УО «Белорусский государственный университет информатики и радиоэлектроники»

Кафедра ПОИТ

Отчет по лабораторной работе №5.1

по предмету

Основы алгоритмизации и программирования

Вариант 3

Выполнил:

Бетеня К.С.

Проверила:

Данилова Г.В.

Группа 351005

Минск 2024

Задание:

Списки. Разработать программу работы с двусвязным списком. Программа должна содержать следующие процедуры, вызываемые из меню:

* построение пустого списка;
* добавление нового элемента;
* удаление указанного элемента;
* просмотр списка в обратном направлении.

Код программы на **Delphi**:

**Unit MainFormUnit;**

Interface

Uses

Winapi.Windows,

Winapi.Messages,

System.SysUtils,

System.Variants,

System.Classes,

Vcl.Graphics,

Vcl.Controls,

Vcl.Forms,

Vcl.Dialogs,

Vcl.Buttons,

Vcl.Menus,

Vcl.ExtCtrls,

Vcl.StdCtrls,

System.ImageList,

Vcl.ImgList,

Vcl.Grids;

Type

TMainForm = Class(TForm)

MainMenu: TMainMenu;

FileMM: TMenuItem;

InstractionMM: TMenuItem;

AboutEditorMM: TMenuItem;

Line: TMenuItem;

ExitMM: TMenuItem;

AddButton: TSpeedButton;

DeleteButton: TSpeedButton;

NavPanel: TPanel;

SpButImgList: TImageList;

MMImgList: TImageList;

ExitButton: TSpeedButton;

LinkedListStrGrid: TStringGrid;

OpenButton: TMenuItem;

SaveButton: TMenuItem;

OpenDialog: TOpenDialog;

SaveDialog: TSaveDialog;

Procedure AddButtonClick(Sender: TObject);

Procedure SpeedButton1Click(Sender: TObject);

Procedure AboutEditorMMClick(Sender: TObject);

Procedure FormCreate(Sender: TObject);

Procedure ExitButtonClick(Sender: TObject);

Procedure ExitMMClick(Sender: TObject);

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

Boolean;

Procedure DeleteButtonClick(Sender: TObject);

Procedure InstractionMMClick(Sender: TObject);

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

Procedure LinkedListStrGridKeyDown(Sender: TObject; Var Key: Word; Shift:

TShiftState);

Procedure SaveButtonClick(Sender: TObject);

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

Procedure OpenButtonClick(Sender: TObject);

Private

{ Private declarations }

Public

{ Public declarations }

End;

Var

MainForm: TMainForm;

IfDataSavedInFile: Boolean = False;

Implementation

{$R \*.dfm}

Uses

AddValueUnit,

FrontendUnit;

Procedure InsertInList(Value: Integer); Stdcall; External 'DoubleLinkedList.dll';

Procedure DeleteFromList(Num: Integer); Stdcall; External 'DoubleLinkedList.dll';

Procedure PrintList(Var LinkedListStrGrid: TStringGrid); Stdcall; External

'DoubleLinkedList.dll';

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

Var

ResultKey: Integer;

Begin

ResultKey := Application.Messagebox('Вы уверены, что хотите закрыть оконное

приложение?', 'Выход', MB\_YESNO +

MB\_ICONQUESTION + MB\_DEFBUTTON2);

If ResultKey = ID\_NO Then

CanClose := False;

If (ResultKey = ID\_YES) And Not IfDataSavedInFile And

(LinkedListStrGrid.RowCount > 1) Then

Begin

ResultKey := Application.Messagebox('Вы не сохранили результат. Хотите

сделать это?', 'Сохранение', MB\_YESNO

+ MB\_ICONQUESTION + MB\_DEFBUTTON2);

If ResultKey = ID\_YES Then

SaveButtonClick(Sender);

End;

End;

Procedure TMainForm.DeleteButtonClick(Sender: TObject);

Const

MAX\_SEEN\_RECORDS: Integer = 11;

Var

ResultKey, CurentRow: Integer;

Begin

CurentRow := MainForm.LinkedListStrGrid.Row;

If (CurentRow <> 0) Then

Begin

ResultKey := Application.Messagebox('Вы уверены, что хотите удалить

запись?', 'Удаление записи', MB\_YESNO

+ MB\_ICONQUESTION + MB\_DEFBUTTON2);

If ResultKey = ID\_YES Then

Begin

DeleteFromList(LinkedListStrGrid.RowCount - CurentRow);

ClearStringGrid;

LinkedListStrGrid.RowCount := LinkedListStrGrid.RowCount - 1;

PrintList(LinkedListStrGrid);

If LinkedListStrGrid.RowCount < 17 Then

LinkedListStrGrid.ColWidths[1] := (LinkedListStrGrid.Width \* 79)

Div 100;

If MainForm.LinkedListStrGrid.RowCount > 1 Then

MainForm.SaveButton.Enabled := True

Else

MainForm.SaveButton.Enabled := False;

IfDataSavedInFile := False;

End;

End

Else

MessageBox(0, 'Необходимо выбрать строку для удаления!', 'Ошибка',

MB\_ICONERROR);

End;

**Unit AddValueUnit;**

Interface

Uses

Clipbrd,

Winapi.Windows,

Winapi.Messages,

System.SysUtils,

System.Variants,

System.Classes,

Vcl.Graphics,

Vcl.Controls,

Vcl.Forms,

Vcl.Dialogs,

Vcl.Menus,

Vcl.StdCtrls,

Vcl.Mask,

Vcl.ExtCtrls,

Vcl.Buttons,

Vcl.Grids;

Type

TLabeledEdit = Class(Vcl.ExtCtrls.TLabeledEdit)

Public

Procedure WMPaste(Var Msg: TMessage); Message WM\_PASTE;

End;

TAddValueForm = Class(TForm)

MainMenu: TMainMenu;

Instraction: TMenuItem;

Label1: TLabel;

ValueLabEdit: TLabeledEdit;

ResultButton: TSpeedButton;

Procedure FormCreate(Sender: TObject);

Procedure ResultButtonClick(Sender: TObject);

Procedure InstractionClick(Sender: TObject);

Procedure ValueLabEditChange(Sender: TObject);

Procedure ValueLabEditContextPopup(Sender: TObject; MousePos: TPoint; Var

Handled: Boolean);

Procedure ValueLabEditKeyDown(Sender: TObject; Var Key: Word; Shift:

TShiftState);

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

Private

{ Private declarations }

Public

{ Public declarations }

End;

Const

NULL\_POINT: Char = #0;

DELETE\_KEY: Char = #127;

BACK\_SPACE: Char = #08;

MINUS\_KEY: Char = '-';

Var

AddValueForm: TAddValueForm;

Implementation

{$R \*.dfm}

Uses

MainFormUnit,

FrontendUnit;

Procedure InsertInList(Value: Integer); Stdcall; External 'DoubleLinkedList.dll';

Procedure DeleteFromList(Num: Integer); Stdcall; External 'DoubleLinkedList.dll';

Procedure PrintList(Var LinkedListStrGrid: TStringGrid); Stdcall; External

'DoubleLinkedList.dll';

Procedure TAddValueForm.ResultButtonClick(Sender: TObject);

Begin

MainForm.LinkedListStrGrid.RowCount := MainForm.LinkedListStrGrid.RowCount +

1;

InsertInList(StrToInt(ValueLabEdit.Text));

ClearStringGrid;

PrintList(MainForm.LinkedListStrGrid);

MainForm.LinkedListStrGrid.FixedRows := 1;

If MainForm.LinkedListStrGrid.RowCount > 16 Then

MainForm.LinkedListStrGrid.ColWidths[1] :=

(MainForm.LinkedListStrGrid.Width \* 70) Div 100;

If MainForm.LinkedListStrGrid.RowCount > 1 Then

MainForm.SaveButton.Enabled := True

Else

MainForm.SaveButton.Enabled := False;

IfDataSavedInFile := False;

AddValueForm.Close;

End;

Procedure TLabeledEdit.WMPaste(Var Msg: TMessage);

Begin

If Clipboard.HasFormat(CF\_TEXT) Then

Begin

Try

If (AddValueForm.ActiveControl = AddValueForm.ValueLabEdit) And Not

IsCorrectPointsClipboard(Clipboard.AsText,

AddValueForm.ValueLabEdit) Then

Raise Exception.Create('Некорректная цифра :(');

Except

On E: Exception Do

Begin

MessageBox(0, PWideChar(E.Message), 'Ошибка', MB\_ICONERROR);

Exit;

End;

End;

End;

Inherited;

End;

**Unit FrontendUnit;**

Interface

Uses

Winapi.Windows,

Winapi.Messages,

System.SysUtils,

System.Variants,

System.Classes,

Vcl.Graphics,

Vcl.Controls,

Vcl.Forms,

Vcl.Dialogs,

Vcl.Buttons,

Vcl.Menus,

Vcl.ExtCtrls,

Vcl.StdCtrls,

System.ImageList,

Vcl.ImgList,

Vcl.Grids;

Procedure CreateModalForm(CaptionText, LabelText: String; ModalWidth, ModalHeight:

Integer);

Function IsCorrectPointsClipboard(ClipbrdText: String; NumLabEd: TLabeledEdit):

Boolean;

Procedure ClearStringGrid();

Function IsCorrectDelete(Key: Char; CurentText: String; SelStart: Integer): Char;

Function IsCorrectSelDelete(Key: Char; CurentText, SelText: String; SelStart:

Integer): Char;

Function IsCorrectSelTextInputWithKey(Key: Char; CurentText, SelText: String;

SelStart: Integer): Char;

Function IsCorrectInput(Key: Char; CurentText: String; SelStart: Integer): Char;

Function IsWriteable(FilePath: String): Boolean;

Procedure InputInFile(Var IsCorrect: Boolean; FilePath: String);

Function IsReadable(FilePath: String): Boolean;

Procedure ReadFromFile(Var IsCorrect: Boolean; FilePath: String);

Const

MIN\_POINTS: Integer = -1\_000\_000\_000;

MAX\_POINTS: Integer = +1\_000\_000\_000;

NULL\_POINT: Char = #0;

ZERO\_KEY: Char = '0';

MINUS\_KEY: Char = '-';

Implementation

Uses

MainFormUnit;

Procedure InsertInList(Value: Integer); Stdcall; External 'DoubleLinkedList.dll';

Procedure DeleteFromList(Num: Integer); Stdcall; External 'DoubleLinkedList.dll';

Procedure PrintList(Var LinkedListStrGrid: TStringGrid); Stdcall; External

'DoubleLinkedList.dll';

Procedure CreateModalForm(CaptionText, LabelText: String; ModalWidth, ModalHeight:

Integer);

Const

LEFT\_MARGIN: Integer = 10;

TOP\_MARGIN: Integer = 5;

Var

ModalForm: TForm;

ModalLabel: TLAbel;

Begin

ModalForm := TForm.Create(Nil);

Try

ModalForm.Caption := CaptionText;

ModalForm.Width := ModalWidth;

ModalForm.Height := ModalHeight;

ModalForm.Position := PoScreenCenter;

ModalForm.BorderStyle := BsSingle;

ModalForm.BorderIcons := [BiSystemMenu];

ModalForm.FormStyle := FsStayOnTop;

ModalForm.Icon := MainForm.Icon;

ModalLabel := TLabel.Create(ModalForm);

ModalLabel.Parent := ModalForm;

ModalLabel.Caption := LabelText;

ModalLabel.Left := LEFT\_MARGIN;

ModalLabel.Top := TOP\_MARGIN;

ModalForm.ShowModal;

Finally

ModalForm.Free;

End;

End;

Procedure InputInFile(Var IsCorrect: Boolean; FilePath: String);

Var

MyFile: TextFile;

I: Integer;

Begin

If IsCorrect Then

Begin

AssignFile(MyFile, FilePath, CP\_UTF8);

Try

ReWrite(MyFile);

Try

Writeln(MyFile, 'Список в обратном порядке: ');

Writeln(MyFile, ' \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_');

Writeln(MyFile, '| № | Значение |');

Writeln(MyFile, '|-------|-----------------------------|');

For I := 1 To MainForm.LinkedListStrGrid.RowCount - 1 Do

Writeln(MyFile, '|', MainForm.LinkedListStrGrid.Cells[0, I]:6,

' |', MainForm.LinkedListStrGrid.Cells[1, I]:28, '

|');

Writeln(MyFile, '|\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|');

Finally

Close(MyFile);

End;

IfDataSavedInFile := True;

Except

IsCorrect := False;

End;

End;

End;

Procedure ReadFromFile(Var IsCorrect: Boolean; FilePath: String);

Var

MyFile: TextFile;

Begin

If IsCorrect Then

Begin

AssignFile(MyFile, FilePath);

Try

Reset(MyFile);

Try

ReadingProcess(IsCorrect, MyFile);

Finally

Close(MyFile);

End;

Except

IsCorrect := False;

End;

End;

End;

**Library DoubleLinkedList;**

Uses

System.SysUtils,

Vcl.Grids;

Type

PNode = ^TNode;

TNode = Record

Data: Integer;

Prev: PNode;

Next: PNode;

End;

Var

Tail: PNode = Nil;

Head: PNode = Nil;

Procedure InsertInList(Value: Integer);

Var

NewNode: PNode;

Begin

New(NewNode);

NewNode.Data := Value;

NewNode.Prev := Tail;

NewNode.Next := Nil;

If Head = Nil Then

Head := NewNode

Else

Tail^.Next := NewNode;

Tail := NewNode;

End;

Procedure DeleteFromList(Num: Integer);

Var

CurrentNode, PreviousNode, NextNode: PNode;

Count: Integer;

Begin

CurrentNode := Head;

Count := 1;

While (Count < Num) Do

Begin

CurrentNode := CurrentNode^.Next;

Inc(Count);

End;

PreviousNode := CurrentNode^.Prev;

NextNode := CurrentNode^.Next;

If PreviousNode <> Nil Then

PreviousNode^.Next := NextNode

Else

Head := NextNode;

If NextNode <> Nil Then

NextNode^.Prev := PreviousNode

Else

Tail := CurrentNode^.Prev;

Dispose(CurrentNode);

End;

Procedure PrintList(Var LinkedListStrGrid: TStringGrid);

Var

CurrentNode: PNode;

HighI, I: Integer;

Begin

CurrentNode := Tail;

HighI := 1;

While CurrentNode <> Nil Do

Begin

LinkedListStrGrid.Cells[1, HighI] := IntToStr(CurrentNode.Data);

CurrentNode := CurrentNode.Prev;

Inc(HighI);

End;

Dec(HighI);

For I := HighI DownTo 1 Do

LinkedListStrGrid.Cells[0, HighI - I + 1] := IntToStr(I);

End;

Exports

InsertInList,

DeleteFromList,

PrintList;

{$R \*.res}

Begin

End.

Код программы на **C#**:

using System.Text;

class Proj4\_2 {

class Node {

public int Data { get; set; }

public Node? Previous { get; set; }

public Node? Next { get; set; }

}

enum MainMenu {

DISPLAY,

ADD,

REMOVE,

FILESAVE,

LOADFILE,

REFERENCE,

EXIT

}

const int MIN\_FILE\_WAY\_SIZE = 4;

const int MIN\_NUM\_OF\_VALUES = 0;

const int MAX\_NUM = 1\_000\_000\_000;

const int MIN\_NUM = -1\_000\_000\_000;

static Node? head;

static Node? tail;

static string[] mainOptions = { " < Просмотреть список > ", " < Добавить

значение > ", " < Удалить значение > ", " <

Сохранить список в файл > ", " < Загрузить

значение из файла > ", " < Справка > ", " <

Выход > " };

static string[] exitOption = { "<- Вернуться" };

static void showReference() {

Console.Clear();

string prompt = $"""

Инструкция:

1. Чтобы просмотреть список в обратном порядке выберите

{mainOptions[0]};

2. Добавить новое значение можно выбрав {mainOptions[1]};

3. Выберите {mainOptions[2]}, чтобы удалить какое-то значение;

4. Просмотрите инструкцию выбрав {mainOptions[3]};

5. Чтобы выйти выбирайте вариант {mainOptions[4]}.

Значения в массиве находятся в диапазоне от {MIN\_NUM} до

{MAX\_NUM};

Работа с файлами:

Файл строго формата .txt!

При вводе числа через файл в файле находится только новое

значение.

При сохранении в файл помещается та же таблица, что и в пункте

{mainOptions[0]}

При прсомотре списка он выводится в виде таблицы и в обратном

порядке

в соответствии с условием задания.

""";

getSelectedIndex(prompt, exitOption);

}

static void addValueInList(int data) {

Node newNode = new Node();

newNode.Data = data;

if (head == null) {

head = newNode;

tail = newNode;

} else {

newNode.Previous = tail;

tail.Next = newNode;

tail = newNode;

}

}

static void removeValueFromList(int numOfValue) {

if (head == null) return;

Node? currentNode = head;

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

currentNode = currentNode.Next ?? currentNode;

if (currentNode.Previous != null)

currentNode.Previous.Next = currentNode.Next;

else head = currentNode.Next;

if (currentNode.Next != null)

currentNode.Next.Previous = currentNode.Previous;

else tail = currentNode.Previous;

}

static string ReversePrint() {

string prompt = string.Empty;

prompt = "Список отображён в обратном порядке: ";

Node? currentNode = tail;

if (currentNode == null) return prompt += "список пуст.";

int valCounter = 0;

string[] values = new string[1];

string[] bufferVal = new string[1];

prompt += '\n';

prompt += " \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \n";

prompt += "| Номер | Значение |\n";

prompt += "|------------|---------------------|\n";

while (currentNode != null) {

values[valCounter] = currentNode.Data.ToString();

bufferVal = new string[values.Length];

for (int i = 0; i < values.Length; ++i)

bufferVal[i] = values[i];

values = new string[values.Length + 1];

for (int i = 0; i < bufferVal.Length; ++i)

values[i] = bufferVal[i];

valCounter++;

currentNode = currentNode.Previous;

}

for (int i = valCounter; i > 0; --i)

prompt += "| " + i.ToString().PadLeft(10) + " | " + values[valCounter

- i].PadLeft(19) + " |\n";

prompt += "|\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|";

return prompt;

}

static void displayOptions(string prompt, string[] options,

int selectedIndex = 0) {

Console.WriteLine(prompt);

for (int i = 0; i < options.Length; ++i) {

string curentOption = options[i];

string prefix;

if (i == selectedIndex) {

prefix = "\*";

Console.ForegroundColor = ConsoleColor.Black;

Console.BackgroundColor = ConsoleColor.White;

}

else {

prefix = " ";

Console.ForegroundColor = ConsoleColor.White;

Console.BackgroundColor = ConsoleColor.Black;

}

Console.WriteLine($"{prefix} {curentOption}");

}

Console.ResetColor();

}

static int getSelectedIndex(string prompt, string[] options,

int selectedIndex = 0) {

ConsoleKey keyPressed;

do {

Console.Clear();

displayOptions(prompt, options, selectedIndex);

ConsoleKeyInfo keyInfo = Console.ReadKey(true);

keyPressed = keyInfo.Key;

if (keyPressed == ConsoleKey.UpArrow)

if (--selectedIndex == -1) selectedIndex = options.Length - 1;

if (keyPressed == ConsoleKey.DownArrow)

if (++selectedIndex == options.Length) selectedIndex = 0;

} while (keyPressed != ConsoleKey.Enter);

return selectedIndex;

}

static MainMenu searchQurentMainMethod(int curentIndex) {

switch (curentIndex) {

case (int)MainMenu.DISPLAY: return MainMenu.DISPLAY;

case (int)MainMenu.ADD: return MainMenu.ADD;

case (int)MainMenu.REMOVE: return MainMenu.REMOVE;

case (int)MainMenu.FILESAVE: return MainMenu.FILESAVE;

case (int)MainMenu.LOADFILE: return MainMenu.LOADFILE;

case (int)MainMenu.REFERENCE: return MainMenu.REFERENCE;

default: return MainMenu.EXIT;

}

}

static int inputNumber(int MAX, int MIN) {

int num = 0;

bool isIncorrect = true;

do {

try {

num = Convert.ToInt32(Console.ReadLine());

isIncorrect = false;

}

catch {

Console.Error.Write("Ошибка!!! Введите целое число: ");

isIncorrect = true;

}

if ((num > MAX || num < MIN) && !isIncorrect) {

Console.Error.Write($"Ошибка!!! Диапазон числа от {MIN} до

{MAX}: ");

isIncorrect = true;

}

} while (isIncorrect);

return num;

}

static void addValueProc() {

Console.Clear();

Console.Write("Введите новое значение: ");

int num = inputNumber(MAX\_NUM, MIN\_NUM);

addValueInList(num);

string prompt = "Новое значение добавлено в список: " + num;

getSelectedIndex(prompt, exitOption);

}

static void printProc() {

Console.Clear();

string prompt = ReversePrint();

getSelectedIndex(prompt, exitOption);

}

static int findNumbOfValues() {

int numOfEl = 0;

Node? currentNode = tail;

while (currentNode != null) {

numOfEl++;

currentNode = currentNode.Previous;

}

return numOfEl;

}

static void removeProc() {

Console.Clear();

string prompt = string.Empty;

int valCounter = findNumbOfValues();

if (valCounter == 0) {

prompt = "У вас нет ниодного значения...";

getSelectedIndex(prompt, exitOption);

return;

}

string allElements = ReversePrint();

Console.WriteLine(allElements);

Console.Write($"{MIN\_NUM\_OF\_VALUES} - покинуть удаление;\n");

Console.Write("Введите номер удаляемого элемента: ");

int num = inputNumber(valCounter, MIN\_NUM\_OF\_VALUES);

if (num == MIN\_NUM\_OF\_VALUES) return;

removeValueFromList(num);

prompt = ReversePrint();

getSelectedIndex(prompt, exitOption);

}

static bool pathCondition(string filePath) {

if (filePath.Length < MIN\_FILE\_WAY\_SIZE) {

Console.Error.Write("Путь слишком котороткий. Попробуйте снова: ");

return false;

}

string bufstr = filePath.Substring(filePath.Length - MIN\_FILE\_WAY\_SIZE);

if (!bufstr.Equals(".txt")) {

Console.Error.Write("Введите .txt файл. Попробуйте снова: ");

return false;

}

return true;

}

static string inputFilePath() {

string filePath = Console.ReadLine() ?? string.Empty;

while (!pathCondition(filePath))

filePath = Console.ReadLine() ?? string.Empty;

return filePath;

}

static bool isCanWrite(string filePath) {

try {

using (StreamWriter writer = new StreamWriter(filePath))

writer.WriteLine(string.Empty);

return true;

}

catch {

return false;

}

}

static bool isCanRead(string filePath)

{

try {

using (StreamReader reader = new StreamReader(filePath))

reader.Read();

return true;

} catch {

return false;

}

}

static bool accessModifierControl(string accessModifier, string filePath) {

bool resultModifier = true;

switch (accessModifier) {

case "input": resultModifier = isCanRead(filePath); break;

case "output": resultModifier = isCanWrite(filePath); break;

}

return resultModifier;

}

static bool isCanOpenFile(string filePath)

{

FileInfo fileInfo = new FileInfo(filePath);

return fileInfo.Exists;

}

/// <summary>

/// Write "input" if you want to get the file path for input.

/// Write "output" if you want to get the path to the output file.

/// </summary>

/// <param name="accessModifier"></param>

/// <returns></returns>

static string inputPathToTheFile(string accessModifier) {

string filePath = string.Empty;

bool isCorrect = true;

do {

filePath = inputFilePath();

isCorrect = accessModifierControl(accessModifier, filePath) &&

isCanOpenFile(filePath);

if (!isCorrect) Console.Error.Write("Невозможно открыть файл.

Попробуйте другой путь: ");

} while (!isCorrect);

return filePath;

}

static bool isProcesOfFileOutputCorrect(string filePath, string resultStr) {

string prompt = string.Empty;

try {

using (StreamWriter writerOutput = new StreamWriter(filePath))

writerOutput.WriteLine(resultStr);

return true;

} catch {

Console.WriteLine("Ошибка при записи. Попробуйте снова.");

return false;

}

}

static void saveInFileProc() {

Console.Clear();

string prompt = string.Empty;

int valCounter = findNumbOfValues();

if (valCounter == 0) {

prompt = "У вас нет ниодного значения...";

getSelectedIndex(prompt, exitOption);

return;

}

string filePath = string.Empty;

do {

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

filePath = inputPathToTheFile("output");

} while (!isProcesOfFileOutputCorrect(filePath, ReversePrint()));

prompt = "Список успешно сохранён в файл.";

getSelectedIndex(prompt, exitOption);

}

static int inputNumberFromFile(StreamReader inputReader, ref bool

isCorrectInput, int MIN\_NUM, int MAX\_NUM) {

int num = 0;

bool isCorrect = true;

int character, bufChar = 0;

while (isCorrect && isCorrectInput && (character = inputReader.Read()) !=

-1) {

bool isServiceSymbol = character == ' ' || character == '\r' ||

character == '\n';

isCorrectInput = (isServiceSymbol || !(character > '9' &&

character < '0'));

if (!isServiceSymbol && isCorrectInput)

num = num \* 10 + character - 48;

isCorrect = !(bufChar != 0 && isCorrectInput && isServiceSymbol);

isCorrect = isCorrect && !(num > MAX\_NUM);

bufChar = character;

}

isCorrectInput = !(isCorrectInput && (num > MAX\_NUM || num < MIN\_NUM

|| bufChar == 0));

return num;

}

static bool isProcesOfFileInputCorrect(ref int num, ref string filePath) {

bool isCorrectInput = true;

string prompt = string.Empty;

using (StreamReader inputReader = new StreamReader(filePath)) {

num = inputNumberFromFile(inputReader, ref isCorrectInput, MIN\_NUM,

MAX\_NUM);

isCorrectInput = isCorrectInput && inputReader.EndOfStream ? true :

false;

addValueInList(num);

if (!isCorrectInput) Console.WriteLine("Ошибка при чтении. Попробуйте

снова.");

}

return isCorrectInput;

}

static void loadFromFileProc() {

Console.Clear();

string filePath = string.Empty;

int num = 0;

do {

Console.Write("Введите путь к вашему файлу (\*.txt): ");

filePath = inputPathToTheFile("input");

} while (!isProcesOfFileInputCorrect(ref num, ref filePath));

string prompt = "Число успешно считано из файла.";

getSelectedIndex(prompt, exitOption);

}

static void workWithMethod(MainMenu curentMethod) {

switch (curentMethod) {

case MainMenu.DISPLAY: printProc(); break;

case MainMenu.ADD: addValueProc(); break;

case MainMenu.REMOVE: removeProc(); break;

case MainMenu.FILESAVE: saveInFileProc(); break;

case MainMenu.LOADFILE: loadFromFileProc(); break;

case MainMenu.REFERENCE: showReference(); break;

}

}

static void mainWorkBlock() {

MainMenu curentMethod = 0;

do {

string prompt = "Выберите, что хотите сделать: ";

curentMethod = searchQurentMainMethod(getSelectedIndex(prompt,

mainOptions));

workWithMethod(curentMethod);

} while (curentMethod != MainMenu.EXIT);

}

static void Main(String[] args) {

Encoding.RegisterProvider(CodePagesEncodingProvider.Instance);

Console.OutputEncoding = Encoding.GetEncoding(1251);

Console.InputEncoding = Encoding.GetEncoding(1251);

Console.Title = "Двусвязный список™";

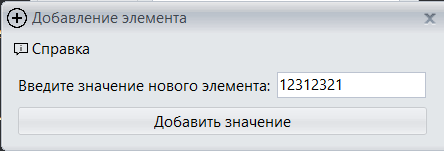
mainWorkBlock();

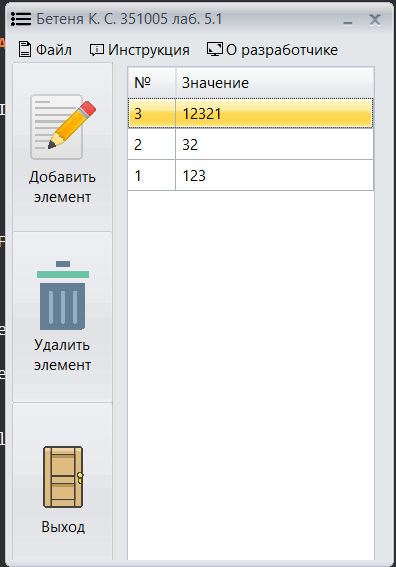
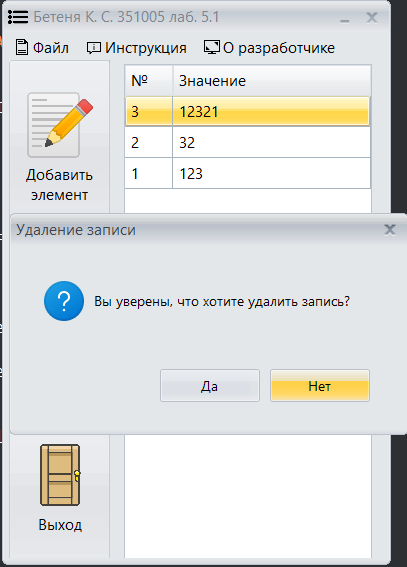
Environment.Exit(0);

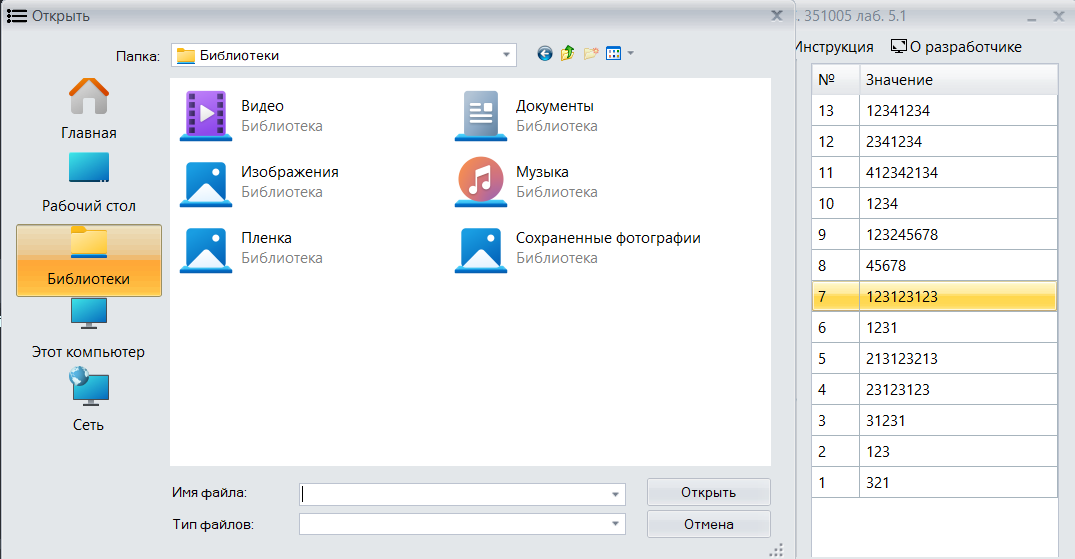
}

}

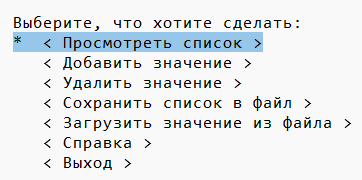
Результат в **Delphi**:

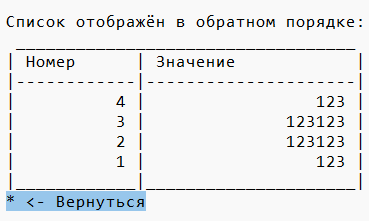


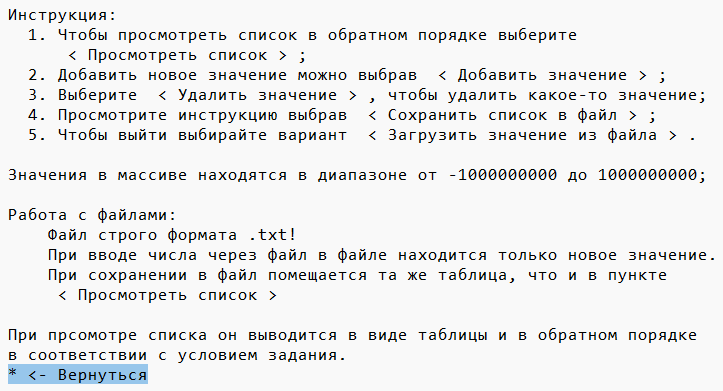
 

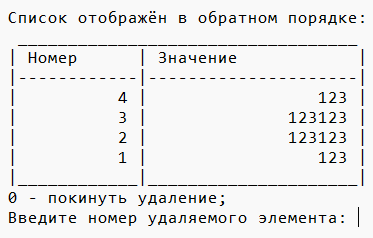


Результат в **C#**:









Блок-схема:

