Министерство науки и высшего образования Российской Федерации Федеральное государственное бюджетное образовательное учреждение высшего образования

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

(СПбГТИ(ТУ))

Факультет информационных технологий и управления

Кафедра систем автоматизированного проектирования и управления Дисциплина: Программирование

Отчет по лабораторной работе №3

«Программа, упорядочивающая каждую нечётную строку матрицы по возрастанию сумм цифр в них»

Исполнитель

студентка гр. 415а Ю. Г. Науменко

(подпись)

Преподаватели И. Г. Корниенко

(подпись)

А. К. Федин

(подпись)

Санкт–Петербург

2022

# Содержание

1. Постановка задачи 3
2. Исходные данные 3
3. Особые ситуации 3
4. Математические методы и алгоритмы решения задач 3
5. Форматы представления данных 4
6. [Структура программы 5](#_TOC_250002)
7. [Описание хода выполнения лабораторной работы 7](#_TOC_250001)
8. Блок-схема алгоритма решения задачи 8
9. [Результаты работы программы 9](#_TOC_250000)
10. Исходный текст программы 10
11. **Постановка задачи**

Необходимо разработать программу для сортировки массива данных методами: пузырьковой, отбора, вставки, Шелла и быстрой сортировки. Составить сравнительную таблицу эффективности методов, в которой необходимо указать число сравнений и перестановок переменных в каждом методе сортировки. Изначальная матрица сортируется следующим образом: каждая её нечётная строка сортируется по возрастанию суммы значений цифр элементов матрицы.

Также необходимо реализовать абстрактный базовый класс ISort, содержащий чистый виртуальный метод Sort и необходимые счетчики, от которого наследовать подклассы для реализации сортировок.

1. **Исходные данные**

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.

1. **Особые ситуации**

Необходимо рассмотреть следующие особые ситуации:

* Пользователь ввел отрицательное значение количества строк или столбцов;
* Пользователь ввел некорректное значение элемента;
* Файл не расширения txt;
* В файле, предоставленном для чтения, некорректные данные;
* Файл, предоставленный для записи, уже существует;
* Пользователь ввел зарезервированное или недопустимое имя файла.

# Алгоритм решения задач

Массив данных сортируется пятью разными методами: пузырьковой, отбора, вставки, Шелла и быстрой сортировкой.

Алгоритм сортировки пузырьковым методом 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 – 1 раз или до тех пор, пока на очередном проходе не окажется, что обмены больше не нужны, что означает — массив отсортирован. При каждом проходе алгоритма по внутреннему циклу, очередной наибольший элемент массива ставится на своё место в конце массива рядом с предыдущим

«наибольшим элементом», а наименьший элемент перемещается на одну позицию к началу массива.

Первый шаг алгоритма сортировки методом отбора заключается в поиске минимального элемента массива (изначально за минимальный элемент принимается первый элемент массива). Вторым шагом производится обмен этого элемента с первой неотсортированной позицией (обмен не нужен, если минимальный элемент уже находится на данной позиции). Далее сортируются

остальные элементы, исключая уже отсортированные.

На каждом шаге алгоритма сортировки вставками выбирается один из элементов массива и помещается на нужную позицию в уже отсортированной последовательности до тех пор, пока все элементы не будут на своих местах.

На первом шаге алгоритма сортировки Шелла два элемента, расположенных друг от друга на расстоянии d равном N\2, сравниваются между собой, и в случае необходимости, меняются местами. На последующих шагах также происходят проверка и обмен, но расстояние d сокращается в два раза. Постепенно расстояние между элементами уменьшается, и на d=1 проход по массиву происходит в последний раз.

На первом шаге алгоритма быстрой сортировки выбирается опорный элемент массива – средний элемент массива. На втором шаге сравниваются все остальные элементы массива с опорным и переставляются таким образом, чтобы разбить массив на два непрерывных отрезка, следующих друг за другом: «элементы меньше опорного» и «большие». Далее для отрезков «меньших» и

«больших» значений выполняется рекурсивно та же последовательность операций, если длина отрезка больше единицы.

1. **Формат предоставления данных**

Таблица 1 – Переменные, используемые в программе

|  |  |  |
| --- | --- | --- |
| **Имя** | **Тип** | **Описание** |
| UserVector | class | Класс для инициализации массива и хранения его полей. |
| Input | string | Принимает значения, введённые пользователем. |
| userVector | UserVector | Массив для хранения данных о всех элементах. |
| UserChoice | int | Хранит выбор пользователя. |
| TestNum | int | Используется в модульных тестах для выбора теста. |
| moduleMass | VectorMatrix | Вектор для модульных тестов. |

# Таблица 2 – Классы, используемые в программе

|  |  |
| --- | --- |
| **Имя** | **Описание** |
| userMatrix | Класс для инициализации массива и хранения его полей. |
| iSort | Абстрактный базовый класс для сортировок. |
| bubbleSort | Класс для осуществления пузырьковой сортировки. |
| selectionSort | Класс для осуществления сортировки отбором. |
| insertSort | Класс для осуществления сортировки вставками. |
| shellsSort | Класс для осуществления сортировки методом Шелла. |
| quickSort | Класс для осуществления быстрой сортировки. |

# Структура программы

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

Таблица 3 – Функции, составляющие Algorithms.cpp

|  |  |
| --- | --- |
| **Имя** | **Описание** |
| is\_Number\_Check | Функция, проверяющая, является ли введённое значение числом. |
| user\_Is\_Number\_Check | Позволяет пользователю вводить только целочисленные элементы |
| File\_Exists | Обрабатывает путь для файла, введённый пользователем, в зависимости от того, существует файл или нет, позволяет работать с данными в файле или выдаёт ошибку. |
| input\_Randomly | Позволяет заполнить матрицу случайными числами. |

Продолжение таблицы 3.

|  |  |
| --- | --- |
| Only\_Read | Если введённый файл только для чтения, то не позволяет брать из него данные. |
| user\_Int\_Input | Позволяет пользователь ввести лишь одну цифру |
| Input\_From\_Console | Позволяет заполнить матрицу с консоли. |
| input\_From\_File | Позволяет заполнить матрицу из файла. |
| module\_Tests | Запускает проверку программы модульными тестами. |
| save\_User\_Data\_To\_File | Сохраняет данные введённые пользователем в файл. |

Таблица 4 – Функции, составляющие Source.cpp

|  |  |
| --- | --- |
| **Имя** | **Описание** |
| main | Основная функция программы. |

Таблица 5 – Функции, составляющие User\_Interface.cpp

|  |  |
| --- | --- |
| **Имя** | **Описание** |
| user\_Menu | Интерфейс программы. |

Source.cpp – основной модуль, осуществляющий запуск программы. После запуска выводится приветствие, информация о программе и меню со следующими вариантами выбора: ввести данные вручную или взять их из файла. При выборе ввода вручную пользователь сам заполняет информацию о каждом элементе, при взятии значения из файла пользователю предлагается ввести путь файла, из которого необходимо взять данные. После получения данных программа предлагает сохранить введённые данные. Если пользователь выберет сохранить данные, ему понадобится ввести путь, в котором будет создан текстовый документ с именем, введённым пользователем, и сохранены данные. Вне зависимости от выбора, далее пользователю будут предложены следующие варианты действий: вывести список элементов, ввести данные заново, перейти в меню модульных тестов или выйти из программы. Если пользователь выбирает вариант с модульными тестами, то проводятся тесты программы и меню выводится повторно. При выборе выйти из программы программа прекращает работу и закрывается.

