**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра «Информационные Системы и Технологий (ИСиТ)»**

отчет

**по лабораторной работе №3**

**по дисциплине «Программирование»**

Тема: «Арифметика указателей и матрицы»

|  |  |  |
| --- | --- | --- |
| Студент гр. 0324 |  | Зимацкий С. Н. |
| Преподаватель |  | Глущенко А.Г. |

Санкт-Петербург

2020

**Цель работы.**

Изучить арифметику указателей, указатели массивов и анимацию вывода квадратных матриц в консоли.

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.



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



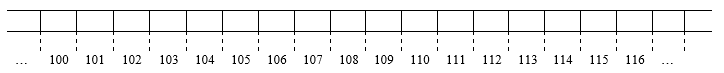
3)    Используя арифметику указателей, сортирует элементы любой сортировкой.

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

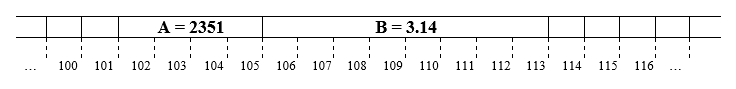
**Основные теоретические положения.**

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.

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: 101

Значение переменной В: 3.14

Адрес переменной В: 105

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.

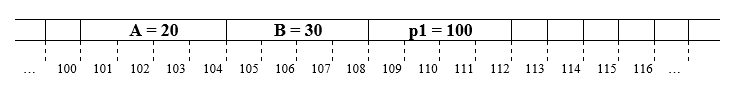
К указателям можно применять некоторые арифметические операции. К таким операциям относятся:  **+**,**-**, **++**, **--**. Результаты выполнения этих операций по отношению к указателям существенно отличаются от результатов соответствующих арифметических операций, выполняющихся с обычными числовыми данными.

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;

}

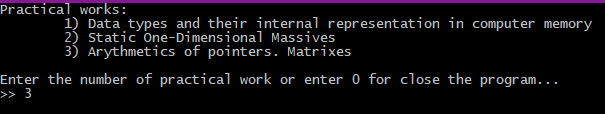
cout << endl;

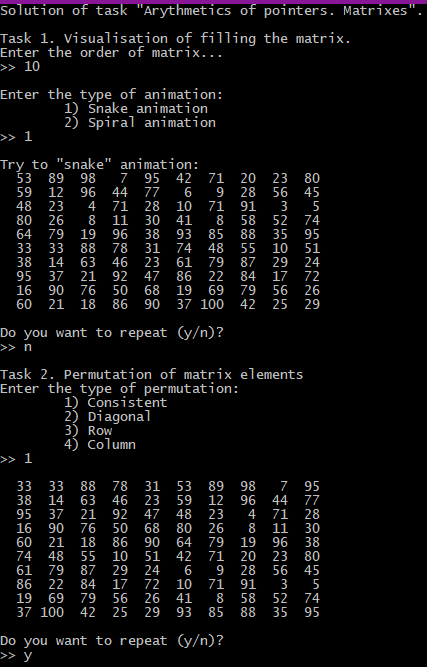
А это пятиэлементный массив указателей на символы, инициализированный некоторыми текстовыми строками:

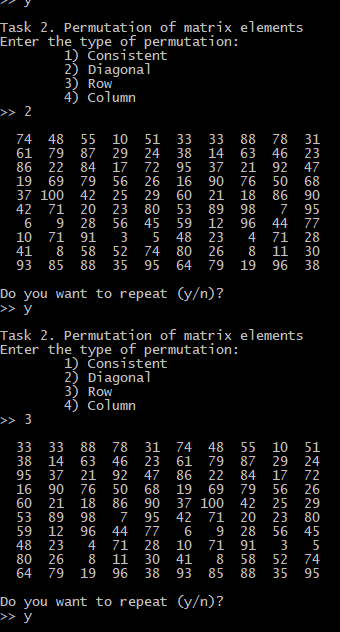
char  \* Words[5]  = { "Слово1", "Слово2", "Слово3", "Слово4", "Слово5" }

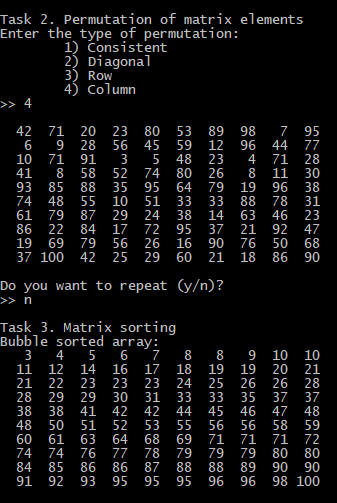
Как это работает: когда компилятор встречает в программе некоторый текст, заключенный в кавычки, в памяти создается символьный массив соответствующей этому тексту длины и адрес этого символьного массива присваивается соответствующему элементу – указателю массива **Words**.

