УО «Белорусский государственный университет информатики и радиоэлектроники»

Кафедра ПОИТ

Отчет по лабораторной работе №3.3

по предмету «Основы алгоритмизации и программирования»

Вариант 20

Выполнил:

Захвей И.В.

Гр. 351005

Проверил:

Данилова Г. В.

Минск 2023

**Задание:**

Сортировка естественным слиянием.

**Код программы Delphi:**

Program Project4;

Uses

System.SysUtils;

Type

TArray = Array Of Integer;

ERRORS\_CODE = (SUCCESS, INCORRECT\_DATA, EMPTY\_LINE, NOT\_TXT, FILE\_NOT\_EXIST,

INCORRECT\_DATA\_FILE, A\_LOT\_OF\_DATA\_FILE, FILE\_NOT\_AVAILABLE,

OUT\_OF\_BORDER\_SIZE, OUT\_OF\_BORDER\_NUMB, INCORRECT\_BORDERS);

Const

MAX\_NUMB = 2000000000;

MIN\_NUMB = -2000000000;

MAX\_SIZE = 100;

MIN\_SIZE = 2;

ERRORS: Array [ERRORS\_CODE] Of String = ('Successfull',

'Data is not correct', 'Line is empty, please be careful',

'This is not a .txt file', 'This file is not exist',

'Data in file is not correct', 'There are two numbers in file should be',

'File is can not be opened', 'Out of border [2, 100]',

'Out of border [-2000000000, 2000000000]', 'Incorrect borders');

Procedure PrintInf();

Begin

Writeln('The program implements sorting by natural merging');

End;

Function MergeWithPointers(Var Arr, PointersArr: TArray): TArray;

Var

Start1, Stop1, Start2, Stop2: Integer;

I, J, Counter, SizeArr, PointerInd: Integer;

MergedArr: TArray;

Begin

I := 0;

Counter := Length(PointersArr) - Length(PointersArr) Mod 4;

SizeArr := Length(PointersArr) Div 2;

PointerInd := 0;

SetLength(MergedArr, Length(Arr));

Repeat

Start1 := PointersArr[PointerInd];

Inc(PointerInd);

Stop1 := PointersArr[PointerInd];

Inc(PointerInd);

Start2 := PointersArr[PointerInd];

Inc(PointerInd);

Stop2 := PointersArr[PointerInd];

Inc(PointerInd);

While (Start1 < Stop1) And (Start2 < Stop2) Do

If Arr[Start1] > Arr[Start2] Then

Begin

MergedArr[I] := Arr[Start2];

Inc(I);

Inc(Start2);

End

Else

Begin

MergedArr[I] := Arr[Start1];

Inc(I);

Inc(Start1);

End;

While Start1 < Stop1 Do

Begin

MergedArr[I] := Arr[Start1];

Inc(I);

Inc(Start1);

End;

While Start2 < Stop2 Do

Begin

MergedArr[I] := Arr[Start2];

Inc(I);

Inc(Start2);

End;

Until (PointerInd = Counter) Or (PointersArr[PointerInd] = 0);

If (I < SizeArr) Then

For J := PointersArr[PointerInd] To PointersArr[PointerInd + 1] Do

Begin

MergedArr[I] := Arr[J];

Inc(I);

End;

MergeWithPointers := MergedArr;

End;

Procedure FillWithZero(Var Arr: TArray);

Var

I: Integer;

Begin

For I := 0 To High(Arr) Do

Arr[I] := 0;

End;

Function MergeSort(Var Arr: TArray): TArray;

Var

I, PointInd, SizePointers: Integer;

PointersArr: TArray;

Begin

SizePointers := 2 \* Length(Arr);

SetLength(PointersArr, SizePointers);

Repeat

FillWithZero(PointersArr);

PointInd := 0;

PointersArr[PointInd] := 0;

Inc(PointInd);

For I := 1 To High(Arr) Do

If Arr[I] < Arr[I - 1] Then

Begin

PointersArr[PointInd] := I;

Inc(PointInd);

PointersArr[PointInd] := I;

Inc(PointInd);

End;

PointersArr[PointInd] := Length(Arr);

Arr := MergeWithPointers(Arr, PointersArr)

Until PointersArr[1] = Length(Arr);

MergeSort := Arr;

End;

Function InpChoice(Var Choice: Integer): ERRORS\_CODE;

Var

Err: ERRORS\_CODE;

ChoiceStr: String;

Begin

Err := SUCCESS;

Readln(ChoiceStr);

If (ChoiceStr = '1') Or (ChoiceStr = '2') Then

Choice := StrToInt(ChoiceStr)

Else If (Length(ChoiceStr) > 0) Then

Err := INCORRECT\_DATA

Else

Err := EMPTY\_LINE;

InpChoice := Err;

End;

Function UserChoice(): Integer;

Var

Choice: Integer;

Err: ERRORS\_CODE;

