**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. 2372 |  | Каплунов А.А. |
| Преподаватель |  | Глущенко А. Г. |

Санкт-Петербург

2022

**Цель работы.**

Изучение внутреннего строения двумерного массива, понимания, как хранятся типы данных в двумерном массиве, его представление в памяти.

**Основные теоретические положения.**

Двумерный массив — это “массив массивов”, т.е. в каждом элементе хранится ещё один массив.

Массив же представляет собой индексированную последовательность однотипных элементов с заранее определенным количеством элементов.

Наглядно двумерный массив можно представить, как матрицу элементов. Индексация массива и его “подмассивов” начинаются с нуля.

Рассмотрим пример. Объявить двумерный массив размером 25 можно так:

int a[5] [5] {};

В данном случае его можно представить как квадратную матрицу 5го порядка:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 |

Теперь рассмотрим строение двумерного массива в памяти. Все его “подмассивы” в памяти расположены последовательно. Зная, что массив в C++ это указатель на его первый элемент, мы можем получить элемент 2го столбца и 3й строки, воспользовавшись арифметикой указателей:

int element = \*(\*a + 11);

Т.к. это двумерный массив, то a – указатель на первый элемент двумерного массива, \*a – это указатель на первый элемент первого подмассива. Разыменовываем указатель *a* и прибавляем к нему 11. Таким образом, получаем нужный элемент.

Так же существуют и другие способы получить этот же элемент:

int element = \*((int\*)arr + 11);

В примере выше мы явно приводим arr к указателю (int\*), затем используем арифметику указателей.

int element = \*(\*(arr + 2) + 1);

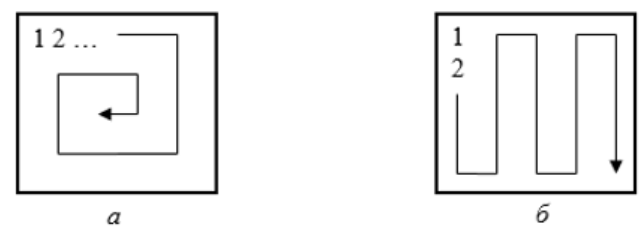
Пример выше практически не отличается от первого, но в разыменовывании указателя arr мы прибавляем 2, тем самым выбирая указатель на 3ю строку, после чего получаем нужный элемент. Эта запись эквивалентна следующей:

int element = arr[2][1];

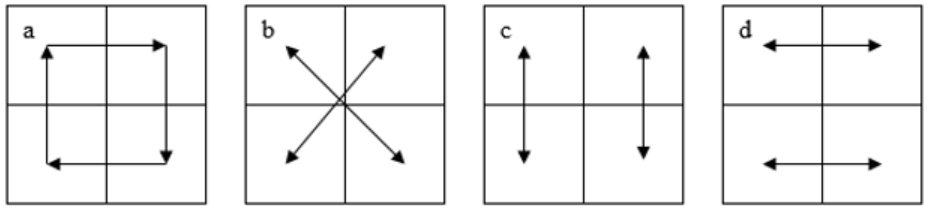
**Постановка задачи.**

Вся работа должна быть выполнена с использованием арифметики указателей.

1. Необходимо создать двумерный массив (матрицу) и заполнить его случайными числами от 1 до N\*N согласно следующим схемам (прогресс заполнения должен отображаться на экране):



1. Необходимо получить новую матрицу, применяя преобразования к матрице из шага 1:

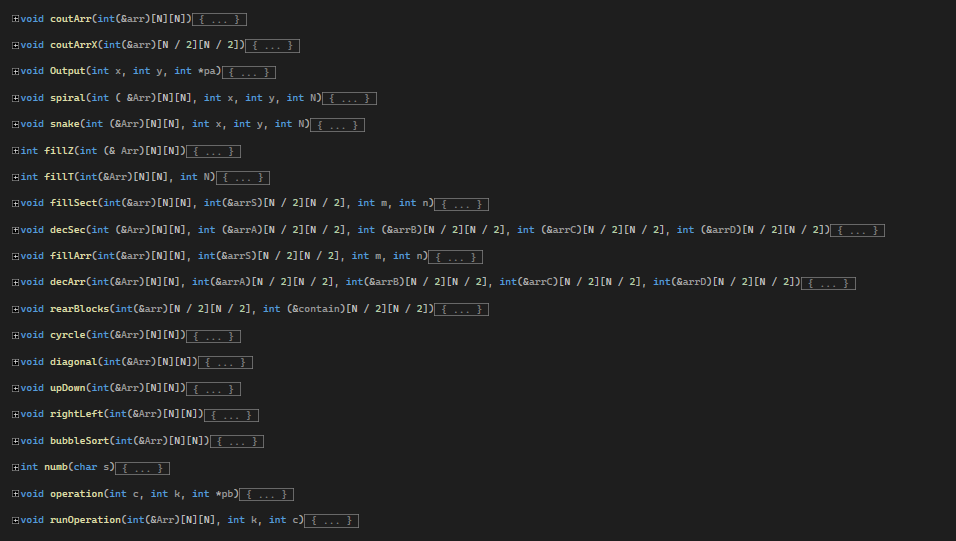


1. Отсортировать элементы матрицы, как будто это одномерный массив
2. Уменьшить, увеличить, умножить, разделить матрицу на введённое пользователем число.

**Выполнение работы.**

Возможность выбора размера матрицы (кратно 2):

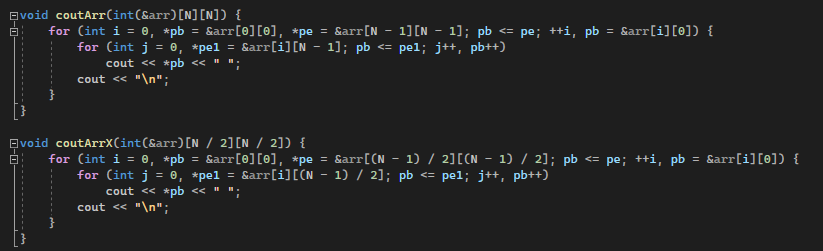


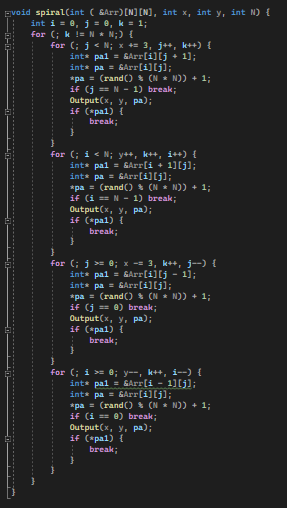


Вспомогательные функции (вывод матрицы, обнуление матрицы, промежуточный этап заполнения):

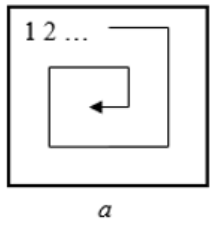
Метод “вывод матрицы” выполняет проход по всем элементам матрицы, выводит их на экран.

Метод “обнуление матрицы” устанавливает все элементы матрицы на 0 (исходное значение).

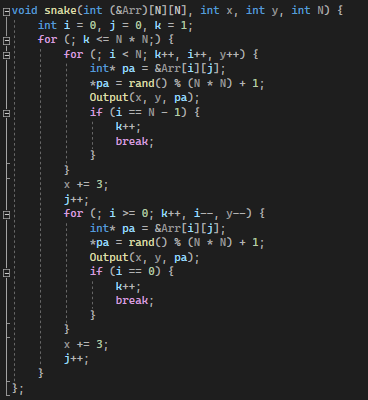
Задание 1, заполнение матрицы:



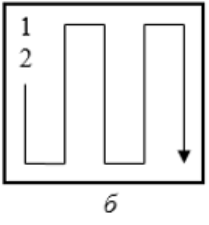
В цикле for поэтапно вводятся изменения в матрицу.



