УО «Белорусский государственный университет информатики и радиоэлектроники»

Кафедра ПОИТ

Отчет по лабораторной работе №3.1

по предмету «Основы алгоритмизации и программирования»

Вариант 12

Выполнил:

Галуха П. А.

Гр. 351005

Проверил:

Данилова Г. В.

Минск 2023

**Задание:**

Даны две строки st1 и st2. Разработать процедуру, которая по значению первого параметра выполняет следующие операции над данными строками: находит символы, встречающиеся в обеих строках, находит символы, встречающиеся только в первой строке, только во второй строке.

**Код программы Delphi:**

Program Lab31;

Uses

System.SysUtils;

Type

TChar = Array Of Char;

Const

MIN\_LEN = 1;

MAX\_LEN = 100;

FACTOR = 1.247;

Procedure PrintTask();

Begin

WriteLn('Данная программа находит элементы в двух строках по одному из критериев.');

WriteLn;

End;

Function CheckStringLen(Str: String) : Boolean;

Var

IsCorrect: Boolean;

Begin

IsCorrect := True;

If (Length(Str) < MIN\_LEN) Or (Length(Str) > MAX\_LEN) Then

Begin

WriteLn('Длина строки не попадает в диапазон!');

IsCorrect := False;

End;

CheckStringLen := IsCorrect;

End;

Function ChooseOption(Count: Integer) : Integer;

Var

SOption: String;

IOption, I: Integer;

IsCorrect, IsNotCorrectChoise: Boolean;

Begin

IOption := 1;

IsNotCorrectChoise := False;

Repeat

IsCorrect := True;

ReadLn(SOption);

Try

IOption := StrToInt(SOption);

Except

WriteLn('Некорректный выбор!');

Write('Повторите попытку: ');

IsCorrect := False;

End;

If IsCorrect Then

Begin

IsNotCorrectChoise := True;

I := 0;

While IsNotCorrectChoise And (I < Count) Do

Begin

If IOption = I + 1 Then

IsNotCorrectChoise := False;

Inc(I);

End;

End;

If IsNotCorrectChoise Then

Begin

WriteLn('Некорректный выбор!');

Write('Повторите попытку: ');

IsCorrect := False;

End;

Until IsCorrect;

ChooseOption := IOption;

End;

Function ReadPathToFile() : String;

Var

PathToFile: String;

IsCorrect: Boolean;

Begin

Repeat

IsCorrect := True;

Write('Введите путь к файлу с расширением .txt с двумя строками, с длинами[', MIN\_LEN, '; ',

MAX\_LEN, ']: ');

ReadLn(PathToFile);

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

Begin

WriteLn('Расширение файла не .txt!');

IsCorrect := False;

End;

Until IsCorrect;

ReadPathToFile := PathToFile;

End;

Function IsNotExists(PathToFile: String) : Boolean;

Var

IsRight: Boolean;

Begin

IsRight := True;

If FileExists(PathToFile) Then

IsRight := False;

IsNotExists := IsRight;

End;

Function IsNotAbleToReading(Var F: TextFile) : Boolean;

Var

IsRight: Boolean;

Begin

IsRight := False;

Try

Reset(F);

CloseFile(F);

Except

IsRight := True;

End;

IsNotAbleToReading := IsRight;

End;

Function IsNotAbleToWriting(PathToFile: String) : Boolean;

Var

IsRight: Boolean;

Begin

IsRight := False;

If FileIsReadOnly(PathToFile) Then

IsRight := True;

IsNotAbleToWriting := IsRight;

End;

Function IsEmpty(Var F: TextFile) : Boolean;

Var

IsRight: Boolean;

Begin

IsRight := False;

Reset(F);

If EOF(F) Then

IsRight := True;

CloseFile(F);

IsEmpty := IsRight;

End;

Function IsNotRightCountStrings(Var F: TextFile) : Boolean;

Var

IsRight: Boolean;

Begin

IsRight := False;

Reset(F);

ReadLn(F);

If EOF(F) Then

IsRight := True;

ReadLn(F);

If Not EOF(F) Then

IsRight := True;

CloseFile(F);

IsNotRightCountStrings := IsRight;

End;

Function IsNotCorrectStrings(Var F: TextFile) : Boolean;

Var

Str: String;

IsRight: Boolean;

Begin

Str := '';

Reset(F);

ReadLn(F, Str);

IsRight := CheckStringLen(Str);

If IsRight Then

Begin

ReadLn(F, Str);

IsRight := CheckStringLen(Str);

End;

CloseFile(F);

IsNotCorrectStrings := Not IsRight;

End;

Procedure GetFileNormalReading(Var F: TextFile);

Var

PathToFile: String;

IsCorrect: Boolean;

Begin

Repeat

IsCorrect := True;

PathToFile := ReadPathToFile();

If IsNotExists(PathToFile) Then

Begin

IsCorrect := False;

Writeln('Проверьте корректность ввода пути к файлу!');

End;

If IsCorrect Then

AssignFile(F, PathToFile);

If IsCorrect And IsNotAbleToReading(F) Then

Begin

IsCorrect := False;

Writeln('Файл закрыт для чтения!');

End;

If IsCorrect And IsEmpty(F) Then

Begin

IsCorrect := False;

WriteLn('Файл пуст!');

End;

If IsCorrect And IsNotRightCountStrings(F) Then

Begin

IsCorrect := False;

Writeln('Количество строк в файле не две!');

End;

If IsCorrect And IsNotCorrectStrings(F) Then

IsCorrect := False;

Until IsCorrect;

End;

Procedure GetFileNormalWriting(Var F: TextFile);

Var

PathToFile: String;

IsCorrect: Boolean;

Begin

Repeat

IsCorrect := True;

PathToFile := ReadPathToFile();

If IsNotExists(PathToFile) Then

Begin

IsCorrect := False;

Writeln('Проверьте корректность ввода пути к файлу!');

End;

If IsCorrect Then

AssignFile(F, PathToFile);

If IsCorrect And IsNotAbleToWriting(PathToFile) Then

Begin

IsCorrect := False;

WriteLn('Файл закрыт для записи!');

End;

Until IsCorrect;

End;

Function ReadFileString(Var F: TextFile) : String;

Var

Str: String;

Begin

ReadLn(F, Str);

ReadFileString := Str;

End;

Function ReadConsoleString(Num: Integer) : String;

Var

Str: String;

IsCorrect: Boolean;

Begin

Repeat

Write('Введите строку номер ', Num, ', с длиной[', MIN\_LEN, ';', MAX\_LEN, ']: ');

Readln(Str);

IsCorrect := CheckStringLen(Str);

Until IsCorrect;

ReadConsoleString := Str;

End;

Procedure ReadStrings(Var Str1: String; Var Str2: String);

Var

RF: TextFile;

Option: Integer;

Begin

WriteLn('Вы хотите: ');

WriteLn('Вводить строки через файл - 1');

WriteLn('Вводить строки через консоль - 2');

Option := ChooseOption(2);

If Option = 1 Then

Begin

GetFileNormalReading(RF);

Reset(RF);

Str1 := ReadFileString(RF);

Str2 := ReadFileString(RF);

CloseFile(RF);

End

Else

Begin

Str1 := ReadConsoleString(1);

Str2 := ReadConsoleString(2);

End;

End;

Procedure FillOneAStr(Str: String; Var AStr: TChar);

Var

I: Integer;

Begin

SetLength(AStr, Length(Str));

For I := Low(AStr) To High(AStr) Do

AStr[I] := Str[I + 1];

End;

Procedure FillAStrs(Str1: String; Str2: String; Var AStr1: TChar; Var AStr2: TChar);

Begin

FillOneAStr(Str1, Astr1);

FillOneAStr(Str2, Astr2);

End;

Procedure SortOneAStr(Var AStr: TChar);

Var

Step: Real;

I, IStep: Integer;

Buf: Char;

Begin

Step := Length(AStr) - 1;

While Step >= 1 Do

Begin

IStep := Trunc(Step);

I := 0;

While Step + I < Length(AStr) Do

Begin

If Ord(AStr[I]) > Ord(AStr[I + IStep]) Then

Begin

Buf := AStr[I];

AStr[I] := AStr[I + IStep];

AStr[I + IStep] := Buf;

End;

Inc(I);

End;

Step := Step / FACTOR;

End;

End;

Procedure SortAStrs(Var AStr1: TChar; Var AStr2: TChar);

Begin

SortOneAStr(AStr1);

SortOneAStr(AStr2);

End;

Function PlusAStr(Var CombAStr: TChar; AStr: TChar; J: Integer) : Integer;

Var

I: Integer;

Begin

I := 0;

While I < High(AStr) Do

Begin

If AStr[I] <> AStr[I + 1] Then

Begin

CombAStr[J] := AStr[I];

Inc(J);

End;

Inc(I);

End;

CombAStr[J] := AStr[High(AStr)];

PlusAStr := J + 1;

End;

Procedure MakeCombSameAStr(Var CombAStr: TChar; AStr1: TChar; AStr2: TChar);

Var

J: Integer;

Begin

SetLength(CombAStr, Length(AStr1) + Length(AStr2));

J := 0;

J := PlusAStr(CombAStr, AStr1, J);

J := PlusAStr(CombAStr, AStr2, J);

SetLength(CombAStr, J);

End;

Procedure MakeCombUniqueAStr(Var CombAStr: TChar; AStr1: TChar; AStr2: TChar; AStr3: TChar);

Var

J: Integer;

Begin

SetLength(CombAStr, Length(AStr1) + Length(AStr2) + Length(AStr3));

J := 0;

J := PlusAStr(CombAStr, AStr1, J);

J := PlusAStr(CombAStr, AStr2, J);

J := PlusAStr(CombAStr, AStr3, J);

SetLength(CombAStr, J);

End;

Procedure FindSame(CombAStr: TChar; Var ResAStr: TChar);

Var

I, J: Integer;

Begin

I := 0;

J := 0;

SetLength(ResAStr, Length(CombAStr));

While I < High(CombAStr) Do

Begin

If CombAStr[I] = CombAStr[I + 1] Then

Begin

ResAStr[J] := CombAStr[I];

Inc(J);

Inc(I);

End;

Inc(I);

End;

If J = 0 Then

SetLength(ResAStr, 1)

Else

SetLength(ResAStr, J);

End;

Procedure FindUnique(CombAStr: TChar; Var ResAStr: TChar);

Var

I, J: Integer;

Begin

I := 0;

J := 0;

SetLength(ResAStr, Length(CombAStr));

While I < High(CombAStr) Do

Begin

If CombAStr[I] = CombAStr[I + 1] Then

If (I = High(CombAStr) - 1) Or (CombAStr[I] <> CombAStr[I + 2]) Then

Begin

ResAStr[J] := CombAStr[I];

Inc(J);

Inc(I);

End

Else

Inc(I, 2);

Inc(I);

End;

If J = 0 Then

SetLength(ResAStr, 1)

Else

SetLength(ResAStr, J);

End;

Function ChooseAction() : Integer;

Begin

WriteLn('Вы хотите: ');

WriteLn('Найти одинаковые символы в обеих строках - 1');

WriteLn('Найти уникальные символы в первой строке - 2');

WriteLn('Найти уникальные символы во второй строке - 3');

ChooseAction := ChooseOption(3);

End;

Procedure PrintConsoleResult(ResAStr: TChar);

Var

I: Integer;

Begin

WriteLn;

Write('Элементы, удовлетворяющие условию: ');

If ResAStr[0] = #0 Then

Write('элементов, удовлетворяющих условию, нет!')

