УО «Белорусский государственный университет информатики и радиоэлектроники»

Кафедра ПОИТ

Отчет по лабораторной работе №3.2

по предмету

Основы алгоритмизации и программирования

Вариант 3

Выполнил:

Бетеня К.С.

Проверила:

Данилова Г.В.

Группа 351005

Минск 2023

Задание:

Разработать программу решения задачи с использованием процедур и функций над строками и операций над множествами. Дана непустая последовательность символов, требуется построить и напечатать множество, элементами которого являются встречающиеся в последовательности знаки арифметических операций и числа.

Код программы на **Delphi**:

Program Lab3;

{$APPTYPE CONSOLE}

{$R \*.res}

Uses

System.SysUtils;

Const

MIN\_FILE\_WAY\_SIZE = 5;

MIN\_SIZE = 1;

GOOD\_SIZE = 1;

BAD\_SIZE = 2;

FILE\_KEY = 1;

CONSOLE\_KEY = 2;

Type

TIOError = (INVALID\_PATH, METHOD\_ERROR, SHORT\_PATH\_ERROR, TXT\_ERROR,

OPEN\_FILE\_ERROR, MIN\_SIZE\_ERROR, FIRST\_STR\_ERROR, EL\_ERROR,

TRY\_AGAIN);

TArrSize = Array Of Integer;

TArrOfElements = Array Of Char;

TAnsiChar = Set Of AnsiChar;

const

Entitlements: TAnsiChar = ['0', '1', '2', '3', '4', '5', '6',

'7', '8', '9', '+', '-', '\*', '/'];

Const

ERRORS: Array [TIOError] Of String =

('Error. You should write a natural number.', 'Error method.',

'The path is too short.', 'Write .txt file.', 'Can not open a file.',

'Min number of elements is 1.', 'First string is natural number.',

'Enter a specific number of characters.', ' Try again: ');

Procedure PrintError(IOErrorMethod: String);

Begin

Write(IOErrorMethod + ERRORS[TRY\_AGAIN]);

End;

//block of text output

Procedure TaskOutput();

Begin

Writeln('the program builds and prints a set, the elements', #13#10,

'of which are the signs of arithmetic operations and', #13#10,

'numbers occurring in the sequence.', #13#10);

End;

Procedure WorkWayConditionOutput();

Begin

Writeln('Where will we work through: ', #13#10#9, 'File: ',

FILE\_KEY, #10#13#9, 'Console: ', CONSOLE\_KEY, #13#10);

End;

Procedure FileRestriction();

Begin

Write(#13#10, '1. The first line in the file is a natural number -',

#13#10, 'N characters of the second line;',

#13#10, '2. The second line is N characters entered by the user.',

#13#10, 'Write way to your file: ');

End;

//choice of direction

Function ChoosingWorkWay(): Integer;

Var

Path: Integer;

IsCorrect, IsCorrectInput: Boolean;

Begin

WorkWayConditionOutput();

Path := 0;

IsCorrect := False;

Write('Please write were we should work: ');

Repeat

IsCorrectInput := False;

Try

Readln(Path);

IsCorrectInput := True;

Except

PrintError(ERRORS[INVALID\_PATH]);

End;

If ((Path = CONSOLE\_KEY) Or (Path = FILE\_KEY)) And IsCorrectInput Then

Begin

IsCorrect := True;

End

Else

If IsCorrectInput Then

Begin

PrintError(ERRORS[METHOD\_ERROR]);

End;

Until IsCorrect;

ChoosingWorkWay := Path;

End;

//input and check path to the file

Function PathCondition(Way: String): Boolean;

Var

IsCorrect: Boolean;

Begin

IsCorrect := False;

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

Begin

Write('Write .txt file. Try again: ');

End

Else

Begin

IsCorrect := True;

End;

PathCondition := IsCorrect;

End;

Function InputPath(): String;

Var

Way: String;

IsCorrect: Boolean;

Begin

Repeat

Readln(Way);

IsCorrect := PathCondition(Way);

Until IsCorrect;

InputPath := Way;

End;

Function InputPathToTheFile(): String;

Var

FileWay: String;

IsCorrect: Boolean;

Begin

IsCorrect := False;

FileWay := '';

Repeat

FileWay := InputPath();

If (Not FileExists(FileWay)) Then

Write('Can not open a file. Try write another way: ')

Else

IsCorrect := True;

Until IsCorrect;

InputPathToTheFile := FileWay;

End;

///input from console

Function CheckSizeCondition(Size: Integer; IsCorrectInput: Boolean): Boolean;

Begin

If (Size < MIN\_SIZE) And (IsCorrectInput) Then

Begin

PrintError(ERRORS[MIN\_SIZE\_ERROR]);

IsCorrectInput := False;

End;

CheckSizeCondition := IsCorrectInput;

End;

Function InputSizeFromConsole(): Integer;

Var

Size: Integer;

IsCorrect, IsCorrectInput: Boolean;

Begin

Size := 0;

Write('How many characters do you want to enter: ');

Repeat

IsCorrectInput := False;

Try

Readln(Size);

IsCorrectInput := True;

Except

PrintError(ERRORS[INVALID\_PATH])

End;

IsCorrect := CheckSizeCondition(Size, IsCorrectInput);

Until IsCorrect;

InputSizeFromConsole := Size;

End;

Function IsCorrectElementsInputFromConsole(Size: Integer; Str: String): Boolean;

Var

Res: Boolean;

Begin

Res := False;

If Size = Str.Length Then

Begin

Res := True;

End;

IsCorrectElementsInputFromConsole := Res;

End;

Function InputStringFromConsole(ArrSize: TArrSize): TArrOfElements;

Var

Size, I: Integer;

IsCorrect: Boolean;

ArrOfElements: TArrOfElements;

Str: String;

Begin

Size := InputSizeFromConsole;

ArrSize[0] := Size;

SetLength(ArrOfElements, Size);

Write('Write your ', Size, ' elements: ');

Repeat

Readln(Str);

If Str.Length = Size Then

For I := 1 To Size Do

ArrOfElements[I - 1] := Str[I]

Else

PrintError(ERRORS[EL\_ERROR]);

IsCorrect := IsCorrectElementsInputFromConsole(Size, Str);

Until IsCorrect;

InputStringFromConsole := ArrOfElements;

End;

//input from file

Function CheckSizeInputFromFile(Var MyFile: TextFile; Size: Integer): Integer;

Var

Res: Integer;

Begin

Res := GOOD\_SIZE;

If Size < MIN\_SIZE Then

Begin

PrintError(ERRORS[MIN\_SIZE\_ERROR]);

Res := BAD\_SIZE;

End;

CheckSizeInputFromFile := Res;

End;

Function InputSizeFromFile(Var MyFile: TextFile; Var ArrSize: TArrSize): Integer;

Var

Size: Integer;

Begin

Readln(MyFile, Size);

ArrSize[1] := CheckSizeInputFromFile(MyFile, Size);

InputSizeFromFile := Size;

End;

Function IsCorrectElInputFromFile(Str: String; Size: Integer): Boolean;

Var

Res: Boolean;

Begin

Res := True;

If (Str.Length <> Size) Then

Begin

PrintError(ERRORS[EL\_ERROR]);

Res := False;

End;

IsCorrectElInputFromFile := Res;

End;

Function InputSetFromFile(Var ArrOfElements: TArrOfElements; Size: Integer; Var MyFile: TextFile): Boolean;

Var

Counter: Char;

Str: String;

I: Integer;

Begin

Str := '';

Repeat

Read(MyFile, Counter);

If (Counter <> #$D) And (Counter <> #$1A) Then

Begin

Str := Str + Counter;

End

Else

Begin

Read(MyFile, Counter);

End;

Until (Counter = #$A) Or (Counter = #$1A);

If Str.Length = Size Then

For I := 1 To Size Do

ArrOfElements[I - 1] := Str[I];

InputSetFromFile := IsCorrectElInputFromFile(Str, Size);

End;

Function InputStringFromFile(Var ArrSize: TArrSize): TArrOfElements;

Var

ArrOfElements: TArrOfElements;

IsCorrect: Boolean;

FileWay: String;

MyFile: TextFile;

Begin

FileRestriction();

IsCorrect := False;

Repeat

FileWay := InputPathToTheFile();

Try

AssignFile(MyFile, FileWay);

Try

Reset(MyFile);

ArrSize[0] := InputSizeFromFile(MyFile, ArrSize);

If (ArrSize[1] = GOOD\_SIZE) Then

Begin

SetLength(ArrOfElements, ArrSize[0]);

IsCorrect := InputSetFromFile(ArrOfElements, ArrSize[0],

MyFile);

End;

Finally

Close(MyFile);

End;

Except

Write('Bad strings input from file. Try again: ');

End;

Until IsCorrect;

InputStringFromFile := ArrOfElements;

End;

//making the set

Procedure RenderingSet(ArrOfElements: TArrOfElements; Var ResultSet: TAnsiChar);

Var

I: Integer;

Current : Char;

Begin

For I := 0 To High(ArrOfElements) Do

for Current in Entitlements do

If (Current = ArrOfElements[i]) Then

Include(ResultSet, AnsiChar(ArrOfElements[I]));

End;

//output from file

Function OutputSetInFile(ResultSet: TAnsiChar): String;

Var

Str: String;

Current: Char;

Begin

Str := '';

For Current In ResultSet Do

Str := '''' + Current + ''';';

OutputSetInFile := Str;

End;

Function OutputResInFile(ResultSet: TAnsiChar): String;

Var

Res: String;

Begin

Res := 'Your result is: ';

If ResultSet = [] Then

Res := Res + ' empty set.'

Else

Res := Res + OutputSetInFile(ResultSet);

OutputResInFile := Res;

End;

Procedure OutputFromFile(ResultSet: TAnsiChar);

Var

IsCorrect: Boolean;

FileWay, Result: String;

MyFile: TextFile;

Begin

IsCorrect := False;

Write('Write way to your file: ');

Repeat

FileWay := InputPathToTheFile();

Try

AssignFile(MyFile, FileWay);

Try

Append(MyFile);

ReWrite(MyFile);

Result := OutputResInFile(ResultSet);

Writeln(MyFile, Result);

Write('Cheack your file.');

IsCorrect := True;

Finally

Close(MyFile);

End;

Except

Write('Bad output file. Try again: ');

End;

Until IsCorrect;

End;

//output from console

Procedure OutputSetFromConsole(Var ResultSet: TAnsiChar);

Var

Current: AnsiChar;

Begin

For Current In ResultSet Do

Write('''', Current, ''';');

End;

Procedure OutputFromConsole(Var ResultSet: TAnsiChar);

Begin

Write('The result is: ');

If ResultSet = [] Then

Write(' empty set.')

Else

OutputSetFromConsole(ResultSet);

End;

//distributive output

Procedure ResultOutputSystem(ResultSet: TAnsiChar);

Var

Path: Integer;

Begin

Writeln(#13#10, 'You need to choose where to write information from.');

Path := ChoosingWorkWay();

If Path = CONSOLE\_KEY Then

OutputFromConsole(ResultSet)

Else

OutputFromFile(ResultSet);

End;

//main dustributive func

Function InputSystem(Var ArrSize: TArrSize): TArrOfElements;

Var

Path: Integer;

Begin

Path := ChoosingWorkWay();

If Path = CONSOLE\_KEY Then

InputSystem := InputStringFromConsole(ArrSize)

Else

InputSystem := InputStringFromFile(ArrSize);

End;

//cleaning func

Procedure ClearMemory(Var ArrSize: TArrSize; Var ArrOfElements: TArrOfElements);

Begin

ArrSize := Nil;

ArrOfElements := Nil;

End;

Var

ArrSize: TArrSize;

ArrOfElements: TArrOfElements;

ResultSet: TAnsiChar;

Begin

TaskOutput();

SetLength(ArrSize, 2);

SetLength(ArrOfElements, 0);

ArrOfElements := InputSystem(ArrSize);

RenderingSet(ArrOfElements, ResultSet);

ResultOutputSystem(ResultSet);

ClearMemory(ArrSize, ArrOfElements);

End.

Код программы на **C++**:

#include <iostream>

#include <fstream> // for work with .txt file

#include <string> // for file way reading

#include <set> // for making the result

using namespace std;

const int MIN\_FILE\_WAY\_SIZE = 5;

const int MIN\_SIZE = 1;

const int GOOD\_SIZE = 1;

const int BAD\_SIZE = 2;

const int FILE\_KEY = 1;

const int CONSOLE\_KEY = 2;

const set<char> Entitlements =

{

'0', '1', '2', '3', '4', '5', '6',

'7', '8', '9', '+', '-', '=', '/'

};

// block of work with errors

enum IOError

{

INVALID\_PATH,

METHOD\_ERROR,

SHORT\_PATH\_ERROR,

TXT\_ERROR,

OPEN\_FILE\_ERROR,

MIN\_SIZE\_ERROR,

FIRST\_STR\_ERROR,

EL\_ERROR,

TRY\_AGAIN

};

const string ERRORS[]

{

"Error. You should write a natural number.",

"Error method.",

"The path is too short.",

"Write .txt file.",

"Can't open a file.",

"Min number of elements is " + to\_string(MIN\_SIZE) + ".",

"First string is natural number.",

"Enter a specific number of characters.",

" Try again: "

};

void printError(string IOErrorMethod)

{

cerr << IOErrorMethod << ERRORS[(int)IOError::TRY\_AGAIN];

}

// block of text output

void taskOutput()

{

cout << "the program builds and prints a set, the elements\n"

"of which are the signs of arithmetic operations and\n"

"numbers occurring in the sequence.\n\n";

}

void workWayConditionOutput()

{

cout << "Where will we work through: \n\tFile: " << FILE\_KEY <<

"\n\tConsole: " << CONSOLE\_KEY << endl << endl;

}

void fileRestriction()

{

cout << "\n1. The first line in the file is a natural number -\n"

"N characters of the second line;\n"

"2. The second line is N characters entered by the user.\n"

"Write way to your file: ";

}

// choice of direction

int choosingWorkWay()

{

workWayConditionOutput();

int path = 0;

bool isIncorrect = true;

cout << "Please write were we should work: ";

do

{

cin >> path;

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

{

printError(ERRORS[(int)IOError::INVALID\_PATH]);

cin.clear();

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

}

else

{

if (path == CONSOLE\_KEY || path == FILE\_KEY)

{

isIncorrect = false;

}

else

{

printError(ERRORS[(int)IOError::METHOD\_ERROR]);

}

}

} while (isIncorrect);

return path;

}

// input and check path to the file

bool pathCondition(string way)

{

if (way.size() < MIN\_FILE\_WAY\_SIZE)

{

printError(ERRORS[(int)IOError::SHORT\_PATH\_ERROR]);

return false;

}

string bufstr = way.substr(way.size() - 4);

if (bufstr != ".txt")

{

printError(ERRORS[(int)IOError::TXT\_ERROR]);

return false;

}

return true;

}

string inputPath()

{

string way;

bool isIncorrect;

do

{

cin >> way;

isIncorrect = !pathCondition(way);

} while (isIncorrect);

return way;

}

bool isCanOpenFile(string way, ios\_base::openmode mode)

{

fstream file(way, mode);

file.close();

return file.good();

}

string inputPathToTheFile()

{

string fileWay;

bool isIncorrect = true;

do

{

fileWay = inputPath();

if (!isCanOpenFile(fileWay, ios::in))

