**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"**

|  |  |  |
| --- | --- | --- |
| Студентка гр. 4373 |  | Андреева П.Д. |
| Преподаватель |  | Глущенко А. Г. |

Санкт-Петербург

2024

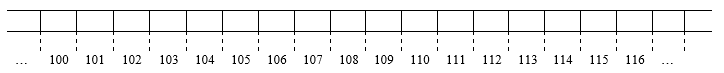
**Цель работы.**

Изучение работы с двумерными массивами и сортировками, перестановками и указателями.

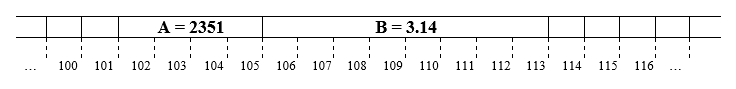
**Основные теоретические положения.**

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. К использованию указателей и ссылок мы будем неоднократно возвращаться в последующих разделах.

Все данные (переменные, константы и др.) хранятся в памяти. Память представляет собой непрерывную последовательность ячеек (байтов), каждая из которых имеет свой номер – адрес:



При определении, например, некоторой переменной, она располагается в памяти по определенному адресу и занимает столько ячеек, сколько требует тип этой переменной. Пусть, например, имеется переменные **int A = 2351** и **double B = 3.1** и пусть они располагаются в памяти так:



Говорят, что переменная **А** располагается по адресу 102 и занимает 4 байта, а переменная **B** имеет адрес 106 и занимает 8 байт памяти.

Для получения адреса какого-либо программного объекта используется оператор **&**. Например, если выполнить фрагмент следующей программы (в предположении, что переменные A и B располагаются в памяти, как это показано на предыдущем рисунке):

int A = 2351;  
  
double B = 3.14;  
  
cout << “Значение переменной А: ” << A << endl;  
  
cout << “Адрес переменной А: ” << &A << endl;  
  
cout << “Значение переменной В: ” << В << endl;  
  
cout << “Адрес переменной В: ” << &В << endl;

получим следующий результат:

Значение переменной А: 2351  
  
Адрес переменной А: 102  
  
Значение переменной В: 3.14  
  
Адрес переменной В: 106

Правда, значения адресов переменных будут выведены в шестнадцатеричном формате.

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

Указатели определяются в программе следующим образом:

**<тип данных> \*<имя переменной>**

Здесь <**тип данных**> определяет так называемый **базовый тип указателя**.

**<Имя переменной>** является идентификатором переменной-указателя.

Признаком того, что это переменная указатель, является символ \*, располагающийся между базовым типом указателя и именем переменной-указателя.

Например:

int \*p1;  
  
double \*p2;

Здесь определены две переменные-указатели (или просто – два указателя). Указатель **p1** является переменной-указателем на базовый тип **int** (или, как говорят, переменная **p1** указывает на **int** - значение), а указатель **p2** указывает на **double** – значение.

Иными словами, переменная **p1** предназначена для хранения адресов участков памяти, размер которых соответствует типу **int** (4 байта), а переменная **p2** - для хранения адресов участков памяти, размер которых соответствует типу **double** (8 байт).

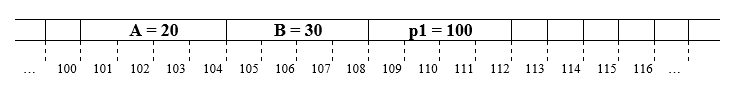
Формально указатели представляют собой обычные целые значения типа **int** и занимают в памяти 4 байта не зависимо от базового типа указателя. Значения указателей при их выводе на экран представляются как целые значения в шестнадцатеричном формате.

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.

Рассмотрим следующий пример:

int A = 20, B = 30;  
  
int \*p1 = &A;

Пусть переменные **A** и **B** расположены в памяти, например, так, как это показано на следующем рисунке:



Указатель **p1** содержит адрес переменной **A**, который равен 100 и \***p1** будет равно значению переменной **A**, то есть 20. Выполним следующую операцию:

p1 = p1 + 1;

или, что то же самое:

p1++;

Значение указателя изменится и станет равным 104, а не 101, как, наверное, ожидалось. То есть теперь указатель ссылается уже на переменную **B** и значение \***p1** будет равно 30.

