**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра информационных систем**

отчет

**по практической работе №3**

**по дисциплине «Программирование»**

Тема: **Двумерные статические массивы. Указатели.**

|  |  |  |
| --- | --- | --- |
| Студент гр. 4372 |  | Максимов Е. В. |
| Преподаватель |  | Глущенко А. Г. |

Санкт-Петербург

2024

**Цель работы.**

Разработка программы для работы с матрицами, включая их заполнение, отображение, сортировку и выполнение арифметических операций с применением указателей для доступа к элементам.

**Основные теоретические положения.**

Двумерные массивы в C++ представляют собой структуру данных, которая позволяет хранить данные в виде таблицы, состоящей из строк и столбцов. Они являются расширением одномерных массивов и могут использоваться для решения различных задач, связанных с обработкой и хранением данных в двумерном пространстве.

**1. Объявление двумерных массивов**

Двумерные массивы в C++ объявляются с использованием синтаксиса, аналогичного одномерным массивам, но с добавлением второго размера. Например, для объявления двумерного массива целых чисел размером 3 на 4 можно использовать следующий код: int array[3][4]; Здесь 3 — это количество строк, а 4 — количество столбцов.

**2. Инициализация двумерных массивов**

Двумерные массивы могут быть инициализированы при объявлении. Например:

int array[3][4] = {

{1, 2, 3, 4},

{5, 6, 7, 8},

{9, 10, 11, 12}

};

Также можно инициализировать массивы частично:

int array[3][4] = {

{1, 2},

{5, 6, 7},

{9}

};

В этом случае неинициализированные элементы будут иметь значение 0.

**3. Доступ к элементам двумерных массивов**

Доступ к элементам двумерного массива осуществляется с помощью индексов, где первый индекс указывает на строку, а второй — на столбец. Например, чтобы получить доступ к элементу, находящемуся во второй строке и третьем столбце, можно использовать следующий код:

int value = array[1][2]; // Получение значения 7

Также доступ к элементам двумерного массива можно осуществить с использованием указателей, рассматривая массив как массив указателей на строки. Например, для двумерного массива matrix[N][M] можно получить доступ к элементу matrix[i][j] через указатели, используя синтаксис \*(\*(matrix + i) + j), где matrix + i указывает на i-ю строку, а \*(...) + j позволяет обратиться к j-му элементу этой строки. Также можно использовать арифметику указателей, рассматривая массив как одномерный, что позволяет получить доступ к элементу через выражение \*(&matrix[0][0] + i \* M + j), где M — количество столбцов.

**4. Перебор элементов двумерного массива**

Для перебора элементов двумерного массива можно использовать вложенные циклы. Например, чтобы вывести все элементы массива на экран, можно использовать следующий код:

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

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

std::cout << array[i][j] << " ";

}

std::cout << std::endl;

}

**Постановка задачи.**

Необходимо написать программу, которая:

1. Используя арифметику указателей, заполняет квадратичную целочисленную матрицу порядка *N* (6,8,10) случайными числами от 1 до N\*N спиралью и змейкой. Пользователь должен видеть процесс заполнения матрицы;
2. Получает новую матрицу, перестанавливая блоки исходных матриц местами;
3. Отсортировывает матрицу при помощи алгоритма Quicksort;
4. Уменьшает, увеличивает, умножает или делит все элементы матрицы на введенное пользователем число;
5. Находит определитель матрицы размером 3x3.

