**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 |  | Гечис В.Р. |
| Преподаватель |  | Глущенко А. Г. |

Санкт-Петербург

2022

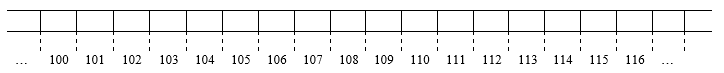
**Цель работы.**

Изучение двумерных статических массив и основных алгоритмов работы с ними. Изучение ссылок и указателей, арифметики указателей. Написание программы, заполняющей матрицу размера N (N = 6, 8, 10) и заполняющий его паттернами «snakd» и «spiral». Программа должна так же менять местами блоки (подмассивы) матрицы в соответствии с четырьмя паттернами обмена из задания. Сортировать матрицу с использованием арифметики указателей и увеличивать, уменьшать, умножать и делить все элементы матрицы на число, введенное пользователем

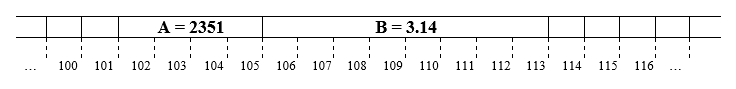
**Основные теоретические положения.**

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 101 и занимает 4 байта, а переменная **B** имеет адрес 105 и занимает 8 байт памяти.

**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.

Указатели поддерживают ряд операций: присваивание, получение адреса указателя, получение значения по указателю, некоторые арифметические операции и операции сравнения.

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.

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.

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).

**Постановка задач.**

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:



1. Используя арифметику указателей, сортирует элементы любой сортировкой.
2. Уменьшает, увеличивает, умножает или делит все элементы матрицы на введенное пользователем число.
3. Написать для каждого задания функцию, структурировать код.
4. Вывести двумерный массив двумя способами, используя работу с кареткой.
5. Отсортировать двумерный массив сортировкой «Вставками», не обращаясь к индексам элементов. (используя арифметику указателей)

**Выполнение работы.**

Код программы представлен в приложении А.

1. При запуске программы пользователь вводит размер квадратной матрицы.
2. Массив выводится змейкой и спиралью
3. Пользователю предлагается ввести число и операцию. Массив умножается/делится/уменьшается/увеличивается на введенное пользователем число.

**Выводы.**

В ходе проведения практической работы были изучены двумерные статические массивы и указатели, работа с кареткой. Результатом является программа, которая позволяет выполнить все необходимые задачи.

**Выполнение работы.**

|  |  |
| --- | --- |
| Ввод пользователем и обработка данных | Работа алгоритма и вывод на экран |
| 1.Заполнение матрицы | |
| Программа создает, создает матрицу N\*N, выводит её двумя способами. |  |

Продолжение Таблицы

|  |  |
| --- | --- |
| 2. Получение новой матрицы | |
| Получает новую матрицу путем перестановки ее блоков |  |

|  |  |
| --- | --- |
| 4 Сортировка | |
| Сортируется матрица с помощью указателей |  |

|  |  |
| --- | --- |
| 5 Изменение значений матрицы | |
| Уменьшает, увеличивает, умножает или делит все элементы матрицы на введенное пользователем число. |  |

|  |  |
| --- | --- |
| Цикличность программы | |
| Eсли пользователь желает, то он может запустить выполнение программы заново, нажав цифру 1 |  |

Продолжение Таблицы

**Полный код программы:**

#include <iostream>  
#include <Windows.h>  
#include <iostream>  
#include <windows.h>  
#include <chrono>  
#include <thread>  
  
using namespace std;  
  
int randInt(int from, int to) {  
 return from + std::rand() % to;  
}  
void setCursorPosition(int x, int y)  
{  
 COORD coord;  
 coord.X = x;  
 coord.Y = y;  
 SetConsoleCursorPosition(GetStdHandle(STD\_OUTPUT\_HANDLE), coord);  
}  
  
void printMatrix(int \*first,int N){  
 for (int \*i = first, \*end = i + N\*N-1, count = 1; i <= end; count++, i++){  
 cout.width(6);  
 cout << \*i << " ";  
 if (count % N == 0){ cout << "\n" ;}  
  
