**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. | Ковальчук И.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.

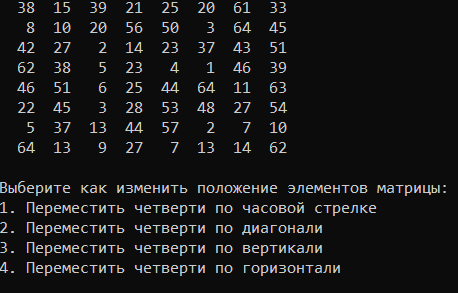
**Выполнение работы.**

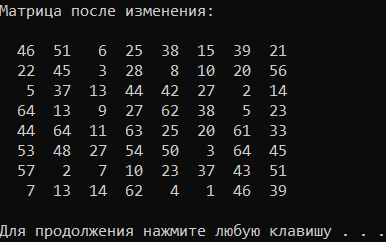
Код программы представлен в приложении А.

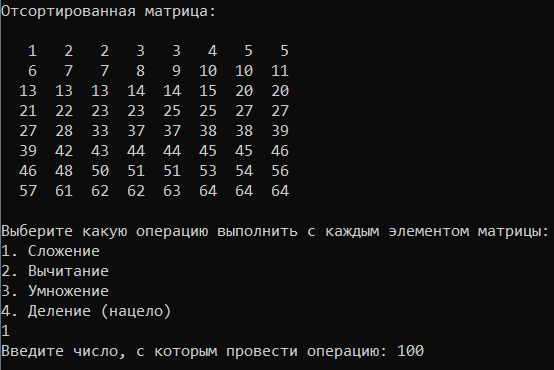
Блок описания кода и использованных алгоритмов

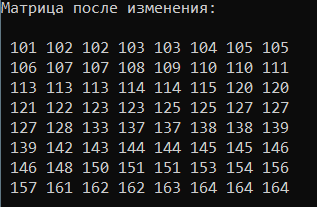
Используя арифметику указателей, производится заполнение матрицы двумя способами. При перемещении блоков матрицы, каждый блок сохраняется в отдельный массив, после чего исходная матрицы собирается заново, но уже в нужном порядке. При сортировке матрицы, все элементы переписываются в одномерный массив, 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 <conio.h>

#include <iomanip>

#include <windows.h>

#include <thread>

using namespace std;

void gotoxy(int x, int y)

{

COORD coord;

coord.X = x;

coord.Y = y;

SetConsoleCursorPosition(GetStdHandle(STD\_OUTPUT\_HANDLE), coord);

}

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

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

if ((i / N) % 2 == 1) {

gotoxy(4 \* (i / N), (N - (i % N)) - 1);

cout << setw(4) << \*(\*(matrix + (N - (i % N) - 1)) + (i / N)); //\*( \*(arr+ряд) + столбец ) ) )

}

else {

gotoxy(4 \* (i / N), (i % N));

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

}

this\_thread::sleep\_for(0.1s);

}

gotoxy(0, N);

}

void upperCorner(int N, int len, int\*\* matrix) {

int x = 0, y = 0;

int margin = (N - len) / 2;

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

if (i <= len - 1) {

gotoxy(4 \* (i + margin), margin);

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

}

else {

gotoxy(4 \* (len - 1 + margin), i - len + 1 + margin);

cout << setw(4) << \*(\*(matrix + (i-len+1+margin)) + (len - 1 + margin));

}

this\_thread::sleep\_for(0.1s);

}

}

void lowerCorner(int N, int len, int\*\* matrix) {

int x = 0, y = 0;

int margin = (N - len) / 2;

for (int i = len \* 2 - 1; i >= 1; i--) {

if (i >= len) {

gotoxy(4 \* ((i % len) + margin), len + margin);

cout << setw(4) << \*( \*(matrix + (len + margin)) + (i%len)+margin);

}

else {

gotoxy(4 \* margin, i + margin);

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

}

this\_thread::sleep\_for(0.1s);

}

}

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

for (int len = N; len > 0; len--) {

if (len % 2 == 0) upperCorner(N, len, matrix);

if (len % 2 == 1) lowerCorner(N, len, matrix);

}

gotoxy(0, N);