Таким образом, добавление или вычитание 1 из указателя приводит к изменению его значения на размер базового типа указателя. В общем случае, например, при выполнении следующей операции:

p1 = p1 + N; // N – некоторое целое значение

значение указателя увеличится на **sizeof(<базовый тип указателя>) \* N** и в нашем случае это приращение будет равно **sizeof(int) \* N = 4 \* N**. Так, если N = 4, а p1= 100, то значение указателя **p1** увеличится на 16 и станет равно 116, и указатель будет ссылаться на данные, расположенные по адресу 116.

**Внимание.**  Добавлять к указателям или вычитать из указателей можно только целые значения.

**Постановка задачи.**

Необходимо написать программу, которая:

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 (\*Для манипуляции с элементами используйте только арифметику указателей):



2) Получает новую матрицу, из матрицы п. 1, переставляя ее блоки в соответствии со схемами (\*Для манипуляции с элементами используйте только арифметику указателей):



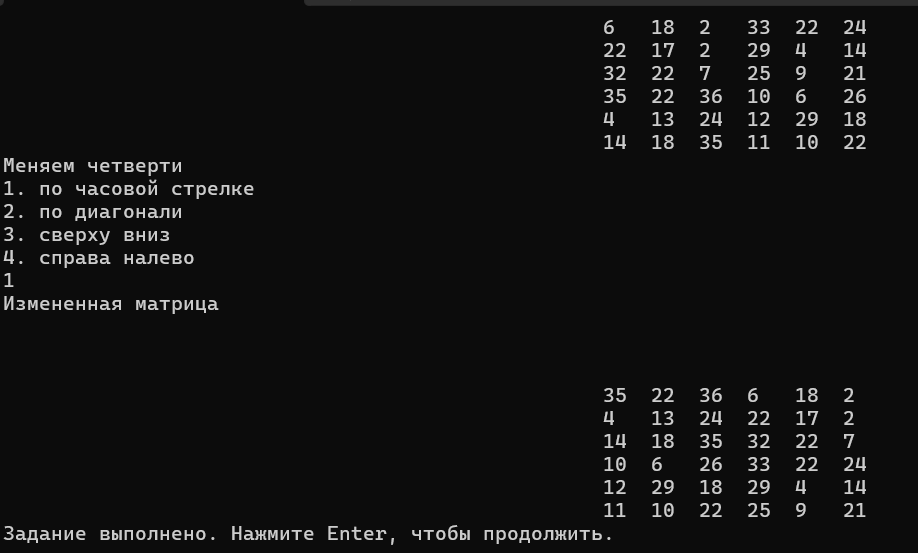
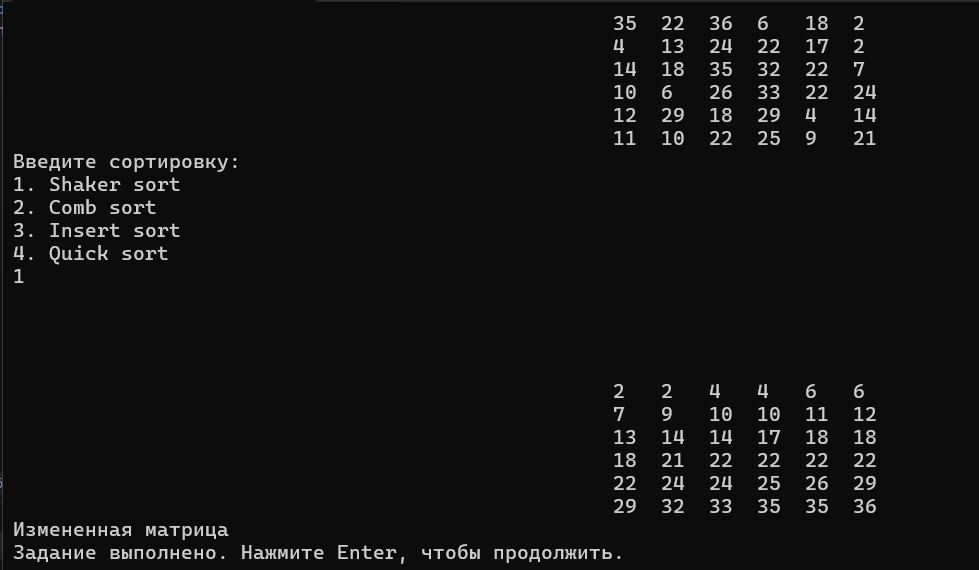
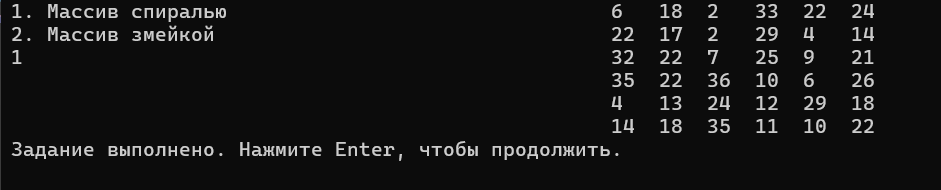
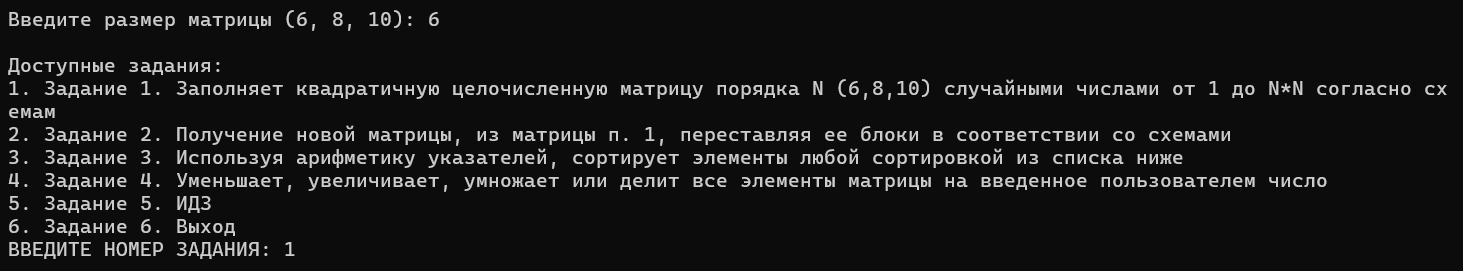
3) Используя арифметику указателей, сортирует элементы любой сортировкой из списка ниже (если во 2 ПР вы реализовывали одну из представленных сортировок, рекомендуется переиспользовать написанный код и модернизировать его для работы с указатями).   
Варианты сортировок:

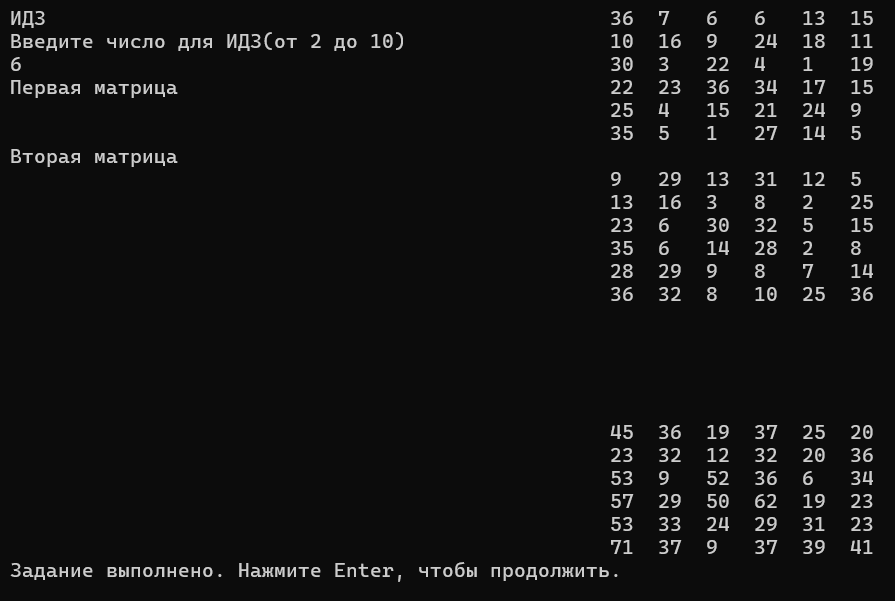
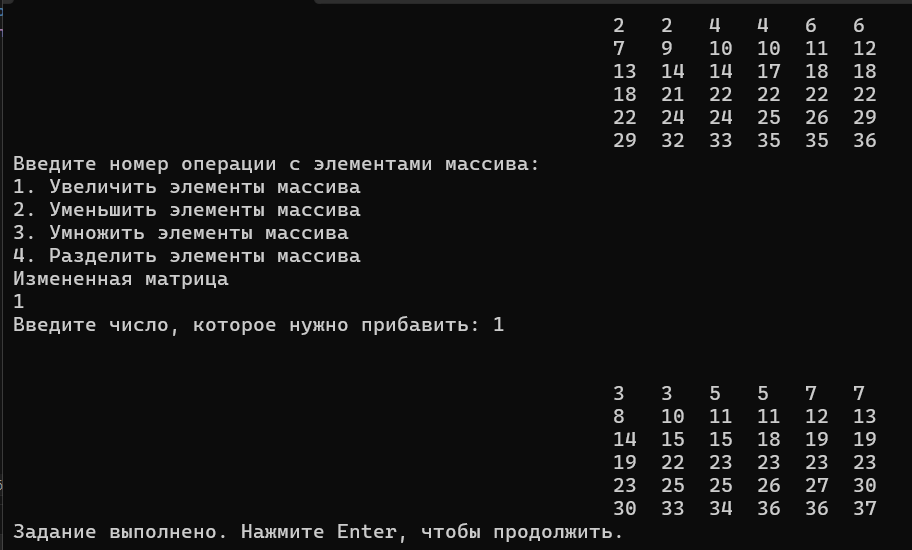
* Shaker sort;
* Comb sort;
* Insert sort;
* Quick sort;

4) Уменьшает, увеличивает, умножает или делит все элементы матрицы на введенное пользователем число (\*Для манипуляции с элементами используйте только арифметику указателей).

**Выполнение работы.**

Код программы представлен в приложении А.





**Выводы.**

Я узнала, как создавать массивы, сортировать их, находить определенные элементы массивов, выводить и менять значения в массиве.

Приложение А

рабочий код

#include <iostream>

#include <thread>

#include <chrono>

#include <Windows.h>

using namespace std;

void print\_matrix(int\* matrix, int N) {

int\* end = matrix + (N \* N) - 1;

for (int\* i = matrix; i <= end; ++i) {

if ((i - matrix) % N == 0) {

cout << "\n";

}

cout << \*i << " ";

}

}

void printMatrix(int\* matrix, int n, int helper) {

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

COORD destCoord;

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

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

destCoord.X = j \* 4 + 50;

destCoord.Y = i + helper;

SetConsoleCursorPosition(hStdout, destCoord);

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

}

cout << endl;

}

}

void idz() {

int N;

cout << "Введите число для ИДЗ(от 2 до 10)" << endl;

cin >> N;

int\* matrix1 = new int[N \* N];

int\* matrix2 = new int[N \* N];

int\* matrix3 = new int[N \* N];

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

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

\*(matrix1 + i \* N + j) = rand() % (N \* N) + 1;

\*(matrix2 + i \* N + j) = rand() % (N \* N) + 1;

\*(matrix3 + i \* N + j) = 1;

