fУО «Белорусский государственный университет информатики и радиоэлектроники»

Кафедра ПОИТ

Отчет по лабораторной работе №4.1

по предмету «Основы алгоритмизации и программирования»

Вариант 12

Выполнил:

Галуха П. А.

Гр. 351005

Проверил:

Данилова Г. В.

Минск 2024

**Задание:**

Создать файл, содержащий сведения о репертуаре театра:

* название спектакля;
* автор сценария;
* главный режиссер;
* дата;
* время.

Вывести: Информацию о спектаклях, поставленных данным режиссером в алфавитном порядке названий спектаклей.

**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, EditUnit, SearchUnit,

Vcl.ExtDlgs, System.ImageList, Vcl.ImgList, Vcl.ExtCtrls,

Vcl.Imaging.pngimage;

Type

TMainForm = Class(TForm)

TabsMainMenu: TMainMenu;

FileMenuItem: TMenuItem;

SaveMenuItem: TMenuItem;

SeparatorMenuItem: TMenuItem;

ExitMenuItem: TMenuItem;

InstructionMenuItem: TMenuItem;

DeveloperMenuItem: TMenuItem;

SaveTextFileDialog1: TSaveTextFileDialog;

AddImage: TImage;

DeleteImage: TImage;

EditImage: TImage;

SearchImage: TImage;

SortLabel: TLabel;

DataSortComboBox: TComboBox;

OrderSortComboBox: TComboBox;

DataEdit: TEdit;

ShowsStringGrid: TStringGrid;

Procedure MainFormCreate(Sender: TObject);

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

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

Procedure InstructionMenuItemClick(Sender: TObject);

Procedure DeveloperMenuItemClick(Sender: TObject);

Procedure SaveMenuItemClick(Sender: TObject);

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

Procedure ComboBoxChange(Sender: TObject);

Procedure AddImageClick(Sender: TObject);

Procedure DeleteImageClick(Sender: TObject);

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

Procedure EditImageClick(Sender: TObject);

Procedure ShowsStringGridDblClick(Sender: TObject);

Procedure SearchImageClick(Sender: TObject);

Procedure ExitMenuItemClick(Sender: TObject);

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

Private

{ Private declarations }

Public

{ Public declarations }

End;

Type

TShow = Record

Name: String[32];

Author: String[32];

Director: String[32];

Date: TDate;

Time: TTime;

End;

TShows = Array Of TShow;

TFileShows = File Of TShow;

TDataCriterion = (Name, Author, Director, Date, Time);

TOrderCriterion = (ToHigh, ToLow);

Const

PathToTypedFile = 'TypedFile.txt';

PathToCorrectFile = 'CorrectFile.txt';

MAX\_SHOWS = 100;

Var

MainForm: TMainForm;

IsEdited: Boolean = False;

IsSaved: Boolean = True;

Implementation

{$R \*.dfm}

Procedure DrawShowsStringGrid(ShowsStringGrid: TStringGrid);

Begin

If ShowsStringGrid.RowCount > 8 Then

Begin

ShowsStringGrid.Height := 415; //(ShowsStringGrid.DefaultRowHeight +

ShowsStringGrid.GridLineWidth) \* 8 +

ShowsStringGrid.GridLineWidth

ShowsStringGrid.Width := 995; //(ShowsStringGrid.DefaultColWidth + ShowsStringGrid.GridLineWidth)

\* 5 + ShowsStringGrid.GridLineWidth + 20

End

Else

Begin

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

ShowsStringGrid.RowCount + 4;

ShowsStringGrid.Width := 975; //(ShowsStringGrid.DefaultColWidth + ShowsStringGrid.GridLineWidth)

\* 5 + ShowsStringGrid.GridLineWidth

End;

End;

Procedure ClearShowsStringGrid(ShowsStringGrid: TStringGrid);

Var

Row, Col: Integer;

Begin

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

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

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

ShowsStringGrid.RowCount := 1;

End;

Function IsReadable(Var TypedFile: TFileShows) : Boolean;

Var

IsCorrect: Boolean;

Begin

IsCorrect := True;

Try

Try

Reset(TypedFile);

Finally

CloseFile(TypedFile);

End;

Except

IsCorrect := False;

End;

IsReadable := IsCorrect;

End;

Function IsNormalTypedFile(Var TypedFile: TFileShows) : Boolean;

Var

IsCorrect: Boolean;

Begin

IsCorrect := FileExists(PathToTypedFile);

If IsCorrect Then

IsCorrect := IsReadable(TypedFile);

IsNormalTypedFile := IsCorrect;

End;

Procedure FillShowsStringGrid(Row: Integer; TempShow: TShow);

Begin

MainForm.ShowsStringGrid.Cells[0, Row] := String(TempShow.Name);

MainForm.ShowsStringGrid.Cells[1, Row] := String(TempShow.Author);

MainForm.ShowsStringGrid.Cells[2, Row] := String(TempShow.Director);

MainForm.ShowsStringGrid.Cells[3, Row] := DateToStr(TempShow.Date);

MainForm.ShowsStringGrid.Cells[4, Row] := FormatDateTime('HH:mm', TempShow.Time);

End;

Procedure ReadTypedFile();

Var

TypedFile: TFileShows;

TempShow: TShow;

IsCorrect: Boolean;

CanSelect: Boolean;

Begin

AssignFile(TypedFile, PathToTypedFile);

IsCorrect := IsNormalTypedFile(TypedFile);

MainForm.ShowsStringGrid.RowCount := 1;

If IsCorrect Then

Begin

Reset(TypedFile);

While Not EOF(TypedFile) Do

Begin

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

Try

Read(TypedFile, TempShow);

DateToStr(TempShow.Date);

TimeToStr(TempShow.Time);

Except

IsCorrect := False;

End;

If IsCorrect And (Length(TempShow.Name) <= 32) And (Length(TempShow.Author) <= 32) And

(Length(TempShow.Director) <= 32) Then

FillShowsStringGrid(MainForm.ShowsStringGrid.RowCount - 1, TempShow)

Else

IsCorrect := False;

End;

CloseFile(TypedFile);

End;

If Not IsCorrect Then

Begin

ReWrite(TypedFile);

Close(TypedFile);

ClearShowsStringGrid(MainForm.ShowsStringGrid);

End;

DrawShowsStringGrid(MainForm.ShowsStringGrid);

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

IsSaved := MainForm.ShowsStringGrid.RowCount = 1;

CanSelect := True;

MainForm.ShowsStringGridSellectCell(MainForm.ShowsStringGrid, MainForm.ShowsStringGrid.Col,

MainForm.ShowsStringGrid.Row, CanSelect);

End;

Procedure TMainForm.MainFormCreate(Sender: TObject);

Const

TitleRow = 0;

Begin

ShowsStringGrid.Cells[0, TitleRow] := 'Название';

ShowsStringGrid.Cells[1, TitleRow] := 'Автор';

ShowsStringGrid.Cells[2, TitleRow] := 'Режиссёр';

ShowsStringGrid.Cells[3, TitleRow] := 'Дата';

ShowsStringGrid.Cells[4, TitleRow] := 'Время';

ReadTypedFile();

End;

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

Begin

CallHelp := False;

InstructionMenuItemClick(InstructionMenuItem);

MainFormHelp := False;

End;

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

Begin

If Key = VK\_INSERT Then

AddImageClick(AddImage);

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 WriteFileData(Var OutputFile: TextFile);

Var

TypedFile: TFileShows;

TempShow: TShow;

Begin

AssignFile(TypedFile, PathToTypedFile);

Reset(TypedFile);

ReWrite(OutputFile);

While Not EOF(TypedFile) Do

Begin

Read(TypedFile, TempShow);

WriteLn(OutputFile, 'Название: ', TempShow.Name);

WriteLn(OutputFile, 'Автор: ', TempShow.Author);

WriteLn(OutputFile, 'Режиссёр', TempShow.Director);

WriteLn(OutputFile, 'Дата: ', DateToStr(TempShow.Date));

WriteLn(OutputFile, 'Время: ', FormatDateTime('HH:mm', TempShow.Time));

WriteLn(OutputFile);

End;

CloseFile(OutputFile);

CloseFile(TypedFile);

End;

Procedure TMainForm.SaveMenuItemClick(Sender: TObject);

Var

OutputFile: TextFile;

Begin

If SaveTextFileDialog1.Execute Then

Begin

If ExpandFileName(PathToTypedFile) = SaveTextFileDialog1.FileName Then

Application.MessageBox('Не для тебя растили!', 'Ошибка', MB\_OK + MB\_ICONERROR)

Else

Begin

AssignFile(OutputFile, SaveTextFileDialog1.FileName);

WriteFileData(OutputFile);

IsSaved := True;

End;

End;

End;

Procedure TMainForm.ShowsStringGridSellectCell(Sender: TObject; ACol, ARow: Integer; Var CanSelect:

Boolean);

Begin

If ARow > 0 Then

DataEdit.Text := ShowsStringGrid.Cells[ACol, ARow]

Else

DataEdit.Text := '';

End;

Procedure ReadTypedFileInArray(Var ShowsArr: TShows);

Var

TypedFile: TFileShows;

I: Integer;

Begin

AssignFile(TypedFile, PathToTypedFile);

Reset(TypedFile);

I := 0;

While Not EOF(TypedFile) Do

Begin

SetLength(ShowsArr, Length(ShowsArr) + 1);

Read(TypedFile, ShowsArr[I]);

Inc(I);

End;

CloseFile(TypedFile);

End;

Function CompareShows(Show1, Show2: TShow; DataCriterion: TDataCriterion; OrderCriterion:

TOrderCriterion) : Boolean;

Var

ResCompareShows: Boolean;

Begin

ResCompareShows := False;

Case DataCriterion Of

Name:

ResCompareShows := AnsiUpperCase(String(Show1.Name)) > AnsiUpperCase(String(Show2.Name));

Author:

ResCompareShows := AnsiUpperCase(String(Show1.Author)) > AnsiUpperCase(String(Show2.Author));

Director:

ResCompareShows := AnsiUpperCase(String(Show1.Director)) > AnsiUpperCase(String(Show2.Director));

Date:

ResCompareShows := Show1.Date > Show2.Date;

Time:

ResCompareShows := Show1.Time > Show2.Time;

End;

If OrderCriterion = ToLow Then

ResCompareShows := Not ResCompareShows;

CompareShows := ResCompareShows;

End;

Procedure RecordArrayInTypedFile(ShowsArr: TShows);

Var

TypedFile: TFileShows;

I: Integer;

Begin

AssignFile(TypedFile, PathToTypedFile);

ReWrite(TypedFile);

I := 0;

While I < Length(ShowsArr) Do

Begin

Write(TypedFile, ShowsArr[I]);

Inc(I);

End;

CloseFile(TypedFile);

End;

Procedure SortShows(DataCriterion: TDataCriterion; OrderCriterion: TOrderCriterion);

Var

ShowsArr: TShows;

Key: TShow;

I, J: Integer;

Begin

SetLength(ShowsArr, 0);

ReadTypedFileInArray(ShowsArr);

For I := 1 To High(ShowsArr) Do

Begin

Key := ShowsArr[I];

J := I - 1;

While (J >= 0) And CompareShows(ShowsArr[J], Key, DataCriterion, OrderCriterion) Do

Begin

ShowsArr[J + 1] := ShowsArr[J];

Dec(J);

End;

ShowsArr[J + 1] := Key;

End;

RecordArrayInTypedFile(ShowsArr);

End;

Procedure TMainForm.ComboBoxChange(Sender: TObject);

Begin

SortShows(TDataCriterion(DataSortComboBox.ItemIndex), TOrderCriterion(OrderSortComboBox.ItemIndex));

ReadTypedFile();

End;

Procedure DeleteRecord(CurrentRow: Integer);

Var

TypedFile: TFileShows;

CorrectFile: TFileShows;

TempShow: TShow;

I: Integer;

Begin

AssignFile(TypedFile, PathToTypedFile);

Reset(TypedFile);

AssignFile(CorrectFile, PathToCorrectFile);

ReWrite(CorrectFile);

For I := 1 To CurrentRow - 1 Do

Begin

Read(TypedFile, TempShow);

Write(CorrectFile, TempShow);

End;

Read(TypedFile, TempShow);

While Not EOF(TypedFile) Do

Begin

Read(TypedFile, TempShow);

Write(CorrectFile, TempShow);

End;

CloseFile(CorrectFile);

CloseFile(TypedFile);

DeleteFile(PathToTypedFile);

RenameFile(PathToCorrectFile, PathToTypedFile);

End;

Procedure TMainForm.AddImageClick(Sender: TObject);

Begin

If ShowsStringGrid.RowCount <= MAX\_SHOWS Then

Begin

EditForm := TEditForm.Create(Self);

EditForm.Icon := MainForm.Icon;

EditForm.Caption := 'Добавление спектакля';

EditForm.OkButton.Caption := 'Добавить';

EditForm.OkButton.Enabled := False;

EditForm.ShowModal;

EditForm.Free;

If IsEdited Then

Begin

SortShows(TDataCriterion(DataSortComboBox.ItemIndex),

TOrderCriterion(OrderSortComboBox.ItemIndex));

ReadTypedFile();

IsEdited := False;

End

End

Else

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

End;

Procedure TMainForm.DeleteImageClick(Sender: TObject);

Var

Confirmation: Integer;

Begin

If ShowsStringGrid.Row > 0 Then

Begin

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

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

If Confirmation = IDYES Then

Begin

DeleteRecord(ShowsStringGrid.Row);

ReadTypedFile();

End;

End

Else

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

End;

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

Begin

If Key = VK\_DELETE Then

DeleteImageClick(DeleteImage);

End;

Procedure TMainForm.EditImageClick(Sender: TObject);

Begin

If ShowsStringGrid.Row > 0 Then

Begin

EditForm := TEditForm.Create(Self);

EditForm.Icon := MainForm.Icon;

EditForm.Caption := 'Редактирование спектакля';

EditForm.OkButton.Caption := 'Редактировать';

EditForm.OkButton.Enabled := True;

EditForm.NameEdit.Text := Copy(ShowsStringGrid.Cells[0, ShowsStringGrid.Row], 2,

Length(ShowsStringGrid.Cells[0, ShowsStringGrid.Row]) - 2);

EditForm.AuthorEdit.Text := ShowsStringGrid.Cells[1, ShowsStringGrid.Row];

EditForm.DirectorEdit.Text := ShowsStringGrid.Cells[2, ShowsStringGrid.Row];

EditForm.DatePicker.Date := StrToDate(ShowsStringGrid.Cells[3, ShowsStringGrid.Row]);

EditForm.TimePicker.Time := StrToTime(ShowsStringGrid.Cells[4, ShowsStringGrid.Row]);

EditForm.ShowModal;

EditForm.Free;

If IsEdited Then

Begin

DeleteRecord(ShowsStringGrid.Row);

SortShows(TDataCriterion(DataSortComboBox.ItemIndex),

TOrderCriterion(OrderSortComboBox.ItemIndex));

ReadTypedFile();

IsEdited := False;

End

End

Else

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

End;

Procedure TMainForm.ShowsStringGridDblClick(Sender: TObject);

Begin

EditImageClick(EditImage);

End;

Procedure TMainForm.SearchImageClick(Sender: TObject);

Begin

If ShowsStringGrid.RowCount > 1 Then

Begin

SearchForm := TSearchForm.Create(Self);

SearchForm.Icon := MainForm.Icon;

SearchForm.ShowModal;

SearchForm.Free;

End

Else

Application.MessageBox('Нет редактируемых полей!', 'Ошибка', MB\_OK + MB\_ICONERROR);

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 в .txt файл?',

'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);

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

Private

{ Private declarations }

Public

{ Public declarations }

End;

Var

InstructionForm: TInstructionForm;

Implementation

{$R \*.dfm}

Uses MainUnit;

Procedure TInstructionForm.InstructionFormCreate(Sender: TObject);

Begin

InstructionLabel.Width := 800;

InstructionLabel.Caption := '1. Для добавления спектакля нажмите Ins или 1 кнопку сверху.'#13#10 +

'2. Для удаления спектакля нажмите Del или 2 кнопку сверху.'#13#10 +

'3. Для редактирования спектакля дважды кликните ЛКМ по спектаклю или

нажмите 3 кнопку сверху.'#13#10 +

'4. Для поиска ближайшего спектакля нажмите 4 кнопку сверху.'#13#10 +

'5. Для просмотра содержимого ячейки кликните ЛКМ на неё.'#13#10 +

'6. Для сортировки в выпадающем списке выберите интересующие вас

пункты.'#13#10 +

'7. Максимальное количество спектаклей равно' + IntToStr(MAX\_SHOWS) +

'.';

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.

Unit EditUnit;

Interface

Uses

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

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

Type

TEditForm = Class(TForm)

NameLabel: TLabel;

NameEdit: TEdit;

AuthorLabel: TLabel;

AuthorEdit: TEdit;

DirectorLabel: TLabel;

DirectorEdit: TEdit;

DateLabel: TLabel;

DatePicker: TDateTimePicker;

TimeLabel: TLabel;

TimePicker: TDateTimePicker;

OkButton: TButton;

CancelButton: TButton;

Procedure EditFormCreate(Sender: TObject);

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

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

Procedure ComponentChange(Sender: TObject);

Procedure OkButtonClick(Sender: TObject);

Procedure CloseButtonClick(Sender: TObject);

Private

{ Private declarations }

Public

{ Public declarations }

End;

Var

EditForm: TEditForm;

Implementation

{$R \*.dfm}

Uses MainUnit;

Procedure TEditForm.EditFormCreate(Sender: TObject);

Begin

DatePicker.Date := Now;

TimePicker.Time := Now;

End;

Procedure TEditForm.EditFormKeyDown(Sender: TObject; Var Key: Word; Shift: TShiftState);

Begin

If Key = VK\_ESCAPE Then

Close;

End;

Function IsFullFields() : Boolean;

Begin

IsFullFields := (EditForm.NameEdit.Text <> '') And (EditForm.AuthorEdit.Text <> '') And

(EditForm.DirectorEdit.Text <> '');

End;

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

Const

ENTER = Ord(#13);

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 = ENTER) And OkButton.Enabled Then

OkButtonClick(OkButton);

End;

Procedure TEditForm.ComponentChange(Sender: TObject);

Begin

If IsFullFields() Then

OkButton.Enabled := True

Else

OkButton.Enabled := False;

End;

Procedure TEditForm.OkButtonClick(Sender: TObject);

Var

TypedFile: TFileShows;

TempShow: TShow;

Begin

TempShow.Name := ShortString('"' + NameEdit.Text + '"');

TempShow.Author := ShortString(AuthorEdit.Text);

TempShow.Director := ShortString(DirectorEdit.Text);

TempShow.Date := DatePicker.Date;

TempShow.Time := TimePicker.Time;

AssignFile(TypedFile, PathToTypedFile);

Reset(TypedFile);

Seek(TypedFile, FileSize(TypedFile));

Write(TypedFile, TempShow);

CloseFile(TypedFile);

IsEdited := True;

Close;

End;

Procedure TEditForm.CloseButtonClick(Sender: TObject);

Begin

Close;

End;

End.

Unit SearchUnit;

Interface

Uses

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

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

Type

TSearchForm = Class(TForm)

SearchLabel: TLabel;

SearchDateTimePicker: TDateTimePicker;

SearchButton: TButton;

CancelButton: TButton;

Procedure SearchFormCreate(Sender: TObject);

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

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

Procedure SearchButtonClick(Sender: TObject);

Procedure CloseButtonClick(Sender: TObject);

Private

{ Private declarations }

Public

{ Public declarations }

End;

Var

SearchForm: TSearchForm;

Implementation

{$R \*.dfm}

Uses MainUnit;

Procedure TSearchForm.SearchFormCreate(Sender: TObject);

Begin

SearchDateTimePicker.DateTime := Now;

End;

Procedure TSearchForm.SearchFormKeyDown(Sender: TObject; Var Key: Word; Shift: TShiftState);

Begin

If Key = VK\_ESCAPE Then

Close;

End;

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

Const

ENTER = Ord(#13);

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 = ENTER) Then

SearchButtonClick(SearchButton);

End;

Procedure TSearchForm.SearchButtonClick(Sender: TObject);

Var

TypedFile: TFileShows;

TempShow: TShow;

I, ClosestDateIndex: Integer;

MinDifference, Difference: Double;

FoundDateAfter: Boolean;

Begin

AssignFile(TypedFile, PathToTypedFile);

Reset(TypedFile);

FoundDateAfter := False;

ClosestDateIndex := 0;

MinDifference := MaxInt;

I := 1;

While Not EOF(TypedFile) Do

Begin

Read(TypedFile, TempShow);

Difference := TempShow.Date - SearchDateTimePicker.Date + TempShow.Time –

SearchDateTimePicker.Time;

If Difference >= 0 Then

Begin

If Difference < MinDifference Then

Begin

MinDifference := Difference;

ClosestDateIndex := I;

FoundDateAfter := True;

End;

End;

Inc(I);

End;

CloseFile(TypedFile);

If FoundDateAfter Then

Begin

MainForm.ShowsStringGrid.Row := ClosestDateIndex;

Application.MessageBox('Ваш спектакль выделен!', 'Подсказка', MB\_OK + MB\_ICONINFORMATION);

Close;

End

Else

Application.MessageBox('Увы ближайших спектаклей не предвидится!', 'Сорри', MB\_OK + MB\_ICONHAND);

End;

Procedure TSearchForm.CloseButtonClick(Sender: TObject);

Begin

Close;

End;

End.

**Код программы C#:**

using System;

using System.Collections.Generic;

using System.Globalization;

using System.IO;

using static Lab41.Program;

using static System.Collections.Specialized.BitVector32;

namespace Lab41

{

internal class Program

{

public enum ERRORS\_CODE

{

CORRECT,

INCORRECT\_CHOICE,

INCORRECT\_NAME,

INCORRECT\_AUTHOR,

INCORRECT\_DIRECTOR,

INCORRECT\_DATE,

INCORRECT\_TIME,

INCORRECT\_DATE\_TIME,

IS\_NOT\_TXT,

IS\_NOT\_EXIST,

IS\_NOT\_READABLE,

IS\_NOT\_WRITEABLE,

EXTRA\_DATA

}

static readonly string[] ERRORS = new string[]

{

"",

"Incorrect choice!",

"Incorrect name!",

"Incorrect author name!",

"Incorrect director name!",

"Incorrect date!",

"Incorrect time!",

"Incorrect date and time!",

"The file extension is not .txt!",

"Incorrect file path!",

"The file is closed for reading!",

"The file is closed for writing",

"Extra data in the file!"

};

public struct Show

{

public int ShowNumber;

public string ShowName;

public string ShowAuthor;

public string ShowDirector;

public DateTime ShowDate;

public DateTime ShowTime;

}

public enum Actions

{

Exit,

AddShow,

DeleteShow,

EditShow,

SearchShow,

WriteFile,

ShowShows,

}

static void WriteError(ERRORS\_CODE error)

{

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

}

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);

Console.Write("Try again: ");

}

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

return option;

}

static ERRORS\_CODE ReadStringTypeShow(string message, ref string tempString, ERRORS\_CODE

probableError)

{

const int MAX\_STRING\_LEN = 20;

ERRORS\_CODE error = ERRORS\_CODE.CORRECT;

if (tempString == "")

{

Console.Write(message + $" [1..{MAX\_STRING\_LEN}]: ");

tempString = Console.ReadLine();

if (tempString == "" || tempString.Length > MAX\_STRING\_LEN)

{

error = probableError;

tempString = "";

}

}

return error;

}

static ERRORS\_CODE ReadDateTimeTypeShow(string message, ref DateTime tempDateTime, string format,

ERRORS\_CODE probableError)

{

ERRORS\_CODE error = ERRORS\_CODE.CORRECT;

if (tempDateTime == DateTime.MinValue)

{

Console.Write(message);

try

{

tempDateTime = DateTime.ParseExact(Console.ReadLine(), format,

CultureInfo.InvariantCulture);

}

catch

{

error = ERRORS\_CODE.INCORRECT\_DATE;

}

}

return error;

}

static Show ReadShow()

{

ERRORS\_CODE error;

Show show = new Show { ShowName = "", ShowAuthor = "", ShowDirector = "", ShowDate =

DateTime.MinValue, ShowTime = DateTime.MinValue };

do

{

error = ReadStringTypeShow("Enter the title", ref show.ShowName,

ERRORS\_CODE.INCORRECT\_NAME);

if (error == ERRORS\_CODE.CORRECT)

error = ReadStringTypeShow("Enter the author's name", ref show.ShowAuthor,

ERRORS\_CODE.INCORRECT\_AUTHOR);

if (error == ERRORS\_CODE.CORRECT)

error = ReadStringTypeShow("Enter director's name", ref show.ShowDirector,

ERRORS\_CODE.INCORRECT\_DIRECTOR);

if (error == ERRORS\_CODE.CORRECT)

error = ReadDateTimeTypeShow("Enter date(dd.mm.yyyy): ", ref show.ShowDate,

"dd.MM.yyyy", ERRORS\_CODE.INCORRECT\_DATE);

if (error == ERRORS\_CODE.CORRECT)

error = ReadDateTimeTypeShow("Enter time (hh:mm): ", ref show.ShowTime, "HH:mm",

ERRORS\_CODE.INCORRECT\_TIME);

if (error != ERRORS\_CODE.CORRECT)

WriteError(error);

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

return show;

}

static void SortShows(List<Show> showsList)

{

Show tempShow1;

int j;

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

{

tempShow1 = showsList[i];

j = i - 1;

while (j >= 0 && showsList[j].ShowName.CompareTo(tempShow1.ShowName) > 0)

{

showsList[j + 1] = showsList[j];

j--;

}

showsList[j + 1] = tempShow1;

}

}

static void FillShowsNumber(List<Show> showsList)

{

Show tempShow;

for (int i = 0; i < showsList.Count; i++)

{

tempShow = showsList[i];

tempShow.ShowNumber = i + 1;

showsList[i] = tempShow;

}

}

static DateTime ReadСlosestShowDateTime()

{

ERRORS\_CODE error;

DateTime closestShowDateTime = DateTime.MinValue;

do

{

error = ERRORS\_CODE.CORRECT;

try

{

closestShowDateTime = DateTime.ParseExact(Console.ReadLine(), "dd.MM.yyyy HH:mm",

CultureInfo.InvariantCulture);

}

catch

{

error = ERRORS\_CODE.INCORRECT\_DATE\_TIME;

}

if (error != ERRORS\_CODE.CORRECT)

{

WriteError(error);

Console.Write("Try again: ");

}

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

return closestShowDateTime;

}

static int FindNumberOfClosestShowDateTime(List<Show> showsList)

{

int numberOfClosestShowDateTime = -1;

TimeSpan difference, minDifference = TimeSpan.MaxValue;

DateTime closestShowDateTime = ReadСlosestShowDateTime();

for (int i = 0; i < showsList.Count; i++)

{

difference = showsList[i].ShowDate.Date + showsList[i].ShowTime.TimeOfDay –

closestShowDateTime;

if (difference >= TimeSpan.FromMinutes(0) && difference < minDifference)

{

minDifference = difference;

numberOfClosestShowDateTime = i;

}

}

return numberOfClosestShowDateTime;

}

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("Write path to file: ");

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)

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 WriteTable(StreamWriter writer, List<Show> showsList)

{

Show show;

using (writer)

{

writer.WriteLine("Shows:");

for (int j = 0; j < 90; j++) writer.Write("-");

writer.WriteLine("\r\n| {0, 2}| {1, 20}| {2, 20}| {3, 20}| {4, 10}| {5, 5}|", "#",

"Name", "Author", "Director", "Date", "Time");

for (int i = 0; i < showsList.Count; i++)

{

for (int j = 0; j < 90; j++) writer.Write("-");

show = showsList[i];

writer.WriteLine("\r\n| {0, 2}| {1, 20}| {2, 20}| {3, 20}| {4, 10}| {5, 5}|",

show.ShowNumber, show.ShowName, show.ShowAuthor, show.ShowDirector,

show.ShowDate.ToString("dd.MM.yyyy"),

show.ShowTime.ToString("HH:mm"));

}

for (int j = 0; j < 90; j++) writer.Write("-");

writer.WriteLine();

}

}

static void WriteCFData(char cfMode, List<Show> showsList)

{

StreamWriter writer = null;

if (showsList.Count == 0)

Console.WriteLine("You haven't added any shows yet!");

else

{

switch (cfMode)

{

case 'c':

writer = new StreamWriter(Console.OpenStandardOutput());

break;

case 'f':

writer = new StreamWriter(ReadIOFilePath('o'));

break;

}

WriteTable(writer, showsList);

}

}

static void AddShow(List<Show> showsList, ref bool isSaved)

{

Show tempShow;

tempShow = ReadShow();

tempShow.ShowNumber = showsList.Count + 1;

showsList.Add(tempShow);

SortShows(showsList);

FillShowsNumber(showsList);

isSaved = false;

}

static void DeleteShow(List<Show> showsList, ref bool isSaved)

{

int numberOfDeletedShow;

if (showsList.Count == 0)

Console.WriteLine("You haven't added any shows yet!");

else

{

WriteCFData('c', showsList);

Console.Write("Enter the number of the show to be deleted: ");

numberOfDeletedShow = ChooseOptionWithinRange(1, showsList.Count);

Console.Write("Are you sure you want to delete the show?:\r\n" +

"1 - yes\r\n" +

"2 - no\r\n" +

"Your choice: ");

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

{

showsList.Remove(showsList[numberOfDeletedShow - 1]);

FillShowsNumber(showsList);

isSaved = showsList.Count == 0;

}

}

}

static void EditShow(List<Show> showsList, ref bool isSaved)

{

int numberOfEditedShow;

if (showsList.Count == 0)

Console.WriteLine("You haven't added any shows yet!");

else

{

WriteCFData('c', showsList);

Console.Write("Enter the number of the show to be edited: ");

numberOfEditedShow = ChooseOptionWithinRange(1, showsList.Count);

showsList[numberOfEditedShow] = ReadShow();

SortShows(showsList);

FillShowsNumber(showsList);

isSaved = false;

}

}

static void SearchShow(List<Show> showsList)

{

int numberOfClosestShowDateTime;

if (showsList.Count == 0)

Console.WriteLine("You haven't added any shows yet!");

else

{

Console.Write("Enter the date and time of the show you are interested in (dd.mm.yyyy

hh:mm): ");

numberOfClosestShowDateTime = FindNumberOfClosestShowDateTime(showsList);

if (numberOfClosestShowDateTime == -1)

Console.WriteLine("Alas, there are no upcoming show(");

else

Console.WriteLine($"Name: {showsList[numberOfClosestShowDateTime].ShowName}\r\n" +

$"Author: {showsList[numberOfClosestShowDateTime].ShowAuthor}\r\n"

+ $"Director:{showsList[numberOfClosestShowDateTime].ShowDirector}\r\n " +

$"Date: {showsList[numberOfClosestShowDateTime].ShowDate.Date}\r\n"

+$"Time:{showsList[numberOfClosestShowDateTime].ShowTime.TimeOfDay}");

}

}

static void WriteFile(List<Show> showsList, ref bool isSaved)

{

WriteCFData('f', showsList);

isSaved = true;

}

static void ShowShows(List<Show> showsList)

{

WriteCFData('c', showsList);

}

static void Exit(List<Show> showsList, bool isSaved)

{

if (!isSaved)

{

Console.Write("You didn't save the file, do you want to save the file before

exiting?\r\n" +

"1 - yes\r\n" +

"2 - no\r\n" +

"Your choice: ");

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

WriteFile(showsList, ref isSaved);

}

}

static void ChooseAction(List<Show> showsList)

{

Actions action;

bool isSaved = false;

do

{

Console.Write("Choose one of the following actions:\r\n" +

"1 - add show(max 100)\r\n" +

"2 - delete show\r\n" +

"3 - edit show\r\n" +

"4 - search closest show\r\n" +

"5 - save file\r\n" +

"6 - show shows\r\n" +

"0 - exit\r\n" +

"Your choice: ");

action = (Actions)ChooseOptionWithinRange(0, Enum.GetValues(typeof(Actions)).Length - 1);

switch (action)

{

case Actions.AddShow:

AddShow(showsList, ref isSaved);

break;

case Actions.DeleteShow:

DeleteShow(showsList, ref isSaved);

break;

case Actions.EditShow:

EditShow(showsList, ref isSaved);

break;

case Actions.SearchShow:

SearchShow(showsList);

break;

case Actions.WriteFile:

WriteFile(showsList, ref isSaved);

break;

case Actions.ShowShows:

ShowShows(showsList);

break;

case Actions.Exit:

Exit(showsList, isSaved);

break;

}

} while (action != Actions.Exit);

}

static void Main(string[] args)

{

List<Show> showsList = new List<Show>();

ChooseAction(showsList);

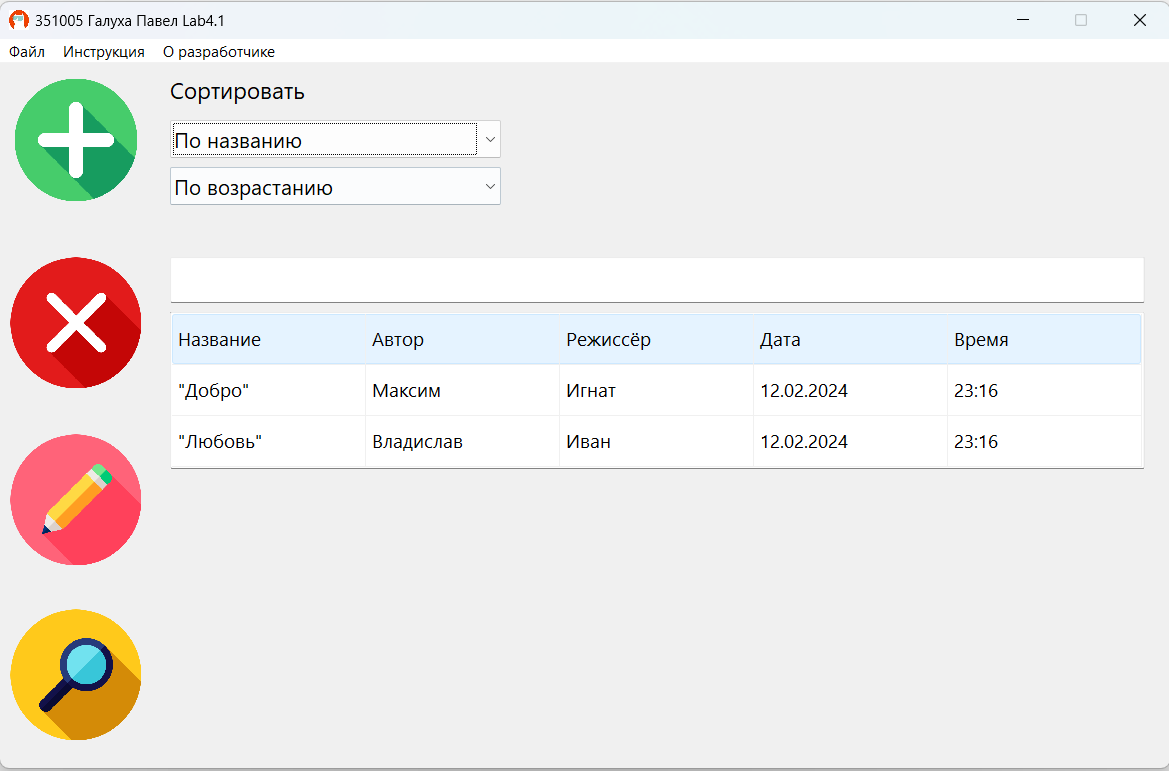
}

}

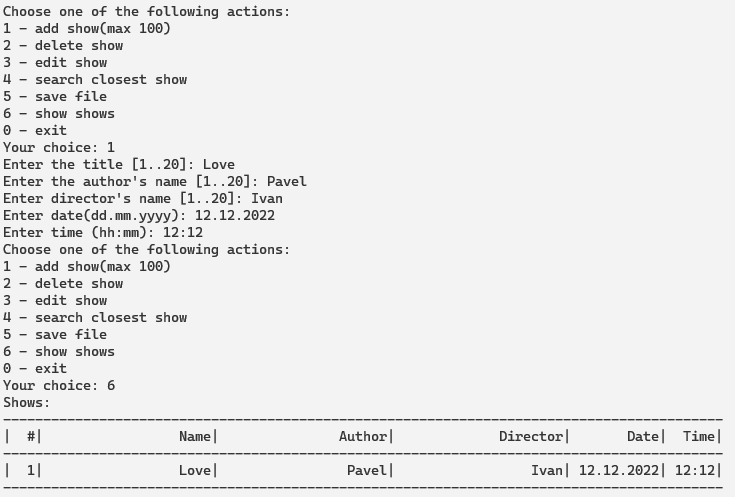
}

**Скриншоты**

**Delphi:**

****

**C#:**



**Блок-схема**

