

МИНОБРНАУКИ РОССИИ

федеральное государственное бюджетное образовательное учреждение высшего образования

«Санкт-Петербургский государственный технологический институт (технический университет)»

УГНС 09.00.00 Информатика и вычислительная техника

Направление подготовки 09.03.01 Информатика и вычислительная

техника

Направленность (профиль) Системы автоматизированного проектирования

Форма обучения очная

Факультет Информационных технологий и управления

Кафедра Систем автоматизированного проектирования и управления

Учебная дисциплина Программирование

Курс I Группа 404

Исполнитель: обучающийся

Отчёт по лабораторной работе № 3 Вариант № 22

группы 404 Рухлова Ксения Алексеевна

(дата, подпись)

Проверили: Корниенко Иван Григорьевич

(дата, подпись) Федин Алексей Константинович

Санкт-Петербург 2021

**Содержание**

[1 Постановка задачи 3](#_Toc72445045)

[2. Исходные данные 3](#_Toc72445046)

[3. Особые ситуации 3](#_Toc72445047)

[4. Математические методы и алгоритмы решения задач 3](#_Toc72445048)

[5. Форматы представления данных 5](#_Toc72445049)

[6. Структура программы 6](#_Toc72445050)

[7. Описание хода выполнения лабораторной работы 6](#_Toc72445051)

[8. Блок-схема алгоритма решения задачи 7](#_Toc72445052)

[9. Результаты работы программы 12](#_Toc72445053)

[10. Исходный текст программы 15](#_Toc72445054)

# 1 Постановка задачи

Необходимо составить программу для сортировки массива данных методами: пузырьковой, отбора, вставки, Шелла и быстрой сортировки. Вывести на экран неупорядоченный (один раз) и упорядоченные (для каждого из методов) массивы данных. Составить сравнительную таблицу эффективности методов, в которой необходимо указать число сравнений и перестановок переменных в каждом методе сортировки. Неупорядоченная матрица из N строк и M столбцов задается и заполняется один раз (с клавиатуры, из файла или случайными числами), далее она используется для каждого из методов сортировки. Реализовать абстрактный базовый класс *ISort*, содержащий метод *Sort* и необходимые счетчики, от которого наследовать подклассы для реализации сортировок.

Упорядочить каждый четный столбец по убыванию, каждую строку по убыванию.

# 2. Исходные данные

В качестве исходных данных программа использует количество столбцов и строк (размер двумерного массива) и сами элементы массива (числа). Исходные данные могут быть введены пользователем, случайно сгенерированы или введены из файла (сначала количество столбцов и строк, дальше элементы массива).

# 3. Особые ситуации

Необходимо рассмотреть следующие особые ситуации:

– Если пользователь ввел количество столбцов или строк меньше, или равно 0, пользователю предлагается ввести значение количества точек заново.

– Если пользователь выбрал ввод данных из файла, в котором записаны некорректные исходные данные, то их считывание невозможно, в таком случае пользователю предлагается ввести название файла и путь к нему заново.

– Если пользователь выбрал запись данных или результата в файл, который уже существует, то в таком случае пользователю предлагается либо ввести новое название файла и путь к нему, либо перезаписать данный файл.

# 4. Математические методы и алгоритмы решения задач

1. Пузырьковая сортировка. Алгоритм состоит из повторяющихся проходов по сортируемому массиву. За каждый проход элементы последовательно сравниваются попарно и, если порядок в паре неверный, выполняется обмен элементов. Проходы по массиву повторяются N раз или до тех пор, пока на очередном проходе не окажется, что обмены больше не нужны, что означает — массив отсортирован.

2. Сортировка вставками. Проходим по массиву слева направо и обрабатываем по очереди каждый элемент. Слева от очередного элемента наращиваем отсортированную часть массива, справа по мере процесса потихоньку испаряется неотсортированная. В отсортированной части массива ищется точка вставки для очередного элемента. Сам элемент отправляется в буфер, в результате чего в массиве появляется свободная ячейка — это позволяет сдвинуть элементы и освободить точку вставки.

3. Сортировка отбором. проходим по массиву в поисках максимального элемента. Найденный максимум меняем местами с первым элементом. Неотсортированная часть массива уменьшилась на один элемент (не включает первый элемент, куда мы переставили найденный максимум). К этой неотсортированной части применяем те же действия — находим максимум и ставим его на первое место в неотсортированной части массива. И так продолжаем до тех пор, пока неотсортированная часть массива не уменьшится до одного элемента.

4. Сортировка Шелла. Идея метода заключается в сравнение разделенных на группы элементов последовательности, находящихся друг от друга на некотором расстоянии. Изначально это расстояние равно *d* или *N/2*, где *N* — общее число элементов. На первом шаге каждая группа включает в себя два элемента расположенных друг от друга на расстоянии *N/2*; они сравниваются между собой, и, в случае необходимости, меняются местами. На последующих шагах также происходят проверка и обмен, но расстояние *d* сокращается на *d/2*, и количество групп, соответственно, уменьшается. Постепенно расстояние между элементами уменьшается, и на *d=1* проход по массиву происходит в последний раз.

5. Быстрая сортировка. Отличительной особенностью быстрой сортировки является операция разбиения массива на две части относительно опорного элемента. Например, если последовательность требуется упорядочить по возрастанию, то в левую часть будут помещены все элементы, значения которых меньше значения опорного элемента, а в правую элементы, чьи значения больше или равны опорному. Вне зависимости от того, какой элемент выбран в качестве опорного, массив будет отсортирован, но все же наиболее удачным считается ситуация, когда по обеим сторонам от опорного элемента оказывается примерно равное количество элементов. Если длина какой-то из получившихся в результате разбиения частей превышает один элемент, то для нее нужно рекурсивно выполнить упорядочивание, т. е. повторно запустить алгоритм на каждом из отрезков.

# 5. Форматы представления данных

Таблица 1 – Переменные, используемые в программе

|  |  |  |
| --- | --- | --- |
| **Имя** | **Тип** | **Описание** |
| input | int | Выбор варианта ввода данных |
| check | bool | Проверка корректности ввода |
| end | bool | Повтор программы |
| Path | string | Ввод имени файла и пути к нему |
| row | int | Обозначает количество строк |
| col | int | Обозначает количество столбцов |
| mode | int | Режим чтения/ записи файла |
| comparisons | int | Обозначает сравнения |
| permutations | int | Обозначает перестановки |
| output\_file | fstream | Чтение из файла |
| result\_file | ofstream | Запись в файл |

Также используются следующие константы

Таблица 2 – Константы, используемые в программе

|  |  |  |
| --- | --- | --- |
| **Имя** | **Тип** | **Описание** |
| MIN\_COL  MAX\_COL  MIN\_ROW  MAX\_ROW  MIN\_ELEMENT  MAX\_ELEMENT | const int | Определяет границы случайной генерации |

Для сортировки массива используется абстрактный класс *ISort*, у которого 5 дочерних класса: *Bubble\_sort, Selection\_sort, Insert\_sort, Shell\_sort, Quick\_sort*. Также используется класс для создания матрицы *Matrix* и для корректного чтения элементов матрицы из файла *Output\_file\_elements*.

