**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. 1323 |  | Чекстер 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.

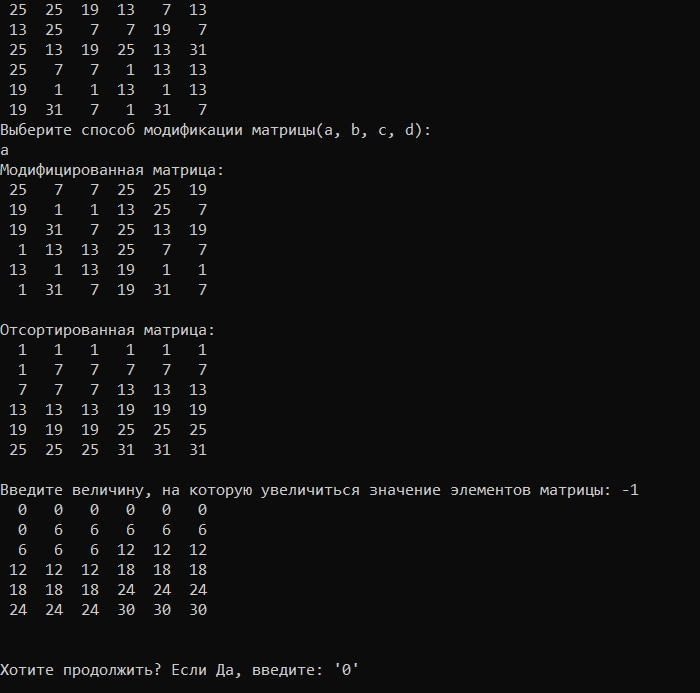
Блок описания кода и использованных алгоритмов

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 описания кода и использованных алгоритмов:**

1. Создаём (.cpp) файл курсовой работы. В нём реализовываем меню с возможностью выбора практической работы. В каждом пункте идёт обращение к каждому отдельному файлу.
2. Создаём общий (.h) файл и объявляем функции, которые в дальнейшем будем прописывать в отдельных файлах.
3. Создаём четыре файла (.cpp). В каждом реализовываем практические работы. Во всех (.cpp) файлах прописываем «#include "Practice.h"», чтобы связать файлы между собой.

**Блок скриншотов работы программы**



**Вывод.**

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 "Prctice.h"

#include <iostream>

#include <thread>

#include <iomanip>

using namespace std;

void print(int\*\*, int, bool = 1);

void clear(int\*\*, int);

void spiral(int\*\*, int);

void snake(int\*\*, int);

void modify(int\*\*, int, char);

void sort(int\*\*, int);

void increase(int\*\*, int, int);

void practice3() {

setlocale(0, "");

int snachala;

while (true) {

srand(time(0));

int N;

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

cin >> N;

while (N != 6 && N != 8 && N != 10) {

cout << "Размерность не подхолит, попробуйте ещё раз" << "\n";

cin >> N;

}

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

for (int\*\* new\_m = matrix, \*\*end = new\_m + N - 1; new\_m <= end; new\_m++)

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

string Type;

cout << "Выберете способ заполнения (a-спиралью or b-змейкой):" << endl;

cin >> Type;

while (Type != "a" && Type != "b") {

cout << "Не подхолит, попробуйте ещё раз(a, b)" << endl;

cin >> Type;

}

clear(matrix, N);

if (Type == "a") {

spiral(matrix, N);

}

else snake(matrix, N);

string rotation;

cout << "Выберите способ модификации матрицы(a, b, c, d):" << endl;

cin >> Type;

while (Type != "a" && Type != "b" && Type != "c" && Type != "d") {

cout << "Не подхолит, попробуйте ещё раз(a, b, c, d)" << endl;

cin >> Type;

}

cout << "Модифицированная матрица: " << '\n';

modify(matrix, N, Type[0]);

cout << '\n' << "Отсортированная матрица: " << '\n';

sort(matrix, N);

cout << '\n' << "Введите величину, на которую увеличиться значение элементов матрицы: ";

int k;

cin >> k;

increase(matrix, N, k);

cout << "\n\nХотите продолжить? Если Да, введите: '0'" << endl;

cin >> snachala;

if (snachala != 0)

break;

}

}

void print(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 clear(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 spiral(int\*\* matrix, int N) {

print(matrix, N);

int lineNum = 0;

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

for (int\* cur\_m\_i = \*(matrix + lineNum) + lineNum, \*end\_m\_i = cur\_m\_i + N - 2 \* lineNum; cur\_m\_i < end\_m\_i; cur\_m\_i++) {

\*cur\_m\_i = rand() % N \* N + 1;

print(matrix, N);

}

for (int\*\* cur\_m\_i = matrix + lineNum + 1, \*\*endi = matrix + N - 2 - lineNum; cur\_m\_i <= endi; cur\_m\_i++) {

\*(\*cur\_m\_i + N - lineNum - 1) = rand() % N \* N + 1;

print(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;

print(matrix, N);

}

for (int\*\* curi = matrix + N - 1 - lineNum, \*\*endi = matrix + lineNum + 1; curi >= endi; curi--) {

\*(\*curi + lineNum) = rand() % N \* N + 1;

print(matrix, N);

}

lineNum++;

}

}

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

print(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;

print(matrix, N);

}

for (int\*\* curi = matrix + N - 1, \*\*endi = matrix; curi >= endi; curi--) {

\*(\*curi + lineNum \* 2 + 1) = rand() % N \* N + 1;

print(matrix, N);

}

}

}

void modify(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;

print(matrix, N, 0);

}

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

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

int i = 0;

int index = 0;

int end = (N \* N) - 1, begin = 0;

int mid, swap;

int f = begin;

int l = end;

mid = arr[(f + l) / 2];

while (f < l)

{

while (arr[f] < mid) f++;

while (arr[l] > mid) l--;

if (f <= l)

{

swap = arr[f];

arr[f] = arr[l];

arr[l] = swap;

f++;

l--;

}

}

if (begin < l) {

int i = 0;

int index = 0;

int end = (N \* N) - 1, begin = 0;

int mid, swap;

int f = begin;

int l = end;

mid = arr[(f + l) / 2];

while (f < l)

{

while (arr[f] < mid) f++;

while (arr[l] > mid) l--;

if (f <= l)

{

swap = arr[f];

arr[f] = arr[l];

arr[l] = swap;

f++;

l--;

}

}

}

if (f < end) {

int i = 0;

int index = 0;

int end = (N \* N) - 1, begin = 0;

int mid, swap;

int f = begin;

int l = end;

mid = arr[(f + l) / 2];

while (f < l)

{

while (arr[f] < mid) f++;

while (arr[l] > mid) l--;

if (f <= l)

{

swap = arr[f];

arr[f] = arr[l];

arr[l] = swap;

f++;

l--;

}

}

}

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++;

}

}

print(matrix, N, 0);

}

void increase(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;

print(matrix, N, 0);

}