Else

For I := 0 To High(ResAStr) Do

Write('''', ResAStr[I], '''; ');

End;

Procedure PrintFileResult(Var F: TextFile; ResAStr: TChar);

Var

I: Integer;

Begin

Append(F);

WriteLn(F);

Write(F, 'Элементы, удовлетворяющие условию: ');

If ResAStr[0] = #0 Then

Write(F, 'элементов, удовлетворяющих условию, нет')

Else

For I := 0 To High(ResAStr) Do

Write(F, '''', ResAStr[I], '''; ');

CloseFile(F);

End;

Procedure PrintResult(ResAStr: TChar);

Var

WF: TextFile;

Option: Integer;

Begin

WriteLn('Вы хотите: ');

WriteLn('Выводить строки через файл - 1');

WriteLn('Выводить строки через консоль - 2');

Option := ChooseOption(2);

If Option = 1 Then

Begin

GetFileNormalWriting(WF);

PrintFileResult(WF, ResAStr);

End

Else

PrintConsoleResult(ResAStr);

End;

Procedure FreeMemory(Var AStr1: TChar; Var AStr2: TChar; Var CombAStr: TChar; Var ResAStr: TChar);

Begin

AStr1 := Nil;

AStr2 := Nil;

CombAStr := Nil;

ResAStr := Nil;

End;

Var

AStr1, AStr2, CombAStr, ResAStr: TChar;

Str1, Str2: String;

Action: Integer;

Begin

PrintTask();

ReadStrings(Str1, Str2);

FillAStrs(Str1, Str2, AStr1, AStr2);

SortAStrs(AStr1, AStr2);

Action := ChooseAction();

If Action = 1 Then

Begin

MakeCombSameAStr(CombAStr, AStr1, AStr2);

SortOneAStr(CombAStr);

FindSame(CombAStr, ResAStr)

End

Else If Action = 2 Then

Begin

MakeCombUniqueAStr(CombAStr, AStr1, AStr2, AStr1);

SortOneAStr(CombAStr);

FindUnique(CombAStr, ResAStr)

End

Else

Begin

MakeCombUniqueAStr(CombAStr, AStr1, AStr2, AStr2);

SortOneAStr(CombAStr);

FindUnique(CombAStr, ResAStr)

End;

PrintResult(ResAStr);

FreeMemory(AStr1, AStr2, CombAStr, ResAStr);

ReadLn;

End.

**Код программы С++:**

#include <iostream>

#include <string>

#include <fstream>

const int

MIN\_LEN = 1,

MAX\_LEN = 100;

const double

FACTOR = 1.247;

void printTask()

{

std::cout << "Данная программа находит элементы в двух строках по одному из критериев.\n\n";

}

bool checkStringLen(std::string str)

{

bool isCorrect;

isCorrect = true;

if (str.size() < MIN\_LEN || str.size() > MAX\_LEN)

{

std::cout << "Длина строки не попадает в диапазон!\n";

isCorrect = false;

}

return isCorrect;

}

int chooseOption(int count)

{

int option, i;

bool isCorrect, isNotCorrectChoise;

option = 0;

isNotCorrectChoise = false;

do {

isCorrect = true;

std::cin >> option;

if (std::cin.fail())

{

std::cin.clear();

std::cout << "Проверьте корректность ввода данных!\n";

std::cout << "Повторите попытку: \n";

isCorrect = false;

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

}

if (isCorrect && std::cin.get() != '\n')

{

std::cout << "Проверьте корректность ввода данных!\n";

std::cout << "Повторите попытку: \n";

isCorrect = false;

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

}

if (isCorrect)

{

isNotCorrectChoise = true;

i = 0;

while (isNotCorrectChoise && i < count)

{

if (option == i + 1)

isNotCorrectChoise = false;

i++;

}

}

if (isNotCorrectChoise)

{

std::cout << "Проверьте корректность ввода данных!\n";

std::cout << "Повторите попытку: \n";

isCorrect = false;

}

} while (!isCorrect);

return option;

}

std::string readPathFile()

{

std::string pathToFile;

bool isCorrect;

do

{

isCorrect = true;

std::cout << "Введите путь к файлу с расширением.txt с двумя строками, с длинами[" << MIN\_LEN <<

"; " << MAX\_LEN << "]: ";

std::cin >> pathToFile;

if (pathToFile.size() < 5 || pathToFile[pathToFile.length() - 4] != '.' ||

pathToFile[pathToFile.length() - 3] != 't' || pathToFile[pathToFile.length() - 2] != 'x' ||

pathToFile[pathToFile.length() - 1] != 't')

{

std::cout << "Расширение файла не .txt!\n";

isCorrect = false;

}

} while (!isCorrect);

return pathToFile;

}

bool isNotExists(std::string pathToFile)

{

bool isRight;

isRight = true;

std::ifstream file(pathToFile);

if (file.good())

isRight = false;

file.close();

return isRight;

}

bool isNotAbleToReading(std::string pathToFile)

{

bool isRight;

isRight = true;

std::ifstream file(pathToFile);

if (file.is\_open())

isRight = false;

file.close();

return isRight;

}

bool isNotAbleToWriting(std::string pathToFile)

{

bool isRight;

isRight = true;

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

if (file.is\_open())

isRight = false;

file.close();

return isRight;

}

bool isEmpty(std::string pathToFile)

{

bool isRight;

isRight = false;

std::ifstream file(pathToFile);

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

isRight = true;

file.close();

return isRight;

}

bool isNotRightCountStrings(std::string pathToFile)

{

bool isRight;

isRight = false;

std::ifstream file(pathToFile);

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

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

isRight = true;

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

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

isRight = true;

file.close();

return isRight;

}

bool isNotCorrectStrings(std::string pathToFile)

{

std::string str;

bool isRight;

std::ifstream file(pathToFile);

std::getline(file, str);

isRight = checkStringLen(str);

if (isRight) {

std::getline(file, str);

isRight = checkStringLen(str);

}

file.close();

return !isRight;

}

void getFileNormalReading(std::string& pathToFile)

{

bool isCorrect;

do

{

isCorrect = true;

pathToFile = readPathFile();

if (isNotExists(pathToFile))

{

isCorrect = false;

std::cout << "Проверьте корректность ввода пути к файлу!\n";

}

if (isCorrect && isNotAbleToReading(pathToFile))

{

isCorrect = false;

std::cout << "Файл закрыт для чтения!\n";

}

if (isCorrect && isEmpty(pathToFile))

{

isCorrect = false;

std::cout << "Файл пуст!\n";

}

if (isCorrect && isNotRightCountStrings(pathToFile))

{

isCorrect = false;

std::cout << "Количество строк в файле не две!\n";

}

if (isCorrect && isNotCorrectStrings(pathToFile))

isCorrect = false;

} while (!isCorrect);

}

void getFileNormalWriting(std::string& pathToFile)

{

bool isCorrect;

do

{

isCorrect = true;

pathToFile = readPathFile();

if (isNotExists(pathToFile))

{

isCorrect = false;

std::cout << "Проверьте корректность ввода пути к файлу!\n";

}

if (isCorrect && isNotAbleToWriting(pathToFile))

{

isCorrect = false;

std::cout << "Файл закрыт для записи!\n";

}

} while (!isCorrect);

}

std::string readFileString(std::ifstream& file)

{

std::string str;

std::getline(file, str);

return str;

}

std::string readConsoleString(int num)

{

std::string str;

bool isCorrect;

do

{

std::cout << "Введите строку номер " << num << ", с длиной[" << MIN\_LEN << ";" << MAX\_LEN << "]: ";

std::cin >> str;

isCorrect = checkStringLen(str);

} while (!isCorrect);

return str;

}

void readStrings(std::string& str1, std::string& str2)

{

std::string pathToFile;

int option;

std::cout << "Вы хотите: \n";

std::cout << "Вводить матрицу через файл - 1\n";

std::cout << "Вводить матрицу через консоль - 2\n";

option = chooseOption(2);

if (option == 1)

{

getFileNormalReading(pathToFile);

std::ifstream file(pathToFile);

str1 = readFileString(file);

str2 = readFileString(file);

file.close();

}

else

{

str1 = readConsoleString(1);

str2 = readConsoleString(2);

}

}

void fillOneAStr(std::string str, char\*& aStr, int& lenAStr)

{

int i;

lenAStr = str.size();

aStr = new char[lenAStr];

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

aStr[i] = str[i];

}

void fillAStrs(std::string str1, std::string str2, char\*& aStr1, char\*& aStr2, int& lenAStr1, int& lenAStr2)

{

fillOneAStr(str1, aStr1, lenAStr1);

fillOneAStr(str2, aStr2, lenAStr2);

}

void sortOneAStr(char\*& aStr, int lenAStr)

{

double step;

int i, iStep;

char buf;

for (step = lenAStr - 1; step >= 1; step /= FACTOR)

{

iStep = trunc(step);

for (i = 0; step + i < lenAStr; i++)

if (static\_cast<int>(aStr[i]) > static\_cast<int>(aStr[i + iStep]))

{

buf = aStr[i];

aStr[i] = aStr[i + iStep];

aStr[i + iStep] = buf;

}

}

}

void sortAStrs(char\*& aStr1, char\*& aStr2, int lenAStr1, int lenAStr2)

{

sortOneAStr(aStr1, lenAStr1);

sortOneAStr(aStr2, lenAStr2);

}

int plusAStr(char\*& combAStr, char\* aStr, int j, int lenAStr1)

{

int i, maxIndex;

maxIndex = lenAStr1 - 1;

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

if (aStr[i] != aStr[i + 1])

{

combAStr[j] = aStr[i];

j++;

}

combAStr[j] = aStr[maxIndex];

return ++j;

}

void makeCombSameAStr(char\*& combAStr, char\* aStr1, char\* aStr2, int& lenCAStr, int lenAStr1, int lenAStr2)

{

char\* bufArr;

int i, j;

bufArr = new char[lenAStr1 + lenAStr2];

j = 0;

j = plusAStr(bufArr, aStr1, j, lenAStr1);

j = plusAStr(bufArr, aStr2, j, lenAStr2);

lenCAStr = j;

combAStr = new char[lenCAStr];

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

combAStr[i] = bufArr[i];

delete[] bufArr;

}

void makeCombUniqueAStr(char\*& combAStr, char\* aStr1, char\* aStr2, char\* aStr3, int& lenCAStr,

int lenAStr1, int lenAStr2, int lenAStr3)

{

char\* bufArr;

int i, j;

bufArr = new char[lenAStr1 + lenAStr2 + lenAStr3];

j = 0;

j = plusAStr(bufArr, aStr1, j, lenAStr1);

j = plusAStr(bufArr, aStr2, j, lenAStr2);

j = plusAStr(bufArr, aStr3, j, lenAStr3);

lenCAStr = j;

combAStr = new char[lenCAStr];

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

combAStr[i] = bufArr[i];

delete[] bufArr;

}

void findSame(char\* combAStr, char\*& resAStr, int lenCAStr, int& lenRAStr)

{

char\* bufArr;

int i, j, maxIndex;

bufArr = new char[lenCAStr];

bufArr[0] = '\0';

i = 0;

j = 0;

maxIndex = lenCAStr - 1;

while (i < maxIndex)

{

if (combAStr[i] == combAStr[i + 1])

{

bufArr[j] = combAStr[i];

j++;

i++;

}

i++;

}

if (j == 0)

lenRAStr = 1;

else

lenRAStr = j;

resAStr = new char[lenRAStr];

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

resAStr[i] = bufArr[i];

delete[] bufArr;

}

void findUnique(char\* combAStr, char\*& resAStr, int lenCAStr, int& lenRAStr)

{

char\* bufArr;

int i, j, maxIndex;

bufArr = new char[lenCAStr];

bufArr[0] = '\0';

i = 0;

j = 0;

maxIndex = lenCAStr - 1;

while (i < maxIndex)

{

if (combAStr[i] == combAStr[i + 1])

if (i == maxIndex - 1 || combAStr[i] != combAStr[i + 2])

{

bufArr[j] = combAStr[i];

j++;

i++;

}

else

i += 2;

i++;

}

if (j == 0)

lenRAStr = 1;

else

lenRAStr = j;

resAStr = new char[lenRAStr];

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

resAStr[i] = bufArr[i];

delete[] bufArr;

}

int chooseAction()

{

std::cout << "Вы хотите: \n";

std::cout << "Найти одинаковые символы в обеих строках - 1\n";

std::cout << "Найти уникальные символы в первой строке - 2\n";

std::cout << "Найти уникальные символы во второй строке - 3\n";

return chooseOption(3);

}

void printConsoleResult(char\* resAStr, int lenRAStr)

{

int i;

std::cout << "\nЭлементы, удовлетворяющие условию: ";

if (resAStr[0] == '\0')

std::cout << "элементов, удовлетворяющих условию, нет!";

else

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

std::cout << "'" << resAStr[i] << "'; ";

}

void printFileResult(std::string pathToFile, char\* resAStr, int lenRAStr)

{

int i;

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

file << "\nЭлементы, удовлетворяющие условию: ";

if (resAStr[0] == '\0')

file << "элементов, удовлетворяющих условию, нет!";

else

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

file << "'" << resAStr[i] << "'; ";

file.close();

}

void printResult(char\* resAStr, int lenRAStr)

{

std::string pathToFile;

int option;

std::cout << "Вы хотите: \n";

std::cout << "Выводить строки через файл - 1\n";

std::cout << "Выводить строки через консоль - 2\n";

option = chooseOption(2);

if (option == 1)

{

getFileNormalWriting(pathToFile);

printFileResult(pathToFile, resAStr, lenRAStr);

}

else

printConsoleResult(resAStr, lenRAStr);

}

void freeMemory(char\*& aStr1, char\*& aStr2, char\*& combAStr, char\*& resAStr)

{

delete[] aStr1;

delete[] aStr2;

delete[] combAStr;

delete[] resAStr;

}

int main()

{

setlocale(LC\_ALL, "RU");

std::string str1, str2;

char\* aStr1;

char\* aStr2;

char\* combAStr;

char\* resAStr;

int lenAStr1, lenAStr2, lenCAStr, lenRAStr, action;

printTask();

readStrings(str1, str2);

fillAStrs(str1, str2, aStr1, aStr2, lenAStr1, lenAStr2);

sortAStrs(aStr1, aStr2, lenAStr1, lenAStr2);

action = chooseAction();

if (action == 1)

{

makeCombSameAStr(combAStr, aStr1, aStr2, lenCAStr, lenAStr1, lenAStr2);

sortOneAStr(combAStr, lenCAStr);

findSame(combAStr, resAStr, lenCAStr, lenRAStr);

}

else if (action == 2)

{

makeCombUniqueAStr(combAStr, aStr1, aStr2, aStr1, lenCAStr, lenAStr1, lenAStr2, lenAStr1);

sortOneAStr(combAStr, lenCAStr);

findUnique(combAStr, resAStr, lenCAStr, lenRAStr);

}

else

{

makeCombUniqueAStr(combAStr, aStr1, aStr2, aStr2, lenCAStr, lenAStr1, lenAStr2, lenAStr2);

sortOneAStr(combAStr, lenCAStr);

findUnique(combAStr, resAStr, lenCAStr, lenRAStr);

}

printResult(resAStr, lenRAStr);

freeMemory(aStr1, aStr2, combAStr, resAStr);

return 0;

}

**Код программы Java:**

import java.util.Scanner;

import java.io.File;

import java.io.FileWriter;

public class Main {

public static final int

MIN\_LEN = 1,

MAX\_LEN = 100;

public static final double

FACTOR = 1.247;

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

static File file;

public static void printTask() {

System.out.print("Данная программа находит элементы в двух строках по одному из критериев.\n\n");

}

public static boolean checkStringLen(String str) {

boolean isCorrect;

isCorrect = true;

if (str.length() < MIN\_LEN || str.length() > MAX\_LEN) {

System.out.print("Длина строки не попадает в диапазон!\n");

isCorrect = false;

}

return isCorrect;

}

public static int chooseOption(int count) {

String sOption;

int iOption, i;

boolean isCorrect, isNotCorrectChoise;

iOption = 0;

isNotCorrectChoise = false;

do {

isCorrect = true;

sOption = scanConsole.nextLine();

try {

iOption = Integer.parseInt(sOption);

} catch (NumberFormatException e) {

System.out.print("Некорректный выбор!\n");

System.out.print("Повторите попытку: \n");

isCorrect = false;

}

if (isCorrect) {

isNotCorrectChoise = true;

i = 0;

while (isNotCorrectChoise && i < count) {

if (iOption == i + 1)

isNotCorrectChoise = false;

i++;

}

}

if (isNotCorrectChoise) {

System.out.print("Некорректный выбор!\n");

System.out.print("Повторите попытку: \n");

isCorrect = false;

}

} while (!isCorrect);

return iOption;

}

public static void readPathFile() {

String pathToFile;

boolean isCorrect;

do {

isCorrect = true;

System.out.print("Введите путь к файлу с расширением.txt с двумя строками, с длинами[" +

MIN\_LEN + "; " + MAX\_LEN + "]: \n");

pathToFile = scanConsole.nextLine();

if (pathToFile.length() < 5 || pathToFile.charAt(pathToFile.length() - 4) != '.' ||

pathToFile.charAt(pathToFile.length() - 3) != 't' ||

pathToFile.charAt(pathToFile.length() - 2) != 'x' ||

pathToFile.charAt(pathToFile.length() - 1) != 't') {

isCorrect = false;

System.out.print("Расширение файла не .txt!\n");

}

} while (!isCorrect);

file = new File(pathToFile);

}

public static boolean isNotExists() {

return !file.exists();

}

public static boolean isNotAbleToReading() {

return !file.canRead();

}

public static boolean isNotAbleToWriting() {

return !file.canWrite();

}

public static boolean isEmpty() {

return file.length() == 0;

}

public static boolean isNotRightCountStrings() {

boolean isRight;

isRight = false;

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

scanFile.nextLine();

if (!scanFile.hasNext())

isRight = true;

scanFile.nextLine();

if (scanFile.hasNext())

isRight = true;

} catch (Exception e) {}

return isRight;

}

public static boolean isNotCorrectStrings() {

String str;

boolean isRight;

isRight = false;

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

str = scanFile.nextLine();

isRight = checkStringLen(str);

if (isRight) {

str = scanFile.nextLine();

isRight = checkStringLen(str);

}

} catch (Exception e) {}

return !isRight;

}

public static void getFileNormalReading() {

boolean isCorrect;

do {

isCorrect = true;

readPathFile();

if (isNotExists()) {

isCorrect = false;

System.out.print("Проверьте корректность ввода пути к файлу!\n");

}

if (isCorrect && isNotAbleToReading()) {

isCorrect = false;

System.out.print("Файл закрыт для чтения!\n");

}

if (isCorrect && isEmpty()) {

isCorrect = false;

System.out.print("Файл пуст!\n");

}

if (isCorrect && isNotRightCountStrings()) {

isCorrect = false;

System.out.print("Количество строк в файле не две!\n");

}

if (isCorrect && isNotCorrectStrings())

isCorrect = false;

} while (!isCorrect);

}

public static void getFileNormalWriting() {

boolean isCorrect;

do {

isCorrect = true;

readPathFile();

if (isNotExists()) {

isCorrect = false;

System.out.print("Проверьте корректность ввода пути к файлу!\n");

}

if (isCorrect && isNotAbleToWriting()) {

isCorrect = false;

System.out.print("Файл закрыт для записи!\n");

}

} while (!isCorrect);

}

public static String readFileString(Scanner scanFile) {

String str;

str = scanFile.nextLine();

return str;

}

public static String readConsoleString(int num) {

String str;

boolean isCorrect;

do {

System.out.print("Введите строку номер " + num + ", с длиной[" + MIN\_LEN + ";" +

MAX\_LEN + "]: \n");

str = scanConsole.nextLine();

isCorrect = checkStringLen(str);

} while (!isCorrect);

return str;

}

public static String[] readStrings() {

String[] twoStrings = new String[2];

int option;

System.out.print("Вы хотите: \n");

System.out.print("Вводить матрицу через файл - 1\n");

System.out.print("Вводить матрицу через консоль - 2\n");

option = chooseOption(2);

if (option == 1) {

getFileNormalReading();

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

twoStrings[0] = readFileString(scanFile);

twoStrings[1] = readFileString(scanFile);

} catch (Exception e) {}

}

else {

twoStrings[0] = readConsoleString(1);

twoStrings[1] = readConsoleString(2);

}

return twoStrings;

}

public static char[] fillOneAStr(String str) {

char[] aStr = new char[str.length()];

int i;

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

aStr[i] = str.charAt(i);

return aStr;

}

public static void sortOneAStr(char[] aStr) {

double step;

int i, iStep;

char buf;

for (step = aStr.length - 1; step >= 1; step /= FACTOR) {

iStep = (int) step;

for (i = 0; step + i < aStr.length; i++)

if ((int) aStr[i] > (int) aStr[i + iStep]) {

buf = aStr[i];

aStr[i] = aStr[i + iStep];

aStr[i + iStep] = buf;

}

}

}

public static void sortAStrs(char[] aStr1, char[] aStr2) {

sortOneAStr(aStr1);

sortOneAStr(aStr2);

}

public static int plusAStr(char[] combAStr, char[] aStr, int j) {

int i, maxIndex;

maxIndex = aStr.length - 1;

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

if (aStr[i] != aStr[i + 1]) {

combAStr[j] = aStr[i];

j++;

}

combAStr[j] = aStr[maxIndex];

return ++j;

}

public static char[] makeCombSameAStr(char[] aStr1, char[] aStr2) {

char[] combAStr;

char[] bufArr;

int i, j;

bufArr = new char[aStr1.length + aStr2.length];

j = 0;

j = plusAStr(bufArr, aStr1, j);

j = plusAStr(bufArr, aStr2, j);

combAStr = new char[j];

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

combAStr[i] = bufArr[i];

return combAStr;

}

public static char[] makeCombUniqueAStr(char[] aStr1, char[] aStr2, char[] aStr3) {

char[] combAStr;

char[] bufArr;

int i, j;

bufArr = new char[aStr1.length + aStr2.length + aStr3.length];

j = 0;

j = plusAStr(bufArr, aStr1, j);

j = plusAStr(bufArr, aStr2, j);

j = plusAStr(bufArr, aStr3, j);

combAStr = new char[j];

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

combAStr[i] = bufArr[i];

return combAStr;

}

public static char[] findSame(char[] combAStr) {

char[] resAStr;

char[] bufArr;

int i, j, maxIndex;

bufArr = new char[combAStr.length];

i = 0;

j = 0;

maxIndex = combAStr.length - 1;

while (i < maxIndex) {

if (combAStr[i] == combAStr[i + 1]) {

bufArr[j] = combAStr[i];

j++;

i++;

}

i++;

}

if (j == 0)

resAStr = new char[1];

else

resAStr = new char[j];

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

resAStr[i] = bufArr[i];

return resAStr;

}

public static char[] findUnique(char[] combAStr) {

char[] resAStr;

char[] bufArr;

int i, j, maxIndex;

bufArr = new char[combAStr.length];

i = 0;

j = 0;

maxIndex = combAStr.length - 1;

while (i < maxIndex) {

if (combAStr[i] == combAStr[i + 1])

if (i == maxIndex - 1 || combAStr[i] != combAStr[i + 2]) {

bufArr[j] = combAStr[i];

j++;

i++;

}

else

i += 2;

i++;

}

if (j == 0)

resAStr = new char[1];

else

resAStr = new char[j];

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

resAStr[i] = bufArr[i];

return resAStr;

}

public static int chooseAction() {

System.out.print("Вы хотите: \n");

System.out.print("Найти одинаковые символы в обеих строках - 1\n");

System.out.print("Найти уникальные символы в первой строке - 2\n");

System.out.print("Найти уникальные символы во второй строке - 3\n");

return chooseOption(3);

}

public static void printConsoleResult(char[] resAStr) {

int i;

System.out.print("\nЭлементы, удовлетворяющие условию: ");

if (resAStr[0] == '\0')

System.out.print("элементов, удовлетворяющих условию, нет!");

else

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

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

}

public static void printFileResult(char[] resAStr) {

int i;

try(FileWriter writer = new FileWriter(file, true)) {

writer.write("\nЭлементы, удовлетворяющие условию: ");

if (resAStr[0] == '\0')

writer.write("элементов, удовлетворяющих условию, нет!");

else

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

writer.write("'" + resAStr[i] + "'; ");

} catch (Exception e) {}

}

public static void printResult(char[] resAStr) {

int option;

System.out.print("Вы хотите: \n");

System.out.print("Выводить строки через файл - 1\n");

System.out.print("Выводить строки через консоль - 2\n");

option = chooseOption(2);

if (option == 1) {

getFileNormalWriting();

printFileResult(resAStr);

}

else

printConsoleResult(resAStr);

}

public static void main(String[] args) {

String[] twoStrings;

char[] aStr1;

char[] aStr2;

char[] combAStr;

char[] resAStr;

int action;

printTask();

twoStrings = readStrings();

aStr1 = fillOneAStr(twoStrings[0]);

aStr2 = fillOneAStr(twoStrings[1]);

sortAStrs(aStr1, aStr2);

action = chooseAction();

if (action == 1) {

combAStr = makeCombSameAStr(aStr1, aStr2);

sortOneAStr(combAStr);

resAStr = findSame(combAStr);

}

else if (action == 2) {

combAStr = makeCombUniqueAStr(aStr1, aStr2, aStr1);

sortOneAStr(combAStr);

resAStr = findUnique(combAStr);

}

else {

combAStr = makeCombUniqueAStr(aStr1, aStr2, aStr2);

sortOneAStr(combAStr);

resAStr = findUnique(combAStr);

}

printResult(resAStr);

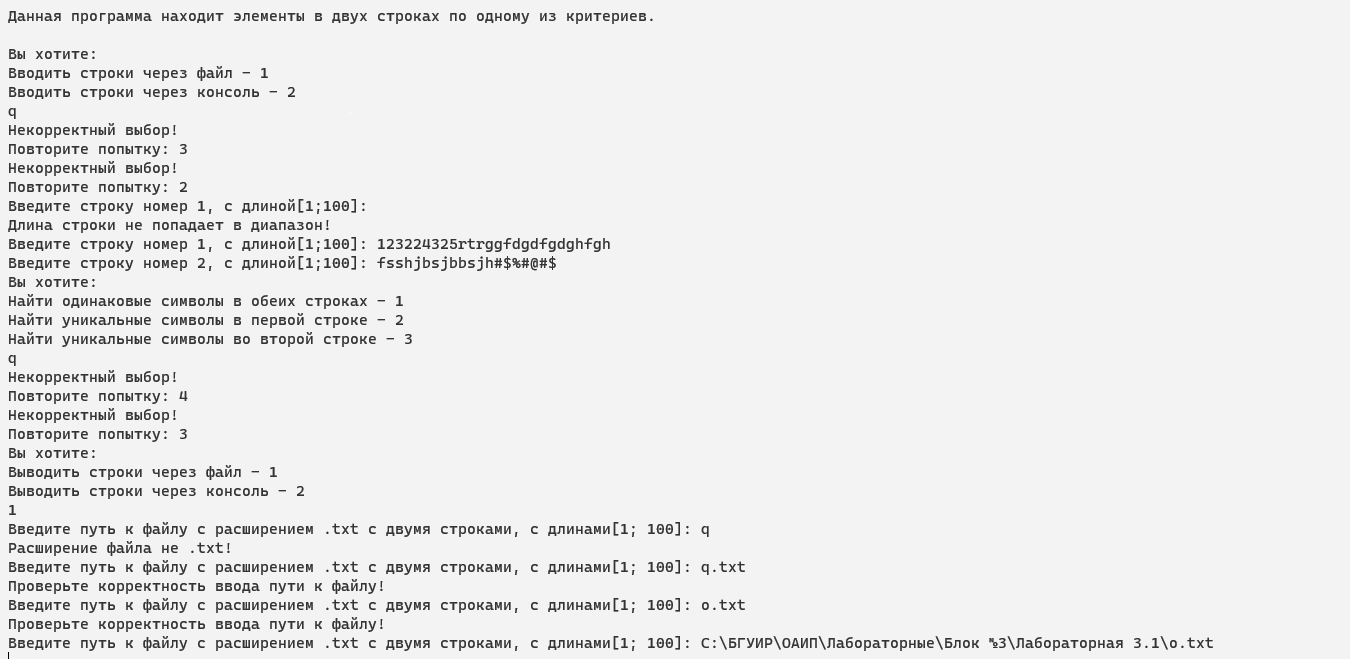
scanConsole.close();

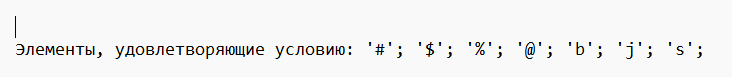
}

}

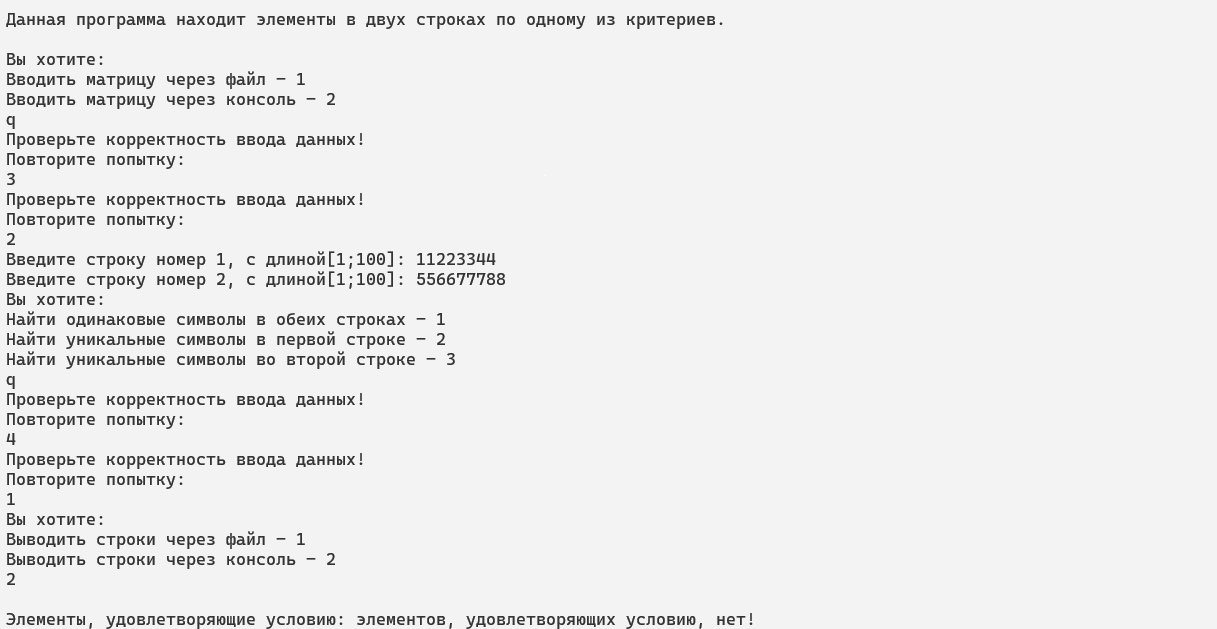
**Скриншоты:**

**Delphi:**

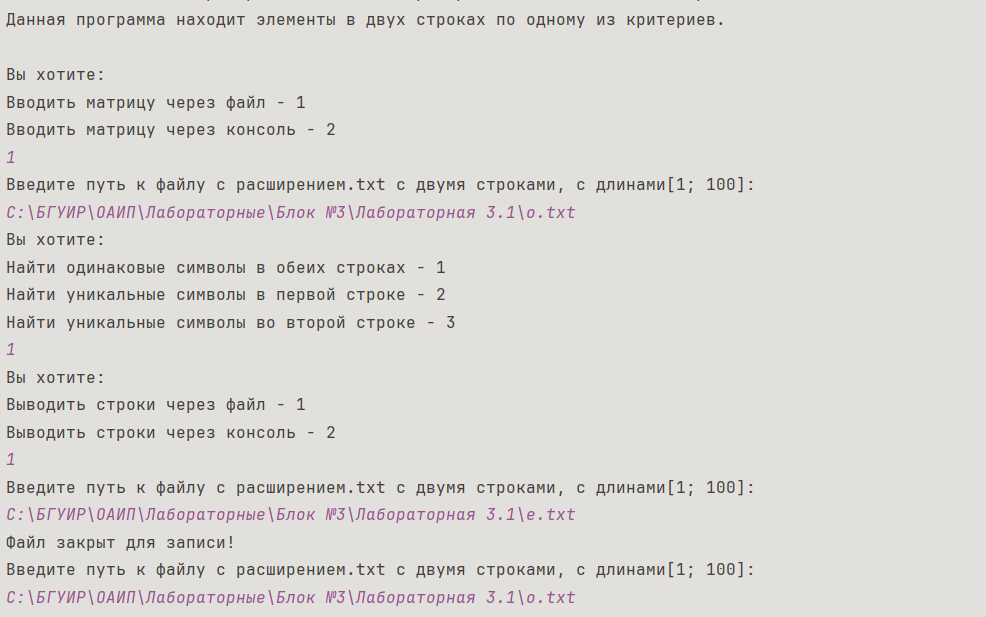


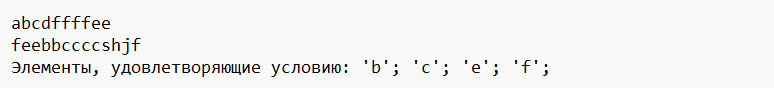


**C++:**



**Java:**





**Блок-схем**















