УО «Белорусский государственный университет информатики и радиоэлектроники»

Кафедра ПОИТ

Отчет по лабораторной работе №3.3

по предмету «Основы алгоритмизации и программирования»

Вариант 5

Выполнил:

Бражалович А. И.

Гр. 351005

Проверил:

Данилова Г. В.

Минск 2023

**Задание:**

Расставить строки данной матрицы в порядке возрастания элементов побочной диагонали.

**Код программы Delphi:**

Program Lab3\_3;

Uses

System.SysUtils;

Type

TMatrix = Array Of Array Of Integer;

TArr = Array Of Integer;

ERRORS\_LIST = (CORRECT, RANGE\_ERR, NUM\_ERR, NOT\_TXT, NOT\_EXIST, NOT\_READABLE, NOT\_WRITEABLE, ORDER\_ERR, CHOICE\_ERR, FILE\_EMPTY);

Const

MIN\_MATRIX = 2;

MAX\_MATRIX = 10;

MIN\_ELEMENT = -100000;

MAX\_ELEMENT = 100000;

FILE\_CHOICE = 1;

CONSOLE\_CHOICE = 2;

ERRORS: Array [ERRORS\_LIST] Of String = ('', 'Значение не попадает в диапазон!', 'Проверьте корректность ввода данных!', 'Расширение не txt!', 'Проверьте корректность ввода пути к файлу!', 'Файл закрыт для чтения!', 'Файл закрыт для записи!', 'Значения порядков не равны!', 'Проверьте корректность выбора!', 'Файл пуст!');

Procedure PrintTask();

Begin

WriteLn('Данная программа располагает строки матрицы по возрастанию элементов побочной диагонали исходной матрицы', #13#10);

End;

Function CheckArea(Num: Integer; MIN, MAX: Integer) : ERRORS\_LIST;

Var

Errors: ERRORS\_LIST;

Begin

Errors := CORRECT;

If (Num < MIN) Or (Num > MAX) Then

Errors := RANGE\_ERR;

CheckArea := Errors;

End;

Procedure PrintError(Error: ERRORS\_LIST);

Begin

WriteLn(ERRORS[Error], #13#10'Повторите попытку: ');

End;

Function CheckNum(MIN, MAX: Integer) : Integer;

Var

Errors: ERRORS\_LIST;

Num: Integer;

Begin

Repeat

Errors := CORRECT;

Try

Readln(Num);

Except

Errors := CHOICE\_ERR;

End;

If Errors = CORRECT Then

Errors := CheckArea(Num, MIN, MAX);

If Errors <> CORRECT Then

PrintError(Errors);

Until Errors = CORRECT;

CheckNum := Num;

End;

Function CheckInOut() : Boolean;

Var

Num: Integer;

Choose: Boolean;

Begin

Choose := False;

Num := CheckNum(FILE\_CHOICE, CONSOLE\_CHOICE);

If Num = 1 Then

Choose := True;

CheckInOut := Choose;

End;

Function ChooseFileInput() : Boolean;

Var

Choose: Boolean;

Begin

WriteLn('Вы хотите вводить матрицу через файл? (Да - ', 1, ' / Нет - ', 2, ')');

Choose := CheckInOut();

ChooseFileInput := Choose;

End;

Function IsReadable (Var F: TextFile) : ERRORS\_LIST;

Var

Errors: ERRORS\_LIST;

Begin

Errors := CORRECT;

Try

Try

Reset(F);

Finally

CloseFile(F);

End;

Except

Errors := NOT\_READABLE;

End;

IsReadable := Errors;

End;

Procedure FileReading(Var F: TextFile);

Var

Errors: ERRORS\_LIST;

PathToFile: String;

Begin

Repeat

Errors := CORRECT;

Write('Введите путь к файлу с расширением .txt: ');

Readln(PathToFile);

If ExtractFileExt(PathToFile) <> '.txt' Then

Errors := NOT\_TXT;

If Not FileExists(PathToFile) And (Errors = CORRECT) Then

Errors := NOT\_EXIST;

If EOF(F) Then

Errors := FILE\_EMPTY;

If Errors = CORRECT Then

AssignFile(F, PathToFile);

If (Errors = CORRECT) And (IsReadable(F) <> CORRECT) Then

Errors := NOT\_READABLE;

If Errors <> CORRECT Then

PrintError(Errors);

Until Errors = CORRECT;

End;

Function FileWriting(Var F: TextFile) : String;

Var

PathToFile: String;

Errors: ERRORS\_LIST;

Begin

Repeat

Errors := CORRECT;

Write('Введите путь к файлу с расширением .txt: ');

Readln(PathToFile);

If ExtractFileExt(PathToFile) <> '.txt' Then

Errors := NOT\_TXT;

If Not FileExists(PathToFile) And (Errors = CORRECT) Then

Errors := NOT\_EXIST;

AssignFile(F, PathToFile);

If (ERRORS = CORRECT) And (FileIsReadOnly(PathToFile)) Then

Errors := NOT\_WRITEABLE;