# 6. Структура программы

В ходе выполнения работы было принято решение разбить программу на модули:

Таблица 3 – Функции, составляющие программу

|  |  |  |
| --- | --- | --- |
| **Имя модуля** | **Функции, входящие в состав модуля** | **Описание** |
| Main | main | Основной модуль |
| Interface | get\_double, get\_int, get\_point, size\_arr, output\_point, greetings,  input\_types,show\_or\_try, end\_program, output\_arr,res\_output\_arr, saving\_results,save\_data | Ввод и вывод исходных данных, подключение файла, общие функции |
| Array | - | Класс матрицы |
| Sorting | update\_dates | Класс сортировок |
| Test | Test1, Test2, Test3, Test4, Run\_all\_tests | Модульные тесты |

# 7. Описание хода выполнения лабораторной работы

1. Сначала для явного преобразования использовалась конвертация *C-style* (с помощью оператора ()), что могло привести к ошибке. Было принято решение заменить ее на оператор *static\_cast.*

2. Количество строчек и столбцов матрицы не могут быть меньше или равно нулю, поэтому каждый ввод данных (из файла или с клавиатуры) обрабатывается на это условие.

3. Программа после запуска выдавала одни и те же результаты, хотя в коде использовался вызов функции *rand*, возвращающей случайное число. После изучения справочной системы выяснилось, что необходимо использовать функцию *srand* для начальной инициализации генератора случайных чисел. После этого программа стала работать правильно.

4. Так как количество данных может быть в файле больше, чем просто элементы массива, было принято решение создать дополнительный класс массива, который хранит все данные, а потом по индексу присваивать нужные нам элементы.

5. В программе использовались глобальные переменные. Чтобы не было никаких ошибок, мы заменили их на локальные в качестве аргументов к функциям.

6. Количество перестановок и сравнений с повтором программы накапливалось, поэтому мы создали отдельную функцию *update\_dates* для их обнуления.

# 8. Блок-схема алгоритма решения задачи

На рисунке ниже предоставлена блок-схема алгоритма решения задачи.

