УО «Белорусский государственный университет информатики и радиоэлектроники»

Кафедра ПОИТ

Отчет по лабораторной работе №4.1

по предмету «Основы алгоритмизации и программирования»

Вариант 15

Выполнила:

Городко К. Е.

Гр. 351005

Проверила:

Данилова Г. В.

Минск 2024

**ЗАДАНИЕ:**

Сведения об автомобилях:

- марка;

- государство производитель;

- год выпуска;

- цена.

Вывести: информацию об автомобилях выпуска прошлого года в порядке убывания цен.

**КОД ПРОГРАММЫ DELPHI:**

Unit UnitMain;

Interface

Uses

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

System.Classes, Vcl.Graphics, Vcl.Controls, Vcl.Forms, Vcl.Dialogs, Vcl.StdCtrls,

Vcl.Menus, Vcl.ExtDlgs, Vcl.ExtCtrls, Vcl.ComCtrls, Vcl.Imaging.Pngimage,

UnitRecord, UnitInstruction, UnitTask;

Const

MINYEAR = 1900;

MAXYEAR = 2024;

MINPRICE = 0;

MAXPRICE = 999999999;

MAXLENGTH = 20;

LASTYEAR = 2023;

MAXRECORDS = 30;

Type

TCar = Record

Model: String[MAXLENGTH];

Country: String[MAXLENGTH];

Year: MINYEAR .. MAXYEAR;

Price: MINPRICE .. MAXPRICE;

End;

TCarList = Array Of TCar;

TCarListFile = File Of TCar;

TMainForm = Class(TForm)

MainMenu: TMainMenu;

MMFile: TMenuItem;

MMOpenFile: TMenuItem;

MMExit: TMenuItem;

MMSeparator: TMenuItem;

MMInstruction: TMenuItem;

MMDevInfo: TMenuItem;

OpenDialog: TOpenDialog;

LabelDescription: TLabel;

SaveDialog: TSaveDialog;

MMSaveFile: TMenuItem;

ButtonAddRecord: TButton;

ButtonDeleteRecord: TButton;

ButtonDoTask: TButton;

MainListView: TListView;

ImageGarage: TImage;

Procedure AddItemToListView(ListView: TListView; List: TCarList);

Procedure ChangeListViewItem(ListView: TListView; List: TCarList);

Procedure DeleteFromList(DeleteIndex: Integer; Var List: TCarList;

ListView: TListView);

Procedure ClearList(ListView: TListView; Var List: TCarList);

Procedure MainListViewChange(Sender: TObject; Item: TListItem; Change:

TItemChange);

Procedure MainListViewDblClick(Sender: TObject);

Procedure MainListViewSelectItem(Sender: TObject; Item: TListItem;

Selected: Boolean);

Procedure OnCloseQuery(List: TCarList; Var IsFileSaved, CanClose: Boolean);

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

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

Procedure ButtonAddRecordClick(Sender: TObject);

Procedure ButtonDeleteRecordClick(Sender: TObject);

Procedure ButtonDoTaskClick(Sender: TObject);

Procedure SaveFile(List: TCarList; Var IsFileSaved: Boolean);

Procedure MMOpenFileClick(Sender: TObject);

Procedure MMSaveFileClick(Sender: TObject);

Procedure MMExitClick(Sender: TObject);

Procedure MMInstructionClick(Sender: TObject);

Procedure MMDevInfoClick(Sender: TObject);

Procedure FormResize(Sender: TObject);

End;

Var

MainForm: TMainForm;

MainList: TCarList;

SelectedIndex: Integer;

Implementation

{$R \*.dfm}

Var

IsSaved: Boolean;

Procedure TMainForm.AddItemToListView(ListView: TListView; List: TCarList);

Var

NewItem: TListItem;

Index: Integer;

Begin

NewItem := ListView.Items.Add;

NewItem.Caption := IntToStr(ListView.Items.Count);

Index := ListView.Items.Count - 1;

With NewItem.SubItems Do

Begin

Add(List[Index].Model);

Add(List[Index].Country);

Add(IntToStr(List[Index].Year));

Add(IntToStr(List[Index].Price));

End;

End;

Procedure TMainForm.ChangeListViewItem(ListView: TListView; List: TCarList);

Begin

With ListView.Items[SelectedIndex] Do

Begin

SubItems[0] := List[SelectedIndex].Model;

SubItems[1] := List[SelectedIndex].Country;

SubItems[2] := IntToStr(List[SelectedIndex].Year);

SubItems[3] := IntToStr(List[SelectedIndex].Price);

End;

End;

Procedure TMainForm.DeleteFromList(DeleteIndex: Integer; Var List: TCarList;

ListView: TListView);

Var

I: Integer;

Begin

For I := DeleteIndex + 1 To High(List) Do

List[I - 1] := List[I];

SetLength(List, Length(List) - 1);

For I := DeleteIndex To High(List) Do

ListView.Items[I].Caption := IntToStr(I + 1);

End;

Procedure TMainForm.ClearList(ListView: TListView; Var List: TCarList);

Var

I: Integer;

Begin

For I := ListView.Items.Count - 1 DownTo 0 Do

ListView.Items[I].Delete;

List := Nil;

End;

Procedure TMainForm.MainListViewChange(Sender: TObject; Item: TListItem;

Change: TItemChange);