**Выполнение работы.**

Код программы представлен в приложении А.

| Ввод пользователем и обработка данных | Работа алгоритма и вывод на экран |
| --- | --- |
| Генерация и отображение исходных матриц | |
| При запуске программы пользователь видит анимацию заполнения матриц исходными значениями, а также все перестановки. | Генерация матрицы спиралью:    Вывод всех перестановок матриц на экран: |
| Сортировка матриц | |
| Пользователь видит исходные матрицы и отсортированные. | Сортировка матриц выполняется методом Quicksort: |
| Модификация элементов матриц | |
| Пользователь вводит число и арифметическое действие, которое будет применено ко всем элементам матриц. | Ввод числа и арифметического действия:    Программа изменяет элементы матриц согласно параметрам, которые задал пользователь и выводит их на экран: |
| Вычисление определителя матрицы 3x3 | |
| Пользователь видит процесс вычисления определителя случайно сгенерированной матрицы размера 3x3. | Вычисление определителя матрицы 3x3: |

**Выводы.**

В ходе разработки программы была создана структурированная и интерактивная система для работы с матрицами, включающая генерацию случайных чисел, визуализацию данных в консоли, реализацию алгоритма быстрой сортировки и выполнение математических операций над элементами матриц.

Приложение А

рабочий код

#include <iostream>  
#include <ctime>  
#include <Windows.h>  
  
using namespace std;  
  
const int N = 6;  
const int WIDTH = 3;  
  
HANDLE hStdout = GetStdHandle(STD\_OUTPUT\_HANDLE);  
COORD destCoord;  
  
  
void set\_element(int matrix[][N], int x, int y, int offset\_x, int offset\_y) {  
 \*(\*(matrix + y) + x) = (rand() % (N \* N)) + 1;  
 destCoord.X = (x + offset\_x) \* WIDTH;  
 destCoord.Y = y + offset\_y;  
 SetConsoleCursorPosition(hStdout, destCoord);  
 cout << \*(\*(matrix + y) + x);  
 Sleep(10);  
}  
  
void print\_matrix(int matrix[][N], int offset\_x, int offset\_y) {  
 for (int i = 0; i < N; i++) {  
 for (int j = 0; j < N; j++) {  
 destCoord.X = (i + offset\_x) \* WIDTH;  
 destCoord.Y = j + offset\_y;  
 SetConsoleCursorPosition(hStdout, destCoord);  
 cout << \*(\*(matrix + i) + j);  
 }  
 }  
}  
  
void quicksort(int arr[][N], int end, int begin) {  
 int mid;  
 int f = begin;  
 int l = end;  
 int index = (f + l) / 2;  
 mid = \*(\*(arr + index % N) + index / N);  
 while (f < l) {  
 while (\*(\*(arr + f % N) + f / N) < mid) f++;  
 while (\*(\*(arr + l % N) + l / N) > mid) l--;  
 if (f <= l) {  
 swap(\*(\*(arr + f % N) + f / N), \*(\*(arr + l % N) + l / N));  
 f++;  
 l--;  
 }  
 }  
 if (begin < l) quicksort(arr, l, begin);  
 if (f < end) quicksort(arr, end, f);  
}  
  
  
void show\_a(int matrix[][N], int offset\_x, int offset\_y) {  
 for (int i = 0; i < N / 2; i++) {  
 for (int j = 0; j < N / 2; j++) {  
 destCoord.X = offset\_x + j \* WIDTH;  
 destCoord.Y = offset\_y + i;  
 SetConsoleCursorPosition(hStdout, destCoord);  
 cout << \*(\*(matrix + i + N / 2) + j);  
  
 destCoord.X = offset\_x + (j + N / 2) \* WIDTH;  
 destCoord.Y = offset\_y + i;  
 SetConsoleCursorPosition(hStdout, destCoord);  
 cout << \*(\*(matrix + i) + j);  
  
 destCoord.X = offset\_x + j \* WIDTH;  
 destCoord.Y = offset\_y + i + N / 2;  
 SetConsoleCursorPosition(hStdout, destCoord);  
 cout << \*(\*(matrix + i + N / 2) + j + N / 2);  
  
 destCoord.X = offset\_x + (j + N / 2) \* WIDTH;  
 destCoord.Y = offset\_y + i + N / 2;  
 SetConsoleCursorPosition(hStdout, destCoord);  
 cout << \*(\*(matrix + i) + j + N / 2);  
 }  
 }  
}  
  
