**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

|  |  |  |
| --- | --- | --- |
| Студент(ка) гр. |  | Саевич И. И. 4372 |
| Преподаватель |  | Глущенко А. Г. |

Санкт-Петербург

2024

**Цель работы.**

Изучение двумерных статических массивов; изучение указателей; получение практических навыков работы с ними (заполнение двумерного массива “змейкой” и “по спирали” при помощи арифметики указателей; перестановка блоков матрицы при помощи арифметики указателей; сортировка двумерного массива при помощи 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. 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.

Поскольку упомянутые арифметические операции выполняются по-разному при их применении к указателям и обычным арифметическим типам данных, а также учитывая высший приоритет операции \*, при использовании указателей в составе выражений следует внимательно обращаться со скобками. Например, выражения (см. предыдущий рисунок)

\*(p1 + 1) и \*p1 + 1

имеют совершенно разный смысл. Первое выражение даст значение 30, а второе выражение будет равно 21 (в первом выражении сначала изменяется адрес, а затем осуществляется обращение в память по этому измененному адресу; во втором выражении мы обращаемся по старому адресу и к значению, хранящемуся по этому адресу добавляем 1).

В изучаемых нами языках программирования между массивами и указателями имеется очень тесная связь.

Кода мы определяем в программе некоторый массив, например,

int Arr[10];

переменная Arr без индексов представляет собой указатель на первый элемент массива в данном случае из 10 целых чисел (содержит адрес первого элемента массива). Если вывести на экран значение переменной Arr

cout << Arr:

мы увидим некоторое целое значение в шестнадцатеричном формате, соответствующее адресу первого элемента этого массива.

Замечание. Именно по этой причине в языке C++ отсутствует операция присвоения сразу всех значений одного массива другому (в некоторых других языках, например, в Pascal такая возможность имеется). Действительно, если имеются два массива

int A1[10], A2[10];

то попытка выполнить присвоение A1 = A2 привела бы к тому, что переменная A1 стала бы указывать на ту же область памяти, что и переменная A2 (мы скопировали адрес из A2 в A1, а не содержимое одного массива в другой). Адрес, который хранился ранее в переменной A1, был бы утерян, что привело бы к утечке памяти (для десяти элементов массива A1 в памяти было выделено место, но теперь мы “забыли”, где оно находится, то есть потеряли память). По этой причине подобные операции с массивами в языке C++ запрещены. Более того, запрещены любые изменения значения переменной массива.

Указателю, имеющему такой же базовый тип, как и элементы массива, можно присвоить массив следующим образом:

int Arr[10];

int \*p;

p = Arr;

Но обратное присвоение выполнить невозможно:

Arr = p; // Ошибка

Такое присвоение невозможно, поскольку переменная массива – это константа, изменение которой запрещено.

Так как переменная массива является указателем на первый элемент массива, появляются дополнительные возможности по работе с массивами на основе использования арифметики указателей. Например, чтобы получить 5–й элемент массива Arr можно воспользоваться одним из следующих выражений:

Arr[4] или \*(Arr + 4) или \*( p + 4)

Первое выражение – это пример обычной индексации элементов массива. Во втором и третьем выражениях мы использовали арифметику указателей и с помощью операции + получили адрес пятого элемента массива. Затем с помощью операции \* взяли значение по этому адресу и получили значение 5-го элемента массива. Обратите внимание на скобки в этих выражениях, если их не поставить и написать \*Arr + 4 или \*p + 4, то эти выражения будут равны значению первого элемента массива увеличенного на 4, так как операция \* имеет больший приоритет, чем операция +.

Вот пример фрагмента программы для работы с массивом с помощью обычной индексации элементов массива. Этот фрагмент обеспечивает ввод элементов целочисленного массива с клавиатуры, вычисление квадратов значений элементов массива, а затем вывод элементов массива на экран:

int A[10];

for (int i = 0; i < 10; ++ i)

{

cin >> A[i];

A[i] = A[i] \* A[i];

}

for (int i = 0; i < 10; ++ i)

cout << A[i] << “ “;

cout << endl;

…..

А вот тот же фрагмент, но с использованием арифметики указателей:

int A[10];

for (int \*Next = A, \*End = Next + 9; Next <= End; ++ Next)

{

cin >> \*Next;

\*Next = \*Next \* \*Next; // \*Next = (\*Next) \* (\*Next);

}

for (int \*Next = A, \*End = Next + 9; Next <= End; ++ Next)

cout << \*Next << “ “;

cout << endl;

…

Использование арифметики указателей при работе с массивами приводит обычно к уменьшению объема генерируемого кода программы и к уменьшению времени ее выполнения, то есть к увеличению быстродействия.

Поскольку указатель и имя массива, в большой степени, взаимозаменяемы, указатели можно индексировать, как обычные массивы:

int A[10], \*P = A;

for (int i = 0; i < 10; ++ i)

cout << P[i] << “ “;

Можно создавать и массивы указателей. Например:

int a = 1, b = 2, c = 3, \*M[3];

M[0] = & a; // Элементам массива М присваиваются адреса переменных a, b и c

M[1] = & b;

M[2] = & c;

for (int i = 0; i < 3; ++ i)

cout << \*M[i] << “ ”;

cout << endl;

Массив M – это трехэлементный массив указателей на целые значения, то есть каждый элемент этого массива представляет собой указатель на целое.

С помощью массивов указателей можно моделировать различные интересные конструкции данных. Например, пусть имеется квадратная матрица размерности 5 х 5 симметричная относительно главной диагонали. Для ее однозначного представления достаточно хранить в памяти не все 25 элементов этой матрицы, а только 15 (например, элементы под главной диагональю вместе с элементами главной диагонали). Для этого можно предложить следующую конструкцию:

int A1[1], A2[2], A3[3], A4[4], A5[5], \*A[5] = { A1, A2, A3, A4, A5 };

// Вводим 15 целых значений - элементы под главной диагональю и диагональные

// элементы матрицы

for (int i = 0; i < 5; ++i)

for (int j = 0; j <= i; ++ j)

cin >> A[i][j];

cout << endl;

// Выводим симметричную матрицу 5 на 5 на экран

for (int i = 0; i < 5; ++i)

{

for (int j = 0; j <= i; ++ j)

cout << A[i][j] << " ";

for (int j = i + 1; j < 5; ++ j)

cout << A[j][i] << " ";

cout << endl;

}

cout << endl;

А это пятиэлементный массив указателей на символы, инициализированный некоторыми текстовыми строками:

char \* Words[5] = { "Слово1", "Слово2", "Слово3", "Слово4", "Слово5" }

Как это работает: когда компилятор встречает в программе некоторый текст, заключенный в кавычки, в памяти создается символьный массив соответствующей этому тексту длины и адрес этого символьного массива присваивается соответствующему элементу – указателю массива Words.

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

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

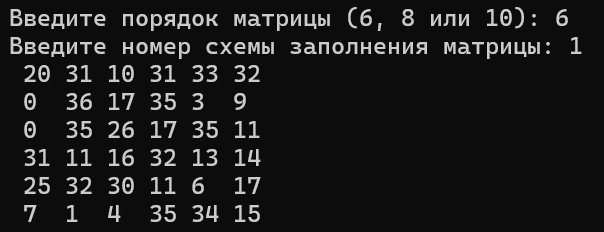
3) Используя арифметику указателей, сортирует элементы любой сортировкой из списка ниже;

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

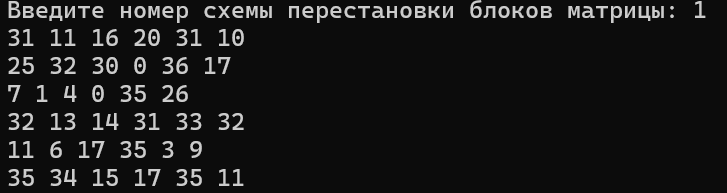
**Выполнение работы.**

Код программы представлен в приложении A.

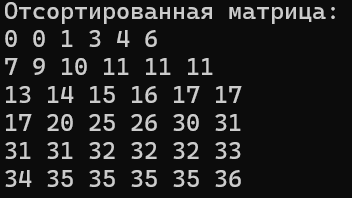
Пользователь вводит размерность матрицы, номер схемы заполнения



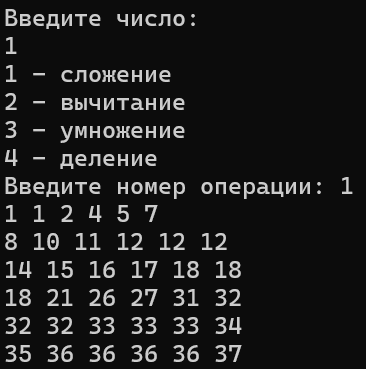
Пользователь вводит номер схемы перестановки блоков матрицы



Матрицы сортируется при помощи алгоритма сортировки вставками. Благодаря использованию арифметики указателей функция работает для массивов любой вложенности.



Пользователь может прибавить, отнять, умножить или разделить все элементы матрицы на введённое им число.



**Выводы.**

В ходе работы мы изучили двумерные статические массивы и указатели; получили практические навыки работы с ними (заполнение двумерного массива “змейкой” и “по спирали” при помощи арифметики указателей; перестановка блоков матрицы при помощи арифметики указателей; сортировка двумерного массива при помощи арифметики указателей);

Приложение А

рабочий код

﻿ #include <iostream>

#include <Windows.h>

using namespace std;

void InputCheck(int& number)

{

while (cin.fail())

{

cin.clear();

cin.sync();

cin.ignore(1000, '\n');

cout << "Введите ЧИСЛО" << endl;

cin >> number;

}

}

COORD GetConsoleCursorPosition(HANDLE hConsoleOutput)

{

CONSOLE\_SCREEN\_BUFFER\_INFO cbsi;

if (GetConsoleScreenBufferInfo(hConsoleOutput, &cbsi))

{

return cbsi.dwCursorPosition;

}

else

{

COORD invalid = { 0, 0 };

return invalid;

}

}

void FillMatrixA(int\* matrix, int N, const int& delay, HANDLE hStdout, COORD destCoord) {

COORD current = GetConsoleCursorPosition(hStdout);

int Q = current.Y;

int num;

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

{

destCoord.X = j \* 3 + 1;

if (j % 2 == 0)

{

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

{

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

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

destCoord.Y = i + Q;

SetConsoleCursorPosition(hStdout, destCoord);

cout << num << '\r';

cout.flush();

Sleep(delay);

}

}

else

{

for (int i = N - 1; i >= 0; --i)

{

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

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

destCoord.Y = i + Q;

SetConsoleCursorPosition(hStdout, destCoord);

cout << num << '\r';

cout.flush();

Sleep(delay);

}

}

}

destCoord.X = 0;

destCoord.Y = Q + N;

SetConsoleCursorPosition(hStdout, destCoord);

}

void FillMatrixB(int\* matrix, int N, const int& delay, HANDLE hStdout, COORD destCoord) {

COORD current = GetConsoleCursorPosition(hStdout);

int Q = current.Y;

int num, k = 1;

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

while (k <= N \* N)

{

destCoord.Y = top + Q;

for (int j = left; j <= right; ++j)

{

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

\*(matrix + top \* N + j) = num;

destCoord.X = j \* 3 + 1;;

SetConsoleCursorPosition(hStdout, destCoord);

cout << num << '\r';

cout.flush();

Sleep(delay);

k++;

}

top++;

destCoord.X = right \* 3 + 1;

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

{

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

\*(matrix + i \* N + right) = num;

destCoord.Y = i + Q;

SetConsoleCursorPosition(hStdout, destCoord);

cout << num << '\r';

cout.flush();

Sleep(delay);

k++;

}

right--;

destCoord.Y = bottom + Q;

if (top <= bottom)

{

for (int j = right; j >= left; --j)

{

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

\*(matrix + bottom \* N + j) = num;

destCoord.X = j \* 3 + 1;

SetConsoleCursorPosition(hStdout, destCoord);

cout << num << '\r';

cout.flush();

Sleep(delay);

k++;

}

bottom--;

}

destCoord.X = left \* 3 + 1;

if (left <= right)

{

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

{

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

\*(matrix + i \* N + left) = num;

destCoord.Y = i + Q;

SetConsoleCursorPosition(hStdout, destCoord);

cout << num << '\r';

cout.flush();

Sleep(delay);

k++;

}

left++;

}

}

destCoord.X = 0;

destCoord.Y = N + Q;

SetConsoleCursorPosition(hStdout, destCoord);

}

void ReplaceBlocksA(int\* matrix, int\* newMatrix, int N)

{

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

{

if ((i + 1) % N <= N / 2 && (i + 1) % N != 0 && i + 1 <= N \* N / 2)

{

\*(newMatrix + i + N / 2) = \*(matrix + i);

}

else if (((i + 1) % N > N / 2 || (i + 1) % N == 0) && i + 1 <= N \* N / 2)

{

\*(newMatrix + i + N \* N / 2) = \*(matrix + i);

}

else if ((i + 1) % N <= N / 2 && (i + 1) % N != 0 && i + 1 > N \* N / 2)

{

\*(newMatrix + i - N \* N / 2) = \*(matrix + i);

}

else if (((i + 1) % N > N / 2 || (i + 1) % N == 0) && i + 1 > N \* N / 2)

{

\*(newMatrix + i - N / 2) = \*(matrix + i);

}

}

}

void ReplaceBlocksB(int\* matrix, int\* newMatrix, int N)

{

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

{

if ((i + 1) % N <= N / 2 && (i + 1) % N != 0 && i + 1 <= N \* N / 2)

{

\*(newMatrix + i + N / 2 + N \* N / 2) = \*(matrix + i);

}

else if (((i + 1) % N > N / 2 || (i + 1) % N == 0) && i + 1 <= N \* N / 2)

{

\*(newMatrix + i + N \* N / 2 - N / 2) = \*(matrix + i);

}

else if ((i + 1) % N <= N / 2 && (i + 1) % N != 0 && i + 1 > N \* N / 2)

{

\*(newMatrix + i + N / 2 - N \* N / 2) = \*(matrix + i);

}

else if (((i + 1) % N > N / 2 || (i + 1) % N == 0) && i + 1 > N \* N / 2)

{

\*(newMatrix + i - N / 2 - N \* N / 2) = \*(matrix + i);

}

}

}

void ReplaceBlocksC(int\* matrix, int\* newMatrix, int N)

{

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

{

if ((i + 1) % N <= N / 2 && (i + 1) % N != 0 && i + 1 <= N \* N / 2)

{

\*(newMatrix + i + N \* N / 2) = \*(matrix + i);

}

else if (((i + 1) % N > N / 2 || (i + 1) % N == 0) && i + 1 <= N \* N / 2)

{

\*(newMatrix + i + N \* N / 2) = \*(matrix + i);

}

else if ((i + 1) % N <= N / 2 && (i + 1) % N != 0 && i + 1 > N \* N / 2)

{

\*(newMatrix + i - N \* N / 2) = \*(matrix + i);

}

else if (((i + 1) % N > N / 2 || (i + 1) % N == 0) && i + 1 > N \* N / 2)

{

\*(newMatrix + i - N \* N / 2) = \*(matrix + i);

}

}

}

void ReplaceBlocksD(int\* matrix, int\* newMatrix, int N)

{

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

{

if ((i + 1) % N <= N / 2 && (i + 1) % N != 0 && i + 1 <= N \* N / 2)

{

\*(newMatrix + i + N / 2) = \*(matrix + i);

}

else if (((i + 1) % N > N / 2 || (i + 1) % N == 0) && i + 1 <= N \* N / 2)

{

\*(newMatrix + i - N / 2) = \*(matrix + i);

}

else if ((i + 1) % N <= N / 2 && (i + 1) % N != 0 && i + 1 > N \* N / 2)

{

\*(newMatrix + i + N / 2) = \*(matrix + i);

}

else if (((i + 1) % N > N / 2 || (i + 1) % N == 0) && i + 1 > N \* N / 2)

{

\*(newMatrix + i - N / 2) = \*(matrix + i);

}

}

}

void InsertSort(int\* matrix, int N)

{

int key, j;

for (int i = 1; i != N; ++i)

{

key = \*(matrix + i);

j = i - 1;

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

{

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

j -= 1;

}

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

}

}

void Multiply(int\* matrix, int N, int X)

{

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

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

}

void Divide(int\* matrix, int N, int X)

{

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

\*(matrix + i) /= X;

}

void Add(int\* matrix, int N, int X)

{

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

\*(matrix + i) += X;

}

void Subtract(int\* matrix, int N, int X)

{

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

\*(matrix + i) -= X;

}

int main() {

setlocale(0, "");

srand(time(0));

HANDLE hStdout;

COORD destCoord;

destCoord.X = 0;

destCoord.Y = 0;

hStdout = GetStdHandle(STD\_OUTPUT\_HANDLE);

int N, delay = 10;

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

cin >> N;

InputCheck(N);

int newMatrix6[6][6];

int matrix6[6][6];

int newMatrix8[8][8];

int matrix8[8][8];

int newMatrix10[10][10];

int matrix10[10][10];

int\* matrix;

int\* newMatrix;

switch (N)

{

case 6:

matrix = \*matrix6;

newMatrix = \*newMatrix6;

break;

case 8:

matrix = \*matrix8;

newMatrix = \*newMatrix8;

break;

case 10:

matrix = \*matrix10;

newMatrix = \*newMatrix10;

break;

default:

cout << "Порядок матрицы должен быть 6, 8 или 10." << endl;

return 1;

}

int choice;

cout << "Введите номер схемы заполнения матрицы: ";

cin >> choice;

InputCheck(choice);

switch (choice)

{

case 1:

FillMatrixA(matrix, N, delay, hStdout, destCoord);

break;

case 2:

FillMatrixB(matrix, N, delay, hStdout, destCoord);

break;

default:

cout << "Нет такого варианта" << endl;

}

cout << "Введите номер схемы перестановки блоков матрицы: ";

cin >> choice;

InputCheck(choice);

switch (choice)

{

case 1:

ReplaceBlocksA(matrix, newMatrix, N);

break;

case 2:

ReplaceBlocksB(matrix, newMatrix, N);

break;

case 3:

ReplaceBlocksC(matrix, newMatrix, N);

break;

case 4:

ReplaceBlocksD(matrix, newMatrix, N);

break;

default:

cout << "Нет такого варианта" << endl;

}

int i = 0;

for (int\* ptr = newMatrix; ptr < newMatrix + N \* N; ++ptr)

{

cout << \*ptr << " ";

if ((i + 1) % N == 0)

cout << "\n";

i++;

}

InsertSort(newMatrix, N \* N);

cout << "Отсортированная матрица: " << endl;

i = 0;

for (int\* ptr = newMatrix; ptr < newMatrix + N \* N; ++ptr)

{

cout << \*ptr << " ";

if ((i + 1) % N == 0)

cout << "\n";

i++;

}

int X;

cout << "Введите число: ";

cin >> X;

InputCheck(X);

cout << "1 - сложение\n2 - вычитание\n3 - умножение\n4 - деление" << endl;

cout << "Введите номер операции: ";

cin >> choice;

InputCheck(choice);

switch (choice)

{

case 1:

Add(newMatrix, N, X);

break;

case 2:

Subtract(newMatrix, N, X);

break;

case 3:

Multiply(newMatrix, N, X);

break;

case 4:

Divide(newMatrix, N, X);

break;

default:

cout << "Нет такого варианта." << endl;

}

i = 0;

for (int\* ptr = newMatrix; ptr < newMatrix + N \* N; ++ptr)

{

cout << \*ptr << " ";

if ((ptr - newMatrix + 1) % N == 0)

cout << "\n";

i++;

}

return 0;

}