*Минобрнауки России*  
*федеральное государственное бюджетное образовательное учреждение*   
*высшего образования*  
*«Санкт-Петербургский государственный технологический институт (технический университет)»*

Факультет информационных технологий и управления

Кафедра систем автоматизированного проектирования и управления

Дисциплина: Лингвистическое и программное обеспечение автоматизированных систем

Отчет по лабораторной работе №2

Вариант 5

# «Обработка данных»

Исполнител

студент гр. 465 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ В. А. Киселев

(подпись)

Преподаватели \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ И. Г. Корниенко

(подпись) А. К. Федин

Санкт-Петербург  
2019

1. **Постановка задачи:**

По данным, имеющимся в журнале покупок бензина, напечатайте разнообразную контрольную статистику, показывающую водителю, во что обходится эксплуатация автомобиля. Исходные данные о каждой покупке — это дата, марка бензина, показание счетчика пройденного пути, цена одного литра, сколько литров куплено и общая стоимость. Выводимая информация должна повторять исходную и, кроме того, включать в себя такие показатели, как пробег между заправками, пробег на один литр, стоимость пробега в один километр, стоимость одного галлона, стоимость одного дня, время расходования одного галлона. Все указанные показатели следует получать для каждой заправки и усреднять двумя способами: за небольшой срок и за все время наблюдений. Кроме того, соберите данные по каждой марке бензина и напечатайте соответствующие средние значения. Не ограничивайте число различных марок

1. **Исходные данные:**

В качестве исходных данных выступает csv файл с определенной структурой.

1. **Особые ситуации:**

Рассмотреть ситуации, когда отсутствует файл, куда/откуда будет считываться информация. Также следует учесть контроль за форматом вводимых пользователем данных, если формат неверен - сообщить об этом пользователю.

1. **Алгоритм решения задачи:**

Загружаем исходные данные. Далее происходит предобработка данных, приведение к необходимым типам данных, применение скользящего среднего и т.д., далее происходит расчет необходимых значений.

1. **Блок-схема**

****

Рисунок 1 - Блок-схема программы

****

Рисунок 2 – Блок-схема алгоритма

1. **Форматы представления данных:**

Таблица 1 – Переменные, используемые в программе.

|  |  |  |
| --- | --- | --- |
| **Имя** | **Тип** | **Описание** |
| table | List<array<String^>^>^ | Отвечает за хранение и обработку исходных данных |
| dataMarkFuel | Dictionary<String^, array<double>^>^ | Используется для хранения данных, рассчитанных для марок бензина |
| preparedate | List<array<double>^>^ | Используется для хранения обработанных данных |
| fileName | string | Путь к файлу (имя файла) |
| clc | Calculate | Отвечает за запуск и работу алгорима |

1. **Структура программы:**

Программа поделена на несколько модулей:

* MyForm
* OpenFiles
* Calculate
* ReadCSV

Модули, в свою очередь, разделены на функции:

Таблица 2 – Методы модуля MyForm

|  |  |
| --- | --- |
| **Имя** | **Описание** |
| MyForm | Точка входа в программу |
| LoadData\_Click | Загрузка данных |
| Render | Вывод данных в интерфейс |
| Validate | Валидация входных данных |
| buttonAddRow\_Click | Добавление новой строки в исходные данные |
| SaveSoirceData\_Click | Сохранение исходных данных |
| SavePrepareData\_Click | Сохранение обработанных данных |
| SaveNarkFuelData\_Click | Сохранение данных по маркам бензина |

Таблица 3 – Методы класса Calculate

|  |  |
| --- | --- |
| **Имя** | **Описание** |
| Work | Предобработка данных |
| GetAvgMarkFuel | Получение обработанных данных для марок бензина |
| GetPrepareData | Получение обработанных данных |
| ToAvgRoll | Применение скользящего среднего к стлобцам |
| DiffDay | Вычисление разницы между датами |
| GetMA | Функция скользящего среднего |

Таблица 4 – Методы класса ReadCSV

|  |  |
| --- | --- |
| **Имя** | **Описание** |
| OpenFile | Чтение из источника |

Таблица 5 – Методы класса OpenFiles

|  |  |
| --- | --- |
| **Имя** | **Описание** |
| write | Сохранение данных в файл |

1. **Описание хода выполнения работы:**

1. Обсуждение задания, а также крайних случаев, которые могут встретится при его выполнении с преподавателями.