Begin

MMSaveFile.Enabled := MainListView.Items.Count > 0;

ButtonAddRecord.Enabled := MainListView.Items.Count < MAXRECORDS;

IsSaved := False;

End;

Procedure TMainForm.MainListViewDblClick(Sender: TObject);

Begin

SelectedIndex := StrToInt(MainListView.Selected.Caption) - 1;

IsRecordChanging := True;

FormRecord.ShowModal;

End;

Procedure TMainForm.MainListViewSelectItem(Sender: TObject; Item: TListItem;

Selected: Boolean);

Begin

ButtonDeleteRecord.Enabled := Selected;

End;

Procedure TMainForm.ButtonAddRecordClick(Sender: TObject);

Begin

IsRecordChanging := False;

FormRecord.ShowModal;

End;

Procedure TMainForm.ButtonDeleteRecordClick(Sender: TObject);

Var

DeleteIndex, ButtonSelected: Integer;

Begin

ButtonSelected := Application.MessageBox('Вы уверены, что хотите удалить

выделенную запись?', 'Удаление записи',

MB\_YESNO + MB\_ICONQUESTION);

If ButtonSelected = MrYes Then

Begin

DeleteIndex := StrToInt(MainListView.Selected.Caption) - 1;

MainListView.Selected.Delete;

DeleteFromList(DeleteIndex, MainList, MainListView);

MMSaveFile.Enabled := MainListView.Items.Count > 0;

End

Else

MainListView.ClearSelection;

End;

Function IsLastYearInList(List: TCarList): Boolean;

Var

I: Integer;

IsLastYear: Boolean;

Begin

IsLastYear := False;

I := 0;

While (I < Length(List)) And Not IsLastYear Do

Begin

If List[I].Year = LASTYEAR Then

IsLastYear := True;

Inc(I);

End;

IsLastYearInList := IsLastYear;

End;

Procedure TMainForm.ButtonDoTaskClick(Sender: TObject);

Begin

If IsLastYearInList(MainList) Then

FormTask.ShowModal

Else

Application.MessageBox('Вывести информацию об автомобилях выпуска прошлого

года в порядке убывания цен.'#13#10#13#10'В списке

нет автомобилей, выпущенных в прошлом году.',

'Задание', MB\_OK);

End;

Procedure TMainForm.OnCloseQuery(List: TCarList; Var IsFileSaved, CanClose: Boolean);

Var

ButtonSelected: Integer;

Begin

If MMSaveFile.Enabled And Not IsFileSaved Then

Begin

ButtonSelected := Application.MessageBox('Желаете сохранить список в файл?',

'Выход', MB\_YESNOCANCEL +

MB\_ICONQUESTION);

If ButtonSelected = MrYes Then

Begin

SaveFile(List, IsFileSaved);

If Not IsFileSaved Then

OnCloseQuery(List, IsFileSaved, CanClose);

End

Else If ButtonSelected = MrNo Then

CanClose := True

Else

CanClose := False;

End

Else

Begin

ButtonSelected := Application.MessageBox('Вы уверены, что хотите выйти?',

'Выход',MB\_YESNO + MB\_ICONQUESTION);

If ButtonSelected = MrYes Then

CanClose := True

Else

CanClose := False;

End;

End;

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

Begin

OnCloseQuery(MainList, IsSaved, CanClose);

End;

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

Begin

If Key = VK\_F1 Then

FormInstruction.ShowModal

Else If ButtonDeleteRecord.Enabled And (Key = VK\_DELETE) Then

ButtonDeleteRecord.Click;

End;

Procedure TMainForm.FormResize(Sender: TObject);

Begin

Left := (Screen.Width - Width) Div 2;

Top := (Screen.Height - Height) Div 2;

End;

Function IsFilePathCorrect(Path: String): Boolean;

Var

IsCorrect: Boolean;

Begin

IsCorrect := True;

If ExtractFileExt(Path) <> '.carlist' Then

Begin

Application.MessageBox('Файл должен иметь разрешение .carlist!', 'Ошибка',

MB\_OK);

IsCorrect := False;

End;

IsFilePathCorrect := IsCorrect;

End;

Procedure TMainForm.SaveFile(List: TCarList; Var IsFileSaved: Boolean);

Var

I: Integer;

IsCorrect: Boolean;

FOut: TCarListFile;

Begin

IsCorrect := SaveDialog.Execute And IsFilePathCorrect(SaveDialog.FileName);

If IsCorrect Then

Try

Try

IsCorrect := True;

AssignFile(FOut, SaveDialog.FileName);

Rewrite(FOut);

For I := 0 To High(List) Do

Write(FOut, List[I]);

Finally

CloseFile(FOut);

End;

Except

IsCorrect := False;

Application.MessageBox('Произошла ошибка при записи в файл!','Ошибка',

MB\_ICONERROR);

End;

IsFileSaved := IsCorrect;

End;

Procedure TMainForm.MMOpenFileClick(Sender: TObject);

Var

I: Integer;

IsCorrect: Boolean;

TempList: TCarList;

FIn: TCarListFile;

Begin

IsCorrect := OpenDialog.Execute And IsFilePathCorrect(OpenDialog.FileName);

If IsCorrect Then

Try

Try

IsCorrect := True;

AssignFile(FIn, OpenDialog.FileName);

Reset(FIn);

SetLength(TempList, FileSize(FIn));