Begin

Writeln('Choose a way of input/output of data', #13#10, '1 -- Console',

#13#10, '2 -- File');

Repeat

Err := InpChoice(Choice);

If (Err <> SUCCESS) Then

Writeln(ERRORS[Err], #13#10, 'Please, enter again');

Until (Err = SUCCESS);

UserChoice := Choice;

End;

Function InpValidSize(Var Size: Integer): ERRORS\_CODE;

Var

Err: ERRORS\_CODE;

Line: String;

Begin

Err := SUCCESS;

Readln(Line);

Try

Size := StrToInt(Line);

Except

Err := INCORRECT\_DATA;

End;

If (Err = SUCCESS) And ((Size > MAX\_SIZE) Or (Size < MIN\_SIZE)) Then

Err := OUT\_OF\_BORDER\_SIZE;

InpValidSize := Err;

End;

Function InpValidArr(Var Arr: TArray): ERRORS\_CODE;

Var

Err: ERRORS\_CODE;

I: Integer;

Line: String;

Begin

Err := SUCCESS;

I := 0;

While (I < Length(Arr)) And (Err = SUCCESS) Do

Begin

Readln(Line);

Try

Arr[I] := StrToInt(Line);

Except

Err := INCORRECT\_DATA;

End;

If (Err = SUCCESS) And ((Arr[I] > MAX\_NUMB) Or (Arr[I] < MIN\_NUMB)) Then

Err := OUT\_OF\_BORDER\_NUMB;

Inc(I);

End;

InpValidArr := Err;

End;

Function InputFromConsole(): TArray;

Var

Err: ERRORS\_CODE;

DefaultArr: TArray;

Size: Integer;

Begin

Writeln('Enter the size[2, 100] and then the', #13#10,

'elements[-2000000000, 2000000000] through the Enter');

Repeat

Err := InpValidSize(Size);

If (Err <> SUCCESS) Then

Writeln(ERRORS[Err], #10#13, 'Please, enter again size');

Until (Err = SUCCESS);

SetLength(DefaultArr, Size);

Writeln('Enter the ', Size, ' elements');

Repeat

Err := InpValidArr(DefaultArr);

If (Err <> SUCCESS) Then

Writeln(ERRORS[Err], #10#13, 'Please, enter again');

Until (Err = SUCCESS);

InputFromConsole := DefaultArr;

End;

Function FileAvailable(Name: String; ForReset: Boolean): ERRORS\_CODE;

Var

Err: ERRORS\_CODE;

MyFile: TextFile;

Begin

Err := SUCCESS;

AssignFile(MyFile, Name);

If ForReset Then

Try

Try

Reset(MyFile);

Finally

CloseFile(MyFile);

End;

Except

Err := FILE\_NOT\_AVAILABLE;

End

Else

Try

Try

Rewrite(MyFile);

Finally

CloseFile(MyFile);

End;

Except

Err := FILE\_NOT\_AVAILABLE;

End;

FileAvailable := Err;

End;

Function GetLastFourChar(Line: String): String;

Var

Start, I, Size: Integer;

LastFourChar: String;

Begin

Size := Length(Line);

Start := Size - 3;

For I := Start To Size Do

LastFourChar := LastFourChar + Line[I];

GetLastFourChar := LastFourChar;

End;

Function FileTxt(Name: String): ERRORS\_CODE;

Var

Err: ERRORS\_CODE;

LastFourChar: String;

Begin

Err := SUCCESS;

If Length(Name) > 4 Then

Begin

LastFourChar := GetLastFourChar(Name);

If LastFourChar <> '.txt' Then

Err := NOT\_TXT;

End

Else

Err := NOT\_TXT;

FileTxt := Err;

End;

Function FileExist(Name: String): ERRORS\_CODE;

Var

Err: ERRORS\_CODE;

Begin

Err := SUCCESS;

If Not FileExists(Name) Then

Err := FILE\_NOT\_EXIST;

FileExist := Err;

End;

Function GetFileName(ForReset: Boolean): String;

Var

IsCorrect: Boolean;

ErrExist, ErrTxt, ErrAvailable: ERRORS\_CODE;

FileName: String;

Begin

Repeat

IsCorrect := True;

Readln(FileName);

ErrExist := FileExist(FileName);

ErrTxt := FileTxt(FileName);

If (ErrExist <> SUCCESS) Then

Begin

Writeln(ERRORS[ErrExist]);

IsCorrect := False;

Writeln('Please, enter full path again');

End

Else If (ErrTxt <> SUCCESS) Then

Begin

Writeln(ERRORS[ErrTxt]);

IsCorrect := False;

Writeln('Please, enter full path again');

End

Else

Begin

ErrAvailable := FileAvailable(FileName, ForReset);

If (ErrAvailable <> SUCCESS) Then

Begin

Writeln(ERRORS[ErrAvailable]);

IsCorrect := False;