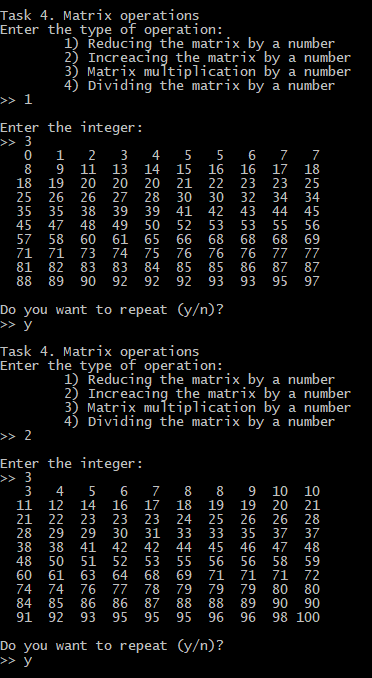
**Экспериментальные результаты.**

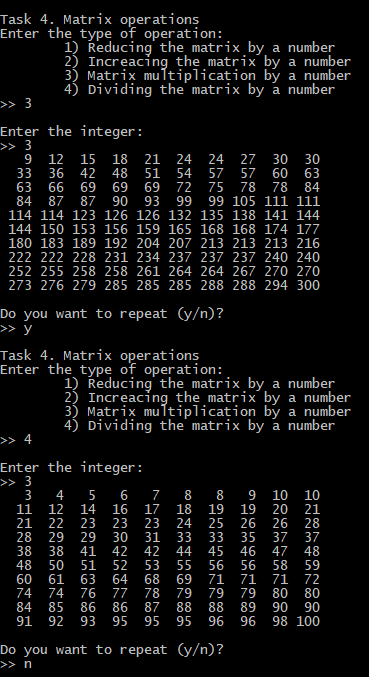


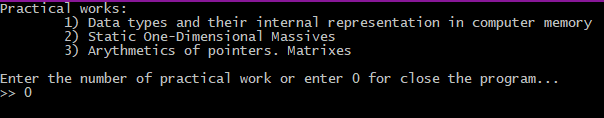










  
Рисунок 5 – Результат работы программы (полный код программы представлен в приложении А)

Приложение а  
полный код программы

#include <iostream>

#include <ctime> // Нужно для рандома

#include <chrono> // Нужно для засечения времени

#include <Windows.h> // Нужно для получения координат консоли и их изменения

#include <iomanip> // нужно для std::setw

using namespace std::chrono; // Нужно для засечения времени

using namespace std;

void practicalWork1();

void practicalWork2();

void practicalWork3();

void intToBinary(int);

void floatToBinary(int);

void doubleToBinary(int, int);

int readInt();

void createAndPrintRandomArray(int[]);

void bubbleSort(int[]);

void shakerSort(int[]);

void combSort(int[]);

void insertSort(int[]);

void quickSort(int[], int, int);

void Merge(int[], int, int, int, int[]);

void InternalMergeSort(int[], int, int, int[]);

void MergeSort(int[], int, int);

int getMinArrayElement(int[]);

int getMaxArrayElement(int[]);

int binarySearch(int[], int, int, int);

void copyArray(int[], int[]);

void printArray(int[]);

bool choiseNextAction();

float stopSecondsTimer(time\_point<steady\_clock>);

long long stopNanoSecondsTimer(time\_point<steady\_clock>);

void snakeAnimation(int[], int, short cellSize = 4, short delay = 100);

void spiralAnimation(int[], int, short cellSize = 4, short delay = 100);

void consistentPermutation(int[], int);

void diagonalPermutation(int[], int);

void rowPermutation(int[], int);

void columnPermutation(int[], int);

int main()

{

while (true) {

system("CLS");

cout << "Practical works: \n"

<< "\t1) Data types and their internal representation in computer memory \n"

<< "\t2) Static One-Dimensional Massives \n"

<< "\t3) Arythmetics of pointers. Matrixes \n\n"

<< "Enter the number of practical work or enter 0 for close the program... \n>> ";

int input;

cin >> input;

switch (input) {

case 1:

system("CLS");

practicalWork1();

break;

case 2:

system("CLS");

practicalWork2();

break;

case 3:

system("CLS");

practicalWork3();

break;

default:

goto Exit;

}

}

Exit:

cout << "\nClosing the program... \n";

system("pause");

return 0;

}

void practicalWork1() {

cout << "Solution of task \"Data types and their internal representation in computer memory\". \n\n";

do {

cout << "\tTask 1. Size of data types. \n"

<< "int: \t\t" << sizeof(int) << " bytes. \n"

<< "short int \t" << sizeof(short int) << " bytes. \n"

<< "long int \t" << sizeof(long int) << " bytes. \n"

<< "float \t\t" << sizeof(float) << " bytes. \n"

<< "double \t\t" << sizeof(double) << " bytes. \n"

<< "long double \t" << sizeof(long double) << " bytes. \n"

<< "char \t\t" << sizeof(char) << " bytes. \n"

<< "bool \t\t" << sizeof(bool) << " bytes. \n";

} while (choiseNextAction());

system("CLS");

cout << "\tTask 2. Binary representation of integer number. \n\n";

int input;

do {

cout << "Enter the integer number... \n>> ";

// Считывание числа с проверкой на принадлежность к целым числам.

input = readInt();

cout << "Binary representation of number " << input << ": ";

intToBinary(int(input));

} while (choiseNextAction());

system("CLS");

cout << "\tTask 3. Binary representation of float number. \n";

union {

float inputFloat;

int inputInt1;

};

do {

cout << "Enter the float number... \n>> ";

cin >> inputFloat;

cout << "Binary representation of number " << inputFloat << ": ";

floatToBinary(inputInt1);

} while (choiseNextAction());

system("CLS");

cout << "\tTask 4. Binary representation of double number. \n";

union {

double inputDouble;

int inputInt2[2] = { NULL, NULL };

};

do {

cout << "Enter the double number... \n>> ";

cin >> inputDouble;

cout << "Binary representation of number " << inputDouble << ":\n";

doubleToBinary(inputInt2[1], inputInt2[0]);

} while (choiseNextAction());

}

void intToBinary(int value) {

int mask = 1 << 31; // Маска представляет собой 2^32, то есть 10...00, где нулей 31.

for (int i = 1; i <= 32; i++) {

/\*

Побитовое сравнение введённого числа с маской.

Если у числа и маски есть хотя бы одна общая единица (в данном случае бОльший бит числа), то выводится "1", иначе "0".

Далее побитовый сдвиг введённого числа влево и так 32 раза (именно столько бит определяется под int).

Например:

0 1000000 00000000 00000000 00000000 - двоичное представление в памяти компьютера введённого числа 2^31

1 0000000 00000000 00000000 00000000 - маска (2^32)

Вывод: 0

Затем побитовый сдвиг введённого числа влево:

1 0000000 00000000 00000000 00000000

1 0000000 00000000 00000000 00000000 - маска (2^32)

Вывод: 1

Далее следуют нули, в выводе: 0 1000000 00000000 00000000 00000000

\*/

putchar(value & mask ? '1' : '0');

value <<= 1;

if (i % 8 == 0 || i == 1)

putchar(' '); // отделение друг от друга байтов, содержащих значение числа, а также знакового бита

}

}

void floatToBinary(int value) {

int mask = 1 << 31;

for (int i = 1; i <= 32; i++) {

putchar(value & mask ? '1' : '0');

value <<= 1;

if (i == 1 || i == 9)

putchar(' '); // отделение друг от друга битов мантиссы и порядка, а также знакового бита

}

}

void doubleToBinary(int firstPart, int secondPart) {

int mask = 1 << 31;

for (int i = 1; i <= 32; i++) {

putchar(firstPart & mask ? '1' : '0');

firstPart <<= 1;

if (i == 1 || i == 12)

putchar(' '); // отделение друг от друга битов мантиссы и порядка, а также знакового бита

}

for (int i = 1; i <= 32; i++) {

putchar(secondPart & mask ? '1' : '0');

secondPart <<= 1;

}

}

int readInt() {

double input;

while (true) {

cin >> input;

// Проверка на целое число

if (static\_cast<int>(input) != input) {

cout << "Need enter the integer number! Try again...\n";

continue;

}

return int(input);

}

}

const int sizeOfArray = 100;

void practicalWork2() {

srand((unsigned)time(NULL));

time\_point<steady\_clock> startTimer = steady\_clock::now();

float stopTimeInSeconds = stopSecondsTimer(startTimer);

long long stopTimeInNanoSeconds = stopNanoSecondsTimer(startTimer);

int count = 0;

cout << "Solution of task \"Static One-Dimensional Massives\". \n\n";

// РЕШЕНИЕ ПЕРВОГО ЗАДАНИЯ

int mainArray[sizeOfArray];

do {

cout << "\tTask 1. Initial array:" << endl;

createAndPrintRandomArray(mainArray);

} while (choiseNextAction());

do { // РЕШЕНИЕ ВТОРОГО ЗАДАНИЯ

cout << "\n\tTask 2. Array sorting." << endl

<< "\t\t1) Bubble sort" << endl

<< "\t\t2) Shaker sort" << endl

<< "\t\t3) Comb sort" << endl

<< "\t\t4) Insert sort" << endl

<< "\t\t5) Quick sort" << endl

<< "\t\t6) Merge sort" << endl

<< "Enter the number of sorting... \n>> ";

int input;

cin >> input;

int copiedArray[sizeOfArray];

copyArray(mainArray, copiedArray); // Копирование массива в другой массив для возможности в дальнейшем на том же массиве проверить другие типы сортировки.

switch (input) {

case 1: // РЕШЕНИЕ 2.1 ЗАДАНИЯ (bubble sort)

startTimer = steady\_clock::now(); // Старт отсчёта времени

bubbleSort(copiedArray);

stopTimeInSeconds = stopSecondsTimer(startTimer);

cout << "\nBubble sorted array: \n";

printArray(copiedArray);

cout << "Array sorted in " << fixed << stopTimeInSeconds << " second(s). \n";

break;

case 2: // РЕШЕНИЕ 2.2 ЗАДАНИЯ (shaker sort)

startTimer = steady\_clock::now();

shakerSort(copiedArray);

stopTimeInSeconds = stopSecondsTimer(startTimer);

cout << "\nShaker sorted array: \n";

printArray(copiedArray);

cout << "Array sorted in " << fixed << stopTimeInSeconds << " second(s). \n";

break;

case 3: // РЕШЕНИЕ 2.3 ЗАДАНИЯ (comb sort)

startTimer = steady\_clock::now();

combSort(copiedArray);

stopTimeInSeconds = stopSecondsTimer(startTimer);

cout << "\nComb sorted array: \n";

printArray(copiedArray);

cout << "Array sorted in " << fixed << stopTimeInSeconds << " second(s). \n";

break;

case 4: // РЕШЕНИЕ 2.4 ЗАДАНИЯ (insert sort)

startTimer = steady\_clock::now();

insertSort(copiedArray);

stopTimeInSeconds = stopSecondsTimer(startTimer);

cout << "\nInsert sorted array: \n";

printArray(copiedArray);

cout << "Array sorted in " << fixed << stopTimeInSeconds << " second(s). \n";

break;

case 5: // РЕШЕНИЕ 2.5 ЗАДАНИЯ (quick sort)

startTimer = steady\_clock::now();

quickSort(copiedArray, sizeOfArray - 1, 0);

stopTimeInSeconds = stopSecondsTimer(startTimer);

cout << "\nQuick sorted array: \n";

printArray(copiedArray);

cout << "Array sorted in " << fixed << stopTimeInSeconds << " second(s). \n";

break;

case 6: // РЕШЕНИЕ ДОПОЛНИТЕЛЬНОГО ЗАДАНИЯ (merge sort)

startTimer = steady\_clock::now();

MergeSort(copiedArray, 0, sizeOfArray - 1);

stopTimeInSeconds = stopSecondsTimer(startTimer);

cout << "\nMerge sorted array: \n";

printArray(copiedArray);

cout << "Array sorted in " << fixed << stopTimeInSeconds << " second(s). \n";

break;

}

} while (choiseNextAction());

int copiedArray[sizeOfArray]; // Создание отсортированного массива, с которым будет производиться работа в дальнейших заданиях

copyArray(mainArray, copiedArray);

quickSort(copiedArray, sizeOfArray - 1, 0);

do { // РЕШЕНИЕ ТРЕТЬЕГО ЗАДАНИЯ

cout << "\n\tTask 3. Searching min and max elements of sorted and unsorted arrays. \n";

startTimer = steady\_clock::now(); // Поиск в неотсортированном массиве

int max = getMaxArrayElement(mainArray);

int min = getMinArrayElement(mainArray);

stopTimeInNanoSeconds = stopNanoSecondsTimer(startTimer);

cout << "Min element of array: " << min << endl

<< "Max element of array: " << max << endl

<< "Elements founds in unsorted array for " << stopTimeInNanoSeconds << " nanoseconds. \n";

startTimer = steady\_clock::now(); // Поиск в отсортированном массиве

max = getMaxArrayElement(copiedArray);

min = getMinArrayElement(copiedArray);

stopTimeInNanoSeconds = stopNanoSecondsTimer(startTimer);

//cout << "Min: " << min << "\tMax: " << max << endl; // Отладка

cout << "Elements founds in sorted array for " << stopTimeInNanoSeconds << " nanoseconds. \n";

} while (choiseNextAction());

do { // РЕШЕНИЕ ЧЕТВЁРТОГО ЗАДАНИЯ

cout << "\n\tTask 4. Search average elements from array. \n";

int averageElement = int(abs(getMaxArrayElement(copiedArray)) - abs(getMinArrayElement(copiedArray)));

cout << "Average element: " << averageElement << endl;

bool isExists = false; // Проверка на то, существует ли вообще в массиве подходящий средний элемент

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

if (copiedArray[i] == averageElement) {

isExists = true;

break;

}

if (isExists) {

cout << "Index(es) of average elements: ";

int count = 0;

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

if (copiedArray[i] == averageElement) {

cout << i << " ";

count++;

}

cout << "\nCount of average elements in array: " << count << endl;

}

else

cout << "Element not found in array. \n";

} while (choiseNextAction());

do { // РЕШЕНИЕ ПЯТОГО ЗАДАНИЯ

cout << "\n\tTask 5. The number of smaller elements than the one entered by the user. \n"

<< "Enter the integer element...\n>> ";

int input;

cin >> input;

int count = 0;

for (int element : copiedArray)

if (element < input)

count++;

cout << "The array has " << count << " element(s) less than you entered. \n";

} while (choiseNextAction());

do { // РЕШЕНИЕ ШЕСТОГО ЗАДАНИЯ

cout << "\n\tTask 6. The number of larger elements than the one entered by the user. \n"

<< "Enter the integer element...\n>> ";

int input;

cin >> input;

int count = 0;

for (int element : copiedArray)

if (element > input)

count++;

cout << "The array has " << count << " element(s) larger than you entered. \n";

} while (choiseNextAction());

do { // РЕШЕНИЕ СЕДЬМОГО ЗАДАНИЯ

cout << "\n\tTask 7. The searching of a user-entered element in the array (traditional and binary search). \n"

