**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 |  | Демьяненко И.А. |
| 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 выбирает какую матрицу и как переставить. | Генерация матрицы змейкой:    Вывод всех перестановок матриц на экран:    Перестановка матрицы одним из способов |
| 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: |
| Сложение матриц | |
| Пользователь видит процесс сложения двух матриц. | Сложение матриц |

**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 = 4;

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

COORD destCoord;

void fillMatrix(int matrix[][N], int N) {

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

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

\*(\*(matrix + j) + i) = rand() % 100;

}

}

}

void printMatrix(int matrix[][N], int N, int offsetx, int offsety) {

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

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

destCoord.X = (i + offsetx) \* width;

destCoord.Y = j + offsety;

SetConsoleCursorPosition(hStdout, destCoord);

cout << \*(\*(matrix + j) + i);

}

}

}

void printFloatMatrix(float matrix[][N], int N, int offsetx, int offsety) {

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

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

destCoord.X = (i + offsetx) \* width;

destCoord.Y = j + offsety;

SetConsoleCursorPosition(hStdout, destCoord);

cout << \*(\*(matrix + j) + i);

}

}

}

void shuffle\_a(int matrix[][N], int newmatrix[][N]) {

for (int i = 0; i < N / 2; i++) {

for (int j = 0; j < N / 2; j++) {

\*(\*(newmatrix + j) + i) = \*(\*(matrix + j + N / 2) + i);

\*(\*(newmatrix + j) + (i + N / 2)) = \*(\*(matrix + j) + i);

\*(\*(newmatrix + (j + N / 2)) + (i + N / 2)) = \*(\*(matrix + j) + i + N / 2);

\*(\*(newmatrix + (j + N / 2)) + i) = \*(\*(matrix + j + N / 2) + i + N / 2);

}

}

}

void shuffle\_b(int matrix[][N], int newmatrix[][N]) {

for (int i = 0; i < N / 2; i++) {

for (int j = 0; j < N / 2; j++) {

\*(\*(newmatrix + j) + i) = \*(\*(matrix + j + N / 2) + i + N / 2);

\*(\*(newmatrix + j) + (i + N / 2)) = \*(\*(matrix + j + N / 2) + i);

\*(\*(newmatrix + (j + N / 2)) + (i + N / 2)) = \*(\*(matrix + j) + i);

\*(\*(newmatrix + (j + N / 2)) + i) = \*(\*(matrix + j) + i + N / 2);

}

}

}

void shuffle\_c(int matrix[][N], int newmatrix[][N]) {

for (int i = 0; i < N / 2; i++) {

for (int j = 0; j < N / 2; j++) {

\*(\*(newmatrix + j) + i) = \*(\*(matrix + j + N / 2) + i);

\*(\*(newmatrix + j) + (i + N/2)) = \*(\*(matrix + j + N / 2) + i + N / 2);

\*(\*(newmatrix + (j + N / 2)) + (i + N / 2)) = \*(\*(matrix + j) + i + N / 2);

\*(\*(newmatrix + (j + N / 2)) + i) = \*(\*(matrix + j) + i);

}

}

}

void shuffle\_d(int matrix[][N], int newmatrix[][N]) {

for (int i = 0; i < N / 2; i++) {

for (int j = 0; j < N / 2; j++) {

\*(\*(newmatrix + j) + i) = \*(\*(matrix + j) + i + N / 2);

\*(\*(newmatrix + j) + (i + N / 2)) = \*(\*(matrix + j) + i);

\*(\*(newmatrix + (j + N / 2)) + (i + N / 2)) = \*(\*(matrix + j + N / 2) + i);

\*(\*(newmatrix + (j + N / 2)) + i) = \*(\*(matrix + j + N / 2) + i + N / 2);

}

}

}

void quicksort(int arr[][N], int begin, int end) {

int b = begin;

int e = end;

int midl = (b + e) / 2;

int mid = (\*(\*(arr + midl / N) + midl % N));

while (b < e) {

while (\*(\*(arr + b / N) + b % N) < mid) b++;

while (\*(\*(arr + e / N) + e % N) > mid) e--;

if (b <= e) {

swap(\*(\*(arr + b / N) + b % N), \*(\*(arr + e / N) + e % N));

b++;

e--;