void show\_b(int matrix[][N], int offset\_x, int offset\_y) {  
 for (int i = 0; i < N / 2; i++) {  
 for (int j = 0; j < N / 2; j++) {  
 destCoord.X = (offset\_x + (j + N / 2)) \* WIDTH;  
 destCoord.Y = offset\_y + i;  
 SetConsoleCursorPosition(hStdout, destCoord);  
 cout << \*(\*(matrix + i + N / 2) + j);  
  
 destCoord.X = (offset\_x + j + N / 2) \* WIDTH;  
 destCoord.Y = offset\_y + i + N / 2;  
 SetConsoleCursorPosition(hStdout, destCoord);  
 cout << \*(\*(matrix + i) + j);  
  
 destCoord.X = (offset\_x + j) \* WIDTH;  
 destCoord.Y = offset\_y + i;  
 SetConsoleCursorPosition(hStdout, destCoord);  
 cout << \*(\*(matrix + i + N / 2) + j + N / 2);  
  
 destCoord.X = (offset\_x + j) \* WIDTH;  
 destCoord.Y = offset\_y + i + N / 2;  
 SetConsoleCursorPosition(hStdout, destCoord);  
 cout << \*(\*(matrix + i) + j + N / 2);  
 }  
 }  
}  
  
void show\_c(int matrix[][N], int offset\_x, int offset\_y) {  
 for (int i = 0; i < N / 2; i++) {  
 for (int j = 0; j < N / 2; j++) {  
 destCoord.X = (offset\_x + j) \* WIDTH;  
 destCoord.Y = offset\_y + i;  
 SetConsoleCursorPosition(hStdout, destCoord);  
 cout << \*(\*(matrix + i + N / 2) + j);  
  
 destCoord.X = (offset\_x + j) \* WIDTH;  
 destCoord.Y = offset\_y + i + N / 2;  
 SetConsoleCursorPosition(hStdout, destCoord);  
 cout << \*(\*(matrix + i) + j);  
  
 destCoord.X = (offset\_x + j + N / 2) \* WIDTH;  
 destCoord.Y = offset\_y + i;  
 SetConsoleCursorPosition(hStdout, destCoord);  
 cout << \*(\*(matrix + i + N / 2) + j + N / 2);  
  
 destCoord.X = (offset\_x + j + N / 2) \* WIDTH;  
 destCoord.Y = offset\_y + i + N / 2;  
 SetConsoleCursorPosition(hStdout, destCoord);  
 cout << \*(\*(matrix + i) + j + N / 2);  
 }  
 }  
}  
  
void show\_d(int matrix[][N], int offset\_x, int offset\_y) {  
 for (int i = 0; i < N / 2; i++) {  
 for (int j = 0; j < N / 2; j++) {  
 destCoord.X = (offset\_x + j + N / 2) \* WIDTH;  
 destCoord.Y = offset\_y + i + N / 2;  
 SetConsoleCursorPosition(hStdout, destCoord);  
 cout << \*(\*(matrix + i + N / 2) + j);  
  
 destCoord.X = (offset\_x + j + N / 2) \* WIDTH;  
 destCoord.Y = offset\_y + i;  
 SetConsoleCursorPosition(hStdout, destCoord);  
 cout << \*(\*(matrix + i) + j);  
  
 destCoord.X = (offset\_x + j) \* WIDTH;  
 destCoord.Y = offset\_y + i + N / 2;  
 SetConsoleCursorPosition(hStdout, destCoord);  
 cout << \*(\*(matrix + i + N / 2) + j + N / 2);  
  
 destCoord.X = (offset\_x + j) \* WIDTH;  
 destCoord.Y = offset\_y + i;  
 SetConsoleCursorPosition(hStdout, destCoord);  
 cout << \*(\*(matrix + i) + j + N / 2);  
 }  
 }  
}  
  
