**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 |  | Музыченко И.А. |
| Преподаватель |  | Глущенко А. Г. |

Санкт-Петербург

2024

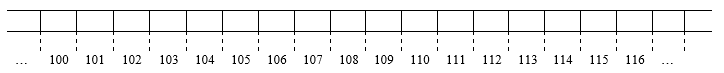
**Цель работы.**

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

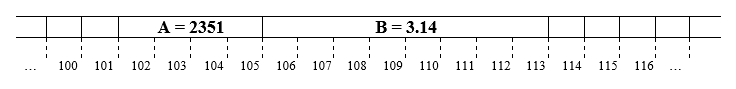
**Основные теоретические положения.**

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).

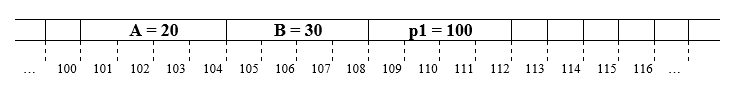
Формально указатели представляют собой обычные целые значения типа **int** и занимают в памяти 4 байта не зависимо от базового типа указателя. Значения указателей при их выводе на экран представляются как целые значения в шестнадцатеричном формате.

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):



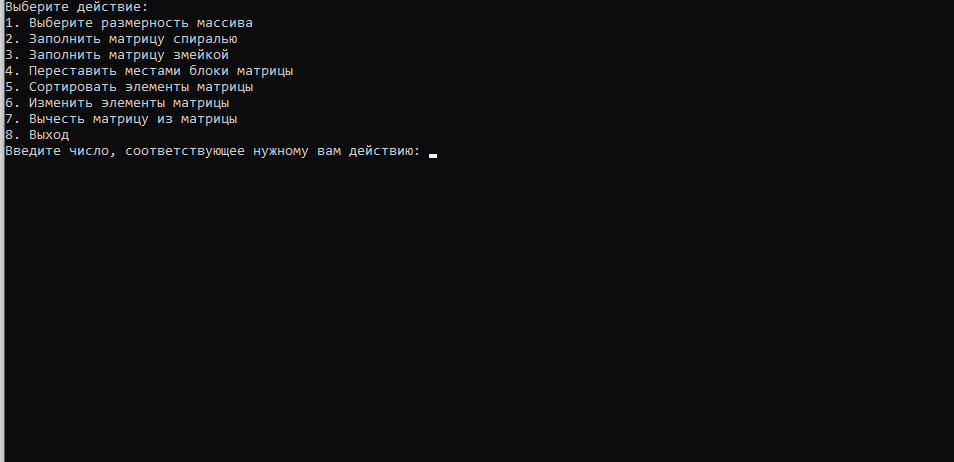
3) Используя арифметику указателей, сортирует элементы любой сортировкой из списка ниже (если во 2 ПР вы реализовывали одну из представленных сортировок, рекомендуется переиспользовать написанный код и модернизировать его для работы с указатями).   
Варианты сортировок:

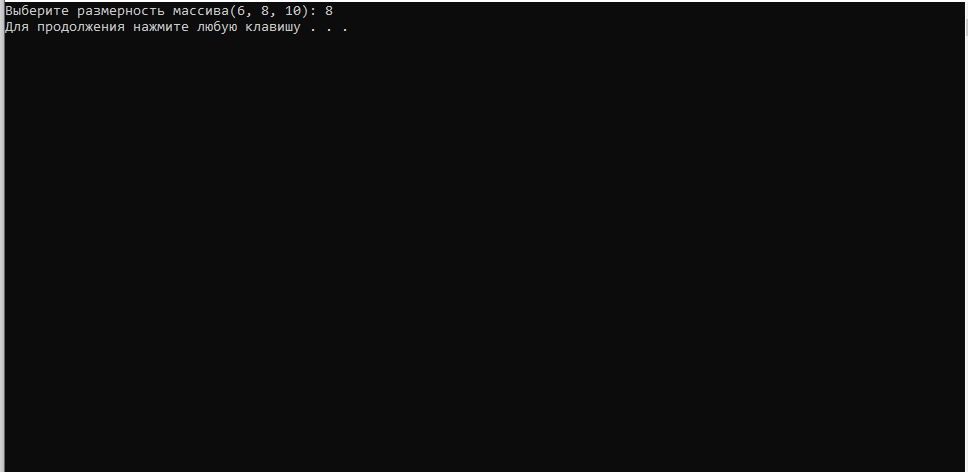
* Shaker sort;
* Comb sort;
* Insert sort;
* Quick sort;