 }  
 cout << "\n";  
}  
  
void spiralFill(int \*start, int N){  
 int countPrinted = 0;  
 int countIterations = 0;  
 getchar();  
 system("cls");  
  
  
 HANDLE hStdout;  
 COORD destCoord;  
 hStdout = GetStdHandle(STD\_OUTPUT\_HANDLE);  
  
 int countGlobalIterations = 0;  
 for (;countPrinted != N \* N;) {  
  
 for (int \*first = (start + countPrinted), \*last = first + (N - 1), i = countGlobalIterations \* (countGlobalIterations > 0? 3:0); i < ((N \* 3) - (3\* countGlobalIterations)) && first <= last; i += 3, first++) {  
  
 setCursorPosition(i,countGlobalIterations);  
 \*first = randInt(1,N\*N);  
 cout << \*first;  
 countPrinted += 1;  
 cout.flush();  
 Sleep(300);  
  
 }  
 for (int \*first = (start + countPrinted), \*last = first + (N - 1), i = countGlobalIterations + 1; i < (N - countGlobalIterations) && first < last; first++, i++) {  
 setCursorPosition((N-1) \* 3 - countGlobalIterations\*3,i);  
 \*first = randInt(1,N\*N);  
  
 cout << \*first;  
 countPrinted += 1;  
  
 cout.flush();  
 Sleep(300);  
  
 }  
 for (int \*first = (start + countPrinted), \*last = first + (N - 1), i = (N-1) \* 3 - (countGlobalIterations +1)\*3;  
 i >= countGlobalIterations \* 3 && first <= last; first++, i -= 3) {  
  
  
 setCursorPosition(i, N -1 - countGlobalIterations);  
 \*first = randInt(1,N\*N);  
  
  
 cout << \*first;  
 countPrinted += 1;  
  
  
 cout.flush();  
 Sleep(300);  
  
 }  
  
 for (int \*first = (start + countPrinted), \*last = first + (N - 1), i = N - 2 - countGlobalIterations; i >= (1 + countGlobalIterations) && first <= last; i--, first++) {  
  
 setCursorPosition(countGlobalIterations \* 3, i);  
  
 \*first = randInt(1,N\*N);  
 cout << \*first;  
 countPrinted += 1;  
  
  
 cout.flush();  
 Sleep(300);  
  
 }  
 countGlobalIterations +=1;  
 }  
  
}  
  
void snakeFill(int \*start, int N){  
// getchar();  
// system("cls");  
 HANDLE hStdout;  
 COORD destCoord;  
 hStdout = GetStdHandle(STD\_OUTPUT\_HANDLE);  
 int countPrinted = 0;  
 int countGlobalIterations = 0;  
 int countIterations = 0;  
  
 for (;countPrinted != N \* N;) {  
  
 for (int \*first = (start + countPrinted), \*last = first + (N - 1), i = 0; first <= last; i++, first++) {  
 setCursorPosition((countIterations \* 3 + (countGlobalIterations>0?1:0) + (N-1)\*3\*2), i);  
 \*first = randInt(1,N\*N);  
 cout << \*first;  
 countPrinted += 1;  
  
  
 cout.flush();  
 Sleep(300);  
 }  
 countIterations +=1;  
  
 for (int \*first = (start + countPrinted), \*last = first + (N - 1), i = N-1; first <= last; i--, first++) {  
 setCursorPosition((countIterations \* 3 + (countGlobalIterations>0?1:0) + (N-1)\*3\*2), i);  
 \*first = randInt(1,N\*N);  
 cout << \*first;  
 countPrinted += 1;  
  
 cout.flush();  
 Sleep(300);  
  