# Описание хода выполнения лабораторной работы

1. В ходе лабораторной работы было создано решение в интегрированной среде разработки Microsoft Visual Studio C++ 2017. В нем был создан проект.
2. После набора текста программы выяснилось, что вывод текста на экран консольного приложения работает неправильно из- за различия кодировок консольного приложения и среды разработки. Для решения этой проблемы была использована функция setlocale(0, ""), которая обеспечивает работу приложения с символами кириллицы.
3. В процессе работы над программой возникли проблемы с корректным считыванием выбора пользователя, из-за чего появлялась возможность вводить некорректные значения. Проблема была решена после подключения библиотеки string.
4. При выборе способа сохранения данных, введённых пользователем, возникли трудности, в частности из-за невозможности по условию задания использовать библиотеку

<vector>. Проблема была решена с помощью создания динамического массива.

# Блок-схемы алгоритма решения задачи

# 

Рисунок 1 – Блок-схема алгоритма сортировки пузырьковым методом

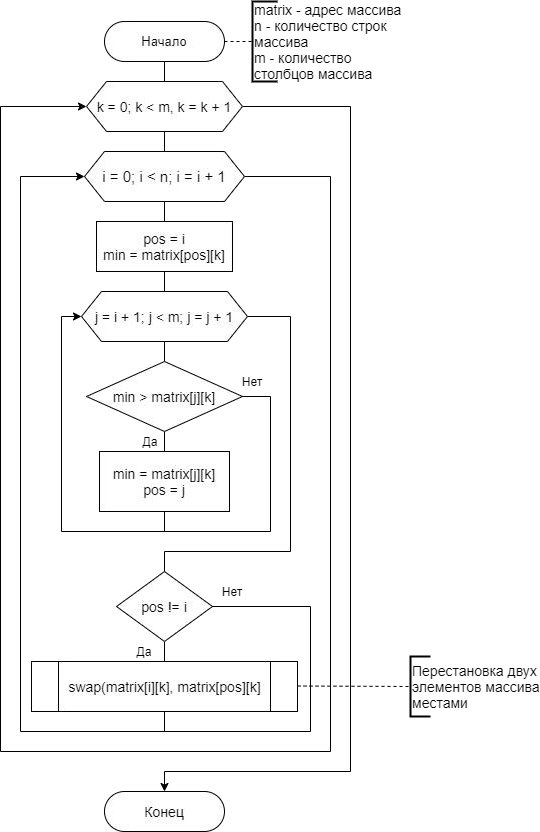


Рисунок 2 – Блок-схема алгоритма сортировки методом отбора

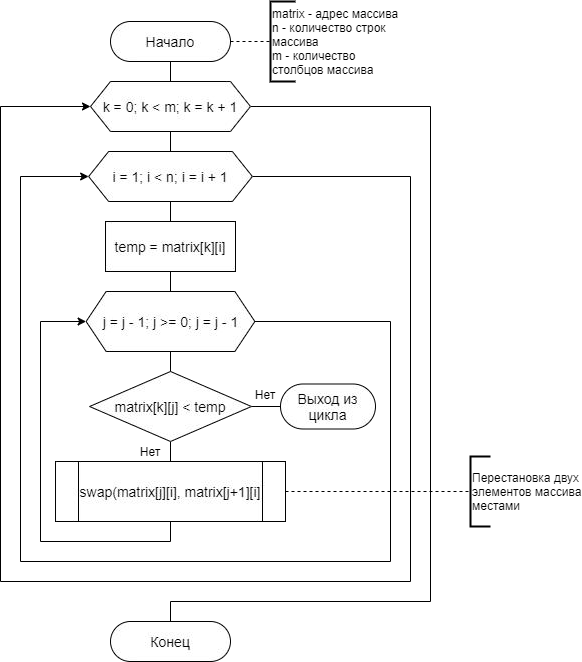


Рисунок 3 – Блок-схема алгоритма сортировка методом вставки

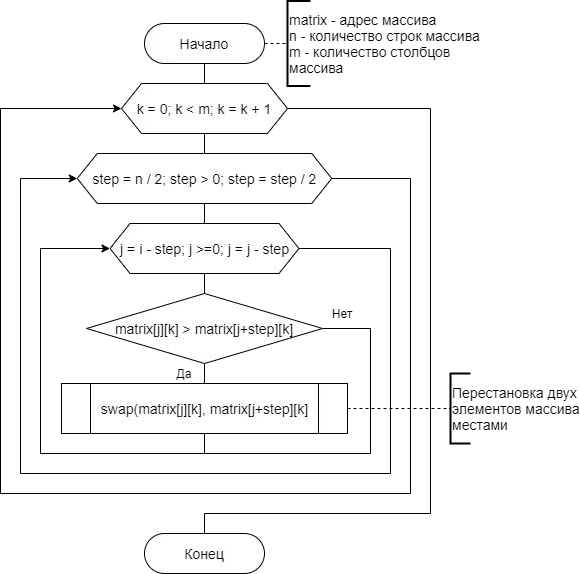


Рисунок 4 – Блок-схема алгоритма сортировки методом Шелла

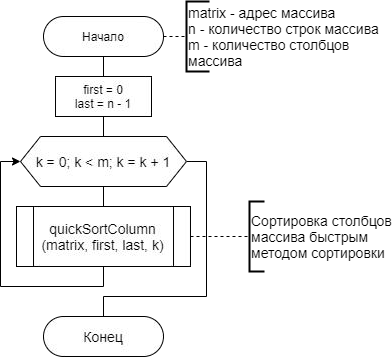


Рисунок 5 – Блок-схема алгоритма быстрого метода сортировки для двумерного массива

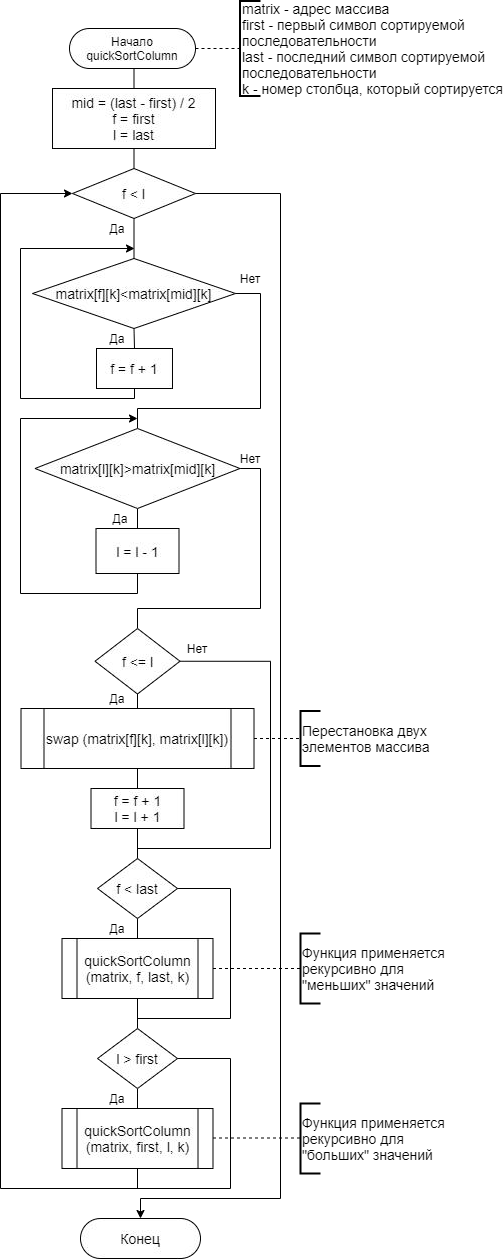


Рисунок 6 – Блок-схема алгоритма быстрой сортировки для одного столбца

# Результаты работы программы

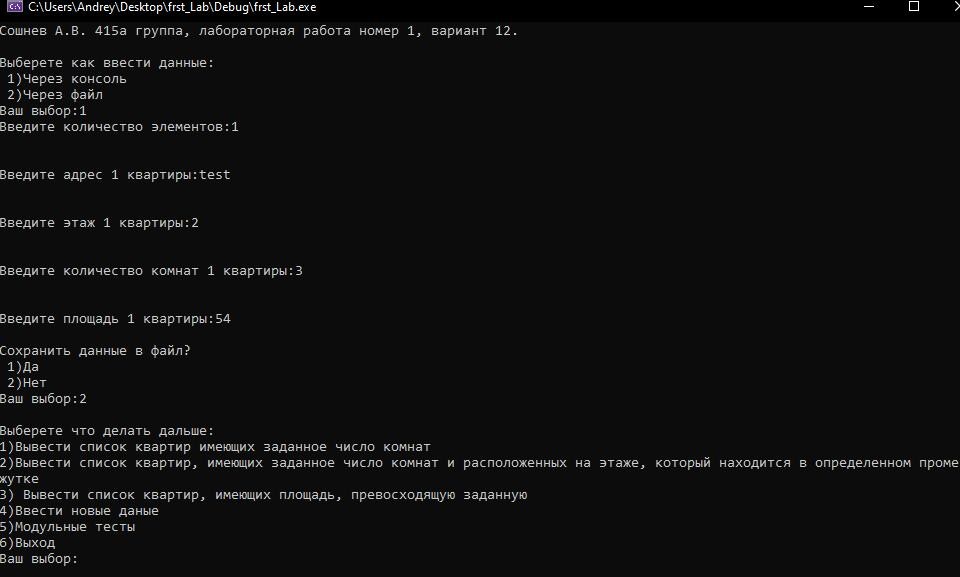


Рисунок 7 – Экранная копия приветствия и ручного ввода

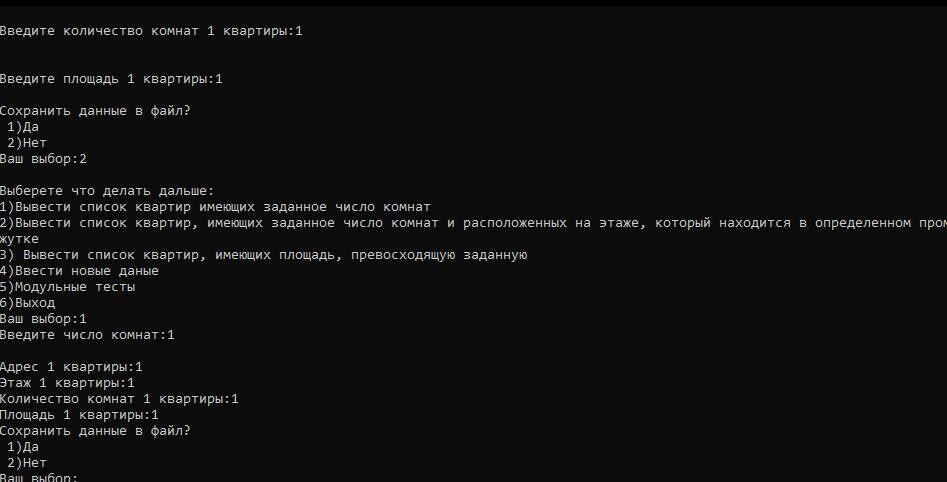


Рисунок 8 – Экранная копия сортировки списка

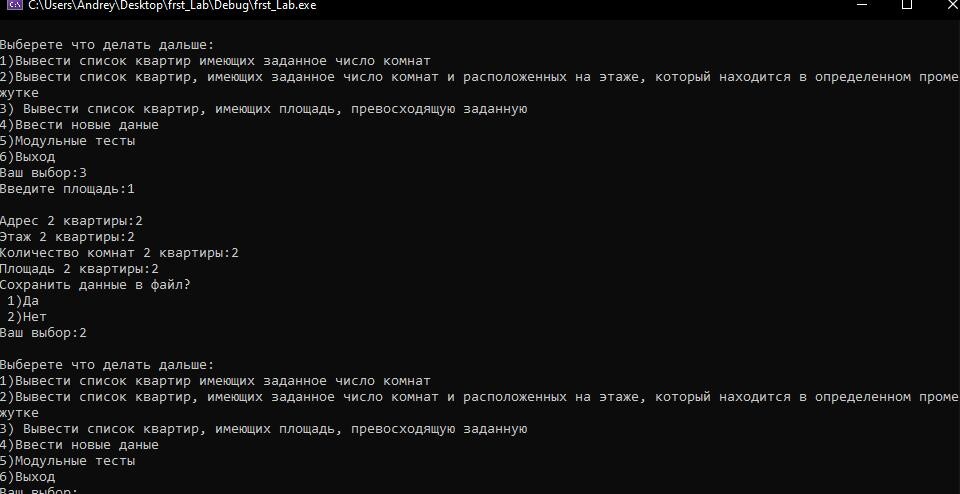


Рисунок 9 – Экранная копия сортировки списка по числу комнат

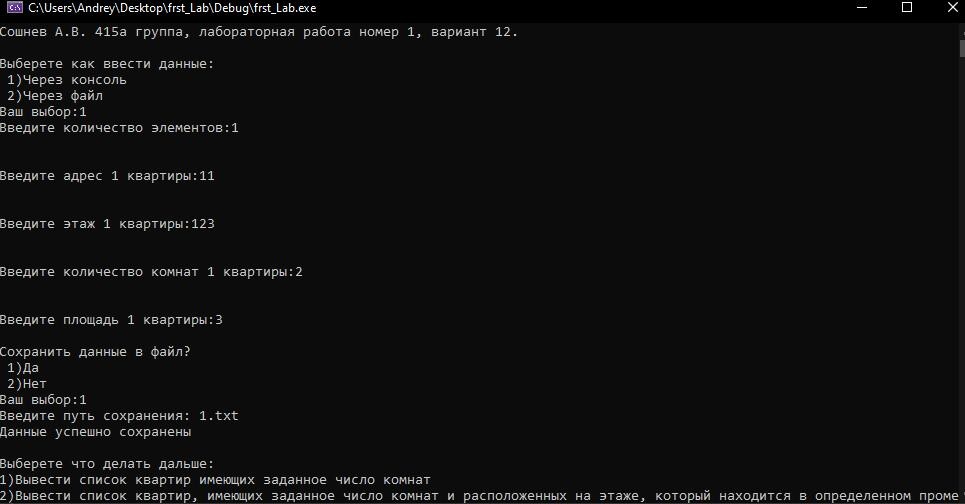


Рисунок 10 – Экранная копия сохранения данных в файл

# Исходный текст программы

**[ Начало User\_Interface.h--- ]**

#pragma once

void greeting();

**[---User\_Interface.h]**

**[ Начало Algorithms.h.--- ]**

#pragma once

#include<random>

#include<vector>

#include <conio.h>

#include <fstream>

#include<iostream>

#include <filesystem>

#include <io.h>

#include <string>

#include <time.h>

#include "User\_Interface.h"

class userMatrix {

public:

int amountOfRows=0;

int amountOfColumns=0;

void new\_Matrix(int rows, int columns) {

amountOfRows = rows;

amountOfColumns = columns;

matrVec.resize(rows);

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

matrVec[i].resize(columns);

}

}

std::vector <std::vector<int>>\* get\_Pointer() {

std::vector<std::vector<int>>\* matrPtr = &matrVec;

return matrPtr;

}

void set\_Element(int rows, int columns, int elem) {

matrVec[rows][columns] = elem;

}

int get\_Element(int rows, int columns) {

return matrVec[rows][columns];

}

std::vector<int> get\_Vector(int rows) {

return matrVec[rows];

}

void matrix\_Clear() {

std::vector<std::vector<int>>\* matrPtr = get\_Pointer();

matrPtr->clear();

}

private:

std::vector <std::vector<int>> matrVec;

};

namespace checks {

bool File\_Exists(std::string fileName);

bool Only\_Read(std::string fileName);

};

namespace input {

void input\_From\_Console(userMatrix\* matrPtr);

int input\_From\_File(userMatrix\* matrPtr);

void input\_Randomly(userMatrix\* matrPtr);

int user\_Int\_Input();

};

namespace output {

void output\_unsorted\_Matrix(userMatrix \* matrPtr);

void save\_Unsorted\_Matrix\_To\_File(userMatrix\* matrPtr);

void module\_Tests();

void save\_Results\_To\_File(userMatrix\* matrPtr, int\* compareBubblePtr, int\* compareSelectPtr, int\* compareInsertPtr, int\* compareShellPtr, int\* compareQuickPtr);

}

enum choicesInFirstMenu { manually = 1, randomly, fromFile, endProgramm };

enum yesOrNo { yes = 1, no };

enum choicesInScndMenu { outputList = 1, newData, tests, exitFromProgramm };

enum fileCatcher { error = 0, allGood };

**[---Конец Algorithms.h]**

**[Начало User\_Interface.cpp---]**

#include "Algorithms.h"

#include "Sorts.h"

#include "User\_Interface.h"

void greeting() {

setlocale(LC\_ALL, "Russian");

int userChoice;

int errorCatcher;

bool loopMenuFirst = true;

bool loopMenuScnd = true;

bool loopMenuGeneral = true;

bool loopMenuSavingData = false;

userMatrix matrix;

userMatrix\* matrixPtr = &matrix;

bubbleSort bubbleMatrix;

selectionSort selectionMatrix;

insertSort insertMatrix;

shellsSort shellMatrix;

quickSort quickMatrix;

int compareBuble[2];

int\* compareBubblePtr = compareBuble;

int compareSelect[2];

int\* compareSelectPtr = compareSelect;

int compareInsert[2];

int\* compareInsertPtr = compareInsert;

int compareShell[2];

int\* compareShellPtr = compareShell;

int compareQuick[2];

int\* compareQuickPtr = compareQuick;

std::cout << "Выполнила Науменко Юлия, 415а группа. Лабораторная работа #3, вариант #16. \nПрограмма сортирует введённую пользователем матрицу пятью сортировками: пузырьковой, быстрой, вставкой, Шелла, отбора." <<

std::endl << "Программа выводит изначальную матрицу, после матрицу, отсортированную каждой сортировкой, а также таблицу эффективности каждой сортировки. Сортируются каждая нечётная строка матрицы по возрастанию суммы цифр каждого числа в строчке."

<< std::endl << std::endl;

while (loopMenuGeneral) {

while (loopMenuFirst) {

std::cout << "Выберите способ ввода данных:" << std::endl;

std::cout << "1) Вручную; \n2) Заполнить случайными числами; \n3) Загрузить из файла; \n4) Выйти из программы.\n\n";

userChoice = input::user\_Int\_Input();

switch (userChoice) {

case manually:

input::input\_From\_Console(matrixPtr);

loopMenuFirst = false;

break;

case randomly:

input::input\_Randomly(matrixPtr);

loopMenuFirst = false;

break;

case fromFile:

errorCatcher = input::input\_From\_File(matrixPtr);

if (errorCatcher == error)

std::cout << "Ошибка в файле, попробуйте снова." << std::endl;

else loopMenuFirst = false;

break;

case endProgramm:

loopMenuFirst = false;

loopMenuScnd = false;

loopMenuGeneral = false;

break;

default:

std::cout << "Такого пункта нет, попробуйте ввести существующий пункт." << std::endl;

loopMenuFirst = true;

break;

}

}

loopMenuSavingData = true;

while (loopMenuSavingData) {

std::cout << "Хотите сохранить введённые данные? \n 1) Да; \n 2) Нет." << std::endl << std::endl;

userChoice = input::user\_Int\_Input();

switch (userChoice) {

case yes: output::save\_Unsorted\_Matrix\_To\_File(matrixPtr);

loopMenuSavingData = false;

break;

case no: loopMenuSavingData = false;

break;

default:

std::cout << "Такого пункта нет, попробуйте ввести существующий пункт." << std::endl;

break;

}

}

loopMenuScnd = true;

while (loopMenuScnd) {

std::cout << "Что дальше?\n1) Вывести отсортированные матрицы и таблицу эффективности;"

<< std::endl <<

"2) Начать заново; \n3) Пройти модульные тесты; \n4) Выйти отсюда." << std::endl;

std::cout << "Ваш выбор: ";

userChoice = input::user\_Int\_Input();

std::cout << std::endl;

switch (userChoice) {

case outputList:

if (matrix.amountOfRows <= 2) {

for (int i = 0; i < matrix.amountOfRows; i++) {

for (int j = 0; j < matrix.amountOfColumns; j++) {

std::cout << matrix.get\_Element(i, j) << " ";

}

std::cout << std::endl;

}

std::cout << "Вы ввели две или меньше строчек. Смысла в сортировке нет - выведена только ваша матрица.";

continue;

}

else {

output::output\_unsorted\_Matrix(matrixPtr);

bubbleMatrix.sort(matrixPtr, compareBubblePtr);

shellMatrix.sort(matrixPtr, compareShellPtr);

selectionMatrix.sort(matrixPtr, compareSelectPtr);

insertMatrix.sort(matrixPtr, compareInsertPtr);

quickMatrix.sort(matrixPtr, compareQuickPtr);

}

loopMenuScnd = true;

loopMenuGeneral = true;

std::cout << "Сортировка пузырьком:";

std::cout << std::endl;

std::cout << '|';

std::cout << "Количество сравнений: ";

std::cout << compareBubblePtr[0] << '|';

std::cout << "Количество перестановок: ";

std::cout << compareBubblePtr[1] << '|';

std::cout << std::endl;

std::cout << "Сортировка выбором:";

std::cout << std::endl;

std::cout << '|';

std::cout << "Количество сравнений: ";

std::cout << compareSelectPtr[0] << '|';

std::cout << "Количество перестановок: ";

std::cout << compareSelectPtr[1] << '|';

std::cout << std::endl;

std::cout << "Сортировка вставками:";

std::cout << std::endl;

std::cout << '|';

std::cout << "Количество сравнений: ";

std::cout << compareInsertPtr[0] << '|';

std::cout << "Количество перестановок: ";

std::cout << compareInsertPtr[1] << '|';

std::cout << std::endl;

std::cout << "Сортировка Шелла:";

std::cout << std::endl;

std::cout << '|';

std::cout << "Количество сравнений: ";

std::cout << compareShellPtr[0] << '|';

std::cout << "Количество перестановок: ";

std::cout << compareShellPtr[1] << '|';

std::cout << std::endl;

std::cout << "Быстрая сортировка:";

std::cout << std::endl;

std::cout << '|';

std::cout << "Количество сравнений: ";

std::cout << compareQuickPtr[0] << '|';

std::cout << "Количество перестановок: ";

std::cout << compareQuickPtr[1] << '|';

std::cout << std::endl;

std::cout << "Сохранить получившуюся после сортировок матрицу и таблицу эффективности в файл?" << std::endl << " 1)Да" << std::endl << " 2)Нет" << std::endl << "Ваш выбор:";

userChoice = input::user\_Int\_Input();

loopMenuSavingData = true;

while(loopMenuSavingData) {

switch (userChoice) {

case yes:

output::save\_Results\_To\_File(matrixPtr, compareBubblePtr, compareSelectPtr, compareInsertPtr, compareShellPtr, compareQuickPtr);

loopMenuSavingData = false;

break;

case no:

loopMenuSavingData = false;

break;

default:

std::cout << "\nТакого пункта нет, попробуйте ввести существующий пункт.\n";

break;

}

}

break;

case newData:

loopMenuFirst = true;

loopMenuScnd = false;

loopMenuGeneral = true;

break;

case tests:

output::module\_Tests();

loopMenuScnd = true;

loopMenuGeneral = true;

break;

case exitFromProgramm:

loopMenuScnd = false;

loopMenuGeneral = false;

break;

default:

std::cout << "\nТакого пункта нет, попробуйте ввести существующий пункт.\n";

break;

}

}

}

}

**[---Конец User\_Interface.cpp]**

**[Начало Source.cpp---]**

#include"user\_Interface.h"

int main() {

user\_Menu();

return 0;

}

**[---Конец Source.cpp]**

**[Начало Algorithms.cpp---]**

#include "Algorithms.h"

using namespace std;

using namespace filesystem;

namespace checks {

bool File\_Exists(std::string fileName) {

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

if (!\_access(path, 0))

return true;

else

return false;

}

bool Only\_Read(std::string fileName) {

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

if (\_access(path, 2))

return true;

else

return false;

}

int is\_Positive\_Number(int num) {

while (num < 1) {

std::cout << "Число должно быть больше нуля. Попробуйте снова: ";

num = input::user\_Int\_Input();

}

return num;

}

bool is\_Number\_Check(std::string inputData) {

bool tmp = true;

for (int i = 0; i < inputData.length(); i++) {

tmp = isdigit(inputData[i]);

}

return tmp;

}

bool is\_Digit\_Check(char inputData) {

return isdigit(inputData);

}

}

namespace input {

int user\_Int\_Input() {

int userInput;

std::cin >> userInput;

while (std::cin.fail()) {

std::cout << "Некорректный ввод." << std::endl;

std::cin.clear();

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

std::cin >> userInput;

}

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

return userInput;

}

void input\_From\_Console(userMatrix \*matrPtr) {

int amountOfRows;

int amountOfColumns;

int tmp;

std::cout << "\nВведите количество строк: ";

amountOfRows = user\_Int\_Input();

checks::is\_Positive\_Number(amountOfRows);

std::cout << "\nВведите количество столбцов: ";

amountOfColumns = user\_Int\_Input();

checks::is\_Positive\_Number(amountOfColumns);

matrPtr->new\_Matrix(amountOfRows, amountOfColumns);

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

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

std::cout << "\nВведите значение в [" << i+1 << "][" << j+1 << "] элементе: ";

tmp = user\_Int\_Input();

while (tmp < 0) {

std::cout << "\nЧисло не может быть меньше нуля. Попробуйте снова: ";

tmp = user\_Int\_Input();

}

matrPtr->set\_Element(i, j, tmp);

}

}

}

void input\_Randomly(userMatrix\* matrPtr) {

int amountOfRows;

int amountOfColumns;

int randomNum;

srand((unsigned)time(0));

std::cout << "\nВведите количество строк: ";

amountOfRows = user\_Int\_Input();

checks::is\_Positive\_Number(amountOfRows);

std::cout << "\nВведите количество столбцов: ";

amountOfColumns = user\_Int\_Input();

checks::is\_Positive\_Number(amountOfColumns);

matrPtr->new\_Matrix(amountOfRows, amountOfColumns);

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

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

randomNum = 1 + rand() % 20;

matrPtr->set\_Element(i, j, randomNum);

}

}

}

int input\_From\_File(userMatrix \*matrPtr) {

string filePath = "";

string tmp = "";

int amountOfRows = 0;

int amountOfColumns = 0;

int elem = 0;

bool isDataLoaded = false;

do {

cout << "Введите путь: ";

getline(cin, filePath);

if (!ifstream(filePath)) {

cout << "\nТакого файла не существует, попробуйте снова.\n\n";

continue;

}

error\_code ec1;

if (!is\_regular\_file(filePath, ec1)) {

cout << "\nАдрес содержит недопустимые значения, попробуйте снова.\n\n";

continue;