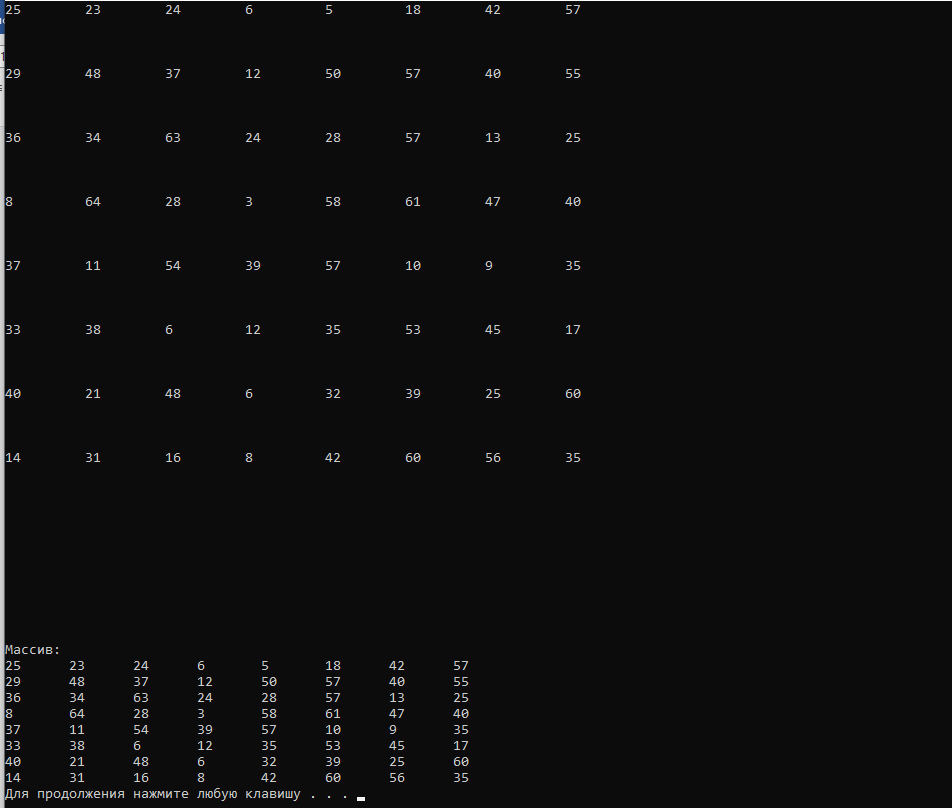
4) Уменьшает, увеличивает, умножает или делит все элементы матрицы на введенное пользователем число (\*Для манипуляции с элементами используйте только арифметику указателей).

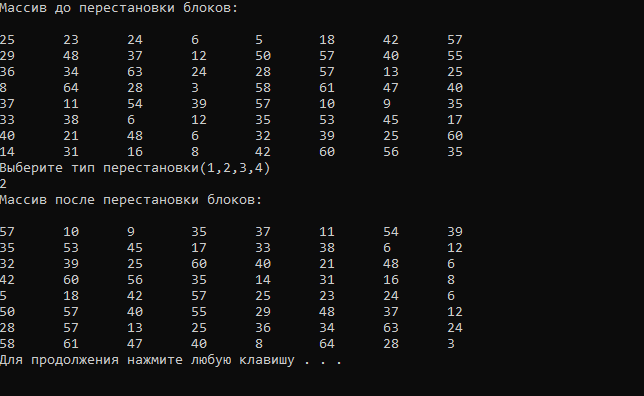
**Выполнение работы.**

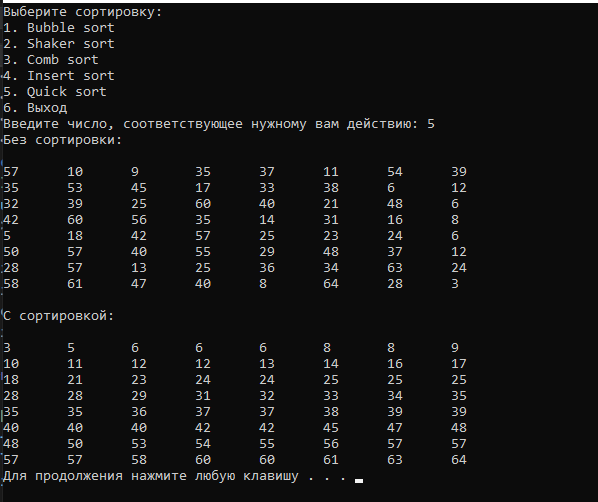
Код программы представлен в приложении А.