}

void increaseMatrix(int\*\* matrix, int N, int k) {

for (int\*\* rows = matrix, \*\*endrows = rows + N - 1; rows <= endrows; rows++)

for (int\* curcol = \*rows, \*endcol = curcol + N - 1; curcol <= endcol; curcol++)

\*curcol += k;

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

gotoxy(4 \* (i % N), (i / N) + 2);

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

}

gotoxy(0, N + 2);

}

void decreaseMatrix(int\*\* matrix, int N, int k) {

for (int\*\* rows = matrix, \*\*endrows = rows + N - 1; rows <= endrows; rows++)

for (int\* curcol = \*rows, \*endcol = curcol + N - 1; curcol <= endcol; curcol++)

\*curcol -= k;

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

gotoxy(4 \* (i % N), (i / N) + 2);

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

}

gotoxy(0, N + 2);

}

void multiplyMatrix(int\*\* matrix, int N, int k) {

for (int\*\* rows = matrix, \*\*endrows = rows + N - 1; rows <= endrows; rows++)

for (int\* curcol = \*rows, \*endcol = curcol + N - 1; curcol <= endcol; curcol++)

\*curcol \*= k;

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

gotoxy(4 \* (i % N), (i / N) + 2);

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

}

gotoxy(0, N + 2);

}

void divideMatrix(int\*\* matrix, int N, int k) {

for (int\*\* rows = matrix, \*\*endrows = rows + N - 1; rows <= endrows; rows++)

for (int\* curcol = \*rows, \*endcol = curcol + N - 1; curcol <= endcol; curcol++)

\*curcol /= k;

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

gotoxy(4 \* (i % N), (i / N) + 2);

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

}

gotoxy(0, N + 2);

}

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

int\*\* NW = new int\* [N / 2], \*\* NE = new int\* [N / 2], \*\* SE = new int\* [N / 2], \*\* SW = new int\* [N / 2];

int\*\*\* newMatrix = new int\*\* [4] {NW, NE, SE, SW};

for (int\*\*\* nM = newMatrix, \*\*\*nMend = nM + 3; nM <= nMend; nM++) {

for (int\*\* curBlock = \*nM, \*\*curBlockend = curBlock + N / 2; curBlock < curBlockend; curBlock++)

\*curBlock = new int[N / 2];

}

for (int\*\* rows = matrix, \*\*endrows = rows + N / 2, \*\*newRow = NW; rows < endrows; rows++, newRow++) {

for (int\* col = \*rows, \*endcol = col + N / 2, \*newCol = \*newRow; col < endcol; col++, newCol++) {

\*newCol = \*col;

}

}

for (int\*\* rows = matrix, \*\*endrows = rows + N / 2, \*\*newRow = NE; rows < endrows; rows++, newRow++) {

for (int\* col = \*rows + N / 2, \*endcol = col + N / 2, \*newCol = \*newRow; col < endcol; col++, newCol++) {

\*newCol = \*col;

}

}

for (int\*\* rows = matrix + N / 2, \*\*endrows = rows + N / 2, \*\*newRow = SW; rows < endrows; rows++, newRow++) {

for (int\* col = \*rows, \*endcol = col + N / 2, \*newCol = \*newRow; col < endcol; col++, newCol++) {

\*newCol = \*col;

}

}

for (int\*\* rows = matrix + N / 2, \*\*endrows = rows + N / 2, \*\*newRow = SE; rows < endrows; rows++, newRow++) {

for (int\* col = \*rows + N / 2, \*endcol = col + N / 2, \*newCol = \*newRow; col < endcol; col++, newCol++) {

\*newCol = \*col;

}

}

cout << "\nВыберите как изменить положение элементов матрицы:\n1. Переместить четверти по часовой стрелке\n2. Переместить четверти по диагонали\n3. Переместить четверти по вертикали\n4. Переместить четверти по горизонтали" << endl;

int option;

cin >> option;

int\*\* newNW, \*\* newNE, \*\* newSW, \*\* newSE;

if (option == 1) {

newNW = SW;

newNE = NW;

newSW = SE;

newSE = NE;

}

else if (option == 2) {

newNW = SE;

newNE = SW;

newSW = NE;

newSE = NW;

}

else if (option == 3) {

newNW = SW;

newNE = SE;

newSW = NW;

newSE = NE;

}

else {

newNW = NE;

newNE = NW;

newSW = SE;

newSE = SW;

}

for (int\*\* rows = matrix, \*\*endrows = rows + N / 2, \*\*newRow = newNW; rows < endrows; rows++, newRow++) {

for (int\* col = \*rows, \*endcol = col + N / 2, \*newCol = \*newRow; col < endcol; col++, newCol++) {

\*col = \*newCol;

}

}

for (int\*\* rows = matrix, \*\*endrows = rows + N / 2, \*\*newRow = newNE; rows < endrows; rows++, newRow++) {

for (int\* col = \*rows + N / 2, \*endcol = col + N / 2, \*newCol = \*newRow; col < endcol; col++, newCol++) {

\*col = \*newCol;

}

}

for (int\*\* rows = matrix + N / 2, \*\*endrows = rows + N / 2, \*\*newRow = newSW; rows < endrows; rows++, newRow++) {

for (int\* col = \*rows, \*endcol = col + N / 2, \*newCol = \*newRow; col < endcol; col++, newCol++) {

\*col = \*newCol;

}

}

for (int\*\* rows = matrix + N / 2, \*\*endrows = rows + N / 2, \*\*newRow = newSE; rows < endrows; rows++, newRow++) {

for (int\* col = \*rows + N / 2, \*endcol = col + N / 2, \*newCol = \*newRow; col < endcol; col++, newCol++) {

\*col = \*newCol;

}

}

system("cls");

cout << "Матрица после изменения: " << '\n';

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

gotoxy(4 \* (i % N), (i / N) + 2);

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

}

gotoxy(0, N+3);

}

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

int n = N \* N;

int\* sortArr = new int[n];

int\* adr;

adr = sortArr;

for (int\*\* rows = matrix, \*\*endrows = rows + N - 1; rows <= endrows; rows++) {

for (int\* col = \*rows, \*endcol = col + N - 1; col <= endcol; col++, sortArr++) {

\*sortArr = \*col;

}

}

sortArr = adr;

for (; n != 1; n--) {

for (int\* cur = sortArr, \*endarr = cur + n - 2; cur <= endarr; cur++) {

if (\*cur > \*(cur + 1)) {

swap(\*cur, \*(cur + 1));

}

}

}

for (int\*\* rows = matrix, \*\*rowsend = rows + N; rows < rowsend; rows++) {

for (int\* col = \*rows, \*colend = col + N; col < colend; col++) {

\*col = \*sortArr++;

}

}

sortArr = adr;

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

gotoxy(4 \* (i % N), (i / N)+2);

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

}

gotoxy(0, N + 2);

}

int main() {

srand(time(0));

setlocale(0, "");

int N;

cout << "Выберите размерность матрицы: 6, 8, 10:" << "\n";

cin >> N;

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

for (int\*\* row = matrix, \*\*endrow = row + N; row < endrow; row++) {

\*row = new int[N];

}

for (int\*\* row = matrix, \*\*endrow = row + N; row < endrow; row++) {

for (int\* col = \*row, \*endcol = col + N; col < endcol; col++) {

\*col = rand() % (N \* N) + 1;

}

}

int filling;

cout << "Выберите как заполнить матрицу:\n1. Спиралью\n2. \"Змейкой\"" << endl;

cin >> filling;

if (filling == 1) {

system("cls");

spiralPrint(N, matrix);

}

else {

system("cls");

snakePrint(N, matrix);;

}

replaceBlocks(N, matrix);

system("Pause");

system("cls");

cout << "Отсортированная матрица:";

sortMatrix(N, matrix);

cout << '\n' << "Выберите какую операцию выполнить с каждым элементом матрицы:\n1. Сложение\n2. Вычитание\n3. Умножение\n4. Деление (нацело)\n";

int operation;

cin >> operation;

cout << "Введите число, с которым провести операцию: ";

int k;

cin >> k;

system("cls");

cout << "Матрица после изменения: " << '\n';

switch (operation) {

case 1:

increaseMatrix(matrix, N, k);

break;

case 2:

decreaseMatrix(matrix, N, k);

break;

case 3:

multiplyMatrix(matrix, N, k);

break;

case 4:

divideMatrix(matrix, N, k);

break;

default:

break;

}

}