УО «Белорусский государственный университет информатики и радиоэлектроники»

Кафедра ПОИТ

Отчет по лабораторной работе №5.1

по предмету «Основы алгоритмизации и программирования»

Вариант 12

Выполнил:

Галуха П. А.

Гр. 351005

Проверил:

Данилова Г. В.

Минск 2024

**Задание:**

**Код программы Delphi:**

Unit MainUnit;

Interface

Uses

Winapi.Windows, Winapi.Messages, System.SysUtils, System.Variants, System.Classes, Vcl.Graphics,

Vcl.Controls, Vcl.Forms, Vcl.Dialogs, Vcl.Grids, Vcl.StdCtrls, Vcl.Menus, Vcl.ExtDlgs,

InstructionUnit, DeveloperUnit, AddUnit;

Type

TMainForm = Class(TForm)

TabsMainMenu: TMainMenu;

FileMenuItem: TMenuItem;

OpenMenuItem: TMenuItem;

SaveMenuItem: TMenuItem;

SeparatorMenuItem: TMenuItem;

ExitMenuItem: TMenuItem;

InstructionMenuItem: TMenuItem;

DeveloperMenuItem: TMenuItem;

SaveDialog: TSaveTextFileDialog;

OpenDialog: TOpenTextFileDialog;

AddButton: TButton;

DeleteButton: TButton;

ReverseButton: TButton;

NumberStringGrid: TStringGrid;

Procedure MainFormCreate(Sender: TObject);

Function MainFormHelp(Command: Word; Data: NativeInt; Var CallHelp: Boolean): Boolean;

Procedure InstructionMenuItemClick(Sender: TObject);

Procedure DeveloperMenuItemClick(Sender: TObject);

Procedure OpenMenuItemClick(Sender: TObject);

Procedure AddButtonClick(Sender: TObject);

Procedure DeleteButtonClick(Sender: TObject);

Procedure ReverseButtonClick(Sender: TObject);

Procedure SaveMenuItemClick(Sender: TObject);

Procedure ExitMenuItemClick(Sender: TObject);

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

Private

{ Private declarations }

Public

{ Public declarations }

End;

Type

ERRORS\_CODE = (CORRECT,

INCORRECT\_NUMBER,

EXTRA\_DATA);

TLimitedString = String[NUMBER\_LENGTH];

Const

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

'Некорректный номер в файле!',

'Лишние данные!');

MAX\_AMOUNT\_NUMBERS = 100;

NUMBER\_LENGTH = 13;

Var

MainForm: TMainForm;

IsEdited: Boolean = False;

IsSaved: Boolean = True;

Implementation

{$R \*.dfm}

Procedure Make(); Stdcall; External 'LinkedList.dll';

Procedure Add(Data: TLimitedString); Stdcall; External 'LinkedList.dll';

Procedure Remove(Number: Integer); Stdcall; External 'LinkedList.dll';

Procedure Reverse(); Stdcall; External 'LinkedList.dll';

Procedure Clear(); Stdcall; External 'LinkedList.dll';

Procedure WriteLinkedList(StringGrid: TStringGrid); External 'LinkedList.dll';

Procedure TMainForm.MainFormCreate(Sender: TObject);

Begin

Make();

NumberStringGrid.Cells[0, 0] := 'Список номеров';

End;

Function TMainForm.MainFormHelp(Command: Word; Data: NativeInt; Var CallHelp: Boolean): Boolean;

Begin

CallHelp := False;

InstructionMenuItemClick(InstructionMenuItem);

MainFormHelp := False;

End;

Procedure TMainForm.InstructionMenuItemClick(Sender: TObject);

Begin

InstructionForm := TInstructionForm.Create(Self);

InstructionForm.Icon := MainForm.Icon;

InstructionForm.ShowModal;

InstructionForm.Free;

End;

Procedure TMainForm.DeveloperMenuItemClick(Sender: TObject);

Begin

DeveloperForm := TDeveloperForm.Create(Self);

DeveloperForm.Icon := MainForm.Icon;

DeveloperForm.ShowModal;

DeveloperForm.Free;

End;

Procedure DrawStringGrid(StringGrid: TStringGrid);

Begin

StringGrid.ScrollBars := ssNone;

StringGrid.Width := 450; //StringGrid.DefaultColWidth + 2 \* StringGrid.GridLineWidth

If StringGrid.RowCount > 8 Then

StringGrid.Height := 252 //(StringGrid.DefaultRowHeight + StringGrid.GridLineWidth) \* 8 +

StringGrid.GridLineWidth

Else

StringGrid.Height := (StringGrid.DefaultRowHeight + StringGrid.GridLineWidth) \*

StringGrid.RowCount + 5;

MainForm.SaveMenuItem.Enabled := StringGrid.RowCount > 1;

IsSaved := StringGrid.RowCount = 1;

End;

Procedure ClearStringGrid(StringGrid: TStringGrid);

Var

Row: Integer;

Begin

For Row := 1 To StringGrid.RowCount - 1 Do

StringGrid.Cells[0, Row] := '';

StringGrid.RowCount := 1;

End;

Function ReadFileNumber(Var InputFile: TextFile; NumberStringGrid: TStringGrid) : ERRORS\_CODE;

Var

Error: ERRORS\_CODE;

Temp: String;

Num: Integer;

Begin

Error := CORRECT;

ReadLn(InputFile, Temp);

If (Copy(Temp, 1, 4) = '+375') And (Length(Temp) = NUMBER\_LENGTH) And TryStrToInt(Copy(String(Temp),

5), Num) And CharInSet(Temp[5],['0'..'9']) Then

Begin

Add(TLimitedString(Temp));

NumberStringGrid.RowCount := NumberStringGrid.RowCount + 1;

End

Else

Error := INCORRECT\_NUMBER;

ReadFileNumber := Error;

End;

Function ReadFileData(Var InputFile: TextFile; NumberStringGrid: TStringGrid) : ERRORS\_CODE;

Var

Error: ERRORS\_CODE;

I: Integer;

Begin

Reset(InputFile);

Error := CORRECT;

I := 1;

While (Error = CORRECT) And Not seekEOF(InputFile) And (I <= MAX\_AMOUNT\_NUMBERS) Do

Begin

Error := ReadFileNumber(InputFile, NumberStringGrid);

Inc(I);

End;

If (Error = CORRECT) And Not seekEOF(InputFile) Then

Error := EXTRA\_DATA;

CloseFile(InputFile);

ReadFileData := Error;

End;

Procedure TMainForm.OpenMenuItemClick(Sender: TObject);

Var

InputFile: TextFile;

Error: ERRORS\_CODE;

Begin

If OpenDialog.Execute Then

Begin

AssignFile(InputFile, OpenDialog.FileName);

Clear();

ClearStringGrid(NumberStringGrid);

Error := ReadFileData(InputFile, NumberStringGrid);

If Error = CORRECT Then

Begin

WriteLinkedList(NumberStringGrid);

DrawStringGrid(NumberStringGrid);

End

Else

Begin

Clear();

ClearStringGrid(NumberStringGrid);

DrawStringGrid(NumberStringGrid);

Application.MessageBox(PWideChar(ERRORS[Error]), 'Ошибка', MB\_OK + MB\_ICONERROR);

End;

End;

End;

Procedure TMainForm.AddButtonClick(Sender: TObject);

Begin

If NumberStringGrid.RowCount <= MAX\_AMOUNT\_NUMBERS Then

Begin

AddForm := TAddForm.Create(Self);

AddForm.Icon := MainForm.Icon;

AddForm.ShowModal;

AddForm.Free;

If IsEdited Then

Begin

DrawStringGrid(NumberStringGrid);

IsEdited := False;

End;

End

Else

Application.MessageBox('Слишком много номеров!', 'Ошибка', MB\_OK + MB\_ICONERROR);

End;

Procedure TMainForm.DeleteButtonClick(Sender: TObject);

Var

Confirmation: Integer;

Begin

If NumberStringGrid.Row > 0 Then

Begin

Confirmation := Application.MessageBox('Вы действительно хотите удалить телефон?', 'Удаление',

MB\_YESNO + MB\_ICONQUESTION + MB\_DEFBUTTON2);

If Confirmation = IDYES Then

Begin

Remove(NumberStringGrid.Row);

NumberStringGrid.Cells[0, NumberStringGrid.RowCount - 1] := '';

NumberStringGrid.RowCount := NumberStringGrid.RowCount - 1;

WriteLinkedList(NumberStringGrid);

DrawStringGrid(NumberStringGrid);

End;

End

Else

Application.MessageBox('Не выбрано редактируемое поле!', 'Ошибка', MB\_OK + MB\_ICONERROR);

End;

Procedure TMainForm.ReverseButtonClick(Sender: TObject);

Begin

If NumberStringGrid.RowCount > 1 Then

Begin

Reverse();

WriteLinkedList(NumberStringGrid);

MainForm.SaveMenuItem.Enabled := True;

IsSaved := False;

End

Else

Application.MessageBox('Не добавлено номеров!', 'Ошибка', MB\_OK + MB\_ICONERROR);

End;

Procedure WriteFileData(Var OutputFile: TextFile; NumberStringGrid: TStringGrid);

Var

I: Integer;

Begin

ReWrite(OutputFile);

For I := 1 To NumberStringGrid.RowCount - 1 Do

WriteLn(OutputFile, NumberStringGrid.Cells[0, I]);

CloseFile(OutputFile);

End;

Procedure TMainForm.SaveMenuItemClick(Sender: TObject);

Var

OutputFile: TextFile;

Begin

If SaveDialog.Execute Then

Begin

AssignFile(OutputFile, SaveDialog.FileName);

WriteFileData(OutputFile, NumberStringGrid);

IsSaved := True;

End;

End;

Procedure TMainForm.ExitMenuItemClick(Sender: TObject);

Begin

Close;

End;

Procedure TMainForm.MainFormCloseQuery(Sender: TObject; Var CanClose: Boolean);

Var

Confirmation: Integer;

Begin

If IsSaved Then

Begin

Confirmation := Application.MessageBox('Вы действительно хотите выйти?', 'Выход', MB\_YESNO + MB\_ICONQUESTION + MB\_DEFBUTTON2);

CanClose := Confirmation = IDYES;

End

Else

Begin

Confirmation := Application.MessageBox('Вы не сохранили файл, хотите ли сохранить?', 'Выход',

MB\_YESNOCANCEl + MB\_ICONQUESTION + MB\_DEFBUTTON2);

Case Confirmation Of

mrYes:

Begin

SaveMenuItemClick(Sender);

If IsSaved Then

CanClose := True

Else

MainFormCloseQuery(Sender, CanClose);

End;

mrNo:

CanClose := True;

mrCancel:

CanClose := False;

End;

End;

End;

End.

Unit AddUnit;

Interface

Uses

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

System.Classes, Vcl.Graphics, Vcl.Grids,

Vcl.Controls, Vcl.Forms, Vcl.Dialogs, Vcl.StdCtrls, Clipbrd;

Type

TAddForm = Class(TForm)

NumberLabel: TLabel;

NumberEdit: TEdit;

AddButton: TButton;

CancelButton: TButton;

Procedure AddFormKeyDown(Sender: TObject; Var Key: Word; Shift: TShiftState);

Procedure ComponentKeyDown(Sender: TObject; Var Key: Word; Shift: TShiftState);

Procedure NumberEditChange(Sender: TObject);

Procedure NumberEditContextPopup(Sender: TObject; MousePos: TPoint; Var Handled: Boolean);

Procedure NumberEditKeyDown(Sender: TObject; Var Key: Word; Shift: TShiftState);

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

Procedure NumberEditKeyUp(Sender: TObject; Var Key: Word; Shift: TShiftState);

Procedure AddButtonClick(Sender: TObject);

Procedure CloseButtonClick(Sender: TObject);

Private

{ Private declarations }

Public

{ Public declarations }

End;

Const

ENTER = #13;

BACKSPACE = #8;

NONE = #0;

DIGITS = ['0'..'9'];

ALPHABET = ['A'..'Z', 'a'..'z'];

NUMBER\_LENGTH = 13;

Type

TLimitedString = String[NUMBER\_LENGTH];

Var

AddForm: TAddForm;

CtrlPressed: Boolean = False;

Implementation

{$R \*.dfm}

Uses MainUnit;

Procedure Add(Data: TLimitedString); Stdcall; External 'LinkedList.dll';

Procedure WriteLinkedList(StringGrid: TStringGrid); Stdcall; External 'LinkedList.dll';

Procedure TAddForm.AddFormKeyDown(Sender: TObject; Var Key: Word; Shift: TShiftState);

Begin

If Key = VK\_ESCAPE Then

Close;

End;

Procedure TAddForm.ComponentKeyDown(Sender: TObject; Var Key: Word; Shift: TShiftState);

Begin

If Key = VK\_UP Then

SelectNext(TWinControl(Sender), False, True)

Else If Key = VK\_DOWN Then

SelectNext(TWinControl(Sender), True, True)

Else If (Key = Ord(ENTER)) And AddButton.Enabled Then

AddButtonClick(AddButton);

End;

Procedure TAddForm.NumberEditChange(Sender: TObject);

Begin

AddButton.Enabled := Length(NumberEdit.Text) = NumberEdit.MaxLength;

End;

Function IsPossiblePaste(SelStart, SelLength: Integer; Text: String) : Boolean;

Var

Num: Integer;

Begin

IsPossiblePaste := Clipboard.HasFormat(CF\_TEXT) And (Length(ClipBoard.AsText) <> 0) And

TryStrToInt(Copy(Text, 1, SelStart) + ClipBoard.AsText + Copy(Text, SelStart +

SelLength + 1), Num) And

(ClipBoard.AsText[1] <> '-');

End;

Procedure TAddForm.NumberEditContextPopup(Sender: TObject; MousePos: TPoint; Var Handled: Boolean);

Begin

Handled := Not IsPossiblePaste(NumberEdit.SelStart, NumberEdit.SelLength, NumberEdit.Text);

End;

Procedure TAddForm.NumberEditKeyDown(Sender: TObject; Var Key: Word; Shift: TShiftState);

Begin

ComponentKeyDown(NumberEdit, Key, Shift);

If (Shift = [ssCtrl]) And (UpCase(Chr(Key)) = 'V') Or (Shift = [ssShift]) And (Key = VK\_INSERT) Then

Begin

If Not IsPossiblePaste(NumberEdit.SelStart, NumberEdit.SelLength, NumberEdit.Text) Then

Key := Ord(NONE);

End;

If (Shift = [ssCtrl]) And CharInSet(Chr(Key), ALPHABET) Then

CtrlPressed := True;

End;

Procedure TAddForm.NumberEditKeyPress(Sender: TObject; Var Key: Char);

Begin

If Not (CtrlPressed Or CharInSet(Key, DIGITS) Or (Key = BACKSPACE)) Then

Key := NONE;

End;

Procedure TAddForm.NumberEditKeyUp(Sender: TObject; Var Key: Word; Shift: TShiftState);

Begin

CtrlPressed := False;

End;

Procedure TAddForm.AddButtonClick(Sender: TObject);

Begin

Add(TLimitedString('+375' + NumberEdit.Text));

MainForm.NumberStringGrid.RowCount := MainForm.NumberStringGrid.RowCount + 1;

WriteLinkedList(MainForm.NumberStringGrid);

IsEdited := True;

Close;

End;

Procedure TAddForm.CloseButtonClick(Sender: TObject);

Begin

Close;

End;

End.

Unit InstructionUnit;

Interface

Uses

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

System.Classes, Vcl.Graphics,

Vcl.Controls, Vcl.Forms, Vcl.Dialogs, Vcl.StdCtrls;

Type

TInstructionForm = Class(TForm)

InstructionLabel: TLabel;

Procedure InstructionFormCreate(Sender: TObject);

Procedure InstructionFormKeyDown(Sender: TObject; Var Key: Word; Shift: TShiftState);

Private

{ Private declarations }

Public

{ Public declarations }

End;

Var

InstructionForm: TInstructionForm;

Implementation

{$R \*.dfm}

Procedure TInstructionForm.InstructionFormCreate(Sender: TObject);

Begin

InstructionLabel.Width := 800;

InstructionLabel.Caption := '1. Для добавления элемента к списку нажмите кнопку "Добавить".'#13#10 +

'2. Для удаления элемента списка нажмите кнопку "Удалить".'#13#10 +

'3. Для переворота списка нажмите кнопку "Перевернуть".'#13#10 +

'4. Файл содержит номера начиная с +375.';

InstructionLabel.Left := (ClientWidth - InstructionLabel.Width) Div 2;

InstructionLabel.Top := (ClientHeight - InstructionLabel.Height) Div 2;

End;

Procedure TInstructionForm.InstructionFormKeyDown(Sender: TObject; Var Key: Word; Shift: TShiftState);

Begin

If Key = VK\_ESCAPE Then

Close;

End;

End.

Unit DeveloperUnit;

Interface

Uses

Winapi.Windows, Winapi.Messages, System.SysUtils, System.Variants, System.Classes, Vcl.Graphics,

Vcl.Controls, Vcl.Forms, Vcl.Dialogs, Vcl.StdCtrls;

Type

TDeveloperForm = Class(TForm)

DeveloperLabel: TLabel;

Procedure DeveloperFormCreate(Sender: TObject);

Procedure DeveloperFormKeyDown(Sender: TObject; Var Key: Word; Shift: TShiftState);

Private

{ Private declarations }

Public

{ Public declarations }

End;

Var

DeveloperForm: TDeveloperForm;

Implementation

{$R \*.dfm}

Procedure TDeveloperForm.DeveloperFormCreate(Sender: TObject);

Begin

DeveloperLabel.Caption := 'Группа: 351005'#13#10 +

'Разработчик: Галуха Павел Александрович'#13#10 +

'Телеграмм: @pavello06';

DeveloperLabel.Left := (ClientWidth - DeveloperLabel.Width) Div 2;

DeveloperLabel.Top := (ClientHeight - DeveloperLabel.Height) Div 2;

End;

Procedure TDeveloperForm.DeveloperFormKeyDown(Sender: TObject; Var Key: Word; Shift: TShiftState);

Begin

If Key = VK\_ESCAPE Then

Close;

End;

End.

**Библиотечный модуль:**

Library LinkedList;

{$R \*.res}

uses

Vcl.Grids;

Const

MAX\_LENGTH = 13;

Type

TLimitedString = String[MAX\_LENGTH];

TLinkedList = ^TNode;

TNode = Record

Data: TLimitedString;

Next: TLinkedList;

End;

Var

List, Head: TLinkedList;

Procedure Make();

Begin

New(Head);

Head.Next := Nil;

End;

Procedure Add(Data: TLimitedString);

Begin

List := Head;

While List.Next <> Nil Do

List := List.Next;

New(List.Next);

List := List.Next;

List.Data := Data;

List.Next := Nil;

End;

Procedure Remove(Number: Integer);

Var

I: Integer;

Rem: TLinkedList;

Begin

List := Head;

For I := 1 To Number - 1 Do

List := List.Next;

Rem := List.Next;

List.Next := List.Next.Next;

Dispose(Rem);

End;

Procedure Reverse();

Var

PreviousNode, CurrentNode, NextNode: TLinkedList;

Begin

PreviousNode := Nil;

CurrentNode := Head.Next;

While CurrentNode <> Nil Do

Begin

NextNode := CurrentNode.Next;

CurrentNode.Next := PreviousNode;

PreviousNode := CurrentNode;

CurrentNode := NextNode;

End;

Head.Next := PreviousNode;

End;

Procedure Clear();

Var

Rem: TLinkedList;

Begin

Rem := Head.Next;

While Rem <> Nil Do

Begin

List := Rem.Next;

Dispose(Rem);

Rem := List;

End;

Head.Next := Nil;

End;

Procedure WriteLinkedList(StringGrid: TStringGrid);

Var

I: Integer;

Begin

List := Head.Next;

I := 1;

While List <> Nil Do

Begin

StringGrid.Cells[0, I] := String(List.Data);

List := List.Next;

Inc(I);

End;

End;

Exports

Make, Add, Remove, Reverse, Clear, WriteLinkedList;

Begin

End.

**Внешний файл:**

Unit LinkedList;

Interface

Implementation

{$R \*.res}

Uses

Vcl.Grids;

Const

MAX\_LENGTH = 13;

Type

TLimitedString = String[MAX\_LENGTH];

TLinkedList = ^TNode;

TNode = Record

Data: TLimitedString;

Next: TLinkedList;

End;

Var

List, Head: TLinkedList;

Procedure Make();

Begin

New(Head);

Head.Next := Nil;

End;

Procedure Add(Data: TLimitedString);

Begin

List := Head;

While List.Next <> Nil Do

List := List.Next;

New(List.Next);

List := List.Next;

List.Data := Data;

List.Next := Nil;

End;

Procedure Remove(Number: Integer);

Var

I: Integer;

Rem: TLinkedList;

Begin

List := Head;

For I := 1 To Number - 1 Do

List := List.Next;

Rem := List.Next;

List.Next := List.Next.Next;

Dispose(Rem);

End;

Procedure Reverse();

Var

PreviousNode, CurrentNode, NextNode: TLinkedList;

Begin

PreviousNode := Nil;

CurrentNode := Head.Next;

While CurrentNode <> Nil Do

Begin

NextNode := CurrentNode.Next;

CurrentNode.Next := PreviousNode;

PreviousNode := CurrentNode;

CurrentNode := NextNode;

End;

Head.Next := PreviousNode;

End;

Procedure Clear();

Var

Rem: TLinkedList;

Begin

Rem := Head.Next;

While Rem <> Nil Do

Begin

List := Rem.Next;

Dispose(Rem);

Rem := List;

End;

Head.Next := Nil;

End;

Procedure WriteLinkedList(StringGrid: TStringGrid);

Var

I: Integer;

Begin

List := Head.Next;

I := 1;

While List <> Nil Do

Begin

StringGrid.Cells[0, I] := String(List.Data);

List := List.Next;

Inc(I);

End;

End;

Exports Make, Add, Remove, Reverse, Clear, WriteLinkedList;

Begin

End.

**Код программы C#:**

using System;

using System.Collections;

using System.Collections.Generic;

using System.IO;

using System.Runtime.Remoting.Messaging;

namespace Lab51

{

public class Node<T>

{

public T Data { get; set; }

public Node<T> Next { get; set; }

}

public class LinkedList<T>

{

private Node<T> head = new Node<T>()

{

Data = default(T),

Next = null

};

int count = 1;

public void Add(T data)

{

Node<T> newNode = new Node<T>()

{

Data = data,

Next = null

};

Node<T> current = head;

while (current.Next != null)

current = current.Next;

current.Next = newNode;

count++;

}

public void Remove(int number)

{

Node<T> current = head;

for (int i = 1; i < number; i++)

current = current.Next;

current.Next = current.Next.Next;

count--;

}

public void Reverse()

{

Node<T> current = head.Next, previos = null, follow;

while (current != null)

{

follow = current.Next;

current.Next = previos;

previos = current;

current = follow;

}

head.Next = previos;

}

public int Count()

{

return count;

}

public void Clear()

{

head = null;

count = 0;

}

public T Meaning(int number)

{

Node<T> current = head;

for (int i = 1; i <= number; i++)

current = current.Next;

return current.Data;

}

public void WriteLinkedList(StreamWriter writer)

{

Node<T> current = head;

while (current != null)

{

writer.WriteLine(current.Data);

current = current.Next;

}

}

}

internal class Program

{

public enum ERRORS\_CODE

{

CORRECT,

INCORRECT\_CHOICE,

INCORRECT\_LASTNAME,

IS\_NOT\_TXT,

IS\_NOT\_EXIST,

IS\_NOT\_READABLE,

IS\_NOT\_WRITEABLE,

EXTRA\_DATA

}

static readonly string[] ERRORS = new string[]

{

"",

"Некорректный выбор!",

"Некорректная фамилия!",

"Файл не .txt!",

"Файл не существует!",

"Файл не доступен для чтения!",

"Файл не доступен для записи",

"Лишние данные!",

};

public enum Actions

{

Add = 1,

Delete,

Reverse,

Open,

Save,

Exit,

}

const int MAX\_LASTNAME\_LENGTH = 20,

MIN\_LASTNAME\_LENGTH = 1,

MAX\_LASTNAMES = 99;

static void WriteTask()

{

Console.WriteLine("Выберите одно из следующих действий:");

Console.WriteLine("Добавить фамилию - 1");

Console.WriteLine("Удалить фамилию - 2");

Console.WriteLine("Перевернуть список - 3");

Console.WriteLine("Открыть файл - 4");

Console.WriteLine("Сохранить файл - 5");

Console.WriteLine("Выйти - 6");

Console.Write("Ваш выбор: ");

}

static void WriteContinue()

{

Console.Write("Нажмите любую клавишу для продолжения: ");

Console.ReadKey();

Console.WriteLine();

}

static void WriteError(ERRORS\_CODE error)

{

Console.Error.WriteLine(ERRORS[(int)error]);

Console.Write("Попробуйте снова: ");

}

static int ChooseOptionWithinRange(int borderBottom, int borderTop)

{

ERRORS\_CODE error;

int option = 1;

do

{

error = ERRORS\_CODE.CORRECT;

try

{

option = int.Parse(Console.ReadLine());

}

catch

{

error = ERRORS\_CODE.INCORRECT\_CHOICE;

}

if ((error == ERRORS\_CODE.CORRECT) && ((option < borderBottom) || (option > borderTop)))

error = ERRORS\_CODE.INCORRECT\_CHOICE;

if (error != ERRORS\_CODE.CORRECT)

WriteError(error);

} while (error != ERRORS\_CODE.CORRECT);

return option;

}

static void WriteTable(LinkedList<string> numberList)

{

for (int j = 1; j < 30; j++) Console.Write("-");

Console.WriteLine("\r\n| {0, 2} | {1, 20} |", "#", "Фамилия");

for (int i = 1; i < numberList.Count(); i++)

{

for (int j = 1; j < 30; j++) Console.Write("-");

Console.WriteLine("\r\n| {0, 2} | {1, 20} |", i, numberList.Meaning(i));

}

for (int j = 1; j < 30; j++) Console.Write("-");

Console.WriteLine();

}

static void AddLastName(LinkedList<string> numberList, ref bool isSaved)

{

ERRORS\_CODE error;

string temp;

if (numberList.Count() <= MAX\_LASTNAMES)

{

Console.Write($"Введите фамилию (макс {MAX\_LASTNAME\_LENGTH} символов): ");

do

{

error = ERRORS\_CODE.CORRECT;

temp = Console.ReadLine();

if (temp.Length < MIN\_LASTNAME\_LENGTH || temp.Length > MAX\_LASTNAME\_LENGTH)

error = ERRORS\_CODE.INCORRECT\_LASTNAME;

if (error != ERRORS\_CODE.CORRECT)

WriteError(error);

} while (error != ERRORS\_CODE.CORRECT);

numberList.Add(temp);

isSaved = false;

}

else

Console.WriteLine("Слишком большой список!");

}

static void DeleteLastName(LinkedList<string> numberList, ref bool isSaved)

{

if (numberList.Count() > 1)

{

Console.Write("Введите номер фамилии в списке: ");

numberList.Remove(ChooseOptionWithinRange(1, numberList.Count() - 1));

if (numberList.Count() > 1)

isSaved = false;

}

else

Console.WriteLine("Вы ещё не добавили фамилий!");

}

static void ReverseLastNames(LinkedList<string> numberList, ref bool isSaved)

{

if (numberList.Count() > 1)

{

numberList.Reverse();

isSaved = false;

}

else

Console.WriteLine("Вы ещё не добавили фамилий!");

}

static ERRORS\_CODE IsReadable(string filePath)

{

ERRORS\_CODE error = ERRORS\_CODE.CORRECT;

try

{

using (StreamReader reader = new StreamReader(filePath)) { }

}

catch

{

error = ERRORS\_CODE.IS\_NOT\_READABLE;

}

return error;

}

static ERRORS\_CODE IsWriteable(string filePath)

{

ERRORS\_CODE error = ERRORS\_CODE.CORRECT;

try

{

using (StreamWriter writer = new StreamWriter(filePath)) { }

}

catch

{

error = ERRORS\_CODE.IS\_NOT\_WRITEABLE;

}

return error;

}

static string ReadIOFilePath(char ioMode)

{

ERRORS\_CODE error;

string filePath;

do

{

error = ERRORS\_CODE.CORRECT;

Console.Write("Введите путь к файлу: ");

filePath = Console.ReadLine();

if (Path.GetExtension(filePath) != ".txt")

error = ERRORS\_CODE.IS\_NOT\_TXT;

if (error == ERRORS\_CODE.CORRECT && !File.Exists(filePath) && ioMode == 'i')

error = ERRORS\_CODE.IS\_NOT\_EXIST;

if (error == ERRORS\_CODE.CORRECT)

switch (ioMode)

{

case 'i':

error = IsReadable(filePath);

break;

case 'o':

error = IsWriteable(filePath);

break;

}

if (error != ERRORS\_CODE.CORRECT)

WriteError(error);

} while (error != ERRORS\_CODE.CORRECT);

return filePath;

}

static void ReadFile(LinkedList<string> numberList)

{

string filePath = ReadIOFilePath('i');

StreamReader reader = new StreamReader(filePath);

ERRORS\_CODE error = ERRORS\_CODE.CORRECT;

string temp;

using (reader)

{

do

{

temp = reader.ReadLine();

if (temp.Length < MIN\_LASTNAME\_LENGTH || temp.Length > MAX\_LASTNAME\_LENGTH)

error = ERRORS\_CODE.INCORRECT\_LASTNAME;

if (error == ERRORS\_CODE.CORRECT && numberList.Count() > MAX\_LASTNAMES)

error = ERRORS\_CODE.EXTRA\_DATA;

if (error == ERRORS\_CODE.CORRECT)

numberList.Add(temp);

else

WriteError(error);

} while (error != ERRORS\_CODE.CORRECT || !reader.EndOfStream);

}

if (error != ERRORS\_CODE.CORRECT)

numberList.Clear();

}

static void WriteFile(LinkedList<string> numberList, ref bool isSaved)

{

string filePath = ReadIOFilePath('o');

StreamWriter writer = new StreamWriter(filePath);

if (numberList.Count() > 1)

{

using (writer)

{

for (int i = 1; i < numberList.Count(); i++)

{

writer.WriteLine(numberList.Meaning(i));

}

}

isSaved = true;

}

else

Console.WriteLine("Вы ещё не добавили фамилий!");

}

static void Exit(LinkedList<string> numberList, bool isSaved)

{

if (!isSaved)

{

Console.Write("Вы не сохранили файл, хотите ли сохранить перед выходом?\r\n" +

"да - 1\r\n" +

"нет - 2\r\n" +

"Ваш выбор: ");

if (ChooseOptionWithinRange(1, 2) == 1)

WriteFile(numberList, ref isSaved);

}

}

static void Main(string[] args)

{

LinkedList<string> numberList = new LinkedList<string>();

Actions action;

bool isSaved = true;

do

{

WriteTable(numberList);

WriteTask();

action = (Actions)ChooseOptionWithinRange(1, Enum.GetValues(typeof(Actions)).Length);

switch (action)

{

case Actions.Add:

AddLastName(numberList, ref isSaved);

break;

case Actions.Delete:

DeleteLastName(numberList, ref isSaved);

break;

case Actions.Reverse:

ReverseLastNames(numberList, ref isSaved);

break;

case Actions.Open:

ReadFile(numberList);

break;

case Actions.Save:

WriteFile(numberList, ref isSaved);

break;

case Actions.Exit:

Exit(numberList, isSaved);

break;

}

if (action != Actions.Exit)

WriteContinue();

} while (action != Actions.Exit);

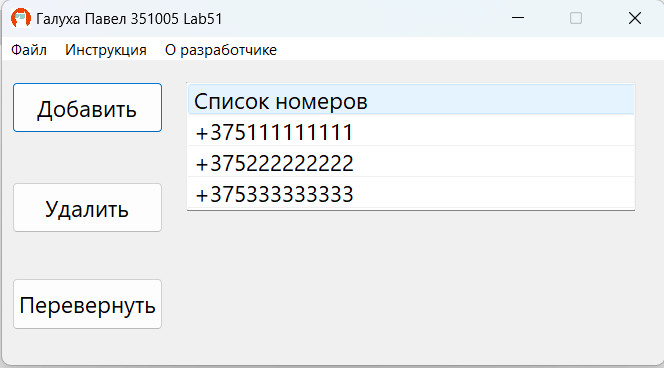
}

}

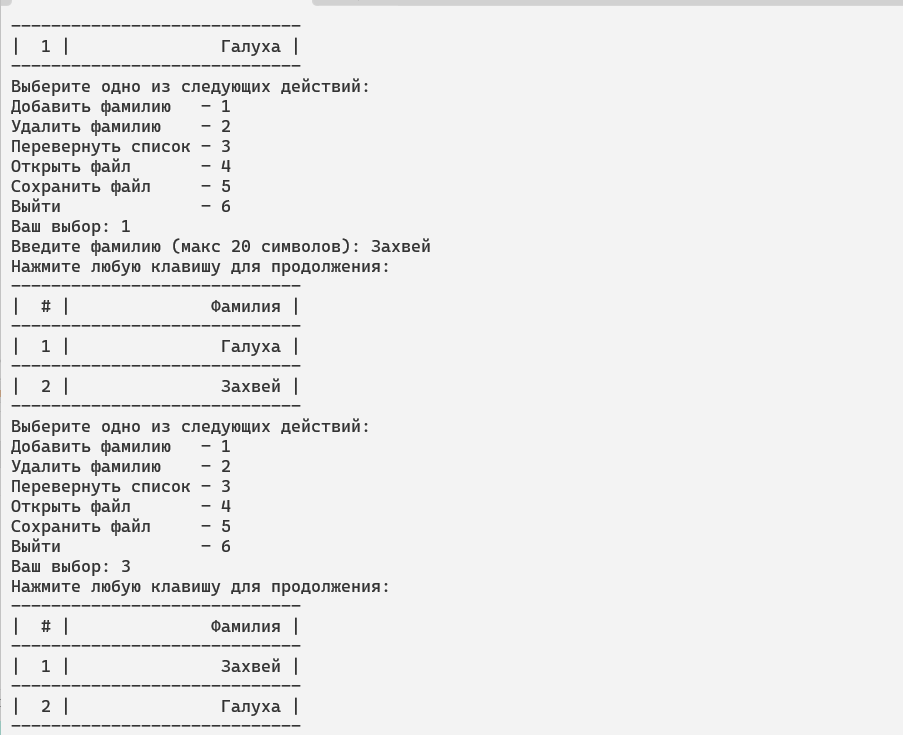
}

**Скриншоты**

**Delphi:**

****

**C#:**



**Блок-схема**

