

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 | | 09.03.03 | Прикладная информатика | |
| Направленность (профиль) | |  | Прикладная информатика в химии | |
| Форма обучения | |  | очная | |
| Факультет | |  | Информационных технологий и управления | |
| Кафедра | |  | Систем автоматизированного проектирования и управления | |
| Учебная дисциплина | |  | Информационные технологии и программирование | |
| Курс | I | | Группа | 425 |

Отчёт по контрольной работе № 3

Вариант № 22

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Исполнитель:  обучающийся группы 425 |  |  |  | Роева Алёна Олеговна |
|  |  | (дата, подпись) |  |  |
| Проверили: |  |  |  | Корниенко Иван Григорьевич |
|  |  | (дата, подпись) |  | Федин Алексей Константинович |
|  |  |  |  |  |

СОДЕРЖАНИЕ

[1 Постановка задачи 3](#_Toc134661388)

[2 Исходные данные 3](#_Toc134661389)

[3 Особые ситуации 3](#_Toc134661390)

[4 Математические методы и алгоритмы решения задач 3](#_Toc134661391)

[5 Блок-схемы алгоритмов решения задачи: 5](#_Toc134661392)

[6 Форматы представления данных 10](#_Toc134661393)

[7 Структура программы 12](#_Toc134661394)

[8 Описание хода выполнения лабораторной работы 14](#_Toc134661395)

[9 Результаты работы программы 15](#_Toc134661396)

[10 Исходный текст программы 19](#_Toc134661397)

# 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 чистый виртуальный метод Sort и необходимые счетчики, от которого наследовать подклассы для реализации сортировок

Упорядочить каждую строку по возрастанию, затем каждый столбец по убыванию.

# 2 Исходные данные

В качестве исходных данных программа использует данные, введённые пользователем из консоли, данные, генерируемые случайным образом, или данные, взятые из файла. Путь к файлу пользователь вводит вручную.

# 3 Особые ситуации

Необходимо рассмотреть следующие особые ситуации:

* Если пользователь выберет несуществующий пункт меню.
* Если пользователь введёт отрицательное количество поездов.
* Если пользователь введёт некорректное значение.
* Пользователь попытается ввести данные из несуществующего файла.
* Пользователь попытается ввести зарезервированное системой имя файла.
* Пользователь попытается создать файл для сохранения данных с уже занятым именем.

# 4 Математические методы и алгоритмы решения задач

Для решения задачи использовались следующие алгоритмы:

1. Сортировка пузырьком: циклически сравниваем стоящие рядом друг с другом элементы и переставляем в случае необходимости, пока максимальное число не окажется в нужном месте. Повторяем сортировку, исключая переставленный до этого элемент. Сортировка выполняется до тех пор, пока массив не будет отсортирован полностью.
2. Сортировка выбором: циклически сравниваем первый элемент массива, при нахождении числа, большего, чем первый элемент, запоминаем его позицию и продолжаем сравнивать числа уже с ним. Дойдя до конца массива, при необходимости, меняем максимальное число с первым элементом. Повторяем сортировку, исключая переставленный до этого элемент. Сортировка выполняется до тех пор, пока массив не будет отсортирован полностью.
3. Сортировка вставками: в ограниченной области сравниваются и перестанавливаются числа до тех пор, пока они не займут свои места, затем область увеличивается на один элемент и повторяются предыдущие действия. Когда ограниченная область станет равной по размерам с исходным, сортировка будет завершена.
4. Сортировка Шелла: работает как сортировка вставками, но вместо ограниченной области, сравниваются элементы, находящие друг от друга на расстоянии, равном половине размера массива. С каждой итерацией цикла шаг уменьшается в 2 раза.
5. Быстрая сортировка: в качестве опорного элемента выбирается крайний левый элемент, слева от которого должны оказаться числа, меньшие, чем опорный элемент, а справа элементы, большие, чем опорный элемент. Далее этот алгоритм рекурсивно повторяется для массивов элементов, расположенных перед опорным числом и после опорного числа, до тех пор, пока массив не будет отсортирован.

# 5 Блок-схемы алгоритмов решения задачи:



Рисунок 1 – Блок-схема алгоритма пузырьковой сортировки



Рисунок 2 – Блок-схема алгоритма сортировки выбором



Рисунок 3 – Блок-схема алгоритма сортировки вставками



Рисунок 4 – Блок-схема алгоритма сортировки Шелла



Рисунок 5 – Блок-схема алгоритма быстрой сортировки

# 6 Форматы представления данных

Таблица 1 – Описание пользовательского класса ISort, используемого в программе

|  |  |  |  |
| --- | --- | --- | --- |
| **Имя класса** | | **Назначение класса** | **Иерархическая позиция класса** |
| ISort | | Интерфейс для алгоритмов сортировки | Исходный класс |
| **Поля класса** | | | |
| **Тип данных** | **Имя** | **Назначение** | |
| int | comparison | Поле класса, хранит кол-во сравнений алгоритма сортировки | |
| int | permutation | Поле класса, хранит кол-во перестановок алгоритма сортировки | |

Таблица 2 – Описание пользовательского класса BubbleSort, используемого в программе

|  |  |  |
| --- | --- | --- |
| **Имя класса** | **Назначение класса** | **Иерархическая позиция класса** |
| BubbleSort | Алгоритм сортировки пузырьком | Наследование от **ISort** |
| **Поля класса** | | |
| Наследуются от ISort | | |

Таблица 3 – Описание пользовательского класса SelectionSort, используемого в программе

|  |  |  |
| --- | --- | --- |
| **Имя класса** | **Назначение класса** | **Иерархическая позиция класса** |
| SelectionSort | Алгоритм сортировки выбором | Наследование от **ISort** |
| **Поля класса** | | |
| Наследуются от ISort | | |

Таблица 4 – Описание пользовательского класса InsertionSort, используемого в программе

|  |  |  |
| --- | --- | --- |
| **Имя класса** | **Назначение класса** | **Иерархическая позиция класса** |
| InsertionSort | Алгоритм сортировки вставками | Наследование от **ISort** |
| **Поля класса** | | |
| Наследуются от ISort | | |

Таблица 5 – Описание пользовательского класса shellsort, используемого в программе

|  |  |  |
| --- | --- | --- |
| **Имя класса** | **Назначение класса** | **Иерархическая позиция класса** |
| ShellSort | Алгоритм сортировки Шелла | Наследование от **ISort** |
| **Поля класса** | | |
| Наследуются от ISort | | |

Таблица 6 – Описание пользовательского класса quicksort, используемого в программе

|  |  |  |
| --- | --- | --- |
| **Имя класса** | **Назначение класса** | **Иерархическая позиция класса** |
| QuickSort | Алгоритм быстрой сортировки | Наследование от **ISort** |
| **Поля класса** | | |
| Наследуются от ISort | | |

Ниже представлена информация о форматах данных, используемых в файлах.

Файл, содержащий начальные данные, состоит из элементов матрицы.

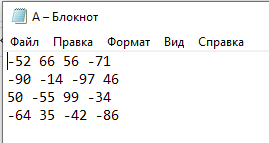


Рисунок 6 – Структура файла с начальными данными программы

Файл, содержащий выходные данные программы, состоит из трёх элементов: количество строк матрицы, количество столбцов матрицы, элементы матрицы.

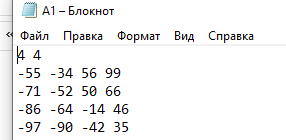


Рисунок 7 – Структура файла с выходными данными программы

# 7 Структура программы

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

1. algoritms.cpp
2. ISort.cpp
3. main.cpp
4. menu.cpp
5. moduletests.cpp

Таблица 7 – Функции, составляющие algorithms.cpp

|  |  |
| --- | --- |
| **Имя** | **Описание** |
| СheckInt | Проверка пользовательского ввода числа. |
| ReadStringWithoutWhitespace | Проверка на пустой ввод строки. |
| File\_Exists | Проверка на существование файла. |
| Only\_Read | Проверка на возможность открытия файла для записи. |
| IsValidFilename | Проверка на допустимые символы в имени файла. |
| getFilename | Проверка на путь к файлу. |

Продолжение таблицы 7

|  |  |
| --- | --- |
| **Имя** | **Описание** |
| IsReservedFileName | Проверка на зарезервированные имена файла. |
| PrintMatrix | Вывод матрицы. |
| CopyMatrix | Копирование элементов одного массива в другой. |

Таблица 8 – Функции, составляющие ISort.cpp

|  |  |
| --- | --- |
| **Имя** | **Описание** |
| Sort | Функция, выполняющая сортировку матрицы. |
| GetCountOfComparison | Метод класса. Возвращает количество сравнений. |
| GetCountOfPermutation | Метод класса. Возвращает количество перестановок. |

Таблица 9 – Функции, составляющие moduletests.cpp

|  |  |
| --- | --- |
| **Имя** | **Описание** |
| CompleteModuleTests | Функция, запускающая модульные тесты. |

Таблица 10 – Функции, составляющие menu.cpp

|  |  |
| --- | --- |
| **Имя** | **Описание** |
| Print\_menu | Вызов пунктов главного меню. |
| InputMethodMenu | Вызов меню выбора способа ввода данных. |
| MainMenu | Вызов главного меню. |
| CompleteControlWork | Функция выполнения контрольной работы. |
| SaveMatrixIntoFile | Функция сохранения данных в файл |

main.cpp – основной модуль, осуществляющий запуск программы. После запуска выводится приветствие, информация о программе и об исполнителе, а также главное меню. Главное меню позволяет начать выполнение основной программы, запустить модульные тесты, или же сразу закрыть всю программу целиком. При выборе запуска основной программы выводится побочное меню, которое позволяет выбрать способ ввода данных: из файла или из консоли. При выборе ввода из консоли пользователь сам указывает размеры массивы и способ заполнения массива: вручную или случайными числами. При выборе ввода из файла, пользователь должен указать путь к файлу, затем программа сама подсчитает размеры массива и считает данные о его элементах. Далее сработают алгоритмы сортировки и выведутся данные о количестве сравнений и количестве перестановок каждого алгоритма. Затем пользователь может сохранить отсортированный массив в файл или вернуться в главное меню, ничего не делая.

# 8 Описание хода выполнения лабораторной работы

1. В ходе лабораторной работы было создано решение в интегрированной среде разработки Microsoft Visual Studio C++ 2022. В нем был создан проект.
2. При написании программы не возникло никаких проблем.

# 9 Результаты работы программы

Далее представлены результаты работы программы с различными вариантами исходных данных.

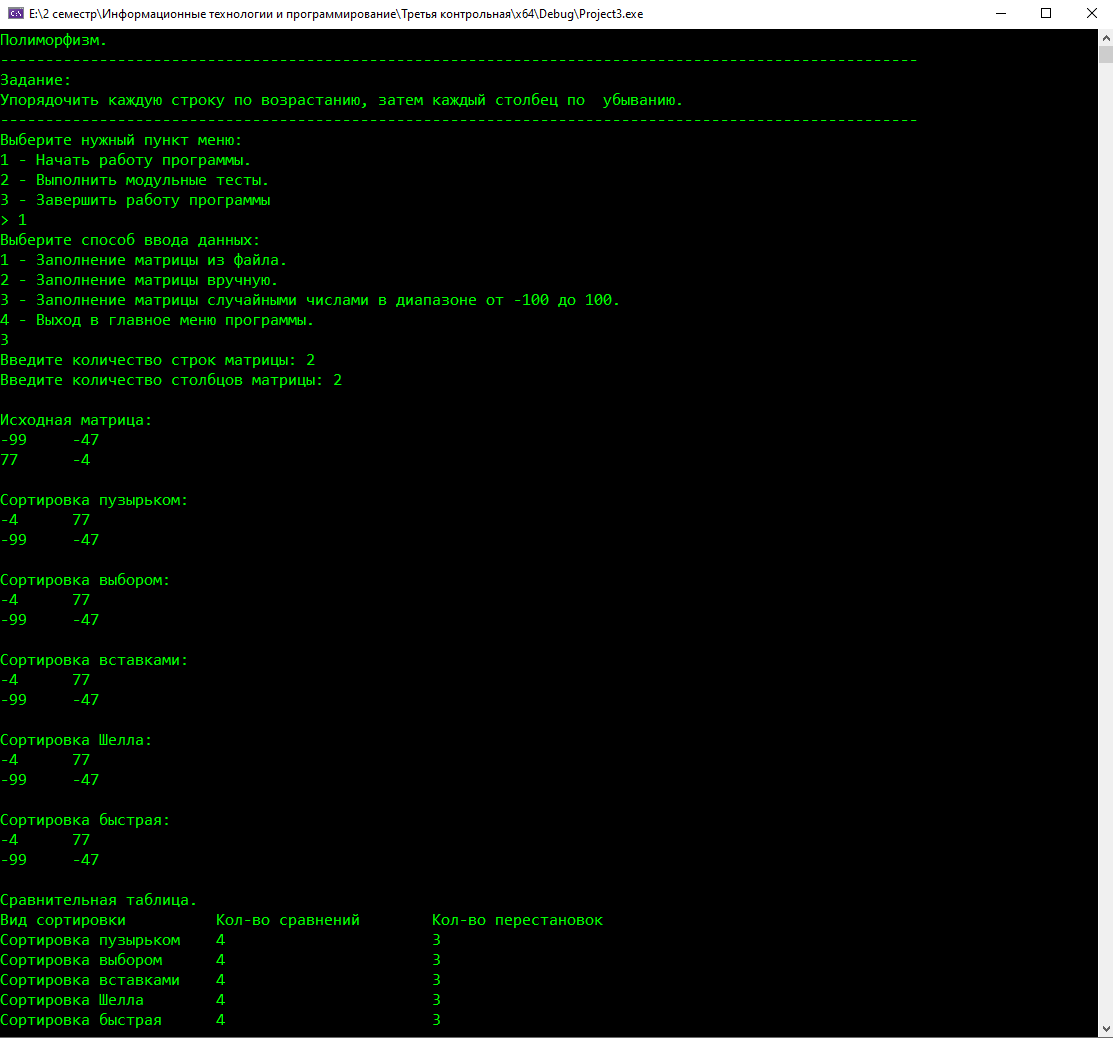


Рисунок 8 – Результаты программы по исходным данным из консоли 1

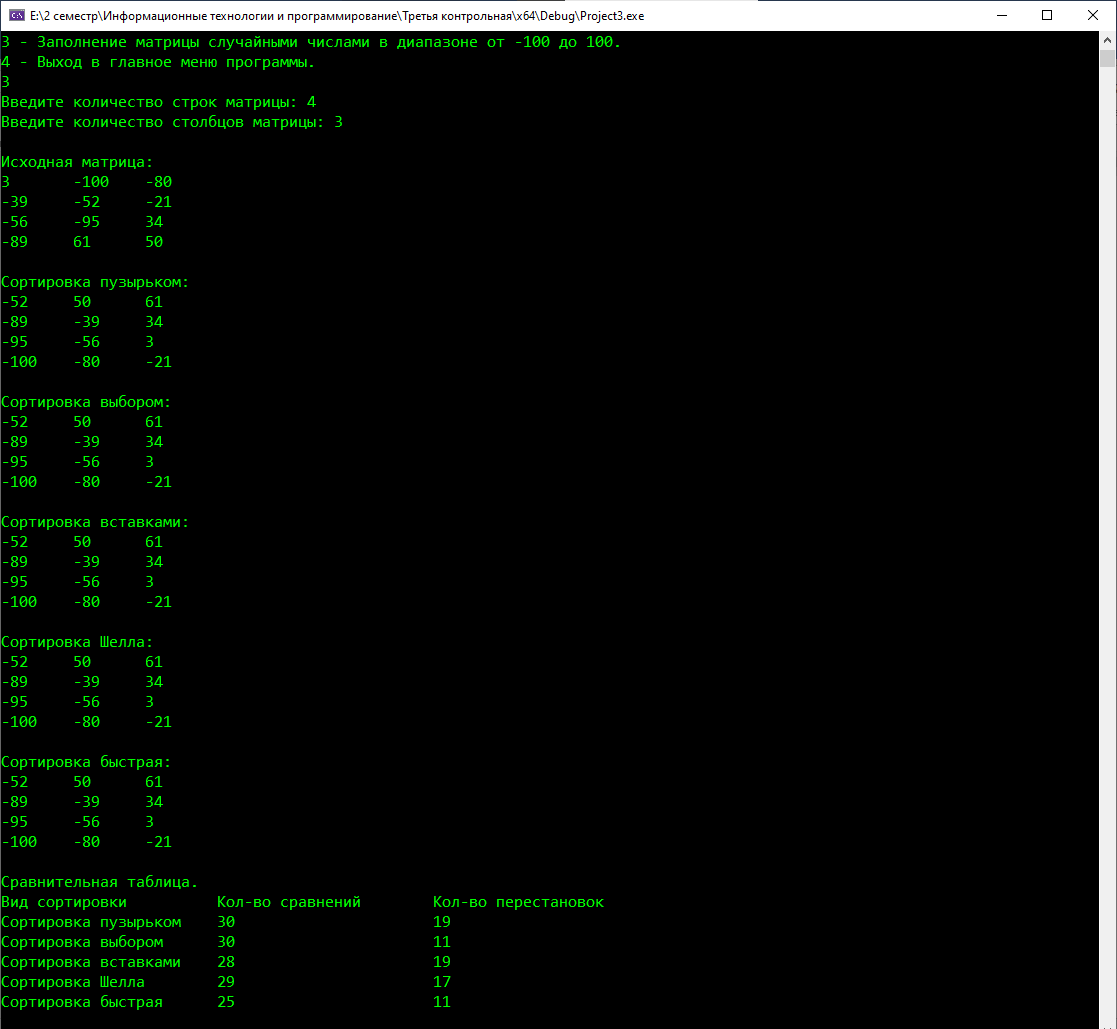


Рисунок 9 – Результаты программы по исходным данным из консоли 2

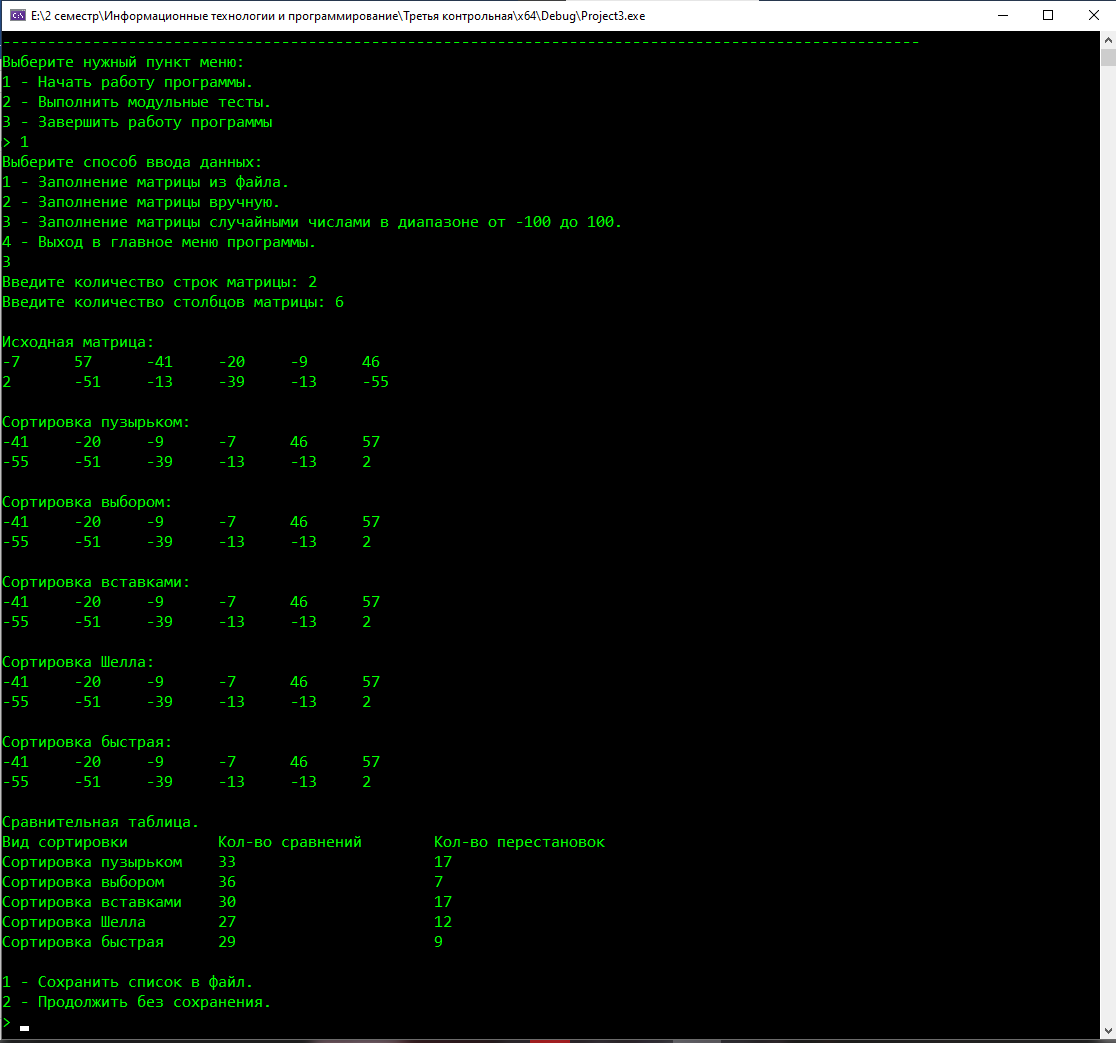


Рисунок 10 – Результаты программы по исходным данным из консоли 3

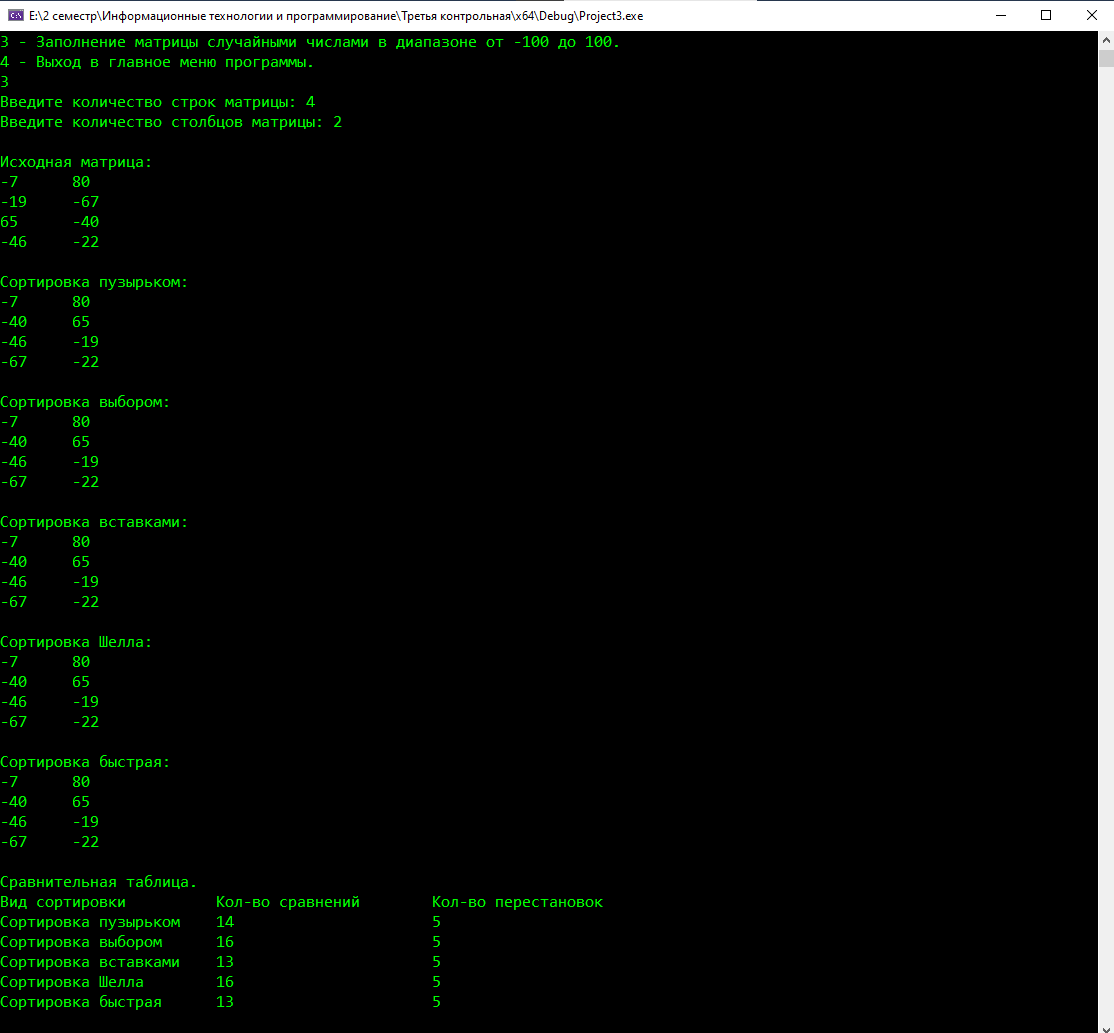


Рисунок 11 – Результаты программы по исходным данным из консоли 4

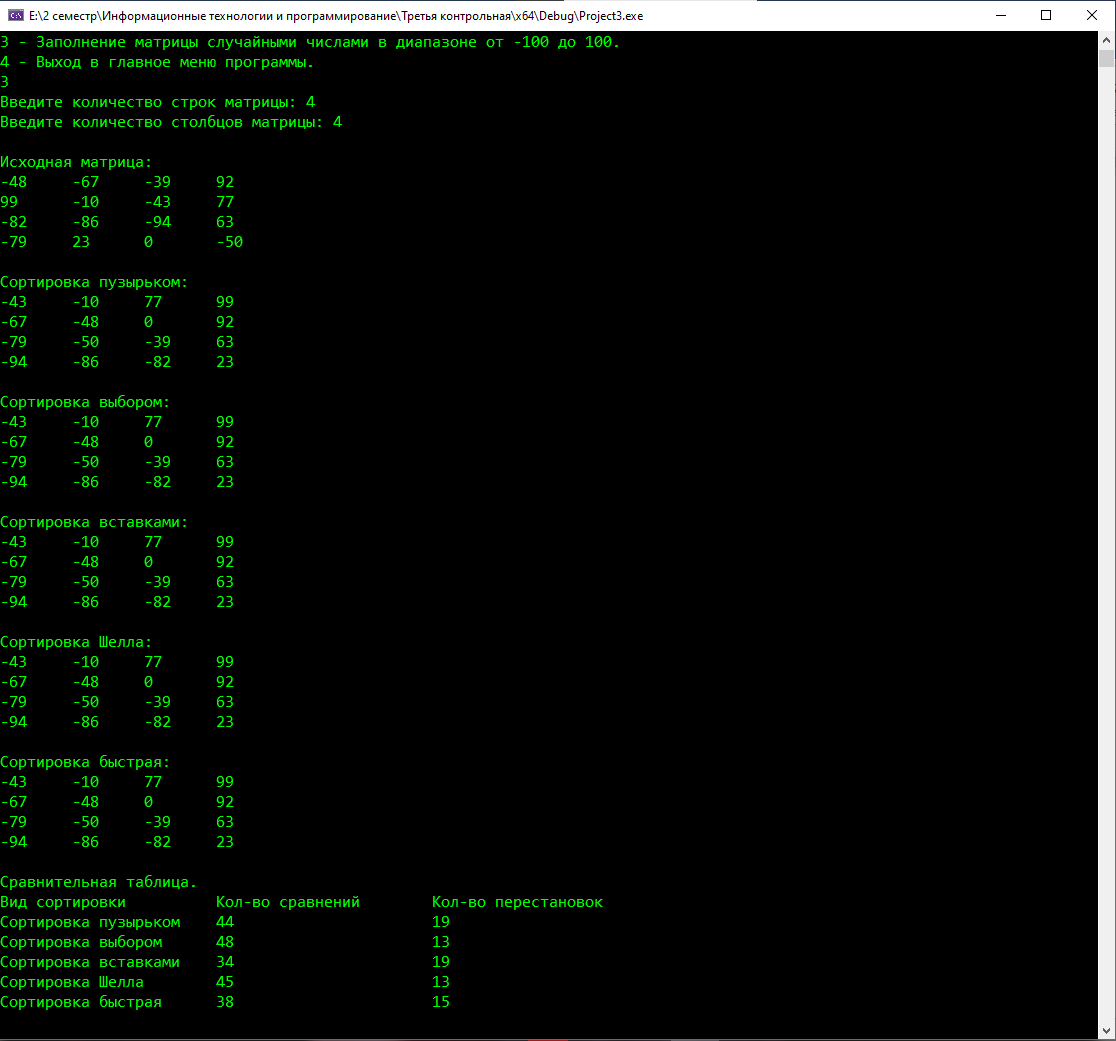


Рисунок 12 – Результаты программы по исходным данным из консоли 5

# 10 Исходный текст программы

**[ Начало moduletests.h--- ]**

//Пономарев Константин, 425.

//заголовочный файл, объявляет все функции для moduletests.cpp.

#pragma once

#include "ISort.h"

void CompleteModuleTests();

**[---moduletests.h]**

**[ Начало algorithms.h--- ]**

//Пономарев Константин, 425.

//заголовочный файл, объявляет все функции для algorithms.cpp.

#pragma once

#include <iostream>

#include <fstream>

#include <string>

#include <sstream>

#include <limits>

#include <io.h>

#include <regex>

using namespace std;

int CheckInt();

string ReadStringWithoutWhitespace();

bool File\_Exists(string FileName);

bool Only\_Read(string FileName);

bool IsValidFilename(const string& filename);

string getFilename(const string path);

bool IsReservedFileName(string FileName);

void CopyMatrix(int\*\* copymatrix, int\*\* matrix, int strings, int columns);

void PrintMatrix(int\*\* matrix, int strings, int columns);

**[---algorithms.h]**

**[ Начало ISort.h--- ]**

//Пономарев Константин, 425.

//заголовочный файл, объявляет интерфейс ISort и классы сортировок.

#pragma once

#include "algorithms.h"

using namespace std;

class ISort {

protected:

int countOfComparison = 0;

int countOfPermutation = 0;

public:

virtual void Sort(int\*\* matrix, int strings, int columns) = 0;

int GetCountOfComparison();

int GetCountOfPermutation();

virtual ~ISort() = default;

};

class BubbleSort : public ISort {

public:

void Sort(int\*\* matrix, int strings, int columns) override;

};

class SelectionSort : public ISort {

public:

void Sort(int\*\* matrix, int strings, int columns) override;

};

class InsertSort : public ISort {

public:

void Sort(int\*\* matrix, int strings, int columns) override;

};

class ShellSort : public ISort {

public:

void Sort(int\*\* matrix, int strings, int columns) override;

};

class QuickSort : public ISort {

public:

void Sort(int\*\* matrix, int strings, int columns) override;

void PartitionStrings(int\*\* matrix, int string, int start, int end);

void PartitionColumns(int\*\* matrix, int column, int start, int end);

};

**[---ISort.h]**

**[ Начало menu.h--- ]**

//Пономарев Константин, 425.

//заголовочный файл, объявляет меню для главной программы, меню для выбора способа заполнения данных.

#pragma once

#include "ISort.h"

using namespace std;

void Print\_menu();

void InputMethodMenu();

void CompleteControlWork(int\*\* matrix, int strings, int columns);

void SaveMatrixIntoFile(int\*\* matrix, int\*\* sortedmatrix, int strings, int columns, BubbleSort a, SelectionSort b, InsertSort c, ShellSort d, QuickSort e);

void MainMenu();

**[---menu.h]**

**[ Начало moduletests.cpp--- ]**

//Пономарев Константин, 425.

//cpp-файл, содержит модульные тесты.

#include "moduletests.h"

#define NumberOfTests 5

using namespace std;

void CompleteModuleTests() {

ifstream initial, final;

bool CompleteTests = true;

bool FilesExists = true;

string initialPaths[NumberOfTests] = { "A.txt", "B.txt", "C.txt", "D.txt", "E.txt" };

string finalPaths[NumberOfTests] = { "A1.txt", "B1.txt", "C1.txt", "D1.txt", "E1.txt" };

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

if (!File\_Exists(initialPaths[i]) || !File\_Exists(finalPaths[i])) {

FilesExists = false;

}

}

if (FilesExists) {

for (int t = 0; t < NumberOfTests; t++) {

string temp;

string value;

bool BubbleCorrect = true;

bool SelectionCorrect = true;

bool InsertCorrect = true;

bool ShellCorrect = true;

bool QuickCorrect = true;

int finalStrings = 0, finalColumns = 0;

int initialStrings = 0, initialColumns = 0;

int\*\* initialMatrix = nullptr;

int\*\* finalMatrix = nullptr;

int\*\* sortedMatrix = nullptr;

BubbleSort a;

SelectionSort b;

InsertSort c;

ShellSort d;

QuickSort e;

initial.open(initialPaths[t]);

final.open(finalPaths[t]);

int input = 0;

while (!initial.eof()) {

if ((input = initial.get()) == '\n' || input == EOF) { initialStrings += 1; }

}

initial.clear();

initial.seekg(0);

getline(initial, temp);

istringstream stream(temp);

while (stream >> value) {

initialColumns++;

}

initial.clear();

initial.seekg(0);

initialMatrix = new int\* [initialStrings];

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

initialMatrix[i] = new int[initialColumns];

}

final >> finalStrings;

final >> finalColumns;

finalMatrix = new int\* [finalStrings];

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

finalMatrix[i] = new int[finalColumns];

}

sortedMatrix = new int\* [initialStrings];

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

sortedMatrix[i] = new int[initialColumns];

}

if (initialStrings != finalStrings || initialColumns != finalColumns) {

cout << "Количество строк или столбцов матрицы в " << initialPaths[t] << " не соответствует количеству строк или столбцов " << finalPaths[t] << endl;

CompleteTests = false;

}

else {

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

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

initial >> initialMatrix[i][j];

}

}

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

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

final >> finalMatrix[i][j];

}

}

CopyMatrix(sortedMatrix, initialMatrix, initialStrings, initialColumns);

a.Sort(sortedMatrix, initialStrings, initialColumns);

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

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

if (sortedMatrix[i][j] != finalMatrix[i][j]) {

BubbleCorrect = false;

}

}

}

if (!BubbleCorrect) {

CompleteTests = false;

cout << "Сортировка пузырьком работает некорректно." << endl;

cout << "Полученный массив:" << endl;

PrintMatrix(sortedMatrix, initialStrings, initialColumns);

cout << "Ожидаемый массив:" << endl;

PrintMatrix(finalMatrix, initialStrings, initialColumns);

}

CopyMatrix(sortedMatrix, initialMatrix, initialStrings, initialColumns);

b.Sort(sortedMatrix, initialStrings, initialColumns);

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

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

if (sortedMatrix[i][j] != finalMatrix[i][j]) {

SelectionCorrect = false;

}

}

}

if (!SelectionCorrect) {

CompleteTests = false;

cout << "Сортировка выбором работает некорректно." << endl;

cout << "Полученный массив:" << endl;

PrintMatrix(sortedMatrix, initialStrings, initialColumns);

cout << "Ожидаемый массив:" << endl;

PrintMatrix(finalMatrix, initialStrings, initialColumns);

}

CopyMatrix(sortedMatrix, initialMatrix, initialStrings, initialColumns);

c.Sort(sortedMatrix, initialStrings, initialColumns);

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

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

if (sortedMatrix[i][j] != finalMatrix[i][j]) {

InsertCorrect = false;

}

}

}

if (!InsertCorrect) {

CompleteTests = false;

cout << "Сортировка вставками работает некорректно." << endl;

cout << "Полученный массив:" << endl;

PrintMatrix(sortedMatrix, initialStrings, initialColumns);

cout << "Ожидаемый массив:" << endl;

PrintMatrix(finalMatrix, initialStrings, initialColumns);

}

CopyMatrix(sortedMatrix, initialMatrix, initialStrings, initialColumns);

d.Sort(sortedMatrix, initialStrings, initialColumns);

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

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

if (sortedMatrix[i][j] != finalMatrix[i][j]) {

ShellCorrect = false;

}

}

}

if (!ShellCorrect) {

CompleteTests = false;

cout << "Сортировка Шелла работает некорректно." << endl;

cout << "Полученный массив:" << endl;

PrintMatrix(sortedMatrix, initialStrings, initialColumns);

cout << "Ожидаемый массив:" << endl;

PrintMatrix(finalMatrix, initialStrings, initialColumns);

}

CopyMatrix(sortedMatrix, initialMatrix, initialStrings, initialColumns);

e.Sort(sortedMatrix, initialStrings, initialColumns);

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

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

if (sortedMatrix[i][j] != finalMatrix[i][j]) {

QuickCorrect = false;

}

}

}

if (!QuickCorrect) {

CompleteTests = false;

cout << "Сортировка быстрая работает некорректно." << endl;

cout << "Полученный массив:" << endl;

PrintMatrix(sortedMatrix, initialStrings, initialColumns);

cout << "Ожидаемый массив:" << endl;

PrintMatrix(finalMatrix, initialStrings, initialColumns);

}

}

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

delete[] initialMatrix[i];

initialMatrix[i] = nullptr;

}

delete[] initialMatrix;

initialMatrix = nullptr;

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

delete[] sortedMatrix[i];

sortedMatrix[i] = nullptr;

}

delete[] sortedMatrix;

sortedMatrix = nullptr;

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

delete[] finalMatrix[i];

finalMatrix[i] = nullptr;

}

delete[] finalMatrix;

finalMatrix = nullptr;

final.close();

initial.close();

}

if (CompleteTests) {

cout << "Задание работает корректно." << endl;

}

}

else {

cout << "Не обнаружено файлов для проведения модульных тестов! Файлы должны называться: начальные данные - A-E, конечные - A1-E1" << endl;

}

}

**[---moduletests.cpp]**

**[ Начало algorithms.cpp--- ]**

//Пономарев Константин, 425.