 }  
 countIterations +=1;  
  
 countGlobalIterations += 1;  
 }  
  
}  
  
 void arrsForSwap(int \*start, int\*start1, int\*start2, int\*start3, int\*start4, int N){  
 int count = 0;  
 for (int k = 1; k<=N/2; k++) {  
 for (int \*first = start + count, \*end = first + N-1, x = 0, y = 0; first <= end; first++, x++, y++) {  
  
 if (x < N/2) {  
 \*start1 = \*first;  
 start1 += 1;  
 count +=1;  
 }  
 if (x >= N/2) {  
 \*start2 = \*first;  
 start2 += 1;  
 count +=1;  
 }  
 }  
 }  
 for (int k = 1; k<=N/2; k++) {  
 for (int \*first = start + count, \*end = first + N-1, x = 0, y = 0; first <= end; first++, x++, y++) {  
  
 if (x < N/2) {  
 \*start4 = \*first;  
 start4 += 1;  
 count +=1;  
 }  
 if (x >= N/2) {  
 \*start3 = \*first;  
 start3 += 1;  
 count +=1;  
 }  
 }  
 }  
  
}  
void swapCircle(int \*start, int\*start1, int\*start2, int\*start3, int\*start4, int N){  
 int count = 0;  
 for (int k = 1; k<=N/2; k++) {  
 for (int \*first = start + count, \*end = first + N-1, x = 0, y = 0; first <= end; first++, x++, y++) {  
  
 if (x < N/2) {  
 \*first = \*start4;  
 start4 += 1;  
 count +=1;  
 }  
 if (x >= N/2) {  
 \*first = \*start1;  
 start1 += 1;  
 count +=1;  
 }  
 }  
 }  
 for (int k = 1; k<=N/2; k++) {  
 for (int \*first = start + count, \*end = first + N-1, x = 0, y = 0; first <= end; first++, x++, y++) {  
  
 if (x < N/2) {  
 \*first = \*start3;  
 start3 += 1;  
 count +=1;  
 }  
 if (x >= N/2) {  
 \*first = \*start2;  
 start2 += 1;  
 count +=1;  
 }  
 }  
 }  
  
}  
void swapDiagonal(int \*start, int\*start1, int\*start2, int\*start3, int\*start4, int N){  
 int count = 0;  
 for (int k = 1; k<=N/2; k++) {  
 for (int \*first = start + count, \*end = first + N-1, x = 0, y = 0; first <= end; first++, x++, y++) {  
  
 if (x < N/2) {  
 \*first = \*start3;  
 start3 += 1;  
 count +=1;  
 }  
 if (x >= N/2) {  
 \*first = \*start4;  
 start4 += 1;  
 count +=1;  
 }  
 }  
 }  
 for (int k = 1; k<=N/2; k++) {  
 for (int \*first = start + count, \*end = first + N-1, x = 0, y = 0; first <= end; first++, x++, y++) {  
  
 if (x < N/2) {  
 \*first = \*start2;  
 start2 += 1;  
 count +=1;  
 }  
 if (x >= N/2) {  
 \*first = \*start1;  
 start1 += 1;  
 count +=1;  
 }  
 }  
 }  
  
}  
void swapVertical(int \*start, int\*start1, int\*start2, int\*start3, int\*start4, int N){  
 int count = 0;  
 for (int k = 1; k<=N/2; k++) {  
 for (int \*first = start + count, \*end = first + N-1, x = 0, y = 0; first <= end; first++, x++, y++) {  
  
 if (x < N/2) {  
 \*first = \*start4;  
 start4 += 1;  
 count +=1;  
 }  
 if (x >= N/2) {  
 \*first = \*start3;  
 start3 += 1;  
 count +=1;  
 }  
 }  
 }  
 for (int k = 1; k<=N/2; k++) {  
 for (int \*first = start + count, \*end = first + N-1, x = 0, y = 0; first <= end; first++, x++, y++) {  
  
 if (x < N/2) {  
 \*first = \*start1;  
 start1 += 1;  
 count +=1;  
 }  
 if (x >= N/2) {  
 \*first = \*start2;  
 start2 += 1;  
 count +=1;  
 }  
 }  
 }  
  