<< "Enter the integer element...\n>> ";

int input;

cin >> input;

startTimer = steady\_clock::now();

bool isExists = false;

for (int element : copiedArray) // Поиск обычным перебором

if (element == input) {

isExists = true;

break;

}

stopTimeInNanoSeconds = stopNanoSecondsTimer(startTimer);

if (isExists)

cout << "The entered element is in the array. \n";

else

cout << "The entered element is not in the array. \n";

cout << "The search has been completed for " << stopTimeInNanoSeconds << " nanoseconds. \n";

startTimer = steady\_clock::now();

int searchedElement = binarySearch(copiedArray, input, 0, sizeOfArray - 1);

stopTimeInNanoSeconds = stopNanoSecondsTimer(startTimer);

/\* Отладка

if (searchedElement != -1)

cout << "Searched element (binary search): " << searchedElement << endl;

else cout << "Element not searched. \n"; \*/

cout << "The binary search has been completed for " << stopTimeInNanoSeconds << " nanoseconds. \n";

} while (choiseNextAction());

do { // РЕШЕНИЕ ВОСЬМОГО ЗАДАНИЯ

cout << "\n\tTask 8. Switching array elements. \n"

<< "Enter the first and second indexes of array elements...\n>> ";

int input1, input2;

cin >> input1 >> input2;

startTimer = steady\_clock::now();

//if (((input1 && input2) >= 0) && ((input1 && input2) < sizeOfArray)) { // Ошибка компилятора о небезопасном условии. Разобраться, почему.

if ((input1 >= 0) && (input2 >= 0) && (input1 < sizeOfArray) && (input2 < sizeOfArray)) {

int t = copiedArray[input1];

copiedArray[input1] = copiedArray[input2];

copiedArray[input2] = t;

}

else {

cout << "Wrong input! Try again...\n\n>> ";

continue;

}

stopTimeInNanoSeconds = stopNanoSecondsTimer(startTimer);

printArray(copiedArray); // Отладка

cout << "The switching has been completed for " << stopTimeInNanoSeconds << " nanoseconds. \n";

} while (choiseNextAction());

}

void createAndPrintRandomArray(int inputArray[]) {

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

inputArray[i] = -99 + rand() % 199; // В массив пишутся случайные целые числа диапазона -99..99 включительно. При желании данную функцию можно расширить возможностью выбирать граничные значения.

cout << inputArray[i] << "\t";

if ((i + 1) % 10 == 0) // Вывод по 10 элементов для удобочитаемости

cout << endl;

}

}

void bubbleSort(int inputArray[]) {

bool swapped;

do {

swapped = false;

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

if (inputArray[i] > inputArray[i + 1]) { // Соседние элементы массива меняются местами

int t = inputArray[i];

inputArray[i] = inputArray[i + 1];

inputArray[i + 1] = t;

swapped = true;

}

}

} while (swapped); // Если за последний проход цикла никакие элементы не поменялись местами, цикл прерывается.

}

void shakerSort(int inputArray[]) {

bool swapped;

do {

swapped = false;

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

if (inputArray[i] > inputArray[i + 1]) {

int t = inputArray[i];

inputArray[i] = inputArray[i + 1];

inputArray[i + 1] = t;

swapped = true;

}

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

if (inputArray[i] < inputArray[i - 1]) {

int t = inputArray[i];

inputArray[i] = inputArray[i - 1];

inputArray[i - 1] = t;

swapped = true;

}

} while (swapped);

}

void combSort(int inputArray[]) {

int swap;

float k = 1.247F, sortingRange = sizeOfArray - 1;

while (sortingRange >= 1)

{

for (int i = 0; i + sortingRange < sizeOfArray; i++)

{

if (inputArray[i] > inputArray[int(i + sortingRange)])

{

swap = inputArray[int(i + sortingRange)];

inputArray[int(i + sortingRange)] = inputArray[i];

inputArray[i] = swap;

}

}

sortingRange /= k;

}

bool swapped;

do {

swapped = false;

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

if (inputArray[i] > inputArray[i + 1]) { // Соседние элементы массива меняются местами

int t = inputArray[i];

inputArray[i] = inputArray[i + 1];

inputArray[i + 1] = t;

swapped = true;

}

}

} while (swapped == true); // Если за последний проход цикла никакие элементы не поменялись местами, цикл прерывается.

}

void insertSort(int inputArray[]) {

int extractedElement, currentPosition;

for (int i = 1; i < sizeOfArray; i++)

{

extractedElement = inputArray[i];

currentPosition = i;

while ((currentPosition >= 1) && (extractedElement < inputArray[currentPosition - 1]))

{

inputArray[currentPosition] = inputArray[currentPosition - 1];

currentPosition--;

}

inputArray[currentPosition] = extractedElement;

}

}

void quickSort(int inputArray[], int end, int begin) {

int middleElement;

int f = begin;

int l = end;

middleElement = inputArray[(f + l) / 2]; // Определяется опорный элемент

while (f < l) { // Массив сортируется так, что по окончании цикла будет представлять собой две части: элементы меньше либо равные опорному (левая), элементы больше либо равные опорному (правая)

while (inputArray[f] < middleElement) f++;

while (inputArray[l] > middleElement) l--;

if (f <= l)

{

int swap = inputArray[f];

inputArray[f] = inputArray[l];

inputArray[l] = swap;

f++;

l--;

}

}

if (begin < l) quickSort(inputArray, l, begin); // Где l - правая граница левой части разделённого пополам массива

if (f < end) quickSort(inputArray, end, f); // Где f - левая граница правой части разделённого пополам массива

}

void Merge(int array[], int first, int middle, int last, int temp[])

{

int idx = first;

int begin1 = first, end1 = middle;

int begin2 = middle + 1, end2 = last;

for (; begin1 <= end1 && begin2 <= end2; )

{

if (array[begin1] < array[begin2])

temp[idx++] = array[begin1++];

else

temp[idx++] = array[begin2++];

}

for (; begin1 <= end1; )

temp[idx++] = array[begin1++];

for (; begin2 <= end2; )

temp[idx++] = array[begin2++];

for (idx = first; idx <= last; idx++)

array[idx] = temp[idx];

}

void InternalMergeSort(int array[], int first, int last, int buffer[])

{

if (first < last)

{

int m = (first + last) / 2;

InternalMergeSort(array, first, m, buffer);

InternalMergeSort(array, m + 1, last, buffer);

Merge(array, first, m, last, buffer);

}

}

void MergeSort(int array[], int first, int last)

{

int \*buffer = new int[sizeOfArray];

InternalMergeSort(array, 0, sizeOfArray - 1, buffer);

delete[] buffer;

}

int getMinArrayElement(int inputArray[]) {

int minElement = INT\_MAX;

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

if (inputArray[i] < minElement)

minElement = inputArray[i];

return minElement;

}

int getMaxArrayElement(int inputArray[]) {

int maxElement = INT\_MIN;

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

if (inputArray[i] > maxElement)

maxElement = inputArray[i];

return maxElement;

}

int binarySearch(int inputArray[], int inputElement, int lowPos, int highPos) {

while (lowPos <= highPos) {

int mid = lowPos + (highPos - lowPos) / 2;

if (inputArray[mid] == inputElement)

return mid;

if (inputArray[mid] < inputElement)

lowPos = mid + 1;

else

highPos = mid - 1;

}

return -1;

}

void copyArray(int originalArray[], int resultArray[]) {

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

resultArray[i] = originalArray[i];

}

void printArray(int inputArray[]) {

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

cout << inputArray[i] << "\t";

if ((i + 1) % 10 == 0) // Вывод по 10 элементов для удобочитаемости

cout << endl;

}

cout << endl;

}

bool choiseNextAction() {

cout << "\nDo you want to repeat (y/n)?" << "\n>> ";

char input;

while (true) {

cin >> input;

if (input == 'y' || input == 'Y')

return true;

else if (input == 'n' || input == 'N')

return false;

else

cout << "Wrong input! Try again...\n>> ";

}

}

float stopSecondsTimer(time\_point<steady\_clock> startTimer) {

using fseconds = duration<float>; // Определение кастомного интервала времени для отображение дробных секунд в таймере.

time\_point<steady\_clock> endTimer = steady\_clock::now(); // Остановка времени

fseconds sortingTime = duration\_cast<fseconds>(endTimer - startTimer); // Вычисление разницы между финальным и стартовым временем

return sortingTime.count();

}

long long stopNanoSecondsTimer(time\_point<steady\_clock> startTimer) {

time\_point<steady\_clock> endTimer = steady\_clock::now();

nanoseconds sortingTime = duration\_cast<nanoseconds>(endTimer - startTimer);

return sortingTime.count();

}

void practicalWork3() {

int order;

system("CLS");

cout << "Solution of task \"Arythmetics of pointers. Matrixes\". \n\n"

<< "Task 1. Visualisation of filling the matrix. \n"

<< "Enter the order of matrix... \n>> ";

bool inputIsCorrected = true;

do {

cin >> order;

if (order > 0)

inputIsCorrected = false;

else {

cout << "Wrong input! Try again...\n>> ";

}

} while (inputIsCorrected);

srand((unsigned)time(NULL));

int \*ptrarray = new int[order\*order]; // Объявление динамического одномерного массива размерности двумерного

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

for (int j = 0; j < order; j++) // В массив пишутся случайные числа от 1 до N\*N, где N - порядок матрицы.

\*(ptrarray + i \* order + j) = 1 + rand() % (order \* order);

do {

cout << "\nEnter the type of animation:"

<< "\n\t1) Snake animation"

<< "\n\t2) Spiral animation \n>> ";

inputIsCorrected = true;

do {

int input;

cin >> input;

switch (input) {

case 1:

snakeAnimation(ptrarray, order);

inputIsCorrected = false;

break;

case 2:

spiralAnimation(ptrarray, order);

inputIsCorrected = false;

break;

default:

cout << "Wrong input! Try again...\n\n>> ";

}

} while (inputIsCorrected);

} while (choiseNextAction());

int \*copiedArr = new int[order \* order]; // Копирование массива в другой массив для возможности повторно произвести перестановку элементов.

do {

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

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

\*(copiedArr + i \* order + j) = \*(ptrarray + i \* order + j);

cout << "\nTask 2. Permutation of matrix elements"

<< "\nEnter the type of permutation:"

<< "\n\t1) Consistent"

<< "\n\t2) Diagonal"

<< "\n\t3) Row"

<< "\n\t4) Column \n>> ";

inputIsCorrected = true;

do {

int input;

cin >> input;

switch (input) {

case 1:

consistentPermutation(copiedArr, order);

inputIsCorrected = false;

break;

case 2:

diagonalPermutation(copiedArr, order);

inputIsCorrected = false;

break;

case 3:

rowPermutation(copiedArr, order);

inputIsCorrected = false;

break;

case 4:

columnPermutation(copiedArr, order);

inputIsCorrected = false;

break;

default:

cout << "Wrong input! Try again...\n\n>> ";

}

} while (inputIsCorrected);

cout << endl; // Вывод получившегося массива.

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

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

cout << setw(4) << \*(copiedArr + i \* order + j);

cout << endl;

}

} while (choiseNextAction());

cout << "\nTask 3. Matrix sorting"

<< "\nBubble sorted array: \n";

bool swapped;

do {

swapped = false;

for (int i = 0; i < order \* order - 1; i++) {

if (\*(copiedArr + i) > \*(copiedArr + i + 1)) { // Соседние элементы массива меняются местами

int t = \*(copiedArr + i);

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

\*(copiedArr + i + 1) = t;

swapped = true;

}

}

} while (swapped);

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

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

cout << setw(4) << \*(copiedArr + i \* order + j);

cout << endl;

}

do {

cout << "\nTask 4. Matrix operations"

<< "\nEnter the type of operation:"

<< "\n\t1) Reducing the matrix by a number"

<< "\n\t2) Increacing the matrix by a number"

<< "\n\t3) Matrix multiplication by a number"

<< "\n\t4) Dividing the matrix by a number \n>> ";

/\*

Если бы данная программа использовалась для чего-то большего, чем написание курсовой, то её однозначно стоило бы доработать на предмет возможности работы с нецелыми числами, т. к. в случае деления чисел матрицы на введённое пользователем число очень часто могут получаться нецелые числа в ячейках матрицы, а в данном случае мы будем видеть искажённый результат.

\*/

inputIsCorrected = true;

int input;

do {

cin >> input;

if (input == 1 || input == 2 || input == 3 || input == 4)

inputIsCorrected = false;

else {

cout << "Wrong input! Try again...\n\n>> ";

break;

}

} while (inputIsCorrected);

int number;

cout << "\nEnter the integer: \n>> ";

cin >> number;

switch (input) {

case 1:

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

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

\*(copiedArr + i \* order + j) -= number;

cout << setw(4) << \*(copiedArr + i \* order + j);

}

cout << endl;

}

break;

case 2:

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

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

\*(copiedArr + i \* order + j) += number;

cout << setw(4) << \*(copiedArr + i \* order + j);

}

cout << endl;

}

break;

case 3:

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

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

\*(copiedArr + i \* order + j) \*= number;

cout << setw(4) << \*(copiedArr + i \* order + j);

}

cout << endl;

}

break;

case 4:

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

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

\*(copiedArr + i \* order + j) /= number;

cout << setw(4) << \*(copiedArr + i \* order + j);

}

cout << endl;

}

break;

}

} while (choiseNextAction());

}

void snakeAnimation(int arr[], int order, short cellSize, short delay) {

/\* Функция принимает на вход:

1) Указатель на первый элемент массива-матрицы (квадратной) (required)

2) Порядок матрицы (required)

3) Размер текстовой ячейки, определяемой под одно число матрицы (минимально это количество цифр в максимальном числе + 1) (опционально)

4) Скорость вывода элементов матрицы, в миллисекундах (опционально)

Функции других анимаций работают аналогично.

\*/

cout << "\nTry to \"snake\" animation: \n";

HANDLE hConsole = GetStdHandle(STD\_OUTPUT\_HANDLE); // Нужно для получения данных курсора консоли

CONSOLE\_SCREEN\_BUFFER\_INFO bi;

int column = 0;

while (column < order) {

for (int row = 0; row < order; row++) { // Вывод вниз

GetConsoleScreenBufferInfo(hConsole, &bi); // Получение координат курсора

bi.dwCursorPosition.Y += 1; // Вычисление новых координат курсора

bi.dwCursorPosition.X -= cellSize;

SetConsoleCursorPosition(hConsole, bi.dwCursorPosition); // Установка положения курсора

cout << setw(cellSize) << \*(arr + row \* order + column);

Sleep(delay); // Задержка

}

column++;

if (column < order) {

for (int row = order - 1; row >= 0; row--) { // Вывод вверх

cout << setw(cellSize) << \*(arr + row \* order + column);

Sleep(delay);

GetConsoleScreenBufferInfo(hConsole, &bi);

bi.dwCursorPosition.Y -= 1;

bi.dwCursorPosition.X -= cellSize;

SetConsoleCursorPosition(hConsole, bi.dwCursorPosition);

}

column++;

GetConsoleScreenBufferInfo(hConsole, &bi);

bi.dwCursorPosition.X += (cellSize \* 2);

SetConsoleCursorPosition(hConsole, bi.dwCursorPosition);

}

else break;

}

if (order % 2 == 0)

for (int i = 0; i < order + 1; i++)

cout << endl;

}

void spiralAnimation(int arr[], int order, short cellSize, short delay) {

cout << "\n\nTry to spiral animation: \n";

HANDLE hConsole = GetStdHandle(STD\_OUTPUT\_HANDLE); // Нужно для получения данных курсора консоли

CONSOLE\_SCREEN\_BUFFER\_INFO bi;

int length = order; // Длина текущего вывода

for (int i = 0; i < length; i++) { // Вывод вправо

cout << setw(cellSize) << \*(arr + (order - length) \* order + i);

Sleep(delay);

}

length--;

int upperRightBorder = order - 1;

int lowerLeftBorder = 0;

while (length > 0)

{

for (int i = 0, t = lowerLeftBorder + 1; i < length; i++, t++) { // Вывод вниз

GetConsoleScreenBufferInfo(hConsole, &bi);

bi.dwCursorPosition.Y += 1;

bi.dwCursorPosition.X -= (cellSize);

SetConsoleCursorPosition(hConsole, bi.dwCursorPosition);

cout << setw(cellSize) << \*(arr + t \* order + upperRightBorder);

Sleep(delay);

}

if (!(length > 0))

break;

for (int i = 0, t = upperRightBorder - 1; i < length; i++, t--) { // Вывод влево

GetConsoleScreenBufferInfo(hConsole, &bi);

bi.dwCursorPosition.X -= (cellSize \* 2);

SetConsoleCursorPosition(hConsole, bi.dwCursorPosition);

cout << setw(cellSize) << \*(arr + upperRightBorder \* order + t);

Sleep(delay);

}

length--;

upperRightBorder--;

if (!(length > 0))

break;

for (int i = 0, t = upperRightBorder; i < length; i++, t--) { // Вывод вверх

GetConsoleScreenBufferInfo(hConsole, &bi);

bi.dwCursorPosition.Y -= 1;

bi.dwCursorPosition.X -= (cellSize);

SetConsoleCursorPosition(hConsole, bi.dwCursorPosition);

cout << setw(cellSize) << \*(arr + t \* order + lowerLeftBorder);

Sleep(delay);

}

lowerLeftBorder++;

if (!(length > 0))

break;

for (int i = 0, t = lowerLeftBorder; i < length; i++, t++) { // Вывод вправо

cout << setw(cellSize) << \*(arr + lowerLeftBorder \* order + t);

Sleep(delay);

}

length--;

}

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

cout << endl;

}

void consistentPermutation(int arr[], int order) {

// Определение порядка "рабочей четверти" матрицы, элементы которой (четверти) далее будут перебираться (разное поведение для матриц чётного и нечётного порядка)

int halfOrder = ceil(static\_cast<float>(order) / 2);

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

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

// Определение индекса текущего опорного элемента, далее будет производиться работа над элементами с таким же индексом, только в других четвертях матрицы

int \*p = (arr + i \* order + j);

int t = \*p;

// \*p - элемент левой верхней четверти, "рабочей четверти" матрицы

\*p = \*(p + halfOrder \* order);

// \*(p + halfOrder \* order) - элемент левой нижней четверти матрицы

\*(p + halfOrder \* order) = \*(p + halfOrder \* order + halfOrder);

// \*(p + halfOrder \* order + halfOrder) - элемент правой нижней четверти матрицы

\*(p + halfOrder \* order + halfOrder) = \*(p + halfOrder);

// элемент правой верхней четверти матрицы

\*(p + halfOrder) = t;

}

}

}

void diagonalPermutation(int arr[], int order) {

int halfOrder = ceil(static\_cast<float>(order) / 2);

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

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

int \*p = (arr + i \* order + j);

int t = \*p;

\*p = \*(p + halfOrder \* order + halfOrder);

\*(p + halfOrder \* order + halfOrder) = t;

t = \*(p + halfOrder);

\*(p + halfOrder) = \*(p + halfOrder \* order);

\*(p + halfOrder \* order) = t;

}

}

}

void rowPermutation(int arr[], int order) {

int halfOrder = ceil(static\_cast<float>(order) / 2);

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

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

int \*p = (arr + i \* order + j);

int t = \*p;

\*p = \*(p + halfOrder \* order);

\*(p + halfOrder \* order) = t;

t = \*(p + halfOrder);

\*(p + halfOrder) = \*(p + halfOrder \* order + halfOrder);

\*(p + halfOrder \* order + halfOrder) = t;

}

}

}

void columnPermutation(int arr[], int order) {

int halfOrder = ceil(static\_cast<float>(order) / 2);

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

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

int \*p = (arr + i \* order + j);

int t = \*p;

\*p = \*(p + halfOrder);

\*(p + halfOrder) = t;

t = \*(p + halfOrder \* order);

\*(p + halfOrder \* order) = \*(p + halfOrder \* order + halfOrder);

\*(p + halfOrder \* order + halfOrder) = t;

}

}

}