УО «Белорусский государственный университет информатики и радиоэлектроники»

Кафедра ПОИТ

Отчет по лабораторной работе №4.2

по предмету «Основы алгоритмизации и программирования»

Вариант 12

Выполнил:

Галуха П. А.

Гр. 351005

Проверил:

Данилова Г. В.

Минск 2024

**Задание:**

Дано n различных натуральных чисел. Разработать рекурсивную процедуру формирования всех перестановок этих чисел.

**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.StdCtrls, Vcl.Grids, Vcl.Menus, InstructionUnit,

DeveloperUnit, Vcl.ExtDlgs, Clipbrd;

Type

TStringGridEx = Class(TStringGrid);

ERRORS\_CODE = (CORRECT,

INCORRECT\_NUM,

INCORRECT\_RANGE,

IS\_NOT\_DIFFERENT,

INCORRECT\_NUMS\_AMOUNT,

EXTRA\_DATA);

TArr = Array Of Integer;

TMainForm = Class(TForm)

TabsMainMenu: TMainMenu;

FileMenuItem: TMenuItem;

OpenMenuItem: TMenuItem;

SaveMenuItem: TMenuItem;

SeparatorMenuItem: TMenuItem;

ExitMenuItem: TMenuItem;

InstructionMenuItem: TMenuItem;

DeveloperMenuItem: TMenuItem;

OpenTextFileDialog1: TOpenTextFileDialog;

SaveTextFileDialog1: TSaveTextFileDialog;

TaskLabel: TLabel;

NumsAmountLabel: TLabel;

NumsAmountEdit: TEdit;

NumsLabel: TLabel;

NumsStringGrid: TStringGrid;

ResultButton: TButton;

ResultStringGrid: TStringGrid;

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

Procedure InstructionMenuItemClick(Sender: TObject);

Procedure DeveloperMenuItemClick(Sender: TObject);

Procedure OpenMenuItemClick(Sender: TObject);

Procedure NumsAmountEditChange(Sender: TObject);

procedure NumsAmountEditContextPopup(Sender: TObject; MousePos: TPoint; var Handled: Boolean);

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

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

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

Procedure NumsStringGridExit(Sender: TObject);

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

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

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

Procedure NumsStringGridSelectCell(Sender: TObject; ACol, ARow: Integer; Var CanSelect: Boolean);

Procedure NumsStringGridSetEditText(Sender: TObject; ACol, ARow: Integer; Const Value: String);

Procedure ResultButtonClick(Sender: TObject);

Procedure SaveMenuItemClick(Sender: TObject);

Procedure ExitMenuItemClick(Sender: TObject);

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

Private

{ Private declarations }

Public

{ Public declarations }

End;

Const

ENTER = #13;

BACKSPACE = #8;

NONE = #0;

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

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

DIGITS\_WITHOUT\_ZERO = ['1'..'9'];

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

'Некорректное число в файле!',

'Значение не попадает в диапазон!',

'Значения не различны!',

'Неправильное количество чисел в файле!',

'Лишние данные в файле!');

MIN\_NUMS\_AMOUNT = 1;

MAX\_NUMS\_AMOUNT = 8;

MIN\_NUM = 1;

MAX\_NUM = 10000;

Var

MainForm: TMainForm;

CtrlPressed: Boolean = False;

IsSaved: Boolean = True;

Implementation

{$R \*.dfm}

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;

Function IsValidRange(Num: Integer; Const MIN, MAX: Integer) : Boolean;

Begin

IsValidRange := (Num >= MIN) And (Num <= MAX);

End;

Function AreFileNumsDifferent(NumsArr: TArr; LastIndex: Integer) : ERRORS\_CODE;

Var

Error: ERRORS\_CODE;

I: Integer;

Begin

Error := CORRECT;

I := 0;

While (Error = CORRECT) And (I < LastIndex) Do

Begin

If (NumsArr[I] = NumsArr[LastIndex]) Then

Error := IS\_NOT\_DIFFERENT;

Inc(I);

End;

AreFileNumsDifferent := Error;

End;

Function ReadFileNum(Var InputFile: TextFile; Var Num: Integer; Const MIN, MAX: Integer) : ERRORS\_CODE;

Var

Error: ERRORS\_CODE;

Begin

Error := CORRECT;

Try

Read(InputFile, Num);

Except

Error := INCORRECT\_NUM;

End;

If (Error = CORRECT) And Not IsValidRange(Num, MIN, MAX) Then

Error := INCORRECT\_RANGE;

ReadFileNum := Error;

End;

Function ReadFileNumsAmount(Var InputFile: TextFile; Var NumsAmount: Integer) : ERRORS\_CODE;

Var

Error: ERRORS\_CODE;

Begin

Error := ReadFileNum(InputFile, NumsAmount, MIN\_NUMS\_AMOUNT, MAX\_NUMS\_AMOUNT);

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

Error := EXTRA\_DATA;

ReadFileNumsAmount := Error;

End;

Function ReadFileNums(Var InputFile: TextFile; Var NumsArr: TArr; NumsAmount: Integer) : ERRORS\_CODE;

Var

Error: ERRORS\_CODE;

I: Integer;

Begin

Error := CORRECT;

SetLength(NumsArr, NumsAmount);

I := 0;

While (Error = CORRECT) And Not seekEOLN(InputFile) Do

Begin

If I < NumsAmount Then

Begin

Error := ReadFileNum(InputFile, NumsArr[I], MIN\_NUM, MAX\_NUM);

If Error = CORRECT Then

Error := AreFileNumsDifferent(NumsArr, I);

End

Else

Error := INCORRECT\_NUMS\_AMOUNT;

Inc(I);

End;

If (Error = CORRECT) And (I <> NumsAmount) Then

Error := INCORRECT\_NUMS\_AMOUNT;

ReadFileNums := Error;

End;

Function ReadFileData(Var InputFile: TextFile; Var NumsArr: TArr; Var NumsAmount: Integer) : ERRORS\_CODE;

Var

Error: ERRORS\_CODE;

Begin

Reset(InputFile);

Error := ReadFileNumsAmount(InputFile, NumsAmount);

ReadLn(InputFile);

If Error = CORRECT Then

Error := ReadFileNums(InputFile, NumsArr, NumsAmount);

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

Error := EXTRA\_DATA;

CloseFile(InputFile);

ReadFileData := Error;

End;

Procedure RecordFileData(Error: ERRORS\_CODE; Var NumsArr: TArr; Var NumsAmount: Integer);

Var

I: Integer;

Begin

If Error = CORRECT Then

Begin

MainForm.NumsAmountEdit.Text := IntToStr(NumsAmount);

For I := 0 To High(NumsArr) Do

MainForm.NumsStringGrid.Cells[I, 1] := IntToStr(NumsArr[I]);

MainForm.ResultButton.Enabled := True;

End

Else

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

End;

Procedure TMainForm.OpenMenuItemClick(Sender: TObject);

Var

Error: ERRORS\_CODE;

InputFile: TextFile;

NumsArr: TArr;

NumsAmount: Integer;

Begin

If OpenTextFileDialog1.Execute Then

Begin

AssignFile(InputFile, OpenTextFileDialog1.FileName);

Error := ReadFileData(InputFile, NumsArr, NumsAmount);

RecordFileData(Error, NumsArr, NumsAmount);

End;

End;

Procedure ClearStringGrid(StringGrid: TStringGrid);

Var

Row, Col: Integer;

Begin

For Row := 0 To StringGrid.RowCount - 1 Do

For Col := 0 To StringGrid.ColCount - 1 Do

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

StringGrid.Enabled := False;

StringGrid.Visible := False;

End;

Function IsFullFields() : Boolean;

Const

NumsRow = 1;

Var

IsFull: Boolean;

Col: Integer;

Begin

IsFull := MainForm.NumsAmountEdit.Text <> '';

Col := 0;

While IsFull And (Col < MainForm.NumsStringGrid.ColCount) Do

Begin

If (MainForm.NumsStringGrid.Cells[Col, NumsRow] = '') Then

IsFull := False;

Inc(Col);

End;

IsFullFields := IsFull;

End;

Function IsPossiblePaste(SelStart, SelLength: Integer; Text: String; Const MIN, MAX: Integer) : 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

((SelStart = 0) And (ClipBoard.AsText[1] <> '0') Or (SelStart > 0)) And

IsValidRange(StrToInt(Copy(Text, 1, SelStart) + ClipBoard.AsText + Copy(Text,

SelStart + SelLength + 1)), MIN, MAX);

End;

Function IsValidChar(SelStart: Integer; Key: Char) : Boolean;

Begin

IsValidChar := (SelStart = 0) And CharInSet(Key, DIGITS\_WITHOUT\_ZERO) Or (SelStart > 0) And

CharInSet(Key, DIGITS);

End;

Procedure DrawNumsStringGrid(NumsStringGrid: TStringGrid; ColCount: Integer);

Const

FixedRow = 0;

Var

I: Integer;

Begin

NumsStringGrid.ColCount := ColCount;

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

NumsStringGrid.Cells[I, FixedRow] := IntToStr(I + 1);

NumsStringGrid.ScrollBars := ssNone;

NumsStringGrid.Width := NumsStringGrid.ColCount \* (NumsStringGrid.DefaultColWidth +

NumsStringGrid.GridLineWidth) + NumsStringGrid.GridLineWidth + 2;

NumsStringGrid.Height := 82; // 2 \* (NumsStringGrid.DefaultRowHeight + NumsStringGrid.GridLineWidth)

+ NumsStringGrid.GridLineWidth

NumsStringGrid.Enabled := True;

NumsStringGrid.Visible := True;

End;

Function AreFormNumsDifferent(NumsStringGrid: TStringGrid) : Boolean;

Const

NumsRow = 1;

Var

IsDifferent: Boolean;

Col: Integer;

Begin

IsDifferent := True;

If NumsStringGrid.Cells[NumsStringGrid.Col, NumsRow] <> '' Then

Begin

Col := 0;

While IsDifferent And (Col < NumsStringGrid.ColCount) Do

Begin

If (Col <> NumsStringGrid.Col) And (NumsStringGrid.Cells[NumsStringGrid.Col, NumsRow] =

NumsStringGrid.Cells[Col, NumsRow]) Then

IsDifferent := False;

Inc(Col);

End;

End;

AreFormNumsDifferent := IsDifferent;

End;

Procedure DeleteIdentical(NumsStringGrid: TStringGrid);

Const

NumsRow = 1;

Var

Col1, Col2: Integer;

IsIdentical: Boolean;

Begin

IsIdentical := False;

Col1 := 0;

Repeat

Col2 := Col1 + 1;

Repeat

If (NumsStringGrid.Cells[Col1, NumsRow] <> '') And (NumsStringGrid.Cells[Col1, NumsRow] =

NumsStringGrid.Cells[Col2, NumsRow]) Then

Begin

IsIdentical := True;

NumsStringGrid.Cells[Col2, NumsRow] := '';

Application.MessageBox('Натуральные числа должны быть различными!', 'Ошибка', MB\_OK Or

MB\_ICONERROR);

End;

Inc(Col2);

Until IsIdentical Or (Col2 >= NumsStringGrid.ColCount);

Inc(Col1);

Until IsIdentical Or (Col1 >= NumsStringGrid.ColCount);

End;

Procedure ComponentChange();

Begin

If MainForm.ResultStringGrid.Cells[0, 0] <> '' Then

ClearStringGrid(MainForm.ResultStringGrid);

IsSaved := True;

MainForm.SaveMenuItem.Enabled := False;

End;

Procedure ComponentKeyDown(Var Key: Word; Shift: TShiftState; SelStart, SelLength: Integer; Text: String; Const MIN, MAX: Integer);

Begin

If (Shift = [ssCtrl]) And (UpCase(Chr(Key)) = 'X') Then

Begin

If (SelLength = 0) And (SelStart = 1) And (Length(Text) > 1) And (Text[2] = '0') Or

(SelLength > 0) And (SelStart = 0) And (SelLength <> Length(Text)) And (Text[SelLength + 1] =

'0') Then

Key := Ord(NONE);

End

Else If Key = VK\_DELETE Then

Begin

If (SelLength = 0) And (SelStart = 0) And (Length(Text) > 1) And (Text[2] = '0') Or

(SelLength > 0) And (SelStart = 0) And (SelLength <> Length(Text)) And (Text[SelLength + 1] =

'0') Then

Key := Ord(NONE);

End

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

Then

Begin

If Not IsPossiblePaste(SelStart, SelLength, Text, MIN, MAX) Then

Key := Ord(NONE);

End;

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

CtrlPressed := True;

End;

Procedure ComponentKeyPress(Var Key: Char; SelStart, SelLength: Integer; Text: String; Const MIN, MAX:

Integer);

Begin

If Key = BACKSPACE Then

Begin

If (SelLength = 0) And (SelStart = 1) And (Length(Text) > 1) And (Text[2] = '0') Or

(SelLength > 0) And (SelStart = 0) And (SelLength <> Length(Text)) And (Text[SelLength + 1] =

'0') Then

Key := NONE;

End

Else If Key = ENTER Then

Begin

If IsFullFields() Then

MainForm.ResultButtonClick(MainForm.ResultButton);

End

Else If Not CtrlPressed Then

Begin

If Not (IsValidChar(SelStart, Key) And IsValidRange(StrToInt(Copy(Text, 1, SelStart) + Key +

Copy(Text, SelStart + SelLength + 1)), MIN, MAX)) Then

Key := NONE;

End;

End;

Procedure ComponentKeyUp();

Begin

CtrlPressed := False;

End;

Procedure TMainForm.NumsAmountEditChange(Sender: TObject);

Begin

ComponentChange();

ResultButton.Enabled := False;

ClearStringGrid(NumsStringGrid);

If NumsAmountEdit.Text = '' Then

Begin

NumsLabel.Enabled := False;

NumsLabel.Visible := False;

End

Else

Begin

NumsLabel.Enabled := True;

NumsLabel.Visible := True;

DrawNumsStringGrid(NumsStringGrid, StrToInt(NumsAmountEdit.Text));

NumsStringGrid.Col := 0;

End;

End;

procedure TMainForm.NumsAmountEditContextPopup(Sender: TObject; MousePos: TPoint; Var Handled: Boolean);

Begin

If Not IsPossiblePaste(NumsAmountEdit.SelStart, NumsAmountEdit.SelLength, NumsAmountEdit.Text,

MIN\_NUMS\_AMOUNT, MAX\_NUMS\_AMOUNT) Or

(NumsAmountEdit.SelLength = 0) And (NumsAmountEdit.SelStart = 1) And (Length(NumsAmountEdit.Text)

> 1) And (NumsAmountEdit.Text[2] = '0') Or

(NumsAmountEdit.SelLength > 0) And (NumsAmountEdit.SelStart = 0) And (NumsAmountEdit.SelLength <>

Length(NumsAmountEdit.Text)) And (NumsAmountEdit.Text[NumsAmountEdit.SelLength + 1] = '0') Then

Handled := True;

End;

Procedure TMainForm.NumsAmountEditKeyDown(Sender: TObject; Var Key: Word; Shift: TShiftState);

Begin

ComponentKeyDown(Key, Shift, NumsAmountEdit.SelStart, NumsAmountEdit.SelLength, NumsAmountEdit.Text,

MIN\_NUMS\_AMOUNT, MAX\_NUMS\_AMOUNT);

End;

Procedure TMainForm.NumsAmountEditKeyPress(Sender: TObject; Var Key: Char);

Begin

ComponentKeyPress(Key, NumsAmountEdit.SelStart, NumsAmountEdit.SelLength, NumsAmountEdit.Text,

MIN\_NUMS\_AMOUNT, MAX\_NUMS\_AMOUNT);

End;

Procedure TMainForm.NumsAmountEditKeyUp(Sender: TObject; Var Key: Word; Shift: TShiftState);

Begin

ComponentKeyUp();

End;

Procedure TMainForm.NumsStringGridExit(Sender: TObject);

Begin

DeleteIdentical(NumsStringGrid);

End;

Procedure TMainForm.NumsStringGridKeyDown(Sender: TObject; Var Key: Word; Shift: TShiftState);

Var

NumsGrid: TStringGridEx;

SelStart, SelLength: Integer;

Text: String;

Begin

NumsGrid := TStringGridEx(Sender);

If Assigned(NumsGrid.InplaceEditor) Then

Begin

SelStart := NumsGrid.InplaceEditor.SelStart;

SelLength := NumsGrid.InplaceEditor.SelLength;

Text := NumsGrid.InplaceEditor.Text;

ComponentKeyDown(Key, Shift, SelStart, SelLength, Text, MIN\_NUM, MAX\_NUM);

End;

End;

Procedure TMainForm.NumsStringGridKeyPress(Sender: TObject; Var Key: Char);

Var

NumsGrid: TStringGridEx;

SelStart, SelLength: Integer;

Text: String;

Begin

NumsGrid := TStringGridEx(Sender);

If Assigned(NumsGrid.InplaceEditor) Then

Begin

SelStart := NumsGrid.InplaceEditor.SelStart;

SelLength := NumsGrid.InplaceEditor.SelLength;

Text := NumsGrid.InplaceEditor.Text;

ComponentKeyPress(Key, SelStart, SelLength, Text, MIN\_NUM, MAX\_NUM);

End;

End;

Procedure TMainForm.NumsStringGridKeyUp(Sender: TObject; Var Key: Word; Shift: TShiftState);

Begin

ComponentKeyUp();

End;

Procedure TMainForm.NumsStringGridSelectCell(Sender: TObject; ACol, ARow: Integer; Var CanSelect: Boolean);

Begin

DeleteIdentical(NumsStringGrid);

End;

Procedure TMainForm.NumsStringGridSetEditText(Sender: TObject; ACol, ARow: Integer; Const Value: String);

Var

NumsGrid: TStringGridEx;

Text: String;

Num: Integer;

Begin

ComponentChange();

ResultButton.Enabled := IsFullFields() And AreFormNumsDifferent(NumsStringGrid);

NumsGrid := TStringGridEx(Sender);

If Assigned(NumsGrid.InplaceEditor) Then

Begin

Text := NumsGrid.InplaceEditor.Text;

If Not (TryStrToInt(Text, Num) And (Text[1] <> '0') And IsValidRange(StrToInt(Text), MIN\_NUM,

MAX\_NUM)) Then

NumsGrid.InplaceEditor.Text := '';

End;

End;

Procedure FillArrayFromStringGrid(NumsStringGrid: TStringGrid; Var NumsArr: TArr);

Const

NumsRow = 1;

Var

I: Integer;

Begin

SetLength(NumsArr, NumsStringGrid.ColCount);

For I := 0 To High(NumsArr) Do

NumsArr[I] := StrToInt(NumsStringGrid.Cells[I, NumsRow]);

End;

Function Factorial(N: Integer) : Integer;

Var

ResFactorial, I: Integer;

Begin

ResFactorial := 1;

For I := 2 To N Do

ResFactorial := ResFactorial \* I;

Factorial := ResFactorial;

End;

Procedure DrawResultStringGrid(ResultStringGrid: TStringGrid);

Const

FixedRow = 0;

Var

Col: Integer;

Begin

ResultStringGrid.ColCount := MainForm.NumsStringGrid.ColCount;

ResultStringGrid.RowCount := Factorial(ResultStringGrid.ColCount) + 1;

For Col := 0 To ResultStringGrid.ColCount - 1 Do

ResultStringGrid.Cells[Col, FixedRow] := IntToStr(Col + 1);

ResultStringGrid.Width := ResultStringGrid.ColCount \* (ResultStringGrid.DefaultColWidth +

ResultStringGrid.GridLineWidth) + ResultStringGrid.GridLineWidth + 2;

If (ResultStringGrid.RowCount > 3) Then

Begin

ResultStringGrid.ScrollBars := ssVertical;

ResultStringGrid.Width := ResultStringGrid.Width + 22;

ResultStringGrid.Height := 120;

End;

End;

Procedure SwapNums(Var Num1, Num2: Integer);

Var

Temp: Integer;

Begin

Temp := Num1;

Num1 := Num2;

Num2 := Temp;

End;

Procedure WriteFormData(NumsArr: TArr; Var PrintingRow: Integer);

Var

I: Integer;

Begin

For I := 0 To High(NumsArr) Do

MainForm.ResultStringGrid.Cells[I, PrintingRow] := IntToStr(NumsArr[I]);

Inc(PrintingRow);

End;

Procedure MakePermutations(NumsArr: TArr; StartIndex: Integer; Var PrintingRow: Integer);

Var

I: Integer;

Begin

If StartIndex = High(NumsArr) Then

WriteFormData(NumsArr, PrintingRow)

Else

For I := StartIndex To High(NumsArr) Do

Begin

SwapNums(NumsArr[I], NumsArr[StartIndex]);

MakePermutations(NumsArr, StartIndex + 1, PrintingRow);

SwapNums(NumsArr[I], NumsArr[StartIndex]);

End;

End;

Procedure TMainForm.ResultButtonClick(Sender: TObject);

Var

NumsArr: TArr;

PrintingRow: Integer;

Begin

FillArrayFromStringGrid(NumsStringGrid, NumsArr);

DrawResultStringGrid(ResultStringGrid);

PrintingRow := 1;

MakePermutations(NumsArr, 0, PrintingRow);

ResultStringGrid.Enabled := True;

ResultStringGrid.Visible := True;

IsSaved := False;

SaveMenuItem.Enabled := True;

End;

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

Var

Row, Col: Integer;

Begin

ReWrite(OutputFile);

WriteLn(OutputFile, 'Перестановки:');

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

Begin

For Col := 0 To ResultStringGrid.ColCount - 1 Do

Write(OutputFile, ResultStringGrid.Cells[Col, Row], ' ');

WriteLn(OutputFile);

End;

CloseFile(OutputFile);

End;

Procedure TMainForm.SaveMenuItemClick(Sender: TObject);

Var

OutputFile: TextFile;

Begin

If SaveTextFileDialog1.Execute Then

Begin

AssignFile(OutputFile, SaveTextFileDialog1.FileName);

WriteFileData(ResultStringGrid, OutputFile);

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 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);

Private

{ Private declarations }

Public

{ Public declarations }

End;

Var

InstructionForm: TInstructionForm;

Implementation

{$R \*.dfm}

Uses MainUnit;

Procedure TInstructionForm.InstructionFormCreate(Sender: TObject);

Begin

InstructionLabel.Width := 600;

InstructionLabel.Caption := '1. Количество чисел должно быть натуральным и лежать в диапазоне [' +

IntToStr(MIN\_NUMS\_AMOUNT) + '; ' + IntToStr(MAX\_NUMS\_AMOUNT) +

'].'#13#10 + '2. Числа должны быть натуральными и различными и лежать в

диапазоне [' + IntToStr(MIN\_NUM) + '; ' + IntToStr(MAX\_NUM) + '].'#13#10

+ '3. Для быстрого открытия инстуркции нажмите F1 .'#13#10 + '4. Файл

должен иметь расширение txt и структуру:'#13#10'Количеством чисел,

которое удовлетворяет первому условию.'#13#10'Числами, которые

удовлетворяют второму условию.';

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

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

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);

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;

End.

**Код программы C#:**

using System;

using System.IO;

namespace Lab42

{

class Program

{

public enum ERRORS\_CODE

{

CORRECT,

INCORRECT\_CHOICE,

INCORRECT\_NUM,

INCORRECT\_RANGE,

IS\_NOT\_DIFFERENT,

IS\_NOT\_TXT,

IS\_NOT\_EXIST,

IS\_NOT\_READABLE,

IS\_NOT\_WRITEABLE,

INCORRECT\_NUMS\_AMOUNT,

EXTRA\_DATA

}

static readonly string[] ERRORS = new string[]

{

"",

"Некорректный выбор!",

"Некорректное число!",

"Значение не попадает в диапазон!",

"Значения не различны!",

"Расширение файла не .txt!",

"Некорректный путь к файлу!",

"Файл закрыт для чтения!",

"Файл закрыт для записи!",

"Неправильное количество чисел в файле!",

"Лишние данные в файле!"

};

const int MIN\_NUM\_AMOUNT = 1,

MAX\_NUM\_AMOUNT = 10,

MIN\_NUM = 1,

MAX\_NUM = 10000;

static void WriteTask()

{

Console.WriteLine("Данная программа делает всевозможные перестановки различных натуральных

чисел.");

}

static void WriteError(ERRORS\_CODE error)

{

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

}

static int ChooseOption(int amount)

{

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 < 1) || (option > amount)))

error = ERRORS\_CODE.INCORRECT\_CHOICE;

if (error != ERRORS\_CODE.CORRECT)

WriteError(error);

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

return option;

}

static ERRORS\_CODE IsValidRange(int num, in int MIN, in int MAX)

{

ERRORS\_CODE error = ERRORS\_CODE.CORRECT;

if ((num < MIN) || (num > MAX))

error = ERRORS\_CODE.INCORRECT\_RANGE;

return error;

}

static ERRORS\_CODE AreNumsDifferent(int[] numsArr, int lastIndex)

{

ERRORS\_CODE error = ERRORS\_CODE.CORRECT;

for (int i = 0; i < lastIndex; i++)

if (numsArr[i] == numsArr[lastIndex])

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

return error;

}

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 StreamReader ReadPathToReadingFile()

{

ERRORS\_CODE error;

string filePath;

do

{

error = ERRORS\_CODE.CORRECT;

filePath = Console.ReadLine();

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

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

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

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

if (error == ERRORS\_CODE.CORRECT)

error = IsReadable(filePath);

if (error != ERRORS\_CODE.CORRECT)

WriteError(error);

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

return new StreamReader(filePath);

}

static ERRORS\_CODE ReadFileNum(StreamReader reader, out int num, in int MIN, in int MAX)

{

ERRORS\_CODE error = ERRORS\_CODE.CORRECT;

string snum = "";

char ch;

while (reader.Peek() != -1 && reader.Peek() != '\r' && reader.Peek() != ' ')

{

ch = Convert.ToChar(reader.Read());

snum = snum + ch;

}

if (reader.Peek() == ' ')

Convert.ToChar(reader.Read());

num = 1;

try

{

num = Int32.Parse(snum);

}

catch

{

error = ERRORS\_CODE.INCORRECT\_NUM;

}

if (error == ERRORS\_CODE.CORRECT)

error = IsValidRange(num, MIN, MAX);

return error;

}

static ERRORS\_CODE ReadFileNumsAmount(StreamReader reader, out int numsAmount)

{

ERRORS\_CODE error;

error = ReadFileNum(reader, out numsAmount, MIN\_NUM\_AMOUNT, MAX\_NUM\_AMOUNT);

if ((error == ERRORS\_CODE.CORRECT) && reader.Peek() != '\r')

error = ERRORS\_CODE.EXTRA\_DATA;

return error;

}

static ERRORS\_CODE ReadFileNums(StreamReader reader, ref int[] numsArr, int numsAmount)

{

ERRORS\_CODE error;

error = ERRORS\_CODE.CORRECT;

numsArr = new int[numsAmount];

int i = 0;

while ((error == ERRORS\_CODE.CORRECT) && (reader.Peek() != -1))

{

if (i < numsAmount)

{

error = ReadFileNum(reader, out numsArr[i], MIN\_NUM, MAX\_NUM);

if (error == ERRORS\_CODE.CORRECT)

error = AreNumsDifferent(numsArr, i);

}

else

error = ERRORS\_CODE.INCORRECT\_NUMS\_AMOUNT;

i++;

}

if ((error == ERRORS\_CODE.CORRECT) && (i != numsAmount))

error = ERRORS\_CODE.INCORRECT\_NUMS\_AMOUNT;

if ((error == ERRORS\_CODE.CORRECT) && !reader.EndOfStream)

error = ERRORS\_CODE.EXTRA\_DATA;

return error;

}

static ERRORS\_CODE ReadFileData(StreamReader reader, ref int[] numsArr)

{

ERRORS\_CODE error;

int numsAmount;

using (reader)

{

error = ReadFileNumsAmount(reader, out numsAmount);

if (error == ERRORS\_CODE.CORRECT)

{

reader.ReadLine();

error = ReadFileNums(reader, ref numsArr, numsAmount);

}

}

return error;

}

static void ReadFile(ref int[] numsArr)

{

StreamReader reader;

ERRORS\_CODE error;

Console.WriteLine("Введите путь к файлу с расширением .txt.");

Console.WriteLine($"Содержимое: \nКоличество чисел целое в диапазоне[{MIN\_NUM\_AMOUNT};

{MAX\_NUM\_AMOUNT}]; \nЧисла различные натуральные в диапазоне[{MIN\_NUM};

{MAX\_NUM}]");

do

{

reader = ReadPathToReadingFile();

using (reader)

{

error = ReadFileData(reader, ref numsArr);

}

if (error != ERRORS\_CODE.CORRECT)

WriteError(error);

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

}

static ERRORS\_CODE ReadConsoleNum(out int num, in int MIN, in int MAX)

{

ERRORS\_CODE error = ERRORS\_CODE.CORRECT;

num = 1;

try

{

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

}

catch

{

error = ERRORS\_CODE.INCORRECT\_NUM;

}

if (error == ERRORS\_CODE.CORRECT)

error = IsValidRange(num, MIN, MAX);

return error;

}

static int ReadConsoleNumsAmount()

{

ERRORS\_CODE error;

int numsAmount;

do

{

Console.WriteLine($"Введите целое количество чисел в диапазоне[{MIN\_NUM\_AMOUNT};

{MAX\_NUM\_AMOUNT}]:");

error = ReadConsoleNum(out numsAmount, MIN\_NUM\_AMOUNT, MAX\_NUM\_AMOUNT);

if (error != ERRORS\_CODE.CORRECT)

WriteError(error);

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

return numsAmount;

}

static void ReadConsoleNums(ref int[] numsArr, int numsAmount)

{

ERRORS\_CODE error;

numsArr = new int[numsAmount];

Console.WriteLine($"Введите числа:");

int i = 0;

do

{

Console.WriteLine($"Введите {i + 1} натуральное уникальное число в диапазоне[{MIN\_NUM};

{MAX\_NUM}]:");

error = ReadConsoleNum(out numsArr[i], MIN\_NUM, MAX\_NUM);

if (error == ERRORS\_CODE.CORRECT)

error = AreNumsDifferent(numsArr, i);

if (error == ERRORS\_CODE.CORRECT)

i++;

else

WriteError(error);

} while (i < numsAmount);

}

static void ReadConsoleData(ref int[] numsArr)

{

int numsAmount;

numsAmount = ReadConsoleNumsAmount();

ReadConsoleNums(ref numsArr, numsAmount);

}

static void ReadConsole(ref int[] numsArr)

{

ReadConsoleData(ref numsArr);

}

static void ReadNums(ref int[] numsArr)

{

Console.WriteLine("Вы хотите: \nВводить числа через файл - 1 \nВводить числа через консоль –

2");

if (ChooseOption(2) == 1)

ReadFile(ref numsArr);

else

ReadConsole(ref numsArr);

}

static void SwapNums(ref int num1, ref int num2)

{

int temp = num1;

num1 = num2;

num2 = temp;

}

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 StreamWriter ReadPathToWritingFile()

{

ERRORS\_CODE error;

string filePath;

StreamWriter writer;

do

{

error = ERRORS\_CODE.CORRECT;

Console.WriteLine("Введите путь к файлу с расширением .txt.");

filePath = Console.ReadLine();

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

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

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

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

if (error == ERRORS\_CODE.CORRECT)

error = IsWriteable(filePath);

if (error != ERRORS\_CODE.CORRECT)

WriteError(error);

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

writer = new StreamWriter(filePath);

return writer;

}

static void WriteFileData(StreamWriter writer, int[] numsArr)

{

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

writer.Write("{0, 5} ", numsArr[i]);

writer.WriteLine();

}

static void MakeFilePermuntations(StreamWriter writer, int[] numsArr, int startIndex)

{

if (startIndex == numsArr.Length - 1)

WriteFileData(writer, numsArr);

else

for (int i = startIndex; i < numsArr.Length; i++)

{

SwapNums(ref numsArr[startIndex], ref numsArr[i]);

MakeFilePermuntations(writer, numsArr, startIndex + 1);

SwapNums(ref numsArr[startIndex], ref numsArr[i]);

}

}

static void WriteFile(int[] numsArr)

{

StreamWriter writer = ReadPathToWritingFile();

using (writer)

{

writer.WriteLine("Перестановки:");

MakeFilePermuntations(writer, numsArr, 0);

}

}

static void WriteConsoleData(int[] numsArr)

{

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

Console.Write("{0, 5} ", numsArr[i]);

Console.WriteLine();

}

static void MakeConsolePermuntations(int[] numsArr, int startIndex)

{

if (startIndex == numsArr.Length - 1)

WriteConsoleData(numsArr);

else

for (int i = startIndex; i < numsArr.Length; i++)

{

SwapNums(ref numsArr[startIndex], ref numsArr[i]);

MakeConsolePermuntations(numsArr, startIndex + 1);

SwapNums(ref numsArr[startIndex], ref numsArr[i]);

}

}

static void WriteConsole(int[] numsArr)

{

Console.WriteLine("Перестановки:");

MakeConsolePermuntations(numsArr, 0);

}

static void MakePermuntations(int[] numsArr)

{

Console.WriteLine("Вы хотите: \nВыводить перестановки через файл - 1 \nВыводить перестановки

через консоль - 2");

if (ChooseOption(2) == 1)

WriteFile(numsArr);

else

WriteConsole(numsArr);

}

static void Main(string[] args)

{

int[] numsArr = null;

WriteTask();

ReadNums(ref numsArr);

MakePermuntations(numsArr);

Console.ReadLine();

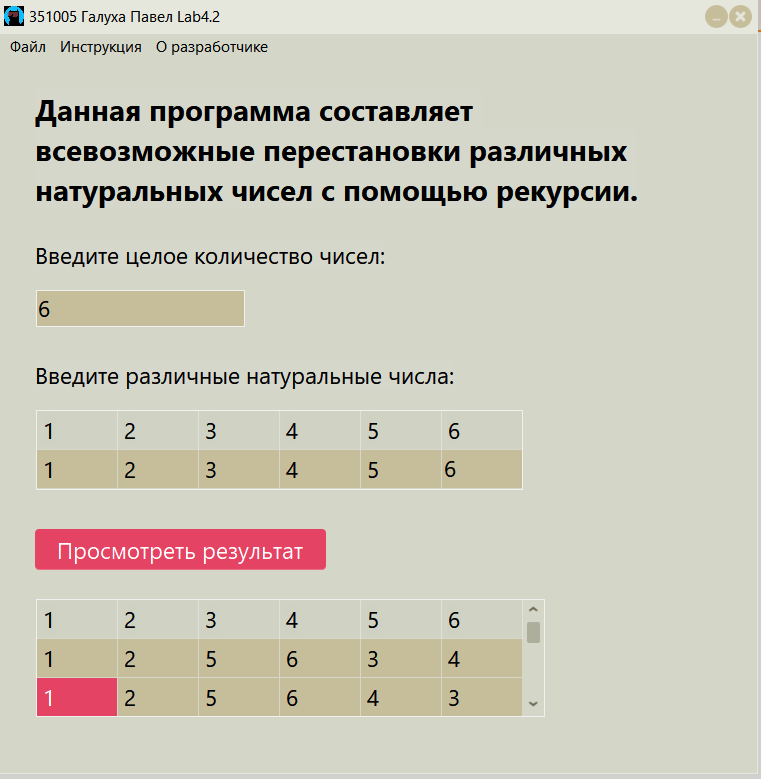
}

}

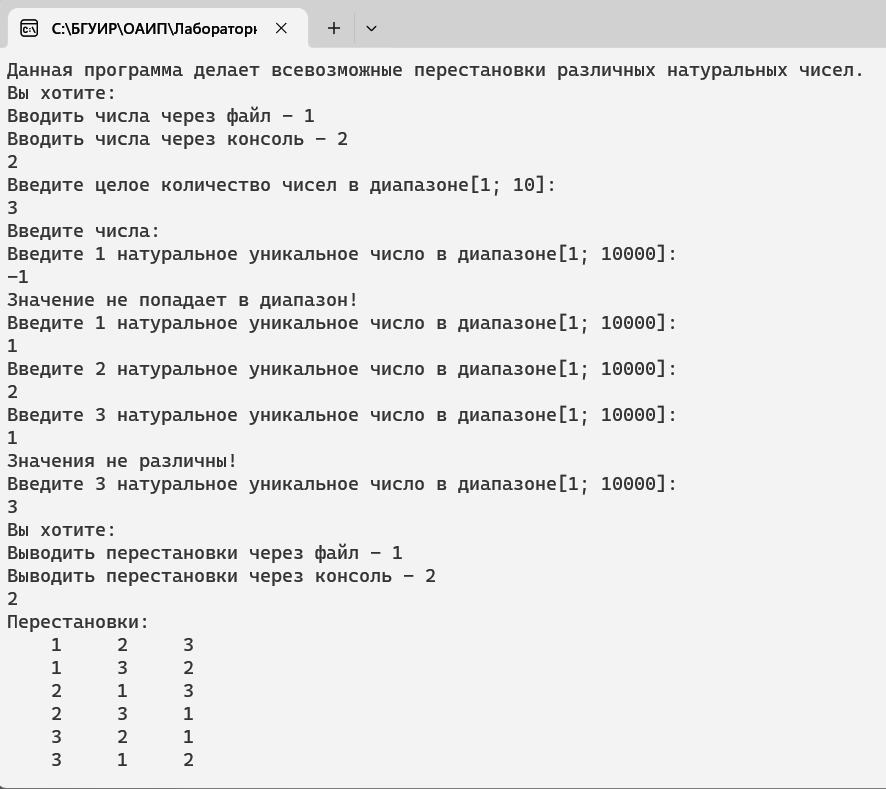
}

**Скриншоты**

**Delphi:**



**C#:**



**Блок-схема**