{

printError(ERRORS[(int)IOError::OPEN\_FILE\_ERROR]);

}

else

{

isIncorrect = false;

}

} while (isIncorrect);

return fileWay;

}

/// input from console

bool checkSizeCondition(int size)

{

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

{

printError(ERRORS[(int)IOError::INVALID\_PATH]);

cin.clear();

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

}

else if (size < MIN\_SIZE)

{

printError(ERRORS[(int)IOError::MIN\_SIZE\_ERROR]);

}

else

{

return false;

}

return true;

}

int inputSizeFromConsole()

{

int size;

bool isIncorrect;

cout << "How many characters do you want to enter: ";

do

{

cin >> size;

isIncorrect = checkSizeCondition(size);

} while (isIncorrect);

return size;

}

bool isCorrectElementsInputFromConsole()

{

if (cin.peek() != '\n')

{

printError(ERRORS[(int)IOError::EL\_ERROR]);

cin.clear();

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

}

else

{

return false;

}

return true;

}

char\* inputStringFromConsole(int\*& arrSize)

{

bool isIncorrect;

int size = inputSizeFromConsole();

char\* arrOfElements = new char[size];

arrSize[0] = size;

cout << "Write your " << size << " elements: ";

do

{

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

{

cin >> arrOfElements[i];

}

isIncorrect = isCorrectElementsInputFromConsole();

} while (isIncorrect);

return arrOfElements;

}

// input from file

int checkSizeInputFromFile(ifstream& file, int size)

{

if (file.fail())

{

printError(ERRORS[(int)IOError::FIRST\_STR\_ERROR]);

return BAD\_SIZE;

}

else if (size < MIN\_SIZE)

{

printError(ERRORS[(int)IOError::MIN\_SIZE\_ERROR]);

return BAD\_SIZE;

}

return GOOD\_SIZE;

}

int inputSizeFromFile(ifstream& file, int& arrSize)

{

int size;

file >> size;

arrSize = checkSizeInputFromFile(file, size);

return size;

}

bool isCorrectElInputFromFile(int i, int size)

{

if (i != size)

{

printError(ERRORS[(int)IOError::EL\_ERROR]);

return true;

}

return false;

}

bool inputSetFromFile(char\*& arrOfElements, int size, ifstream& file)

{

char counter;

int i = 0;

while (file.get(counter))

{

if (counter != '\n')

{

if (i <= size)

{

arrOfElements[i] = counter;

}

i++;

}

}

return isCorrectElInputFromFile(i, size);

}

char\* inputStringFromFile(int\*& arrSize)

{

fileRestriction();

char\* arrOfElements = new char[0];

bool isIncorrect = true;

do

{

string fileWay = inputPathToTheFile();

ifstream file(fileWay, ios::in);

arrSize[0] = inputSizeFromFile(file, arrSize[1]);

if (arrSize[1]) {

arrOfElements = new char[arrSize[0]];

isIncorrect = inputSetFromFile(arrOfElements, arrSize[0],

file);

}

file.close();

} while (isIncorrect);

return arrOfElements;

}

// making the set

void renderingSet(char\* arrOfElements, int size, set<char>& resultSet)

{

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

{

for (char current : Entitlements)

{

if (arrOfElements[i] == current)

{

resultSet.insert(arrOfElements[i]);

}

}

}

}

// output from file

void outputSetInFile(set<char> resultSet, ofstream& file)

{

for (auto currentElement : resultSet)

{

file << " '" << currentElement << "';";

}

}

void outputResInFile(set<char> resultSet, string fileWay)

{

ofstream file(fileWay, ios::out);

file << "The result is:";

if (resultSet.empty())

{

file << " empty set.";

}

else

{

outputSetInFile(resultSet, file);

}

file.close();

}

void outputFromFile(set<char> resultSet)

{

bool isIncorrect = true;

cout << "Write way to your file: ";

do

{

string fileWay = inputPathToTheFile();

if (isCanOpenFile(fileWay, ios::out))

{

outputResInFile(resultSet, fileWay);

cout << "Check your file.";

isIncorrect = false;

}

else

{

printError(ERRORS[(int)IOError::OPEN\_FILE\_ERROR]);

}

} while (isIncorrect);

}

// output from console

void outputSetFromConsole(set<char> resultSet)

{

for (auto currentElement : resultSet)

{

cout << " '" << currentElement << "';";

}

}

void outputFromConsole(set<char> resultSet)

{

cout << "The result is:";

if (resultSet.empty())

{

cout << " empty set.";

}

else

{

outputSetFromConsole(resultSet);

}

}

// distributive output

void resultOutputSystem(set<char> resultSet)

{

cout << "\nYou need to choose where to write information from.\n";

int path = choosingWorkWay();

path == CONSOLE\_KEY ? outputFromConsole(resultSet)

: outputFromFile(resultSet);

}

// main distributive func

char\* inputSystem(int\*& arrSize)

{

int path = choosingWorkWay();

return path == CONSOLE\_KEY ? inputStringFromConsole(arrSize)

: inputStringFromFile(arrSize);

}

// cleaning func

void clearMemory(int\*& arrSize, char\*& arrOfElements, set<char> resultSet)

{

arrOfElements = nullptr;

delete[] arrSize;

arrSize = nullptr;

resultSet.clear();

}

//main

int main()

{

taskOutput();

int\* arrSize = new int[2];

char\* arrOfElements = inputSystem(arrSize);

set<char> resultSet;

renderingSet(arrOfElements, arrSize[0], resultSet);

resultOutputSystem(resultSet);

clearMemory(arrSize, arrOfElements, resultSet);

return 0;

}

Код программы на **Java**:

package lab3;  
  
import java.io.File;  
import java.io.FileWriter;  
import java.util.\*;  
  
public class Lab2 {  
 static Scanner *in* = new Scanner(System.*in*);  
 static final int *MIN\_FILE\_WAY\_SIZE* = 5;  
 static final int *MIN\_SIZE* = 1;  
 static final int *GOOD\_SIZE* = 1;  
 static final int *BAD\_SIZE* = 2;  
 static final int *FILE\_KEY* = 1;  
 static final int *CONSOLE\_KEY* = 2;  
  
 // block of work with errors  
 public enum IOError  
 {  
 *INVALID\_PATH*,  
 *METHOD\_ERROR*,  
 *SHORT\_PATH\_ERROR*,  
 *TXT\_ERROR*,  
 *OPEN\_FILE\_ERROR*,  
 *MIN\_SIZE\_ERROR*,  
 *FIRST\_STR\_ERROR*,  
 *EL\_ERROR*,  
 *TRY\_AGAIN* }  
 static final String[] *ERRORS* =  
 {  
 "Error. You should write a natural number.",  
 "Error method.",  
 "The path is too short.",  
 "Write .txt file.",  
 "Can't open a file.",  
 "Min number of elements is " + MIN\_SIZE + ".",  
 "First string is natural number.",  
 "Enter a specific number of characters.",  
 " Try again: "  
 };  
 static void printError(String IOErrorMethod)  
 {  
 System.err.print(IOErrorMethod + ERRORS[IOError.TRY\_AGAIN.ordinal()]);  
 }  
 // block of text output  
 static void taskOutput()  
 {  
 System.out.println("""  
 the program builds and prints a set, the elements  
 of which are the signs of arithmetic operations and  
 numbers occurring in the sequence.  
 """);  
 }  
 static void workWayConditionOutput()  
 {  
 System.out.printf("""  
 Where will we work through:  
 File: %d  
 Console: %d  
   
 """, FILE\_KEY, CONSOLE\_KEY);  
 }  
 static void fileRestriction()  
 {  
 System.out.print("""  
   
 1. The first line in the file is a natural number -  
 N characters of the second line;  
 2. The second line is N characters entered by the user.  
 Write way to your file:\s""");  
 }  
 // choice of direction  
 static int choosingWorkWay()  
 {  
 workWayConditionOutput();  
  
 int path = 0;  
 boolean isIncorrect = true;  
 System.out.print("Please write were we should work: ");  
 do  
 {  
 boolean isCorrectInput = false;  
 try {  
 path = Integer.parseInt(in.nextLine());  
 isCorrectInput = true;  
 } catch(Exception error){  
 printError(ERRORS[IOError.INVALID\_PATH.ordinal()]);  
 }  
 if (path == CONSOLE\_KEY || path == FILE\_KEY)  
 {  
 isIncorrect = false;  
 }  
 else if (isCorrectInput)  
 {  
 printError(ERRORS[IOError.METHOD\_ERROR.ordinal()]);  
 }  
 } while (isIncorrect);  
  
 return path;  
 }  
 // input and check path to the file  
 static boolean pathCondition(String way)  
 {  
 if (way.length() < MIN\_FILE\_WAY\_SIZE)  
 {  
 printError(ERRORS[IOError.SHORT\_PATH\_ERROR.ordinal()]);  
 return false;  
 }  
 String bufstr = way.substring(way.length() - 4);  
 if (!bufstr.equals(".txt"))  
 {  
 printError(ERRORS[IOError.TXT\_ERROR.ordinal()]);  
 return false;  
 }  
 return true;  
 }  
 static String inputPath()  
 {  
 String way;  
 boolean isIncorrect;  
 do  
 {  
 way = in.nextLine();  
 isIncorrect = !pathCondition(way);  
 } while (isIncorrect);  
  
 return way;  
 }  
 static boolean isCanOpenFile(String way)  
 {  
 File file = new File(way);  
 return file.canRead();  
 }  
 static String inputPathToTheFile()  
 {  
 String fileWay;  
 boolean isIncorrect = true;  
  
 do  
 {  
 fileWay = inputPath();  
 if (!isCanOpenFile(fileWay))  
 {  
 printError(ERRORS[IOError.MIN\_SIZE\_ERROR.ordinal()]);  
 }  
 else  
 {  
 isIncorrect = false;  
 }  
 } while (isIncorrect);  
  
 return fileWay;  
 }  
 /// input from console  
 static boolean checkSizeCondition(int size, boolean isIncorrectInput)  
 {  
 if (size < MIN\_SIZE && !isIncorrectInput)  
 {  
 printError(ERRORS[IOError.MIN\_SIZE\_ERROR.ordinal()]);  
 return true;  
 }  
  
 return isIncorrectInput;  
 }  
 static int inputSizeFromConsole()  
 {  
 int size = 0;  
 boolean isIncorrect;  
  
 System.out.print("How many characters do you want to enter: ");  
 do  
 {  
 boolean isIncorrectInput = true;  
 try {  
 size = Integer.parseInt(in.nextLine());  
 isIncorrectInput = false;  
 } catch (Exception error){  
 printError(ERRORS[IOError.INVALID\_PATH.ordinal()]);  
 }  
 isIncorrect = checkSizeCondition(size, isIncorrectInput);  
 } while (isIncorrect);  
  
 return size;  
 }  
 static boolean isCorrectElementsInputFromConsole(int size, String str)  
 {  
 if (size != str.length())  
 {  
 printError(ERRORS[IOError.EL\_ERROR.ordinal()]);  
 }  
 else  
 {  
 return false;  
 }  
  
 return true;  
 }  
 static char[] inputStringFromConsole(int[] arrSize)  
 {  
 boolean isIncorrect;  
 int size = inputSizeFromConsole();  
 char[] arrOfElements = new char[size];  
 arrSize[0] = size;  
  
 System.out.printf("Write your %d elements: ", size);  
 do  
 {  
 String str = in.nextLine();  
 for (int i = 0; i < size && size == str.length(); i++)  
 {  
 arrOfElements[i] = str.charAt(i);  
 }  
 isIncorrect = isCorrectElementsInputFromConsole(size, str);  
 } while (isIncorrect);  
  
 return arrOfElements;  
 }  
 // input from file  
 static int checkSizeInputFromFile(int size, int[] arrSize)  
 {  
 if (size < MIN\_SIZE && arrSize[1] != BAD\_SIZE)  
 {  
 printError(ERRORS[IOError.MIN\_SIZE\_ERROR.ordinal()]);  
 return BAD\_SIZE;  
 }  
  
 return GOOD\_SIZE;  
 }  
 static int inputSizeFromFile(Scanner fileScanner, int[] arrSize)  
 {  
 int size = 0;  
 try {  
 size = Integer.parseInt(fileScanner.nextLine());  
 } catch(Exception error) {  
 printError(ERRORS[IOError.FIRST\_STR\_ERROR.ordinal()]);  
 arrSize[1] = BAD\_SIZE;  
 }  
 arrSize[1] = checkSizeInputFromFile(size, arrSize);  
  
 return size;  
 }  
 static boolean isCorrectElInputFromFile(int size, String str)  
 {  
 if (size != str.length())  
 {  
 printError(ERRORS[IOError.EL\_ERROR.ordinal()]);  
 return true;  
 }  
  
 return false;  
 }  
 static boolean inputSetFromFile(char[] arrOfElements, int size,

int sizeSignal,Scanner fileScanner)  
 {  
 if (sizeSignal == BAD\_SIZE)  
 {  
 return true;  
 }  
  
 String str = fileScanner.nextLine();  
 for (int i = 0; i < size && size == str.length(); i++){  
 arrOfElements[i] = str.charAt(i);  
 }  
  
 return isCorrectElInputFromFile(size, str);  
 }  
 static char[] inputStringFromFile(int[] arrSize)  
 {  
 fileRestriction();  
 char[] arrOfElements = new char[0];  
 boolean isIncorrect = true;  
 do  
 {  
 String fileWay = inputPathToTheFile();  
 try (Scanner fileScanner = new Scanner(new File(fileWay))){  
 arrSize[0] = inputSizeFromFile(fileScanner, arrSize);  
 if (arrSize[1] == GOOD\_SIZE) {  
 arrOfElements = new char[arrSize[0]];  
 }  
 isIncorrect = inputSetFromFile(arrOfElements, arrSize[0],

arrSize[1], fileScanner);  
 } catch (Exception error) {  
 printError(ERRORS[IOError.OPEN\_FILE\_ERROR.ordinal()]);  
 }  
 } while (isIncorrect);  
  
 return arrOfElements;  
 }  
 // making the set  
 static void addEntitlements(Set<Character> Entitlements){  
 Collection<Character> AddList = Arrays.*asList*(