}

}

if (begin < e) quicksort(arr, begin, e);

if (e < end) quicksort(arr, b, end);

}

int main() {

srand(time(NULL));

setlocale(0, "");

int matrixA[N][N], matrixB[N][N], shufflematrix[N][N];

fillMatrix(matrixA, N);

cout << "1) Квадратичные матрицы (A и B): " << endl;

for (int i = 0; i <= N \* 2 - 1; i++) {

int a = i % 4;

switch (a) {

case(0):

for (int x = i / 4; x < N - i / 4; x++) {

int y = i / 4;

destCoord.X = x \* width;

destCoord.Y = y+1;

SetConsoleCursorPosition(hStdout, destCoord);

cout << \*(\*(matrixA + y) + x);

Sleep(10);

}

break;

case(1):

for (int y = i / 4 + 1; y < N - i / 4; y++) {

int x = N - i / 4 - 1;

destCoord.X = x \* width;

destCoord.Y = y+1;

SetConsoleCursorPosition(hStdout, destCoord);

cout << \*(\*(matrixA + y) + x);

Sleep(10);

}

break;

case(2):

for (int x = N - i / 4 - 2; x > 0 + i / 4; x--) {

int y = N - i / 4 - 1;

destCoord.X = x \* width;

destCoord.Y = y+1;

SetConsoleCursorPosition(hStdout, destCoord);

cout << \*(\*(matrixA + y) + x);

Sleep(10);

}

break;

case(3):

for (int y = N - i / 4 - 1; y > 0 + i / 4; y--) {

int x = i / 4;

destCoord.X = x \* width;

destCoord.Y = y+1;

SetConsoleCursorPosition(hStdout, destCoord);

cout << \*(\*(matrixA + y) + x);

Sleep(10);

}

break;

}

}

fillMatrix(matrixB, N);

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

int a = i % 2;

switch (a) {

case(0):

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

destCoord.X = i\*width + N\*width+3;

destCoord.Y = y + 1;

SetConsoleCursorPosition(hStdout, destCoord);

cout << \*(\*(matrixB + y) + i);

Sleep(10);

}

break;

case(1):

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

destCoord.X = i\*width + N \* width + 3;

destCoord.Y = y + 1;

SetConsoleCursorPosition(hStdout, destCoord);

cout << \*(\*(matrixB + y) + i);

Sleep(10);

}

break;

}

}

destCoord.X = 0;

destCoord.Y = N + 2;

SetConsoleCursorPosition(hStdout, destCoord);

cout << "2) Перестановки блоков матрицы" << "\n" << "Выберите матрицу (A or B): ";

char choice;

cin >> choice;

do

{

switch (choice)

{

case 'A':

cout << "Выберите способ перестановки (a, b, c, d): ";

char shufchoiceA;

cin >> shufchoiceA;

switch (shufchoiceA)

{

case 'a':

shuffle\_a(matrixA, shufflematrix);

printMatrix(shufflematrix, N, 0, N + 6);

choice = 0;

cout << endl;

break;

case 'b':

shuffle\_b(matrixA, shufflematrix);

printMatrix(shufflematrix, N, 0, N + 6);

choice = 0;

cout << endl;

break;

case 'c':

shuffle\_c(matrixA, shufflematrix);

printMatrix(shufflematrix, N, 0, N + 6);

choice = 0;

cout << endl;

break;

case 'd':

shuffle\_d(matrixA, shufflematrix);

printMatrix(shufflematrix, N, 0, N + 6);

choice = 0;

cout << endl;

break;

default:

cout << "Некорректный ввод" << endl;

choice = 0;

break;

}

break;

case 'B':

cout << "Выберите способ перестановки (a, b, c ,d): ";

char shufchoiceB;

cin >> shufchoiceB;

switch (shufchoiceB)

{

case 'a':

shuffle\_a(matrixB, shufflematrix);

printMatrix(shufflematrix, N, 0, N + 6);

choice = 0;

cout << endl;

break;

case 'b':

shuffle\_b(matrixB, shufflematrix);

printMatrix(shufflematrix, N, 0, N + 6);

choice = 0;

cout << endl;

break;

case 'c':

shuffle\_c(matrixB, shufflematrix);

printMatrix(shufflematrix, N, 0, N + 6);

choice = 0;

cout << endl;

break;

case 'd':

shuffle\_d(matrixB, shufflematrix);

printMatrix(shufflematrix, N, 0, N + 6);

choice = 0;

cout << endl;

break;

default:

cout << "Некорректный ввод" << endl;

choice = 0;

break;

}

break;

default:

cout << "Некорректный ввод" << endl;

choice = 0;

break;

}

} while (choice != 0);

system("pause");

system("cls");

destCoord.X = 0;

destCoord.Y = 0;

SetConsoleCursorPosition(hStdout, destCoord);

cout << "3) Отсортированная матрицы A и B:";

quicksort(matrixA, 0, N \* N - 1);

quicksort(matrixB, 0, N\* N - 1);

printMatrix(matrixA, N, 0, 1);

printMatrix(matrixB, N, N+1, 1);

destCoord.X = 0;

destCoord.Y = N + 2;

SetConsoleCursorPosition(hStdout, destCoord);

cout << "4) Введите знак операции: ";

char opsign; int number; float matrixfornull[N][N];

cin >> opsign;

if (opsign != '+' && opsign != '-' && opsign != '\*' && opsign != '/') {

cout << "Неверный знак. Матрицы остаются без изменения.";

}

else {

cout << "Введите число: ";

cin >> number;

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

{

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

switch (opsign)

{

case '+':

\*(\*(matrixA + j) + i) += number;

\*(\*(matrixB + j) + i) += number;

break;

case '-':

\*(\*(matrixA + j) + i) -= number;

\*(\*(matrixB + j) + i) -= number;

break;

case '\*':

\*(\*(matrixA + j) + i) \*= number;

\*(\*(matrixB + j) + i) \*= number;

break;

case '/':

if (number == 0) {

\*(\*(matrixfornull + j) + i) /= number;

}

else {

\*(\*(matrixA + j) + i) /= number;

\*(\*(matrixB + j) + i) /= number;

}

break;

}

}

}

}

if (opsign == '/' && number == 0)

{

printFloatMatrix(matrixfornull, N, 0, N + 5);

printFloatMatrix(matrixfornull, N, N + 1, N + 5);

}

else {

printMatrix(matrixA, N, 0, N + 5);

printMatrix(matrixB, N, N + 1, N + 5);

}

cout << endl;

system("pause");

system("cls");

destCoord.X = 0;

destCoord.Y = 0;

SetConsoleCursorPosition(hStdout, destCoord);

// ИДЗ 1

int IDZmatrix[N][N];

fillMatrix(IDZmatrix, N);

cout << "Сложение двух матриц";

printMatrix(matrixA, N, 0, 1);

destCoord.X = (N \* width) + 2;

destCoord.Y = N/2;

SetConsoleCursorPosition(hStdout, destCoord);

cout << '+';

printMatrix(IDZmatrix, N, N + 2, 1);

destCoord.X = (2\*N \* width) + 2\*width;

destCoord.Y = N / 2;

SetConsoleCursorPosition(hStdout, destCoord);

cout << '=';

destCoord.X = 0;

destCoord.Y = N + 1 + N / 2;

SetConsoleCursorPosition(hStdout, destCoord);

cout << '=';

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

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

destCoord.X = i \* (width\*2 + 1) + 2;

destCoord.Y = (N+2)+j;

SetConsoleCursorPosition(hStdout, destCoord);

cout << \*(\*(matrixA + j) + i) << "+" << \*(\*(IDZmatrix + j) + i);

}

}

destCoord.X = N \* (width \* 2 + 1) + 3;

destCoord.Y = N + 1 + N / 2;

SetConsoleCursorPosition(hStdout, destCoord);

cout << '=';

destCoord.X = 0;

destCoord.Y = N + N + 2 + N / 2;

SetConsoleCursorPosition(hStdout, destCoord);

cout << '=';

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

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

destCoord.X = (i+1) \* width;

destCoord.Y = (N + N + 3) + j;

SetConsoleCursorPosition(hStdout, destCoord);

cout << \*(\*(matrixA + j) + i) + \*(\*(IDZmatrix + j) + i);

}

}

}