2. В качестве основы был использован проект по работе с текстом, который был разработан в процессе обучения предмету «Программирование» на 1 курсе.

3. В качестве среды разработки была использована Visual Studio 2019.

4. Была переработана архитектура программной системы с учётом знаний, полученных на курсе «Лингвистическое и программное обеспечение автоматизированных систем».

5. Был разработан алгоритм обработки данных.

7. Составление отчёта.

1. **Результаты работы программы:**

В результате работы программа выводит рассчитанные значения по всем данным за промежуток времени и так же общую статистику по маркам бензина.

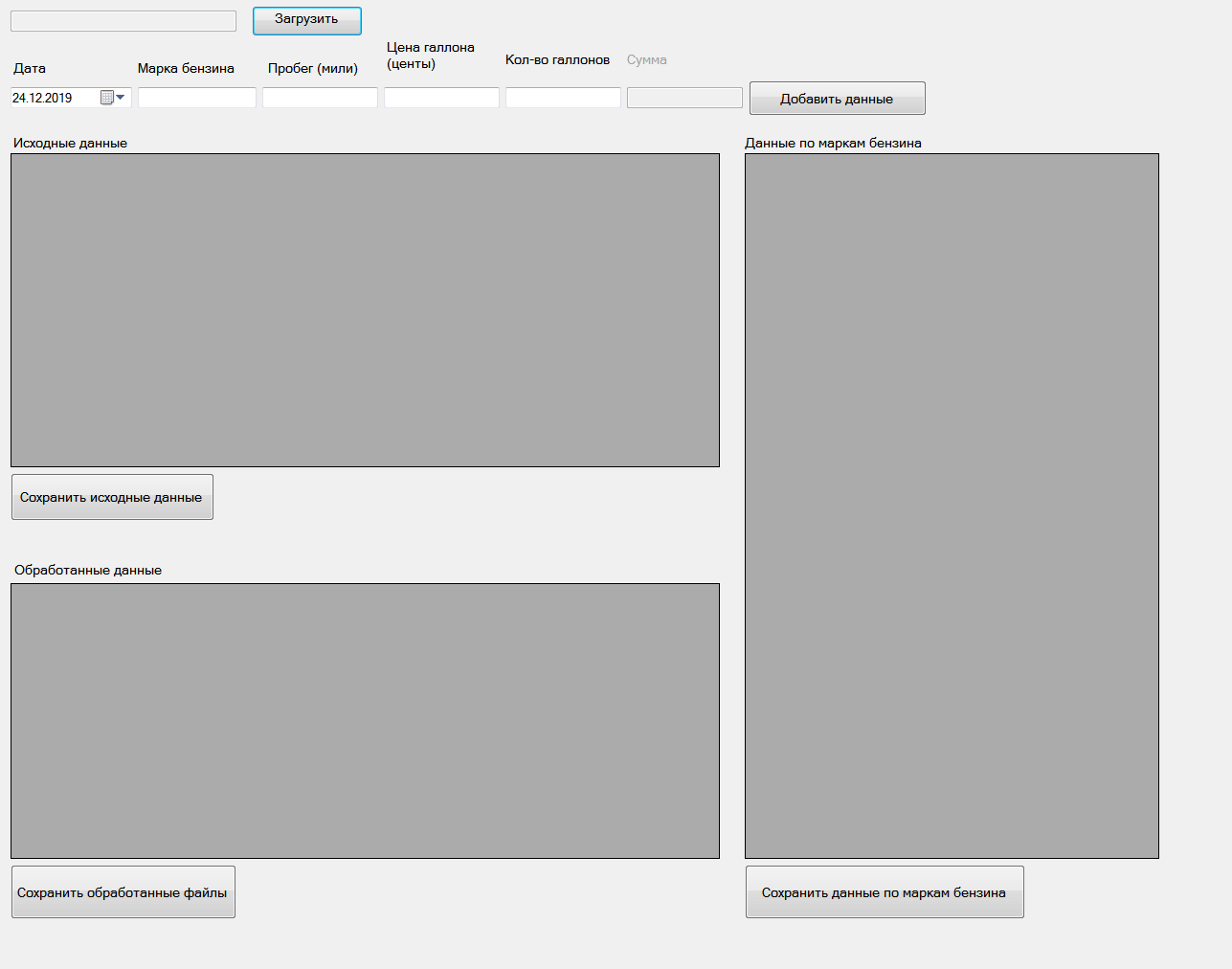


Рисунок 2 – Главное меню программы

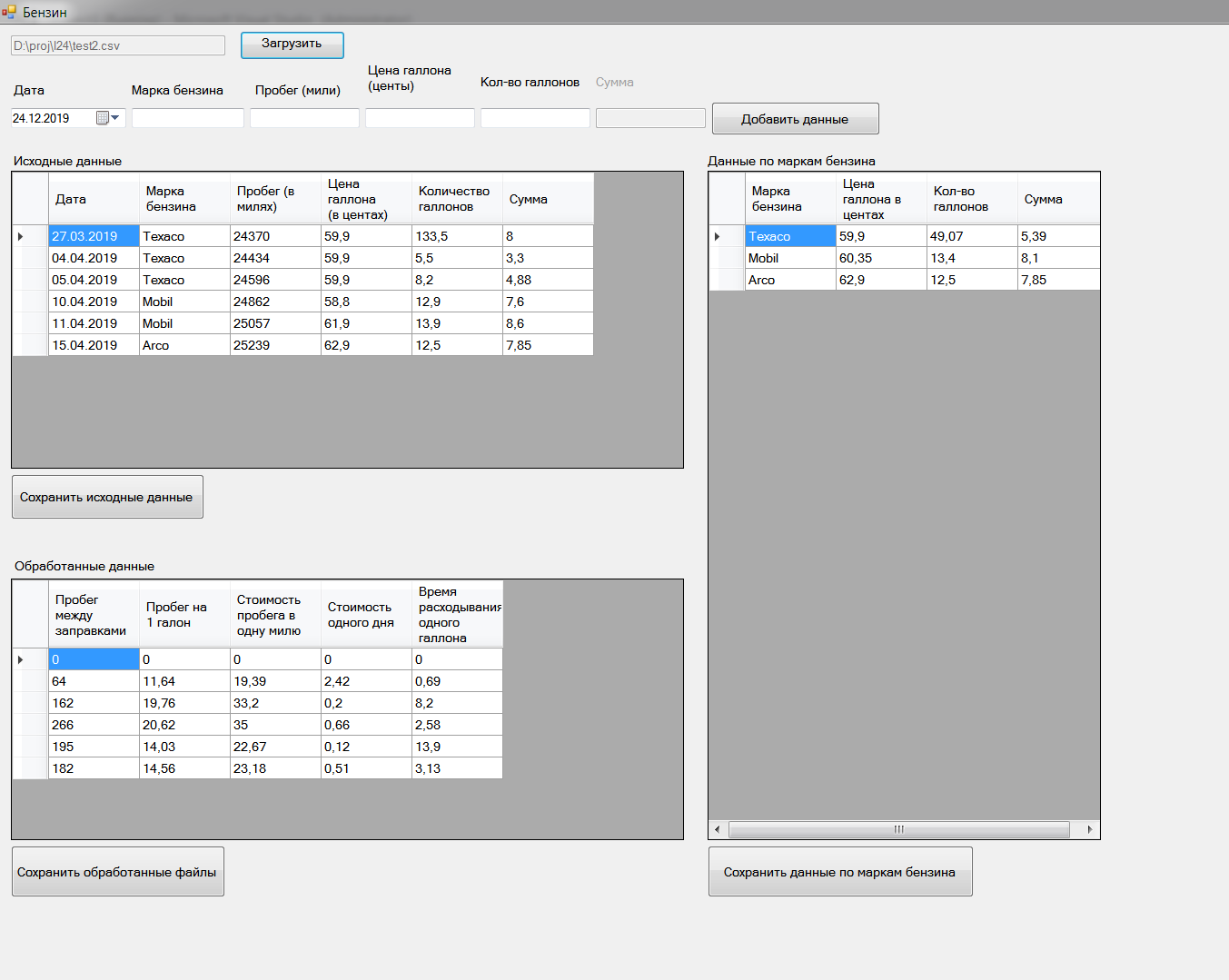


Рисунок 3 – Результат работы программы

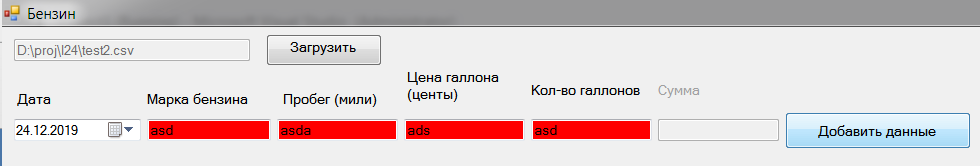


Рисунок 4 – Обработка ошибок пользовательского ввода

1. **Исходный текст программы:**

**----------------------------** **MyForm.h----------------------------**

#pragma once

#include "ReadCSV.h"

#include "Calculate.h"

#include <locale>

namespace Project1 {

using namespace System;

using namespace System::ComponentModel;

using namespace System::Collections;

using namespace System::Windows::Forms;

using namespace System::Data;

using namespace System::Drawing;