int main() {  
 srand(time(nullptr));  
 system("cls");  
  
 int matrixA[N][N], matrixB[N][N], y = 0, x = 0;  
  
 cout << "Исходные матрицы: ";  
 for (int i = 0; i < N \* 2 - 1; ++i) {  
 switch (i % 4) {  
 case 0:  
 y = i / 4;  
 for (x = i / 4; x < N - i / 4; ++x) {  
 set\_element(matrixA, x, y, 0, 1);  
 }  
 break;  
 case 1:  
 x = N - i / 4 - 1;  
 for (y = i / 4 + 1; y < N - i / 4; ++y) {  
 set\_element(matrixA, x, y, 0, 1);  
 }  
 break;  
 case 2:  
 y = N - i / 4 - 1;  
 for (x = N - i / 4 - 2; x >= i / 4; --x) {  
 set\_element(matrixA, x, y, 0, 1);  
 }  
 break;  
 case 3:  
 x = i / 4;  
 for (y = N - i / 4 - 2; y >= i / 4 + 1; --y) {  
 set\_element(matrixA, x, y, 0, 1);  
 }  
 break;  
 }  
 }  
  
 for (int i = 0; i < N; ++i) {  
 if (i % 2 == 0) {  
 for (int j = 0; j < N; ++j) {  
 set\_element(matrixB, i, j, N + 2, 1);  
 }  
 } else {  
 for (int j = N - 1; j >= 0; --j) {  
 set\_element(matrixB, i, j, N + 2, 1);  
 }  
 }  
 }  
  
 destCoord.X = 0;  
 destCoord.Y = N + 2;  
 SetConsoleCursorPosition(hStdout, destCoord);  
  
 cout << "Перестановки первой матрицы: ";  
  
 show\_a(matrixA, 0, N + 3);  
 show\_b(matrixA, N + 1, N + 3);  
 show\_c(matrixA, (N + 1) \* 2, N + 3);  
 show\_d(matrixA, (N + 1) \* 3, N + 3);  
  
 destCoord.X = 0;  
 destCoord.Y = (N + 2) \* 2;  
 SetConsoleCursorPosition(hStdout, destCoord);  
  
 cout << "Перестановки второй матрицы: ";  
  
 show\_a(matrixB, 0, (N + 2) \* 2 + 1);  
 show\_b(matrixB, N + 1, (N + 2) \* 2 + 1);  
 show\_c(matrixB, (N + 1) \* 2, (N + 2) \* 2 + 1);  
 show\_d(matrixB, (N + 1) \* 3, (N + 2) \* 2 + 1);  
  
 destCoord.X = 0;  
 destCoord.Y = (N + 2) \* 3;  
 SetConsoleCursorPosition(hStdout, destCoord);  
  
 system("pause");  
 system("cls");  
  
 cout << "Исходные матрицы: ";  
 print\_matrix(matrixA, 0, 1);  
 print\_matrix(matrixB, N + 1, 1);  
  
 destCoord.X = 0;  
 destCoord.Y = (N + 2) \* 1;  
 SetConsoleCursorPosition(hStdout, destCoord);  
  
 cout << "Отсортированные матрицы: ";  
 quicksort(matrixA, N \* N - 1, 0);  
 quicksort(matrixB, N \* N - 1, 0);  
 print\_matrix(matrixA, 0, (N + 2) \* 1 + 1);  
 print\_matrix(matrixB, N + 1, (N + 2) \* 1 + 1);  
  
 destCoord.X = 0;  
 destCoord.Y = (N + 2) \* 2;  
 SetConsoleCursorPosition(hStdout, destCoord);  
  
 system("pause");  
 system("cls");  
  
 cout << "Модификация всех элементов матрицы" << endl;  
 cout << "Введите число: ";  
 int number;  
 cin >> number;  
  
 char operation;  
 do {  
 cout << "Введите знак операции (+, -, \*, /): ";  
 cin >> operation;  
 } while (operation != '+' && operation != '-' && operation != '\*' && operation != '/');  
  
 system("cls");  
 cout << "Исходные матрицы: ";  
 print\_matrix(matrixA, 0, 1);  
 print\_matrix(matrixB, N + 1, 1);  
  
 for (int i = 0; i < N; ++i) {  
 for (int j = 0; j < N; ++j) {  
 switch (operation) {  
 case '+':  
 \*(\*(matrixA + i) + j) += number;  
 \*(\*(matrixB + i) + j) += number;  
 break;  
 case '-':  
 \*(\*(matrixA + i) + j) -= number;  
 \*(\*(matrixB + i) + j) -= number;  
 break;  
 case '\*':  
 \*(\*(matrixA + i) + j) \*= number;  
 \*(\*(matrixB + i) + j) \*= number;  
 break;  
 case '/':  
 \*(\*(matrixA + i) + j) /= number;  
 \*(\*(matrixB + i) + j) /= number;  
 break;  
 }  
 }  
 }  
  
 destCoord.X = 0;  
 destCoord.Y = (N + 2) \* 1;  
 SetConsoleCursorPosition(hStdout, destCoord);  
  
 cout << "Результат: ";  
 print\_matrix(matrixA, 0, (N + 2) \* 1 + 1);  
 print\_matrix(matrixB, N + 1, (N + 2) \* 1 + 1);  
  
 cout << endl;  
 system("pause");  
 system("cls");  
  
 cout << "Индивидуальное домашнее задание №10" << endl;  
 cout << "Матрица 3x3 со случайными числами от -30 до 30: " << endl;  
  
 int matrixC[3][3];  
 for (int i = 0; i < 3; ++i) {  
 for (int j = 0; j < 3; ++j) {  
 \*(\*(matrixC + i) + j) = rand() % 61 - 30;  
 cout << \*(\*(matrixC + i) + j) << '\t';  
 }  
 cout << endl;  
 }  
 int a = \*(\*(matrixC + 0) + 0) \* \*(\*(matrixC + 1) + 1) \* \*(\*(matrixC + 2) + 2);  
 int b = \*(\*(matrixC + 0) + 1) \* \*(\*(matrixC + 1) + 2) \* \*(\*(matrixC + 2) + 0);  
 int c = \*(\*(matrixC + 0) + 2) \* \*(\*(matrixC + 1) + 0) \* \*(\*(matrixC + 2) + 1);  
 int d = \*(\*(matrixC + 0) + 2) \* \*(\*(matrixC + 1) + 1) \* \*(\*(matrixC + 2) + 0);  
 int e = \*(\*(matrixC + 0) + 0) \* \*(\*(matrixC + 1) + 2) \* \*(\*(matrixC + 2) + 1);  
 int f = \*(\*(matrixC + 0) + 1) \* \*(\*(matrixC + 1) + 0) \* \*(\*(matrixC + 2) + 2);  
 int det = a + b + c - (d + e + f);  
 cout << "Вычисление определителя матрицы..." << endl;  
 Sleep(50);  
 cout << "Произведение элементов главной диагонали: " << a << endl;  
 Sleep(50);  
 cout << "Произведение элементов первого треугольника параллельного главной диагонали: " << b << endl;  
 Sleep(50);  
 cout << "Произведение элементов второго треугольника параллельного главной диагонали: " << c << endl;  
 Sleep(50);  
 cout << "Произведение элементов побочной диагонали: " << d << endl;  
 Sleep(50);  
 cout << "Произведение элементов первого треугольника параллельного побочной диагонали: " << e << endl;  
 Sleep(50);  
 cout << "Произведение элементов второго треугольника параллельного побочной диагонали: " << f << endl;  
 Sleep(50);  
 cout << "Вычисленный определитель: " << det << endl;  
 return 0;  
}