Writeln('Please, enter full path again');

End;

End;

Until IsCorrect;

GetFileName := FileName;

End;

Function ReadSizeFromFile(Var Size: Integer; Var MyFile: TextFile): ERRORS\_CODE;

Var

Line: String;

Err: ERRORS\_CODE;

Begin

Err := SUCCESS;

Try

Read(MyFile, Size);

Except

Err := INCORRECT\_DATA\_FILE;

End;

If (Err = SUCCESS) And ((Size > MAX\_SIZE) Or (Size < MIN\_SIZE)) Then

Err := OUT\_OF\_BORDER\_SIZE;

ReadSizeFromFile := Err;

End;

Function ReadArrFromFile(Var Arr: TArray; Var MyFile: TextFile): ERRORS\_CODE;

Var

Line: String;

I: Integer;

Err: ERRORS\_CODE;

Begin

Err := SUCCESS;

I := 0;

While (I < Length(Arr)) And (Err = SUCCESS) Do

Begin

Try

Read(MyFile, Arr[I]);

Except

Err := INCORRECT\_DATA\_FILE;

End;

If (Err = SUCCESS) And ((Arr[I] > MAX\_NUMB) Or (Arr[I] < MIN\_NUMB)) Then

Err := OUT\_OF\_BORDER\_NUMB;

Inc(I);

If (I = Length(Arr)) And (Not Eof(MyFile)) Then

Err := A\_LOT\_OF\_DATA\_FILE;

End;

ReadArrFromFile := Err;

End;

Function InputFromFile(): TArray;

Var

Err: ERRORS\_CODE;

FileName: String;

InfFile: TextFile;

DefaultArr: TArray;

Size: Integer;

Begin

Writeln('Enter full path to file');

Repeat

FileName := GetFileName(True);

AssignFile(InfFile, FileName);

Reset(InfFile);

Err := ReadSizeFromFile(Size, InfFile);

If (Err = SUCCESS) Then

Begin

SetLength(DefaultArr, Size);

Err := ReadArrFromFile(DefaultArr, InfFile);

End;

If Err <> SUCCESS Then

Writeln(ERRORS[Err], #13#10, 'Enter full path to file');

CloseFile(InfFile);

Until (Err = SUCCESS);

Writeln('Reading is successfull');

InputFromFile := DefaultArr;

End;

Function InputInf(): TArray;

Var

DefaultArr: TArray;

Choice: Integer;

Begin

Choice := UserChoice();

If (Choice = 1) Then

DefaultArr := InputFromConsole()

Else

DefaultArr := InputFromFile();

InputInf := DefaultArr;

End;

Procedure OutputInConsole(Var DefaultArr, SortedArr: TArray);

Var

I: Integer;

Begin

Writeln('Default array');

For I := 0 To High(DefaultArr) Do

Write(DefaultArr[I], ' ');

Writeln(#13#10, 'Sorted array');

For I := 0 To High(SortedArr) Do

Write(SortedArr[I], ' ');

End;

Procedure OutputInFile(Var DefaultArr, SortedArr: TArray);

Var

FileName: String;

I: Integer;

MyFile: TextFile;

Begin

Writeln('Enter full path to file');

FileName := GetFileName(False);

AssignFile(MyFile, FileName);

Rewrite(MyFile);

Writeln(MyFile, 'Default array');

For I := 0 To High(DefaultArr) Do

Write(MyFile, DefaultArr[I], ' ');

Writeln(MyFile, #13#10, 'Sorted array');

For I := 0 To High(SortedArr) Do

Write(MyFile, SortedArr[I], ' ');

CloseFile(MyFile);

Writeln('Writing is successfull');

End;

Procedure OutputInf(Var DefaultArr, SortedArr: TArray);

Var

Choice: Integer;

Begin

Choice := UserChoice();

If (Choice = 1) Then

OutputInConsole(DefaultArr, SortedArr)

Else

OutputInFile(DefaultArr, SortedArr);

End;

Function CopyArr(Var Arr: TArray): TArray;

Var

CopyedArr: TArray;

I: Integer;

Begin

SetLength(CopyedArr, Length(Arr));

For I := 0 To High(Arr) Do

CopyedArr[I] := Arr[I];

CopyArr := CopyedArr;

End;

Var

DefaultArr, SortedArr: TArray;

Begin

PrintInf();

DefaultArr := InputInf();

SortedArr := CopyArr(DefaultArr);

SortedArr := MergeSort(SortedArr);

OutputInf(DefaultArr, SortedArr);

Readln;

End.

