УО «Белорусский государственный университет информатики и радиоэлектроники»

Кафедра ПОИТ

Отчет по лабораторной работе №3.2

по предмету «Основы алгоритмизации и программирования»

Вариант 20

Выполнил:

Машевский Д.В

Гр. 351003

Проверил:

Данилова Г. В.

Минск 2023

**Задание:**

Имеется множество, содержащее натуральные числа из некоторого диапазона. Сформировать два множества, первое из которых содержит все простые числа из данного множества, а второе все составные.

**Код Delphi:**

Program laba32;

Uses

SysUtils;

Type

TSet = Set Of Byte;

TBorderArr = Array [0 .. 1] Of Byte;

Const

MAX\_DIG = 255;

MIN\_DIG = 0;

Procedure GetInfo();

Begin

Writeln('This program forms two sets from the natural numbers: one with primes

and the other with composites.');

Writeln('Numbers should be in the interval [0, 255].');

End;

Function ChooseOption(): Integer;

Var

Choice: Integer;

IsCorrect: Boolean;

Begin

Choice := 0;

Writeln('Enter 0 to work with data from the console, and 1 to work from the

file:');

Repeat

IsCorrect:= True;

Try

Readln(Choice);

Except

IsCorrect:= False;

Writeln('Invalid data entered. Re-enter:');

End;

Until (IsCorrect) and ((Choice = 0) Or (Choice = 1));

ChooseOption := Choice;

End;

Function CorrectInputBorder(Var Number: Byte): Boolean;

Var

Line: String;

NumbInteg: Integer;

Begin

CorrectInputBorder := False;

Readln(Line);

Try

NumbInteg := StrToInt(Line);

If ((NumbInteg > MIN\_DIG) Or (NumbInteg = MIN\_DIG)) And

((NumbInteg < MAX\_DIG) Or (NumbInteg = MAX\_DIG)) Then

Begin

Number := NumbInteg;

CorrectInputBorder := True;

End

Else

Writeln('The number must be in the range from 0 to 255');

Except

Writeln('Invalid data entered. Re-enter:');

End;

End;

Function InputBorders(Var Borders: TBorderArr): Boolean;

Begin

InputBorders := True;

Writeln('Enter the borders through Enter:');

InputBorders := CorrectInputBorder(Borders[0]) And

CorrectInputBorder(Borders[1]);

If (Borders[0] > Borders[1]) Then

Begin

InputBorders := False;

Writeln('Incorrect borders');

End;

End;

Function IsCorrectBorderFromFile(Var Str: String; Var F: TextFile): Boolean;

Var

IsCorrect: Boolean;

Begin

IsCorrect := True;

If Not Eof(F) Then

Begin

Readln(F, Str);

If (Length(Str) = 0) Then

Begin

Writeln('Verify that the string you entered is correct');

IsCorrect := False;

End;

If (IsCorrect) and Not Eof(F) Then

Begin

Writeln('Remove extra data');

IsCorrect := False;

End;

End

Else

Begin

Writeln('No data for row');

IsCorrect := False;

End;

IsCorrectBorderFromFile := IsCorrect;

End;

Function PTF(): String;

Var

Path: String;

IsCorrect: Boolean;

Begin

Repeat

IsCorrect := True;

Write('Path to file: ');

Readln(Path);

If Not FileExists(Path) Then

Begin

Write('No such file was found. ');

IsCorrect := False;

End;

If IsCorrect And (ExtractFileExt(Path) <> '.txt') Then

Begin

IsCorrect := False;

Write('The file must have a txt extension.');

End;

Until IsCorrect;

PTF := Path;

End;

Function InputFromFile(Var Borders: TBorderArr; Var F: TextFile): TBorderArr;

Var

Str, Path: String;

IsCorrect: Boolean;

Begin

Path := PTF();

AssignFile(F, Path);

Reset(F);

Try

Readln(F, Borders[0]);

If Not Eof(F) Then

Begin

IsCorrect := IsCorrectBorderFromFile(Str, F);

Readln(F, Borders[1]);

End

Else

Begin

Writeln('No input data for the second border');

IsCorrect := False;

End;

Finally

CloseFile(F);

End;

InputFromFile := Borders;

End;

Function InputFromConsole(): TBorderArr;

Var

Borders: TBorderArr;

IsCorrect: Boolean;

Begin

IsCorrect := False;

Repeat

Try

If InputBorders(Borders) Then

IsCorrect := True

Else

Writeln('Invalid input. Re-enter:');

Except

Writeln('Invalid input. Re-enter:');

End;

Until IsCorrect;

InputFromConsole := Borders;

End;

Function Input(): TBorderArr;

Var

Borders: TBorderArr;

Choise: Integer;

F: TextFile;

Begin

Choise := ChooseOption();

If Choise = 0 Then

Begin

Borders := InputFromConsole();

End

Else

Borders := InputFromFile(Borders, F);

Input := Borders;

End;

Function CheckSimple(Number: Byte): Boolean;

Var

I: Byte;

Begin

CheckSimple := True;

If Number > 3 Then

Begin

For I := 2 To Trunc(Sqrt(Number)) Do

Begin

If Number Mod I = 0 Then

CheckSimple := False;

End;

End;

End;

Function MadeSetSimple(Start: TSet): TSet;

Var

Res: TSet;

Number: Byte;

Begin

Res := [];

For Number In Start Do

Begin

If CheckSimple(Number) Then

Begin

Include(Res, Number);

End;

End;

MadeSetSimple := Res;

End;

Function MadeSetComposit(Start, SimpleSet: TSet): TSet;

Begin

MadeSetComposit := Start - SimpleSet;

End;

Function MadeSet(Var Borders: TBorderArr): TSet;

Var

NumSet: TSet;

I: Byte;

Begin

NumSet := [];

For I := Borders[0] To Borders[1] Do

Begin

Include(NumSet, I);

End;

MadeSet := NumSet;

End;

Procedure OutputConsole(Start, SimpleSet, CompositSet: TSet);

Var

Number: Byte;

Begin

Writeln('Start set:');

For Number In Start Do

Write(Number, ' ');

Writeln(#13#10,'Set with simple numbers:');

For Number In SimpleSet Do

Write(Number, ' ');

Writeln(#13#10,'Set with composite numbers:');

For Number In CompositSet Do

Write(Number, ' ');

End;

Procedure OutputFile(Start, SimpleSet, CompositSet: TSet);

Var

FileName: String;

Number: Byte;

MyFile: TextFile;

Begin

FileName := PTF();

AssignFile(MyFile, FileName);

Rewrite(MyFile);

Write(MyFile, 'Start set', #13#10);

For Number In Start Do

Write(MyFile, Number, ' ');

Write(MyFile, #13#10, 'Set with simple numbers', #13#10, '{ ');

For Number In SimpleSet Do

Write(MyFile, Number, ' ');

Write(MyFile, #13#10, 'Set with composit numbers', #13#10, '{ ');

For Number In CompositSet Do

Write(MyFile, Number, ' ');

CloseFile(MyFile);

Writeln('Writing is successfull');

End;

Procedure Output(Start, SimpleSet, CompositSet: TSet);

Var

Choice: Integer;

Begin

Choice := ChooseOption();

If Choice = 0 Then

OutputConsole(Start, SimpleSet, CompositSet)

Else

OutputFile(Start, SimpleSet, CompositSet);

End;

Var

Borders: TBorderArr;

Start, SetSimple, SetComposit: TSet;

Begin

GetInfo();

Borders := Input();

Start := MadeSet(Borders);

SetSimple := MadeSetSimple(Start);

SetComposit := MadeSetComposit(Start, SetSimple);

Output(Start, SetSimple, SetComposit);

End.

**Код С++:**

#include <iostream>

#include <fstream>

#include <set>

#include <string>

using namespace std;

const int MAX\_DIG = 255;

const int MIN\_DIG = 0;

void getInfo() {

cout << "This program forms two sets from the natural numbers: one with primes

and the other with composites." << endl;

cout << "Numbers should be in the interval [0, 255]." << endl;

}

int chooseOption() {

int choice;

bool isInCorrect;

cout << "Enter 0 if you want to work with data from the console, and 1 if from

the file : ";

do {

isInCorrect = false;

cin >> choice;

if (cin.fail() || cin.peek() != '\n') {

isInCorrect = true;

cout << "Invalid data entered. Re-enter:";

cin.clear();

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

}

if (!(isInCorrect) && (choice != 1) && (choice != 0)) {

isInCorrect = true;

cout << "Re-enter: ";

cin.clear();

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

}

} while (isInCorrect);

return choice;

}

bool checkSimple(int number) {

int i;

if (number <= 1) return false;

if (number == 2) return true;

for (i = 2; i <= sqrt(number); i++)

{

if (number % i == 0)

{

return false;

}

}

return true;

}

set<int> madeSetSimple(set<int> start) {

set<int> res;

for (int number : start)

{

if (checkSimple(number))

{

res.insert(number);

}

}

return res;

}

set<int> madeSetComposite(set<int> start, set<int> simpleSet) {

set<int> res;

for (int number : start)

{

if (simpleSet.find(number) == simpleSet.end())

{

res.insert(number);

}

}

return res;

}

set<int> madeSet(int start, int end) {

set<int> numSet;

int i;

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

{

numSet.insert(i);

}

return numSet;

}

void outputConsole(set<int> start, set<int> simpleSet, set<int> compositeSet) {

cout << "Default set:" << endl;

for (int number : start)

{

cout << number << " ";

}

cout << "\nSet with simple numbers:" << endl;

for (int number : simpleSet)

{

cout << number << " ";

}

cout << "\nSet with composite numbers:" << endl;

for (int number : compositeSet)

{

cout << number << " ";

}

}

string pTF() {

string path;

bool isInСorrect;

do

{

isInСorrect = false;

cout << "Path to the file: ";

cin >> path;

ifstream fin(path);

if (!fin.is\_open())

{

cout << "No such file was found. ";

isInСorrect = true;

}

fin.close();

} while (isInСorrect);

return path;

}

void readFile(int& start, int& end, string fileName) {

ifstream file(fileName);

bool isInCorrect;

string str, path;

path = pTF();

file.open(path);

isInCorrect = false;

file >> start;

if (!file.eof()) {

file >> end;

}

else {

cout << "No input data for the second border" << endl;

}

file.close();

}

void inputRangeFromConsole(int& start, int& end) {

bool isInCorrect;

cout << "Enter the borders:" << endl;

do {

cin >> start >> end;

isInCorrect = false;

if (cin.fail() || cin.peek() != '\n') {

isInCorrect = true;

cout << "Invalid data entered. Re-enter:";

cin.clear();

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

}

if (start < MIN\_DIG || end > MAX\_DIG || start > end) {

isInCorrect = true;

cout << "Invalid data entered. Re-enter:";

cin.clear();

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

}

} while (isInCorrect);

}

void inputRange(int& start, int& end, int choice, const string& fileName) {

if (choice == 0)

{

inputRangeFromConsole(start, end);

}

else

{

readFile(start, end, fileName);

}

}

void outputFile(set<int> start, set<int> simpleSet, set<int> compositeSet, string path) {

ofstream outFile(path);

outFile << "Default set:" << endl;

for (int number : start)

{

outFile << number << " ";

}

outFile << "\nSet with simple numbers:" << endl;

for (int number : simpleSet)

{

outFile << number << " ";

}

outFile << "\nSet with composite numbers:" << endl;

for (int number : compositeSet)

{

outFile << number << " ";

}

outFile.close();

}

void outPut(set<int> startSet, set<int> simpleSet, set<int> compositeSet, int choice, string path) {

choice = chooseOption();

if (choice == 0)

{

outputConsole(startSet, simpleSet, compositeSet);

}

else

{

outputFile(startSet, simpleSet, compositeSet, path);

}

}

int main() {

getInfo();

int choice = chooseOption();

int start, end;

string path;

string fileName;

inputRange(start, end, choice, fileName);

set<int> startSet = madeSet(start, end);

set<int> simpleSet = madeSetSimple(startSet);

set<int> compositeSet = madeSetComposite(startSet, simpleSet);

outPut(startSet, simpleSet, compositeSet, choice, path);

return 0;

}

**Код Java:**

import java.io.File;

import java.io.FileWriter;

import java.io.IOException;

import java.util.HashSet;

import java.util.Scanner;

import java.util.Set;

public class Main {

private static final int MAX\_DIG = 255;

private static final int MIN\_DIG = 0;

public static void main(String[] args) {

int choice;

System.out.println("This program forms two sets from the natural numbers: one

with primes and the other with composites.");

System.out.println("Numbers should be in the interval [0, 255].");

choice = chooseOption();

TBorderArr borders;

if (choice == 0) {

borders = inputFromConsole();

} else {

borders = inputFromFile();

}

Set<Byte> start = madeSet(borders);

Set<Byte> setSimple = madeSetSimple(start);

Set<Byte> setComposite = madeSetComposit(start, setSimple);

output(choice, start, setSimple, setComposite);

}

private static int chooseOption() {

int choice;

choice=0;

Scanner scanner = new Scanner(System.in);

do {

System.out.println("Enter 0 to work with data from the console, and 1 to

work from the file:");

try {

choice = scanner.nextInt();

} catch (Exception e) {

System.out.println("Invalid data entered. Re-enter:");

scanner.nextLine();

continue;

}

} while (choice != 0 && choice != 1);

return choice;

}

private static TBorderArr inputFromConsole() {

TBorderArr borders;

borders = null;

boolean inIsCorrect;

inIsCorrect = false;

Scanner scanner = new Scanner(System.in);

do {

try {

System.out.println("Enter the borders through Enter:");

byte border1 = inpValidBorder(scanner);

byte border2 = inpValidBorder(scanner);

if (border1 > border2) {

System.out.println("Incorrect borders");

continue;

}

borders = new TBorderArr(border1, border2);

inIsCorrect = true;

} catch (Exception e) {

System.out.println("Invalid input. Re-enter:");

}

} while (!inIsCorrect);

return borders;

}

private static byte inpValidBorder(Scanner scanner) {

byte number;

number = 0;

boolean isInCorrect;

do {

isInCorrect = false;

try {

number = scanner.nextByte();

if ((number >= MIN\_DIG) && (number <= MAX\_DIG)) {

return number;

} else {

System.out.println("The number must be in the range from 0 to

255");

isInCorrect = true;

}

} catch (Exception e) {

System.out.println("Invalid data entered. Re-enter:");

isInCorrect = true;

scanner.nextLine();

}

} while (isInCorrect);

return number;

}

private static TBorderArr inputFromFile() {

TBorderArr borders;

borders = null;

boolean isInCorrect;

Scanner scanner = new Scanner(System.in);

do {

isInCorrect = false;

System.out.print("Path to file: ");

String path = scanner.nextLine();

if (!path.endsWith(".txt")) {

System.out.println("The file must have a txt extension.");

isInCorrect = true;

continue;

}

try {

scanner = new Scanner(new File(path));

byte border1 = Byte.parseByte(scanner.nextLine());

String str = "";

if (!scanner.hasNextLine()) {

System.out.println("No input data for the second border");

isInCorrect = true;

continue;

}

if (!isCorrectBorderFromFile(str, scanner)) {

isInCorrect = true;

continue;

}

byte border2 = Byte.parseByte(scanner.nextLine());

borders = new TBorderArr(border1, border2);

} catch (Exception e) {

System.out.println("Invalid data in the file");

isInCorrect = true;

}

} while (isInCorrect);

return borders;

}

private static boolean isCorrectBorderFromFile(String str, Scanner scanner) {

boolean isCorrect = true;

if (scanner.hasNextLine()) {

str = scanner.nextLine();

if (str.isEmpty()) {

System.out.println("Verify that the string you entered is correct");

isCorrect = false;

}

if (scanner.hasNextLine()) {

System.out.println("Remove extra data");

isCorrect = false;

}

} else {

System.out.println("No data for row");

isCorrect = false;

}

return isCorrect;

}

private static Set<Byte> madeSet(TBorderArr borders) {

Set<Byte> numSet = new HashSet<>();

byte i;

for (i = borders.getBorder1(); i <= borders.getBorder2(); i++) {

numSet.add(i);

}

return numSet;

}

private static Set<Byte> madeSetSimple(Set<Byte> start) {

Set<Byte> res = new HashSet<>();

for (byte number : start) {

if (checkSimple(number)) {

res.add(number);

}

}

return res;

}

private static boolean checkSimple(byte number) {

byte i;

if (number <= 3) {

return true;

}

for (i = 2; i <= Math.sqrt(number); i++) {

if (number % i == 0) {

return false;

}

}

return true;

}

private static Set<Byte> madeSetComposit(Set<Byte> start, Set<Byte> simpleSet) {

Set<Byte> compositeSet = new HashSet<>(start);

compositeSet.removeAll(simpleSet);

return compositeSet;

}

private static void output(int choice, Set<Byte> start, Set<Byte> setSimple, Set<Byte> setComposite) {

if (choice == 0) {

outputConsole(start, setSimple, setComposite);

} else {

outputFile(start, setSimple, setComposite);

}

}

private static void outputConsole(Set<Byte> start, Set<Byte> setSimple, Set<Byte> setComposite) {

System.out.println("Default set:");

for (byte number : start) {

System.out.print(number + " ");

}

System.out.println("\nSet with simple numbers:");

for (byte number : setSimple) {

System.out.print(number + " ");

}

System.out.println("\nSet with composite numbers:");

for (byte number : setComposite) {

System.out.print(number + " ");

}

}

private static void outputFile(Set<Byte> start, Set<Byte> setSimple, Set<Byte> setComposite) {

Scanner scanner = new Scanner(System.in);

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

String fileName = scanner.nextLine();

try (FileWriter writer = new FileWriter(fileName)) {

writer.write("Default set:\n{ ");

for (byte number : start) {

writer.write(number + " ");

}

writer.write("}\nSet with simple numbers:\n{ ");

for (byte number : setSimple) {

writer.write(number + " ");

}

writer.write("}\nSet with composite numbers:\n{ ");

for (byte number : setComposite) {

writer.write(number + " ");

}

writer.write("}");

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

} catch (IOException e) {

System.out.println("Error writing to file");

}

}

private static class TBorderArr {

private final byte border1;

private final byte border2;

public TBorderArr(byte border1, byte border2) {

this.border1 = border1;

this.border2 = border2;

}

public byte getBorder1() {

return border1;

}

public byte getBorder2() {

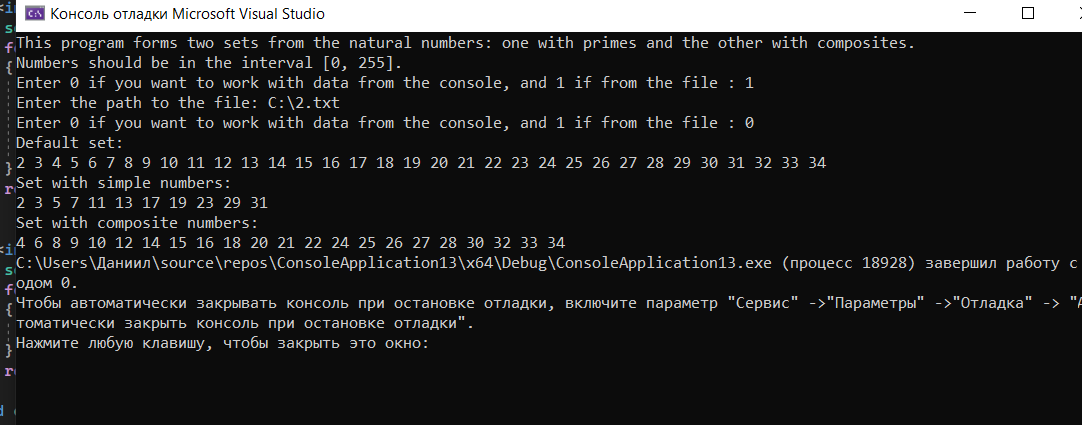
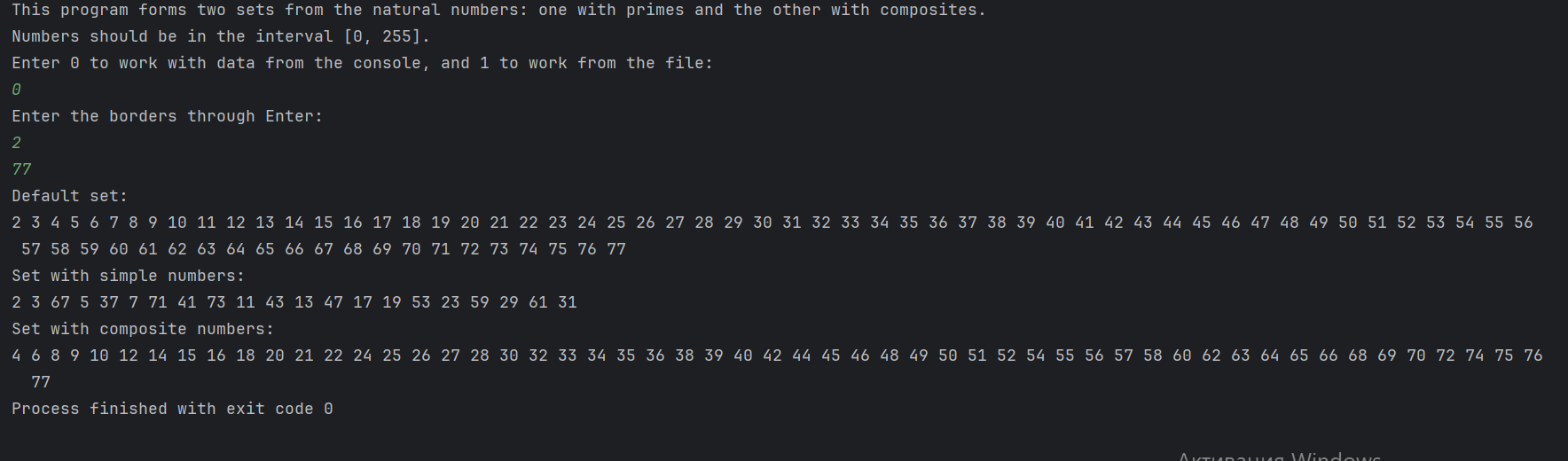
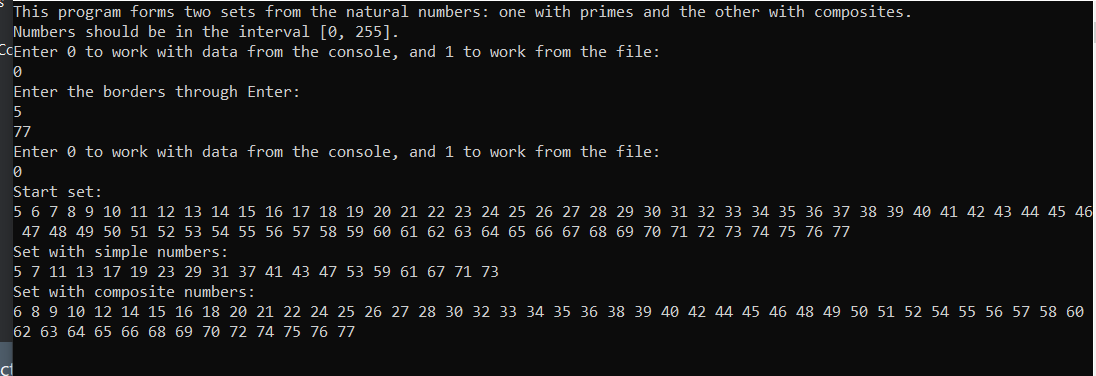
return border2;

}

}

}

**Скриншоты:**



**Блок-схема:**