For I := 0 To High(TempList) Do

Read(FIn, TempList[I]);

Finally

CloseFile(FIn);

End;

Except

IsCorrect := False;

Application.MessageBox('Файл недоступен для чтения!', 'Ошибка',

MB\_ICONERROR);

End;

If IsCorrect Then

Begin

ClearList(MainListView, MainList);

MainList := Copy(TempList);

For I := 0 To High(MainList) Do

AddItemToListView(MainListView, MainList);

End;

IsSaved := IsCorrect;

End;

Procedure TMainForm.MMSaveFileClick(Sender: TObject);

Begin

SaveFile(MainList, IsSaved);

End;

Procedure TMainForm.MMExitClick(Sender: TObject);

Begin

MainForm.Close;

End;

Procedure TMainForm.MMInstructionClick(Sender: TObject);

Begin

FormInstruction.ShowModal;

End;

Procedure TMainForm.MMDevInfoClick(Sender: TObject);

Begin

Application.MessageBox('Городко Ксения Евгеньвна'#13#10'351005 группа',

'О разработчике', MB\_OK);

End;

End.

Unit UnitRecord;

Interface

Uses

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

System.Classes, Vcl.Graphics, Vcl.Controls, Vcl.Forms, Vcl.Dialogs, Vcl.Menus,

Vcl.StdCtrls, UnitInstruction;

Type

TFormRecord = Class(TForm)

MainMenu: TMainMenu;

MMInstruction: TMenuItem;

LabelModel: TLabel;

EditModel: TEdit;

LabelCountry: TLabel;

EditCountry: TEdit;

LabelYear: TLabel;

EditYear: TEdit;

LabelPrice: TLabel;

EditPrice: TEdit;

ButtonSave: TButton;

ButtonCancel: TButton;

PopupMenu: TPopupMenu;

Procedure MMInstructionClick(Sender: TObject);

Procedure EditOnChange(Sender: TObject);

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

TShiftState);

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

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

TShiftState);

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

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

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

Procedure EditYearExit(Sender: TObject);

Procedure EditPriceKeyDown(Sender: TObject; Var Key: Word;

Shift: TShiftState);

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

Procedure EditPriceExit(Sender: TObject);

Procedure ClearEdits();

Procedure ChangeRecord(Index: Integer);

Procedure ButtonCancelClick(Sender: TObject);

Procedure ButtonSaveClick(Sender: TObject);

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

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

Procedure FormShow(Sender: TObject);

Procedure FormClose(Sender: TObject; Var Action: TCloseAction);

End;

Var

FormRecord: TFormRecord;

IsRecordChanging: Boolean;

Implementation

{$R \*.dfm}

Uses UnitMain;

Const

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

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

SPACE = ' ';

HYPHEN = '-';

NULL = #0;

BACKSPACE = #8;

Var

IsChanged: Boolean;

Procedure TFormRecord.MMInstructionClick(Sender: TObject);

Begin

FormInstruction.ShowModal;

End;

Function IsNumCorrect(NumStr: String; Const MINNUM, MAXNUM: Integer): Boolean;

Var

Num: Integer;

IsCorrect: Boolean;

Begin

IsCorrect := True;

If Not TryStrToInt(NumStr, Num) Then

Begin

IsCorrect := False;

Application.MessageBox('Значения года и цены должны быть целыми числами!',

'Ошибка', MB\_OK + MB\_ICONERROR);

End

Else If ((Num > MAXNUM) Or (Num < MINNUM)) Then

Begin

IsCorrect := False;

Application.MessageBox('Значение года и/или цены не попадает в диапазон!',

'Ошибка', MB\_OK + MB\_ICONERROR);

End;

IsNumCorrect := IsCorrect;

End;

Procedure NumEditKeyPress(SelStart, Len: Integer; EditText: String; Var Key: Char);

Begin

If Not CharInSet(Key, NUMBERS) And (Key <> BACKSPACE) Then

Key := NULL

Else If (SelStart = 0) And (Key = '0') And (Len > 0) Then

Key := NULL

Else If (EditText = '0') And (SelStart = Len) And (Key <> BACKSPACE) Then

Key := NULL;

End;

Procedure StringEditKeyPress(Edit: TEdit; Var Key: Char);

Var

Len, SelStart: Integer;

EditText: String;

Begin

EditText := Edit.Text;

Len := Length(EditText);

SelStart := Edit.SelStart;

If (Len = MAXLENGTH) And (Key <> BACKSPACE) Then

Key := NULL

Else If ((Key < 'А') Or (Key > 'я')) And Not CharInSet(Key, [SPACE, HYPHEN,

BACKSPACE]) And Not CharInSet(Key, LATINLETTERS) Then

Key := NULL

Else If (SelStart = 0) And CharInSet(Key, [SPACE, HYPHEN]) Then

Key := NULL

Else If CharInSet(Key, [SPACE, HYPHEN]) And (Len > 0) And ((SelStart < Len) And

CharInSet(EditText[SelStart + 1], [SPACE, HYPHEN]) Or

CharInSet(EditText[SelStart], [SPACE, HYPHEN])) Then

Key := NULL;

End;

Procedure TFormRecord.EditCountryKeyDown(Sender: TObject; Var Key: Word; Shift:

TShiftState);

