**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(ка) гр. | Цатинян А.А. |  |
| Преподаватель | Глущенко А. Г. |  |

Санкт-Петербург

2022

**Цель работы.**

Изучение двумерных статических массивов, указателей, арифметики указателей.

**Основные теоретические положения.**

Указатели и ссылки являются одними из самых важных и достаточно сложных для понимания и использования средств языка программирования. Они ориентированы на прямую работу с памятью компьютера. С помощью этих средств реализуется работа с динамической памятью и динамическими объектами, возвращение из функций измененных данных и многое другое. Массив представляет собой индексированную последовательность однотипных элементов с заранее определенным количеством элементов. Наглядно одномерный массив можно представить, как набор пронумерованных ячеек, в каждой из которых содержится определенное значение.

Элементы массива нумеруются с нуля. При описании массива используются те же модификаторы (класс памяти, const и инициализатор), что и для простых переменных.

Сортировка – процесс размещения элементов заданного множества объектов в определенном порядке. Когда элементы отсортированы, их проще найти, производить с ними различные операции. Сортировка напрямую влияет на скорость алгоритма, в котором нужно обратиться к определенному элементу массива.

**Указатели** – это тоже обычные переменные, но они **служат для хранения адресов памяти**.

Нулевой указатель (null pointer) — это указатель, который не указывает ни на какой объект. Если мы не хотим, чтобы указатель указывал на какой-то конкретный адрес, то можно присвоить ему условное нулевое значение.

Указатель хранит адрес переменной, и по этому адресу мы можем получить значение этой переменной. Но кроме того, указатель, как и любая переменная, сам имеет адрес, по которому он располагается в памяти. Этот адрес можно получить также через операцию &. К указателям могут применяться операции сравнения >, >=, <, <=, ==, !=. Операции сравнения применяются только к указателям одного типа и к значениям NULL и nullptr.

К указателям можно применять некоторые арифметические операции. К таким операциям относятся:  **+**,**-**, **++**, **--**. Результаты выполнения этих операций по отношению к указателям существенно отличаются от результатов соответствующих арифметических операций, выполняющихся с обычными числовыми данными. Указатели – это очень мощное, полезное, но и очень опасное средство. Ошибки, которые возникают при неправильном использовании указателей, кроме того, что могут приводить к серьезным и непредсказуемым ошибкам в работе программы, еще и очень трудно диагностировать (обнаруживать). Основная и наиболее часто встречающаяся ошибка при работе с указателями связана с использованием неинициализированных указателей.

**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 согласно схемам, приведенным на рисунках. Пользователь должен видеть процесс заполнения квадратичной матрицы.



2)    Получает новую матрицу, из матрицы п. 1, переставляя ее блоки в соответствии со схемами:



3)    Используя арифметику указателей, сортирует элементы любой сортировкой.

4)    Уменьшает, увеличивает, умножает или делит все элементы матрицы на введенное пользователем число.

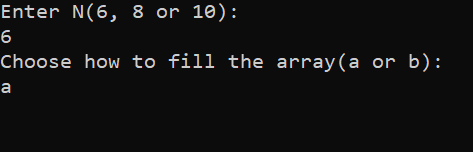
**Выполнение работы.**

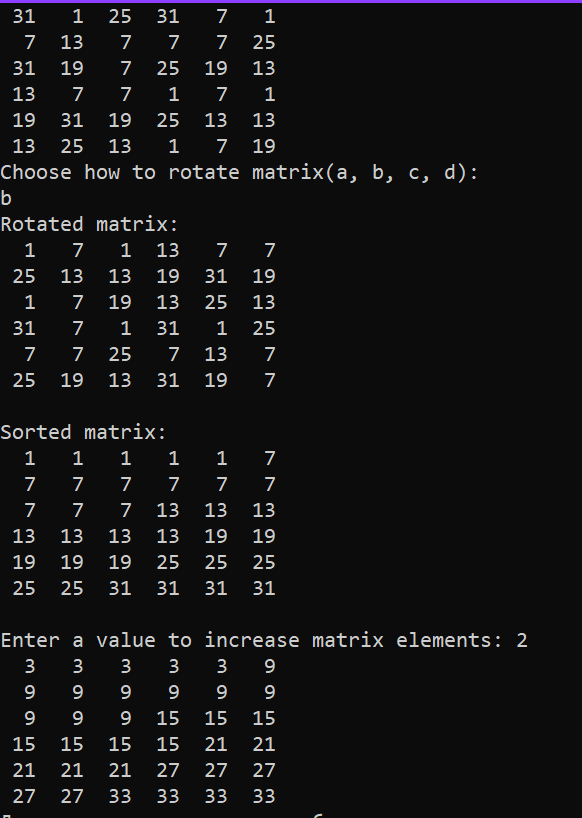
Код программы представлен в приложении А.

Блок описания кода и использованных алгоритмов

Используя арифметику указателей, производится заполнение матрицы двумя способами. При перемещении блоков матрицы, каждый блок сохраняется в отдельный массив, после чего исходная матрицы собирается заново, но уже в нужном порядке. При сортировке матрицы, все элементы переписываются в одномерный массив, сортируются, после чего заново строится матрица. При увеличении или уменьшении всех элементов, происходит последовательный проход по всем элементам матрицы и изменяются их элементы.

Блок скриншотов работы программы





**Выводы.**

Изучен навык работы с двумерными статическими массивами. Изучен навык работы с указателями, основы арифметики указателей. Изучена базовая связь между массивами и указателями, способы обращения к элементам массивов через указатели.

Приложение А

рабочий код

#include <iostream>  
#include <thread>  
#include <iomanip>  
  
*using namespace* std;  
  
*void* printMatrix(*int* \*\*, *int*, *bool* = 1);  
  
*void* clearMatrix(*int* \*\*, *int*);  
  
*void* spiralMatrix(*int* \*\*, *int*);  
  
*void* snakeMatrix(*int* \*\*, *int*);  
  
*void* rotateMatrix(*int* \*\*, *int*, *char*);  
  
*void* sortMatrix(*int* \*\*, *int*);  
  
*void* increaseMatrix(*int* \*\*, *int*, *int*);  
  
*int* main() {  
 srand(time(0));  
 *int* N;  
 cout << "Enter N(6, 8 or 10):" << "\n";  
 cin >> N;  
 *while* (N != 6 && N != 8 && N != 10) {  
 cout << "Wrong N value, try again" << "\n";  
 cin >> N;  
 }  
 *int* \*\*matrix = *new int* \*[N];  
 *for* (*int* \*\*cur = matrix, \*\*end = cur + N - 1; cur <= end; cur++) {  
 \*cur = *new int*[N];  
 }  
 string fillType;  
 cout << "Choose how to fill the array(a or b):" << endl;  
 cin >> fillType;  
 *while* (fillType != "a" && fillType != "b") {  
 cout << "Wrong value, try again(a, b)" << endl;  
 cin >> fillType;  
 }  
 clearMatrix(matrix, N);  
 *if* (fillType == "a") {  
 spiralMatrix(matrix, N);  
 } *else* snakeMatrix(matrix, N);  
  
 string rotateOption;  
 cout << "Choose how to rotate matrix(a, b, c, d):" << endl;  
 cin >> fillType;  
 *while* (fillType != "a" && fillType != "b" && fillType != "c" && fillType != "d") {  
 cout << "Wrong value, try again(a, b)" << endl;  
 cin >> fillType;  
 }  
 cout << "Rotated matrix: " << '\n';  
 rotateMatrix(matrix, N, fillType[0]);  
 cout << '\n' << "Sorted matrix: " << '\n';  
 sortMatrix(matrix, N);  
 cout << '\n' << "Enter a value to increase matrix elements: ";  
 *int* k;  
 cin >> k;  
 increaseMatrix(matrix, N, k);  
 system("Pause");  
}  
  
*void* printMatrix(*int* \*\*matrix, *int* N, *bool* clear) {  
 *if* (clear) system("cls");  
 *for* (*int* \*\*curi = matrix, \*\*endi = curi + N - 1; curi <= endi; curi++) {  
 *for* (*int* \*curj = \*curi, \*endj = curj + N - 1; curj <= endj; curj++) {  
 cout << setw(3) << \*curj;  
 *if* (curj != endj) cout << " ";  
 }  
 cout << "\n";  
 }  
 std::this\_thread::sleep\_for(0.1s);  
}  
  
*void* clearMatrix(*int* \*\*matrix, *int* N) {  
 *for* (*int* \*\*curi = matrix, \*\*endi = curi + N - 1; curi <= endi; curi++) {  
 *for* (*int* \*curj = \*curi, \*endj = curj + N - 1; curj <= endj; curj++) {  
 \*curj = 0;  
 }  
 }  
}  
  
*void* spiralMatrix(*int* \*\*matrix, *int* N) {  
 printMatrix(matrix, N);  
 *int* lineNum = 0;  
 *for* (*int* j = 0; j < N / 2; ++j) {  
 *for* (*int* \*curi = \*(matrix + lineNum) + lineNum, \*endi = curi + N - 2 \* lineNum; curi < endi; curi++) {  
 \*curi = rand() % N \* N + 1;  
 printMatrix(matrix, N);  
 }  
 *for* (*int* \*\*curi = matrix + lineNum + 1, \*\*endi = matrix + N - 2 - lineNum; curi <= endi; curi++) {  
 \*(\*curi + N - lineNum - 1) = rand() % N \* N + 1;  
 printMatrix(matrix, N);  
 }  
 *for* (*int* \*curi = \*(matrix + N - lineNum - 1) + N - lineNum - 1,  
 \*endi = curi - N + 1 + 2 \* lineNum; curi > endi; curi--) {  
 \*curi = rand() % N \* N + 1;  
 printMatrix(matrix, N);  
 }  
 *for* (*int* \*\*curi = matrix + N - 1 - lineNum, \*\*endi = matrix + lineNum + 1; curi >= endi; curi--) {  
 \*(\*curi + lineNum) = rand() % N \* N + 1;  
 printMatrix(matrix, N);  
 }  
 lineNum++;  
 }  
}  
  
*void* snakeMatrix(*int* \*\*matrix, *int* N) {  
 printMatrix(matrix, N);  
 *for* (*int* lineNum = 0; lineNum < N / 2; ++lineNum) {  
 *for* (*int* \*\*curi = matrix, \*\*endi = matrix + N - 1; curi <= endi; curi++) {  
 \*(\*curi + lineNum \* 2) = rand() % N \* N + 1;  
 printMatrix(matrix, N);  
 }  
 *for* (*int* \*\*curi = matrix + N - 1, \*\*endi = matrix; curi >= endi; curi--) {  
 \*(\*curi + lineNum \* 2 + 1) = rand() % N \* N + 1;  
 printMatrix(matrix, N);  
 }  
 }  
}  
  
*void* rotateMatrix(*int* \*\*matrix, *int* N, *char* option) {  
 *int* \*\*a = *new int* \*[N / 2], \*\*b = *new int* \*[N / 2],  
 \*\*c = *new int* \*[N / 2], \*\*d = *new int* \*[N / 2], \*\*\*arr = *new int* \*\*[4];  
 arr[0] = a;  
 arr[1] = b;  
 arr[2] = c;  
 arr[3] = d;  
 *for* (*int* \*\*\*i = arr, \*\*\*iEnd = arr + 3; i <= iEnd; i++) {  
 *for* (*int* \*\*cur = \*i, \*\*end = cur + N / 2 - 1; cur <= end; cur++) {  
 \*cur = *new int*[N / 2];  
 }  
 }  
 *for* (*int* \*\*curi = matrix, \*\*endi = curi + N / 2 - 1, \*\*curArray = a; curi <= endi; curi++, curArray++) {  
 *for* (*int* \*curj = \*curi, \*endj = curj + N / 2 - 1, \*curArrayJ = \*curArray; curj <= endj; curj++, curArrayJ++) {  
 \*curArrayJ = \*curj;  
 }  
 }  
 *for* (*int* \*\*curi = matrix, \*\*endi = curi + N / 2 - 1, \*\*curArray = b; curi <= endi; curi++, curArray++) {  
 *for* (*int* \*curj = \*curi + N / 2, \*endj = curj + N / 2 - 1, \*curArrayJ = \*curArray;  
 curj <= endj; curj++, curArrayJ++) {  
 \*curArrayJ = \*curj;  
 }  
 }  
 *for* (*int* \*\*curi = matrix + N / 2, \*\*endi = curi + N / 2 - 1, \*\*curArray = c; curi <= endi; curi++, curArray++) {  
 *for* (*int* \*curj = \*curi, \*endj = curj + N / 2 - 1, \*curArrayJ = \*curArray; curj <= endj; curj++, curArrayJ++) {  
 \*curArrayJ = \*curj;  
 }  
 }  
 *for* (*int* \*\*curi = matrix + N / 2, \*\*endi = curi + N / 2 - 1, \*\*curArray = d; curi <= endi; curi++, curArray++) {  
 *for* (*int* \*curj = \*curi + N / 2, \*endj = curj + N / 2 - 1, \*curArrayJ = \*curArray;  
 curj <= endj; curj++, curArrayJ++) {  
 \*curArrayJ = \*curj;  
 }  
 }  
 *int* \*\*pa, \*\*pb, \*\*pc, \*\*pd;  
 *if* (option == 'a') {  
 pa = c; pb = a; pc = d; pd = b;  
 } *else if* (option == 'b') {  
 pa = d; pb = c; pc = b; pd = a;  
 } *else if* (option == 'c') {  
 pa = c; pb = d; pc = a; pd = b;  
 } *else* {  
 pa = b; pb = a; pc = d; pd = c;  
 }  
 *for* (*int* \*\*curi = matrix, \*\*endi = curi + N / 2 - 1, \*\*curArray = pa; curi <= endi; curi++, curArray++) {  
 *for* (*int* \*curj = \*curi, \*endj = curj + N / 2 - 1, \*curArrayJ = \*curArray; curj <= endj; curj++, curArrayJ++) {  
 \*curj = \*curArrayJ;  
 }  
 }  
 *for* (*int* \*\*curi = matrix, \*\*endi = curi + N / 2 - 1, \*\*curArray = pb; curi <= endi; curi++, curArray++) {  
 *for* (*int* \*curj = \*curi + N / 2, \*endj = curj + N / 2 - 1, \*curArrayJ = \*curArray;  
 curj <= endj; curj++, curArrayJ++) {  
 \*curj = \*curArrayJ;  
 }  
 }  
 *for* (*int* \*\*curi = matrix + N / 2, \*\*endi = curi + N / 2 - 1, \*\*curArray = pc; curi <= endi; curi++, curArray++) {  
 *for* (*int* \*curj = \*curi, \*endj = curj + N / 2 - 1, \*curArrayJ = \*curArray; curj <= endj; curj++, curArrayJ++) {  
 \*curj = \*curArrayJ;  
 }  
 }  
 *for* (*int* \*\*curi = matrix + N / 2, \*\*endi = curi + N / 2 - 1, \*\*curArray = pd; curi <= endi; curi++, curArray++) {  
 *for* (*int* \*curj = \*curi + N / 2, \*endj = curj + N / 2 - 1, \*curArrayJ = \*curArray;  
 curj <= endj; curj++, curArrayJ++) {  
 \*curj = \*curArrayJ;  
 }  
 }  
 printMatrix(matrix, N, 0);  
}  
  
*void* sortMatrix(*int* \*\*matrix, *int* N) {  
 *int* \*arr = *new int* [N \* N];  
 *int* i = 0;  
 *for* (*int* \*\*curi = matrix, \*\*endi = curi + N - 1; curi <= endi; curi++) {  
 *for* (*int* \*curj = \*curi, \*endj = curj + N - 1; curj <= endj; curj++) {  
 \*(arr + i) = \*curj;  
 i++;  
 }  
 }  
 *bool* isChanged = 1;  
 *while* (isChanged) {  
 isChanged = 0;  
 *for* (*int* \*j = arr, \*jEnd = j + (N \* N) - 2; j <= jEnd; j++) {  
 *if* (\*j > \*(j + 1)) {  
 isChanged = 1;  
 *int* t = \*(j + 1);  
 \*(j + 1) = \*j;  
 \*j = t;  
 }  
 }  
 }  
  
 i = 0;  
 *for* (*int* \*\*curi = matrix, \*\*endi = curi + N - 1; curi <= endi; curi++) {  
 *for* (*int* \*curj = \*curi, \*endj = curj + N - 1; curj <= endj; curj++) {  
 \*curj = \*(arr + i);  
 i++;  
 }  
 }  
  
 printMatrix(matrix, N, 0);  
}  
  
*void* increaseMatrix(*int* \*\*matrix, *int* N, *int* k) {  
 *for* (*int* \*\*curi = matrix, \*\*endi = curi + N - 1; curi <= endi; curi++) {  
 *for* (*int* \*curj = \*curi, \*endj = curj + N - 1; curj <= endj; curj++) {  
 \*curj += k;  
 }  
 }  
 printMatrix(matrix, N, 0);  
}