УО «Белорусский государственный университет информатики и радиоэлектроники»

Кафедра ПОИТ

Отчет по лабораторной работе №4.1

по предмету «Основы алгоритмизации и программирования»

Вариант 18

Выполнил:

Егоров А.С.

Гр. 351005

Проверил:

Данилова Г. В.

Минск 2023

**Задание:**

1. Записи

Сведения о количестве изделий категорий А, В, С, собранных рабочим за месяц

- фамилия сборщика;

- наименование цеха;

- количество изделий по категории, собранных рабочим за месяц.

Считая заданными значения расценок Sa, Sb, Sc за выполненную работу по сборке единицы

изделия категорий А, В и С соответственно, вывести в текстовый файл следующую информацию:

общее количество изделий категорий А, В, С, собранных рабочим цеха X; ведомость заработной

платы рабочих цеха X; средний размер заработной платы работников этого цеха.

**Код программы Delphi:**

**UnitMain.pas**

Unit UnitMain;

Interface

Uses

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

System.Classes, Vcl.Graphics,

Vcl.Controls, Vcl.Forms, Vcl.Dialogs, Vcl.Menus, Vcl.Buttons,

System.ImageList, Vcl.ImgList, Vcl.StdCtrls, Vcl.Grids,

UnitBackend, UnitAboutTheDeveloper, UnitInstruction, UnitDeleter, UnitExit;

Type

TuVCLMain = Class(TForm)

MainMenu1: TMainMenu;

BtFile: TMenuItem;

BtOpenFile: TMenuItem;

BtSaveFile: TMenuItem;

BtInstruction: TMenuItem;

BtAboutTheDeveloper: TMenuItem;

OpenDialog1: TOpenDialog;

SaveDialog1: TSaveDialog;

ImageList1: TImageList;

StrGrWorkers: TStringGrid;

LbInfo: TLabel;

EFieldA: TEdit;

EFieldB: TEdit;

EFieldC: TEdit;

LbInfoA: TLabel;

LbInfoB: TLabel;

LbInfoC: TLabel;

LTableName: TLabel;

BitBtAddWorker: TBitBtn;

BitBtDeleteWorker: TBitBtn;

BitbtShowDopInfo: TBitBtn;

ECompany: TEdit;

LCompanyInfo: TLabel;

BtSaveDopInfo: TMenuItem;

Procedure FormCreate(Sender: TObject);

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

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

Shift: TShiftState);

Procedure EFieldChange(Sender: TObject);

Procedure BtAboutTheDeveloperClick(Sender: TObject);

Procedure BtInstructionClick(Sender: TObject);

Procedure BtOpenFileClick(Sender: TObject);

Procedure BtSaveFileClick(Sender: TObject);

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

Shift: TShiftState);

Procedure BitBtAddWorkerClick(Sender: TObject);

Procedure BitBtDeleteWorkerClick(Sender: TObject);

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

Shift: TShiftState);

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

Shift: TShiftState);

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

Shift: TShiftState);

Procedure ECompanyChange(Sender: TObject);

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

Shift: TShiftState);

Procedure BitbtShowDopInfoClick(Sender: TObject);

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

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

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

Private

Workers: TWorkers;

IsTableFilled: Boolean;

BufferHandler: TBufferHandler;

IsFileSaved: Boolean;

Procedure ClearRow(Const Index: Integer);

Public

{ Public declarations }

End;

Var

UVCLMain: TuVCLMain;

Implementation

{$R \*.dfm}

Procedure TuVCLMain.BitBtAddWorkerClick(Sender: TObject);

Begin

StrGrWorkers.RowCount := StrGrWorkers.RowCount + 1;

StrGrWorkers.Cells[0, StrGrWorkers.RowCount - 1] :=

IntToStr(StrGrWorkers.RowCount - 1);

BitBtDeleteWorker.Enabled := (StrGrWorkers.RowCount > 2);

// выключение кнопки сохранить

BtSaveFile.Enabled := False;

End;

Procedure TuVCLMain.BitBtDeleteWorkerClick(Sender: TObject);

Var

IndexOfDeletedWorker: Integer;

I: Integer;

Begin

Application.CreateForm(TuVCLDeleter, UVCLDeleter);

UVCLDeleter.CountOfWorkers := StrGrWorkers.RowCount -

StrGrWorkers.FixedRows;

UVCLDeleter.ShowModal();

If UVCLDeleter.IsChosen And (StrGrWorkers.RowCount > 2) Then

Begin

// знать бы как такое получилось)

IndexOfDeletedWorker := StrToInt(UVCLDeleter.ENumberOfWorker.Text);

// смещение всех последующих строк на 1 строку назад

For I := IndexOfDeletedWorker To StrGrWorkers.RowCount - 1 Do

Begin

StrGrWorkers.Cells[1, I] := StrGrWorkers.Cells[1, I + 1];

StrGrWorkers.Cells[2, I] := StrGrWorkers.Cells[2, I + 1];

StrGrWorkers.Cells[3, I] := StrGrWorkers.Cells[3, I + 1];

StrGrWorkers.Cells[4, I] := StrGrWorkers.Cells[4, I + 1];

StrGrWorkers.Cells[5, I] := StrGrWorkers.Cells[5, I + 1];

End;

// очистка последней строки

ClearRow(StrGrWorkers.RowCount);

// удаление этой(последней) строки

StrGrWorkers.RowCount := StrGrWorkers.RowCount - 1;

// выключение кнопки сохранить

If IsTableFilled Then

BtSaveFile.Enabled := True;

End

Else If UVCLDeleter.IsChosen And (StrGrWorkers.RowCount = 2) Then

Begin

// очистка последней строки

ClearRow(StrGrWorkers.FixedRows);

StrGrWorkers.Cells[0, StrGrWorkers.FixedRows] := '1';

End;

UVCLDeleter.Destroy();

End;

Procedure TuVCLMain.BitbtShowDopInfoClick(Sender: TObject);

Var

I: Integer;

IsCompanyExist: Boolean;

FileWriter: TFileWriter;

CurCompany: String;

CountOfAllDetailsByWorker: Integer;

PriceA, PriceB, PriceC: Integer;

SalaryOfWorker: Integer;

AllSalariesOfCompany: Real;

CountOfCompanyWorkers: Integer;

Begin

// TODO

IsCompanyExist := False;

CurCompany := ECompany.Text;

For I := 1 To Workers.Count Do

If Not IsCompanyExist And (Workers[I - 1].CompanyName = CurCompany) Then

IsCompanyExist := True;

If IsCompanyExist And SaveDialog1.Execute() Then

Begin

// Создаем обьект для записи данных

FileWriter := TFileWriter.Create(TfText);

FileWriter.FileName := SaveDialog1.FileName;

FileWriter.CheckFile();

If FileWriter.Status = FsGood Then

Begin

// стераем данные из файла

FileWriter.ResetFile();

// инициализая начального значения всех ЗП на заводе

AllSalariesOfCompany := 0;

// инициализия начальног значения бедолаг этого завода

CountOfCompanyWorkers := 0;

// задаем занчение расценок для ЗП

PriceA := StrToInt(EFieldA.Text);

PriceB := StrToInt(EFieldB.Text);

PriceC := StrToInt(EFieldC.Text);

FileWriter.WriteStrln(('Информация о цехе "' + CurCompany + '":'));

For I := 1 To Workers.Count Do

Begin

If (Workers[I - 1].CompanyName = CurCompany) Then

Begin

// подсчет и запись общего количества изделий категорий А, В, С,

// собранных рабочим компани X

CountOfAllDetailsByWorker := +Workers[I - 1].CountOfDetailsA

+ Workers[I - 1].CountOfDetailsB + Workers[I - 1]

.CountOfDetailsC;

FileWriter.WriteStrln

(('Рабочий ' + Workers[I - 1].Surname + ' сделал ' +

IntToStr(CountOfAllDetailsByWorker) + ' деталий.'));

// подсчет и запись ведомость заработной платы рабочих цеха X;

SalaryOfWorker := PriceA \* Workers[I - 1].CountOfDetailsA +

PriceB \* Workers[I - 1].CountOfDetailsB + PriceC \*

Workers[I - 1].CountOfDetailsC;

FileWriter.WriteStrln

(('Рабочий ' + Workers[I - 1].Surname +

' получает ЗП в размере ' + IntToStr(SalaryOfWorker)

+ ' руб'));

// Добавление ЗП к общему число

AllSalariesOfCompany := AllSalariesOfCompany +

SalaryOfWorker;

// Изменения счетчика работчих этой компании на 1

Inc(CountOfCompanyWorkers);

End;

End;

// Запись средний размер заработной платы работников этой компании.

AllSalariesOfCompany := AllSalariesOfCompany /

CountOfCompanyWorkers;

FileWriter.WriteStrln(('Средня ЗП в ' + CurCompany + ' - ' +

FloatToStr(AllSalariesOfCompany) + ' руб.'));

MessageBox(UVCLMain.Handle, ListOfMessages[FileWriter.Status],

'Ой-йой', MB\_OK);

End

Else

MessageBox(UVCLMain.Handle, ListOfMessages[FileWriter.Status],

'Ой-йой', MB\_ICONERROR);

End

Else

MessageBox(UVCLMain.Handle, 'Цех не существует!', 'Ой-йой',

MB\_ICONERROR);

End;

Procedure TuVCLMain.BtAboutTheDeveloperClick(Sender: TObject);

Begin

Application.CreateForm(TuVCLAboutTheDeveloper, UVCLAboutTheDeveloper);

UVCLAboutTheDeveloper.Show();

If UVCLAboutTheDeveloper.IsClose Then

UVCLAboutTheDeveloper.Destroy();

End;

Procedure TuVCLMain.BtInstructionClick(Sender: TObject);

Begin

Application.CreateForm(TuVCLInstruction, UVCLInstruction);

UVCLInstruction.Show();

If UVCLInstruction.IsClose Then

UVCLInstruction.Destroy();

End;

Procedure TuVCLMain.EFieldChange(Sender: TObject);

Var

CurEdit: TEdit;

Begin

CurEdit := TEdit(Sender);

BufferHandler.EditText := CurEdit.Text;

If Not BufferHandler.CheckInput(TpInteger) Then

Begin

MessageBox(UVCLMain.Handle, 'Вы ввели неправильные смиволы!', 'Ой-йой',

MB\_ICONERROR);

CurEdit.Text := '';

End;

BufferHandler.DeleteLeadingZeros(TpInteger);

CurEdit.Text := BufferHandler.EditText;

BitbtShowDopInfo.Enabled := Not String.IsNullOrEmpty(EFieldA.Text) And

Not String.IsNullOrEmpty(EFieldB.Text) And

Not String.IsNullOrEmpty(EFieldC.Text) And

Not String.IsNullOrEmpty(ECompany.Text) And IsTableFilled;

BtSaveDopInfo.Enabled := BitbtShowDopInfo.Enabled;

End;

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

Var

ExitCode: Integer;

Begin

If Not IsTableFilled Or IsFileSaved Then

Begin

Application.CreateForm(TuVCLExit, UVCLExit);

UVCLExit.ShowModal;

CanClose := UVCLExit.GetStatus();

UVCLExit.Destroy();

End

Else If Not IsFileSaved Then

Begin

Repeat

ExitCode := MessageBox(UVCLMain.Handle,

'Сохранить данные в файл перед выходом?', 'Подверждение',

MB\_ICONQUESTION + MB\_YESNOCANCEL);

If ExitCode = ID\_YES Then

Begin

BtSaveFileClick(UVCLMain);

CanClose := True;

End

Else If ExitCode = ID\_NO Then

CanClose := True

Else

CanClose := False;

Until IsFileSaved Or (ExitCode = ID\_NO) Or (ExitCode = ID\_CANCEL);

End;

End;

Procedure TuVCLMain.FormCreate(Sender: TObject);

Begin

// инициализация таблицы

StrGrWorkers.FixedCols := 1;

StrGrWorkers.FixedRows := 1;

StrGrWorkers.RowCount := 2;

StrGrWorkers.Cells[0, 0] := '№';

StrGrWorkers.ColWidths[0] := 35;

StrGrWorkers.Cells[1, 0] := 'Фамилия';

StrGrWorkers.ColWidths[1] := 100;

StrGrWorkers.Cells[2, 0] := 'Название Цеха';

StrGrWorkers.ColWidths[2] := 120;

StrGrWorkers.Cells[3, 0] := 'A(кол.)';

StrGrWorkers.ColWidths[3] := 55;

StrGrWorkers.Cells[4, 0] := 'B(кол.)';

StrGrWorkers.ColWidths[4] := 55;

StrGrWorkers.Cells[5, 0] := 'C(кол.)';

StrGrWorkers.ColWidths[5] := 55;

StrGrWorkers.Cells[0, 1] := '1';

// инициализация работников

Workers := TWorkers.Create();

// инициализация обработчика буфера

BufferHandler := TBufferHandler.Create();

// инициализация переменных

IsFileSaved := False;

End;

Procedure TuVCLMain.BtOpenFileClick(Sender: TObject);

Var

FileReader: TFileReader;

I: Integer;

Worker: TWorker;

Begin

If OpenDialog1.Execute() Then

Begin

FileReader := TFileReader.Create();

FileReader.FileName := OpenDialog1.FileName;

FileReader.CheckFile();

// TODO статусы

If FileReader.Status = FsGood Then

Begin

Workers := FileReader.ReadTable();

For I := Workers.Count + StrGrWorkers.FixedRows To StrGrWorkers.

RowCount - StrGrWorkers.FixedRows Do

ClearRow(I);

StrGrWorkers.RowCount := Workers.Count + StrGrWorkers.FixedRows;

For I := StrGrWorkers.FixedRows To Workers.Count Do

Begin

Worker := Workers[I - 1];

StrGrWorkers.Cells[0, I] := IntToStr(I);

StrGrWorkers.Cells[1, I] := Worker.Surname;

StrGrWorkers.Cells[2, I] := Worker.CompanyName;

StrGrWorkers.Cells[3, I] := IntToStr(Worker.CountOfDetailsA);

StrGrWorkers.Cells[4, I] := IntToStr(Worker.CountOfDetailsB);

StrGrWorkers.Cells[5, I] := IntToStr(Worker.CountOfDetailsC);

End;

// включение кнопки сохранить

BtSaveFile.Enabled := True;

IsTableFilled := True;

End

Else

MessageBox(UVCLMain.Handle, ListOfMessages[FileReader.Status],

'Ой-йой', MB\_ICONERROR);

End;

End;

Procedure TuVCLMain.BtSaveFileClick(Sender: TObject);

Var

FileWriter: TFileWriter;

Begin

If SaveDialog1.Execute() Then

Begin

FileWriter := TFileWriter.Create(TfWorkers);

FileWriter.FileName := SaveDialog1.FileName;

FileWriter.CheckFile();

If FileWriter.Status = FsGood Then

Begin

FileWriter.SaveTable(Workers);

If FileWriter.Status <> FsGood Then

Begin

MessageBox(UVCLMain.Handle, ListOfMessages[FileWriter.Status],

'Ой-йой', MB\_ICONERROR);

End

else

IsFileSaved := True;

End

Else

MessageBox(UVCLMain.Handle, ListOfMessages[FileWriter.Status],

'Ой-йой', MB\_ICONERROR);

FileWriter.Destroy();

End;

End;

Procedure TuVCLMain.ClearRow(Const Index: Integer);

Begin

StrGrWorkers.Cells[0, Index] := '';

StrGrWorkers.Cells[1, Index] := '';

StrGrWorkers.Cells[2, Index] := '';

StrGrWorkers.Cells[3, Index] := '';

StrGrWorkers.Cells[4, Index] := '';

StrGrWorkers.Cells[5, Index] := '';

End;

Procedure TuVCLMain.ECompanyChange(Sender: TObject);

Var

CurEdit: TEdit;

Begin

CurEdit := TEdit(Sender);

BufferHandler.EditText := CurEdit.Text;

If Not BufferHandler.CheckInput(TpString) Then

Begin

MessageBox(UVCLMain.Handle, 'Вы ввели неправильные смиволы!', 'Ой-йой',

MB\_ICONERROR);

CurEdit.Text := '';

End;

BufferHandler.DeleteLeadingZeros(TpInteger);

CurEdit.Text := BufferHandler.EditText;

BitbtShowDopInfo.Enabled := Not String.IsNullOrEmpty(EFieldA.Text) And

Not String.IsNullOrEmpty(EFieldB.Text) And

Not String.IsNullOrEmpty(EFieldC.Text) And

Not String.IsNullOrEmpty(ECompany.Text) And IsTableFilled;

BtSaveDopInfo.Enabled := BitbtShowDopInfo.Enabled;

End;

Procedure TuVCLMain.StrGrWorkersKeyUp(Sender: TObject; Var Key: Word;

Shift: TShiftState);

Var

I: Integer;

Worker: TWorker;

Begin

// считывание елементов

IsTableFilled := True;

For I := 1 To StrGrWorkers.RowCount - 1 Do

If (StrGrWorkers.Cells[1, I] = '') Or (StrGrWorkers.Cells[2, I] = '') Or

(StrGrWorkers.Cells[3, I] = '') Or (StrGrWorkers.Cells[4, I] = '')

Or (StrGrWorkers.Cells[5, I] = '') Then

IsTableFilled := False;

If IsTableFilled Then

Begin

// запись данных

Workers.Count := StrGrWorkers.RowCount - 1;

For I := 1 To StrGrWorkers.RowCount - 1 Do

Begin

Worker.ID := I;

Worker.Surname := StrGrWorkers.Cells[1, I];

Worker.CompanyName := StrGrWorkers.Cells[2, I];

Worker.CountOfDetailsA := StrToInt(StrGrWorkers.Cells[3, I]);

Worker.CountOfDetailsB := StrToInt(StrGrWorkers.Cells[4, I]);

Worker.CountOfDetailsC := StrToInt(StrGrWorkers.Cells[5, I]);

Workers[I - 1] := Worker;

End;

// включение кнопки сохранить

BtSaveFile.Enabled := True;

End

Else // выключение кнопки сохранить

BtSaveFile.Enabled := False;

// проверка на доступ к доп кнопки после ввода в табицу

BitbtShowDopInfo.Enabled := Not String.IsNullOrEmpty(EFieldA.Text) And

Not String.IsNullOrEmpty(EFieldB.Text) And

Not String.IsNullOrEmpty(EFieldC.Text) And

Not String.IsNullOrEmpty(ECompany.Text) And IsTableFilled;

BtSaveDopInfo.Enabled := BitbtShowDopInfo.Enabled;

// меняем статус сохранению

IsFileSaved := False;

End;

End.  
  
**UnitBackend.pas**  
Unit UnitBackend;

Interface

Uses System.SysUtils;

Type

TWorker = Record

ID: Integer;

Surname: String[13];

CompanyName: String[18];

CountOfDetailsA: Integer;

CountOfDetailsB: Integer;

CountOfDetailsC: Integer;

End;

TWorkers = Class

Private

FWorkersArray: Array Of TWorker;

FWorkersCount: Integer;

Procedure SetWorkersCount(Const WorkersCount: Integer);

Procedure SetWorker(Index: Integer; Const Worker: TWorker);

Function GetWorker(Index: Integer): TWorker;

Public

Constructor Create();

Property Count: Integer Read FWorkersCount Write SetWorkersCount;

Property Worker[Index: Integer]: TWorker Read GetWorker

Write SetWorker; Default;

End;

TTypes = (TpInteger, TpUInteger, TpReal, TpString);

TBufferHandler = Class

Private

FEditText: String;

Function CountSymbol(Const Symbol: Char): Integer;

Public

Function CheckInput(Const InputType: TTypes): Boolean;

Procedure DeleteLeadingZeros(Const InputType: TTypes);

Property EditText: String Read FEditText Write FEditText;

End;

TFileStatus = (FsGood, FsNotFound, FsNotTxt, FsNotReadable,

FsNotWritable, FsUnexpected);

TFileReader = Class

Private

FFileName: String;

FInFile: File Of TWorker;

FFileStatus: TFileStatus;

Function IsFileTxt(): Boolean;

Function IsFileReadable(): Boolean;

Procedure SetFileName(Const FFileName: String);

Public

Property Status: TFileStatus Read FFileStatus;

Property FileName: String Read FFileName Write SetFileName;

Procedure CheckFile();

Function ReadTable(): TWorkers;

End;

TTypeFile = (TfText, TfWorkers);

TFileWriter = Class

Private

FFileName: String;

FTypeFile: TTypeFile;

FOutFileTxt: TextFile;

FOutFile: File Of TWorker;

FFileStatus: TFileStatus;

Function IsFileTxt(): Boolean;

Function IsFileWritable(): Boolean;

Procedure SetFileName(Const FFileName: String);

Public

Constructor Create(Const TypeFile: TTypeFile);

Property Status: TFileStatus Read FFileStatus;

Property FileName: String Read FFileName Write SetFileName;

Procedure CheckFile();

Procedure SaveTable(Const Workers: TWorkers);

Procedure WriteStr(Const Str: String);

Procedure WriteStrln(Const Str: String);

Procedure ResetFile();

End;

TListOfMessages = Array [TFileStatus] Of PWideChar;

Const

ListOfMessages: TListOfMessages = ('Информация записана!',

'Файл не найден! Повторите ещё раз.',

'Файл не текстовый! Повторите ещё раз.',

'Файл не доступен для чтения! Повторите ещё раз.',

'Файл не доступен для записи! Повторите ещё раз.',

'Упс... Что-то пошло не так. Потворите ещё раз.');

Implementation

{ TWorkers }

Constructor TWorkers.Create;

Begin

FWorkersCount := 0;

End;

Function TWorkers.GetWorker(Index: Integer): TWorker;

Begin

GetWorker := FWorkersArray[Index];

End;

Procedure TWorkers.SetWorker(Index: Integer; Const Worker: TWorker);

Begin

Try

FWorkersArray[Index] := Worker;

Except

Inc(FWorkersCount);

SetLength(FWorkersArray, FWorkersCount);

FWorkersArray[Index] := Worker;

End;

End;

Procedure TWorkers.SetWorkersCount(Const WorkersCount: Integer);

Begin

FWorkersCount := WorkersCount;

FWorkersArray := Nil;

Setlength(FWorkersArray, FWorkersCount);

End;

{ TBufferHandler }

Function TBufferHandler.CheckInput(Const InputType: TTypes): Boolean;

Const

GOOD\_KEYS: Set Of Char = ['0' .. '9'];

Var

Status: Boolean;

I: Integer;

CountOfMinuses: Integer;

CountOfCommas: Integer;

Begin

Status := True;

Case InputType Of

TpInteger:

Begin

CountOfMinuses := CountSymbol('-');

If (CountOfMinuses = 0) Then

Begin

For I := Low(FEditText) To High(FEditText) Do

If Status And Not(FEditText[I] In GOOD\_KEYS) Then

Status := False;

End

Else If (CountOfMinuses = 1) Then

Begin

For I := 2 To High(FEditText) Do

If Status And Not(FEditText[I] In GOOD\_KEYS) Then

Status := False;

End

Else

Status := False;

End;

TpUInteger:

Begin

For I := Low(FEditText) To High(FEditText) Do

If Status And Not(FEditText[I] In GOOD\_KEYS) Then

Status := False;

End;

TpReal:

Begin

CountOfMinuses := CountSymbol('-');

CountOfCommas := CountSymbol(',');

If (CountOfMinuses = 0) And (CountOfCommas = 0) Then

Begin

For I := Low(FEditText) To High(FEditText) Do

If Status And Not(FEditText[I] In GOOD\_KEYS) Then

Status := False;

End

Else If (CountOfMinuses = 1) And (CountOfCommas = 0) Then

Begin

For I := 2 To High(FEditText) Do

If Status And Not(FEditText[I] In GOOD\_KEYS) Then

Status := False;

End

Else If (CountOfMinuses = 0) And (CountOfCommas = 1) Then

Begin

For I := 1 To High(FEditText) Do

If Status And

Not((FEditText[I] In GOOD\_KEYS) Or

(FEditText[I] = ',')) Then

Status := False;

End

Else If (CountOfMinuses = 1) And (CountOfCommas = 1) Then

Begin

For I := 2 To High(FEditText) Do

If Status And

Not((FEditText[I] In GOOD\_KEYS) Or

(FEditText[I] = ',')) Then

Status := False;

End

Else

Status := False;

End;

TpString:

Begin

// why?

End;

End;

CheckInput := Status;

End;

Function TBufferHandler.CountSymbol(Const Symbol: Char): Integer;

Var

I: Integer;

Count: Integer;

Begin

Count := 0;

For I := 1 To Length(FEditText) Do

If FEditText[I] = Symbol Then

Inc(Count);

CountSymbol := Count;

End;

Procedure TBufferHandler.DeleteLeadingZeros(Const InputType: TTypes);

Begin

Case InputType Of

TpInteger:

Begin

If CountSymbol('-') = 1 Then

While (Length(FEditText) > 1) And (FEditText[2] = '0') Do

Delete(FEditText, 2, 1)

Else

While (Length(FEditText) > 0) And (FEditText[1] = '0') Do

Delete(FEditText, 1, 1);

End;

TpUInteger:

While (Length(FEditText) > 0) And (FEditText[1] = '0') Do

Delete(FEditText, 1, 1);

TpReal:

// TODO

;

TpString:

// Why?

;

End;

End;

{ TFileReader }

Procedure TFileReader.CheckFile();

Begin

If Not FileExists(FileName) Then

FFileStatus := FsNotFound

Else If Not IsFileReadable Then

FFileStatus := FsNotReadable

Else

FFileStatus := FsGood;

End;

Function TFileReader.IsFileReadable(): Boolean;

Var

Status: Boolean;

Begin

Status := True;

Try

Reset(FInFile);

CloseFile(FInFile);

Except

Status := False;

End;

IsFileReadable := Status;

End;

Function TFileReader.IsFileTxt(): Boolean;

Var

FileType: String;

Status: Boolean;

Begin

FileType := FileName.Substring(FileName.Length - 4);

If FileType = '.txt' Then

Status := True

Else

Status := False;

IsFileTxt := Status;

End;

Function TFileReader.ReadTable: TWorkers;

Var

I: Integer;

Worker: TWorker;

Workers: TWorkers;

Begin

I := 0;

Workers := TWorkers.Create();

Try

ReSet(FInFile);

While Not EOF(FINFile) Do

Begin

Read(FInFile, Worker);

Workers[I] := Worker;

Inc(I);

End;

CloseFile(FInFile);

Except

FFileStatus := FsUnexpected;

End;

ReadTable := Workers;

End;

Procedure TFileReader.SetFileName(Const FFileName: String);

Begin

Self.FFileName := FFileName;

Assign(FInFile, Self.FFileName);

End;

{ TFileWriter }

Procedure TFileWriter.CheckFile;

Begin

If Not FileExists(FileName) Then

FFileStatus := FsNotFound

Else If Not IsFileWritable() Then

FFileStatus := FsNotWritable

Else

FFileStatus := FsGood;

End;

Function TFileWriter.IsFileWritable(): Boolean;

Var

Status: Boolean;

Begin

Status := True;

If FTypeFile = TfText Then

Begin

Try

Rewrite(FOutFileTxt);

CloseFile(FOutFiletxt);

Except

Status := False;

End;

End

Else

Begin

Try

Rewrite(FOutFile);

CloseFile(FOutFile);

Except

Status := False;

End;

End;

IsFileWritable := Status;

End;

procedure TFileWriter.ResetFile;

begin

Try

ReWrite(FOutFileTxt);

CloseFile(FOutFileTxt);

Except

FFileStatus := FsUnexpected;

End;

end;

Procedure TFileWriter.SaveTable(Const Workers: TWorkers);

Var

I: Integer;

Worker: TWorker;

Begin

Try

ReWrite(FOutFile);

For I := 1 To Workers.Count Do

Begin

Worker := Workers[I - 1];

Write(FOutFile, Worker);

End;

CloseFile(FOutFile);

Except

FFileStatus := FsUnexpected;

End;

End;

Procedure TFileWriter.SetFileName(Const FFileName: String);

Begin

Self.FFileName := FFileName;

If FTypeFile = TfText Then

Assign(FOutFileTxt, Self.FFileName)

Else

Assign(FOutFile, Self.FFileName);

End;

Procedure TFileWriter.WriteStr(Const Str: String);

Begin

Try

Append(FOutFileTxt);

Write(FOutFileTxt, Str);

CloseFile(FOutFileTxt);

Except

FFileStatus := FsUnexpected;

End;

End;

Procedure TFileWriter.WriteStrln(Const Str: String);

Begin

Try

Append(FOutFileTxt);

Writeln(FOutFileTxt, Str);

CloseFile(FOutFileTxt);

Except

FFileStatus := FsUnexpected;

End;

End;

Constructor TFileWriter.Create(Const TypeFile: TTypeFile);

Begin

FTypeFile := TypeFile;

End;

Function TFileWriter.IsFileTxt: Boolean;

Var

FileType: String;

Status: Boolean;

Begin

FileType := FileName.Substring(FileName.Length - 4);

If FileType = '.txt' Then

Status := True

Else

Status := False;

IsFileTxt := Status;

End;

End.

**Код программы СSharp:**

**ErrorMesagges.cs**

enum ErrorMesagges  
{  
 EmWrongType,  
 EmWrongBoundOfNumber,  
 EmWrongWorkerId,  
 EmWrongKey,  
 EmEmptyTable,  
 EmCompanyNotFound  
}

**FileStatus.cs**  
  
public enum FileStatus  
{  
 FsGood,  
 FsNotFound,  
 FsNotTxt,  
 FsNotReadable,  
 FsNotWritable,  
 FsUnexpacted  
}

**MenuStatus.cs**

enum MenuStatus  
{  
 MsShowTable = 1,  
 MsDownloadTable,  
 MsSaveTable,  
 MsSaveDopInfo,  
 MsExit  
}

**TableMenuStatus.cs**

enum TableMenuStatus  
{  
 TmsAddWorker = 1,  
 TmsDeleteWorker,  
 TmsChangeWorker,  
 TmsExitToMainMenu  
}

**Worker.cs**

public record Worker()  
{  
 public int WId { get; set; }  
 public string? WSurname { get; set; }  
 public string? WCompany { get; set; }  
 public int WCountOfDetailsA { get; set; }  
 public int WCountOfDetailsB { get; set; }  
 public int WCountOfDetailsC { get; set; }  
};

**Workers.cs**

public class Workers  
{  
 private List<Worker> listOfWorkers = new List<Worker>();  
  
 public Worker this[int i]  
 {  
 get => listOfWorkers[i];  
 set => listOfWorkers[i] = value;  
 }  
  
 public int Count => listOfWorkers.Count;  
  
 public void ChangeWorker(int id, Worker worker)  
 {  
 worker.WId = id - 1;  
 listOfWorkers[id - 1] = worker;  
 }  
  
 public void AddWorker(Worker worker)  
 {  
 worker.WId = listOfWorkers.Count;  
 listOfWorkers.Add(worker);  
 }  
  
 public void DeleteWorker(int id)  
 {  
 Worker worker;  
 listOfWorkers.Remove(listOfWorkers[id - 1]);  
 // move all indexes after we were removed worker with id  
 for (int i = id; i < listOfWorkers.Count; i++)  
 {  
 listOfWorkers[i].WId = i + 1;  
 }  
 }  
  
}

**Program.cs**  
  
int choose;  
  
MainMenu mainMenu = new MainMenu();  
Workers workers = new Workers();  
  
mainMenu.ShowProgramInfo();  
do  
{  
 mainMenu.ShowMainMenu();  
 choose = mainMenu.InputChoose();  
 switch ((MenuStatus)choose)  
 {  
 case MenuStatus.MsShowTable:  
 {  
 do  
 {  
 // show table  
 mainMenu.ShowTable(workers);  
 // new request  
 mainMenu.ShowTableMenu();  
 choose = mainMenu.InputChoose();  
 switch ((TableMenuStatus)choose)  
 {  
 case TableMenuStatus.TmsAddWorker:  
 {  
 workers.AddWorker(mainMenu.InputWorker());  
 }  
 break;  
 case TableMenuStatus.TmsDeleteWorker:  
 {  
 if (workers.Count > 0)  
 workers.DeleteWorker(mainMenu.InputId(workers.Count));  
 else  
 mainMenu.ShowErrorEmptyMessage();  
 }  
 break;  
 case TableMenuStatus.TmsChangeWorker:  
 {  
 if (workers.Count > 0)  
 workers.ChangeWorker(mainMenu.InputId(workers.Count),

mainMenu.InputWorker());  
 else  
 mainMenu.ShowErrorEmptyMessage();  
 }  
 break;  
 case TableMenuStatus.TmsExitToMainMenu:  
 {  
 mainMenu.ShowTableMenuExitMessage();   
 }  
 break;   
 default:  
 {  
 mainMenu.ShowErrorKeyMessage();  
 choose = 0; // to continue loop  
 }  
 break;  
 }  
 //Console.Clear();   
 } while ((-1 < choose) &&

(choose < (int)TableMenuStatus.TmsExitToMainMenu));  
 }  
 break;  
 case MenuStatus.MsDownloadTable:  
 {  
 FileReader fileReader = new FileReader();  
 do  
 {  
 fileReader.FilePath = mainMenu.InputFilePath();  
 // reading file to get workers  
 if (fileReader.FileStatus == FileStatus.FsGood)  
 {  
 // output info about starting reading  
 mainMenu.LoadDataMessage();  
 // launch this process  
 workers = fileReader.ReadTable();  
 }  
 // showing fileStatus message for user   
 mainMenu.ShowFileStatusMessage(fileReader.FileStatus);  
 } while (fileReader.FileStatus != FileStatus.FsGood);  
 }  
 break;  
 case MenuStatus.MsSaveTable:  
 {  
 FileWriter fileWriter = new FileWriter();  
 do  
 {  
 fileWriter.FilePath = mainMenu.InputFilePath();  
 // writing workers to file  
 if (fileWriter.FileStatus == FileStatus.FsGood)  
 fileWriter.WriteWorkers(workers);  
 // showing fileStatus message for user  
 mainMenu.ShowFileStatusMessage(fileWriter.FileStatus);  
 } while (fileWriter.FileStatus != FileStatus.FsGood);  
 }  
 break;  
 case MenuStatus.MsSaveDopInfo:  
 {  
 FileWriter fileWriter = new FileWriter();  
 do  
 {  
 fileWriter.FilePath = mainMenu.InputFilePath();  
 // writing dop info into file  
 if (fileWriter.FileStatus == FileStatus.FsGood)  
 fileWriter.WriteDopInfo(mainMenu.InputPrices(),

mainMenu.InputCompany(workers),workers);  
 // showing fileStatus message for user  
 mainMenu.ShowFileStatusMessage(fileWriter.FileStatus);  
 } while (fileWriter.FileStatus != FileStatus.FsGood);  
 }  
 break;  
 case MenuStatus.MsExit:  
 {  
 mainMenu.ShowProgramExitMessage();  
 }  
 break;  
 default:  
 {  
 mainMenu.ShowErrorKeyMessage();  
 choose = 0; // to continue loop  
 }  
 break;  
 }  
 //Console.Clear();   
} while ((-1 < choose) && (choose < (int)MenuStatus.MsExit));

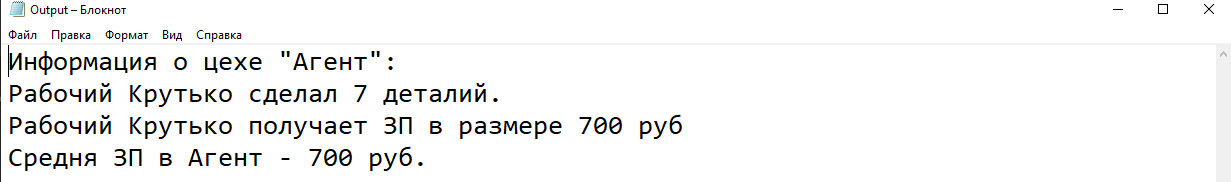
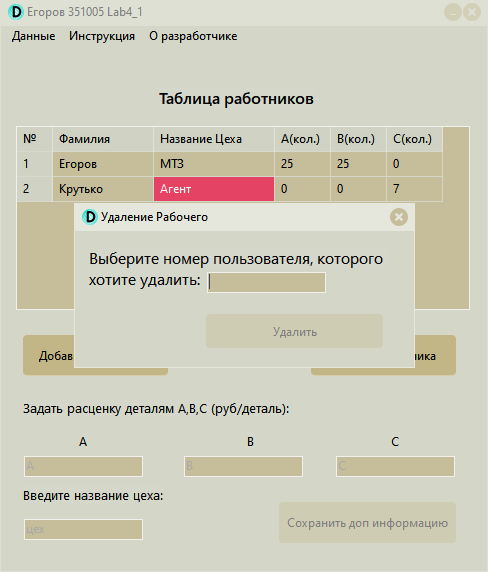
**FileReader.cs**

using Microsoft.VisualBasic;  
  
namespace Lab4\_1;  
  
public class FileReader  
{  
 public FileReader()  
 {  
 filePath = null;  
 }  
   
   
 private string? filePath;  
  
 public string? FilePath  
 {  
 get => filePath;  
 set => filePath = value;  
 }  
  
 private FileStatus fileStatus;  
  
 public FileStatus FileStatus  
 {  
 get => CheckFileStatus();  
 private set => fileStatus = value;  
 }  
  
 private FileStatus CheckFileStatus()  
 {  
 FileInfo fileInfo = new FileInfo(filePath);  
 if (!fileInfo.Exists)  
 {  
 fileStatus = FileStatus.FsNotFound;  
 }  
 else if (!filePath.EndsWith(".txt"))  
 {  
 fileStatus = FileStatus.FsNotTxt;  
 }  
 // else if (!fileInfo.IsReadOnly)  
 // {  
 // fileStatus = FileStatus.FsNotReadable;  
 // }  
 else if (fileStatus != FileStatus.FsUnexpacted)  
 {  
 fileStatus = FileStatus.FsGood;  
 }  
 return fileStatus;  
 }  
   
 public Workers ReadTable()  
 {  
 Workers workers = new Workers();  
 Worker worker = new Worker();  
 String? line = null;  
 int details = 0;  
 const char splitter = ' ';  
 try  
 {  
 using (StreamReader reader = new StreamReader(filePath))  
 {  
 try  
 {  
 while ((line = reader.ReadLine()) != null)  
 {  
 // split for 5 string our line  
 String[] strWorker = line.Split(splitter);  
 // writing data to worker  
 worker.WSurname = strWorker[0];  
 worker.WCompany = strWorker[1];  
 if (int.TryParse(strWorker[2], out details))  
 {  
 worker.WCountOfDetailsA = details;  
 }  
 else  
 {  
 fileStatus = FileStatus.FsUnexpacted;  
 }  
 if (int.TryParse(strWorker[2], out details))  
 {  
 worker.WCountOfDetailsB = details;  
 }  
 else  
 {  
 fileStatus = FileStatus.FsUnexpacted;  
 }  
 if (int.TryParse(strWorker[2], out details))  
 {  
 worker.WCountOfDetailsC = details;  
 }  
 else  
 {  
 fileStatus = FileStatus.FsUnexpacted;  
 }  
 workers.AddWorker(worker);  
 }  
 }  
 catch (IOException e)  
 {  
 fileStatus = FileStatus.FsUnexpacted;  
 }  
 }  
 }  
 catch (IOException e)  
 {  
 fileStatus = FileStatus.FsNotReadable;  
 }  
 return workers;  
 }  
}

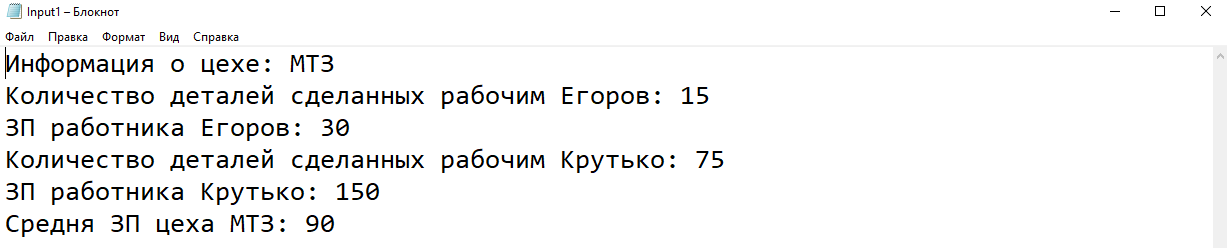
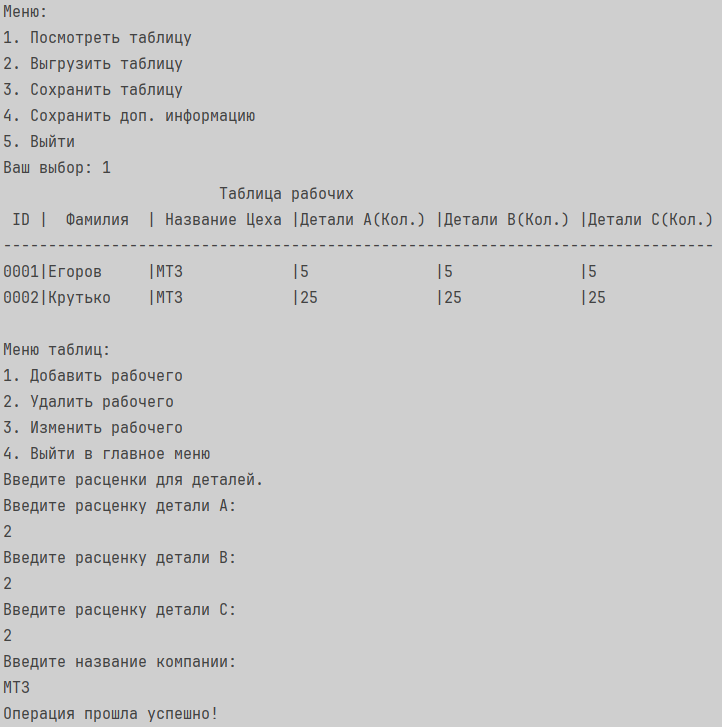
**FileWriter.cs**  
  
public class FileWriter  
{  
 private string? filePath;  
  
 public string? FilePath  
 {  
 get => filePath;  
 set => filePath = value;  
 }  
  
 private FileStatus fileStatus;  
  
 public FileStatus FileStatus  
 {  
 get => CheckFileStatus();  
 private set => fileStatus = value;  
 }  
  
 private FileStatus CheckFileStatus()  
 {  
 FileInfo fileInfo = new FileInfo(filePath);  
 if (!fileInfo.Exists)  
 {  
 fileStatus = FileStatus.FsNotFound;  
 }  
 else if (!filePath.EndsWith(".txt"))  
 {  
 fileStatus = FileStatus.FsNotTxt;  
 }  
 // TODO: добавь fileWritable?  
 else if (fileStatus != FileStatus.FsUnexpacted)  
 {  
 fileStatus = FileStatus.FsGood;  
 }  
 return fileStatus;  
 }  
   
   
 public void WriteWorkers(Workers workers)  
 {  
 Worker worker;  
 using StreamWriter writer = new StreamWriter(filePath,false);  
 for (int i = 0; i < workers.Count; i++)  
 {  
 worker = workers[i];  
 try  
 {  
 writer.WriteLine(worker.WSurname + " " +  
 worker.WCompany + " " +  
 worker.WCountOfDetailsA + " " +  
 worker.WCountOfDetailsB + " " +  
 worker.WCountOfDetailsC + " ");  
 }  
 catch (IOException e)  
 {  
 fileStatus = FileStatus.FsUnexpacted;  
 }  
 }  
 }  
  
 public void WriteDopInfo(int[] price, String? company, Workers workers)  
 {  
 Worker worker;  
 double allWorkersSalaryOfCompany = 0;  
 int salaryOfWorker = 0;  
 int counterOfWorkersOfCompany = 0;  
 try  
 {  
 using StreamWriter writer = new StreamWriter(filePath, false);  
 {  
 try  
 {  
 writer.WriteLine("Информация о цехе: " + company);  
 }  
 catch (IOException e)  
 {  
 fileStatus = FileStatus.FsUnexpacted;  
 }  
 for (int i = 0; i < workers.Count; i++)  
 {  
 worker = workers[i];  
 // finding a worker which works in this company  
 if (worker.WCompany.Equals(company))  
 {  
 counterOfWorkersOfCompany++;  
 salaryOfWorker = worker.WCountOfDetailsA \* price[0] +  
 worker.WCountOfDetailsB \* price[1] +  
 worker.WCountOfDetailsC \* price[2];  
 allWorkersSalaryOfCompany += (double)salaryOfWorker;  
 try  
 {  
 // writing summary count of details has done by worker  
 writer.WriteLine("Количество деталей сделанных рабочим " + worker.WSurname + ": " +  
 (worker.WCountOfDetailsA + worker.WCountOfDetailsB +  
 worker.WCountOfDetailsC));  
 // writing worker's salary  
  
 writer.WriteLine("ЗП работника " + worker.WSurname + ": " + salaryOfWorker);  
 }  
 catch (IOException e)  
 {  
 fileStatus = FileStatus.FsUnexpacted;  
 }  
 }  
 }  
 try  
 {  
 // writing Avg salary of company  
 writer.WriteLine("Средня ЗП цеха " + company + ": " +  
 (allWorkersSalaryOfCompany / (double)counterOfWorkersOfCompany));  
 }  
 catch (IOException e)  
 {  
 fileStatus = FileStatus.FsNotWritable;  
 }  
 }  
 }  
 catch (IOException e)  
 {  
 fileStatus = FileStatus.FsUnexpacted;  
 }  
 }  
}

**Скриншоты:**

**Delphi:**



**СSharp:**



**Блок-схема:**