//cpp-файл, содержит проверки ввода, проверку на зарезервированные системой имена, функции для работы с матрицами.

#include "algorithms.h"

using namespace std;

int CheckInt() {

int tmp = 0;

cin >> tmp;

while (cin.fail()) {

cout << "Введено некорректное значение, попробуйте еще раз: ";

cin.clear();

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

cin >> tmp;

}

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

return tmp;

}

string ReadStringWithoutWhitespace() {

string input;

while (true) {

getline(cin, input);

istringstream iss(input);

if (iss >> input && !isspace(input[0])) {

return input;

}

else {

cout << "Ошибка: входные данные не должны содержать пробелов. Повторите попытку ввода." << endl;

}

}

}

bool File\_Exists(string FileName) {

const char\* path = FileName.c\_str();

if (!\_access(path, 0)) {

return true;

}

else {

return false;

}

}

bool Only\_Read(string FileName) {

const char\* path = FileName.c\_str();

if (\_access(path, 2)) {

return true;

}

else

return false;

}

bool IsValidFilename(const string& filename) {

static const regex pattern("^[a-zA-Z0-9\_\\-\\.]+$");

return regex\_match(filename, pattern);

}

string getFilename(const string path) {

size\_t pos = path.find\_last\_of("\\"); // Ищем последний символ '/' или '\'

if (pos != string::npos) { // Если символ найден

return path.substr(pos + 1); // Возвращаем имя файла после найденного символа

}

return path; // Если символ не найден, возвращаем исходный путь

}

bool IsReservedFileName(string FileName) {

const int QuantityOfReservedNames = 22;

string ReservedNames[QuantityOfReservedNames] = { "con", "prn" , "aux" , "nul" , "com1" , "com2" , "com3" , "com4" , "com5" , "com6" , "com7" , "com8" , "com9" ,

"lpt1", "lpt2", "lpt3", "lpt4", "lpt5", "lpt6", "lpt7", "lpt8", "lpt9" };

size\_t pos = FileName.find\_first\_of(".");

if (pos != string::npos) {

FileName = FileName.substr(0, pos);

}

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

if (\_stricmp(FileName.c\_str(), ReservedNames[i].c\_str()) == 0) {

return true;

}

}

return false;

}

void CopyMatrix(int\*\* copymatrix, int\*\* matrix, int strings, int columns) {

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

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

copymatrix[i][j] = matrix[i][j];

}

}

}

void PrintMatrix(int\*\* matrix, int strings, int columns) {

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

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

cout << matrix[i][j] << '\t';

}

cout << endl;

}

}

**[---algorithms.cpp]**

**[ Начало ISort.cpp--- ]**

//Пономарев Константин, 425.

//cpp-файл, содержит определение интерфейса ISort и классов сортировки.

#include "ISort.h"

using namespace std;

int ISort::GetCountOfComparison() {

return countOfComparison;

}

int ISort::GetCountOfPermutation() {

return countOfPermutation;

}

void BubbleSort::Sort(int\*\* matrix, int strings, int columns) {

bool IsSwap = false;

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

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

IsSwap = false;

for (int k = 0; k < columns - j - 1; k++) {

if (matrix[i][k] > matrix[i][k + 1]) {

swap(matrix[i][k], matrix[i][k + 1]);

IsSwap = true;

countOfPermutation++;

countOfComparison++;

}

else {

countOfComparison++;

}

}

if (!IsSwap) {

break;

}

}

}

IsSwap = false;

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

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

IsSwap = false;

for (int k = 0; k < strings - j - 1; k++) {

if (matrix[k][i] < matrix[k + 1][i]) {

swap(matrix[k][i], matrix[k + 1][i]);

IsSwap = true;

countOfPermutation++;

countOfComparison++;

}

else {

countOfComparison++;

}

}

if (!IsSwap) {

break;

}

}

}

}

void SelectionSort::Sort(int\*\* matrix, int strings, int columns) {

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

int MinPose = 0;

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

MinPose = j;

for (int k = j+1; k < columns; k++) {

if (matrix[i][k] < matrix[i][MinPose]) {

MinPose = k;

}

countOfComparison++;

}

if (MinPose != j) {

swap(matrix[i][j], matrix[i][MinPose]);

countOfPermutation++;

}

}

}

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

int MaxPose = 0;

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

MaxPose = j;

for (int k = j+1; k < strings; k++) {

if (matrix[k][i] > matrix[MaxPose][i]) {

MaxPose = k;

}

countOfComparison++;

}

if (MaxPose != j) {

swap(matrix[j][i], matrix[MaxPose][i]);

countOfPermutation++;

}

}

}

}

void InsertSort::Sort(int\*\* matrix, int strings, int columns) {

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

for (int j = 0; j < columns - 1; j++) {

for (int k = j + 1; k > 0; k--) {

if (matrix[i][k] < matrix[i][k-1]) {

swap(matrix[i][k], matrix[i][k-1]);

countOfPermutation++;

countOfComparison++;

}

else {

countOfComparison++;

break;

}

}

}

}

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

for (int j = 0; j < strings - 1; j++) {

for (int k = j + 1; k > 0; k--) {

if (matrix[k][i] > matrix[k - 1][i]) {

swap(matrix[k][i], matrix[k - 1][i]);

countOfPermutation++;

countOfComparison++;

}

else {

countOfComparison++;

break;

}

}

}

}

}

void ShellSort::Sort(int\*\* matrix, int strings, int columns) {

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

for (int step = columns / 2; step > 0; step /= 2) {

for (int j = step; j < columns; j++) {

for (int k = j; k >= step; k-=step) {

if (matrix[i][k] < matrix[i][k - step]) {

swap(matrix[i][k], matrix[i][k - step]);

countOfPermutation++;

countOfComparison++;

}

else {

countOfComparison++;

break;

}

}

}

}

}

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

for (int step = strings / 2; step > 0; step /= 2) {

for (int j = step; j < strings; j++) {

for (int k = j; k >= step; k-=step) {

if (matrix[k][i] > matrix[k - step][i]) {

swap(matrix[k][i], matrix[k - step][i]);

countOfPermutation++;

countOfComparison++;

}

else {

countOfComparison++;

break;

}

}

}

}

}

}

void QuickSort::Sort(int\*\* matrix, int strings, int columns) {

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

PartitionStrings(matrix, i, 0, columns - 1);

}

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

PartitionColumns(matrix, i, 0, strings - 1);

}

}

void QuickSort::PartitionStrings(int\*\* matrix, int string, int start, int end) {

if (start < end) {

int i = start - 1;

int j = start;

int piv = end;

while (j < piv && i < j) {

if (matrix[string][j] > matrix[string][piv]) {

j++;

}

else {

i++;

if (i != j) {

swap(matrix[string][i], matrix[string][j]);

countOfPermutation++;

}

j++;

}

countOfComparison++;

}

if (i + 1 != j) {

swap(matrix[string][i + 1], matrix[string][j]);

countOfPermutation++;

piv = i + 1;

}

PartitionStrings(matrix, string, start, i);

PartitionStrings(matrix, string, piv + 1, j);

}

return;

}

void QuickSort::PartitionColumns(int\*\* matrix, int column, int start, int end) {

if (start < end) {

int i = start - 1;

int j = start;

int piv = end;

while (j < piv && i < j) {

if (matrix[j][column] < matrix[piv][column]) {

j++;

}

else {

i++;

if (i != j) {

swap(matrix[i][column], matrix[j][column]);

countOfPermutation++;

}

j++;

}

countOfComparison++;

}

if (i + 1 != j) {

swap(matrix[i + 1][column], matrix[j][column]);

countOfPermutation++;

piv = i + 1;

}

PartitionColumns(matrix, column, start, i);

PartitionColumns(matrix, column, piv + 1, j);

}

return;

}

**[---ISort.cpp]**

**[ Начало menu.cpp--- ]**

//Пономарев Константин, 425.

//cpp-файл, предоставляет пользовательский интерфейс.

#include "menu.h"

#include "moduletests.h"

using namespace std;

enum PrintMenu {

StartProgramm = 1,

StartModuleTests = 2,

StopProgramm = 3

};

enum InputMethod {

FromFile = 1,

ManualMethod = 2,

RandomMethod = 3,

ReturnMenu = 4

};

enum SaveListMenu {

Save = 1,

ContinueWithoutSaving = 2

};

enum SaveMethod {

RewriteFile = 1,

WriteToTheEndOfTheFile = 2,

SelectAnotherFile = 3

};

enum CreatingAFile {

CreateFile = 1,

ContunueWithoutCreatingFile = 2

};

void Print\_menu() {

system("cls");

cout << "Пономарев Константин 425 группа." << endl << "Контрольная работа №3. Вариант 20." << endl;

cout << "Полиморфизм." << endl;

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

cout << "Задание:" << endl;

cout << "Упорядочить каждую строку по возрастанию, затем каждый столбец по убыванию." << endl;

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

cout << "Выберите нужный пункт меню:" << endl << "1 - Начать работу программы." << endl << "2 - Выполнить модульные тесты." << endl

<< "3 - Завершить работу программы" << endl << "> ";

}

void InputMethodMenu() {

bool FileIsCorrect = true;

int strings = 0;

int columns = 0;

int input = 0;

int intTmp;

string temp;

string path;

string value;

int\* tmp = nullptr;

int\*\* matrix = nullptr;

ifstream file;

int UserChoise1 = 0;

do {

cout << "Выберите способ ввода данных:" << endl << "1 - Заполнение матрицы из файла." << endl

<< "2 - Заполнение матрицы вручную." << endl << "3 - Заполнение матрицы случайными числами в диапазоне от -100 до 100." << endl

<< "4 - Выход в главное меню программы." << endl;

UserChoise1 = CheckInt();

switch (UserChoise1) {

case FromFile:

cout << "Введите путь файла:" << endl << "> ";

path = ReadStringWithoutWhitespace();

while (!File\_Exists(path)) {

cout << "Ошибка при открытии файла! Попробуйте ввести путь снова." << endl << "> ";

path = ReadStringWithoutWhitespace();

}

file.open(path);

cout << "Файл успешно открыт!" << endl;

if (file.peek() == ifstream::traits\_type::eof()) {

cout << "Файл пуст. Заполните файл и попробуйте еще раз." << endl;

}

else {

while (!file.eof()) {

file >> intTmp;

if (file.fail()) {

file.clear();

FileIsCorrect = false;

break;

}

}

file.clear();

file.seekg(0);

while (!file.eof()) {

if ((input = file.get()) == '\n' || input == EOF) { strings += 1; }

}

file.clear();

file.seekg(0);

tmp = new int[strings];

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

tmp[i] = 0;

getline(file, temp);

istringstream stream(temp);

while (stream >> value) {

tmp[i]++;

}

columns = tmp[i];

}

file.clear();

file.seekg(0);

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

if (tmp[i] != tmp[i + 1]) {

FileIsCorrect = false;

}

}

delete[] tmp;

tmp = nullptr;

if (!FileIsCorrect) {

cout << "В файле содержится некорректно заполненная матрица. Исправьте файл и попробуйте еще раз." << endl;

file.close();

UserChoise1 = ReturnMenu;

break;

}

matrix = new int\* [strings];

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

matrix[i] = new int[columns];

}

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

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

file >> matrix[i][j];

}

}

CompleteControlWork(matrix, strings, columns);

}

file.close();

UserChoise1 = ReturnMenu;

break;

case ManualMethod:

cout << "Введите количество строк матрицы: ";

strings = CheckInt();

while (strings < 1) {

cout << "Количество строк не может быть меньше нуля. Попробуйте ввести другое число." << endl;

strings = CheckInt();

}

cout << "Введите количество столбцов матрицы: ";

columns = CheckInt();

while (columns < 1) {

cout << "Количество столбцов не может быть меньше нуля. Попробуйте ввести другое число." << endl;

columns = CheckInt();

}

matrix = new int\* [strings];

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

matrix[i] = new int[columns];

}

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

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

cout << "Ввод " << i + 1 << " строки " << j + 1 << " столбца." << endl;

matrix[i][j] = CheckInt();

}

}

CompleteControlWork(matrix, strings, columns);

UserChoise1 = ReturnMenu;

break;

case RandomMethod:

cout << "Введите количество строк матрицы: ";

strings = CheckInt();

while (strings < 1) {

cout << "Количество строк не может быть меньше нуля. Попробуйте ввести другое число." << endl;

strings = CheckInt();

}

cout << "Введите количество столбцов матрицы: ";

columns = CheckInt();

while (columns < 1) {

cout << "Количество столбцов не может быть меньше нуля. Попробуйте ввести другое число." << endl;

columns = CheckInt();

}

matrix = new int\* [strings];

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

matrix[i] = new int[columns];

}

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

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

matrix[i][j] = -100 + rand() % 201;

}

}

CompleteControlWork(matrix, strings, columns);

UserChoise1 = ReturnMenu;

break;

case ReturnMenu:

break;

default:

cout << "Такого пункта нет в меню, попробуйте еще раз" << endl;

break;

}

if (UserChoise1 != ReturnMenu) system("pause");

} while (UserChoise1 != ReturnMenu);

}

void SaveMatrixIntoFile(int\*\* matrix, int\*\* sortedmatrix, int strings, int columns, BubbleSort a, SelectionSort b, InsertSort c, ShellSort d, QuickSort e) {

string FileName;

ofstream file;

int SaveChoise = 0;

int SaveMethod = 0;

int FileMethod = 0;

int SaveData = 0;

do {

cout << "1 - Сохранить список в файл." << endl << "2 - Продолжить без сохранения." << endl << "> ";

SaveChoise = CheckInt();

switch (SaveChoise) {

case Save:

while (true) {

cout << "Выберите данные для сохранения:" << endl << "1 - Сохранить результат выполнения программы." << endl << "2 - Сохранить исходные данные." << endl << "> ";

SaveData = CheckInt();

while (SaveData != 1 && SaveData != 2) {

cout << "Такого пункта меню нету, попробуйте ввести другое число." << endl << "> ";

SaveData = CheckInt();

}

cout << "Введите путь к файлу для сохранения:" << endl << "> ";