'0', '1', '2', '3', '4', '5', '6', '7',

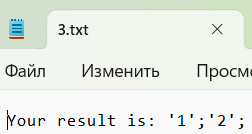
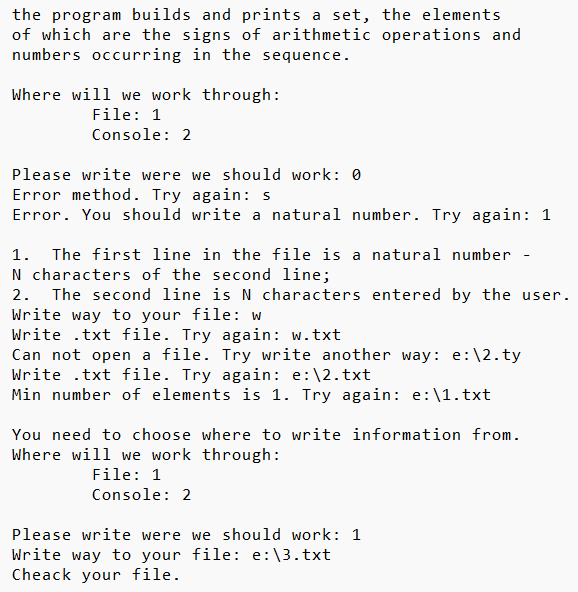
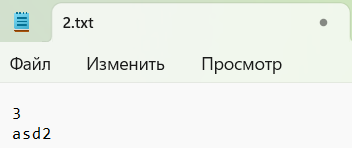
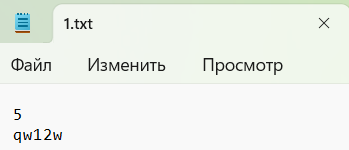
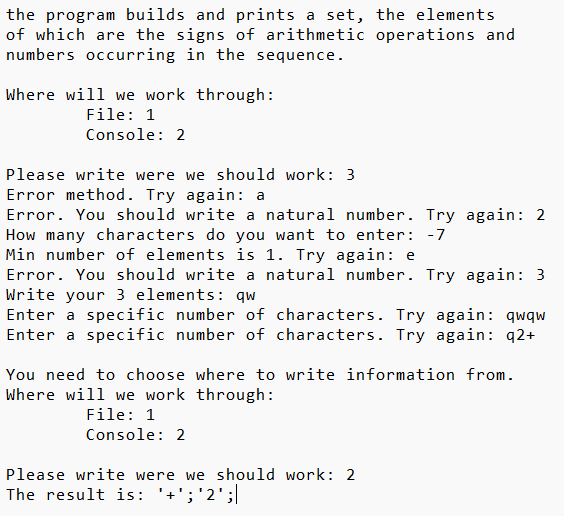
'8', '9', '+', '-', '/', '\*');  
 Entitlements. addAll(AddList);  
 }  
 static void renderingSet(char[] arrOfElements, int size,

Set<Character> resultSet,

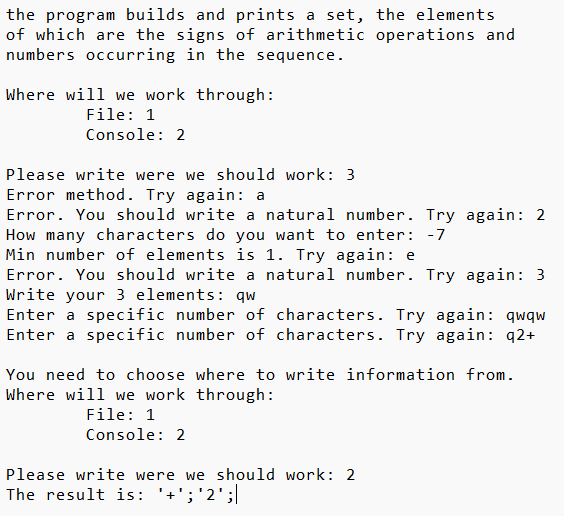
Set<Character> Entitlements)  
 {  
 for (int i = 0; i < size; i++)  
 {  
 for (char current : Entitlements)  
 {  
 if (arrOfElements[i] == current)  
 {  
 resultSet.add(arrOfElements[i]);  
 }  
 }  
 }  
 }  
 // output from file  
 static String outputSetInFile(Set<Character> resultSet)  
 {  
 StringBuilder res = new StringBuilder();  
 for (char currentElement : resultSet)  
 {  
 res.append(" '").append(currentElement).append("';");  
 }  
  
 return res.toString();  
 }  
 static String outputResInFile(Set<Character> resultSet)  
 {  
  
 String res = "The result is:";  
 if (resultSet.isEmpty())  
 {  
 res += " empty set.";  
 }  
 else  
 {  
 res += outputSetInFile(resultSet);  
 }  
  
 return res;  
 }  
 static void outputFromFile(Set<Character> resultSet)  
 {  
 boolean isIncorrect = true;  
 System.out.print("Write way to your file: ");  
 do  
 {  
 String fileWay = inputPathToTheFile();  
 File file = new File(fileWay);  
 StringBuilder builder;  
 if (file.canWrite())  
 {  
 try  
 {  
 FileWriter writer = new FileWriter(fileWay);  
 builder = new StringBuilder();  
 String result = outputResInFile(resultSet);  
 builder.append(result);  
 writer.write(builder.toString());  
 writer.close();  
 System.out.print("Check your file.");  
 isIncorrect = false;  
 } catch (Exception error)  
 {  
 printError(ERRORS[IOError.OPEN\_FILE\_ERROR.ordinal()]);  
 }  
 }  
 else  
 {  
 System.err.print("Can't open a file. Try write another way: ");  
 }  
 } while (isIncorrect);  
 //  
 }  
 // output from console  
 static void outputSetFromConsole(Set<Character> resultSet)  
 {  
 for (char currentElement : resultSet)  
 {  
 System.out.printf(" '%c';", currentElement);  
 }  
 }  
 static void outputFromConsole(Set<Character> resultSet)  
 {  
 System.out.print("The result is:");  
 if (resultSet.isEmpty())  
 {  
 System.out.print(" empty set.");  
 }  
 else  
 {  
 outputSetFromConsole(resultSet);  
 }  
 }  
 // distributive output  
 static void resultOutputSystem(Set<Character> resultSet)  
 {  
 System.out.print("\nYou need to choose where to write information