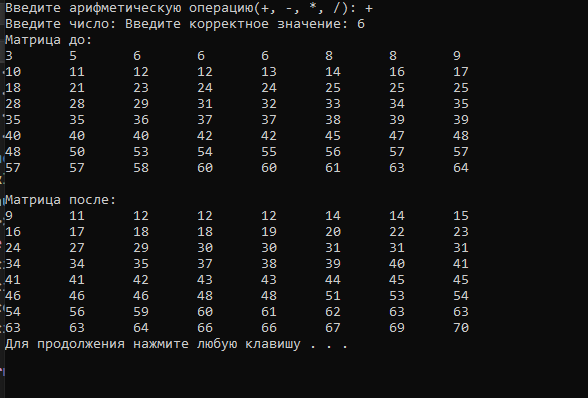


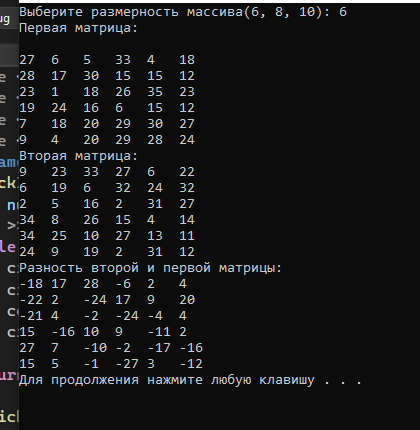


****

****

****

****

****

**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 <Windows.h>

#include <limits>

#include <iomanip>

using namespace std;

int checkInput(int floor = (numeric\_limits<int>::min)(), int ceiling = (numeric\_limits<int>::max)()) {

int num;

cin >> num;

while (cin.fail() || cin.peek() != '\n' || num > ceiling || num < floor) {

cin.clear();

cin.ignore((numeric\_limits<streamsize>::max)(), '\n');

cout << "Введите корректное значение: ";

cin >> num;

}

return num;

}

void quickSort(int\* begin, int\* end) {

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) quickSort(begin, l);

if (f < end) quickSort(f, end);

}

void combSort(int\* arr, int N) {

float k = 1.247;

int s = N;

bool swapped = false;

while (s > 1 || swapped) {

if (s > 1) {

s /= k;

}

swapped = false;

int\* ptr1 = arr;

int\* ptr2 = arr + s;

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

if (\*ptr1 > \*ptr2) {

swap(\*ptr1, \*ptr2);

swapped = true;

}

ptr1++;

ptr2++;

}

}

}

void insertSort(int\* arr, int N) {

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

int key = \*(arr + i);

int j = i - 1;

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

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

j = j - 1;

}

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

}

}

void shakerSort(int\* arr, int N) {

int\* begin = arr;

int\* end = arr + N \* N - 1;

bool swapped = true;

while (swapped) {

swapped = false;

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

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

swap(\*(i - 1), \*i);

swapped = true;

}

}

if (!swapped) {

break;

}

swapped = false;

for (int\* i = end - 1; i >= begin; i--) {

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

swap(\*(i - 1), \*i);

swapped = true;

}

}

}

}

void bubbleSort(int\* arr, int N) {

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

for (int\* j = arr; j - arr < end - i; j++) {

if (\*j > \*(j + 1))

swap(\*j, \*(j + 1));

}

}

}

void mainMenu() {

system("cls");

cout << "Выберите действие: " << endl;

cout << "1. Выберите размерность массива" << endl;

cout << "2. Заполнить матрицу спиралью" << endl;

cout << "3. Заполнить матрицу змейкой" << endl;

cout << "4. Переставить местами блоки матрицы" << endl;

cout << "5. Сортировать элементы матрицы" << endl;

cout << "6. Изменить элементы матрицы" << endl;

cout << "7. Вычесть матрицу из матрицы" << endl;

cout << "8. Выход" << endl;

cout << "Введите число, соответствующее нужному вам действию: ";

}

void choosingSortMenu() {

system("cls");

cout << "Выберите сортировку: " << endl;

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

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

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

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

cout << "5. Quick sort" << endl;

cout << "6. Выход" << endl;

cout << "Введите число, соответствующее нужному вам действию: ";

}

void swapPointers(int\* p1, int\* p2) {

int tmp = \*p1;

\*p1 = \*p2;

\*p2 = tmp;