FileName = ReadStringWithoutWhitespace();

while (IsReservedFileName(getFilename(FileName)) || !IsValidFilename(getFilename(FileName))) {

cout << "Недопустимое название файла. Попробуйте еще раз." << endl << "> ";

FileName = ReadStringWithoutWhitespace();

}

if (File\_Exists(FileName)) {

do {

cout << "Файл уже существует. Выберите режим ввода: " << endl << "1 - Перезаписать файл" << endl << "2 - Записать в конец файла" << endl << "3 - Выбрать другой файл" << endl << "> ";

SaveMethod = CheckInt();

switch (SaveMethod) {

case RewriteFile:

file.open(FileName, ios::trunc);

break;

case WriteToTheEndOfTheFile:

file.open(FileName, ios::app);

file << '\n';

break;

case SelectAnotherFile:

break;

default:

cout << "Такого пункта нет в меню, попробуйте еще раз" << endl;

break;

}

} while (SaveMethod != RewriteFile && SaveMethod != WriteToTheEndOfTheFile && SaveMethod != SelectAnotherFile);

}

if (SaveMethod == RewriteFile || SaveMethod == WriteToTheEndOfTheFile || !File\_Exists(FileName)) {

break;

}

}

if (!File\_Exists(FileName)) {

do {

cout << "Файла по данному пути не найдено, хотите создать его?" << endl

<< "1 - Создать файл." << endl << "2 - Продолжить без сохранения." << endl << "> ";

FileMethod = CheckInt();

switch (FileMethod) {

case CreateFile:

file.open(FileName);

break;

case ContunueWithoutCreatingFile:

break;

default:

cout << "Такого пункта нет в меню, попробуйте еще раз" << endl;

break;

}

} while (FileMethod != CreateFile && FileMethod != ContunueWithoutCreatingFile);

}

else if (Only\_Read(FileName)) {

cout << "Ошибка доступа к файлу. Файл доступен только для чтения." << endl;

break;

}

if ((SaveMethod == RewriteFile || SaveMethod == WriteToTheEndOfTheFile || FileMethod == CreateFile) && SaveData == 1) {

file << "Исходная матрица: " << '\n';

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

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

if (matrix[i][j] == matrix[i][columns - 1]) {

file << matrix[i][j];

}

else {

file << matrix[i][j] << ' ';

}

}

if (i < strings - 1) file << '\n';

}

file << '\n' << "Сортированная матрица:" << '\n';

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

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

if (sortedmatrix[i][j] == sortedmatrix[i][columns - 1]) {

file << sortedmatrix[i][j];

}

else {

file << sortedmatrix[i][j] << ' ';

}

}

if (i < strings - 1) file << '\n';

}

file << '\n' << "Сравнительная таблица" << '\n' << "Вид сортировки\t\tКол-во сравнений\tКол-во перестановок" << '\n'

<< "Сортировка пузырьком\t" << a.GetCountOfComparison() << "\t\t\t" << a.GetCountOfPermutation() << '\n'

<< "Сортировка выбором\t" << b.GetCountOfComparison() << "\t\t\t" << b.GetCountOfPermutation() << '\n'

<< "Сортировка вставками\t" << c.GetCountOfComparison() << "\t\t\t" << c.GetCountOfPermutation() << '\n'

<< "Сортировка Шелла\t" << d.GetCountOfComparison() << "\t\t\t" << d.GetCountOfPermutation() << '\n'

<< "Сортировка быстрая\t" << e.GetCountOfComparison() << "\t\t\t" << e.GetCountOfPermutation() << '\n';

}

else if ((SaveMethod == RewriteFile || SaveMethod == WriteToTheEndOfTheFile || FileMethod == CreateFile) && SaveData == 2) {

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

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

if (matrix[i][j] == matrix[i][columns - 1]) {

file << matrix[i][j];

}

else {

file << matrix[i][j] << ' ';

}

}

if (i < strings - 1) file << '\n';

}

}

file.close();

break;

case ContinueWithoutSaving:

break;

default:

cout << "Такого пункта нет в меню, попробуйте еще раз" << endl;

break;

}

} while (SaveChoise != Save && SaveChoise != ContinueWithoutSaving);

}

void CompleteControlWork(int\*\* matrix, int strings, int columns) {

int\*\* copymatrix = nullptr;

copymatrix = new int\* [strings];

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

copymatrix[i] = new int[columns];

}

cout << endl << "Исходная матрица:" << endl;

PrintMatrix(matrix, strings, columns);

cout << endl << "Сортировка пузырьком:" << endl;

CopyMatrix(copymatrix, matrix, strings, columns);

BubbleSort a;

a.Sort(copymatrix, strings, columns);

PrintMatrix(copymatrix, strings, columns);

cout << endl << "Сортировка выбором:" << endl;

CopyMatrix(copymatrix, matrix, strings, columns);

SelectionSort b;

b.Sort(copymatrix, strings, columns);

PrintMatrix(copymatrix, strings, columns);

cout << endl << "Сортировка вставками:" << endl;

CopyMatrix(copymatrix, matrix, strings, columns);

InsertSort c;

c.Sort(copymatrix, strings, columns);

PrintMatrix(copymatrix, strings, columns);

cout << endl << "Сортировка Шелла:" << endl;

CopyMatrix(copymatrix, matrix, strings, columns);

ShellSort d;

d.Sort(copymatrix, strings, columns);

PrintMatrix(copymatrix, strings, columns);

cout << endl << "Сортировка быстрая:" << endl;

CopyMatrix(copymatrix, matrix, strings, columns);

QuickSort e;

e.Sort(copymatrix, strings, columns);

PrintMatrix(copymatrix, strings, columns);

cout << endl;

cout << "Сравнительная таблица." << endl;

cout << "Вид сортировки\t\tКол-во сравнений\tКол-во перестановок" << endl

<< "Сортировка пузырьком\t" << a.GetCountOfComparison() << "\t\t\t" << a.GetCountOfPermutation() << endl

<< "Сортировка выбором\t" << b.GetCountOfComparison() << "\t\t\t" << b.GetCountOfPermutation() << endl

<< "Сортировка вставками\t" << c.GetCountOfComparison() << "\t\t\t" << c.GetCountOfPermutation() << endl

<< "Сортировка Шелла\t" << d.GetCountOfComparison() << "\t\t\t" << d.GetCountOfPermutation() << endl

<< "Сортировка быстрая\t" << e.GetCountOfComparison() << "\t\t\t" << e.GetCountOfPermutation() << endl;

cout << endl;

SaveMatrixIntoFile(matrix, copymatrix, strings, columns, a, b, c, d, e);

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

delete[] matrix[i];

delete[] copymatrix[i];

matrix[i] = nullptr;

copymatrix[i] = nullptr;

}

delete[] matrix;

delete[] copymatrix;

matrix = nullptr;

copymatrix = nullptr;

}

void MainMenu() {

setlocale(LC\_ALL, "RU");

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

int UserChoise = 0;

do {

system("cls");

Print\_menu();

UserChoise = CheckInt();

switch (UserChoise) {

case StartProgramm:

InputMethodMenu();

break;

case StartModuleTests:

CompleteModuleTests();

break;

case StopProgramm:

break;

default:

cout << "Такого пункта нет в меню, попробуйте еще раз" << endl;

break;

}

if (UserChoise != StopProgramm) system("pause");

} while (UserChoise != StopProgramm);

}

**[---menu.cpp]**

**[ Начало программы--- ]**

**[ Начало main.cpp--- ]**

//Пономарев Константин, 425.

//cpp-файл, основной файл программы.

#include "menu.h"

using namespace std;

int main() {

MainMenu();

}

**[---main.cpp]**

**[---Конец программы]**