/// <summary>

/// Summary for MyForm

/// </summary>

public ref class MyForm : public System::Windows::Forms::Form

{

public:

List<array<String^>^>^ table;

Dictionary<String^, array<double>^>^ dataMarkFuel;

List<array<double>^>^ preparedate;

MyForm(void)

{

InitializeComponent();

//

//TODO: Add the constructor code here

//

}

protected:

/// <summary>

/// Clean up any resources being used.

/// </summary>

~MyForm()

{

if (components)

{

delete components;

}

}

private: System::Windows::Forms::OpenFileDialog^ openFileDialog1;

private: System::Windows::Forms::Button^ loadData;

private: System::Windows::Forms::TextBox^ textBox2;

private: System::Windows::Forms::DataGridView^ SourceDataGrid;

private: System::Windows::Forms::DataGridView^ PrepDataView;

private: System::Windows::Forms::DataGridView^ MarkData;

private: System::Windows::Forms::Label^ label1;

private: System::Windows::Forms::Label^ label2;

private: System::Windows::Forms::Label^ label3;

private: System::Windows::Forms::Button^ button2;

private: System::Windows::Forms::Button^ button3;

private: System::Windows::Forms::Button^ button4;

private: System::Windows::Forms::TextBox^ textBoxDate;

private: System::Windows::Forms::TextBox^ textBoxMark;

private: System::Windows::Forms::TextBox^ textBoxMileage;

private: System::Windows::Forms::TextBox^ textBoxPrice;

private: System::Windows::Forms::TextBox^ textBoxCount;

private: System::Windows::Forms::Label^ label4;

private: System::Windows::Forms::Label^ label5;

private: System::Windows::Forms::Label^ label6;

private: System::Windows::Forms::Label^ label7;

private: System::Windows::Forms::Label^ label8;

private: System::Windows::Forms::Label^ label9;

private: System::Windows::Forms::TextBox^ textBoxSum;

private: System::Windows::Forms::Button^ buttonAddRow;

protected:

private:

/// <summary>

/// Required designer variable.

/// </summary>

System::ComponentModel::Container ^components;

#pragma region Windows Form Designer generated code

#pragma endregion

private: System::Void LoadData\_Click (System::Object^ sender, System::EventArgs^ e) {

/\*

\* Считывание исходных данных

\*/

if (this->openFileDialog1->ShowDialog() == System::Windows::Forms::DialogResult::OK)

{

System::String^ filepath = this->openFileDialog1->FileName;

this->textBox2->Text = filepath;

ReadCSV csv;

//List<array<String^>^>^ table = csv.OpenFile(filepath->ToString(), ';');

this->table = csv.OpenFile(filepath->ToString(), ';');

//this->textBox1->Text = System::IO::File::ReadAllText(this->textBox2->Text);

this->Render();

}

}

private: void Render() {

Calculate clc(this->table);

clc.work();

this->dataMarkFuel = clc.getAvgMarkFuel(clc.table\_prepare, clc.fuel\_codes);

preparedate = clc.getPrepareData(clc.table\_prepare);

//SourceDataGrid->DataSource = table->ToArray();

array<String^>^ headers = this->table[0];

SourceDataGrid->Rows->Clear();

for each (array<String^>^ row in this->table)

{

if (row[0] == "date")

{

SourceDataGrid->ColumnCount = row->Length;

for (int i = 0; i < row->Length; i++)

{

SourceDataGrid->Columns[i]->Name = row[i];

//SourceDataGrid->Columns[i].Name = row[i];

}

continue;

}

array<String^>^ tmp = {

row[0],

row[1],

row[2],

row[3],

row[4],

row[5],

};

SourceDataGrid->Rows->Add(tmp);

}

PrepDataView->ColumnCount = 5;

PrepDataView->Columns[0]->Name = "Пробег между заправками";

PrepDataView->Columns[1]->Name = "Пробег на 1 галон";

PrepDataView->Columns[2]->Name = "Стоимость пробега в одну милю";

PrepDataView->Columns[3]->Name = "Стоимость одного дня";

PrepDataView->Columns[4]->Name = "Время расходывания одного галлона";

PrepDataView->Rows->Clear();

for each (array<double>^ row in preparedate)

{

array<double^>^ tmp = {

row[0],

row[1],

row[2],

row[3],

row[4],

};

PrepDataView->Rows->Add(tmp);

}

MarkData->ColumnCount = 4;

MarkData->Columns[0]->Name = "Марка бензина";

MarkData->Columns[1]->Name = "Цена галлона в центах";

MarkData->Columns[2]->Name = "Кол-во галлонов";

MarkData->Columns[3]->Name = "Сумма";

MarkData->Rows->Clear();

for each (auto marks in this->dataMarkFuel)

{

array<String^>^ tmp = {

marks.Key,

marks.Value[0].ToString(),

marks.Value[1].ToString(),

marks.Value[2].ToString(),

};

MarkData->Rows->Add(tmp);

}

}

private: bool Validate() {

int countGood = 6;

try

{

DateTime::Parse(textBoxDate->Text);

}

catch (System::FormatException ^ e)

{

textBoxDate->BackColor = Color::FromName("Red");

countGood--;

}

double tmp;

if (!(Double::TryParse(textBoxMileage->Text, tmp))) {

textBoxMark->BackColor = Color::FromName("Red");

countGood--;

}

if (!(Double::TryParse(textBoxMileage->Text, tmp))) {

textBoxMileage->BackColor = Color::FromName("Red");

countGood--;

}

if (!(Double::TryParse(textBoxPrice->Text, tmp))) {

textBoxPrice->BackColor = Color::FromName("Red");

countGood--;

}

if (!(Double::TryParse(textBoxCount->Text, tmp))) {

textBoxCount->BackColor = Color::FromName("Red");

countGood--;

}

if (!(Double::TryParse(textBoxSum->Text, tmp))) {

textBoxSum->BackColor = Color::FromName("Red");

countGood--;

}

if (table == nullptr) {

MessageBox::Show("Необходимо загрузить основной файл данных");

countGood--;

}

return countGood == 6;

}

private: System::Void buttonAddRow\_Click(System::Object^ sender, System::EventArgs^ e) {

bool correctValid = this->validate();

if (!correctValid)

return;

array<String^>^ tmp = {

textBoxDate->Text,

textBoxMark->Text,

textBoxMileage->Text,

textBoxPrice->Text,

textBoxCount->Text,

textBoxSum->Text,

};

table->Add(tmp);

this->render();

}

private: System::Void SaveSourceData\_Click (System::Object^ sender, System::EventArgs^ e) {

// Сохранение исходных данных

SaveFileDialog^ sfd = gcnew SaveFileDialog();

if (table == nullptr) {

MessageBox::Show("Нет данных для сохранения");

return;

}

try

{

sfd->Filter = "SourceData|\*.csv";

sfd->FileName = "Source Data";

sfd->Title = "Save Source Data";

if (sfd->ShowDialog() == System::Windows::Forms::DialogResult::OK)