}

ifstream newFileMatrix(filePath);

if (!newFileMatrix) {

cout << "\nЭтот файл загрузить нельзя, попробуйте снова.\n\n";

newFileMatrix.close();

continue;

}

newFileMatrix >> amountOfRows;

newFileMatrix >> amountOfColumns;

/\*try {

matrPtr->new\_Matrix(amountOfRows, amountOfColumns);

}

catch (...) {

return error;

}\*/

matrPtr->new\_Matrix(amountOfRows, amountOfColumns);

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

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

if (!(newFileMatrix >> elem)) {

cout << "Данные некорректны, попробуйте снова.\n\n";

newFileMatrix.close();

break;

}

matrPtr->set\_Element(i, j, elem);

/\*tmp.clear();

newFileMatrix >> tmp;

if (tmp.length() == 0) {

return error;

}

if (!checks::is\_Number\_Check(tmp)) {

return error;

}

elem = stoi(tmp);

matrPtr->set\_Element(i, j, elem);\*/

}

}

newFileMatrix.close();

isDataLoaded = true;

} while (!isDataLoaded);

return allGood;

}

}

namespace output {

void module\_Tests() {

int testNum = 0;

int tmp;

userMatrix moduleMass;

moduleMass.new\_Matrix(5, 5);

try {

for (int i = 0; i < 5; ++i)

{

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

tmp = rand() % 10;

moduleMass.set\_Element(i, j, tmp);

}

}

std::cout << "\n\nТестирование прошло успешно. ";

}

catch (...) {

std::cout << "Тест номер " << testNum << std::endl;

std::cout << "Набор данных - " << std::endl;

for (int i = 0; i < 5; i++)

{

std::cout << std::endl;

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

std::cout << moduleMass.get\_Element(i, j) << std::endl;

}

}

std::cout << "Ожидаемый результат - заполнение массива структур в классе данными." << std::endl;

std::cout << "Полученный результат - критическая ошибка. " << std::endl;

}

}

void save\_Unsorted\_Matrix\_To\_File(userMatrix\* matrPtr) {

int userChoice = 0;

string filePath = "";

bool isDataSaved = false;

do {

cout << "Введите путь сохранения: ";

getline(cin, filePath);

while (checks::File\_Exists(filePath)) {

cout << "Файл уже существует." << endl << "1) Перезаписать файл." << endl << "2) Ввести другой путь." << endl;

if (checks::Only\_Read(filePath)) {

cout << "Файл доступен только для чтения, автоматический переход к пункту 2." << endl;

userChoice = no;

}

else userChoice = input::user\_Int\_Input();

if (userChoice == yes) {

cout << "Файл успешно перезаписан." << endl;

break;

}

if (userChoice == no) {

cout << "Введите путь сохранения: ";

cin >> filePath;

}

}

ofstream userFile(filePath, ofstream::app);

error\_code ec;

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

cout << "В имени пути есть недопустимые значения, попробуйте снова.\n\n";

continue;

}

if (!userFile) {

cout << "Запись запрещена, попробуйте снова.\n\n";

userFile.close();

continue;

}

userFile.close();

userFile.open(filePath, ofstream::trunc);

userFile << matrPtr->amountOfRows << " ";

userFile << matrPtr->amountOfColumns << " ";

for (int i = 0; i < matrPtr->amountOfRows; i++) {

for (int j = 0; j < matrPtr->amountOfColumns; j++) {

userFile << matrPtr->get\_Element(i, j) << " ";

}

}

userFile.close();

isDataSaved = true;

} while (!isDataSaved);

}

void output\_unsorted\_Matrix(userMatrix\* matrPtr) {

std::cout << "Ваша матрица:\n\n";

for (int i = 0; i < matrPtr->amountOfRows; i++) {

for (int j = 0; j < matrPtr->amountOfColumns; j++) {

std::cout << matrPtr->get\_Element(i, j) << " ";

}

std::cout << std::endl << std::endl;

}

}

void save\_Results\_To\_File(userMatrix\* matrPtr, int\* compareBubblePtr, int\* compareSelectPtr, int\* compareInsertPtr, int\* compareShellPtr, int\* compareQuickPtr) {

int amountOfRows = matrPtr->amountOfRows;

int amountOfColumns = matrPtr->amountOfColumns;

int digitsSumInRow = 0;

vector<vector<int>> bubbleMatrix;

vector <vector<int>> sumAndNumRows;

vector<int> tmpVec;

tmpVec.push\_back(0);

tmpVec.push\_back(0);

bubbleMatrix.resize(amountOfRows);

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

{

bubbleMatrix[i].resize(amountOfColumns);

}

for (int i = 0; i < matrPtr->amountOfRows; i++) {

for (int j = 0; j < matrPtr->amountOfColumns; j++) {

bubbleMatrix[i][j] = matrPtr->get\_Element(i, j);

}

}

for (int i = 0; i < matrPtr->amountOfRows; i++) {

if (i % 2 == 0) {

digitsSumInRow = 0;

for (int j = 0; j < matrPtr->amountOfColumns; j++) {

int digitsSumInNum = 0;

int matrElem = matrPtr->get\_Element(i, j);

while (matrElem != 0) {

digitsSumInNum += matrElem % 10;

matrElem /= 10;

}

digitsSumInRow += digitsSumInNum;

}

tmpVec[0] = digitsSumInRow;

tmpVec[1] = i;

sumAndNumRows.push\_back(tmpVec);

}

}

for (int i = 0; i < sumAndNumRows.size(); i++) {

for (unsigned int j = 0; j < sumAndNumRows.size() - 1; j++) {

if (sumAndNumRows[j].at(0) > sumAndNumRows[j + 1].at(0)) {

swap(sumAndNumRows[j], sumAndNumRows[j + 1]);

}

}

}

int row = 0;

int col = 0;

while (row < bubbleMatrix.size()) {

bubbleMatrix[row] = matrPtr->get\_Vector(sumAndNumRows[col].at(1));

row += 2;

col += 1;

}

int userChoice = 0;

string filePath = "";

bool isDataSaved = false;

do {

cout << "Введите путь сохранения: ";

getline(cin, filePath);

while (checks::File\_Exists(filePath)) {

cout << "Файл уже существует." << endl << "1) Перезаписать файл." << endl << "2) Ввести другой путь." << endl;

if (checks::Only\_Read(filePath)) {

cout << "Файл доступен только для чтения, автоматический переход к пункту 2." << endl;

userChoice = no;

}

else userChoice = input::user\_Int\_Input();

if (userChoice == yes) {

cout << "Файл успешно перезаписан." << endl;

break;

}

if (userChoice == no) {

cout << "Введите путь сохранения: ";

cin >> filePath;

}

}

ofstream userFile(filePath, ofstream::app);

error\_code ec;

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

cout << "В имени пути есть недопустимые значения, попробуйте снова.\n\n";

continue;

}

if (!userFile) {

cout << "Запись запрещена, попробуйте снова.\n\n";

userFile.close();

continue;