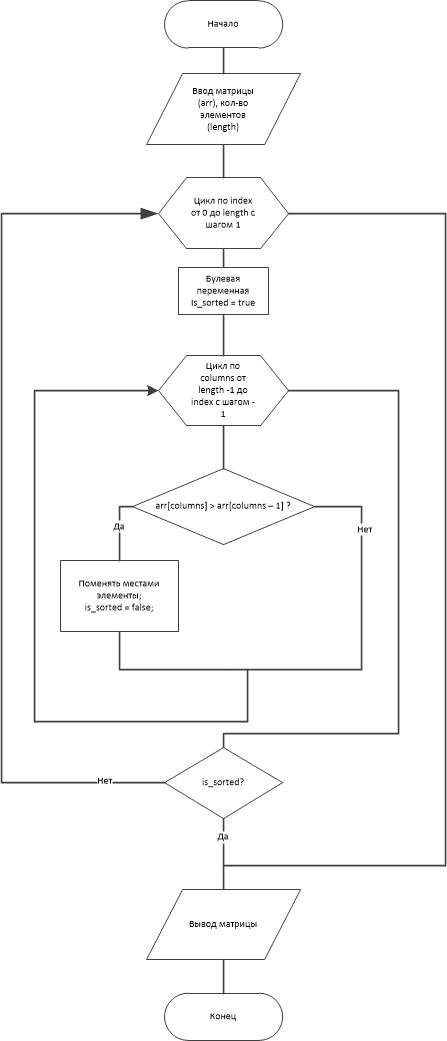


Рисунок 1 – Блок-схема алгоритма сортировки пузырьком

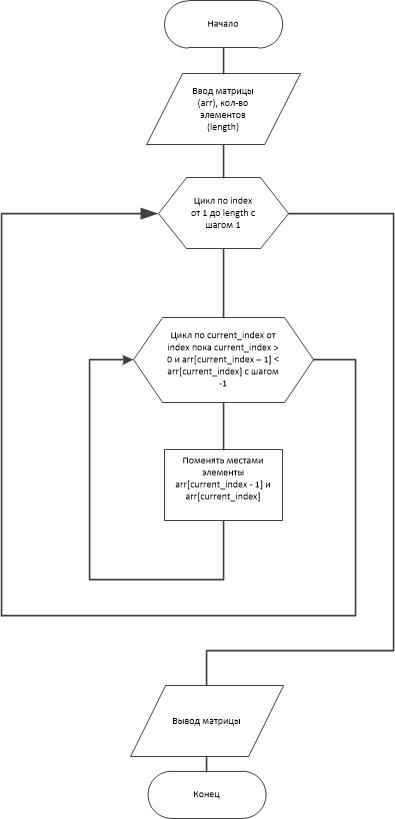


Рисунок 2 – Блок-схема алгоритма сортировки вставкой

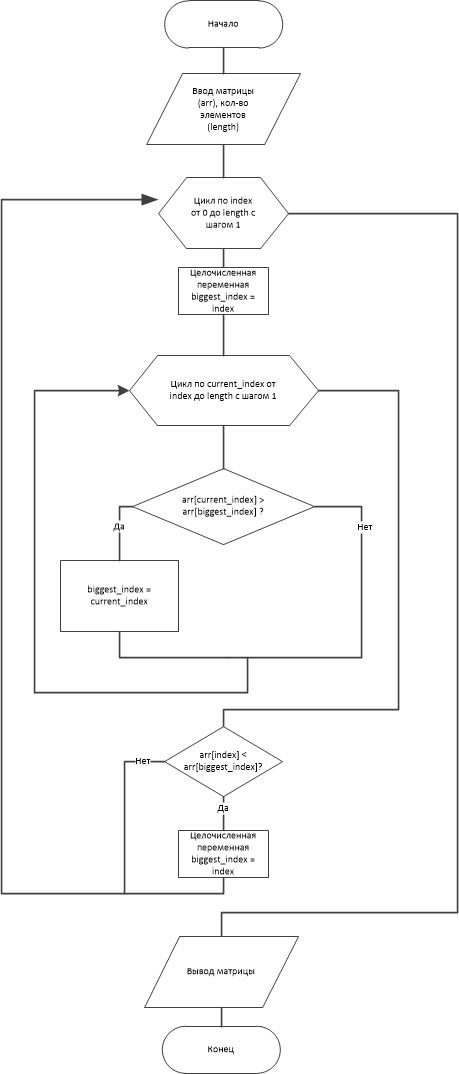


Рисунок 3 – Блок-схема алгоритма сортировки отбором

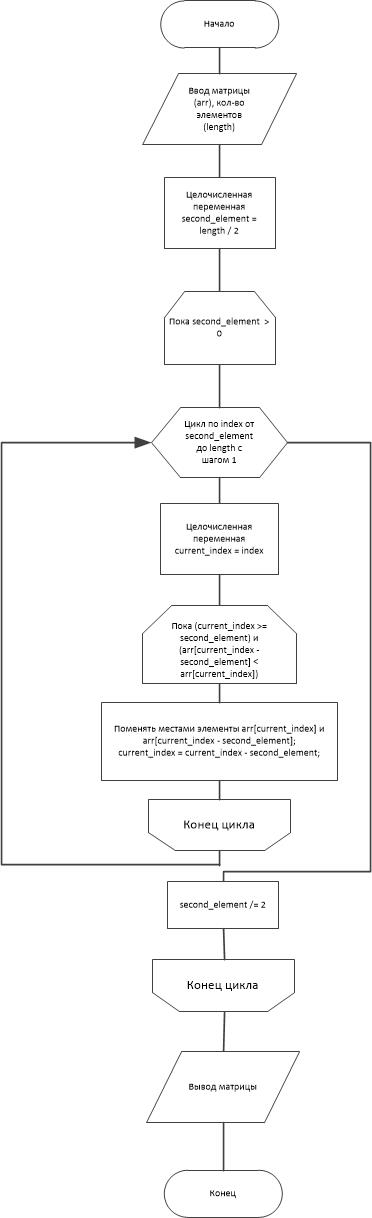
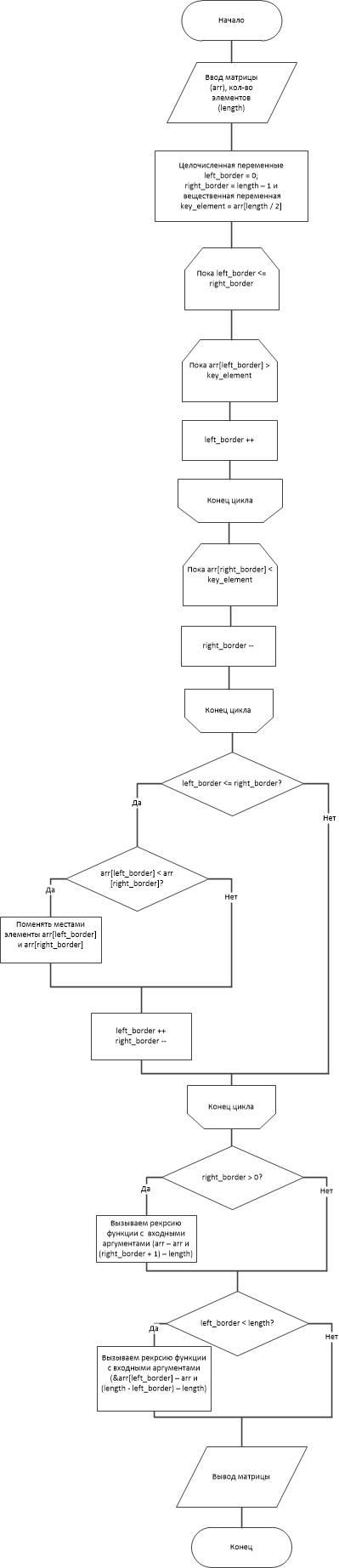


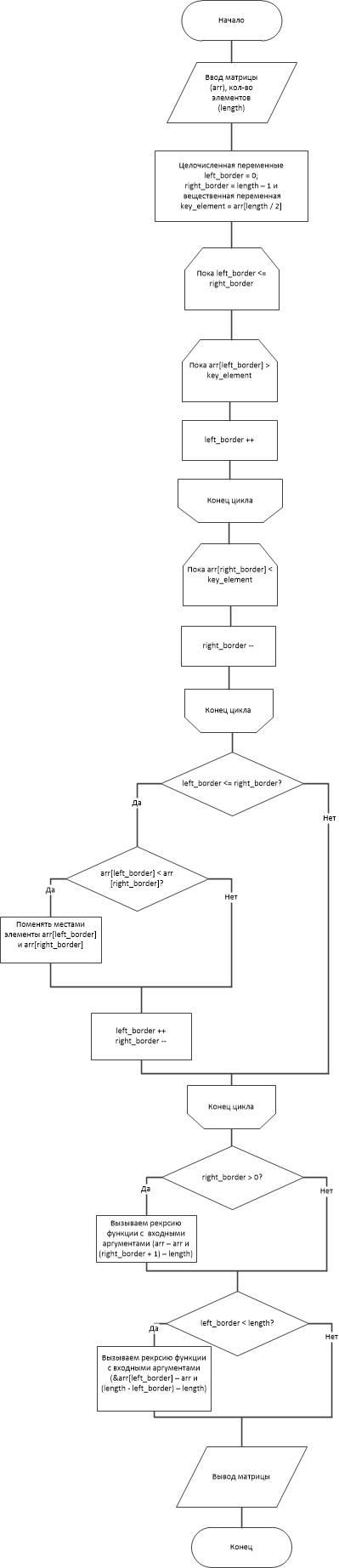
Рисунок 4 – Блок-схема алгоритма сортировки Шелла



1

Рисунок 5 – Блок-схема алгоритма быстрой сортировки

(часть 1)



1

Рисунок 6 – Блок-схема алгоритма быстрой сортировки

(часть 2)

# 9. Результаты работы программы

В результате вычислений программа выводит отсортированную матрицу и таблицу сравнений и перестановок видов сортировок. Ниже, на рисунках 7 - 11 приведены примеры результатов работы программы:

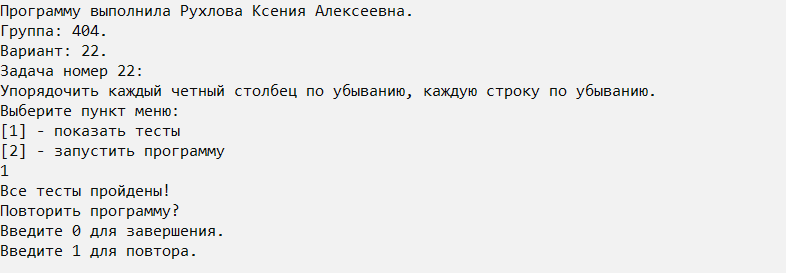


Рисунок 7 – Экранная копия результата работы разработанной программы

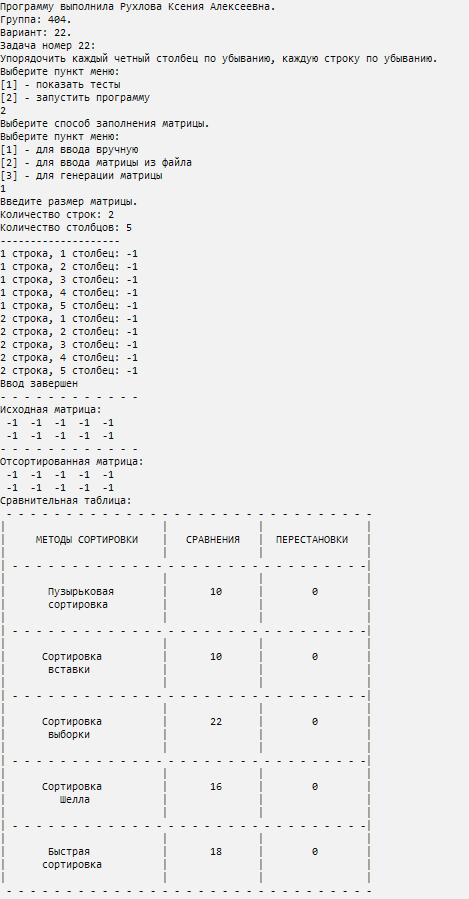


Рисунок 8 – Экранная копия результата работы разработанной программы

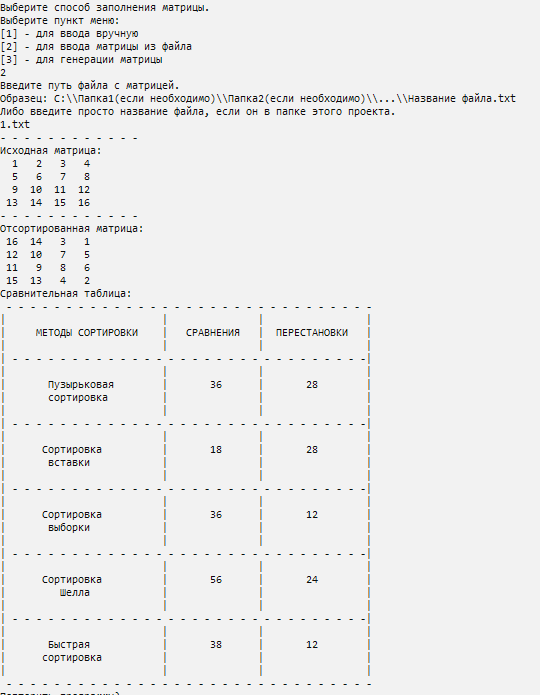


Рисунок 9 – Экранная копия результата работы разработанной программы

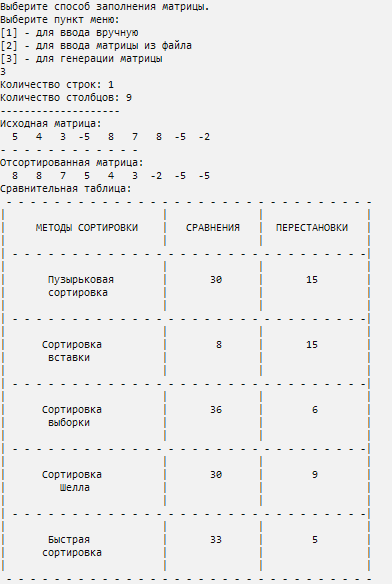


Рисунок 10 – Экранная копия результата работы, разработанной программ

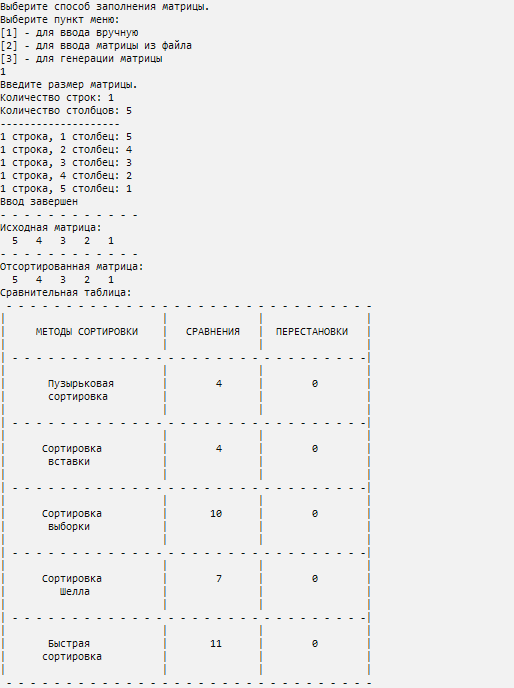


Рисунок 11 – Экранная копия результата работы, разработанной программ

# 10. Исходный текст программы

[Начало программы ---]

// Лабораторная работа № 3.

// Использование языка С++ для изучения методов сортировок

// Студентка группы 404, Рухлова Ксения Алексеевна. 2021 год

[Начало модуля main ---]

//Main.h

#pragma once

#include <iostream>

#include <ctime>

#include <string>

#include <Windows.h>

//Main.cpp

#include "Main.h"

#include "Array.h"

#include "Interface.h"

#include "sorting.h"

#include "Test.h"

using namespace std;

int main()

{

setlocale(LC\_ALL, "Russian");

greetings();

bool end = false;

int col = 0, mode = 0, row = 0;

Matrix arr, t\_arr;

Bubble\_sort bubble;

Selection\_sort selection;

Insert\_sort insert;

Shell\_sort shell;

Quick\_sort quick;

srand(static\_cast<unsigned char>(time(NULL)));

do {

if (test\_or\_try()) run\_tests();

else {

for (int index = 0; index < 5; index++) {

}

input\_types(arr, t\_arr, row, col, mode);

bubble.sort(arr, t\_arr, arr.get\_row(), arr.get\_col());

insert.sort(arr, t\_arr, arr.get\_row(), arr.get\_col());

selection.sort(arr, t\_arr, arr.get\_row(), arr.get\_col());

shell.sort(arr, t\_arr, arr.get\_row(), arr.get\_col());

quick.sort(arr, t\_arr, arr.get\_row(), arr.get\_col());

results(t\_arr, bubble, selection, insert, shell, quick);

save\_file(arr, t\_arr, bubble, selection, insert, shell, quick);

}

end = end\_program();

update\_dates(bubble, insert, selection, shell, quick);

} while (end != true);

}

[--- Конец модуля Main]

[Начало модуля Interface ---]

//Interface.h

#pragma once

#include <iostream>

#include <fstream>

#include <string>

#include <filesystem>

#include "Main.h"

#include "Array.h"

class Output\_file\_elements {

private:

double\* arr\_elements;

int arr\_elements\_size;

public:

double& operator[] (const int index) {

return arr\_elements[index];

}

int get\_size() {

return arr\_elements\_size;

}

void set\_size(int size) {

arr\_elements\_size = size;

arr\_elements = new double[arr\_elements\_size];

}

~Output\_file\_elements() {

delete[]arr\_elements;

}

};

class Bubble\_sort;

class Selection\_sort;

class Insert\_sort;

class Shell\_sort;

class Quick\_sort;

enum Actions {

END = 0, CONTINUE,

MANUAL = 1, FILE\_INPUT, GENERATION,

YES = 1, NO,

TEST = 1, TRY

};

int get\_int();

double get\_double();

void greetings();

bool end\_program();

void result\_file\_name(std:: string& name);

bool test\_or\_try(void);

void input\_types(Matrix& arr, Matrix& t\_arr, int& col, int& row, int& mode);

void output\_arr(Matrix& arr);

void results(Matrix& arr,Bubble\_sort& bubble, Selection\_sort& selection, Insert\_sort& insert, Shell\_sort& shell, Quick\_sort& quick);

void generation\_input(Matrix& arr, Matrix& t\_arr, int& col, int& row);

//Interface.cpp

#include "Interface.h"

#include "Array.h"

#include "sorting.h"

using namespace std;

using namespace filesystem;

double get\_double() {

double input = 0;

cin >> input;

while (cin.fail()) {

cin.clear();

cin.ignore(INT\_MAX, '\n');

cout << "Введите число." << endl;

cin >> input;

}

return input;

}

int get\_int() {

int input = 0;

cin >> input;

while (cin.fail()) {

cin.clear();

cin.ignore(INT\_MAX, '\n');

cout << "Введите число." << endl;

cin >> input;

}

return input;

}

void input\_file(Matrix& arr, Matrix& t\_arr, int& col, int& row) {

bool repeat = true;

do {

repeat = false;

cout << "Введите путь файла с матрицей." << endl

<< "Образец: C:\\\\Папка1(если необходимо)\\\\Папка2(если необходимо)\\\\...\\\\Название файла.txt " << endl

<< "Либо введите просто название файла, если он в папке этого проекта."

<< endl;

string name;

while (true) {

cin >> name;

ofstream output\_file\_text(name, ofstream::app);

error\_code ec;

if (!is\_regular\_file(name, ec)) {

cout << "Адрес содержит недопустимые значения. Повторите ввод." << endl;

ec.clear();

}

else break;

output\_file\_text.close();

}

ifstream output\_file\_matrix(name);

Output\_file\_elements file\_elements;

if (!(repeat)) {

output\_file\_matrix >> col >> row;

int AMOUNT\_DATA = abs(col \* row);

file\_elements.set\_size(AMOUNT\_DATA);

for (int index = 0; index < AMOUNT\_DATA; index++) {

if (!(output\_file\_matrix >> file\_elements[index])) {

repeat = true;

}

}

if (repeat) cout << "Некорректные данные." << endl;

else {

if (row <= 0) {

cout << "Количество строк должно быть больше нуля." << endl;

repeat = true;

}

if (col <= 0) {

cout << "Количество столбцов должно быть больше нуля." << endl;

repeat = true;

}

if (!(repeat)) {

arr.set\_size(col, row);

t\_arr.set\_size(col, row);

int index = 0;

for (int rows = 0; rows < row; rows++) {

for (int columns = 0; columns < col; columns++) {

arr(rows, columns) = file\_elements[index];

index++;

}

}

}

}

}

} while (repeat);

}

void user\_input(Matrix& arr, Matrix& t\_arr, int& col, int& row) {

cout << "Введите размер матрицы." << endl

<< "Количество строк: ";

while (true) {

row = get\_int();

if (row <= 0) cout << "Количество строк должно быть больше нуля." << endl;

else break;

}

cout << "Количество столбцов: ";

while (true) {

col = get\_int();

if (col <= 0) cout << "Количество столбцов должно быть больше нуля." << endl;

else break;

}

cout << "--------------------" << endl;

arr.set\_size(col, row);

t\_arr.set\_size(col, row);

for (int rows = 0; rows < row; rows++) {

for (int columns = 0; columns < col; columns++) {

cout << rows + 1 << " строка, " << columns + 1 << " столбец: ";

arr(rows,columns) = get\_double();

}

}

cout << "Ввoд завершен" << endl;

}

void generation\_input(Matrix& arr, Matrix& t\_arr, int& col, int& row) {

const int MIN\_COL = 1;

const int MAX\_COL = 10;

const int MIN\_ROW = 1;

const int MAX\_ROW = 10;

const int MIN\_ELEMENT = -5;

const int MAX\_ELEMENT = 15;

col = rand() % MAX\_COL + MIN\_COL;

row = rand() % MAX\_ROW + MIN\_ROW;

cout << "Количество строк: " << row << endl;

cout << "Количество столбцов: " << col << endl;

cout << "--------------------" << endl;

arr.set\_size(col, row);

t\_arr.set\_size(col, row);

for (int rows = 0; rows < row; rows++) {

for (int columns = 0; columns < col; columns++) {

arr(rows, columns) = rand() % MAX\_ELEMENT + MIN\_ELEMENT;

}

}

}

void output\_arr(Matrix& arr) {

for (int rows = 0; rows < arr.get\_row(); rows++) {

for (int columns = 0; columns < arr.get\_col(); columns++) {

cout << setw(3)

<< arr(rows, columns) << " ";

}

cout << endl;

}

}

void results(Matrix& arr, Bubble\_sort& bubble, Selection\_sort& selection, Insert\_sort& insert, Shell\_sort& shell, Quick\_sort& quick) {

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

output\_arr(arr);

cout << "--------------------" << endl;

cout << "Сравнительная таблица: " << endl;

cout << " - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - \n";

cout << "| | | |" << endl;

cout << "| МЕТОДЫ СОРТИРОВКИ | СРАВНЕНИЯ | ПЕРЕСТАНОВКИ |" << endl;

cout << "| | | |" << endl;

cout << "| - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -|" << endl;

cout << "| | | |" << endl;

cout << "| Пузырьковая |" << setw(9) << bubble.get\_comparisons() << " |" << setw(9) << bubble.get\_permutations() << " |" << endl;

cout << "| сортировка | | |" << endl;

cout << "| | | |" << endl;

cout << "| - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -|" << endl;

cout << "| | | |" << endl;

cout << "| Сортировка |" << setw(9) << insert.get\_comparisons() << " |" << setw(9) << insert.get\_permutations() << " |" << endl;

cout << "| вставки | | |" << endl;

cout << "| | | |" << endl;

cout << "| - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -|" << endl;

cout << "| | | |" << endl;

cout << "| Сортировка |" << setw(9) << selection.get\_comparisons() << " |" << setw(9) << selection.get\_permutations() << " |" << endl;

cout << "| выборки | | |" << endl;

cout << "| | | |" << endl;

cout << "| - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -|" << endl;

cout << "| | | |" << endl;

cout << "| Сортировка |" << setw(9) << shell.get\_comparisons() << " |" << setw(9) << shell.get\_permutations() << " |" << endl;

cout << "| Шелла | | |" << endl;

cout << "| | | |" << endl;

cout << "| - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -|" << endl;

cout << "| | | |" << endl;

cout << "| Быстрая |" << setw(9) << quick.get\_comparisons() << " |" << setw(9) << quick.get\_permutations() << " |" << endl;

cout << "| сортировка | | |" << endl;

cout << "| | | |" << endl;

cout << " - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -\n";