**Код программы С++:**

#include <iostream>

#include <string>

#include <fstream>

using namespace std;

enum ErrorsCode

{

SUCCESS,

INCORRECT\_DATA,

EMPTY\_LINE,

NOT\_TXT,

FILE\_NOT\_EXIST,

INCORRECT\_DATA\_FILE,

A\_LOT\_OF\_DATA\_FILE,

OUT\_OF\_BORDER\_SIZE,

OUT\_OF\_BORDER\_NUMB,

};

const int MIN\_NUMB = -2000000000,

MAX\_NUMB = 2000000000,

MIN\_SIZE = 2,

MAX\_SIZE = 100;

const string ERRORS[] = { "Successfull",

"Data is not correct, or number is too large\n",

"Line is empty, please be careful\n",

"This is not a .txt file\n",

"This file is not exist\n",

"Data in file is not correct\n",

"There are only elements of array should be in file\n",

"Out of border size [2, 100]\n",

"Out of border [-2000000000, 2000000000]\n" };

void printInf()

{

cout << "The program implements sorting by natural merging";

}

int\* mergeWithPointers(int\* arr, int sizeArr, int\* pointersArr, int sizePointer)

{

int start1, stop1, start2, stop2, i, j, counter, sizeArr, pointerInd;

int\* mergedArr = new int[sizeArr];

i = 0;

counter = sizePointer - sizePointer % 4;

pointerInd = 0;

do

{

start1 = pointersArr[pointerInd++];

stop1 = pointersArr[pointerInd++];

start2 = pointersArr[pointerInd++];

stop2 = pointersArr[pointerInd++];

while (start1 < stop1 && start2 < stop2)

{

if (arr[start1] > arr[start2])

mergedArr[i++] = arr[start2++];

else

mergedArr[i++] = arr[start1++];

}

while (start1 < stop1)

{

mergedArr[i++] = arr[start1++];

}

while (start2 < stop2)

{

mergedArr[i++] = arr[start2++];

}

} while (pointerInd < counter && pointersArr[pointerInd] > 0);

if (i < sizeArr)

{

for (j = pointersArr[pointerInd++]; j < pointersArr[pointerInd]; j++)

{

mergedArr[i++] = arr[j];

}

}

return mergedArr;

}

void freeMemory(int\* arr)

{

delete[] arr;

}

void fillWithZero(int\* arr, int size) {

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

arr[i] = 0;

}

int\* mergeSort(int\* arr, int size) {

int i, pointInd, sizePointers;

sizePointers = 2 \* size;

int\* pointersArr = new int[sizePointers];

do {

fillWithZero(pointersArr, sizePointers);

pointInd = 0;

pointersArr[pointInd++] = 0;

for (i = 1; i < size; i++) {

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

pointersArr[pointInd++] = i;

pointersArr[pointInd++] = i;

}

}

pointersArr[pointInd] = i;

arr = mergeWithPointers(arr, size, pointersArr, sizePointers);

} while (pointersArr[1] != size);

freeMemory(pointersArr);

return arr;

}

ErrorsCode inpChoice(int& choice)

{

ErrorsCode err;

string choiceStr;

err = SUCCESS;

getline(cin, choiceStr);

if (choiceStr == "1" || choiceStr == "2")

choice = stoi(choiceStr);

else

err = choiceStr.length() > 0 ? INCORRECT\_DATA : EMPTY\_LINE;

return err;

}

int userChoice()

{

int choice;

cout << "Choose a way of input/output of data\n"

<< "1 -- Console\n"

<< "2 -- File\n";

ErrorsCode err;

do

{

err = inpChoice(choice);

if (err != SUCCESS)

cout << ERRORS[err] << "Please, enter again\n";

} while (err != SUCCESS);

return choice;

}

ErrorsCode inpValidSize(int& size)

{

ErrorsCode err;

err = SUCCESS;

bool isCorrect = true;

cin >> size;

if (cin.fail())

{

cin.clear();

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

err = INCORRECT\_DATA;

}

if (err == SUCCESS && cin.get() != '\n')

{

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

err = INCORRECT\_DATA;

}

if (err == SUCCESS && ((size > MAX\_SIZE) || (size < MIN\_SIZE)))

err = OUT\_OF\_BORDER\_SIZE;

return err;

}

ErrorsCode inpValidArr(int\* arr, int size)

{

ErrorsCode err;

int i;

i = 0;

err = SUCCESS;

while (i < size && err == SUCCESS)

{

cin >> arr[i];

if (cin.fail())

{

cin.clear();

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

err = INCORRECT\_DATA;

}

if (err == SUCCESS && cin.get() != '\n')

{

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

err = INCORRECT\_DATA;

}

if (err == SUCCESS && ((arr[i] > MAX\_NUMB) || (arr[i] < MIN\_NUMB)))

err = OUT\_OF\_BORDER\_NUMB;

i++;

}

return err;

}

int\* inputFromConsole(int& size)

{

cout << "Enter the size[2, 100] and then the \n"

<< "elements[-2000000000, 2000000000] through the Enter\n";

ErrorsCode err;

do

{

err = inpValidSize(size);

if (err != SUCCESS)

cout << ERRORS[err] << "Please, enter again size\n";

} while (err != SUCCESS);

int\* defaultArr = new int[size];

cout << "Enter the " << size << " elements\n";

do

{

err = inpValidArr(defaultArr, size);

if (err != SUCCESS)

cout << ERRORS[err] << "Enter the " << size << " elements\n";

} while (err != SUCCESS);

return defaultArr;

}

ErrorsCode readSizeFromFile(int& size, ifstream& file)

{

ErrorsCode err;

err = SUCCESS;

file >> size;

if (file.fail())

{

file.clear();

err = INCORRECT\_DATA\_FILE;

}

if (err == SUCCESS && ((size > MAX\_SIZE) || (size < MIN\_SIZE)))

err = OUT\_OF\_BORDER\_NUMB;

return err;

}

ErrorsCode readArrFromFile(int\* arr, int size, ifstream& file)

{

ErrorsCode err;

err = SUCCESS;

int i;

i = 0;

while (i < size && err == SUCCESS)

{

file >> arr[i];

if (file.fail())

{

file.clear();

err = INCORRECT\_DATA\_FILE;

}

if (err == SUCCESS && ((arr[i] > MAX\_NUMB) || (arr[i] < MIN\_NUMB)))

err = OUT\_OF\_BORDER\_NUMB;

i++;

if (i == size && !file.eof())

err = A\_LOT\_OF\_DATA\_FILE;

}

return err;

}

ErrorsCode isFileExist(string nameOfFile)

{

ErrorsCode err;

ifstream file(nameOfFile);

err = file.is\_open() ? SUCCESS : FILE\_NOT\_EXIST;

file.close();

return err;

}

string getLastFourChar(string line)

{

string lastFourChar;

int start, i, size;

size = line.length();

start = size - 4;

for (i = start; i < size; i++)

lastFourChar += line[i];

return lastFourChar;

}

ErrorsCode thisIsTxtFile(string& fileName)

{

ErrorsCode err = SUCCESS;

string lastFourChar;

if (fileName.length() > 4)

{

lastFourChar = getLastFourChar(fileName);

if (lastFourChar != ".txt")

err = NOT\_TXT;

}

else

err = NOT\_TXT;

return err;

}

string getFileName()

{

bool isIncorrect;

string name;

int errExist, errTxt;

cout << "Enter full path to file\n";

do

{

getline(cin, name);

errExist = isFileExist(name);

errTxt = thisIsTxtFile(name);

isIncorrect = false;

if (errTxt > 0)

{

cout << ERRORS[errTxt];

isIncorrect = true;

}

else if (errExist > 0)

{

cout << ERRORS[errExist];

isIncorrect = true;

}

} while (isIncorrect);

return name;

}

int\* inputFromFile(int& size)

{

string fileName;

int\* defaultArr = new int;

ErrorsCode err;

do

{

fileName = getFileName();

ifstream file = ifstream(fileName);

err = readSizeFromFile(size, file);

if (err == SUCCESS)

{

defaultArr = new int[size];

err = readArrFromFile(defaultArr, size, file);

}

if (err != SUCCESS)

cout << ERRORS[err];

file.close();

} while (err != SUCCESS);

cout << "Reading is successfull\n";

return defaultArr;

}

int\* inputInf(int& size)

{

int\* defaultArr;

int choice = userChoice();

if (choice == 1)

defaultArr = inputFromConsole(size);

else

defaultArr = inputFromFile(size);

return defaultArr;

}

void outputInConsole(int\* defaultArr, int\* sortedArr, int size)

{

int i;

cout << "Default array\n";

for (i = 0; i < size; i++)

cout << defaultArr[i] << " ";

cout << "\nSorted array\n";

for (i = 0; i < size; i++)

cout << sortedArr[i] << " ";

}

void outputInFile(int\* defaultArr, int\* sortedArr, int size)

{

string fileName = getFileName();

ofstream file(fileName);

int i;

file << "Default array\n";

for (i = 0; i < size; i++)

file << defaultArr[i] << " ";

file << "\nSorted array\n";

for (i = 0; i < size; i++)

file << sortedArr[i] << " ";

cout << "Writing is successfull\n";

file.close();

}

void outputInf(int\* defaultArr, int\* sortedArr, int size)

{

int choice = userChoice();

if (choice == 1)

outputInConsole(defaultArr, sortedArr, size);

else

outputInFile(defaultArr, sortedArr, size);

freeMemory(defaultArr);

freeMemory(sortedArr);

}

int\* copyArr(int\* arr, int size) {

int\* copyedArr = new int[size];

int i;

for (i = 0; i < size; i++) {

copyedArr[i] = arr[i];

}

return copyedArr;

}

int main()

{

int size;

int\* defaultArr;

int\* sortedArr;

inputInf;

defaultArr = inputInf(size);

sortedArr = copyArr(defaultArr, size);

sortedArr = mergeSort(sortedArr, size);

outputInf(defaultArr, sortedArr, size);

return 0;

}

**Код программы Java:**

import java.io.File;

import java.io.IOException;

import java.io.PrintWriter;

import java.nio.file.Path;

import java.nio.file.Paths;

import java.util.Scanner;

public class Main {

static enum Codes {

SUCCESS,

INCORRECT\_DATA,

EMPTY\_LINE,

NOT\_TXT,

FILE\_NOT\_EXIST,

INCORRECT\_DATA\_FILE,

A\_LOT\_OF\_DATA\_FILE,

IN\_OUT\_FILE\_EXCEPTION,

OUT\_OF\_BORDER\_SIZE,

OUT\_OF\_BORDER\_NUMB,

}

static final String[] ERRORS ={"Successfull",

"Data is not correct, or number is too large",

"Line is empty, please be careful",

"This is not a .txt file",

"This file is not exist",

"Data in file is not correct",

"There are only elements of array should be in file",

"Exception with output/input from the file",

"Out of border size [2, 100]",

"Out of border [-2000000000, 2000000000]"};

static final int MAX\_NUMB = 2000000000,

MIN\_NUMB = -2000000000,

MAX\_SIZE = 100,

MIN\_SIZE = 2;

static void printInf() {

System.out.println("The program implements sorting by natural merging");

}

static int[] mergeWithPointers(int[] arr, int[] pointersArr) {

int start1, stop1, start2, stop2, i, j, counter, pointerInd;

int [] mergedArr = new int[arr.length];

i = 0;

counter = pointersArr.length - pointersArr.length % 4;

pointerInd = 0;

do {

start1 = pointersArr[pointerInd++];

stop1 = pointersArr[pointerInd++];

start2 = pointersArr[pointerInd++];

stop2 = pointersArr[pointerInd++];

while (start1 < stop1 && start2 < stop2) {

if (arr[start1] > arr[start2])

mergedArr[i++] = arr[start2++];

else

mergedArr[i++] = arr[start1++];

}

while (start1 < stop1) {

mergedArr[i++] = arr[start1++];

}

while (start2 < stop2) {

mergedArr[i++] = arr[start2++];

}

} while (pointerInd < counter && pointersArr[pointerInd] > 0);

if (i < arr.length) {

for (j = pointersArr[pointerInd++]; j < pointersArr[pointerInd]; j++) {

mergedArr[i++] = arr[j];

}

}

return mergedArr;

}

static void fillWithZero(int[] arr) {

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

arr[i] = 0;

}

}

static int[] mergeSort(int[] arr) {

int i, pointInd, size;

size = arr.length;

int[] pointersArr = new int[2\*size];

do {

fillWithZero(pointersArr);

pointInd = 0;

pointersArr[pointInd++] = 0;

for (i = 1; i < size; i++) {

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

pointersArr[pointInd++] = i;

pointersArr[pointInd++] = i;

}

}

pointersArr[pointInd] = i;

arr = mergeWithPointers(arr, pointersArr);

} while (pointersArr[1] != arr.length);

return arr;

}

static Codes inputChoice(Scanner input, int[] choice){

Codes err;

String choiceStr;

err = Codes.SUCCESS;

choiceStr = input.nextLine();

if (choiceStr.equals("1") || choiceStr.equals("2")) {

choice[0] = Integer.parseInt(choiceStr);

} else {

err = choiceStr.isEmpty() ? Codes.EMPTY\_LINE : Codes.INCORRECT\_DATA;

}

return err;

}

static int userChoice(Scanner input) {

int[] choice = {0};

System.out.println("Choose a way of input/output of data\n"

+ "1 -- Console\n"

+ "2 -- File");

Codes err;

do {

err = inputChoice(input, choice);

if (err != Codes.SUCCESS) {

System.err.println(ERRORS[err.ordinal()]);

System.out.println("Please, enter again");

}

} while (err != Codes.SUCCESS);

return choice[0];

}

static Codes inpValidSize(int[] size, Scanner input) {

Codes err;

int sizeInt;

sizeInt = 0;

err = Codes.SUCCESS;

try {

sizeInt = Integer.parseInt(input.nextLine());

} catch (NumberFormatException e) {

err = Codes.INCORRECT\_DATA;

}

if (err == Codes.SUCCESS && (sizeInt > MAX\_SIZE || sizeInt < MIN\_SIZE)) {

err = Codes.OUT\_OF\_BORDER\_SIZE;

}

size[0] = sizeInt;

return err;

}

static Codes inpValidArr(int[] arr, Scanner input) {

Codes err;

int i;

i = 0;

err = Codes.SUCCESS;

while (i < arr.length && err == Codes.SUCCESS) {

try {

arr[i] = Integer.parseInt(input.nextLine());

} catch (NumberFormatException e) {

err = Codes.INCORRECT\_DATA;