}

int matrixSize() {

int size;

cout << "Выберите размерность массива(6, 8, 10): ";

cin >> size;

switch (size) {

case 6:

return 6;

cout << "Выбран размер 6 на 6" << endl;

break;

case 8:

return 8;

cout << "Выбран размер 8 на 8" << endl;

break;

case 10:

return 10;

cout << "Выбран размер 10 на 10" << endl;

break;

default:

cout << "Неверный выбор размера массива, по умолчанию массив будет 6 на 6" << endl;

return 6;

break;

}

}

void printArr(int\* arr, int N) {

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

for (int\* ptr = arr; ptr <= end; ptr++) {

if ((ptr - arr) % N == 0)

cout << "\n";

cout << \*ptr << "\t";

}

cout << endl;

}

void minusMatrix() {

const int n = 10;

int N = matrixSize();

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

COORD destCoord;

int matrix[n][n], matrix2[n][n], matrixDif[n][n];

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

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

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

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

}

}

cout << "Первая матрица: " << endl;

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

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

destCoord.X = j \* 4;

destCoord.Y = i + 3;

SetConsoleCursorPosition(hStdout, destCoord);

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

}

}

cout << endl;

cout << "Вторая матрица: " << endl;

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

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

destCoord.X = j \* 4;

destCoord.Y = i + N + 4;

SetConsoleCursorPosition(hStdout, destCoord);

cout << \*(\*(matrix2 + i) + j) << " ";

}

}

cout << endl;

cout << "Разность второй и первой матрицы: " << endl;

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

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

\*(\*(matrixDif + i) + j) = \*(\*(matrix2 + i) + j) - \*(\*(matrix + i) + j);

int\* element = (\*(matrixDif + i) + j);

\*element = \*(\*(matrix2 + i) + j) - \*(\*(matrix + i) + j);

destCoord.X = j \* 4;

destCoord.Y = i + 2\*N + 5;

SetConsoleCursorPosition(hStdout, destCoord);

cout << \*element << '\r';

Sleep(150);

cout.flush();

cout << endl;

}

}

}

void changeElements(int \*arr, int N, int move, int num) {

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

switch (move) {

case 0:

\*p += num;

break;

case 1:

\*p -= num;

break;

case 2:

\*p \*= num;

break;

case 3:

if (num == 0)

break;

\*p /= num;

break;

}

}

}

void fillMatrixWorm(int \*arr, int N) {

HANDLE hStdout;

COORD destCoord;

hStdout = GetStdHandle(STD\_OUTPUT\_HANDLE);

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

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

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

int\* element = arr + i\*N + left;

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

destCoord.X = left \* 10;

destCoord.Y = i \* 4;

SetConsoleCursorPosition(hStdout, destCoord);

cout << \*element << '\r';

Sleep(100);

cout.flush();

}

left++;

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

int\* element = arr + i \* (N) + left;

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

destCoord.X = left \* 10;

destCoord.Y = i \* 4;

SetConsoleCursorPosition(hStdout, destCoord);

cout << \*element << '\r';

Sleep(100);

cout.flush();

}

left++;

}

destCoord.X = 0;

destCoord.Y = 40;

SetConsoleCursorPosition(hStdout, destCoord);

}

void fillMatrixSpiral(int \*arr, int N) {

HANDLE hStdout;

COORD destCoord;

hStdout = GetStdHandle(STD\_OUTPUT\_HANDLE);

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

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

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

int \*element = (arr + top\*N + i);

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

destCoord.X = i \* 10;

destCoord.Y = top \* 4;

SetConsoleCursorPosition(hStdout, destCoord);

cout << \*element << '\r';

Sleep(100);

cout.flush();

}

top++;

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

int\* element = (arr + i\*N + right);

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

destCoord.X = right \* 10;

destCoord.Y = i \* 4;

SetConsoleCursorPosition(hStdout, destCoord);

cout << \*element << '\r';

Sleep(100);

cout.flush();

}

right--;

if (top <= bottom) {

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

int\* element = (arr + bottom \* N + i);

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

destCoord.X = i \* 10;

destCoord.Y = bottom \* 4;

SetConsoleCursorPosition(hStdout, destCoord);

cout << \*element << '\r';

Sleep(100);

cout.flush();

}

bottom--;

}

if (left <= right) {

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

int\* element = (arr + (i ) \* N + left);

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

destCoord.X = left \* 10;

destCoord.Y = i \* 4;

SetConsoleCursorPosition(hStdout, destCoord);

cout << \*element << '\r';

Sleep(100);

cout.flush();

}

left++;

}

}

destCoord.X = 0;

destCoord.Y = 25;

SetConsoleCursorPosition(hStdout, destCoord);