}

void input\_types(Matrix& arr, Matrix& t\_arr, int& col, int& row, int& mode) {

cout << "Выберите способ заполнения матрицы." << endl

<< "Выберите пункт меню: " << endl << "[1] - для ввода вручную" << endl

<< "[2] - для ввода матрицы из файла" << endl << "[3] - для генерации матрицы" << endl;

while (true) {

mode = get\_int();

if (mode == MANUAL) {

user\_input(arr, t\_arr, col, row);

cout << "- - - - - - - - - - - -" << endl;

break;

}

if (mode == FILE\_INPUT) {

input\_file(arr, t\_arr, col, row);

cout << "- - - - - - - - - - - -" << endl;

break;

}

if (mode == GENERATION) {

generation\_input(arr, t\_arr, col, row);

break;

}

else cout << "Введите либо 1, либо 2, либо 3." << endl;

}

cout << "Исходная матрица:" << endl;

output\_arr(arr);

cout << "- - - - - - - - - - - -" << endl;

}

bool test\_or\_try(void) {

cout << "Выберите пункт меню:" << endl;

cout << "[1] - показать тесты" << endl;

cout << "[2] - запустить программу" << endl;

int mode = 0;

while (!((mode == TEST) || (mode == TRY))) {

mode = get\_int();

switch (mode) {

case TEST: return true;

break;

case TRY: return false;

break;

default: cout << "Введите либо 1, либо 2." << endl;

}

}

return true;

}

void save\_file(Matrix& arr, Matrix& arr\_t, Bubble\_sort& bubble, Selection\_sort& selection, Insert\_sort& insert, Shell\_sort& shell, Quick\_sort& quick) {

cout << "Сохранить результаты?" << endl

<< "Выберите пункт меню:" << endl;

cout << "[1] - сохранить" << endl << "[2] - пропустить" << endl;

bool save\_text = false;

while (true) {

int save = get\_int();

if (save == YES) {

save\_text = true;

break;

}

else if (save == NO) {

cout << "Вы выбрали пропустить сохранение." << endl;

cout << "- - - - - - - - - -" << endl;

break;

}

else cout << "Введите либо 1, либо 2." << endl;

}

string name = "";

if (save\_text) {

cout << "Куда сохранить результаты?" << endl;

result\_file\_name(name);

ofstream result\_file(name);

result\_file.clear();

result\_file << "Исходная матрица: " << endl;

for (int rows = 0; rows < arr.get\_row(); rows++) {

for (int columns = 0; columns < arr.get\_col(); columns++) {

result\_file << arr(rows, columns) << " ";

}

result\_file << endl;

}

result\_file << "Отсортированная матрица: " << endl;

for (int rows = 0; rows < arr\_t.get\_row(); rows++) {

for (int columns = 0; columns < arr\_t.get\_col(); columns++) {

result\_file << arr\_t(rows, columns) << " ";

}

result\_file << endl;

}

result\_file << "--------------------" << endl;

result\_file << "Сравнения и перестановки: " << endl;

result\_file << "Пузырьковая сортировка: " << bubble.get\_comparisons() << " " << bubble.get\_permutations() << " " << endl;

result\_file << "Сортировка вставки: " << insert.get\_comparisons() << " " << insert.get\_permutations() << " " << endl;

result\_file << "Сортировка выборки: " << selection.get\_comparisons() << " " << selection.get\_permutations() << " " << endl;

result\_file << "Сортировка Шелла: " << shell.get\_comparisons() << " " << shell.get\_permutations() << " " << endl;

result\_file << "Быстрая сортировка: " << quick.get\_comparisons() << " " << quick.get\_permutations() << " " << endl;

cout << "Данные сохранены." << endl;

cout << "- - - - - - - - - -" << endl;

}

}

void result\_file\_name(string& name) {

bool repeat = true;

do {

repeat = false;

cout << "Введите путь файла, в который запишутся данные." << endl << "Образец: C:\\\\Папка1(если необходимо)\\\\"

"Папка2(если необходимо)\\\\...\\\\Название файла.txt " << endl << "Либо введите просто название файла, тогда он будет в папке этого проекта" << endl;

cin >> name;

if (ifstream(name)) {

cout << "Файл уже существует." << endl;

cout << "Выберите пункт меню:" << endl;

cout << "[0] - перезаписать существующий файл" << endl << "[1] - повторить ввод" << endl;

while (true) {

int rewrite = get\_int();

if (rewrite == 0) {

cout << "Вы выбрали опцию перезаписать файл." << endl;

break;

}

if (rewrite == 1) {

cout << "Вы выбрали опцию повторить ввод." << endl;

repeat = true;

break;

}

else

cout << "Введите 1, либо 2";

}

}

if (!(repeat)) {

ofstream result\_file(name, ofstream::app);

error\_code ec;

if (!is\_regular\_file(name, ec)) {

cout << "Адрес содержит недопустимые значения. Повторите ввод." << endl;

ec.clear();

repeat = true;

}

result\_file.close();

}

} while (repeat);

}

void greetings() {

cout << "Программу выполнила Рухлова Ксения Алексеевна." << endl << "Группа: 404." << endl << "Вариант: 22." << endl <<

"Задача номер 22:" << endl <<

"Упорядочить каждый четный столбец по убыванию, каждую строку по убыванию." << endl;

}

bool end\_program() {

cout << "Повторить программу? " << endl;

cout << "Выберите пункт меню:" << endl

<< "[1] - завершить программу" << endl << "[2] - повторить программу" << endl;

while (true) {

int end = get\_int();

switch (end) {

case END:

cout << "Программа завершается..." << endl;

return true;

break;

case CONTINUE:

cout << "Вы выбрали повторить программу: " << endl;

return false;

break;

default:

cout << "Введите либо 1, либо 2." << endl;

}

}

}

[--- Конец модуля Interface]

[ Начало модуля Sorting ---]

// Sorting.h

#pragma once

#include "Array.h"

#include "Main.h"

#include "Interface.h"

class Isort {

protected:

int c\_comparisons = 0;

int c\_permutations = 0;

virtual void sorting(double arr[], int length, int& comparisons, int& permutations) = 0;

public:

void sort(Matrix& arr, Matrix& t\_arr, int row, int col);

int get\_comparisons(){

return c\_comparisons;

}

int get\_permutations() {

return c\_permutations;

}

void update\_dates() {

c\_comparisons = 0;

c\_permutations = 0;

}

};

class Bubble\_sort final: public Isort {

void sorting(double arr[], int length, int& comparisons, int& permutations);

};

class Selection\_sort final: public Isort {

void sorting(double arr[], int length, int& comparisons, int& permutations);

};

class Insert\_sort final: public Isort {

void sorting(double arr[], int length, int& comparisons, int& permutations);

};

class Shell\_sort final: public Isort {

void sorting(double arr[], int length, int& comparisons, int& permutations);

};

class Quick\_sort final: public Isort {

void sorting(double arr[], int length, int& comparisons, int& permutations);

};

enum {

BUBBLE, SELECTION, INSERT, SHELL, QUICK

};

void update\_dates(Bubble\_sort& bubble, Insert\_sort& insert, Selection\_sort& selection, Shell\_sort& shell, Quick\_sort& quick);

// Sorting.cpp

#include "sorting.h"

using namespace std;

void Isort::sort(Matrix& arr, Matrix& t\_arr, int row, int col) {

t\_arr.copy(arr);

const int odd\_col = 2;

int odd\_index = 0;

double\* col\_sort = new double[row];

double\* row\_sort = new double[col];

for (int index = 0; index < col; index++) {

if (!(index % odd\_col == 0)) {

for (int columns = 0; columns < row; columns++) {

col\_sort[odd\_index] = arr(columns, index);

odd\_index++;

}

sorting(col\_sort, odd\_index, c\_comparisons, c\_permutations);

odd\_index = 0;

for (int columns = 0; columns < row; columns++) {

t\_arr(columns, index) = col\_sort[odd\_index];

odd\_index++;

}

odd\_index = 0;

}

}

for (int index = 0; index < row; index++) {

for (int columns = 0; columns < col; columns++) {

row\_sort[columns] = t\_arr(index, columns);

}

sorting(row\_sort, col, c\_comparisons, c\_permutations);

for (int columns = 0; columns < col; columns++) {

t\_arr(index, columns) = row\_sort[columns];

}

}

delete[] col\_sort;

delete[] row\_sort;

}

void Bubble\_sort::sorting(double arr[], int length, int& comparisons, int& permutations) {

bool is\_sorted = true;

for (int index = 0; index < length; index++) {

is\_sorted = true;

for (int columns = (length - 1); columns > index; columns--) {

if (arr[columns] > arr[columns - 1]) {

swap(arr[columns], arr[columns - 1]);

permutations++;

is\_sorted = false;

}

comparisons++;

}

if (is\_sorted) break;

}

}

void Insert\_sort::sorting(double arr[], int length, int& comparisons, int& permutations) {

int current\_index = 0;

for (int index = 1; index < length; index++) {

comparisons++;

for (current\_index = index; current\_index > 0 && arr[current\_index - 1] < arr[current\_index]; current\_index--) {

swap(arr[current\_index - 1], arr[current\_index]);

permutations++;

}

}

}

void Selection\_sort::sorting(double arr[], int length, int& comparisons, int& permutations) {

for (int index = 0; index < length; index++) {

int biggest\_index = index;

for (int current\_index = index; current\_index < length; current\_index++) {

comparisons++;

if (arr[current\_index] > arr[biggest\_index]) {

biggest\_index = current\_index;

}

}

if (arr[index] < arr[biggest\_index]) {

swap(arr[index], arr[biggest\_index]);

permutations++;

}

}

comparisons-=length;

}

void Shell\_sort::sorting(double arr[], int length, int& comparisons, int& permutations) {

int second\_element = length / 2;

while (second\_element > 0) {

for (int index = second\_element; index < length; index++) {

int current\_index = index;

if (!(arr[current\_index - second\_element] < arr[current\_index])) comparisons++;

while ((current\_index >= second\_element) && (arr[current\_index - second\_element] < arr[current\_index])) {

swap(arr[current\_index], arr[current\_index - second\_element]);

current\_index = current\_index - second\_element;

comparisons += 2;

permutations++;

}

}

second\_element = second\_element / 2;

}

}

void Quick\_sort::sorting(double arr[], int length, int& comparisons, int& permutations) {

int left\_border = 0;

int right\_border = length - 1;

double key\_element = arr[length / 2];

do {

while (arr[left\_border] > key\_element) {

comparisons++;

left\_border++;

}

while (arr[right\_border] < key\_element) {

comparisons++;

right\_border--;

}

if (left\_border <= right\_border) {

if (arr[left\_border] < arr[right\_border]) {

swap(arr[left\_border], arr[right\_border]);

permutations++;

}

comparisons++;

left\_border++;

right\_border--;

}

} while (left\_border <= right\_border);

if (right\_border > 0) {

sorting(arr, right\_border + 1, comparisons, permutations);

}

if (left\_border < length) {

sorting(&arr[left\_border], length - left\_border, comparisons, permutations);

}

}

void update\_dates(Bubble\_sort& bubble, Insert\_sort& insert, Selection\_sort& selection, Shell\_sort& shell, Quick\_sort& quick) {

bubble.update\_dates();

selection.update\_dates();

insert.update\_dates();

shell.update\_dates();

quick.update\_dates();

}

[--- Конец модуля Sorting]

[Начало модуля Array ---]

//Array.h

#pragma once

class Matrix {

private:

double\*\* arr;

int c\_col, c\_row;

public:

double& operator () (const int row, const int col) {

return arr [row][col];

}

int get\_col() {

return c\_col;

}

int get\_row() {

return c\_row;

}

void set\_size(int col\_t, int row\_t) {

c\_col = col\_t;

c\_row = row\_t;

arr = new double\*[c\_row];

for (int rows = 0; rows < c\_row; rows++) {

arr[rows] = new double[c\_col];

}

}

void copy(Matrix& new\_arr) {

c\_row = new\_arr.c\_row;

c\_col = new\_arr.c\_col;

set\_size(c\_col, c\_row);

for (int rows = 0; rows < c\_row; rows++) {

for (int columns = 0; columns < c\_col; columns++) {

arr[rows][columns] = new\_arr.arr[rows][columns];

}

}

}

~Matrix() {

{

for (int index = 0; index < c\_row; index++) {

delete[] arr[index];

}

delete[] arr;

c\_col = -1;

c\_row = -1;

}

}

};

[--- Конец модуля Array]

[Начало модуля Test ---]

// Test.h

#pragma once

#include <Windows.h>

#include <iostream>

using namespace std;

void run\_tests(void);

// Test.cpp

#include "Test.h"

void update\_dates(void) {

}

bool Test1(Matrix& arr, Matrix& exp\_arr, Matrix& arr\_t) {

Bubble\_sort bubble;

int col = 2;

int row = 2;

arr.set\_size(col, row);

arr\_t.copy(arr);

exp\_arr.set\_size(col, row);

arr(0, 0) = 1;

arr(0, 1) = 2;

arr(1, 0) = 3;

arr(1, 1) = 4;

exp\_arr(0, 0) = 4;

exp\_arr(0, 1) = 1;

exp\_arr(1, 0) = 3;

exp\_arr(1, 1) = 2;

bool correct = true;

bubble.sort(arr, arr\_t, row, col);

for (int rows = 0; rows < arr\_t.get\_row(); rows++) {

for (int columns = 0; columns < arr\_t.get\_col(); columns++) {

if ((exp\_arr(rows, columns) != arr\_t(rows, columns)))

correct = false;

}

}

if (correct) {

return true;

}

else {

cout << "Тест пузырьковой сортировки не пройден!" << endl;

cout << "Ожидалась такая матрица: " << endl;

output\_arr(exp\_arr);

cout << "Получилась такая матрица: " << endl;

output\_arr(arr\_t);

return false;

}

}

bool Test2(Matrix& arr, Matrix& arr\_t, Matrix& exp\_arr ) {

Insert\_sort insert;

int col = 2;

int row = 2;

arr.set\_size(col, row);

arr\_t.copy(arr);

exp\_arr.set\_size(col, row);

arr(0, 0) = 1;

arr(0, 1) = 2;

arr(1, 0) = 3;

arr(1, 1) = 4;

exp\_arr(0, 0) = 4;

exp\_arr(0, 1) = 1;

exp\_arr(1, 0) = 3;

exp\_arr(1, 1) = 2;

bool correct = true;

insert.sort(arr, arr\_t, row, col);

for (int rows = 0; rows < arr\_t.get\_row(); rows++) {

for (int columns = 0; columns < arr\_t.get\_col(); columns++) {

if ((exp\_arr(rows, columns) != arr\_t(rows, columns)))

correct = false;

}

}

if (correct) {

return true;

}

else {

cout << "Тест сортировки вставок не пройден!" << endl;

cout << "Ожидалась такая матрица: " << endl;

output\_arr(exp\_arr);

cout << "Получилась такая матрица: " << endl;

output\_arr(arr\_t);

return false;

}

}

bool Test3(Matrix& arr, Matrix& arr\_t, Matrix& exp\_arr) {

Selection\_sort selection;

int col = 2;

int row = 2;

arr.set\_size(col, row);

arr\_t.copy(arr);

exp\_arr.set\_size(col, row);

arr(0, 0) = 1;

arr(0, 1) = 2;

arr(1, 0) = 3;

arr(1, 1) = 4;

exp\_arr(0, 0) = 4;

exp\_arr(0, 1) = 1;

exp\_arr(1, 0) = 3;

exp\_arr(1, 1) = 2;

bool correct = true;

selection.sort(arr, arr\_t, row, col);

for (int rows = 0; rows < arr\_t.get\_row(); rows++) {

for (int columns = 0; columns < arr\_t.get\_col(); columns++) {

if ((exp\_arr(rows, columns) != arr\_t(rows, columns)))

correct = false;

}

}

if (correct) {

return true;

}

else {

cout << "Тест сортировки выборки не пройден!" << endl;

cout << "Ожидалась такая матрица: " << endl;

output\_arr(exp\_arr);

cout << "Получилась такая матрица: " << endl;

output\_arr(arr\_t);

return false;

}

}

bool Test4(Matrix& arr, Matrix& arr\_t, Matrix& exp\_arr) {

Shell\_sort shell;

int col = 2;

int row = 2;

arr.set\_size(col, row);

arr\_t.copy(arr);

exp\_arr.set\_size(col, row);

arr(0, 0) = 1;

arr(0, 1) = 2;

arr(1, 0) = 3;

arr(1, 1) = 4;

exp\_arr(0, 0) = 4;

exp\_arr(0, 1) = 1;

exp\_arr(1, 0) = 3;

exp\_arr(1, 1) = 2;

bool correct = true;

shell.sort(arr, arr\_t, row, col);

for (int rows = 0; rows < arr\_t.get\_row(); rows++) {

for (int columns = 0; columns < arr\_t.get\_col(); columns++) {

if ((exp\_arr(rows, columns) != arr\_t(rows, columns)))

correct = false;

}

}

if (correct) {

return true;

}

else {

cout << "Тест сортировки Шелла не пройден!" << endl;

cout << "Ожидалась такая матрица: " << endl;

output\_arr(exp\_arr);

cout << "Получилась такая матрица: " << endl;

output\_arr(arr\_t);

return false;

}

}

bool Test5(Matrix& arr, Matrix& arr\_t, Matrix& exp\_arr) {

Quick\_sort quick;

int col = 2;

int row = 2;

arr.set\_size(col, row);

arr\_t.copy(arr);

exp\_arr.set\_size(col, row);

arr(0, 0) = 1;

arr(0, 1) = 2;

arr(1, 0) = 3;

arr(1, 1) = 4;

exp\_arr(0, 0) = 4;

exp\_arr(0, 1) = 1;

exp\_arr(1, 0) = 3;

exp\_arr(1, 1) = 2;

bool correct = true;

quick.sort(arr, arr\_t, row, col);

for (int rows = 0; rows < arr\_t.get\_row(); rows++) {

for (int columns = 0; columns < arr\_t.get\_col(); columns++) {

if ((exp\_arr(rows, columns) != arr\_t(rows, columns)))

correct = false;

}

}

if (correct) {

return true;

}

else {

cout << "Тест быстрой сортировки не пройден!" << endl;

cout << "Ожидалась такая матрица: " << endl;

output\_arr(exp\_arr);

cout << "Получилась такая матрица: " << endl;

output\_arr(arr\_t);

return false;

}

}

void run\_tests(void) {

Matrix arr, exp\_arr, arr\_t;

Bubble\_sort bubble;

Selection\_sort selection;

Insert\_sort insert;

Shell\_sort shell;

Quick\_sort quick;

if ((Test1(arr, arr\_t, exp\_arr)) && (Test2(arr, arr\_t, exp\_arr)) && (Test3(arr, arr\_t, exp\_arr)) &&

(Test4(arr, arr\_t, exp\_arr)) && (Test5(arr, arr\_t, exp\_arr))) cout << "Все тесты пройдены!" << endl;

}

[--- Конец модуля Test]

[--- Конец программы]