}

userFile.close();

userFile.open(filePath, ofstream::trunc);

userFile << matrPtr->amountOfRows << " ";

userFile << matrPtr->amountOfColumns << " ";

for (int i = 0; i < matrPtr->amountOfRows; i++) {

for (int j = 0; j < matrPtr->amountOfColumns; j++) {

userFile << bubbleMatrix[i][j] << " ";

}

}

userFile << '\n';

userFile << '|';

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

userFile << compareBubblePtr[i] << '|';

}

userFile << '\n';

userFile << '|';

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

userFile << compareSelectPtr[i] << '|';

}

userFile << '\n';

userFile << '|';

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

userFile << compareInsertPtr[i] << '|';

}

userFile << '\n';

userFile << '|';

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

userFile << compareShellPtr[i] << '|';

}

userFile << '\n';

userFile << '|';

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

userFile << compareQuickPtr[i] << '|';

}

userFile << '\n';

//userFile.close();

isDataSaved = true;

} while (!isDataSaved);

}

}

**[---Конец Algorithms.cpp]**

**[Начало Sorts.h---]**

#pragma once

#include "Algorithms.h"

class iSort {

public:

virtual void sort(userMatrix\* matrPtr, int\* comparisons) = 0;

};

class bubbleSort : public iSort {

public: void sort(userMatrix\* matrPtr, int\* comparisons) override;

};

class quickSort : public iSort {

public:

void sort(userMatrix\* matrPtr, int\* comparisons) override;

void quick\_Rows\_Sort(std::vector<std::vector <int>> &array, size\_t first, size\_t last, int &comparisonsCounter, int &shiftsCounter);

};

class insertSort : public iSort {

public: void sort(userMatrix\* matrPtr, int\* comparisons) override;

};

class selectionSort : public iSort {

public: void sort(userMatrix\* matrPtr, int\* comparisons) override;

};

class shellsSort : public iSort {

public: void sort(userMatrix\* matrPtr, int\* comparisons) override;

};

**[---Конец Sorts.h]**

**[Начало Sorts.cpp---]**

#include "Sorts.h"

using namespace std;

void bubbleSort::sort(userMatrix \*matrPtr, int\* comparisons) {

int comparisonsCounter = 0;

int shiftsCounter = 0;

int digitsSumInRow = 0;

int amountOfRows = matrPtr->amountOfRows;

int amountOfColumns = matrPtr->amountOfColumns;

vector<vector<int>> bubbleMatrix;

vector <vector<int>> sumAndNumRows;

vector<int> tmpVec; tmpVec.push\_back(0); tmpVec.push\_back(0);

bubbleMatrix.resize(amountOfRows);

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

{

bubbleMatrix[i].resize(amountOfColumns);

}

for (int i = 0; i < matrPtr->amountOfRows; i++) {

for (int j = 0; j < matrPtr->amountOfColumns; j++) {

bubbleMatrix[i][j] = matrPtr->get\_Element(i, j);

}

}

for (int i = 0; i < matrPtr->amountOfRows; i++) {

if (i % 2 == 0) {

digitsSumInRow = 0;

for (int j = 0; j < matrPtr->amountOfColumns; j++) {

int digitsSumInNum = 0;

int matrElem = matrPtr->get\_Element(i, j);

while (matrElem != 0) {

digitsSumInNum += matrElem % 10;

matrElem /= 10;

}

digitsSumInRow += digitsSumInNum;

}

tmpVec[0] = digitsSumInRow;

tmpVec[1] = i;

sumAndNumRows.push\_back(tmpVec);

}

}

bool isWorking = true;

while (isWorking) {

isWorking = false;

for (int i = 0; i < sumAndNumRows.size()-1; i++) {

if (sumAndNumRows[i].at(0) > sumAndNumRows[i + 1].at(0)) {

swap(sumAndNumRows[i], sumAndNumRows[i + 1]);

isWorking = true;

shiftsCounter++;

}

comparisonsCounter++;

}

}

int row = 0;

int col = 0;

while(row < bubbleMatrix.size()) {

bubbleMatrix[row] = matrPtr->get\_Vector(sumAndNumRows[col].at(1));

row += 2;

col += 1;

}

std::cout << "Матрица после пузырьковой сортировки:\n\n";

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

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

std::cout << bubbleMatrix[i][j] << " ";

}

std::cout << endl << endl;

}

comparisons[0] = comparisonsCounter;

comparisons[1] = shiftsCounter;

}

void quickSort::quick\_Rows\_Sort(vector <vector <int>> &array, size\_t first, size\_t last, int& comparisonsCounter, int& shiftsCounter) {

int mid = static\_cast<int>((first + last) / 2);

int f = static\_cast<int>(first);

int l = static\_cast<int>(last);

do {

comparisonsCounter++;

while (array[f].at(0) < array[mid].at(0))

{

f++;

}

while (array[l].at(0) > array[mid].at(0))

{

l--;

}

if (f <= l) {

swap(array[f], array[l]);

shiftsCounter++;

f++;

l--;

}

} while (f < l);

if (first < l)

quick\_Rows\_Sort(array, first, l, comparisonsCounter, shiftsCounter);

if (f < last)

quick\_Rows\_Sort(array, f, last, comparisonsCounter, shiftsCounter);

}

void quickSort::sort(userMatrix\* matrPtr, int\* comparisons) {

int comparisonsCounter = 0;

int shiftsCounter = 0;

int digitsSumInRow = 0;

int rows = matrPtr->amountOfRows;

int columns = matrPtr->amountOfColumns;

vector<vector <int>> quickMatrix;

vector <vector<int>> sumAndNumRows;

vector<int> tmpVec; tmpVec.push\_back(0); tmpVec.push\_back(0);

quickMatrix.resize(rows);

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

quickMatrix[i].resize(columns);

}

for (int i = 0; i < matrPtr->amountOfRows; i++) {

for (int j = 0; j < matrPtr->amountOfColumns; j++) {

quickMatrix[i][j] = matrPtr->get\_Element(i, j);

}

}

for (int i = 0; i < matrPtr->amountOfRows; i++) {

if (i % 2 == 0) {

digitsSumInRow = 0;

for (int j = 0; j < matrPtr->amountOfColumns; j++) {

int digitsSumInNum = 0;

int matrElem = matrPtr->get\_Element(i, j);

while (matrElem != 0) {

digitsSumInNum += matrElem % 10;

matrElem /= 10;

}

digitsSumInRow += digitsSumInNum;

}

tmpVec[0] = digitsSumInRow;

tmpVec[1] = i;

sumAndNumRows.push\_back(tmpVec);

}

}

size\_t first = 0;

size\_t last = sumAndNumRows.size() - 1;

quick\_Rows\_Sort(sumAndNumRows, first, last, comparisonsCounter, shiftsCounter);

int row = 0;

int col = 0;

while (row < quickMatrix.size()) {

quickMatrix[row] = matrPtr->get\_Vector(sumAndNumRows[col].at(1));

row += 2;

col += 1;

}

std::cout << "Матрица после быстрой сортировки:\n";

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

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

std::cout << quickMatrix[i][j] << " ";

}

std::cout << endl << endl;

}

comparisons[0] = comparisonsCounter;

comparisons[1] = shiftsCounter;

};

void insertSort::sort(userMatrix\* matrPtr, int\* comparisons) {

int comparisonsCounter = 0;

int shiftsCounter = 0;

int digitsSumInRow = 0;

int rows = matrPtr->amountOfRows;

int columns = matrPtr->amountOfColumns;

vector <vector <int>> insertMatrix;

vector <vector<int>> sumAndNumRows;

vector <int> tmpVec; tmpVec.push\_back(0); tmpVec.push\_back(0);

insertMatrix.resize(rows);

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

insertMatrix[i].resize(columns);

}

for (int i = 0; i < matrPtr->amountOfRows; i++) {

for (int j = 0; j < matrPtr->amountOfColumns; j++) {

insertMatrix[i][j] = matrPtr->get\_Element(i, j);

}

}

for (int i = 0; i < matrPtr->amountOfRows; i++) {

if (i % 2 == 0) {

digitsSumInRow = 0;

for (int j = 0; j < matrPtr->amountOfColumns; j++) {

int digitsSumInNum = 0;

int matrElem = matrPtr->get\_Element(i, j);

while (matrElem != 0) {

digitsSumInNum += matrElem % 10;

matrElem /= 10;

}

digitsSumInRow += digitsSumInNum;

}

tmpVec[0] = digitsSumInRow;

tmpVec[1] = i;

sumAndNumRows.push\_back(tmpVec);

}

}

for ( int i = 1; i < sumAndNumRows.size(); i++) {

for (int j = i; j > 0; j--) {

comparisonsCounter++;

if (sumAndNumRows[j - 1].at(0) > sumAndNumRows[j].at(0)) {

swap(sumAndNumRows[j - 1], sumAndNumRows[j]);

shiftsCounter++;

}

}

}

int row = 0;

int col = 0;

while (row < insertMatrix.size()) {

insertMatrix[row] = matrPtr->get\_Vector(sumAndNumRows[col].at(1));

row += 2;

col += 1;

}

std::cout << "Матрица после сортировки вставками:\n";

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

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

std::cout << insertMatrix[i][j] << " ";

}

std::cout << endl << endl;

}

comparisons[0] = comparisonsCounter;

comparisons[1] = shiftsCounter;

}

void selectionSort::sort(userMatrix\* matrPtr, int\* comparisons) {

int comparisonsCounter = 0;

int shiftsCounter = 0;

int digitsSumInRow = 0;

int rows = matrPtr->amountOfRows;

int columns = matrPtr->amountOfColumns;

vector <vector <int>> selectionMatrix;

vector <vector<int>> sumAndNumRows;

vector <int> tmpVec; tmpVec.push\_back(0); tmpVec.push\_back(0);

selectionMatrix.resize(rows);

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

selectionMatrix[i].resize(columns);

}

for (int i = 0; i < matrPtr->amountOfRows; i++) {

for (int j = 0; j < matrPtr->amountOfColumns; j++) {

selectionMatrix[i][j] = matrPtr->get\_Element(i, j);

}

}

for (int i = 0; i < matrPtr->amountOfRows; i++) {

if (i % 2 == 0) {

digitsSumInRow = 0;

for (int j = 0; j < matrPtr->amountOfColumns; j++) {

int digitsSumInNum = 0;

int matrElem = matrPtr->get\_Element(i, j);

while (matrElem != 0) {

digitsSumInNum += matrElem % 10;

matrElem /= 10;

}

digitsSumInRow += digitsSumInNum;

}

tmpVec[0] = digitsSumInRow;

tmpVec[1] = i;

sumAndNumRows.push\_back(tmpVec);

}

}

//сама сортировка

for (int i = 0; i < sumAndNumRows.size() - 1; i++) {

int min = i;

for (int j = i + 1; j < sumAndNumRows.size(); j++) {

comparisonsCounter++;

if (sumAndNumRows[j].at(0) < sumAndNumRows[min].at(0)) {

min = j;

}

}

swap(sumAndNumRows[i], sumAndNumRows[min]);

shiftsCounter++;

}

int row = 0;

int col = 0;

while (row < selectionMatrix.size()) {

selectionMatrix[row] = matrPtr->get\_Vector(sumAndNumRows[col].at(1));

row += 2;

col += 1;

}

std::cout << "Матрица после сортировки отбором:\n";

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

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

std::cout << selectionMatrix[i][j] << " ";

}

std::cout << endl << endl;

}

comparisons[0] = comparisonsCounter;

comparisons[1] = shiftsCounter;

}

void shellsSort::sort(userMatrix\* matrPtr, int\* comparisons) {

int comparisonsCounter = 0;

int shiftsCounter = 0;

int digitsSumInRow = 0;

int rows = matrPtr->amountOfRows;

int columns = matrPtr->amountOfColumns;

vector <vector <int>> shellsMatrix;

vector <vector<int>> sumAndNumRows;

vector <int> tmpVec; tmpVec.push\_back(0); tmpVec.push\_back(0);

shellsMatrix.resize(rows);

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

shellsMatrix[i].resize(columns);

}

for (int i = 0; i < matrPtr->amountOfRows; i++) {

for (int j = 0; j < matrPtr->amountOfColumns; j++) {

shellsMatrix[i][j] = matrPtr->get\_Element(i, j);

}

}

for (int i = 0; i < matrPtr->amountOfRows; i++) {

if (i % 2 == 0) {

digitsSumInRow = 0;

for (int j = 0; j < matrPtr->amountOfColumns; j++) {

int digitsSumInNum = 0;

int matrElem = matrPtr->get\_Element(i, j);

while (matrElem != 0) {

digitsSumInNum += matrElem % 10;

matrElem /= 10;

}

digitsSumInRow += digitsSumInNum;

}

tmpVec[0] = digitsSumInRow;

tmpVec[1] = i;

sumAndNumRows.push\_back(tmpVec);

}

}

size\_t step = sumAndNumRows.size();

step /= 2;

while (step > 0) {

for (int i = 0; i < sumAndNumRows.size() - step; i++) {

int j = i;

while ((j >= 0) && (sumAndNumRows[j].at(0) > sumAndNumRows[j + step].at(0)) ) {

swap(sumAndNumRows[j], sumAndNumRows[j + step]);

shiftsCounter++;

j--;

}

comparisonsCounter++;

}

step /= 2;

}

int row = 0;

int col = 0;

while (row < shellsMatrix.size()) {

shellsMatrix[row] = matrPtr->get\_Vector(sumAndNumRows[col].at(1));

row += 2;

col += 1;

}

std::cout << "Матрица после сортировки методом Шелла:\n";

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

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

std::cout << shellsMatrix[i][j] << " ";

}

std::cout << endl << endl;

}

comparisons[0] = comparisonsCounter;

comparisons[1] = shiftsCounter;

}

**[---Конец Sorts.cpp]**