}

}

cout << "Первая матрица" << endl;

printMatrix(matrix1, N, 0);

cout << "Вторая матрица" << endl;

printMatrix(matrix2, N, N + 1);

cout << endl;

int top = 0, bottom = N - 1, left = 0, right = N - 1;

while (top <= bottom && left <= right) {

for (int i = left; i <= right; i++) {

int \* num = matrix3 + i + top\*N;

\*num = \*(matrix1 + i + top\*N) + \*(matrix2 + i+ top\*N);

printMatrix(matrix3, N, N + 12);

Sleep(50);

}

top++;

}

printMatrix(matrix3, N, N + 12);

}

void shaker\_sort(int \*matrix, int N) {

bool swapped = true;

while (swapped) {

swapped = false;

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

if (\*(i - 1) > \*i) {

int temp = \*(i - 1);

\*(i - 1) = \*i;

\*i = temp;

swapped = true;

}

}

if (!swapped) {

break;

}

swapped = false;

for (int\* i = matrix + N\*N - 2; i >= matrix; i--) {

if (\*(i - 1) > \*i) {

int temp = \*(i - 1);

\*(i - 1) = \*i;

\*i = temp;

swapped = true;

}

}

}

}

void comb\_sort(int\* matrix, int N) {

int temp;

float k = 1.247, S = N - 1;

int count = 0;

while (S >= 1)

{

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

{

if (\*(matrix+i) > \*(matrix+int(i + S)))

{

temp = \*(matrix+int(i + S));

\*(matrix+int(i + S)) = \*(matrix+i);

\*(matrix+i) = temp;

}

}

S /= k;

}

while (true)

{

for (int i = 0; i < N - 1; i++)

{

if (\*(matrix+i) > \*(matrix+i + 1))

{

temp = \*(matrix+i + 1);

\*(matrix+i + 1) = \*(matrix+i);

\*(matrix+i) = temp;

}

else count++;

}

if (count == N - 1)

break;

else

count = 0;

}

}

void quick\_sort(int\* matrix, int\* begin, int\* end) {

//if (begin >= end) return;

int\* f = begin;

int\* l = end;

int mid = \*(begin + (end - begin) / 2);

while (f <= l) {

while (\*f < mid) f++;

while (\*l > mid) l--;

if (f <= l) {

swap(\*f, \*l);

f++;

l--;

}

}

if (begin < l) quick\_sort(matrix, begin, l);

if (f < end) quick\_sort(matrix, f, end);

}

void insert\_sort(int\* matrix, int N) {

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

int key = \*(matrix + i);

int j = i - 1;

while (j >= 0 && \*(matrix + j) > key) {

\*(matrix + j + 1) = \*(matrix + j);

j--;

}

\*(matrix + j + 1) = key;

}

}

void fillSnake(int\* matrix, int n) {

int top = 0, bottom = n - 1, left = 0, right = n - 1;

while (top <= bottom && left <= right) {

for (int i = top; i <= bottom; i++) {

int\* num = matrix + i \* n + left;

\*num = rand() % (n \* n) + 1;

printMatrix(matrix, n, 0);

Sleep(400);

}

left++;

for (int i = bottom; i >= top; i--) {

int\* num = matrix + i \* n + left;

\*num = rand() % (n \* n) + 1;

printMatrix(matrix, n, 0);

Sleep(400);

}

left++;

}

}

void fillSpiral(int \*matrix, int n) {

int top = 0, bottom = n - 1, left = 0, right = n - 1;

while (top <= bottom && left <= right) {

for (int i = left; i <= right; i++) {

\*(matrix + top \* n + i)= rand() % (n \* n) + 1;

printMatrix(matrix, n, 0);

Sleep(100);

}

top++;

for (int i = top; i <= bottom; i++) {

\*(matrix + i\* n + right) = rand() % (n \* n) + 1;

printMatrix(matrix, n, 0);

Sleep(100);

}

right--;

for (int i = right; i >= left; i--) {

\*(matrix + bottom \* n + i) = rand() % (n \* n) + 1;

printMatrix(matrix, n, 0);