}

void swapBlocks(int \*arr, int n) {

cout << "Выберите тип перестановки(1,2,3,4)\n";

int typeswap = checkInput(1, 4);

switch (typeswap) {

case 1: {

for (int i = 0; i < (n / 2); i++) {

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

swapPointers((arr + i \* n + j), (arr + i \* n + j + n / 2));

swapPointers((arr + i \* n + j + n / 2), (arr + (i + n / 2) \* n + j + n / 2));

swapPointers((arr + (i + n / 2) \* n + j + n / 2), (arr + i \* n + j));

swapPointers((arr + i \* n + j), (arr + (n / 2 + i) \* n + j));

}

}

break;

}

case 2: {

for (int i = 0; i < (n / 2); i++) {

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

swapPointers((arr + i \* n + j), (arr + (i + n / 2) \* n + j + n / 2));

swapPointers((arr + i \* n + j + n / 2), (arr + (i + n / 2) \* n + j));

}

}

break;

}

case 3: {

for (int i = 0; i < (n / 2); i++) {

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

swapPointers((arr + i \* n + j), (arr + (i + n / 2) \* n + j));

swapPointers((arr + i \* n + j + n / 2), (arr + (i + n / 2) \* n + j + n / 2));

}

}

break;

}

case 4: {

for (int i = 0; i < (n / 2); i++) {

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

swapPointers((arr + i \* n + j), (arr + (i) \* n + j + n/2));

swapPointers((arr + (i + n/2) \* n + j), (arr + (i + n / 2) \* n + j + n/2));

}

}

break;

}

}

}

void main() {

setlocale(0, "");

srand(time(NULL));

const int n = 10;

int N = 6;

int arr[n][n] = {0};

bool inizialized = 0;

int choice;

cin.sync();

do {

mainMenu();

choice = checkInput(1, 8);

switch (choice) {

case 1: {

system("cls");

N = matrixSize();

break;

}

case 2: {

system("cls");

fillMatrixSpiral(\*arr, N);

cout << "Массив:";

printArr(\*arr, N);

inizialized = 1;

break;

}

case 3: {

system("cls");

fillMatrixWorm(\*arr, N);

cout << "Массив:";

printArr(\*arr, N);

inizialized = 1;

break;

}

case 4: {

system("cls");

if (!inizialized) {

cout << "Сначала нужно создать массив" << endl;

break;

}

cout << "Массив до перестановки блоков: " << endl;

printArr(\*arr, N);

swapBlocks(\*arr, N);

cout << "Массив после перестановки блоков: " << endl;

printArr(\*arr, N);

break;

}

case 5: {

system("cls");

if (!inizialized) {

cout << "Сначала нужно создать массив" << endl;

break;

}

int choice2;

do {

choosingSortMenu();

choice2 = checkInput(1, 6);

switch (choice2) {

case 1:

cout << "Без сортировки: " << endl;

printArr(\*arr, N);

cout << "\nС сортировкой: " << endl;

bubbleSort(\*arr, N);

printArr(\*arr, N);

break;

case 2:

cout << "Без сортировки: " << endl;

printArr(\*arr, N);

cout << "\nС сортировкой: " << endl;

shakerSort(\*arr, N);

printArr(\*arr, N);

break;

case 3:

cout << "Без сортировки: " << endl;

printArr(\*arr, N);

cout << "\nС сортировкой: " << endl;

combSort(\*arr, N);

printArr(\*arr, N);

break;

case 4:

cout << "Без сортировки: " << endl;

printArr(\*arr, N);

cout << "\nС сортировкой: " << endl;

insertSort(\*arr, N);

printArr(\*arr, N);

break;

case 5:

{

cout << "Без сортировки: " << endl;

printArr(\*arr, N);

cout << "\nС сортировкой: " << endl;

quickSort(\*arr, \*arr + N\*N - 1);

printArr(\*arr, N);

break;

}

}

if (choice2 != 6)

system("pause");

} while (choice2 != 6);

if (choice2 == 6)

continue;

break;

}

case 6: {

system("cls");

if (!inizialized) {

cout << "Сначала нужно создать массив" << endl;

break;

}

cout << "Введите арифметическую операцию(+, -, \*, /): ";

int move;

cin >> move;

cout << "Введите число: ";

int num = checkInput();

cout << "Матрица до: ";

printArr(\*arr, N);

cout << endl;

changeElements(\*arr, N, move, num);

cout << "Матрица после: ";

printArr(\*arr, N);

break;

}

case 7:

system("cls");

minusMatrix();

break;

}

if (choice != 8)

system("pause");

} while (choice != 8);

}