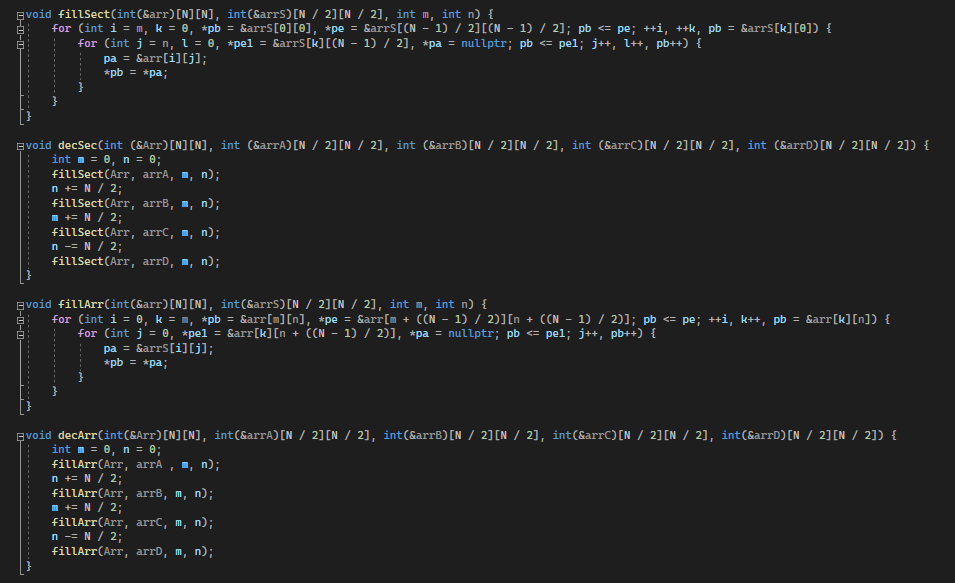
*snake – заполнение змейкой:*

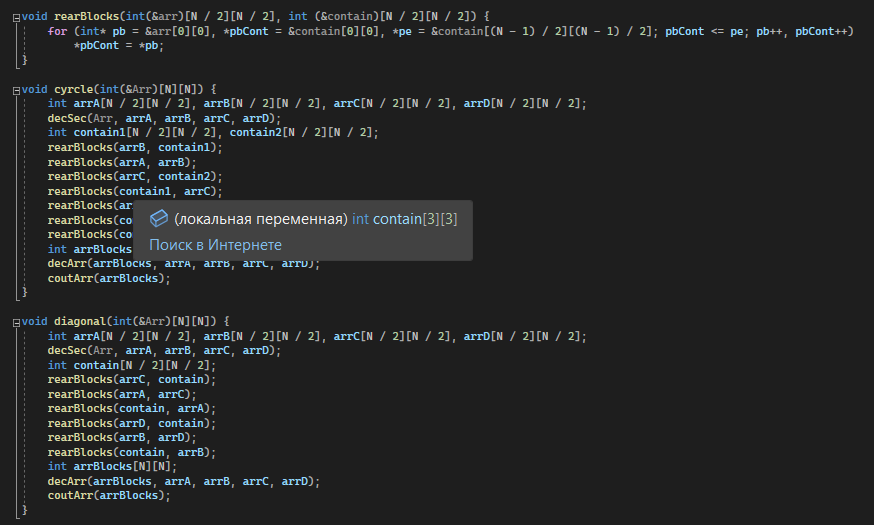


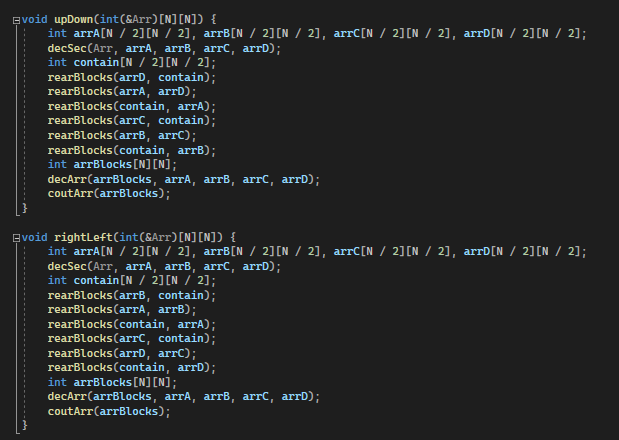
В цикле for поэтапно вводятся изменения в матрицу.



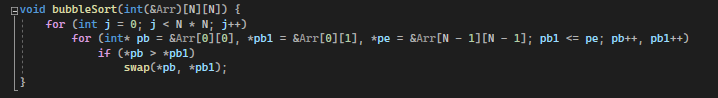
Задание 2, смена блоков в матрице:



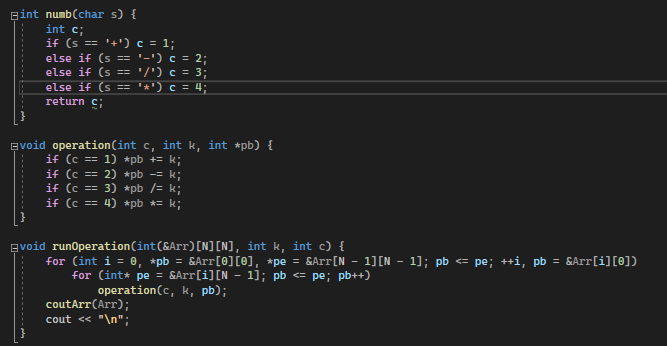


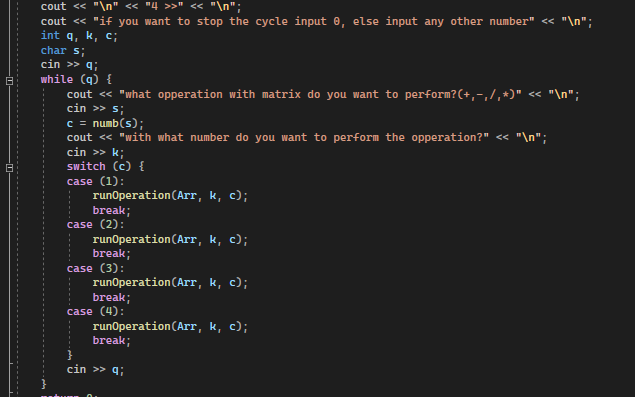


Задание 3, сортировка элементов двумерного массива. Я выбрал сортировку пузырьком.



Сложение, Вычитание, Умножение, Деление элементов матрицы и числа. Ввод числа и простой проход по всем элементам матрицы, выполнение действия.





**Выводы.**

Я научился работать с двумерными массивами, изучил указатели, изучил арифметику указателей и строение двумерного массива в памяти.

**ПРОТОКОЛ**

#include <iostream>

#include <Windows.h>

using namespace std;

const int N = 6;

void coutArr(int(&arr)[N][N]) {

for (int i = 0, \*pb = &arr[0][0], \*pe = &arr[N - 1][N - 1]; pb <= pe; ++i, pb = &arr[i][0]) {

for (int j = 0, \*pe1 = &arr[i][N - 1]; pb <= pe1; j++, pb++)

cout << \*pb << " ";

cout << "\n";

}

}

void coutArrX(int(&arr)[N / 2][N / 2]) {

for (int i = 0, \*pb = &arr[0][0], \*pe = &arr[(N - 1) / 2][(N - 1) / 2]; pb <= pe; ++i, pb = &arr[i][0]) {

for (int j = 0, \*pe1 = &arr[i][(N - 1) / 2]; pb <= pe1; j++, pb++)

cout << \*pb << " ";

cout << "\n";

}

}

void Output(int x, int y, int \*pa) {

HANDLE hStdout;

COORD destCoord;

hStdout = GetStdHandle(STD\_OUTPUT\_HANDLE);

destCoord.X = x;

destCoord.Y = y;

SetConsoleCursorPosition(hStdout, destCoord);

cout << \*pa << '\r';

cout.flush();

Sleep(100);

}

void spiral(int ( &Arr)[N][N], int x, int y, int N) {

int i = 0, j = 0, k = 1;

for (; k != N \* N;) {

for (; j < N; x += 3, j++, k++) {

int\* pa1 = &Arr[i][j + 1];

int\* pa = &Arr[i][j];

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

if (j == N - 1) break;

Output(x, y, pa);

if (\*pa1) {

break;

}

}

for (; i < N; y++, k++, i++) {

int\* pa1 = &Arr[i + 1][j];

int\* pa = &Arr[i][j];

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

if (i == N - 1) break;

Output(x, y, pa);

if (\*pa1) {

break;

}

}

for (; j >= 0; x -= 3, k++, j--) {

int\* pa1 = &Arr[i][j - 1];

int\* pa = &Arr[i][j];

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

if (j == 0) break;

Output(x, y, pa);

if (\*pa1) {

break;

}

}

for (; i >= 0; y--, k++, i--) {

int\* pa1 = &Arr[i - 1][j];

int\* pa = &Arr[i][j];

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

if (i == 0) break;

Output(x, y, pa);

if (\*pa1) {

break;

}

}

}

}

void snake(int (&Arr)[N][N], int x, int y, int N) {

int i = 0, j = 0, k = 1;

for (; k <= N \* N;) {

for (; i < N; k++, i++, y++) {

int\* pa = &Arr[i][j];

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

Output(x, y, pa);

if (i == N - 1) {

k++;

break;

}

}

x += 3;

j++;

for (; i >= 0; k++, i--, y--) {

int\* pa = &Arr[i][j];

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

Output(x, y, pa);

if (i == 0) {

k++;

break;

}

}

x += 3;

j++;