Sleep(100);

}

bottom--;

for (int i = bottom; i >= top; i--) {

\*(matrix + i\*n + left) = rand() % (n \* n) + 1;

printMatrix(matrix, n, 0);

Sleep(100);

}

left++;

}

printMatrix(matrix, n, 0);

}

void swap\_quarters\_a(int \*matrix, int N) {

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

int middle = N / 2;

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

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

\*(temp + i\*N + j) = \*(matrix + i\*N + j);

}

}

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

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

if (i < middle && j < middle) {

\*(matrix + i\*N + j + middle) = \*(temp + i\*N + j);

}

else if (i < middle && j >= middle) {

\*(matrix + (i+ middle)\*N + j) = \*(temp + i\*N + j);

}

else if (i >= middle && j >= middle) {

\*(matrix + i\*N + (j - middle)) = \*(temp + i\*N + j);

}

else {

\*(matrix + (i - middle)\*N + j) = \*(temp + i\*N + j);

}

}

}

printMatrix(matrix, N, N + 10);

}

void swap\_quarters\_b(int \*matrix, int N) {

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

int middle = N / 2;

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

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

\*(temp+i\*N+j) = \*(matrix + i \* N + j);

}

}

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

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

if (i < middle && j < middle) {

\*(matrix + (i + middle)\*N + j + middle) = \*(temp + i\*N + j);

}

else if (i < middle && j >= middle) {

\*(matrix + (i + middle)\*N + j - middle) = \*(temp + i\*N + j);

}

else if (i >= middle && j >= middle) {

\*(matrix + (i - middle)\*N + j - middle) = \*(temp + i\*N + j);

}

else {

\*(matrix + (i - middle) \* N + j + middle) = \*(temp + i \* N + j);

}

}

}

printMatrix(matrix, N, N + 10);

}

void swap\_quarters\_c(int \*matrix, int N) {

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

int middle = N / 2;

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

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

\*(temp + i \* N + j) = \*(matrix + i \* N + j);

}

}

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

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

if (i < middle) {

\*(matrix + (i + middle)\*N + j) = \*(temp + i\*N + j);

}

else {

\*(matrix + (i - middle)\*N + j) = \*(temp + i\*N + j);

}

}

}

printMatrix(matrix, N, N + 10);

}

void swap\_quarters\_d(int \*matrix, int N) {

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

int middle = N / 2;

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

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

\*(temp + i \* N + j) = \*(matrix + i \* N + j);

}

}

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

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

if (j < middle) {

\*(matrix + i\*N + j + middle) = \*(temp + i\*N + j);

}

else {

\*(matrix + i\*N + j - middle) = \*(temp + i\*N + j);

}

}

}

printMatrix(matrix, N, N + 10);

}

void enlarge\_each\_element(int \*matrix, int N) {

int enlarge\_num;

cout << "Введите число, которое нужно прибавить: ";

cin >> enlarge\_num;

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

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

\*(matrix + i\*N + j) += enlarge\_num;

}

}

printMatrix(matrix, N, N + 10);

}

void reduce\_each\_element(int \*matrix, int N) {

int reduce\_num;

cout << "Введите число, которое нужно отнять: ";

cin >> reduce\_num;

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

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

\*(matrix + i\*N + j) -= reduce\_num;

}

}

printMatrix(matrix, N, N + 10);

}

void divide\_each\_element(int \*matrix, int N) {

int divide\_num;

cout << "Введите число, на которое нужно разделить: ";

cin >> divide\_num;

if (divide\_num == 0) {

cout << "Деление на ноль!" << endl;

return;

}

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

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

\*(matrix + i\*N + j) /= divide\_num;

}

}

printMatrix(matrix, N, N + 10);

}

void multiply\_each\_element(int \*matrix, int N) {

int multiply\_num;

cout << "Введите число, на которое нужно умножить: ";

cin >> multiply\_num;

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

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

\*(matrix + i\*N + j) \*= multiply\_num;

}

}

printMatrix(matrix, N, N + 10);