}

if (err == Codes.SUCCESS && (arr[i] > MAX\_NUMB || arr[i] < MIN\_NUMB)) {

err = Codes.OUT\_OF\_BORDER\_NUMB;

}

i++;

}

return err;

}

static int[] inputFromConsole(Scanner input) {

System.out.println("Enter the size[2, 100] and then the \n"

+ "elements[-2000000000, 2000000000] through the Enter");

Codes err;

int[] size = {0};

do {

err = inpValidSize(size, input);

if (err != Codes.SUCCESS) {

System.err.println(ERRORS[err.ordinal()]);

System.out.println("Please, enter again size");

}

} while (err != Codes.SUCCESS);

int[] defaultArr = new int[size[0]];

System.out.println("Enter the " + size[0] + " elements");

do {

err = inpValidArr(defaultArr, input);

if (err != Codes.SUCCESS) {

System.err.println(ERRORS[err.ordinal()]);

System.out.println("Please, enter again " + size[0] + " elements");

}

} while (err != Codes.SUCCESS);

return defaultArr;

}

static Codes readSizeFromFile(int[] size, Scanner file) {

Codes err;

int sizeInt;

err = Codes.SUCCESS;

sizeInt = 0;

try {

sizeInt = Integer.parseInt(file.next());

} catch (NumberFormatException e) {

err = Codes.INCORRECT\_DATA\_FILE;

}

if (err == Codes.SUCCESS)

if ((sizeInt > MAX\_SIZE) || (sizeInt < MIN\_SIZE))

err = Codes.OUT\_OF\_BORDER\_SIZE;

else

size[0] = sizeInt;

return err;

}

static Codes readArrFromFile(int[] arr, Scanner file){

Codes err;

int i;

err = Codes.SUCCESS;

i = 0;

while (i < arr.length && err == Codes.SUCCESS) {

try {

arr[i] = Integer.parseInt(file.next());

} catch (NumberFormatException e) {

err = Codes.INCORRECT\_DATA\_FILE;

}

if (err == Codes.SUCCESS && (arr[i] > MAX\_NUMB || arr[i] < MIN\_NUMB)) {

err = Codes.OUT\_OF\_BORDER\_NUMB;

}

i++;

}

if (i == arr.length && file.hasNextLine())

err = Codes.A\_LOT\_OF\_DATA\_FILE;

return err;

}

static Codes isFileExist(String fileName) {

File file = new File(fileName);

Codes err;

err = file.exists() ? Codes.SUCCESS : Codes.FILE\_NOT\_EXIST;

return err;

}

static Codes thisIsTxtFile(String fileName) {

Codes err;

err = fileName.endsWith(".txt") ? Codes.SUCCESS : Codes.NOT\_TXT;

return err;

}

static String getFileName(Scanner input) {

boolean isIncorrect;

String fileName;

Codes errTxt, errExist;

System.out.println("Enter full path to file");

do {

isIncorrect = false;

fileName = input.nextLine();

errTxt = thisIsTxtFile(fileName);

errExist = isFileExist(fileName);

if (errTxt != Codes.SUCCESS) {

isIncorrect = true;

System.err.println(ERRORS[errTxt.ordinal()]);

}

else if (errExist != Codes.SUCCESS) {

isIncorrect = true;

System.err.println(ERRORS[errExist.ordinal()]);

}

} while (isIncorrect);

return fileName;

}

static int[] inputFromFile(Scanner input){

Codes err;

int[] defaultArr = {};

err = Codes.SUCCESS;

int[] size = {0};

do {

String fileName = getFileName(input);

Path path = Paths.get(fileName);

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

err = readSizeFromFile(size, file);

defaultArr = new int[size[0]];

if (err == Codes.SUCCESS) {

err = readArrFromFile(defaultArr, file);

}

if (err != Codes.SUCCESS) {

System.err.println(ERRORS[err.ordinal()]);

System.out.println("Please, enter full path again");

}

} catch (IOException e) {

System.err.println(ERRORS[Codes.IN\_OUT\_FILE\_EXCEPTION.ordinal()]);

System.out.println("Please, enter full path again");

}

} while (err != Codes.SUCCESS);

System.out.println("Reading is successfull");

return defaultArr;

}

static int[] inputInf(Scanner input){

int choice = userChoice(input);

int[] borders = {0, 0};

if (choice == 1) {

borders = inputFromConsole(input);

} else {

borders = inputFromFile(input);

}

return borders;

}

static void writeInConsole(int[] defaultArr, int[] sortedArr) {

int i;

System.out.println("Default arr");

for (i = 0; i < defaultArr.length; i++)

System.out.print(defaultArr[i] + " ");

System.out.println("\nSorted arr");

for (i = 0; i < sortedArr.length; i++)

System.out.print(sortedArr[i] + " ");

}

static void writeInFile(int[] defaultArr, int[] sortedArr, Scanner input) {

boolean isIncorrect;

int i;

do {

String fileName = getFileName(input);

isIncorrect = false;

try (PrintWriter file = new PrintWriter(fileName)) {

file.println("Default arr");

for (i = 0; i < defaultArr.length; i++)

file.print(defaultArr[i] + " ");

file.println("\nSorted arr");

for (i = 0; i < sortedArr.length; i++)

file.print(sortedArr[i] + " ");

} catch (IOException e) {

isIncorrect = true;

System.err.println(ERRORS[Codes.IN\_OUT\_FILE\_EXCEPTION.ordinal()]);

}

} while (isIncorrect);

System.out.println("Writing is successfull");

}

static void outputInf(int[] defaultArr, int[] sortedArr, Scanner input){

int choice = userChoice(input);

if (choice == 1) {

writeInConsole(defaultArr, sortedArr);

} else {

writeInFile(defaultArr, sortedArr, input);

}

}

static int[] copyArr(int[] arr) {

int[] copyedArr = new int[arr.length];

int i;

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

copyedArr[i] = arr[i];

}

return copyedArr;

}

public static void main(String[] args) {

Scanner input = new Scanner(System.in);

int[] defaultArr;

int[] sortedArr;

printInf();

defaultArr = inputInf(input);

sortedArr = copyArr(defaultArr);

sortedArr = mergeSort(sortedArr);

outputInf(defaultArr, sortedArr, input);

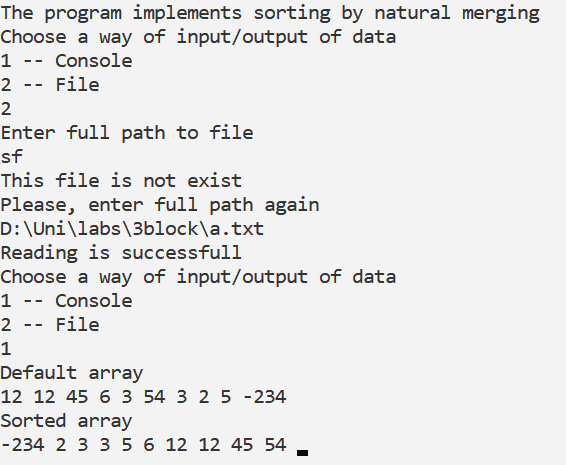
input.close();

}

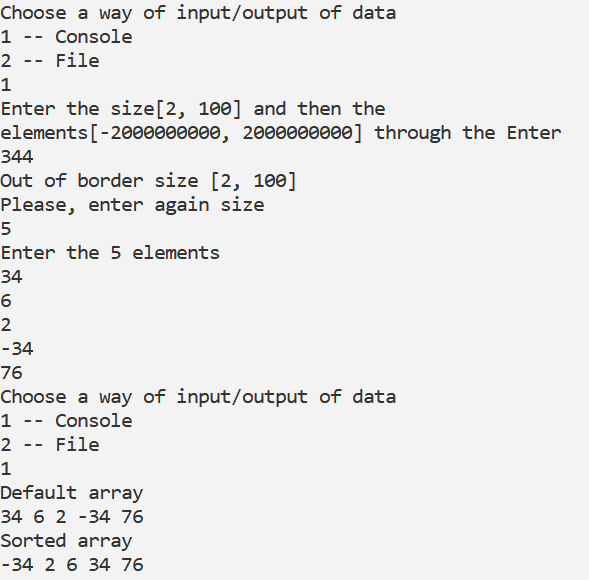
}

**Скриншоты:**

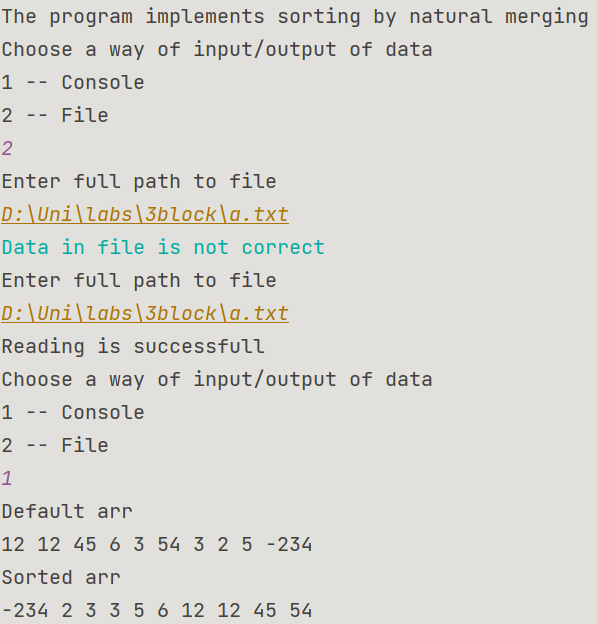
**Delphi:**

****

**C++:**

****

**Java:**

****

**Блок-схема:**





