 0; --i) {**

**\*(matrix + i \* N + j) = dis(gen);**

**}**

**}**

**}**

**cout << "Задание 2" << endl;**

**cout << "Матрица из рандомных элементов для последующего разделения на квадраты" << endl;**

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

**for (int j = 0; j < N; ++j) {**

**cout << setw(4) << \*(matrix + i \* N + j);**

**}**

**cout << endl;**

**}**

**cout << endl;**

**printQuadrantsA(matrix, N);**

**cout << endl;**

**printQuadrantsB(matrix, N);**

**cout << endl;**

**printQuadrantsD(matrix, N);**

**cout << endl;**

**printQuadrantsC(matrix, N);**

**cout << endl;**

**quickSort(matrix, 0, N \* N - 1);**

**cout << endl;**

**cout << "Задание 3" << endl;**

**cout << "Отсортированная матрица quicksort" << endl;**

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

**for (int j = 0; j < N; ++j) {**

**cout << setw(4) << \*(matrix + i \* N + j);**

**}**

**cout << endl;**

**}**

**cout << endl;**

**cout << "Задание 4" << endl;**

**char operation;**

**int number;**

**cout << "Выберите операцию (+, -, \*, /)";**

**cin >> operation;**

**cout << "Введите число для операции: ";**

**cin >> number;**

**manipulateMatrix(matrix, N, operation, number);**

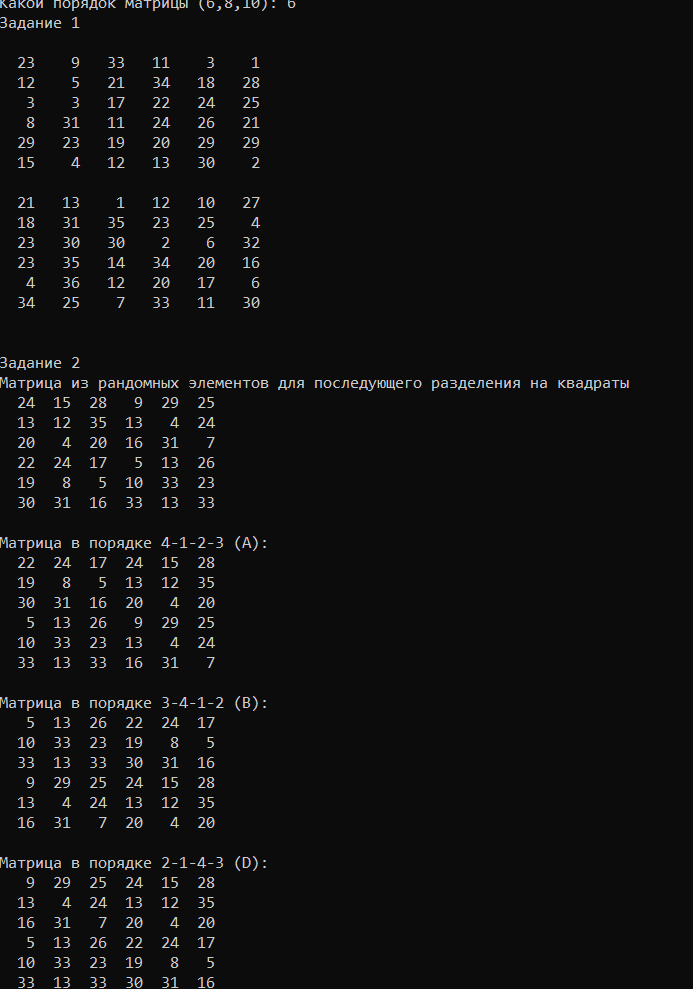
**cout << endl;**

**cout << "Идз 10" << endl;**

**idz();**

**return 0;**

**}**

****