}

void display\_menu() {

cout << "\nДоступные задания:" << endl;

cout << "1. Задание 1. Заполняет квадратичную целочисленную матрицу порядка N (6,8,10) случайными числами от 1 до N\*N согласно схемам" << endl;

cout << "2. Задание 2. Получение новой матрицы, из матрицы п. 1, переставляя ее блоки в соответствии со схемами" << endl;

cout << "3. Задание 3. Используя арифметику указателей, сортирует элементы любой сортировкой из списка ниже" << endl;

cout << "4. Задание 4. Уменьшает, увеличивает, умножает или делит все элементы матрицы на введенное пользователем число" << endl;

cout << "5. Задание 5. ИДЗ" << endl;

cout << "6. Задание 6. Выход" << endl;

}

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

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

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

\*(matrix + i \* N + j) = 1;

}

}

}

int main() {

setlocale(0, "");

srand(time(NULL));

int N, identificator, num1, num2, num3, num4;

cout << "Введите размер матрицы (6, 8, 10): ";

cin >> N;

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

while (true) {

display\_menu();

cout << "ВВЕДИТЕ НОМЕР ЗАДАНИЯ: ";

cin >> identificator;

system("cls"); // Для Windows

switch (identificator) {

case 1:

start(matrix, N);

cout << "1. Массив спиралью" << endl;

cout << "2. Массив змейкой" << endl;

cin >> num1;

switch (num1) {

case 1:

fillSpiral(matrix, N);

break;

case 2:

fillSnake(matrix, N);

break;

}

break;

case 2:

printMatrix(matrix, N, 0);

cout << "Меняем четверти" << endl;

cout << "1. по часовой стрелке" << endl;

cout << "2. по диагонали " << endl;

cout << "3. сверху вниз" << endl;

cout << "4. справа налево" << endl;

cin >> num2;

cout << "Измененная матрица" << endl;

switch (num2) {

case 1:

swap\_quarters\_a(matrix, N);

break;

case 2:

swap\_quarters\_b(matrix, N);

break;

case 3:

swap\_quarters\_c(matrix, N);

break;

case 4:

swap\_quarters\_d(matrix, N);

break;

}

break;

case 3:

printMatrix(matrix, N, 0);

cout << "Введите сортировку:" << endl;

cout << "1. Shaker sort" << endl;

cout << "2. Comb sort" << endl;

cout << "3. Insert sort" << endl;

cout << "4. Quick sort" << endl;

cin >> num3;

switch (num3) {

case 1:

shaker\_sort(matrix, N);

printMatrix(matrix, N, N+10);

break;

case 2:

comb\_sort(matrix, N);

printMatrix(matrix, N, N + 10);

break;

case 3:

insert\_sort(matrix, N);

printMatrix(matrix, N, N + 10);

break;

case 4:

quick\_sort(matrix, matrix, matrix + N \* N - 1);

printMatrix(matrix, N, N + 10);

break;

}

cout << "Измененная матрица" << endl;

break;

case 4:

printMatrix(matrix, N, 0);

cout << "Введите номер операции с элементами массива:" << endl;

cout << "1. Увеличить элементы массива" << endl;

cout << "2. Уменьшить элементы массива " << endl;

cout << "3. Умножить элементы массива" << endl;

cout << "4. Разделить элементы массива" << endl;

cout << "Измененная матрица" << endl;

cin >> num4;

switch (num4) {

case 1:

enlarge\_each\_element(matrix, N);

break;

case 2:

reduce\_each\_element(matrix, N);

break;

case 3:

multiply\_each\_element(matrix, N);

break;

case 4:

divide\_each\_element(matrix, N);

break;

}

break;

case 5:

cout << "ИДЗ" << endl;

idz();

break;

case 6:

cout << "Выход!" << endl;

return 0;

default:

cout << "Некорректный номер задания." << endl;

break;

}

cout << "Задание выполнено. Нажмите Enter, чтобы продолжить." << endl;

cin.ignore();

cin.get();

}

return 0;

}