If Errors <> CORRECT Then

PrintError(Errors);

Until Errors = CORRECT;

FileWriting := PathTofile;

End;

Function ReadOrder(Var F: TextFile; Num: Integer): Integer;

Var

Buf, Order, Rows, Cols: Integer;

Errors: ERRORS\_LIST;

Begin

Order := 0;

Errors := CORRECT;

If Num = 2 Then

Begin

Write('Введите порядок матрицы [', MIN\_MATRIX, '; ', MAX\_MATRIX, ']: ');

Order := CheckNum(MIN\_MATRIX, MAX\_MATRIX);

End

Else

Begin

Errors := CORRECT;

Rows := 1;

Cols := 0;

Buf := 0;

Reset(F);

Readln(F, Order);

If CheckArea(Order, MIN\_MATRIX, MAX\_MATRIX) <> CORRECT Then

Begin

Errors := ORDER\_ERR;

CloseFile(F);

End

Else

Begin

While (ERRORS = CORRECT) And (Not EOF(F)) Do

Begin

Cols := 0;

While (ERRORS = CORRECT) And (Not EOLN(F)) Do

Begin

Read(F, Buf);

Inc(Cols);

End;

If Errors = CORRECT Then

Begin

Readln(F, Buf);

Inc(Rows);

If Cols = Order Then

Errors := CORRECT;

End;

End;

CloseFile(F);

If Errors = CORRECT Then

Begin

Errors := CheckArea(Rows, MIN\_MATRIX, MAX\_MATRIX);

Errors := CheckArea(Cols, MIN\_MATRIX, MAX\_MATRIX);

End;

If (Errors = CORRECT) And ((Rows Or Cols) <> Order) Then

Errors := ORDER\_ERR;

End;

End;

If Errors <> CORRECT Then

PrintError(Errors);

If Errors = CORRECT Then

ReadOrder := Order;

End;

Procedure ReadMatrix(Var Matrix: TMatrix; Var Order: Integer);

Var

RF: TextFile;

Errors: ERRORS\_LIST;

IsCorrect, FromFile: Boolean;

OrderCheck: Integer;

Row, Col: Integer;

Begin

OrderCheck := 0;

FromFile := ChooseFileInput();

If FromFile Then

Begin

Repeat

IsCorrect := True;

FileReading(RF);

Order := ReadOrder(RF,1);

OrderCheck := Order;

If OrderCheck = 0 Then

IsCorrect := False;

Reset(RF);

Readln(RF);

until (IsCorrect);

End

Else

Order := ReadOrder(RF, 2);

SetLength(Matrix, Order, Order);

For Row := Low(Matrix) To High(Matrix) Do

Begin

For Col := Low(Matrix[Row]) To High(Matrix[Row]) Do

Begin

If FromFile Then

Begin

Read(RF, Matrix[Row][Col]);

Errors := CheckArea(Matrix[Row][Col], MIN\_ELEMENT, MAX\_ELEMENT);

End

Else

Begin

Write('Введите в ', (Row + 1), ' строке ', (Col + 1), ' столбце элемент [', MIN\_ELEMENT, ' : ', MAX\_ELEMENT, ']: ');

Matrix[Row][Col] := CheckNum(MIN\_ELEMENT, MAX\_ELEMENT);

End;

End;

If FromFile Then

Readln(RF);

End;

If FromFile Then

CloseFile(RF);

End;

Procedure SwapMatrixColumns(DiagonalJ: TMatrix; Col1, Col2: Integer);

Var

I, Temp: Integer;

begin

For I := 0 To High(DiagonalJ) Do

Begin

Temp := DiagonalJ[I][Col1];

DiagonalJ[I][Col1] := DiagonalJ[I][Col2];

DiagonalJ[I][Col2] := Temp;

End;

End;

Function CreateMatrixOfDiagonalElements (Matrix:TMatrix): TMatrix;

Var

Diagonal: TMatrix;

I: Integer;

Begin

Setlength(Diagonal, 2, High(Matrix) + 1);

For I := 0 To High(Matrix) Do

Begin

Diagonal[0][I] := Matrix[I][High(Matrix) - I];

Diagonal[1][I] := I;

End;

CreateMatrixOfDiagonalElements := Diagonal;

End;

Function SortDiagonal(Diagonal: TMatrix): TMatrix;

Var

J, I, MinInColumn, NumOfColumn: Integer;

Begin

For J := 0 To Length(Diagonal[0]) - 1 Do

Begin

I := J + 1;

MinInColumn := Diagonal[0][J];

NumOfcolumn := J;

While (I < Length(Diagonal[0])) Do

Begin

If MinInColumn > Diagonal[0][I] Then

Begin

MinInColumn := Diagonal[0][I];

NumOfColumn := I;

End;

Inc(I);

End;

If NumOfColumn <> J Then

SwapMatrixColumns(Diagonal, J, NumOfColumn);

End;

SortDiagonal := Diagonal;

End;

Procedure MoveLines(ResMatrix, Matrix: TMatrix; NewLine, PrevLine: Integer);

Begin

ResMatrix[PrevLine] := Matrix[NewLine];

End;

Function SortMatrix(Matrix, Diagonal: TMatrix; N: Integer): TMatrix;

Var

ResMatrix: TMatrix;

I, NewLine: Integer;

Begin

ResMatrix := Copy(Matrix);

NewLine := 0;

For I := 0 To Length(Diagonal[1]) - 1 Do

Begin

NewLine := Diagonal[1][I];

MoveLines(ResMatrix, Matrix, NewLine, I);

End;

SortMatrix := ResMatrix;

End;

Procedure FreeMemory(Var Matrix: TMatrix; Var ResMatrix: TMatrix; Var DiagonalJ: TMatrix);

Begin

Matrix := Nil;

ResMatrix := Nil;

DiagonalJ := Nil;

End;

Function ChooseFileOutput() : Boolean;

Var

Choose: Boolean;

Begin

WriteLn;

WriteLn('Вы хотите выводить ответ через файл? (Да - ', 1, ' / Нет - ', 2, ')');

Choose := CheckInOut();

ChooseFileOutput := Choose;

End;

Procedure PrintResult(ResMatrix: TMatrix);

Var

Line, Column: Integer;

F: TextFile;

PrintToFile: Boolean;

Begin

Line := 0;

Column := 0;

PrintToFile := ChooseFileOutput();

If PrintToFile Then

Begin

FileWriting(F);

Append(F);

Writeln(F, 'Отсортированная Матрица:');

End

Else

Writeln('Отсортированная Матрица:');

For Line := 0 To High(ResMatrix) Do

Begin

For Column := 0 To High(ResMatrix) Do

Begin

If PrintToFile Then

Write(F, ResMatrix[Line][Column], ' ')

Else

Write(ResMatrix[Line][Column], ' ');

End;

If PrintToFile Then

Writeln(F)

Else

Writeln;

End;

If PrintToFile Then

CloseFile(F);

End;

Var

F: TextFile;

Order: Integer;

Matrix, Diagonal, ResMatrix: TMatrix;

Begin

PrintTask();

ReadMatrix(Matrix, Order);

Diagonal := CreateMatrixOfDiagonalElements(Matrix);

Diagonal := SortDiagonal(Diagonal);

ResMatrix := SortMatrix(Matrix, Diagonal, Order);

PrintResult(ResMatrix);

FreeMemory(Matrix, ResMatrix, Diagonal);

ReadLn;

End.

**Код программы С++:**

#include <iostream>

#include <string>

#include <fstream>

using namespace std;

enum ERRORS\_LIST {

CORRECT, RANGE\_ERR, NUM\_ERR, NOT\_TXT, NOT\_EXIST, NOT\_READABLE, NOT\_WRITEABLE, ORDER\_ERR, CHOICE\_ERR, FILE\_EMPTY

};

const string ERRORS[] = {

"", "Значение не попадает в диапазон!", "Проверьте корректность ввода данных!", "Расширение не txt!", "Проверьте корректность ввода пути к файлу!", "Файл закрыт для чтения!", "Файл закрыт для записи!", "Значения порядков не равны!", "Проверьте корректность выбора!", "Файл пуст!"

};

constexpr int MIN\_MATRIX = 2;

constexpr int MAX\_MATRIX = 100;

void printTask()

{

cout << "Данная программа располагает строки матрицы по возрастанию элементов побочной диагонали исходной матрицы:\n\n";

}

ERRORS\_LIST checkArea(int num, const int MIN, const int MAX)

{

ERRORS\_LIST error;

error = CORRECT;

if (num < MIN || num > MAX)

error = RANGE\_ERR;

return error;

}

void setLengthMatrix(int\*\*& matrix, int order)

{

matrix = new int\* [order];

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

matrix[i] = new int[order];

}

void printError(ERRORS\_LIST error)

{

cout << ERRORS[error] << "\nПовторите попытку";

}

void setLegnthDiagonal(int\*\*& diagonal, int order)

{

diagonal = new int\* [order];

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

diagonal[i] = new int[order];

}

int checkNum(int MIN, int MAX)

{

ERRORS\_LIST error;

int num;

do

{

error = CORRECT;

cin >> num;

if (cin.fail())

{

error = CHOICE\_ERR;

cin.clear();

while (cin.get() != '\n');

}

if (error == CORRECT && cin.get() != '\n')

{

error = CHOICE\_ERR;

cout << "Некорректный выбор!\n";

while (cin.get() != '\n');

}

if (error == CORRECT)

error = checkArea(num, MIN, MAX);

if (error != CORRECT)

printError(error);

} while (error != CORRECT);

return num;

}

int\*\* copyMatrix(int\*\*& resMatrix, int\*\* matrix, int order)

{

setLengthMatrix(resMatrix, order);

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

resMatrix[i] = new int[order];

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

resMatrix[i][j] = matrix[i][j];