Begin

If Key = VK\_DOWN Then

EditYear.SetFocus

Else If Key = VK\_UP Then

EditModel.SetFocus

Else If Key = VK\_CONTROL Then

Key := Ord(NULL);

End;

Procedure TFormRecord.EditCountryKeyPress(Sender: TObject; Var Key: Char);

Begin

StringEditKeyPress(EditCountry, Key);

End;

Procedure TFormRecord.EditModelKeyDown(Sender: TObject; Var Key: Word; Shift:

TShiftState);

Begin

If Key = VK\_DOWN Then

EditCountry.SetFocus

Else If Key = VK\_UP Then

EditPrice.SetFocus

Else If Key = VK\_CONTROL Then

Key := Ord(NULL);

End;

Procedure TFormRecord.EditModelKeyPress(Sender: TObject; Var Key: Char);

Begin

StringEditKeyPress(EditModel, Key);

End;

Procedure EditOnExit(Var Edit: TEdit; Const MINNUM, MAXNUM: Integer);

Begin

If (Edit.Text <> '') And Not(IsNumCorrect(Edit.Text, MINNUM, MAXNUM)) Then

Edit.Text := '';

End;

Procedure TFormRecord.EditYearKeyDown(Sender: TObject; Var Key: Word; Shift:

TShiftState);

Begin

If Key = VK\_DOWN Then

EditPrice.SetFocus

Else If Key = VK\_UP Then

EditCountry.SetFocus;

End;

Procedure TFormRecord.EditYearExit(Sender: TObject);

Begin

EditOnExit(EditYear, MINYEAR, MAXYEAR);

End;

Procedure TFormRecord.EditYearKeyPress(Sender: TObject; Var Key: Char);

Var

SelStart, Len: Integer;

EditText: String;

Begin

SelStart := EditYear.SelStart;

EditText := EditYear.Text;

Len := Length(EditText);

NumEditKeyPress(SelStart, Len, EditText, Key);

If (Len = 4) And (Key <> BACKSPACE) Then

Key := NULL

Else If (SelStart = 0) And Not CharInSet(Key, ['1', '2', BACKSPACE]) Then

Key := NULL

Else If (Len > 0) And (SelStart = 0) And ((Key = '1') And (EditText[1] <> '9') Or

(Key = '2') And (EditText[1] <> '0')) Then

Key := NULL

Else If (Len > 0) And (SelStart = 1) And ((EditText[1] = '1') And (Key <> '9') Or

(EditText[1] = '2') And (Key <> '0')) And (Key <> BACKSPACE) Then

Key := NULL

Else If (Len > 1) And (SelStart = 2) And (EditText[1] = '2') And Not

CharInSet(Key, [BACKSPACE, '0' .. '2']) Then

Key := NULL

Else If (EditText = '202') And Not CharInSet(Key, ['0' .. '4', BACKSPACE]) Then

Key := NULL;

End;

Procedure TFormRecord.EditPriceKeyDown(Sender: TObject; Var Key: Word; Shift:

TShiftState);

Begin

If Key = VK\_DOWN Then

EditModel.SetFocus

Else If Key = VK\_UP Then

EditYear.SetFocus;

End;

Procedure TFormRecord.EditPriceKeyPress(Sender: TObject; Var Key: Char);

Var

SelStart, Len: Integer;

EditText: String;

Begin

SelStart := EditPrice.SelStart;

EditText := EditPrice.Text;

Len := Length(EditText);

NumEditKeyPress(SelStart, Len, EditText, Key);

If (Len > 8) And ((StrToInt(EditText) > MAXYEAR Div 10) Or (StrToInt(EditText) =

MAXYEAR)) And (Key <> BACKSPACE) Then

Key := NULL;

End;

Procedure TFormRecord.EditPriceExit(Sender: TObject);

Begin

EditOnExit(EditPrice, MINPRICE, MAXPRICE);

End;

Procedure TFormRecord.EditOnChange(Sender: TObject);

Begin

ButtonSave.Enabled := (EditModel.Text <> '') And (EditCountry.Text <> '') And

(EditYear.Text <> '') And (EditPrice.Text <> '');

IsChanged := True;

End;

Procedure TFormRecord.ClearEdits();

Begin

EditModel.Text := '';

EditCountry.Text := '';

EditYear.Text := '';

EditPrice.Text := '';

End;

Procedure TFormRecord.ChangeRecord(Index: Integer);

Begin

With MainList[Index] Do

Begin

Model := EditModel.Text;

Country := EditCountry.Text;

Year := StrToInt(EditYear.Text);

Price := StrToInt(EditPrice.Text);

End;

End;

Procedure TFormRecord.ButtonCancelClick(Sender: TObject);

Begin

Close;

End;

Procedure TFormRecord.ButtonSaveClick(Sender: TObject);

Begin

If IsRecordChanging Then

Begin

ChangeRecord(SelectedIndex);

MainForm.ChangeListViewItem(MainForm.MainListView, MainList);

End

Else

Begin

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

ChangeRecord(High(MainList));

MainForm.AddItemToListView(MainForm.MainListView, MainList);

End;

IsChanged := False;

Close;

End;

Procedure TFormRecord.FormClose(Sender: TObject; Var Action: TCloseAction);

Begin

ClearEdits();

End;

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

Var

ButtonSelected: Integer;