{

String^ path = sfd->FileName;

StreamWriter^ writer = gcnew StreamWriter(File::Create(path));

try

{

for each (array<String^>^ row in table)

{

String^ saveStr = "";

for each (auto field in row)

{

saveStr += field + ";";

}

writer->Write(saveStr + "\t\r\n");

}

}

finally

{

delete writer;

}

}

}

finally

{

delete sfd;

}

}

private: System::Void SavePrepareData\_Click (System::Object^ sender, System::EventArgs^ e) {

//Сохранение обработанных данных

SaveFileDialog^ sfd = gcnew SaveFileDialog();

if (preparedate == nullptr) {

MessageBox::Show("Нет данных для сохранения");

return;

}

try

{

sfd->Filter = "PrepareDate|\*.csv";

sfd->FileName = "Prepare Date";

sfd->Title = "Save Prepare Date";

if (sfd->ShowDialog() == System::Windows::Forms::DialogResult::OK)

{

String^ path = sfd->FileName;

StreamWriter^ writer = gcnew StreamWriter(File::Create(path), System::Text::Encoding::UTF8);

try

{

writer->Write("Пробег между заправками;Пробег на 1 галон;Стоимость пробега в одну милю;Стоимость одного дня;Время расходывания одного галлона\t\r\n");

PrepDataView->Rows->Clear();

for each (array<double>^ row in preparedate)

{

String^ saveStr = "";

for each (auto field in row)

{

saveStr += field + ";";

}

writer->Write(saveStr + "\t\r\n");

}

}

finally

{

delete writer;

}

}

}

finally

{

delete sfd;

}

}

private: System::Void SaveMarkFuelData\_Click (System::Object^ sender, System::EventArgs^ e) {

//Сохранение данных по маркам бензина

SaveFileDialog^ sfd = gcnew SaveFileDialog();

if (dataMarkFuel == nullptr) {

MessageBox::Show("Нет данных для сохранения");

return;

}

try

{

sfd->Filter = "MarkFuel|\*.csv";

sfd->FileName = "Mark Fuel";

sfd->Title = "Save Mark Fuel";

if (sfd->ShowDialog() == System::Windows::Forms::DialogResult::OK)

{

String^ path = sfd->FileName;

StreamWriter^ writer = gcnew StreamWriter(File::Create(path), System::Text::Encoding::UTF8);

try

{

String^ str = "Марка бензина;Цена галлона в центах;Кол-во галлонов;Сумма\t\r\n";

writer->Write(str);

for each (auto row in dataMarkFuel)

{

String^ saveStr = "";

saveStr += row.Key;

saveStr += row.Value[0].ToString(),

saveStr += row.Value[1].ToString(),

saveStr += row.Value[2].ToString(),

writer->Write(saveStr + "\t\r\n");

}

}

finally

{

delete writer;

}

}

}

finally

{

delete sfd;

}

}

};

} **----------------------------Calculate.h----------------------------**

#include <vector>

using namespace System;

using namespace System::IO;

using namespace System::Diagnostics;

using namespace System::Collections::Generic;

using namespace System::Collections;

namespace Project1 {

ref class Calculate

{

public:

List<array<String^>^>^ table;

List<array<double>^>^ table\_prepare;

Dictionary<String^, double>^ fuel\_codes;

Calculate(List<array<String^>^>^ table);

void work();

Dictionary<String^, array<double>^>^ getAvgMarkFuel(List<array<double>^>^ data, Dictionary<String^, double>^ fuel\_dict);

List<array<double>^>^ getPrepareData(List<array<double>^>^ data);

List<array<double>^>^ toAvgRoll(List<array<double>^>^ data, int workIndex);

List<double>^ getMA(List<double>^ x, int n, int size);

double DiffDay(String^ date1, String^ date2);

List<double>^ getMA(List<double> x, int n, int size);

};

}

**----------------------------Calculate.h----------------------------**

#include "Calculate.h"

namespace Project1 {

Calculate::Calculate(List<array<String^>^>^ table)

{

this->table = table;

}

void Calculate::work() {

List<array<String^>^>^ table = Calculate::table;

List<array<double>^>^ table\_double = gcnew List<array<double>^>();

Dictionary<String^, double>^ fuel\_codes = gcnew Dictionary<String^, double>();

double j = 0;

for each (array<String^>^ row in table)

{

if (row[1] == "mark") {

continue;

}

try

{

fuel\_codes->Add(row[1], j);

j++;

}

catch (ArgumentException^)

{

continue;

}

}

Calculate::fuel\_codes = fuel\_codes;

List<double>^ mileage = gcnew List<double>();

int i = 0;

for each (array<String^>^ row in table)

{

if (i == 0) {

i++;

continue;

}

array<String^>^ prevRow = table[i - 1];

double day = DiffDay(i == 1 ? "" : prevRow[0], row[0]);

Trace::WriteLine(row[2]);

array<double>^ tmp = {

day,

fuel\_codes[row[1]],

System::Convert::ToDouble(row[2]),

System::Convert::ToDouble(row[3]),

System::Convert::ToDouble(row[4]),

System::Convert::ToDouble(row[5]),

};

table\_double->Add(tmp);

//mileage->Add(System::Convert::ToDouble(row[2]));

i++;

}

table\_double = toAvgRoll(table\_double, 2);

table\_double = toAvgRoll(table\_double, 3);

table\_double = toAvgRoll(table\_double, 4);

table\_double = toAvgRoll(table\_double, 5);

Calculate::table\_prepare = table\_double;

//List<double>^ rollAvgMileage = getMA(mileage, mileage->Count, 2);

}

Dictionary<String^, array<double>^>^ Calculate::getAvgMarkFuel(List<array<double>^>^ data, Dictionary<String^, double>^ fuel\_dict) {

Dictionary<String^, array<double>^>^ prepData = gcnew Dictionary<String^, array<double>^>();

for each (auto fuel\_mark in fuel\_dict)

{

int count\_mrk = 0;

double price\_gallon = 0;

double count\_gallon = 0;

double summ = 0;

for each (array<double>^ row in data)

{

if (row[1] == fuel\_mark.Value) {

count\_mrk++;

price\_gallon += row[3];

count\_gallon += row[4];

summ += row[5];

}

}

price\_gallon = price\_gallon / count\_mrk;

count\_gallon = count\_gallon / count\_mrk;

summ = summ / count\_mrk;

array<double>^ tmp = {

round(price\_gallon \* 100) / 100,

round(count\_gallon \* 100) / 100,

round(summ \* 100) / 100

};

prepData->Add(fuel\_mark.Key, tmp);

}

return prepData;

}

List<array<double>^>^ Calculate::getPrepareData(List<array<double>^>^ data) {

List<array<double>^>^ prepData = gcnew List<array<double>^>();

int i = 0;

for each (array<double>^ row in data)

{

double prevMileage = 0;

if (i - 1 >= 0)

{

array<double>^ prewRow = data[i - 1];

prevMileage = prewRow[2];

}

double diff\_mileage = row[2] - prevMileage; // пробег между заправками

if (i - 1 < 0) diff\_mileage = 0;

double mileageInGallon = diff\_mileage / row[4]; // пробега на 1 галлон

double priceMileage = diff\_mileage / row[5]; // стоимость 1 мили

double priceDay = row[0] / row[5]; //Стоимость 1 дня

double consumptionGallon = row[4] / row[0]; //Время расходывания одного галона

if (consumptionGallon == INFINITY) consumptionGallon = 0;

array<double>^ tmp = {

round(diff\_mileage \* 100) / 100,

round(mileageInGallon \* 100) / 100,

round(priceMileage \* 100) / 100,

round(priceDay \* 100) / 100,

round(consumptionGallon \* 100) / 100

};

prepData->Add(tmp);

i++;

}

return prepData;

}

List<array<double>^>^ Calculate::toAvgRoll(List<array<double>^>^ data, int workIndex)

{

List<double>^ tmp = gcnew List<double>();

for each (array<double>^ row in data)

{

tmp->Add(row[workIndex]);

}

List<double>^ rollAvgTmp = getMA(tmp, tmp->Count, 2);

/\*for (size\_t i = 0; i < rollAvgMileage->Count; i++)

{

data.

data[i][workIndex] = rollAvgMileage[i];

}\*/

int i = 0;

for each (array<double>^ row in data)

{

row[workIndex] = rollAvgTmp[i];

i++;

}

return data;

}

List<double>^ Calculate::getMA(List<double>^ x, int n, int size) {

// size - количество отсчетов интервала усреднения

double sumx = 0; // сумма отсчетов на интервале

//double \*mas; // массив для хранения size отсчетов

List<double>^ mas = gcnew List<double>(size);

List<double>^ ret = gcnew List<double>();

int index = 0; // индекс элемента массива

//mas = new double[size];

for (int i = 0; i<size; i++) mas->Add(0);

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

sumx -= mas[index];

mas[index] = x[i];

sumx += mas[index];

index++;

if (index >= size)

index = 0; // возврат к началу "окна"

ret->Add(sumx / size);

}

return ret;

}

double Calculate::DiffDay(String^ date1, String^ date2) {

if (date1 == ""|| date2 == "") return 0;

DateTime startTime = DateTime(1970, 1, 1, 0, 0, 0);

double oneDay = 86400;

DateTime dt1 = DateTime();

dt1 = DateTime::Parse(date1);

TimeSpan \_UnixTimeSpan = (dt1 - startTime);

DateTime dt2 = DateTime();

dt2 = DateTime::Parse(date2);

TimeSpan \_UnixTimeSpan2 = (dt2 - startTime);

double diff = \_UnixTimeSpan2.TotalSeconds - \_UnixTimeSpan.TotalSeconds;

return diff / oneDay;

}

}

**----------------------------OpenFiles.h----------------------------**

#pragma once

using namespace System;

using namespace System::ComponentModel;

using namespace System::Collections;

using namespace System::Windows::Forms;

using namespace System::Data;

using namespace System::Drawing;

using namespace System::Collections::Generic;

ref class OpenFiles : public System::Windows::Forms::Form

{

public:

void Write(System::String^ filename, List<array<double>^>^ preparedate);

};

**---------------------------- OpenFiles.cpp----------------------------**

#include "OpenFiles.h"

void OpenFiles::Write(System::String ^ filename, List<array<double>^>^ preparedate)

{

using namespace System::Windows::Forms;

using namespace System::IO;

SaveFileDialog^ sfd = gcnew SaveFileDialog();

if (preparedate == nullptr) {

MessageBox::Show("Нет данных для сохранения");

return;

}

try

{

sfd->Filter = "SourceData|\*.csv";

sfd->FileName = "Source Data";

sfd->Title = "Save Source Data";

if (sfd->ShowDialog() == System::Windows::Forms::DialogResult::OK)

{

String^ path = sfd->FileName;

StreamWriter^ writer = gcnew StreamWriter(File::Create(path));

try

{

for each (array<double>^ row in preparedate)

{

String^ saveStr = "";

for each (auto field in row)

{

saveStr += field + ";";

}

writer->Write(saveStr + "\t\r\n");

}

}

finally

{

delete writer;

}

}

}

finally

{

delete sfd;

}

}

**----------------------------ReadCSV.h----------------------------**

#pragma once

#include <istream>

#include <string>

using namespace System;

using namespace System::IO;

using namespace System::Diagnostics;

using namespace System::Collections::Generic;

namespace Project1 {

class ReadCSV

{

public:

std::string filepath;

List<array<String^>^>^ OpenFile(System::String^ filepath, char sep);

};

}

**---------------------------- ReadCSV.cpp----------------------------**

#include "ReadCSV.h"

namespace Project1 {

List<array<String^>^>^ ReadCSV::OpenFile(System::String^ filepath, char sep) {

System::IO::StreamReader^ sr = gcnew System::IO::StreamReader(filepath);

List<array<String^>^>^ table = gcnew List<array<String^>^>();

while (sr->Peek() > -1)

{

array<String^>^ words;

String^ str = sr->ReadLine();

//Trace::WriteLine(str);

words = str->Split(sep);

table->Add(words);

}

sr->Close();

return table;

}

}