}

}

return resMatrix;

}

bool checkInOut()

{

const int FILE\_CHOICE = 1;

const int CONSOLE\_CHOICE = 2;

ERRORS\_LIST error;

int num;

bool choose;

choose = false;

num = checkNum(FILE\_CHOICE, CONSOLE\_CHOICE);

if (num == 1)

choose = true;

return choose;

}

void setLegnthMaxInColumn(int\*& maxInColumn, int order)

{

maxInColumn = new int[order];

}

bool chooseFileInput()

{

bool choose;

choose = false;

cout << "Вы хотите вводить матрицу через файл? (Да - " << 1 << " / Нет - " << 2 << ")\n";

choose = checkInOut();

return choose;

}

void fileReading(string& pathToFile)

{

ERRORS\_LIST error;

do

{

error = CORRECT;

cout << "Введите путь к файлу с расширением .txt: ";

getline(cin, pathToFile);

if (pathToFile.substr(pathToFile.length() - 4, 4) != ".txt")

error = NOT\_TXT;

else if (!ifstream(pathToFile))

error = NOT\_EXIST;

else

{

ifstream file(pathToFile);

if (!file.is\_open())

error = NOT\_READABLE;

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

{

error = FILE\_EMPTY;

file.close();

}

file.close();

}

if (error != CORRECT)

printError(error);

} while (error != CORRECT);

}

void fileWriting(string& pathToFile)

{

ERRORS\_LIST error;

do

{

error = CORRECT;

cout << "Введите путь к файлу с расширением .txt: ";

getline(cin, pathToFile);

if (pathToFile.substr(pathToFile.length() - 4, 4) != ".txt")

error = NOT\_TXT;

else if (!ifstream(pathToFile))

error = NOT\_EXIST;

else

{

ifstream file(pathToFile);

if (!file.is\_open())

error = NOT\_WRITEABLE;

}

if (error != CORRECT)

printError(error);

} while (error != CORRECT);

}

int readOrder(string& pathTofile, int num)

{

int buf;

int order;

int rows;

int cols;

ERRORS\_LIST error;

order = 0;

error = CORRECT;

ifstream file(pathTofile);

if (num == 2)

{

cout << "Введите порядок матрицы [" << MIN\_MATRIX << ": " << MAX\_MATRIX << "]: ";

order = checkNum(MIN\_MATRIX, MAX\_MATRIX);

}

else

{

rows = 0;

cols = 0;

buf = 0;

file >> order;

file.get() != '\n';

if (checkArea(order, MIN\_MATRIX, MAX\_MATRIX) != CORRECT)

{

error = ORDER\_ERR;

file.close();

}

else

{

while (error == CORRECT && !file.eof())

{

cols = 0;

while (error == CORRECT && file.peek() != '\n' && !file.eof())

{

file >> buf;

cols++;

}

if (error == CORRECT)

{

file.get() != '\n';

rows++;

if (cols == order)

error = CORRECT;

}

}

file.get() != '\n';

}

file.close();

if (error == CORRECT)

{

error = checkArea(rows, MIN\_MATRIX, MAX\_MATRIX);

error = checkArea(cols, MIN\_MATRIX, MAX\_MATRIX);

}

if ((error == CORRECT) && (rows != order || cols != order))

error = ORDER\_ERR;

}

if (error != CORRECT)

printError(error);

if (error == CORRECT)

return order;

}

void readMatrix(int\*\*& matrix, int& order)

{

string pathToFile;

const int MIN\_ELEMENT = -1000;

const int MAX\_ELEMENT = 1000;

ERRORS\_LIST error;

bool isCorrect;

bool fromFile;

int orderCheck;

orderCheck = 0;

fromFile = chooseFileInput();

pathToFile = ' ';

if (fromFile)

{

do

{

isCorrect = true;

fileReading(pathToFile);

order = readOrder(pathToFile, 1);

orderCheck = order;

if (order == 0)

isCorrect = false;

} while (!isCorrect);

}

else

order = readOrder(pathToFile, 2);

setLengthMatrix(matrix, order);

ifstream file(pathToFile);

file.get() != '\n';

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

{

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

{

if (fromFile)

{

file >> matrix[row][col];

error = checkArea(matrix[row][col], MIN\_ELEMENT, MAX\_ELEMENT);

}

else

{

cout << "Введите в " << (row + 1) << " строке " << (col + 1) << " столбце элемент [" << MIN\_ELEMENT << " : " << MAX\_ELEMENT << "]: ";

matrix[row][col] = checkNum(MIN\_ELEMENT, MAX\_ELEMENT);

}

}

if (fromFile)

file.get() != '\n';

}

if (fromFile)

file.close();

}

void swapMatrixColumns(int\*\*& diagonal, int col1, int col2, int order)

{

int temp;

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

{

temp = diagonal[i][col1];

diagonal[i][col1] = diagonal[i][col2];

diagonal[i][col2] = temp;

}

}

int\*\* createMatrixOfDiagonalElements(int\*\* matrix, int& order)

{

int\*\* diagonal;

setLegnthDiagonal(diagonal, order);

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

{

diagonal[0][i] = matrix[i][order - 1 - i];

diagonal[1][i] = i;

}

return diagonal;

}

int\*\* sortDiagonal(int\*\*& diagonal, int& order)

{

int minInColumn;

int numOfColumn;

int i;

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

{

i = j + 1;

minInColumn = diagonal[0][j];

numOfColumn = j;

while (i < order)

{

if (minInColumn > diagonal[0][i])

{

minInColumn = diagonal[0][i];

numOfColumn = i;

}

i++;

}

if (numOfColumn != j)

swapMatrixColumns(diagonal, j, numOfColumn, order);

}

return diagonal;

}

void moveLines(int\*\* resMatrix, int\*\* matrix, int newLine, int prevLine, int order)

{

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

resMatrix[prevLine][col] = matrix[newLine][col];

}

int\*\* sortMatrix(int\*\* matrix, int\*\* diagonal, int order)

{

int\*\* resMatrix;

int newLine;

resMatrix = copyMatrix(resMatrix, matrix, order);

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

{

newLine = diagonal[1][i];

moveLines(resMatrix, matrix, newLine, i, order);

}

return resMatrix;

}

void freeMemory(int\*\*& matrix, int\*\*& resMatrix, int\*\*& diagonal, int order)

{

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

delete[] matrix[i];

delete[] matrix;

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

delete[] resMatrix[i];

delete[] resMatrix;

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

delete[] diagonal[i];

delete[] diagonal;

}

bool chooseFileOutput()

{

cout << "\nВы хотите выводить матрицу через файл? (Да - " << 1 << " / Нет - " << 2 << ")\n";

return checkInOut();

}

void printResult(int\*\* resMatrix, int order)

{

int line = 0;

int column = 0;

string pathToFile;

bool printToFile;

printToFile = chooseFileOutput();

pathToFile = ' ';

if (printToFile)

{

fileWriting(pathToFile);

ofstream fileOut(pathToFile, std::ios::app);

fileOut << "Отсортированная Матрица:\n";

}

else

cout << "Отсортированная Матрица:\n";

ofstream fileOut(pathToFile, std::ios::app);

for (line = 0; line < order; line++)

{

for (column = 0; column < order; column++)

{

if (printToFile)

fileOut << resMatrix[line][column] << " ";

else

cout << resMatrix[line][column] << " ";

}

if (printToFile)

fileOut << "\n";

else

cout << "\n";

}

if (printToFile)

fileOut.close();

}

int main()

{

int\*\* matrix;

int order;

int\*\* diagonal;

int\*\* resMatrix;

setlocale(LC\_ALL, "RU");

printTask();

readMatrix(matrix, order);

diagonal = createMatrixOfDiagonalElements(matrix, order);

diagonal = sortDiagonal(diagonal, order);

resMatrix = sortMatrix(matrix, diagonal, order);

printResult(resMatrix, order);

freeMemory(matrix, resMatrix, diagonal, order);

return 0;

}

**Код программы Java:**

import java.util.Scanner;

import java.io.File;

import java.io.FileNotFoundException;

import java.io.FileWriter;

import java.io.IOException;

public class Main {

public static final int MIN\_MATRIX = 2;

public static final int MAX\_MATRIX = 100;

public static final int MIN\_ELEMENT = -1000;

public static final int MAX\_ELEMENT = 1000;

public enum ERRORS\_LIST {

CORRECT, RANGE\_ERR, NUM\_ERR, NOT\_TXT, NOT\_EXIST, NOT\_READABLE, NOT\_WRITEABLE, ORDER\_ERR, CHOICE\_ERR, FILE\_EMPTY

}

public static final String[]

ERRORS = {

"", "Значение не попадает в диапазон!", "Проверьте корректность ввода данных!", "Расширение не txt!", "Проверьте корректность ввода пути к файлу!", "Файл закрыт для чтения!", "Файл закрыт для записи!", "Значения порядков не равны!", "Проверьте корректность выбора!", "Файл пуст!" };

public static Scanner scanConsole = new Scanner(System.in);

public static Scanner scanFile;

public static File file;

public static void printTask() {

System.out.println("Данная программа располагает строки матрицы по возрастанию элементов побочной диагонали исходной матрицы:\n\n");

}

public static void printError (ERRORS\_LIST error) {

System.out.println(ERRORS[error.ordinal()] + "\nПовторите попытку");

}

public static ERRORS\_LIST checkArea(int num, final int MIN, final int MAX) {

ERRORS\_LIST error;

error = ERRORS\_LIST.CORRECT;

if (num < MIN || num > MAX)

error = ERRORS\_LIST.RANGE\_ERR;

return error;

}

public static int checkNum(int MIN, int MAX) {

ERRORS\_LIST error;

int num = 0;

do {

error = ERRORS\_LIST.CORRECT;

try {

num = Integer.parseInt(scanConsole.nextLine());

} catch (NumberFormatException e) {

error = ERRORS\_LIST.CHOICE\_ERR;

}

} while (error != ERRORS\_LIST.CORRECT);

return num;

}

public static int[][] copyMatrix(int[][] matrix, int order) {

int [][] resMatrix = new int[order][order];

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

resMatrix[i] = new int[order];

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

resMatrix[i][j] = matrix[i][j];

}

return resMatrix;

}

public static boolean checkInOut() {

final int FILE\_CHOICE = 1;

final int CONSOLE\_CHOICE = 2;

int num = 0;

boolean choose = false;

num = checkNum(FILE\_CHOICE, CONSOLE\_CHOICE);

if (num == 1)

choose = true;

return choose;

}

public static boolean chooseFileInput() {

boolean choose = true;

System.out.println("Вы хотите вводить матрицу через файл? (Да - " + 1 + " / Нет - " + 2 + ")");

choose = checkInOut();

return choose;

}

public static String ReadPath(){

ERRORS\_LIST error;

String pathToFile = " ";

do {

error = ERRORS\_LIST.CORRECT;

System.out.print("Введите путь к файлу с расширением .txt: ");

pathToFile = scanConsole.nextLine();

if (!pathToFile.endsWith(".txt"))

error = ERRORS\_LIST.NOT\_TXT;

} while (error != ERRORS\_LIST.CORRECT);

return pathToFile;

}

public static void fileReading () {

ERRORS\_LIST error;

String pathToFile = "";

do {

pathToFile = ReadPath();

error = ERRORS\_LIST.CORRECT;

file = new File(pathToFile);

if (!(file.exists()))

error = ERRORS\_LIST.NOT\_EXIST;

else {

if (!file.canRead())

error = ERRORS\_LIST.NOT\_READABLE;

else {

try (Scanner fileScanner = new Scanner(file)){

if (!fileScanner.hasNext())

error = ERRORS\_LIST.FILE\_EMPTY;

} catch (FileNotFoundException e) {

printError(error);

}

}

}

if (error != ERRORS\_LIST.CORRECT)

printError(error);

} while (error != ERRORS\_LIST.CORRECT);

}

public static String fileWriting() {

ERRORS\_LIST error;

String pathToFile = "";

do {

pathToFile = ReadPath();

error = ERRORS\_LIST.CORRECT;

if (!(new File(pathToFile)).exists())

error = ERRORS\_LIST.NOT\_EXIST;

else {

File file = new File(pathToFile);

if (!file.canWrite())

error = ERRORS\_LIST.NOT\_WRITEABLE;

}

if (error != ERRORS\_LIST.CORRECT)

printError(error);

} while (error != ERRORS\_LIST.CORRECT);

return pathToFile;

}

public static int readOrderFile(){

int order = 0;

ERRORS\_LIST error;

error = ERRORS\_LIST.CORRECT;

int rows, cols, buf;

rows = 0;

buf = 0;

cols = 0;

try {scanFile = new Scanner(file);} catch (FileNotFoundException e) {}

order = scanFile.nextInt();

error = checkArea(order, MIN\_MATRIX, MAX\_MATRIX);

scanFile.nextLine();

while (error == ERRORS\_LIST.CORRECT && scanFile.hasNextLine()) {

cols = 0;

String line = scanFile.nextLine();

Scanner lineScanner = new Scanner(line);

while (error == ERRORS\_LIST.CORRECT && lineScanner.hasNextInt()) {

buf = lineScanner.nextInt();

cols++;

}

lineScanner.close();

if (error == ERRORS\_LIST.CORRECT) {

rows++;

if (cols == order)

error = ERRORS\_LIST.CORRECT;

}

}

scanFile.close();

if (error == ERRORS\_LIST.CORRECT) {

error = checkArea(rows, MIN\_MATRIX, MAX\_MATRIX);

error = checkArea(cols, MIN\_MATRIX, MAX\_MATRIX);

}

if (error == ERRORS\_LIST.CORRECT && (rows != order || cols != order))

error = ERRORS\_LIST.ORDER\_ERR;

if (error != ERRORS\_LIST.CORRECT)

printError(error);

return order;

}

public static int readOrderConsole() {

int order = 0;

System.out.print("Введите порядок матрицы [" + MIN\_MATRIX + ": " + MAX\_MATRIX + "]: ");

order = checkNum(MIN\_MATRIX, MAX\_MATRIX);

return order;

}

public static int[][] readMatrixConsole(int order) {

int[][] matrix = new int[order][order];

Scanner scanConsole = new Scanner(System.in);

ERRORS\_LIST error;

error = ERRORS\_LIST.CORRECT;

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

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

do {

System.out.print("Введите в " + (row + 1) + " строке " + (col + 1) + " столбце элемент[" + MIN\_ELEMENT + ":" + MAX\_ELEMENT + "]: ");

error = ERRORS\_LIST.CORRECT;

try {

matrix[row][col] = Integer.parseInt(scanConsole.nextLine());

} catch (NumberFormatException e) {

error = ERRORS\_LIST.NUM\_ERR;

}

if (error == ERRORS\_LIST.CORRECT)

error = checkArea(matrix[row][col], MIN\_ELEMENT, MAX\_ELEMENT);

} while (!(error == ERRORS\_LIST.CORRECT));

}

}

return matrix;

}

public static int[][] readMatrixFile(int order) {

int[][] matrix;

matrix = new int[order][order];

ERRORS\_LIST error = ERRORS\_LIST.CORRECT;

try {scanFile = new Scanner(file);} catch (FileNotFoundException e) {}

scanFile.nextLine();

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

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

try {

matrix[row][col] = scanFile.nextInt();

} catch (NumberFormatException e) {

error = checkArea(MIN\_ELEMENT, MAX\_ELEMENT, matrix[row][col]);

}

}

scanFile.close();

if (error != ERRORS\_LIST.CORRECT)

printError(error);

return matrix;

}

public static int[][] readMatrix() {

int[][] matrix;

int order;

boolean isFromConsole = (!chooseFileInput());

if (!isFromConsole) {

fileReading();

order = readOrderFile();

matrix = readMatrixFile(order);

}

else {

order = readOrderConsole();

matrix = readMatrixConsole(order);

}

return matrix;

}

public static void swapMatrixColumns(int[][] diagonal, int col1, int col2) {

int temp;

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

temp = diagonal[i][col1];

diagonal[i][col1] = diagonal[i][col2];

diagonal[i][col2] = temp;

}

}

public static int[][] createMatrixOfDiagonalElements(int[][] matrix) {

int[][] diagonal = new int[2][matrix.length];

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

diagonal[0][i] = matrix[i][matrix.length - 1 - i];

diagonal[1][i] = i;

}

return diagonal;

}

public static int[][] sortDiagonal(int[][] diagonal) {

int minInColumn;

int numOfColumn;

int i;

for (int j = 0; j < diagonal[0].length - 1; j++) {

i = j + 1;

minInColumn = diagonal[0][j];

numOfColumn = j;

while (i < diagonal[0].length) {

if (minInColumn > diagonal[0][i]) {

minInColumn = diagonal[0][i];

numOfColumn = i;

}

i++;

}

if (numOfColumn != j)

swapMatrixColumns(diagonal, j, numOfColumn);

}

return diagonal;

}

public static int[][] sortMatrix(int[][] matrix, int[][] diagonal) {

int[][] resMatrix = null;

int newLine;

resMatrix = copyMatrix(matrix, matrix.length) ;

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

newLine = diagonal[1][i];

moveLines(resMatrix, matrix, newLine, i);

}

return resMatrix;

}

public static void moveLines(int[][] resMatrix, int[][] matrix, int newLine, int prevLine) {

for (int col = 0; col < matrix.length; ++col)

resMatrix[prevLine][col] = matrix[newLine][col];

}

public static boolean chooseFileOutput() {

System.out.println("Вы хотите выводить матрицу через файл? (Да - " + 1 + " / Нет - " + 2 + ")");

return checkInOut();

}

public static void printResult(int[][] resMatrix) {

int line = 0;

int column = 0;

String pathToFile = "";

boolean printToFile = chooseFileOutput();

if (printToFile) {

pathToFile = fileWriting();

try (FileWriter fileOut = new FileWriter(pathToFile)) {

fileOut.write("Отсортированная Матрица:\n");

} catch (IOException e) {

e.printStackTrace();

}

} else

System.out.println("Отсортированная Матрица:");

for (line = 0; line < resMatrix.length; line++) {

for (column = 0; column < resMatrix.length; column++) {

if (printToFile) {

try (FileWriter fileOut = new FileWriter(pathToFile, true)){

fileOut.write(resMatrix[line][column]+" ");

} catch (IOException e) {

e.printStackTrace();

}

} else

System.out.print(resMatrix[line][column] + " ");

}

if (printToFile)

try (FileWriter fileOut = new FileWriter(pathToFile, true)) {

fileOut.write("\n");

} catch (IOException e) {}

else

System.out.println();

}

}

public static void main(String[] args) {

int[][] matrix = null;

int[][] diagonal = null;

int[][] resMatrix = null;

printTask();

matrix = readMatrix();

diagonal = createMatrixOfDiagonalElements(matrix);

diagonal = sortDiagonal(diagonal);

resMatrix = sortMatrix(matrix, diagonal);

printResult(resMatrix);

scanConsole.close();

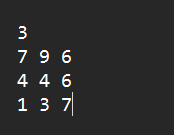
scanFile.close();

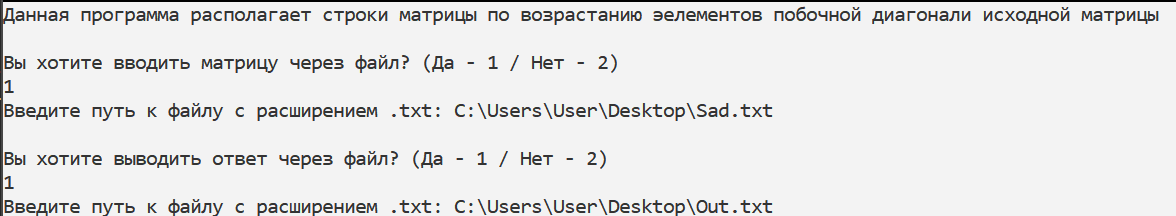
}

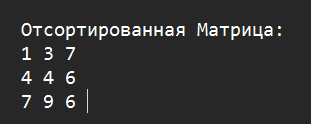
}

**Скриншоты:**

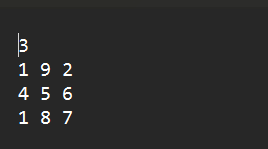
**Delphi:**

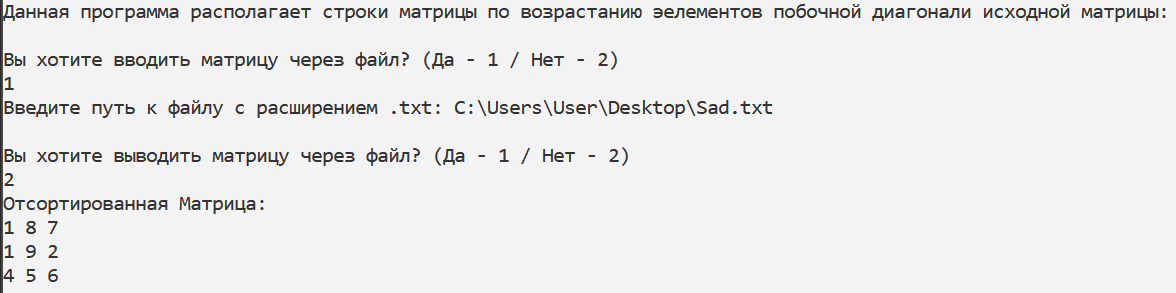
****

****

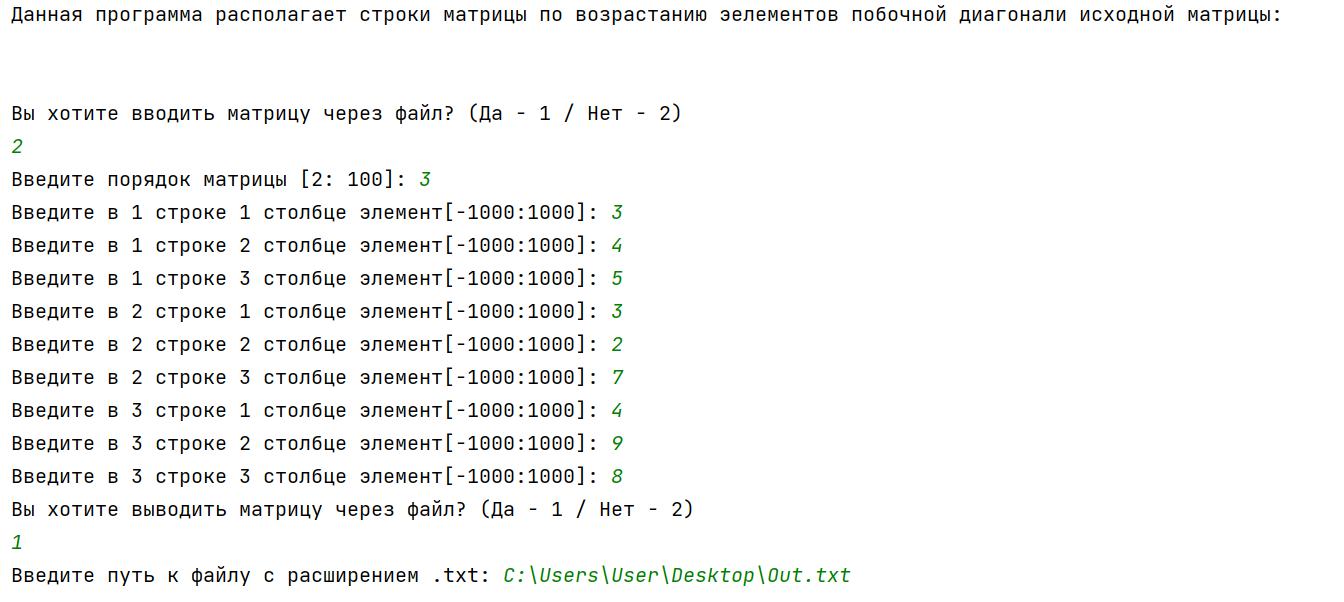
****

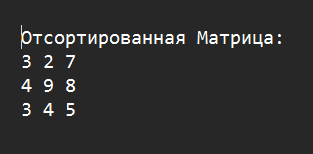
**C++:**

****

****

**Java:**

****

****

**Блок-схема:**

