УО «Белорусский государственный университет информатики и радиоэлектроники»

Кафедра ПОИТ

Отчет по лабораторной работе №5.1

по предмету «Основы алгоритмизации и программирования»

Вариант 20

Выполнил:

Захвей И.В.

Гр. 351005

Проверил:

Данилова Г. В.

Минск 2024

**Задание:**

Разработать программу слияния двух односвязных упорядоченных по неубыванию линейных списков в один упорядоченный список.

**Код программы Delphi:**

**MainUnit.pas**

Type

TGridCracker = Class(TStringGrid);

Var

ManeForm: TManeForm;

IsSaved: Boolean = True;

FirstHead, SecondHead: ListPointer;

Implementation

{$R \*.dfm}

Procedure FillGrid(HeadPt: ListPointer; Grid: TStringGrid);

Var

I, Size: Integer;

Begin

Grid.Visible := True;

Size := Len(HeadPt);

If Size > 5 Then

Begin

Grid.Width := (Grid.DefaultColWidth + 3) \* 5;

Grid.Height := (Grid.DefaultRowHeight + 3) \* 2 + 25;

End

Else

Begin

Grid.Width := (Grid.DefaultColWidth + 4) \* Size;

Grid.Height := (Grid.DefaultRowHeight + 3) \* 2;

End;

Grid.ColCount := Size;

Grid.RowCount := 2;

For I := 0 To Size - 1 Do

Begin

HeadPt := HeadPt^.Next;

Grid.Cells[I, 0] := IntToStr(I + 1);

Grid.Cells[I, 1] := IntToStr(HeadPt^.Data);

End;

End;

Procedure ClearGrid(Grid: TStringGrid);

Var

J, I: Integer;

Begin

Grid.Visible := False;

For I := 0 To Grid.ColCount - 1 Do

For J := 0 To Grid.RowCount - 1 Do

Grid.Cells[I, J] := '';

End;

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

Begin

If SaveButtonMenu.Enabled And Not IsSaved Then

Case Application.MessageBox('Сохранить данные перед выходом?', 'Выход',

MB\_YESNOCANCEL + MB\_ICONQUESTION + MB\_DEFBUTTON3) Of

IDYES:

Begin

SaveButtonMenu.Click;

CanClose := True;

End;

IDNO:

CanClose := True;

IDCANCEL:

CanClose := False;

End

Else

Case Application.MessageBox('Вы точно хотите выйти?', 'Выход',

MB\_YESNO + MB\_ICONQUESTION + MB\_DEFBUTTON2) Of

IDYES:

CanClose := True;

IDNO:

CanClose := False;

End;

End;

Procedure TManeForm.FormCreate(Sender: TObject);

Begin

New(FirstHead);

FirstHead^.Next := Nil;

New(SecondHead);

SecondHead^.Next := Nil;

InfLabel.Caption := INFTEXT;

End;

Function ReadOneFromFile(Var Numb: Integer; Var MyFile: TextFile;

IsElemRead: Boolean = True): ERRORS\_CODE;

Var

Err: ERRORS\_CODE;

NumbInt: Integer;

NumbStr: String;

Begin

Err := SUCCESS;

NumbInt := 0;

Try

Read(MyFile, NumbInt);

Except

Err := INCORRECT\_DATA\_FILE;

End;

If Err = SUCCESS Then

If IsElemRead Then

If (NumbInt > MAX\_NUMB) Or (NumbInt < MIN\_NUMB) Then

Err := OUT\_OF\_BORDER

Else

Numb := NumbInt

Else If (NumbInt > MAX\_SIZE) Or (NumbInt < MIN\_SIZE) Then

Err := OUT\_OF\_BORDER\_SIZE

Else

Numb := NumbInt;

ReadOneFromFile := Err;

End;

Procedure TManeForm.SaveButtonMenuClick(Sender: TObject);

Var

OutFile: TextFile;

I, J: Integer;

Begin

If SaveTextFileDialog.Execute() Then

Begin

AssignFile(OutFile, SaveTextFileDialog.FileName);

Rewrite(OutFile);

Writeln(OutFile, 'Merget List');

for I := 0 to MergedListGrid.ColCount-1 do

Write(OutFile, MergedListGrid.Cells[I, 1] + ' ');

CloseFile(OutFile);

IsSaved := True;

End;

End;

Procedure TManeForm.ManualButtonMenuClick(Sender: TObject);

Var

ManualForm: TManualForm;

Begin

ManualForm := TManualForm.Create(Self);

ManualForm.ShowModal;

ManualForm.Free;

End;

Procedure TManeForm.AddFirstButtonClick(Sender: TObject);

Var

AddForm: TAddForm;

Res: TModalResult;

NewElem: Integer;

Begin

AddForm := TAddForm.Create(Self);

AddForm.FormCreate(FirstHead, Self);

Res := AddForm.ShowModal();

if Res = mrOk then

Begin

FillGrid(FirstHead, FirstListGrid);

ClearGrid(MergedListGrid);

End;

AddForm.Free();

if FirstListGrid.Visible and SecondListGrid.Visible then

CheckButton.Enabled := True;

End;

procedure TManeForm.AddSecondButtonClick(Sender: TObject);

Var

AddForm: TAddForm;

Res: TModalResult;

NewElem: Integer;

Begin

AddForm := TAddForm.Create(Self);

AddForm.FormCreate(SecondHead, Self);

Res := AddForm.ShowModal();

if Res = mrOk then

FillGrid(SecondHead, SecondListGrid);

AddForm.Free();

if FirstListGrid.Enabled and SecondListGrid.Enabled then

CheckButton.Enabled := True;

end;

procedure TManeForm.CheckButtonClick(Sender: TObject);

Var

MergedHead: ListPointer;

begin

New(MergedHead);

Merge(MergedHead, FirstHead, SecondHead);

FillGrid(MergedHead, MergedListGrid);

IsSaved := False;

end;

Procedure TManeForm.DeveloperButtonMenuClick(Sender: TObject);

Var

DeveloperForm: TDeveloperForm;

Begin

DeveloperForm := TDeveloperForm.Create(Self);

DeveloperForm.ShowModal;

DeveloperForm.Free;

End;

Procedure TManeForm.ExitButtonMenuClick(Sender: TObject);

Begin

Close();

End;

Function IsAllCellFill(Grid: TStringGrid; Key: Char;

CurCell: TInplaceEdit): Boolean;

Var

IsFilled: Boolean;

I, J: Integer;

Begin

IsFilled := True;

For I := 1 To Grid.ColCount - 1 Do

For J := 1 To Grid.RowCount - 1 Do

Begin

If (Grid.Col = I) And (Grid.Row = J) And Not(Key = VOID) Then

Begin

If (Grid.Cells[I, J] = '') And Not CharInSet(Key, DIGITS) Then

IsFilled := False;

With CurCell Do

If (Key = BACKSPACE) And

(InsertKey(SelStart, Key, SelLength, Text) = '') Then

IsFilled := False;

End

Else If (Grid.Cells[I, J] = '') Or (Grid.Cells[I, J] = '-') Then

IsFilled := False;

End;

IsAllCellFill := IsFilled;

End;

Procedure TManeForm.ListGridKeyDown(Sender: TObject; Var Key: Word;

Shift: TShiftState);

Begin

If ((SsShift In Shift) Or (SsCtrl In Shift)) And

Not((Key = VK\_RIGHT) Or (Key = VK\_LEFT)) Then

Key := 0;

If Key = VK\_DELETE Then

Key := 0;

If (Key = VK\_RETURN) And (CheckButton.Enabled) Then

CheckButton.Click;

If (Key = VK\_DOWN) And CheckButton.Enabled Then

CheckButton.SetFocus;

End;

Procedure TManeForm.ListGridKeyPress(Sender: TObject; Var Key: Char);

Var

GridCel: TGridCracker;

EditingCell: TInplaceEdit;

Begin

GridCel := TGridCracker(Sender);

EditingCell := GridCel.InplaceEditor;

TotalKeyPress(Key, EditingCell.SelStart, EditingCell.SelLength, MIN\_NUMB,

MAX\_NUMB, EditingCell.Text);

If IsAllCellFill(GridCel, Key, EditingCell) Then

CheckButton.Enabled := True

Else

CheckButton.Enabled := False;

If Key <> VOID Then

Begin

SaveButtonMenu.Enabled := False;

End;

End;

End.

**AddUnit.pas**

Var

AddForm: TAddForm;

Implementation

{$R \*.dfm}

Procedure TAddForm.AddButtonClick(Sender: TObject);

Begin

Add(HeadPt, StrToInt(NewElemEdit.Text));

End;

Procedure TAddForm.FormCreate(HeadPt: ListPointer; Sender: TObject);

Begin

Self.HeadPt := HeadPt;

End;

Procedure TAddForm.FormShow(Sender: TObject);

Begin

NewElemEdit.SetFocus();

End;

Procedure TAddForm.NewElemEditChange(Sender: TObject);

Begin

AddButton.Enabled := NewElemEdit.Text <> '';

End;

Procedure TAddForm.NewElemEditKeyDown(Sender: TObject; Var Key: Word;

Shift: TShiftState);

Begin

If (Key = VK\_RETURN) And (AddButton.Enabled) Then

AddButton.Click();

If ((SsShift In Shift) Or (SsCtrl In Shift)) And

Not((Key = VK\_RIGHT) Or (Key = VK\_LEFT)) Then

Key := 0;

If Key = VK\_DELETE Then

Key := 0;

End;

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

Begin

With NewElemEdit Do

TotalKeyPress(Key, SelStart, SelLength, MIN\_NUMB, MAX\_NUMB, Text);

End;

End.

DevInfUnit.pas

Var

DeveloperForm: TDeveloperForm;

Implementation

{$R \*.dfm}

Procedure TDeveloperForm.FormCreate(Sender: TObject);

Begin

InfLabel.Caption := 'Студент группы 351005, Захвей Иван'

End;

End.

**ManualUnit.pas**

Const

TextInf = '1. Элементы списка должны быть от 0 до 99999999 ' + #13#10 +

'2. Чтобы добавить элемент нажмите на соответствующую кнопку' + #13#10 +

'3. Данные можно сохранить в файл';

Var

ManualForm: TManualForm;

Implementation

{$R \*.dfm}

Procedure TManualForm.FormCreate(Sender: TObject);

Begin

ManualLabel.Caption := TextInf;

End;

End.

**BackUnit.pas**

Implementation

Function InsertKey(Index: Integer; SubStr: Char; SelLen: Integer;

Text: String): String;

Var

ResultText: String;

Begin

ResultText := Text;

If (SubStr = BACKSPACE) And (SelLen = 0) Then

Delete(ResultText, Index, 1)

Else

Begin

Delete(ResultText, Index + 1, SelLen);

If Substr <> BACKSPACE Then

ResultText.Insert(Index, String(SubStr));

End;

InsertKey := ResultText;

End;

Function CountOfSymbolInt(Num: Integer): Integer;

Var

NumLen: Integer;

Begin

NumLen := 0;

If Num < 0 Then

Inc(NumLen);

Repeat

Inc(NumLen);

Num := Num Div 10;

Until (Num = 0);

CountOfSymbolInt := NumLen;

End;

Procedure TotalKeyPress(Var Key: Char; SelStart, SelLength: Integer;

Const MIN, MAX: Integer; Text: String);

Var

ResultNum, RBorder, NumLen: Integer;

Buffer, Output: String;

Begin

Output := InsertKey(SelStart, Key, SelLength, Text);

If (Length(Output) <> 0) And (Output <> '-') Then

Begin

Try

ResultNum := StrToInt(Output);

Except

Key := VOID;

End;

If Key <> VOID Then

Begin

NumLen := CountOfSymbolInt(ResultNum);

If NumLen <> Length(Output) Then

Key := VOID;

If (ResultNum > MAX) Or (ResultNum < MIN) Then

Key := VOID;

End;

End

Else If (Output = '-') And (MIN >= 0) Then

Key := VOID;

End;

End.

**LinkedListUnit.pas**

Type

ListPointer = ^LinkedList;

LinkedList = Record

Data: Integer;

Next: ListPointer;

End;

Procedure Add(HeadPt: ListPointer; Elem: Integer);

Procedure Merge(Dest, First, Second: ListPointer);

Function Len(HeadPt: ListPointer): Integer;

Implementation

Procedure Merge(Dest, First, Second: ListPointer);

Begin

First := First^.Next;

Second := Second^.Next;

Repeat

New(Dest^.Next);

Dest := Dest^.Next;

if First^.Data > Second^.Data then

Begin

Dest^.Data := Second^.Data;

Second := Second^.Next;

End

Else

Begin

Dest^.Data := First^.Data;

First := First^.Next;

End;

Until (First = Nil) Or (Second = Nil);

while First <> Nil do

Begin

New(Dest^.Next);

Dest := Dest^.Next;

Dest^.Data := First^.Data;

First := First^.Next;

End;

while Second <> Nil do

Begin

New(Dest^.Next);

Dest := Dest^.Next;

Dest^.Data := Second^.Data;

Second := Second^.Next;

End;

End;

Function Len(HeadPt: ListPointer): Integer;

Var

Length: Integer;

Begin

Length := 0;

While HeadPt^.Next <> Nil Do

Begin

Inc(Length);

HeadPt := HeadPt^.Next;

End;

Len := Length;

End;

Procedure Add(HeadPt: ListPointer; Elem: Integer);

Var

Temp: Integer;

TempPt: ListPointer;

IsAdded: Boolean;

Begin

IsAdded := False;

While (HeadPt^.Next <> Nil) And (Not IsAdded) Do

Begin

HeadPt := HeadPt^.Next;

If HeadPt^.Data > Elem Then

Begin

Temp := HeadPt^.Data;

TempPt := HeadPt^.Next;

HeadPt^.Data := Elem;

New(HeadPt^.Next);

HeadPt := HeadPt^.Next;

HeadPt^.Data := Temp;

HeadPt^.Next := TempPt;

IsAdded := True;

End;

End;

If Not IsAdded Then

Begin

New(HeadPt^.Next);

HeadPt := HeadPt^.Next;

HeadPt^.Data := Elem;

End;

End;

Function Delete(HeadPt: ListPointer; Index: Integer): Boolean;

Var

IsDelete: Boolean;

Count: Integer;

Begin

IsDelete := False;

Count := 0;

While HeadPt^.Next <> Nil Do

Begin

Inc(Count);

HeadPt := HeadPt^.Next;

If Count = Index Then

Begin

HeadPt^.Data := HeadPt^.Next^.Data;

HeadPt^.Next := HeadPT^.Next^.Next;

IsDelete := True;

End;

End;

Delete := IsDelete;

End;

End.

БИБЛИОТЕЧНЫЙ МОДУЛЬ

library LinkedListDll;

{$R \*.res}

Type

ListPointer = ^LinkedList;

LinkedList = Record

Data: Integer;

Next: ListPointer;

End;

Procedure Merge(Dest, First, Second: ListPointer);

Begin

First := First^.Next;

Second := Second^.Next;

Repeat

New(Dest^.Next);

Dest := Dest^.Next;

if First^.Data > Second^.Data then

Begin

Dest^.Data := Second^.Data;

Second := Second^.Next;

End

Else

Begin

Dest^.Data := First^.Data;

First := First^.Next;

End;

Until (First = Nil) Or (Second = Nil);

while First <> Nil do

Begin

New(Dest^.Next);

Dest := Dest^.Next;

Dest^.Data := First^.Data;

First := First^.Next;

End;

while Second <> Nil do

Begin

New(Dest^.Next);

Dest := Dest^.Next;

Dest^.Data := Second^.Data;

Second := Second^.Next;

End;

End;

Function Len(HeadPt: ListPointer): Integer;

Var

Length: Integer;

Begin

Length := 0;

While HeadPt^.Next <> Nil Do

Begin

Inc(Length);

HeadPt := HeadPt^.Next;

End;

Len := Length;

End;

Procedure Add(HeadPt: ListPointer; Elem: Integer);

Var

Temp: Integer;

TempPt: ListPointer;

IsAdded: Boolean;

Begin

IsAdded := False;

While (HeadPt^.Next <> Nil) And (Not IsAdded) Do

Begin

HeadPt := HeadPt^.Next;

If HeadPt^.Data > Elem Then

Begin

Temp := HeadPt^.Data;

TempPt := HeadPt^.Next;

HeadPt^.Data := Elem;

New(HeadPt^.Next);

HeadPt := HeadPt^.Next;

HeadPt^.Data := Temp;

HeadPt^.Next := TempPt;

IsAdded := True;

End;

End;

If Not IsAdded Then

Begin

New(HeadPt^.Next);

HeadPt := HeadPt^.Next;

HeadPt^.Data := Elem;

End;

End;

Function Delete(HeadPt: ListPointer; Index: Integer): Boolean;

Var

IsDelete: Boolean;

Count: Integer;

Begin

IsDelete := False;

Count := 0;

While HeadPt^.Next <> Nil Do

Begin

Inc(Count);

HeadPt := HeadPt^.Next;

If Count = Index Then

Begin

HeadPt^.Data := HeadPt^.Next^.Data;

HeadPt^.Next := HeadPT^.Next^.Next;

IsDelete := True;

End;

End;

Delete := IsDelete;

End;

Exports Delete, Add, Len, Merge;

begin

end.

**Код программы Java:**

import java.io.File;

import java.io.IOException;

import java.io.PrintWriter;

import java.util.Scanner;

class ListPt {

int data;

ListPt next;

ListPt (int data, ListPt next) {

this.data = data;

this.next = next;

}

}

public class Main {

enum Choice {

addElem("Добавить"),

merge("Слить два списка"),

close("Закрыть");

private final String inf;

Choice (String infLine) {

this.inf = infLine;

}

private String getInf(){return this.ordinal() + ") " + this.inf;}

}

enum ErrCode {

SUCCESS,

INCORRECT\_DATA,

FILE\_NOT\_EXIST,

EMPTY\_LINE,

NOT\_TXT,

IN\_OUT\_FILE\_EXCEPTION,

EMPTY\_LISTS,

}

static final String[] ERRORS = {"Удача",

"Данные не корректные или число слишком большое (должно быть

от %d до %d)\n",

"Такого файла не существует",

"Строка пустая, будьте внимательны",

"Файл не .txt",

"Exception with output/input from the file",

"Списки должны иметь хотя бы по одному элементу"};

static final String INFORMATION\_TEXT = """

Инструкция:

1) Элементы списка должны быть от 0 до 99999999

2) Чтобы добавить элемент выберите соответствующую кнопку

3) Данные предложится сохранить в файл

""",

CHOICE\_SAVE\_TEXT = """

Вы хотите сохранить результат?

1) да

2) нет

""";

static final int MIN\_ELEM = 0,

MAX\_ELEM = 99999999;

static void add(ListPt headPt, int elem) {

boolean isAdded = false;

int temp;

ListPt tempPt;

while ((headPt.next != null) && !isAdded) {

headPt = headPt.next;

if (headPt.data > elem) {

temp = headPt.data;

tempPt = headPt.next;

headPt.data = elem;

headPt.next = new ListPt(temp, tempPt);

isAdded = true;

}

}

if (!isAdded)

headPt.next = new ListPt(elem, null);

}

static void merge(ListPt dest, ListPt first, ListPt second) {

first = first.next;

second = second.next;

int temp;

do {

if (first.data > second.data) {

temp = second.data;

second = second.next;

} else {

temp = first.data;

first = first.next;

}

dest.next = new ListPt(temp, null);

dest = dest.next;

} while (first != null && second != null);

while (first != null) {

dest.next = new ListPt(first.data, null);

dest = dest.next;

first = first.next;

}

while (second != null) {

dest.next = new ListPt(second.data, null);

dest = dest.next;

second = second.next;

}

}

static void printList(ListPt headPt, int countList) {

System.out.printf("Список %d: ", countList);

if (headPt.next == null)

System.out.print("пусто");

while (headPt.next != null) {

headPt = headPt.next;

System.out.print(headPt.data + " ");

}

System.out.println();

}

static void printMenu() {

Choice[] choices = Choice.values();

for (Choice choice : choices) {

System.out.println(choice.getInf());

}

}

static void printInf(Scanner input) {

System.out.println(INFORMATION\_TEXT);

System.out.println("нажмите enter чтобы продолжить");

input.nextLine();

}

static ErrCode enterOneNum(int[] numberArr, Scanner input, final int MIN, final int MAX) {

int number = 0;

ErrCode err = ErrCode.SUCCESS;

try {

number = Integer.parseInt(input.nextLine());

} catch (NumberFormatException e) {

err = ErrCode.INCORRECT\_DATA;

}

if ((err == ErrCode.SUCCESS) && (number < MIN || number > MAX))

err = ErrCode.INCORRECT\_DATA;

numberArr[0] = err == ErrCode.SUCCESS ? number : 0;

return err;

}

static int getNumConsole(Scanner input, final int MIN, final int MAX) {

ErrCode err;

int[] numberArr = {0};

do {

err = enterOneNum(numberArr, input, MIN, MAX);

if (err != ErrCode.SUCCESS) {

System.err.printf(ERRORS[err.ordinal()], MIN, MAX);

System.out.println("Введите снова");

}

} while (err != ErrCode.SUCCESS);

return numberArr[0];

}

static Choice getChoice(Scanner input) {

int choice;

int maxChoice = Choice.values().length - 1;

choice = getNumConsole(input, 0, maxChoice);

return Choice.values()[choice];

}

static void addToList(ListPt headPt, Scanner input) {

System.out.printf("Введите новый элемент списка (от %d до %d)\n", MIN\_ELEM, MAX\_ELEM);

int elem = getNumConsole(input, MIN\_ELEM, MAX\_ELEM);

add(headPt, elem);

}

static ErrCode validateFileExistence(String fileName) {

File file = new File(fileName);

return file.exists() ? ErrCode.SUCCESS : ErrCode.FILE\_NOT\_EXIST;

}

static ErrCode validateFileExtension(String fileName) {

return fileName.endsWith(".txt") ? ErrCode.SUCCESS : ErrCode.NOT\_TXT;

}

static ErrCode enterFileName(String[] fileName, Scanner input) {

ErrCode err;

fileName[0] = input.nextLine();

if (fileName[0].isEmpty())

err = ErrCode.EMPTY\_LINE;

else {

err = validateFileExistence(fileName[0]);

if (err.equals(ErrCode.SUCCESS)) {

err = validateFileExtension(fileName[0]);

}

}

return err;

}

static String getFileName(Scanner input) {

String[] fileName = {""};

ErrCode err;

System.out.println("Enter full path to file");

do {

err = enterFileName(fileName, input);

if (err != ErrCode.SUCCESS) {

System.err.println(ERRORS[err.ordinal()]);

}

} while (err != ErrCode.SUCCESS);

return fileName[0];

}

static void saveToFile(Scanner input, ListPt headPt) {

ErrCode err;

do {

err = ErrCode.SUCCESS;

String fileName = getFileName(input);

try (PrintWriter file = new PrintWriter(fileName)) {

while (headPt.next != null) {

headPt = headPt.next;

file.write(headPt.data + " ");

}

} catch (IOException e) {

err = ErrCode.IN\_OUT\_FILE\_EXCEPTION;

System.err.println(ERRORS[err.ordinal()]);

}

} while (err != ErrCode.SUCCESS);

}

static void saveOrNot(Scanner input, ListPt headPt) {

System.out.println(CHOICE\_SAVE\_TEXT);

int choice = getNumConsole(input, 1, 2);

if (choice == 1) {

saveToFile(input, headPt);

}

}

static boolean doFunction(ListPt firstList, ListPt secondList, Scanner input) {

Choice choice = getChoice(input);

boolean isClose = false;

switch (choice) {

case addElem -> {

System.out.println("Введите список в который хотите добавить элемент (1 или 2)");

int numList = getNumConsole(input, 1, 2);

if (numList == 1) {

addToList(firstList, input);

} else {

addToList(secondList, input);

}

}

case merge -> {

ListPt mergedList = new ListPt(0, null);

if (firstList.next != null && secondList.next != null) {

merge(mergedList, firstList, secondList);

printList(mergedList, 0);

saveOrNot(input, mergedList);

} else {

System.err.println(ERRORS[ErrCode.EMPTY\_LISTS.ordinal()]);

}

System.out.println();

}

case close -> isClose = true;

}

return isClose;

}

public static void main(String[] args) {

ListPt firstHead = new ListPt(0, null);

ListPt secondHead = new ListPt(0, null);

Scanner input = new Scanner(System.in);

printInf(input);

boolean isClose;

do {

printList(firstHead, 1);

printList(secondHead, 2);

printMenu();

isClose = doFunction(firstHead, secondHead, input);

} while (!isClose);

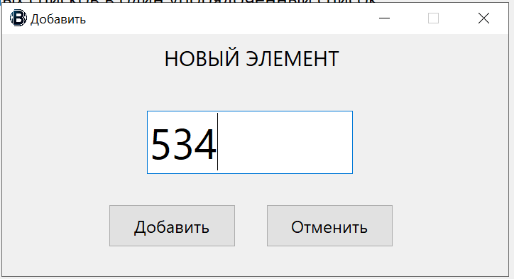
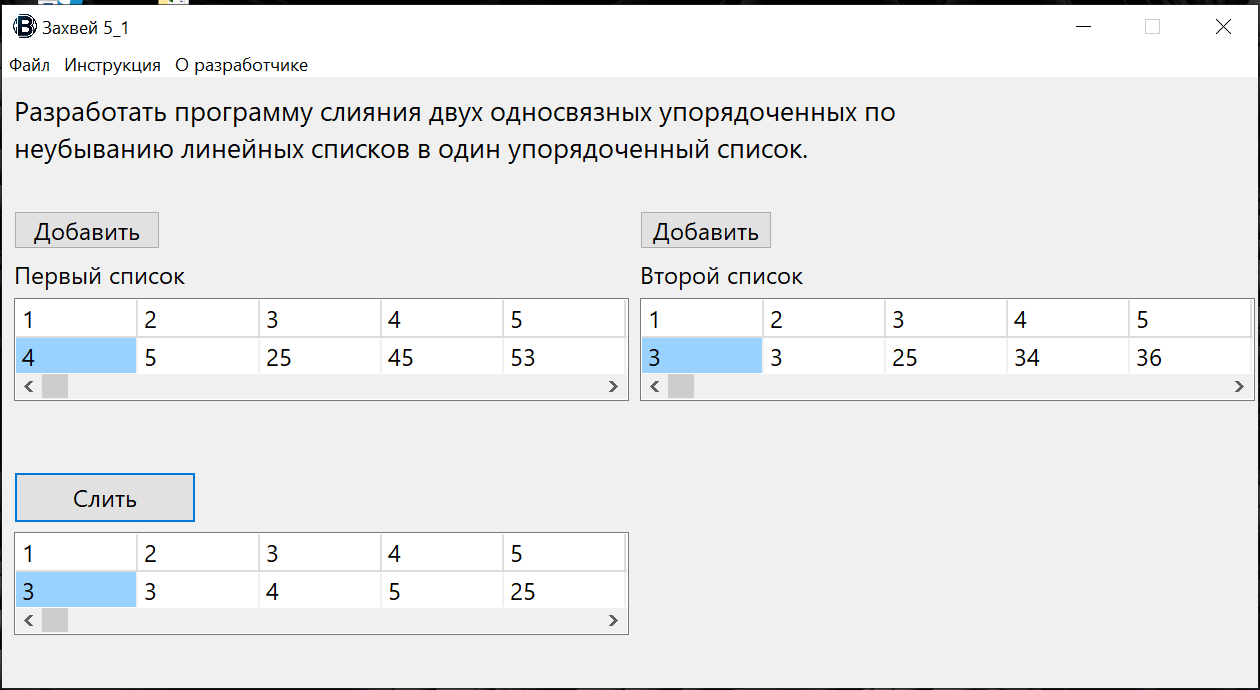
input.close();

}

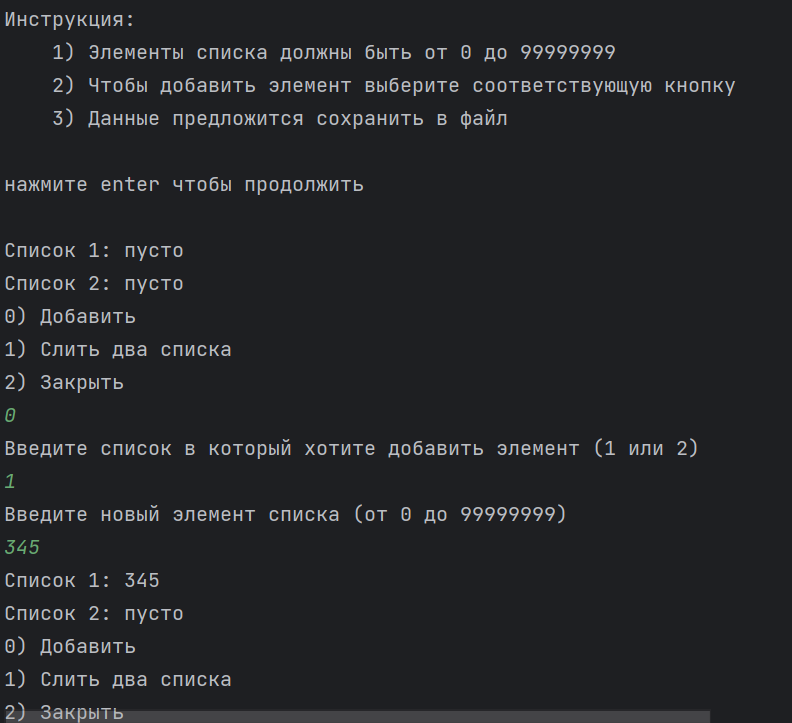
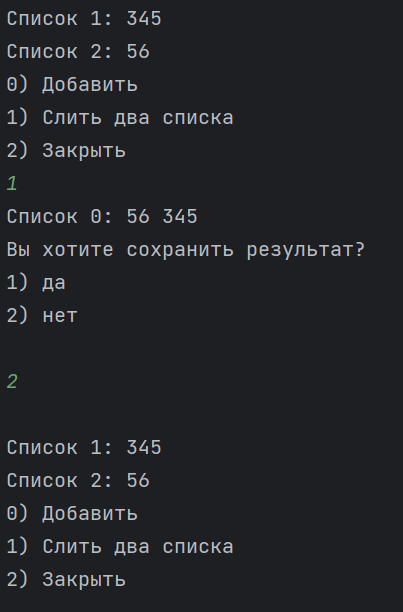
}

**Скриншоты:**

**Delphi:**

****

**Java:**

** **

**Блок-схема:** 