Begin

If IsChanged Then

Begin

ButtonSelected := Application.MessageBox('Вы уверены, что хотите отменить

ввод записи?', 'Отмена',

MB\_YESNO + MB\_ICONQUESTION);

If ButtonSelected = MrYes Then

CanClose := True

Else

CanClose := False;

End

Else

CanClose := True;

End;

Procedure TFormRecord.FormKeyDown(Sender: TObject; Var Key: Word;

Shift: TShiftState);

Begin

If Key = VK\_F1 Then

FormInstruction.ShowModal

Else If (Key = 13) And (ButtonSave.Enabled) Then

If Not IsNumCorrect(EditYear.Text, MINYEAR, MAXYEAR) Then

EditYear.Text := ''

Else If Not IsNumCorrect(EditPrice.Text, MINPRICE, MAXPRICE) Then

EditPrice.Text := ''

Else

ButtonSave.Click

Else If Key = VK\_ESCAPE Then

Close;

End;

Procedure TFormRecord.FormShow(Sender: TObject);

Begin

EditModel.SetFocus;

If IsRecordChanging Then

Begin

FormRecord.Caption := 'Редактировать запись';

ButtonSave.Caption := 'Сохранить';

With MainList[SelectedIndex] Do

Begin

EditModel.Text := Model;

EditCountry.Text := Country;

EditYear.Text := IntToStr(Year);

EditPrice.Text := IntToStr(Price);

End;

End

Else

Begin

FormRecord.Caption := 'Добавить запись';

ButtonSave.Caption := 'Добавить';

End;

IsChanged := False;

End;

End.

Unit UnitTask;

Interface

Uses

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

System.Classes, Vcl.Graphics, Vcl.Controls, Vcl.Forms, Vcl.Dialogs, Vcl.Menus,

Vcl.StdCtrls, Vcl.ComCtrls, UnitInstruction;

Type

TFormTask = Class(TForm)

MainMenu: TMainMenu;

MMFile: TMenuItem;

MMSaveFile: TMenuItem;

MMSeparator: TMenuItem;

MMClose: TMenuItem;

MMInstruction: TMenuItem;

LabelTask: TLabel;

TaskListView: TListView;

ButtonClose: TButton;

Procedure ButtonCloseClick(Sender: TObject);

Procedure FormShow(Sender: TObject);

Procedure FormClose(Sender: TObject; Var Action: TCloseAction);

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

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

Procedure MMSaveFileClick(Sender: TObject);

Procedure MMInstructionClick(Sender: TObject);

Procedure MMCloseClick(Sender: TObject);

End;

Var

FormTask: TFormTask;

Implementation

{$R \*.dfm}

Uses UnitMain;

Var

IsTaskSaved: Boolean;

SortedTaskList: TCarList;

Procedure TFormTask.ButtonCloseClick(Sender: TObject);

Begin

Close;

End;

Function CreateTaskList(): TCarList;

Var

I: Integer;

TaskList: TCarList;

Begin

For I := 0 To High(MainList) Do

If MainList[I].Year = LASTYEAR Then

Begin

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

TaskList[High(TaskList)] := MainList[I];

End;

CreateTaskList := TaskList;

End;

Procedure SwapElements(Var List: TCarList; I, J: Integer);

Var

Temp: TCar;

Begin

Temp := List[I];

List[I] := List[J];

List[J] := Temp;

End;

Function SortList(List: TCarList): TCarList;

Var

I, J, MaxIndex: Integer;

Begin

For I := 0 To High(List) - 1 Do

Begin

MaxIndex := I;

For J := I + 1 To High(List) Do

If List[J].Price > List[MaxIndex].Price Then

MaxIndex := J;

If MaxIndex <> I Then

SwapElements(List, I, MaxIndex);

End;

SortList := List;

End;

Procedure TFormTask.FormClose(Sender: TObject; Var Action: TCloseAction);

Begin

MainForm.ClearList(TaskListView, SortedTaskList);

End;

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

Begin

MainForm.OnCloseQuery(SortedTaskList, IsTaskSaved, CanClose);

End;

Procedure TFormTask.FormKeyDown(Sender: TObject; Var Key: Word;

Shift: TShiftState);

Begin

If Key = VK\_ESCAPE Then

Close;

End;

Procedure TFormTask.FormShow(Sender: TObject);

Var

I: Integer;

TaskList: TCarList;

Begin

IsTaskSaved := False;

TaskList := CreateTaskList();

SortedTaskList := SortList(TaskList);

For I := 0 To High(SortedTaskList) Do

MainForm.AddItemToListView(TaskListView, SortedTaskList);

End;

Procedure TFormTask.MMCloseClick(Sender: TObject);

Begin

Close;

End;

Procedure TFormTask.MMInstructionClick(Sender: TObject);

Begin

FormInstruction.ShowModal;

End;

Procedure TFormTask.MMSaveFileClick(Sender: TObject);

Begin

MainForm.SaveFile(SortedTaskList, IsTaskSaved);

End;

End.

Unit UnitInstruction;

Interface

Uses

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

System.Classes, Vcl.Graphics, Vcl.Controls, Vcl.Forms, Vcl.Dialogs, Vcl.StdCtrls;

Type

TFormInstruction = Class(TForm)

LabelInstruction: TLabel;

ButtonClose: TButton;

Procedure FormCreate(Sender: TObject);

Procedure ButtonCloseClick(Sender: TObject);

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

End;

Var

FormInstruction: TFormInstruction;

Implementation

{$R \*.dfm}

Procedure TFormInstruction.ButtonCloseClick(Sender: TObject);

Begin

Close;

End;

Procedure TFormInstruction.FormCreate(Sender: TObject);

Begin

LabelInstruction.Caption :='ИЗМЕНЕНИЕ СПИСКА:'#13#10'1. Для добавления записи,

нажмите на кнопку "Новая запись" и введите данные в новом окне.'#13#10'2. Для

изменения записи, нажмите на нужную запись 2 раза.'#13#10'3. Для удаления,

выберите нужную запись и нажмите на кнопку "Удалить".'#13#10#13#10

+ 'ДАННЫЕ:'#13#10'1. Максимальная длина данных о марке и государстве-

производителе - 20 символов.'#13#10'2. Год выпуска и цена - целые числа.'

#13#10'3. Диапазон года выпуска: 1900..2024.'#13#10'4. Диапазон цены:

0..999999999.'#13#10'5. Максимальное количество записей - 30.'#13#10#13#10

+ 'ФАЙЛЫ:'#13#10'1. Открываемый/сохраняемый файл должен быть формата \*.carlist.'

#13#10'2. При сохранении данные указанного файла перезаписываются.'#13#10#13#10;

LabelInstruction.Width := 500;

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

End;

Procedure TFormInstruction.FormKeyDown(Sender: TObject; Var Key: Word; Shift:

TShiftState);

Begin

If Key = VK\_ESCAPE Then

Close;

End;

End.

**КОД ПРОГРАММЫ С++:**

#include <iostream>

#include <fstream>

#include <locale>

#include <Windows.h>

#include <string>

using namespace std;

const int MINYEAR = 1900;

const int MAXYEAR = 2024;

const int MINPRICE = 0;

const int MAXPRICE = 999999999;

const int MAXLENGTH = 20;

const int LASTYEAR = 2023;

const int MAXRECORDS = 30;

struct car {

string model;

string country;

int year;

int price;

};

string inputString(string outputMessage) {

string str;

bool isIncorrect;

do {

isIncorrect = false;

cout << outputMessage;

getline(cin, str, '\n');

if (str.size() > MAXLENGTH) {

isIncorrect = true;

cout << "Длина не должна превышать 20 символов!\n";

}

} while (isIncorrect);

return str;

}

int inputNum(string outputMessage, const int MIN, const int MAX) {

int num;

bool isIncorrect;

do {

isIncorrect = false;

cout << outputMessage;

cin >> num;

if (cin.get() != '\n') {

cin.clear();

while (cin.get() != '\n');

isIncorrect = true;

cout << "Введенное значение должно быть целым числом! Повторите

попытку.\n";

}

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

cout << "Диапазон числа не соответствует условию! Повторите

попытку.\n";

isIncorrect = true;

}

} while (isIncorrect);

return num;

}

car inputRecord() {

car newCar;

newCar.model = inputString("Введите марку автомобиля (до 20 символов): ");

newCar.country = inputString("Введите государство-производителя (до 20

символов): ");

newCar.year = inputNum("Введите год выпуска (целое число от 1900 до 2024): ",

MINYEAR, MAXYEAR);

newCar.price = inputNum("Введите цену (целое число от 0 до 999999999): ",

MINPRICE, MAXPRICE);

return newCar;

}

bool isListEmpty(int listLength) {

if (listLength == 0) {

cout << "В данный момент список пуст.\n";

return true;

}

else

return false;

}

car\* addRecordToList(car\* list, int& listLength, bool& isSaved) {

car newCar;

car\* newList;

if (listLength < MAXRECORDS) {

newCar = inputRecord();

newList = new car[listLength + 1];

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

newList[i] = list[i];

newList[listLength] = newCar;

cout << "Автомобиль успешно добавлен в список!\n";

listLength++;

isSaved = false;

}

else {

cout << "Достигнуто максимальное количество записей!\n";

newList = list;

}

return newList;

}

void outputList(car\* list, int listLength) {

if (!isListEmpty(listLength)) {

cout << "№ || Марка || Государство || Год || Цена\n";

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

cout << i + 1 << ". " << list[i].model << " " << list[i].country

<< " " << list[i].year << " " << list[i].price << "\n";

}

}

car\* editRecord(car\* list, int listLength, bool& isSaved) {

int selectedIndex;

selectedIndex = 0;

outputList(list, listLength);

if (listLength != 0) {

selectedIndex = inputNum("Введите номер записи, которую хотите

изменить, или 0 для выхода в меню: ", 0, listLength) - 1;

if (selectedIndex != -1) {

list[selectedIndex] = inputRecord();

cout << "Запись успешно изменена!\n";

isSaved = false;

}

}

return list;

}

car\* deleteRecord(car\* list, int& listLength, bool& isSaved) {

int deleteIndex;

car\* newList;

deleteIndex = 0;

outputList(list, listLength);

if (listLength != 0) {

deleteIndex = inputNum("Введите номер записи, которую хотите удалить,

или 0 для выхода в меню: ", 0, listLength) - 1;

if (deleteIndex != -1) {

for (int i = deleteIndex; i < listLength - 1; i++)

list[i] = list[i + 1];

newList = new car[listLength - 1];

for (int i = 0; i < listLength - 1; i++)

newList[i] = list[i];

cout << "Запись успешно удалена!\n";

listLength--;

isSaved = false;

}

else

newList = list;

}

else

newList = list;

return newList;

}

bool isFileExtIncorrect(string path, bool isIncorrect) {

if (!isIncorrect && (path.substr(path.size() - 8) != ".carlist")) {

cout << "Введенный Вами файл не является формата \*.carlist! Повторите

попытку.\n";

isIncorrect = true;

}

return isIncorrect;

}

string getFileInputPath() {

string path;

bool isIncorrect;

do {

isIncorrect = false;

cout << "Введите путь к файлу:\n";

getline(cin, path, '\n');

ifstream fin(path);

fin.open(path);

if (!fin.is\_open()) {

cout << "Файл недоступен для чтения! Повторите попытку.\n";

isIncorrect = true;

}

isIncorrect = isFileExtIncorrect(path, isIncorrect);

fin.close();

} while (isIncorrect);

return path;

}

car\* getListFromFile(int& listLength, bool& isSaved) {

bool isIncorrect;

string path;

car\* list;

path = getFileInputPath();

list = new car();

do {

isIncorrect = false;

ifstream fin(path);

try {

fin >> listLength;

isIncorrect = listLength > MAXRECORDS;

list = new car[listLength];

int i = 0;

while ((!isIncorrect) && (i < (listLength))) {

getline(fin, list[i].model, '\n');

getline(fin, list[i].country, '\n');

fin >> list[i].year;

fin >> list[i].price;

isIncorrect = (list[i].model.size() > MAXLENGTH) ||

(list[i].country.size() > MAXLENGTH) ||

(list[i].year > MAXYEAR) || (list[i].year <

MINYEAR) || (list[i].price > MAXPRICE) ||

(list[i].price < MINPRICE);

i++;

}

if (!isIncorrect && !fin.eof())

isIncorrect = true;

}

catch (string errorMessage) {

cout << "Данные выбранного файла некорректны! Повторите

попытку.\n";

isIncorrect = true;

}

fin.close();

if (isIncorrect) {

cout << "Данные выбранного файла некорректны! Повторите

попытку.\n";

path = getFileInputPath();

}

} while (isIncorrect);

cout << "Данные из файла успешно считаны.\n";

isSaved = true;

return list;

}

string getFileOutputPath() {

string path;

bool isIncorrect;

do {

isIncorrect = false;

cout << "Введите путь к файлу, в который нужно записать результат:\n";

getline(cin, path, '\n');

ofstream fout(path);

fout.open(path);

if (!fout.is\_open()) {

cout << "Файл недоступен для записи! Повторите попытку.\n";

isIncorrect = true;

}

isIncorrect = isFileExtIncorrect(path, isIncorrect);

fout.close();

} while (isIncorrect);

return path;

}

void writeListIntoFile(car\* list, int listLength, bool& isSaved) {

if (!isListEmpty(listLength)) {

bool isIncorrect;

string path;

path = getFileOutputPath();

do {

isIncorrect = false;

ofstream fout(path);

try {

fout << listLength << " ";

for (int i = 0; i < listLength; i++) {

fout << list[i].model << "\n" << list[i].country <<

"\n" << list[i].year << "\n" << list[i].price;

if (i < listLength - 1)

fout << " ";

}

}

catch (string errorMessage) {

cout << "Произошла ошибка. Повторите попытку.\n";

isIncorrect = true;

}

fout.close();

if (isIncorrect)

path = getFileOutputPath();

} while (isIncorrect);

cout << "Результат записан.\n";

isSaved = true;

}

}

void saveFileOnExit(car\* list, int listLength, string outputMessage, bool& isSaved) {

const int NOTSAVETOFILE = 0;

const int SAVETOFILE = 1;

int choice;

choice = inputNum(outputMessage, NOTSAVETOFILE, SAVETOFILE);

if (choice == SAVETOFILE)

writeListIntoFile(list, listLength, isSaved);

}

bool isLastYearInList(car\* list, int listLength) {

int i;

bool isLastYear;

i = 0;

isLastYear = false;

while ((i < listLength) && !isLastYear) {

if (list[i].year == LASTYEAR)

isLastYear = true;

i++;

}

return isLastYear;

}

int findTaskListLength(car\* mainList, int mainListLength) {

int length;

length = 0;

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

if (mainList[i].year == LASTYEAR)

length++;

return length;

}

car\* createTaskList(car\* mainList, int mainListLength, int taskListLength) {

car\* taskList;

int j;

taskList = new car[taskListLength];

j = 0;

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

if (mainList[i].year == LASTYEAR) {

taskList[j] = mainList[i];

j++;

}

return taskList;

}

car\* sortList(car\* list, int listLength) {

int maxIndex;

for (int i = 0; i < listLength - 1; i++) {

maxIndex = i;

for (int j = i + 1; j < listLength; j++)

if (list[j].price > list[maxIndex].price)

maxIndex = j;

if (maxIndex != i)

swap(list[maxIndex], list[i]);

}

return list;

}

void performTask(car\* mainList, int mainListLength, bool isSaved) {

cout << "Вывести информацию об автомобилях выпуска прошлого года в порядке

убывания цен.\n";

if (isLastYearInList(mainList, mainListLength)) {

int taskListLength;

car\* taskList;

car\* sortedTaskList;

taskListLength = findTaskListLength(mainList, mainListLength);

taskList = createTaskList(mainList, mainListLength, taskListLength);

sortedTaskList = sortList(taskList, taskListLength);

outputList(sortedTaskList, taskListLength);

saveFileOnExit(sortedTaskList, taskListLength, "\nЖелаете сохранить

список из задания в файл? Введите 1, если да, и 0, если

нет: ", isSaved);

}

else

cout << "В списке нет автомобилей, выпущенных в прошлом году.\n";

}

void outputInstruction() {

cout << "Инструкция\n\nДАННЫЕ:\n1. Максимальная длина данных о марке и

государстве-производителе - 20 символов.\n" << "2. Год выпуска и цена

- целые числа.\n" << "3. Диапазон года выпуска: 1900..2024.\n" << "4.

Диапазон цены: 0..999999999.\n" << "5. Максимальное количество

записей - 30.\n\n" << "ФАЙЛЫ:\n1. Открываемый / сохраняемый файл

должен быть формата \* .carlist.\n" << "2. При сохранении данные

указанного файла перезаписываются.\n\n" << "Разработчик: Городко

Ксения\nЛабораторная №4.1\n";

}

void outputProgramInfo() {

cout << "Добро пожаловать в Гараж - программу для создания и редактирования

списков со сведениями об автомобилях!\n";

}

void outputMenu() {

cout << "Меню:\n";

cout << "1) Добавить новую запись;\n" << "2) Изменить уже имеющуюся запись;\n"

<< "3) Удалить запись;\n" << "4) Вывести текущий список записей;\n"

<< "5) Вывести информацию по заданию;\n" << "6) Открыть файл;\n"

<< "7) Сохранить записи в файл;\n" << "8) Вывести инструкцию;\n"

<< "9) Выход из программы.\n";

cout << "Для выбора пункта меню введите его номер.\n";

}

void returnToMenu() {

cout << "\nНажмите Enter для возвращения в меню...\n";

getchar();

}

void performMenuOption() {

const int ADDRECORD = 1;

const int EDITRECORD = 2;

const int DELETERECORD = 3;

const int SHOWLIST = 4;

const int SHOWTASK = 5;

const int OPENFILE = 6;

const int SAVEFILE = 7;

const int SHOWINSTRUCTION = 8;

const int EXIT = 9;

int choice;

bool isSaved;

int listLength;

car\* mainList;

choice = 0;

isSaved = true;

listLength = 0;

mainList = new car();

while (choice != EXIT) {

outputMenu();

choice = inputNum("", ADDRECORD, EXIT);

cout << "\n";

switch (choice) {

case ADDRECORD: {

mainList = addRecordToList(mainList, listLength, isSaved);

break;

}

case EDITRECORD: {

mainList = editRecord(mainList, listLength, isSaved);

break;

}

case DELETERECORD: {

mainList = deleteRecord(mainList, listLength, isSaved);

break;

}

case SHOWLIST: {

outputList(mainList, listLength);

break;

}

case SHOWTASK: {

performTask(mainList, listLength, isSaved);

break;

}

case OPENFILE: {

mainList = getListFromFile(listLength, isSaved);

break;

}

case SAVEFILE: {

writeListIntoFile(mainList, listLength, isSaved);

break;

}

case SHOWINSTRUCTION: {

outputInstruction();

break;

}

}

if (choice != EXIT)

returnToMenu();

else if (!isSaved && (listLength != 0)) {

saveFileOnExit(mainList, listLength, "Желаете сохранить список в

файл? Введите 1, если да, и 0, если нет: ",

isSaved);

}

}

cout << "Благодарим за приобретение программы Гараж!";

}

int main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

outputProgramInfo();

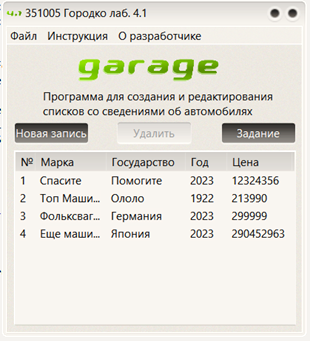
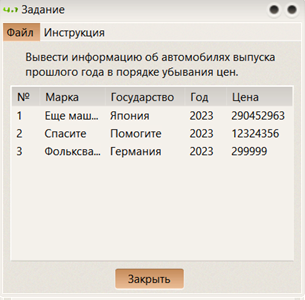
performMenuOption();

return 0;

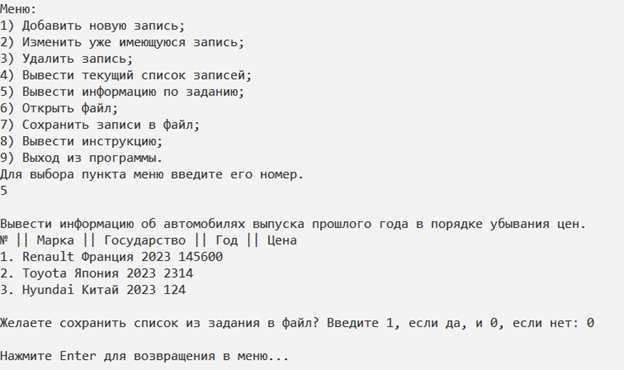
}

**СКРИНШОТЫ:**

**Delphi:**

**C++****:**

****

**БЛОК-СХЕМА:**