}  
  
void swapHorizontal(int \*start, int\*start1, int\*start2, int\*start3, int\*start4, int N){  
 int count = 0;  
 for (int k = 1; k<=N/2; k++) {  
 for (int \*first = start + count, \*end = first + N-1, x = 0; first <= end; first++, x++) {  
  
 if (x < N/2) {  
 \*first = \*start2;  
 start2 += 1;  
 count +=1;  
 }  
 if (x >= N/2) {  
 \*first = \*start1;  
 start1 += 1;  
 count +=1;  
 }  
 }  
 }  
 for (int k = 1; k<=N/2; k++) {  
 for (int \*first = start + count, \*end = first + N-1, x = 0; first <= end; first++, x++) {  
  
 if (x < N/2) {  
 \*first = \*start3;  
 start3 += 1;  
 count +=1;  
 }  
 if (x >= N/2) {  
 \*first = \*start4;  
 start4 += 1;  
 count +=1;  
 }  
 }  
 }  
  
}  
  
void arr\_sort(int \*arr, int N){  
  
 for (int \*first = arr + 1, \*last = first + N\*N-1; first <= last; first++){  
 for (int \*parr = first;parr > arr;parr-- ){  
 if (\*parr < \*(parr - 1)) {  
 swap(\*parr,\*(parr - 1));  
 }  
 }  
  
 }  
}  
void userOperation(int \*start, int N, int userChoice, int userValue){  
 switch(userChoice){  
 case 1:  
 for (int \*first = start, \*last = first + ((N \* N) - 1), i = 0; first <= last; first++, i++) {  
 \*first = \*first \* userValue;  
 }  
 break;  
  
 case 2:  
 for (int \*first = start, \*last = first + ((N \* N) - 1), i = 0; first <= last; first++, i++) {  
 \*first = \*first / userValue;  
  
 }  
 break;  
  
 case 3:  
 for (int \*first = start, \*last = first + ((N \* N) - 1), i = 0; first <= last; first++, i++) {  
 \*first = \*first + userValue;  
  
 }  
 break;  
  
 case 4:  
 for (int \*first = start, \*last = first + ((N \* N) - 1), i = 0; first <= last; first++, i++) {  
 \*first = \*first - userValue;  
  
 }  
 break;  
  
  
 }  
  
  
}  
  
void magicSquare(int N) {  
 srand(time(0));  
 int arr[N][N];  
// Магический квадрат для проверки работы алгоритма  
// int arr[3][3] = {3,4,2,  
// 2,3,4,  
// 4,2,3};  
 int sum[N + N + 2];  
 int \*psum = sum;  
 int countSummed = 0;  
 for (int \*first = sum, \*last = first + (N + N + 1), i = 0; first <= last; first++, i++) {  
 \*first = 0;  
 }  
  
 int \*start = &arr[0][0];  
 for (int \*first = start, \*last = first + ((N \* N) - 1), i = 0; first <= last; first++, i++) {  
 \*first = rand() % 100 + 1;  
 }  
 printMatrix(start, N);  
  
 int count = 0;  
  
 for (int k = 1; k <= N; k++) {  
 for (int \*first = start + count, \*end = first + N - 1; first <= end; first++) {  
  
 \*psum = \*psum + \*first;  
 count += 1;  
 }  
 cout << "\nSum of " << k << " line: " << \*psum;  
  
 psum += 1;  
 }  
 cout << "\n-----------------------------------\n";  
 for (int k = 1; k <= N; k++) {  
 for (int \*first = start + (k - 1), \*end = first + (N \* N - N); first <= end; first += N) {  
  
 \*psum = \*psum + \*first;  
 count += 1;  
 }  
 cout << "\nSum of " << k << " line: " << \*psum;  
  
 psum += 1;  
 }  
 cout << "\n-----------------------------------\n";  
 for (int k = 1; k <= N; k++) {  
 int \*first = start + (k - 1) \* N + k - 1;  
  
 \*psum = \*psum + \*first;  
 count += 1;  
 }  
 cout << "\nSum of main diagonal: " << \*psum;  
  
 cout << "\n-----------------------------------\n";  
  
 psum += 1;  
 for (int k = 1; k <= N; k++) {  
 int \*first = start + (N - 1) \* k;  
  
 \*psum = \*psum + \*first;  
 count += 1;  
 }  
 cout << "\nSum of main diagonal: " << \*psum;  
  
  
 cout << "\n-----------------------------------\n";  
  
  
 for (int i = 0; i < N + N + 2; i++) {  
 for (int j = 0; j< N + N + 2; j++) {  
 if (sum[i] != sum[j]) {  
 cout << "\nNOT MAGIC\n";  
 return;  
 }  
 }  
 }  
 cout << "\nMAGIC\n";  
  
}  
  
  
  
int main(){  
  
 int mainLoop;  
  
 cout << "'1' - Start the program\n'0' - End the program\n";  
 cin >> mainLoop;  
  
 while(mainLoop) {  
  
 int choiceIdz;  
 cout << "'1' - Start the IDZ\n'0' - Go to the main program\n";  
 cin >> choiceIdz;  
 while(choiceIdz) {  
  
 int sizeIdz;  
 cout << "Enter size of the matrix:\n";  
 cin >> sizeIdz;  
 magicSquare(sizeIdz);  
 cout << "\n'1' - Repeat task '0' - Go to the main program:\n";  
 cin >> choiceIdz;  
 }  
  
  
 int N;  
 getchar();  
 system("cls");  
 cout << "Enter size of the matrix:\n";  
 cin >> N;  
 getchar();  
 system("cls");  
 int arr[N][N];  
  
 int array1[N / 2][N / 2];  
 int array2[N / 2][N / 2];  
 int array3[N / 2][N / 2];  
 int array4[N / 2][N / 2];  
 int \*start = &arr[0][0];  
  
 int \*start1 = &array1[0][0];  
  
 int \*start2 = &array2[0][0];  
  
 int \*start3 = &array3[0][0];  
  
 int \*start4 = &array4[0][0];  
  
 spiralFill(start, N);  
 snakeFill(start, N);  
 setCursorPosition(0, N \* 2);  
 cout << "Press any key to continue:\n";  
 cin.get();  
 getchar();  
 system("cls");  
 printMatrix(start, N);  
 cout << "----------------\n";  
  
 cout << "\na) Circle swap\n";  
  
 arrsForSwap(start, start1, start2, start3, start4, N);  
 swapCircle(start, start1, start2, start3, start4, N);  
 printMatrix(start, N);  
  
 Sleep(500);  
 cout << "\nb) Diagonal swap\n";  
  
  
 arrsForSwap(start, start1, start2, start3, start4, N);  
 swapDiagonal(start, start1, start2, start3, start4, N);  
 printMatrix(start, N);  
  
 Sleep(500);  
 cout << "\nc) Vertical swap\n";  
  
  
 arrsForSwap(start, start1, start2, start3, start4, N);  
 swapVertical(start, start1, start2, start3, start4, N);  
 printMatrix(start, N);  
  
 Sleep(500);  
 cout << "\nd) Horizontal swap\n";  
  
  
 arrsForSwap(start, start1, start2, start3, start4, N);  
 swapHorizontal(start, start1, start2, start3, start4, N);  
 printMatrix(start, N);  
 Sleep(1000);  
 cout << "\n-------------------------------------\nArray after sort:\n";  
  
  
 arr\_sort(start, N);  
 printMatrix(start, N);  
  
  
 int userChoiceOper, userValue, userChoice;  
 cout << "\n'1' - Start next task '0' - Go to next task:\n";  
 cin >> userChoice;  
  
  
 while (userChoice) {  
 getchar();  
 system("cls");  
 printMatrix(start, N);  
 cout << "\nchoose the operation:\n";  
 cout << "1 - '\*'\n2 - '/'\n3 - '+'\n4 - '-'\n";  
 cin >> userChoiceOper;  
 cout << "Enter an integer value:\n";  
 cin >> userValue;  
 userOperation(start, N, userChoiceOper, userValue);  
 printMatrix(start, N);  
 cout << "\n'1' - Repeat task '0' - End the program :c :\n";  
 cin >> userChoice;  
  
 }  
 cout << "\n'1' - Repeat program '0' - End the program :c :\n";  
 cin >> mainLoop;  
  
 }  
  
}