}

};

int fillZ(int (& Arr)[N][N]) {

for (int i = 0, \*pb = &Arr[0][0], \*pe = &Arr[N - 1][N - 1]; pb <= pe; ++i, pb = &Arr[i][0])

for (int \*pe = &Arr[i][N - 1]; pb <= pe; pb++)

\*pb = 0;

return 0;

}

int fillIDZ(int(&Arr)[N][N], int N) {

for (int i = 0, \*pb = &Arr[0][0], \*pe = &Arr[N - 1][N - 1]; pb <= pe; ++i, pb = &Arr[i][0])

for (int\* pe = &Arr[i][N - 1]; pb <= pe; pb++)

\*pb = (rand() % 100) + 1;

return 0;

}

void fillSect(int(&arr)[N][N], int(&arrS)[N / 2][N / 2], int m, int n) {

for (int i = m, k = 0, \*pb = &arrS[0][0], \*pe = &arrS[(N - 1) / 2][(N - 1) / 2]; pb <= pe; ++i, ++k, pb = &arrS[k][0]) {

for (int j = n, l = 0, \*pe1 = &arrS[k][(N - 1) / 2], \*pa = nullptr; pb <= pe1; j++, l++, pb++) {

pa = &arr[i][j];

\*pb = \*pa;

}

}

}

void decSec(int (&Arr)[N][N], int (&arrA)[N / 2][N / 2], int (&arrB)[N / 2][N / 2], int (&arrC)[N / 2][N / 2], int (&arrD)[N / 2][N / 2]) {

int m = 0, n = 0;

fillSect(Arr, arrA, m, n);

n += N / 2;

fillSect(Arr, arrB, m, n);

m += N / 2;

fillSect(Arr, arrC, m, n);

n -= N / 2;

fillSect(Arr, arrD, m, n);

}

void fillArr(int(&arr)[N][N], int(&arrS)[N / 2][N / 2], int m, int n) {

for (int i = 0, k = m, \*pb = &arr[m][n], \*pe = &arr[m + ((N - 1) / 2)][n + ((N - 1) / 2)]; pb <= pe; ++i, k++, pb = &arr[k][n]) {

for (int j = 0, \*pe1 = &arr[k][n + ((N - 1) / 2)], \*pa = nullptr; pb <= pe1; j++, pb++) {

pa = &arrS[i][j];

\*pb = \*pa;

}

}

}

void decArr(int(&Arr)[N][N], int(&arrA)[N / 2][N / 2], int(&arrB)[N / 2][N / 2], int(&arrC)[N / 2][N / 2], int(&arrD)[N / 2][N / 2]) {

int m = 0, n = 0;

fillArr(Arr, arrA , m, n);

n += N / 2;

fillArr(Arr, arrB, m, n);

m += N / 2;

fillArr(Arr, arrC, m, n);

n -= N / 2;

fillArr(Arr, arrD, m, n);

}

void rearBlocks(int(&arr)[N / 2][N / 2], int (&contain)[N / 2][N / 2]) {

for (int\* pb = &arr[0][0], \*pbCont = &contain[0][0], \*pe = &contain[(N - 1) / 2][(N - 1) / 2]; pbCont <= pe; pb++, pbCont++)

\*pbCont = \*pb;

}

void cyrcle(int(&Arr)[N][N]) {

int arrA[N / 2][N / 2], arrB[N / 2][N / 2], arrC[N / 2][N / 2], arrD[N / 2][N / 2];

decSec(Arr, arrA, arrB, arrC, arrD);

int contain1[N / 2][N / 2], contain2[N / 2][N / 2];

rearBlocks(arrB, contain1);

rearBlocks(arrA, arrB);

rearBlocks(arrC, contain2);

rearBlocks(contain1, arrC);

rearBlocks(arrD, contain1);

rearBlocks(contain2, arrD);

rearBlocks(contain1, arrA);

int arrBlocks[N][N];

decArr(arrBlocks, arrA, arrB, arrC, arrD);

coutArr(arrBlocks);

}

void diagonal(int(&Arr)[N][N]) {

int arrA[N / 2][N / 2], arrB[N / 2][N / 2], arrC[N / 2][N / 2], arrD[N / 2][N / 2];

decSec(Arr, arrA, arrB, arrC, arrD);

int contain[N / 2][N / 2];

rearBlocks(arrC, contain);

rearBlocks(arrA, arrC);

rearBlocks(contain, arrA);

rearBlocks(arrD, contain);

rearBlocks(arrB, arrD);

rearBlocks(contain, arrB);

int arrBlocks[N][N];

decArr(arrBlocks, arrA, arrB, arrC, arrD);

coutArr(arrBlocks);

}

void upDown(int(&Arr)[N][N]) {

int arrA[N / 2][N / 2], arrB[N / 2][N / 2], arrC[N / 2][N / 2], arrD[N / 2][N / 2];

decSec(Arr, arrA, arrB, arrC, arrD);

int contain[N / 2][N / 2];

rearBlocks(arrD, contain);

rearBlocks(arrA, arrD);

rearBlocks(contain, arrA);

rearBlocks(arrC, contain);

rearBlocks(arrB, arrC);

rearBlocks(contain, arrB);

int arrBlocks[N][N];

decArr(arrBlocks, arrA, arrB, arrC, arrD);

coutArr(arrBlocks);

}

void rightLeft(int(&Arr)[N][N]) {

int arrA[N / 2][N / 2], arrB[N / 2][N / 2], arrC[N / 2][N / 2], arrD[N / 2][N / 2];

decSec(Arr, arrA, arrB, arrC, arrD);

int contain[N / 2][N / 2];

rearBlocks(arrB, contain);

rearBlocks(arrA, arrB);

rearBlocks(contain, arrA);

rearBlocks(arrC, contain);

rearBlocks(arrD, arrC);

rearBlocks(contain, arrD);

int arrBlocks[N][N];

decArr(arrBlocks, arrA, arrB, arrC, arrD);

coutArr(arrBlocks);

}

void bubbleSort(int(&Arr)[N][N]) {

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

for (int\* pb = &Arr[0][0], \*pb1 = &Arr[0][1], \*pe = &Arr[N - 1][N - 1]; pb1 <= pe; pb++, pb1++)

if (\*pb > \*pb1)

swap(\*pb, \*pb1);

}

int numb(char s) {

int c;

if (s == '+') c = 1;

else if (s == '-') c = 2;

else if (s == '/') c = 3;

else if (s == '\*') c = 4;

return c;

}

void operation(int c, int k, int \*pb) {

if (c == 1) \*pb += k;

if (c == 2) \*pb -= k;

if (c == 3) \*pb /= k;

if (c == 4) \*pb \*= k;

}

void runOperation(int(&Arr)[N][N], int k, int c) {

for (int i = 0, \*pb = &Arr[0][0], \*pe = &Arr[N - 1][N - 1]; pb <= pe; ++i, pb = &Arr[i][0])

for (int\* pe = &Arr[i][N - 1]; pb <= pe; pb++)

operation(c, k, pb);

coutArr(Arr);

cout << "\n";

}

int main()

{

cout << "1 >>";

int x = 0, y = 1;

int Arr[N][N] = {};

fillZ(Arr);

spiral(Arr, x, y, N);

HANDLE hStdout;

COORD destCoord;

hStdout = GetStdHandle(STD\_OUTPUT\_HANDLE);

y = N + 2;

destCoord.X = 0;

destCoord.Y = y;

SetConsoleCursorPosition(hStdout, destCoord);

fillZ(Arr);

snake(Arr, x, y, N);

y = N + N + 2;

destCoord.X = 0;

destCoord.Y = y;

SetConsoleCursorPosition(hStdout, destCoord);

cout << "\n" << "2 >>" << "\n";

cyrcle(Arr);

Sleep(1000);

cout << "\n";

diagonal(Arr);

Sleep(1000);

cout << "\n";

upDown(Arr);

Sleep(1000);

cout << "\n";

rightLeft(Arr);

Sleep(1000);

cout << "\n" << "3 >>" << "\n";

bubbleSort(Arr);

coutArr(Arr);

cout << "\n" << "4 >>" << "\n";

cout << "if you want to stop the cycle input 0, else input any other number" << "\n";

int q, k, c;

char s;

cin >> q;

while (q) {

cout << "what opperation with matrix do you want to perform?(+,-,/,\*)" << "\n";

cin >> s;

c = numb(s);

cout << "with what number do you want to perform the opperation?" << "\n";

cin >> k;

switch (c) {

case (1):

runOperation(Arr, k, c);

break;

case (2):

runOperation(Arr, k, c);

break;

case (3):

runOperation(Arr, k, c);

break;

case (4):

runOperation(Arr, k, c);

break;

}

cin >> q;

}

return 0;

}