УО «Белорусский государственный университет информатики и радиоэлектроники»

Кафедра ПОИТ

Отчет по лабораторной работе №5.1

по предмету «Основы алгоритмизации и программирования»

Вариант 18

Выполнил:

Егоров А.С.

Гр. 351005

Проверил:

Данилова Г. В.

Минск 2023

**Задание:**

Списки. Разработать программу слияния двух односвязных

упорядоченных по неубыванию линейных списков в один

упорядоченный список, изменив только указатели в исходных списках.

**Код программы Delphi:**

**UnitMain.pas**

Unit MainFormUnit5\_1;

Interface

Uses

Winapi.Windows, Winapi.Messages, System.SysUtils, System.Variants,

System.Classes, Vcl.Graphics,

Vcl.Controls, Vcl.Forms, Vcl.Dialogs, Vcl.Buttons, System.ImageList,

Vcl.ImgList, Vcl.Grids, Vcl.StdCtrls, Vcl.Menus,

InstructionUnit5\_1, AboutTheDeveloperUnit5\_1, BackendUnit5\_1,

OutputSortedArrayUnit5\_1, ExitUnit5\_1, System.Generics.Collections;

Type

// форма

TuVCLMain = Class(TForm)

LbTaskInfo: TLabel;

LbSize1Info: TLabel;

ESize1: TEdit;

BtAcceptSize1: TButton;

StrGrList1: TStringGrid;

ImageList1: TImageList;

OpenDialog1: TOpenDialog;

SaveDialog1: TSaveDialog;

MainMenu1: TMainMenu;

BtFile: TMenuItem;

BtOpenFile: TMenuItem;

BtSaveFile: TMenuItem;

BtInstruction: TMenuItem;

BtAboutTheDeveloper: TMenuItem;

LbList1Info: TLabel;

BitBtMergeLists: TBitBtn;

BitBtShowList: TBitBtn;

StrGrList2: TStringGrid;

LbList2Info: TLabel;

LbSize2Info: TLabel;

ESize2: TEdit;

BtAcceptSize2: TButton;

Procedure ESize1KeyDown(Sender: TObject; Var Key: Word;

Shift: TShiftState);

Procedure ESize1KeyPress(Sender: TObject; Var Key: Char);

Procedure BtInstructionClick(Sender: TObject);

Procedure BtAboutTheDeveloperClick(Sender: TObject);

Procedure ESize1Change(Sender: TObject);

Procedure StrGrList1KeyDown(Sender: TObject; Var Key: Word;

Shift: TShiftState);

Procedure BtAccept1SizeClick(Sender: TObject);

Procedure StrGrListKeyPress(Sender: TObject; Var Key: Char);

Procedure BitBtMergeListsClick(Sender: TObject);

Procedure BitBtShowListClick(Sender: TObject);

Procedure BtOpenFileClick(Sender: TObject);

Procedure BtSaveFileClick(Sender: TObject);

Procedure StrGrList1KeyUp(Sender: TObject; Var Key: Word;

Shift: TShiftState);

Procedure BitBtMergeListsKeyDown(Sender: TObject; Var Key: Word;

Shift: TShiftState);

Procedure FormCloseQuery(Sender: TObject; Var CanClose: Boolean);

Procedure FormCreate(Sender: TObject);

Procedure FormDestroy(Sender: TObject);

Procedure BtAcceptSize2Click(Sender: TObject);

Procedure ESize2Change(Sender: TObject);

Procedure StrGrList2KeyUp(Sender: TObject; Var Key: Word;

Shift: TShiftState);

Private

IsFileSaved: Boolean;

IsList1Filled: Boolean;

IsList2Filled: Boolean;

IsListDecreasing: Boolean;

IsSortButtonPressed: Boolean;

WasChanges1: Boolean;

WasChanges2: Boolean;

BufferHandler: TBufferHandler;

Public

{ Public declarations }

End;

Var

UVCLMain: TuVCLMain;

Implementation

{$R \*.dfm}

Procedure TuVCLMain.BtAboutTheDeveloperClick(Sender: TObject);

Begin

Application.CreateForm(TuVCLAboutTheDeveloper, UVCLAboutTheDeveloper);

UVCLAboutTheDeveloper.ShowModal;

UVCLAboutTheDeveloper.Destroy();

UVCLAboutTheDeveloper := Nil;

End;

Procedure TuVCLMain.BtAcceptSize2Click(Sender: TObject);

Const

MAX\_SIZE: Integer = 1000;

MIN\_SIZE: Integer = 0;

Var

I: Integer;

Size: Integer;

Begin

// creating Grid

Size := StrToInt(ESize2.Text);

If WasChanges2 Then

Begin

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

Begin

// list2

StrGrList2.RowCount := 2;

StrGrList2.ColCount := Size + 1;

StrGrList2.FixedCols := 1;

StrGrList2.FixedRows := 1;

StrGrList2.Cells[0, 0] := '№';

StrGrList2.Cells[0, 1] := 'Элемент';

For I := 1 To Size Do

Begin

StrGrList2.Cells[I, 0] := IntToStr(I);

StrGrList2.Cells[I, 1] := '';

End;

StrGrList2.Enabled := True;

StrGrList2.Visible := True;

LbList2Info.Visible := True;

If StrGrList1.Visible Then

Begin

BitBtMergeLists.Visible := True;

BitBtShowList.Visible := True;

BitBtMergeLists.Enabled := False;

BitBtShowList.Enabled := False;

End;

BtAcceptSize2.Enabled := False;

End

Else

MessageBox(UVCLMain.Handle,

'Размер не соответствует границам! Проверьте данные.', 'Ой-йой',

MB\_ICONERROR);

End;

WasChanges2 := False;

End;

Procedure TuVCLMain.BtInstructionClick(Sender: TObject);

Begin

Application.CreateForm(TuVCLInstruction, UVCLInstruction);

UVCLInstruction.ShowModal;

UVCLInstruction.Destroy();

UVCLInstruction := Nil;

End;

Procedure TuVCLMain.BtOpenFileClick(Sender: TObject);

Var

FileReader: TFileReader;

I: Integer;

Arr: TArray;

List1, List2: TList<Integer>;

A : Word;

b : TShiftState;

Begin

If OpenDialog1.Execute() Then

Begin

FileReader := TFileReader.Create();

FileReader.FileName := OpenDialog1.FileName;

FileReader.CheckFile();

// TODO статусы

If FileReader.Status = FsGood Then

Begin

List1 := FileReader.ReadList(FileReader.ReadSize(1),1);

List2 := FileReader.ReadList(FileReader.ReadSize(2),2);

If (FileReader.Status = FsGood) Then

Begin

ESize1.Text := IntToStr(List1.Count);

BtAcceptSize1.Click;

ESize2.Text := IntToStr(List2.Count);

BtAcceptSize2.Click;

For I := 1 To List1.Count Do

Begin

StrGrList1.Cells[I, 0] := IntToStr(I);

StrGrList1.Cells[I, 1] := IntToStr(List1[I - 1]);

End;

StrGrList1KeyUp(Sender,a,b);

For I := 1 To List2.Count Do

Begin

StrGrList2.Cells[I, 0] := IntToStr(I);

StrGrList2.Cells[I, 1] := IntToStr(List2[I - 1]);

End;

StrGrList2KeyUp(Sender,a,b);

End

else

MessageBox(UVCLMain.Handle, ListOfMessages[FileReader.Status],

'Ой-йой', MB\_ICONERROR);

End

Else

MessageBox(UVCLMain.Handle, ListOfMessages[FileReader.Status],

'Ой-йой', MB\_ICONERROR);

End;

End;

Procedure TuVCLMain.BtSaveFileClick(Sender: TObject);

Var

FileWriter: TFileWriter;

Begin

If SaveDialog1.Execute() Then

Begin

FileWriter := TFileWriter.Create();

FileWriter.FileName := SaveDialog1.FileName;

FileWriter.CheckFile();

If FileWriter.Status = FsGood Then

Begin

FileWriter.WriteList(MergedList);

If FileWriter.Status <> FsGood Then

Begin

MessageBox(UVCLMain.Handle, ListOfMessages[FileWriter.Status],

'Ой-йой', MB\_ICONERROR);

End

else

IsFileSaved := True;

End

Else

MessageBox(UVCLMain.Handle, ListOfMessages[FileWriter.Status],

'Ой-йой', MB\_ICONERROR);

FileWriter.Destroy();

End;

End;

Procedure TuVCLMain.BitBtShowListClick(Sender: TObject);

Begin

Application.CreateForm(TuVCLOutputSortedArray, UVCLOutputSortedArray);

UVCLOutputSortedArray.Show;

If UVCLOutputSortedArray.IsClosed Then

Begin

UVCLOutputSortedArray.Destroy;

UVCLOutputSortedArray := Nil;

End;

End;

Procedure TuVCLMain.BtAccept1SizeClick(Sender: TObject);

Const

MAX\_SIZE: Integer = 1000;

MIN\_SIZE: Integer = 0;

Var

I: Integer;

Size: Integer;

Begin

// creating Grid

Size := StrToInt(ESize1.Text);

If WasChanges1 Then

Begin

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

Begin

// list1

StrGrList1.RowCount := 2;

StrGrList1.ColCount := Size + 1;

StrGrList1.FixedCols := 1;

StrGrList1.FixedRows := 1;

StrGrList1.Cells[0, 0] := '№';

StrGrList1.Cells[0, 1] := 'Элемент';

For I := 1 To Size Do

Begin

StrGrList1.Cells[I, 0] := IntToStr(I);

StrGrList1.Cells[I, 1] := '';

End;

StrGrList1.Enabled := True;

StrGrList1.Visible := True;

LbList1Info.Visible := True;

If StrGrList2.Visible Then

Begin

BitBtMergeLists.Visible := True;

BitBtShowList.Visible := True;

BitBtMergeLists.Enabled := False;

BitBtShowList.Enabled := False;

End;

BtAcceptSize1.Enabled := False;

End

Else

MessageBox(UVCLMain.Handle,

'Размер не соответствует границам! Проверьте данные.', 'Ой-йой',

MB\_ICONERROR);

End;

WasChanges1 := False;

End;

Procedure TuVCLMain.ESize1Change(Sender: TObject);

Var

I, J: Integer;

SizeEdit: TEdit;

TempStr: String;

Begin

SizeEdit := TEdit(Sender);

BtAcceptSize1.Enabled := Not String.IsNullOrEmpty(SizeEdit.Text);

// проверка на вставку

BufferHandler.EditText := SizeEdit.Text;

If Not BufferHandler.CheckInput(TpInteger) Then

Begin

MessageBox(UVCLMain.Handle, 'Вы ввели неправильные смиволы!', 'Ой-йой',

MB\_ICONERROR);

SizeEdit.Text := '';

End;

// ведущие 0

BufferHandler.DeleteLeadingZeros(TpInteger);

SizeEdit.Text := BufferHandler.EditText;

If Not BtAcceptSize1.Enabled Then

Begin

StrGrList1.Enabled := False;

End;

If WasChanges1 Then

Begin

// Очистка всех ячеек StringGrid

For I := 1 To StrGrList1.RowCount Do

Begin

For J := 1 To StrGrList1.ColCount Do

Begin

StrGrList1.Cells[J - 1, I - 1] := '';

End;

End;

IsList1Filled := False;

LbList1Info.Visible := False;

StrGrList1.Visible := False;

BitBtMergeLists.Visible := False;

BitBtShowList.Visible := False;

BitBtMergeLists.Enabled := False;

BitBtShowList.Enabled := False;

BtSaveFile.Enabled := False;

End;

WasChanges1 := True;

End;

Procedure TuVCLMain.ESize2Change(Sender: TObject);

Var

I, J: Integer;

SizeEdit: TEdit;

TempStr: String;

Begin

SizeEdit := TEdit(Sender);

BtAcceptSize2.Enabled := Not String.IsNullOrEmpty(SizeEdit.Text);

// проверка на вставку

BufferHandler.EditText := SizeEdit.Text;

If Not BufferHandler.CheckInput(TpInteger) Then

Begin

MessageBox(UVCLMain.Handle, 'Вы ввели неправильные смиволы!', 'Ой-йой',

MB\_ICONERROR);

SizeEdit.Text := '';

End;

// ведущие 0

BufferHandler.DeleteLeadingZeros(TpInteger);

SizeEdit.Text := BufferHandler.EditText;

If Not BtAcceptSize2.Enabled Then

Begin

StrGrList2.Enabled := False;

End;

If WasChanges2 Then

Begin

// Очистка всех ячеек StringGrid

For I := 1 To StrGrList2.RowCount Do

Begin

For J := 1 To StrGrList2.ColCount Do

Begin

StrGrList2.Cells[J - 1, I - 1] := '';

End;

End;

IsList2Filled := False;

LbList2Info.Visible := False;

StrGrList2.Visible := False;

BitBtMergeLists.Visible := False;

BitBtShowList.Visible := False;

BitBtMergeLists.Enabled := False;

BitBtShowList.Enabled := False;

BtSaveFile.Enabled := False;

End;

WasChanges2 := True;

End;

Procedure TuVCLMain.BitBtMergeListsClick(Sender: TObject);

Var

List1, List2: TList<Integer>;

I: Integer;

Size1, Size2: Integer;

Begin

Size1 := StrToInt(ESize1.Text);

Size2 := StrToInt(ESize2.Text);

List1 := TList<Integer>.Create;

List2 := TList<Integer>.Create;

If Not IsListDecreasing Then

Begin

For I := 1 To Size1 Do

Begin

List1.Add(StrToInt(StrGrList1.Cells[I, 1]));

End;

For I := 1 To Size2 Do

Begin

List2.Add(StrToInt(StrGrList2.Cells[I, 1]));

End;

MergedList := MergeTwoLists(List1, List2);

End

Else

MessageBox(UVCLMain.Handle, 'Списки введены не по возрастанию!',

'ой-йой', MB\_ICONEXCLAMATION);

BitbtShowList.Enabled := True;

BtSaveFile.Enabled := True;

IsFileSaved := False;

IsSortButtonPressed := True;

End;

Procedure TuVCLMain.StrGrList1KeyUp(Sender: TObject; Var Key: Word;

Shift: TShiftState);

Var

Counter, I: Integer;

Size: Integer;

Begin

Counter := 0;

Size := StrGrList1.ColCount - 1;

IsListDecreasing := False;

For I := 1 To Size Do

Begin

If (Length(StrGrList1.Cells[I, 1]) <> 0) And

(StrGrList1.Cells[I, 1] <> '-') Then

Begin

If (I = 1) Then

Inc(Counter)

Else If ((Length(StrGrList1.Cells[I - 1, 1]) <> 0) And

(StrGrList1.Cells[I - 1, 1] <> '-')) Then

Begin

Inc(Counter);

If (StrToInt(StrGrList1.Cells[I - 1, 1]) >

StrToInt(StrGrList1.Cells[I, 1])) Then

IsListDecreasing := True;

End;

End;

End;

If Counter = Size Then

IsList1Filled := True

Else

Begin

IsList1Filled := False;

BitBtShowList.Enabled := False;

BitBtMergeLists.Enabled := False;

BtSaveFile.Enabled := False;

End;

If IsList2Filled Then

BitBtMergeLists.Enabled := IsList1Filled;

End;

Procedure TuVCLMain.StrGrList2KeyUp(Sender: TObject; Var Key: Word;

Shift: TShiftState);

Var

Counter, I: Integer;

Size: Integer;

Begin

Counter := 0;

Size := StrGrList2.ColCount - 1;

IsListDecreasing := False;

For I := 1 To Size Do

Begin

If (Length(StrGrList2.Cells[I, 1]) <> 0) And

(StrGrList2.Cells[I, 1] <> '-') Then

Begin

If (I = 1) Then

Inc(Counter)

Else If ((Length(StrGrList2.Cells[I - 1, 1]) <> 0) And

(StrGrList2.Cells[I - 1, 1] <> '-')) Then

Begin

Inc(Counter);

If (StrToInt(StrGrList2.Cells[I - 1, 1]) >

StrToInt(StrGrList2.Cells[I, 1])) Then

IsListDecreasing := True;

End;

End;

End;

If Counter = Size Then

IsList2Filled := True

Else

Begin

IsList2Filled := False;

BitBtShowList.Enabled := False;

BitBtMergeLists.Enabled := False;

BtSaveFile.Enabled := False;

End;

If IsList1Filled Then

BitBtMergeLists.Enabled := IsList2Filled;

End;

Procedure TuVCLMain.FormCloseQuery(Sender: TObject; Var CanClose: Boolean);

Begin

If Not IsSortButtonPressed Or IsFileSaved Then

Begin

Application.CreateForm(TuVCLExit, UVCLExit);

UVCLExit.ShowModal;

CanClose := UVCLExit.GetStatus();

UVCLExit.Destroy();

End

Else If IsSortButtonPressed Then

Begin

Repeat

ExitCode := MessageBox(UVCLMain.Handle,

'Сохранить данные в файл перед выходом?',

'Подверждение', MB\_ICONQUESTION + MB\_YESNOCANCEL);

If ExitCode = ID\_YES Then

Begin

BtSaveFileClick(UVCLMain);

CanClose := True;

End

Else If ExitCode = ID\_NO Then

CanClose := True

Else

CanClose := False;

Until IsFileSaved Or (ExitCode = ID\_NO) Or (ExitCode = ID\_CANCEL);

End;

End;

Procedure TuVCLMain.FormCreate(Sender: TObject);

Begin

BufferHandler := TBufferHandler.Create();

End;

Procedure TuVCLMain.FormDestroy(Sender: TObject);

Begin

BufferHandler.Destroy();

End;

End.

**Backend.pas**

Unit BackendUnit5\_1;

Interface

Uses System.SysUtils, System.Generics.Collections;

Type

TTypes = (TpInteger, TpUInteger, TpReal, TpString);

TBufferHandler = Class

Private

FEditText: String;

Function CountSymbol(Const Symbol: Char): Integer;

Public

Function CheckInput(Const InputType: TTypes): Boolean;

Procedure DeleteLeadingZeros(Const InputType: TTypes);

Property EditText: String Read FEditText Write FEditText;

End;

TFileStatus = (FsGood, FsNotFound, FsNotTxt, FsNotReadable, FsNotWritable,

FsEmpty, FsWrongData, FsUnexpected);

TFileReader = Class

Private

FFileName: String;

FInFile: TextFile;

FFileStatus: TFileStatus;

Function IsFileTxt(): Boolean;

Function IsFileReadable(): Boolean;

Function IsEmpty(): Boolean;

Procedure SetFileName(Const FFileName: String);

Public

Property Status: TFileStatus Read FFileStatus;

Property FileName: String Read FFileName Write SetFileName;

Procedure CheckFile();

Function ReadSize(Const NumberOfList: Integer): Integer;

Function ReadList(Const Size, NumberOfList: Integer): TList<Integer>;

End;

TFileWriter = Class

Private

FFileName: String;

FOutFile: TextFile;

FFileStatus: TFileStatus;

Function IsFileTxt(): Boolean;

Function IsFileWritable(): Boolean;

Procedure SetFileName(Const FFileName: String);

Public

Constructor Create();

Property Status: TFileStatus Read FFileStatus;

Property FileName: String Read FFileName Write SetFileName;

Procedure CheckFile();

Procedure WriteList(Var List: TList<Integer>);

End;

TListOfMessages = Array [TFileStatus] Of PWideChar;

Const

ListOfMessages: TListOfMessages = ('Информация записана!',

'Файл не найден! Повторите ещё раз.',

'Файл не текстовый! Повторите ещё раз.',

'Файл не доступен для чтения! Повторите ещё раз.',

'Файл не доступен для записи! Повторите ещё раз.',

'Файл пустой! Повторите ещё раз.',

'Не верные данные в файле! Повторите ещё раз.',

'Упс... Что-то пошло не так. Потворите ещё раз.');

Function MergeTwoLists(Var List1, List2: TList<Integer>): TList<Integer>;

Var

MergedList: TList<Integer>;

Implementation

{ MergeFunction }

Function MergeTwoLists(Var List1, List2: TList<Integer>): TList<Integer>;

Var

LeftPtr: Integer;

RightPtr: Integer;

Steps: Integer;

List3: TList<Integer>;

Begin

LeftPtr := 0;

RightPtr := 0;

Steps := List1.Count + List2.Count;

List3 := TList<Integer>.Create;

While ((LeftPtr < List1.Count) Or (RightPtr < List2.Count)) Do

Begin

If LeftPtr > List1.Count - 1 Then

Begin

While RightPtr < List2.Count Do

Begin

List3.Add(List2[RightPtr]);

Inc(RightPtr);

End;

End

Else If RightPtr > List2.Count - 1 Then

Begin

While LeftPtr < List1.Count Do

Begin

List3.Add(List1[LeftPtr]);

Inc(LeftPtr);

End;

End

Else If List1[LeftPtr] > List2[RightPtr] Then

Begin

List3.Add(List2[RightPtr]);

Inc(RightPtr);

End

Else

Begin

List3.Add(List1[LeftPtr]);

Inc(LeftPtr);

End;

End;

MergeTwoLists := List3;

End;

{ TBufferHandler }

Function TBufferHandler.CheckInput(Const InputType: TTypes): Boolean;

Const

GOOD\_KEYS: Set Of Char = ['0' .. '9'];

Var

Status: Boolean;

I: Integer;

CountOfMinuses: Integer;

CountOfCommas: Integer;

Begin

Status := True;

Case InputType Of

TpInteger:

Begin

CountOfMinuses := CountSymbol('-');

If (CountOfMinuses = 0) Then

Begin

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

If Status And Not(FEditText[I] In GOOD\_KEYS) Then

Status := False;

End

Else If (CountOfMinuses = 1) Then

Begin

For I := 2 To High(FEditText) Do

If Status And Not(FEditText[I] In GOOD\_KEYS) Then

Status := False;

End

Else

Status := False;

End;

TpUInteger:

Begin

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

If Status And Not(FEditText[I] In GOOD\_KEYS) Then

Status := False;

End;

TpReal:

Begin

CountOfMinuses := CountSymbol('-');

CountOfCommas := CountSymbol(',');

If (CountOfMinuses = 0) And (CountOfCommas = 0) Then

Begin

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

If Status And Not(FEditText[I] In GOOD\_KEYS) Then

Status := False;

End

Else If (CountOfMinuses = 1) And (CountOfCommas = 0) Then

Begin

For I := 2 To High(FEditText) Do

If Status And Not(FEditText[I] In GOOD\_KEYS) Then

Status := False;

End

Else If (CountOfMinuses = 0) And (CountOfCommas = 1) Then

Begin

For I := 1 To High(FEditText) Do

If Status And

Not((FEditText[I] In GOOD\_KEYS) Or

(FEditText[I] = ',')) Then

Status := False;

End

Else If (CountOfMinuses = 1) And (CountOfCommas = 1) Then

Begin

For I := 2 To High(FEditText) Do

If Status And

Not((FEditText[I] In GOOD\_KEYS) Or

(FEditText[I] = ',')) Then

Status := False;

End

Else

Status := False;

End;

TpString:

Begin

// why?

End;

End;

CheckInput := Status;

End;

Function TBufferHandler.CountSymbol(Const Symbol: Char): Integer;

Var

I: Integer;

Count: Integer;

Begin

Count := 0;

For I := 1 To Length(FEditText) Do

If FEditText[I] = Symbol Then

Inc(Count);

CountSymbol := Count;

End;

Procedure TBufferHandler.DeleteLeadingZeros(Const InputType: TTypes);

Begin

Case InputType Of

TpInteger:

Begin

If CountSymbol('-') = 1 Then

While (Length(FEditText) > 1) And (FEditText[2] = '0') Do

Delete(FEditText, 2, 1)

Else

While (Length(FEditText) > 0) And (FEditText[1] = '0') Do

Delete(FEditText, 1, 1);

End;

TpUInteger:

While (Length(FEditText) > 0) And (FEditText[1] = '0') Do

Delete(FEditText, 1, 1);

TpReal:

// TODO

;

TpString:

// Why?

;

End;

End;

{ TFileReader }

Procedure TFileReader.CheckFile();

Begin

If Not FileExists(FileName) Then

FFileStatus := FsNotFound

Else If Not IsFileTxt() Then

FFileStatus := FsNotTxt

Else If Not IsFileReadable() Then

FFileStatus := FsNotReadable

Else If IsEmpty() Then

FFileStatus := FsEmpty

Else

FFileStatus := FsGood;

End;

function TFileReader.IsEmpty: Boolean;

Var

Status: Boolean;

Begin

Try

Reset(FInFile);

Status := Eof(FInFile);

CloseFile(FInFile);

Except

End;

IsEmpty := Status;

End;

Function TFileReader.IsFileReadable(): Boolean;

Var

Status: Boolean;

Begin

Status := True;

Try

Reset(FInFile);

CloseFile(FInFile);

Except

Status := False;

End;

IsFileReadable := Status;

End;

Function TFileReader.IsFileTxt(): Boolean;

Var

FileType: String;

Status: Boolean;

Begin

FileType := FileName.Substring(FileName.Length - 4);

If FileType = '.txt' Then

Status := True

Else

Status := False;

IsFileTxt := Status;

End;

Function TFileReader.ReadList(Const SIZE, NumberOfList: Integer)

: TList<Integer>;

Var

List: TList<Integer>;

I: Integer;

Num: Integer;

Begin

List := TList<Integer>.Create;

Try

ReSet(FInFile);

If NumberOfList = 2 Then

Begin

Readln(FInFile);

Readln(FInFile);

End;

Readln(FInFile);

For I := 1 To SIZE Do

Begin

if Eof(FInFile) then

FFileStatus := FsWrongData;

Read(FInFile, Num);

List.Add(Num);

End;

CloseFile(FInFile);

Except

FFileStatus := FsUnexpected;

End;

ReadList := List;

End;

Function TFileReader.ReadSize(Const NumberOfList: Integer): Integer;

Var

Size: Integer;

Begin

Size := 0;

Try

ReSet(FInFile);

If (NumberOfList = 2) Then

Begin

Readln(FInFile);

Readln(FInFile);

End;

Readln(FInFile, Size);

CloseFile(FInFile);

Except

FFileStatus := FsUnexpected;

End;

if Size < 1 then

FFileStatus := FsWrongData;

ReadSize := Size;

End;

Procedure TFileReader.SetFileName(Const FFileName: String);

Begin

Self.FFileName := FFileName;

Assign(FInFile, Self.FFileName);

End;

{ TFileWriter }

Procedure TFileWriter.CheckFile;

Begin

If Not FileExists(FileName) Then

FFileStatus := FsNotFound

Else If Not IsFileWritable() Then

FFileStatus := FsNotWritable

Else

FFileStatus := FsGood;

End;

Function TFileWriter.IsFileWritable(): Boolean;

Var

Status: Boolean;

Begin

Status := True;

Try

Rewrite(FOutFile);

CloseFile(FOutFile);

Except

Status := False;

End;

IsFileWritable := Status;

End;

Procedure TFileWriter.SetFileName(Const FFileName: String);

Begin

Self.FFileName := FFileName;

Assign(FOutFile, Self.FFileName);

End;

Procedure TFileWriter.WriteList(Var List: TList<Integer>);

Var

I: Integer;

Begin

Try

ReWrite(FOutFile);

Writeln(FOutFile, 'Слитый воедино лист:');

For I := 1 To List.Count Do

Write(FOutFile, List[I-1], ' ');

Writeln(FOutFile);

CloseFile(FOutFile);

Except

FFileStatus := FsUnexpected;

End;

End;

Constructor TFileWriter.Create();

Begin

End;

Function TFileWriter.IsFileTxt: Boolean;

Var

FileType: String;

Status: Boolean;

Begin

FileType := FileName.Substring(FileName.Length - 4);

If FileType = '.txt' Then

Status := True

Else

Status := False;

IsFileTxt := Status;

End;

End.

**Код программы СSharp:**

**ErrorMesagges.cs**

public enum ErrorMessages  
{  
 EmWrongKey,  
 EmWrongType,  
 EmWrongBoundOfNumber,  
 EmWrongSequence  
}

**FileStatus.cs**

public enum FileStatus  
{  
 FsGood,  
 FsNotFound,  
 FsNotTxt,  
 FsNotReadable,  
 FsNotWritable,  
 FsEmpty,  
 FsWrongDataType,  
 FsWrongCount,  
 FsUnexpacted  
}

**WorkingTypes.cs**

public enum WorkingTypes  
{  
 WtConsole = 1,  
 WtFile  
}

**TableMenuStatus.cs**

enum TableMenuStatus  
{  
 TmsAddWorker = 1,  
 TmsDeleteWorker,  
 TmsChangeWorker,  
 TmsExitToMainMenu  
}

**Worker.cs**

public record Worker()  
{  
 public int WId { get; set; }  
 public string? WSurname { get; set; }  
 public string? WCompany { get; set; }  
 public int WCountOfDetailsA { get; set; }  
 public int WCountOfDetailsB { get; set; }  
 public int WCountOfDetailsC { get; set; }  
};

**Workers.cs**

public class Workers  
{  
 private List<Worker> listOfWorkers = new List<Worker>();  
  
 public Worker this[int i]  
 {  
 get => listOfWorkers[i];  
 set => listOfWorkers[i] = value;  
 }  
  
 public int Count => listOfWorkers.Count;  
  
 public void ChangeWorker(int id, Worker worker)  
 {  
 worker.WId = id - 1;  
 listOfWorkers[id - 1] = worker;  
 }  
  
 public void AddWorker(Worker worker)  
 {  
 worker.WId = listOfWorkers.Count;  
 listOfWorkers.Add(worker);  
 }  
  
 public void DeleteWorker(int id)  
 {  
 Worker worker;  
 listOfWorkers.Remove(listOfWorkers[id - 1]);  
 // move all indexes after we were removed worker with id  
 for (int i = id; i < listOfWorkers.Count; i++)  
 {  
 listOfWorkers[i].WId = i + 1;  
 }  
 }  
  
}

**Program.cs**

using Lab5\_1;  
  
MainMenu mainMenu = new MainMenu();  
  
int choose = 0;  
int size = 0;  
List<int> list1 = null;   
List<int> list2 = null;  
List<int> mergedList = null;  
Merger merger = new Merger();  
  
mainMenu.ShowProgramInfo();  
// input block  
do  
{  
 mainMenu.InputShowMenu();  
 choose = mainMenu.InputChoice();  
 // initialing reader  
 switch ((WorkingTypes)choose)  
 {  
 case WorkingTypes.WtConsole:  
 {  
 ConsoleReader consoleReader = new ConsoleReader();  
 do  
 {  
 list1 = consoleReader.InputElementsOfList(consoleReader.

InputSizeOfList());  
 } while (merger.IsListDecreasing(list1));  
 do  
 {  
 list2 = consoleReader.InputElementsOfList(consoleReader.

InputSizeOfList());  
 } while (merger.IsListDecreasing(list2));  
 }  
 break;  
 case WorkingTypes.WtFile:  
 {  
 FileReader fileReader = new FileReader();  
 do  
 {  
 fileReader.FilePath = mainMenu.InputFilePath();  
 // reading size and elements and checking status every operation  
 if (fileReader.FileStatus == FileStatus.FsGood)  
 {  
 size = fileReader.InputSizeOfList();  
 }  
 if (fileReader.FileStatus == FileStatus.FsGood)  
 {  
 list1 = fileReader.InputElementsOfList(size);  
 }  
 if (fileReader.FileStatus == FileStatus.FsGood)  
 {  
 size = fileReader.InputSizeOfList();  
 }  
 if (fileReader.FileStatus == FileStatus.FsGood)  
 {  
 list2 = fileReader.InputElementsOfList(size);  
 }  
 if (fileReader.FileStatus == FileStatus.FsGood &&  
 (merger.IsListDecreasing(list1) || merger.IsListDecreasing(list2)))  
 {  
 mainMenu.ShowWrongSequenceMessage();  
 fileReader.MakeWrongData();  
 }  
 else  
 mainMenu.ShowFileStatusMessage(fileReader.FileStatus);  
 } while (fileReader.FileStatus != FileStatus.FsGood);  
 }  
 break;  
 default:  
 {  
 mainMenu.ShowWrongKeyMessage();  
 choose = 0;  
 }  
 break;  
 }  
} while (choose < (int)WorkingTypes.WtConsole || (int)WorkingTypes.WtFile < choose);  
  
// main block  
mergedList = merger.MergeTwoLists(list1, list2);  
  
// output block  
do  
{  
 mainMenu.OutputShowMenu();  
 choose = mainMenu.InputChoice();  
 // initialing reader  
 switch ((WorkingTypes)choose)  
 {  
 case WorkingTypes.WtConsole:  
 {  
 ConsoleWriter consoleWriter = new ConsoleWriter();  
 consoleWriter.Output(mergedList);  
 }  
 break;  
 case WorkingTypes.WtFile:  
 {  
 FileWriter fileWriter = new FileWriter();  
 do  
 {  
 fileWriter.FilePath = mainMenu.InputFilePath();  
 if (fileWriter.FileStatus == FileStatus.FsGood)  
 fileWriter.Output(mergedList);  
 mainMenu.ShowFileStatusMessage(fileWriter.FileStatus);  
 } while (fileWriter.FileStatus != FileStatus.FsGood);  
 }  
 break;  
 default:  
 {  
 mainMenu.ShowWrongKeyMessage();  
 choose = 0;  
 }  
 break;  
 }  
} while (choose < 0 || (int)WorkingTypes.WtFile < choose);

**ConsoleReader.cs**  
  
public class ConsoleReader  
{  
 public int InputSizeOfList()  
 {  
 int size;  
 bool isBad = true;  
 do  
 {  
 Console.WriteLine("Введите размер списка: ");  
 if (int.TryParse(Console.ReadLine(), out size))  
 isBad = false;  
 else  
 Console.WriteLine(MainMenu.ErrorMessagesArr

[(int)ErrorMessages.EmWrongType]);  
 if (!isBad && size < 1)  
 {  
 Console.WriteLine(MainMenu.ErrorMessagesArr

[(int)ErrorMessages.EmWrongBoundOfNumber]);  
 isBad = true;  
 }  
 } while (isBad);  
 return size;  
 }  
  
 public List<int> InputElementsOfList( int size)  
 {  
 List<int> list = new List<int>();  
 bool isBad = true;  
 int number;  
 for (int i = 0; i < size; i++)  
 {  
 isBad = true;  
 do  
 {  
 Console.Write("Введите " + (i + 1) + " элемент: ");  
 if (int.TryParse(Console.ReadLine(), out number))  
 isBad = false;  
 else  
 Console.WriteLine(MainMenu.ErrorMessagesArr

[(int)ErrorMessages.EmWrongType]);  
 } while (isBad);  
 list.Add(number);  
 }  
 return list;  
 }  
}

**FileReader.cs**

public class FileReader  
{  
 // constructors   
 public FileReader()  
 {  
 filePath = null;  
 }  
  
 // filePath field  
 private string? filePath;  
  
 public string? FilePath  
 {  
 get => filePath;  
 set   
 {   
 filePath = value;  
 fileStatus = FileStatus.FsGood;  
 }  
  
 }  
  
 // fileStatus field  
 private FileStatus fileStatus;  
  
 public FileStatus FileStatus  
 {  
 get => CheckFileStatus();  
 private set => fileStatus = value;  
 }  
  
 // methods of object  
 private FileStatus CheckFileStatus()  
 {  
 FileInfo fileInfo = new FileInfo(filePath);  
 if (!fileInfo.Exists)  
 {  
 fileStatus = FileStatus.FsNotFound;  
 }  
 else if (!filePath.EndsWith(".txt"))  
 {  
 fileStatus = FileStatus.FsNotTxt;  
 }  
 return fileStatus;  
 }  
  
 public int InputSizeOfList()  
 {  
 int size = 0;  
 String? line = null;  
 try  
 {  
 using (StreamReader reader = new StreamReader(filePath))  
 {  
 try  
 {  
 line = reader.ReadLine();  
 if (line == null)  
 fileStatus = FileStatus.FsEmpty;  
 else if (!int.TryParse(line, out size))  
 fileStatus = FileStatus.FsWrongDataType;  
 else if (size < 1)  
 fileStatus = FileStatus.FsWrongDataType;  
 }  
 catch (IOException e)  
 {  
 fileStatus = FileStatus.FsUnexpacted;  
 }  
 }  
 }  
 catch (UnauthorizedAccessException e)  
 {  
 fileStatus = FileStatus.FsNotReadable;  
 }  
  
 return size;  
 }  
  
 public void MakeWrongData()  
 {  
 fileStatus = FileStatus.FsWrongDataType;  
 }  
 public List<int> InputElementsOfList(int size)  
 {  
 String? line = null;  
 String[] strElements = null;  
 char splitter = ' ';  
 List<int> list = new List<int>();  
 int number;  
 try  
 {  
 using (StreamReader reader = new StreamReader(filePath))  
 {  
 try  
 {  
 reader.ReadLine(); // skip reading size  
 line = reader.ReadLine();  
 strElements = line.Split(splitter);  
 }  
 catch (IOException e)  
 {  
 fileStatus = FileStatus.FsUnexpacted;  
 }  
 }  
 }  
 catch (UnauthorizedAccessException e)  
 {  
 fileStatus = FileStatus.FsNotReadable;  
 }  
 if (size != strElements.Length)  
 {  
 fileStatus = FileStatus.FsWrongCount;  
 }  
 for (int i = 0; i < size; i++)  
 {  
 if ((fileStatus == FileStatus.FsGood))  
 {  
 if (int.TryParse(strElements[i], out number))  
 list.Add(number);  
 else  
 fileStatus = FileStatus.FsWrongDataType;  
 }  
 }  
 return list;  
  
 }  
}

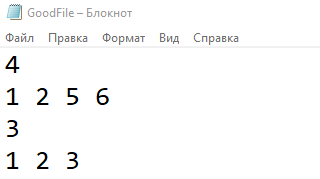
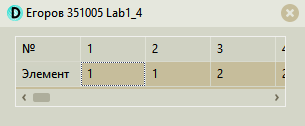
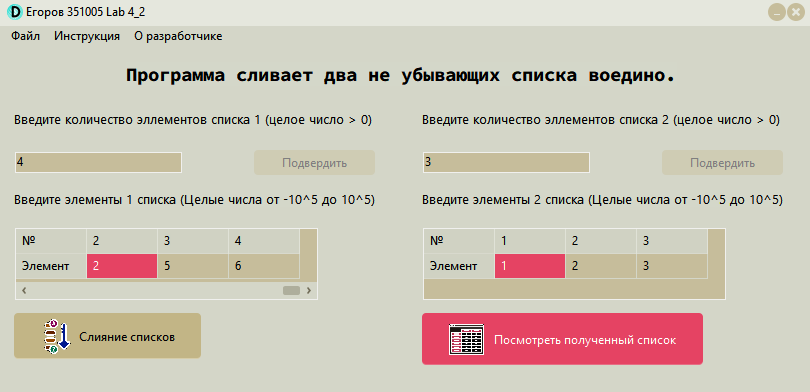
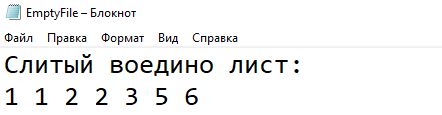
**ConsoleWriter.cs**  
  
public class ConsoleWriter  
{  
 public void Output(List<int> list)  
 {  
 Console.WriteLine("Слитый воедино список:");  
 for (int i = 0; i < list.Count; i++)  
 {  
 Console.Write(list[i] + " ");  
 }  
 Console.WriteLine();  
 }  
}

**FileWriter.cs**

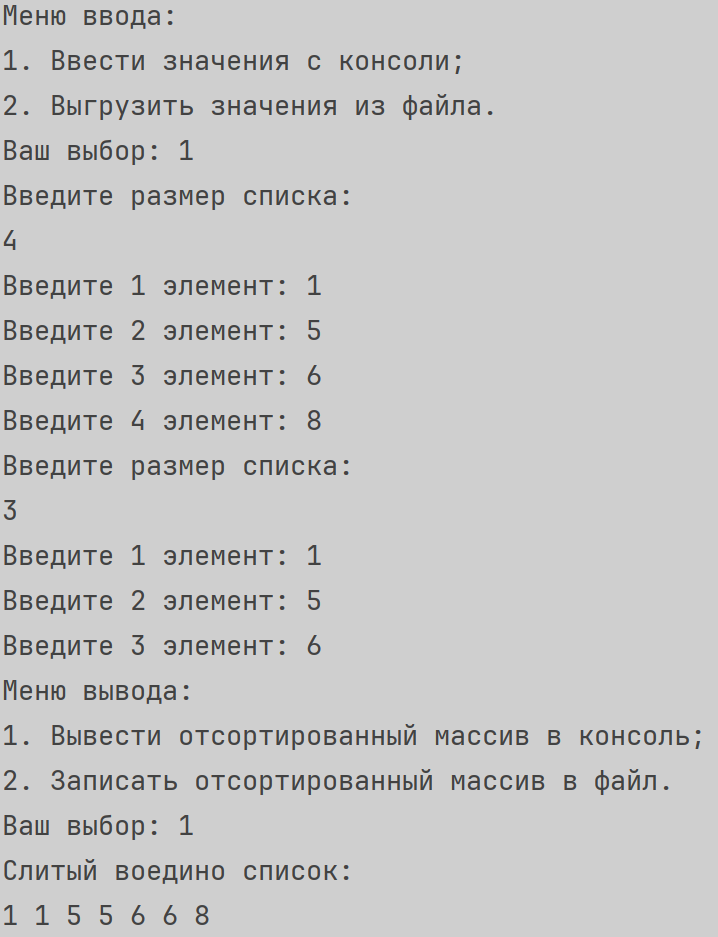
public class FileWriter  
{  
 // constructors   
 public FileWriter()  
 {  
 filePath = null;  
 }  
  
 public FileWriter(String? filePath)  
 {  
 this.filePath = filePath;  
 }  
  
 // filePath field  
 private string? filePath;  
  
 public string? FilePath  
 {  
 get => filePath;  
 set => filePath = value;  
 }  
  
 // fileStatus field  
 private FileStatus fileStatus;  
  
 public FileStatus FileStatus  
 {  
 get => CheckFileStatus();  
 private set => fileStatus = value;  
 }  
  
 // methods of object  
 private FileStatus CheckFileStatus()  
 {  
 FileInfo fileInfo = new FileInfo(filePath);  
 if (!fileInfo.Exists)  
 {  
 fileStatus = FileStatus.FsNotFound;  
 }  
 else if (!filePath.EndsWith(".txt"))  
 {  
 fileStatus = FileStatus.FsNotTxt;  
 }  
 return fileStatus;  
 }  
  
 public void Output(List<int> list)  
 {  
 try  
 {  
 using StreamWriter writer = new StreamWriter(filePath);  
 writer.WriteLine("Слитый воедино список:");  
 for (int i = 0; i < list.Count; i++)  
 {  
 writer.Write(list[i] + " ");  
 }  
  
 writer.WriteLine();  
 }  
 catch (UnauthorizedAccessException e)  
 {  
 fileStatus = FileStatus.FsNotWritable;  
 }  
 }  
}

**Скриншоты:**

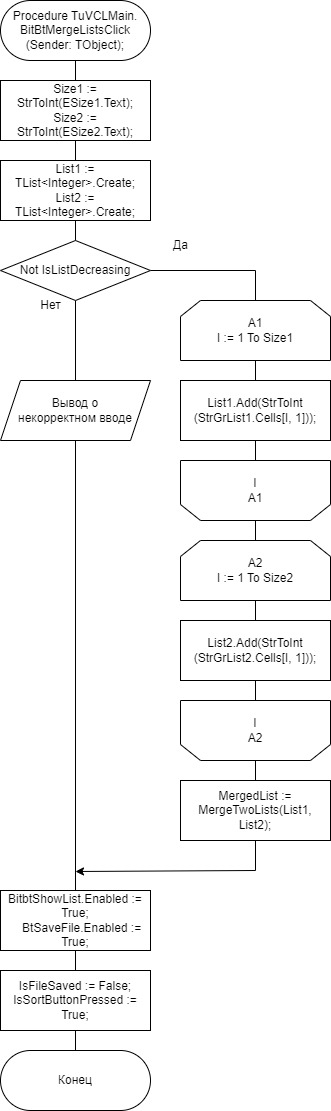
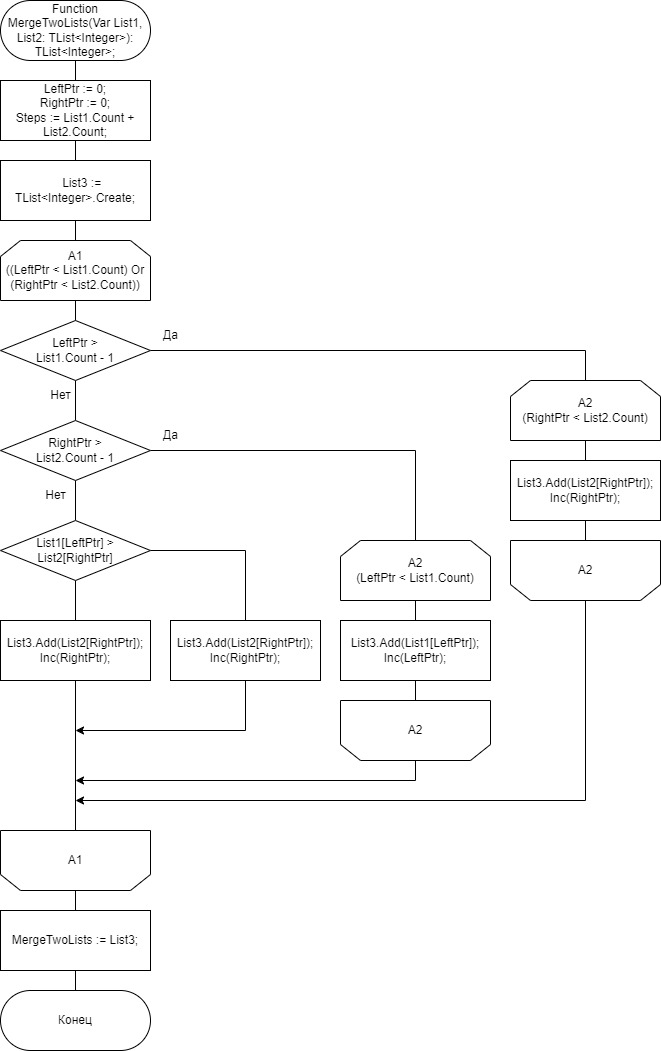
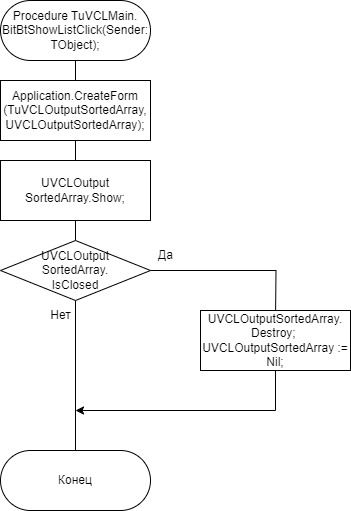
**Delphi:**

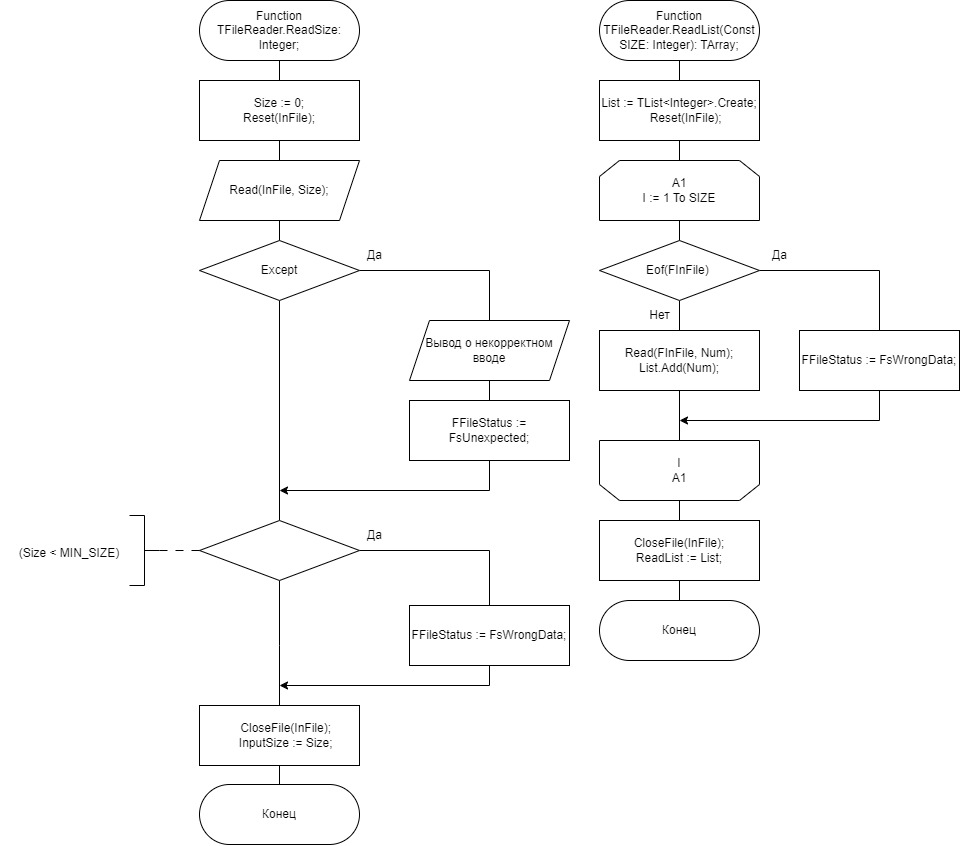


**СSharp:**



**Блок-схема:**



****

**UnitTests на CSharp**

**Код Lab5\_1.Tests.cs**

using System.Collections.Generic;  
using JetBrains.Annotations;  
using Lab5\_1;  
using Microsoft.VisualStudio.TestTools.UnitTesting;  
  
namespace Lab5\_1.Tests;  
  
[TestClass]  
[TestSubject(typeof(FileReader))]  
public class FileReaderTests  
{  
 private static FileReader fileReader = new FileReader();   
  
 [TestMethod]  
 public void CheckFileStatus\_givenOnlyWriteFile\_FsNotReadable()  
 {  
 fileReader.FilePath = "D:\\Уроки\\Уник\\ОАиП\\Лабы\\onlyWriteFile.txt";  
 fileReader.InputSizeOfList();  
 Assert.AreEqual(fileReader.FileStatus,FileStatus.FsNotReadable);  
 }  
   
 [TestMethod]  
 public void CheckFileStatus\_givenMp3File\_FsNotTxt()  
 {  
 fileReader.FilePath = "D:\\Уроки\\Уник\\ОАиП\\Лабы\\music.mp3";  
 Assert.AreEqual(fileReader.FileStatus,FileStatus.FsNotTxt);  
 }  
   
 [TestMethod]  
 public void CheckFileStatus\_givenUnexistenFile\_FsNotFound()  
 {  
 fileReader.FilePath = "D:\\Уроки\\Уник\\ОАиП\\Лабы\\NotExist.txt";  
 Assert.AreEqual(fileReader.FileStatus,FileStatus.FsNotFound);  
 }  
   
 [TestMethod]  
 public void CheckFileStatus\_givenWrongSizeFile\_FsWrongDataType()  
 {  
 fileReader.FilePath = "D:\\Уроки\\Уник\\ОАиП\\Лабы\\WrongSizeFile.txt";  
 fileReader.InputSizeOfList();  
 Assert.AreEqual(fileReader.FileStatus,FileStatus.FsWrongDataType);  
 }  
   
 [TestMethod]  
 public void CheckFileStatus\_givenWrongCountFile\_FsWrongCount()  
 {  
 fileReader.FilePath = "D:\\Уроки\\Уник\\ОАиП\\Лабы\\WrongCountFile.txt";  
 fileReader.InputElementsOfList(fileReader.InputSizeOfList());  
 Assert.AreEqual(fileReader.FileStatus,FileStatus.FsWrongCount);  
 }  
   
 [TestMethod]  
 public void CheckFileStatus\_givenWrongElementFile\_FsWrongDataType()  
 {  
 fileReader.FilePath = "D:\\Уроки\\Уник\\ОАиП\\Лабы\\WrongTypeFile.txt";  
 fileReader.InputElementsOfList(fileReader.InputSizeOfList());  
 Assert.AreEqual(fileReader.FileStatus,FileStatus.FsWrongDataType);  
 }  
   
 [TestMethod]  
 public void CheckFileStatus\_givenEmptyFile\_FsEmpty()  
 {  
 fileReader.FilePath = "D:\\Уроки\\Уник\\ОАиП\\Лабы\\EmptyFile.txt";  
 fileReader.InputSizeOfList();  
 Assert.AreEqual(fileReader.FileStatus,FileStatus.FsEmpty);  
 }  
   
 [TestMethod]  
 public void CheckFileStatus\_givenWrongElementFile\_FsGood()  
 {  
 fileReader.FilePath = "D:\\Уроки\\Уник\\ОАиП\\Лабы\\GoodFile.txt";  
 fileReader.InputElementsOfList(fileReader.InputSizeOfList());  
 Assert.AreEqual(fileReader.FileStatus,FileStatus.FsGood);  
 }  
}  
  
[TestClass]  
[TestSubject(typeof(FileWriter))]  
public class FileWriterTexts  
{  
 private static FileWriter fileWriter = new FileWriter();  
   
 [TestMethod]  
 public void CheckFileStatus\_givenOnlyReadFile\_FsNotWritable()  
 {  
 fileWriter.FilePath = "D:\\Уроки\\Уник\\ОАиП\\Лабы\\onlyReadFile.txt";  
 List<int> testList = new List<int>(){1,2,3,4};  
 fileWriter.Output(testList);  
 Assert.AreEqual( fileWriter.FileStatus,FileStatus.FsNotWritable);  
 }   
   
 [TestMethod]  
 public void CheckFileStatus\_givenMp3File\_FsNotTxt()  
 {  
 fileWriter.FilePath = "D:\\Уроки\\Уник\\ОАиП\\Лабы\\music.mp3";  
 Assert.AreEqual(fileWriter.FileStatus,FileStatus.FsNotTxt);  
 }  
   
 [TestMethod]  
 public void CheckFileStatus\_givenUnexistenFile\_FsNotFound()  
 {  
 fileWriter.FilePath = "D:\\Уроки\\Уник\\ОАиП\\Лабы\\NotExist.txt";  
 Assert.AreEqual(fileWriter.FileStatus,FileStatus.FsNotFound);  
 }  
   
 [TestMethod]  
 public void CheckFileStatus\_givenWriteFile\_FsGood()  
 {  
 fileWriter.FilePath = "D:\\Уроки\\Уник\\ОАиП\\Лабы\\OnlyWriteFile.txt";  
 Assert.AreEqual(fileWriter.FileStatus,FileStatus.FsGood);  
 }  
}  
  
[TestClass]  
[TestSubject(typeof(Merger))]  
public class MergeTowListsTest  
{  
 private static Merger merger = new Merger();   
 [TestMethod]  
 public void MergeTwoLists\_givenTwoEqualsLists\_SortedArray()  
 {  
 List<int> testList1 = new List<int>(){1,1,2,3};  
 List<int> testList2 = new List<int>(){1,1,2,3};  
 List<int> answer = new List<int>(){1,1,1,1,2,2,3,3};  
 CollectionAssert.AreEquivalent(merger.MergeTwoLists(testList1,testList2),answer);  
 }  
   
 [TestMethod]  
 public void MergeTwoLists\_givenTwoListsOfUnsignedNumbers\_SortedArray()  
 {  
 List<int> testList1 = new List<int>(){1,5,6,7};  
 List<int> testList2 = new List<int>(){2,4,9,15};  
 List<int> answer = new List<int>(){1,2,4,5,6,7,9,15};  
 CollectionAssert.AreEquivalent(merger.MergeTwoLists(testList1,testList2),answer);  
 }   
   
 [TestMethod]  
 public void MergeTwoLists\_givenTwoListsOfAllNumbers\_SortedArray()  
 {  
 List<int> testList1 = new List<int>(){-1,1,7,12512};  
 List<int> testList2 = new List<int>(){-5,0,101,3376};  
 List<int> answer = new List<int>(){-5,-1,0,1,7,101,12512,3376};  
 CollectionAssert.AreEquivalent(merger.MergeTwoLists(testList1,testList2),answer);  
 }   
   
 [TestMethod]  
 public void MergeTwoLists\_givenTwoListsOfUnsignedNumbersWithDuplicates\_SortedArray()  
 {  
 List<int> testList1 = new List<int>(){1,5,6,7,7};  
 List<int> testList2 = new List<int>(){2,5,9,15};  
 List<int> answer = new List<int>(){1,2,5,5,6,7,7,9,15};  
 CollectionAssert.AreEquivalent(merger.MergeTwoLists(testList1,testList2),answer);  
 }  
  
 [TestMethod]  
 public void MergeTwoLists\_givenTwoListsOfAllNumbersWithDuplicates\_SortedArray()  
 {  
 List<int> testList1 = new List<int>() { -1, 1, 7, 101, 12512 };  
 List<int> testList2 = new List<int>() { -5, 1, 0, 101, 3376 };  
 List<int> answer = new List<int>() { -5, -1, 0, 1, 1, 7, 101, 101, 12512, 3376 };  
 CollectionAssert.AreEquivalent(merger.MergeTwoLists(testList1, testList2), answer);  
 }  
}  
  
[TestClass]  
[TestSubject(typeof(Merger))]  
public class IsListDecreasing  
{  
 private static Merger merger = new Merger();   
 [TestMethod]  
 public void isListDecreasing\_givenIncreasingListsOfAllNumbers\_False()  
 {  
 Assert.IsFalse(merger.IsListDecreasing(new List<int>() { -1, 1, 7, 101, 12512 }));  
 }  
 [TestMethod]  
 public void isListDecreasing\_givenDecreasingListsOfAllNumbers\_True()  
 {  
 Assert.IsTrue(merger.IsListDecreasing(new List<int>() { -1, 1, -5, 101, 12512 }));  
 }  
 [TestMethod]  
 public void isListDecreasing\_givenDecreasingListsOfAllNumbersWithDuplicates\_True()  
 {  
 Assert.IsTrue(merger.IsListDecreasing(new List<int>() { -1, 1 ,1,-5, -5, 101, 12512 }));  
 }  
 [TestMethod]  
 public void isListDecreasing\_givenIncreasingListsOfAllNumbersWithDuplicates\_True()  
 {  
 Assert.IsFalse(merger.IsListDecreasing(new List<int>() { -1, 1, 7, 7, 101, 12512 }));  
 }  
}

**Скриншоты:**