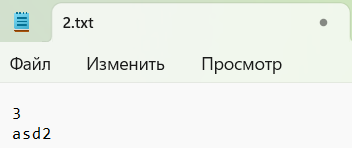
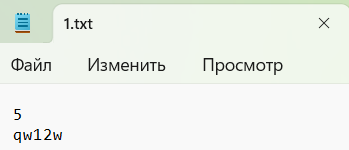
from.\n");  
 int path = choosingWorkWay();  
 if (path == CONSOLE\_KEY) {  
 outputFromConsole(resultSet);  
 } else {  
 outputFromFile(resultSet);  
 }  
 }  
// main distributive func  
 static char[] inputSystem(int[] arrSize)  
 {  
 int path = choosingWorkWay();  
 return path == CONSOLE\_KEY ? inputStringFromConsole(arrSize)  
 : inputStringFromFile(arrSize);  
 }  
 //main  
 public static void main(String[] args)  
 {  
 taskOutput();  
 int[] arrSize = new int[2];  
 char[] arrOfElements = inputSystem(arrSize);  
 Set<Character> resultSet = new HashSet<>();  
 Set<Character> Entitlements = new HashSet<>();  
 addEntitlements(Entitlements);  
 renderingSet(arrOfElements, arrSize[0], resultSet, Entitlements);  
 resultOutputSystem(resultSet);  
 in.close();  
 }  
}

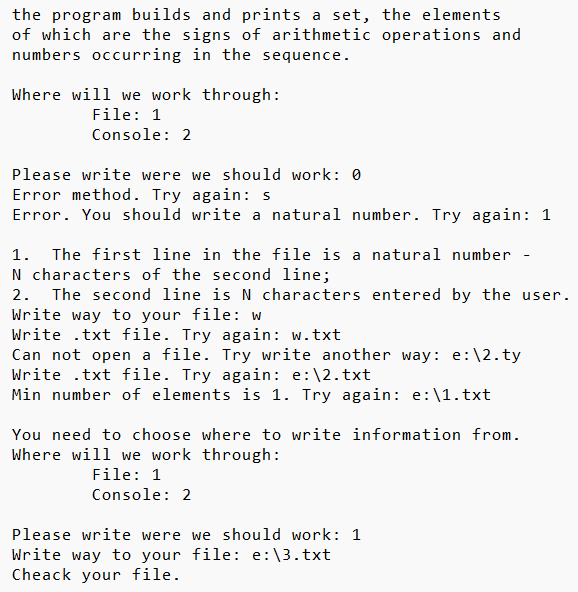
Результат на **Delphi**:

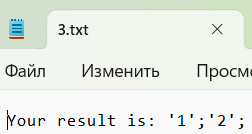


Результат на **Java**:

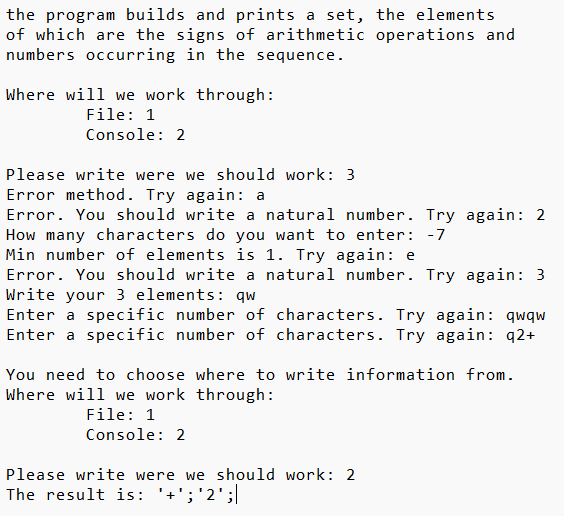


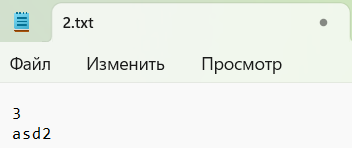
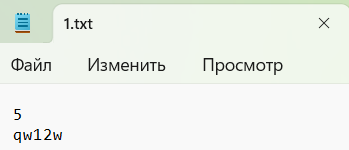


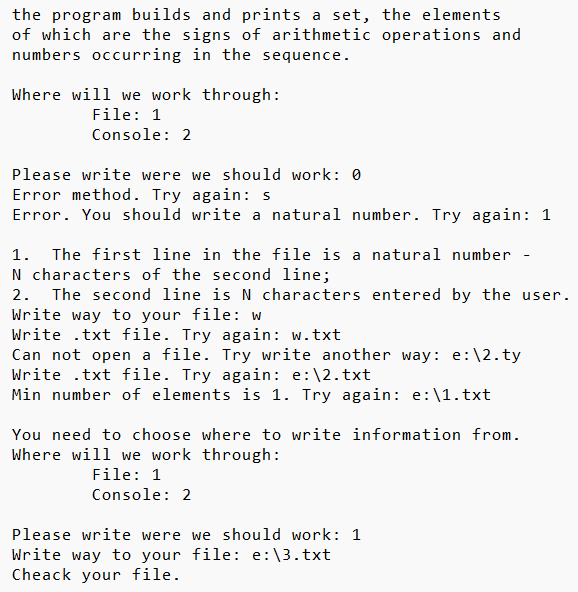


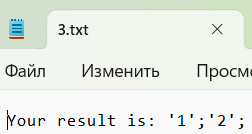


Результат на **C++**:









Блок-схема:

