Содержание

[1 Постановка задачи 5](#_Toc72594470)

[2 Глобальные переменные 6](#_Toc72594471)

[3 Переменные модуля main() 6](#_Toc72594472)

[4 Структура записи в файле 6](#_Toc72594473)

[5 Схема иерархии классов и модулей 7](#_Toc72594474)

[6 Спецификация модулей 8](#_Toc72594475)

[7 Таблица тестов и результаты тестирования, результаты работы программы 12](#_Toc72594476)

[8 Руководство для пользователя 20](#_Toc72594477)

[9 Исходный код 21](#_Toc72594478)

Введение

В рамках курсовой работы нужно создать программу “Автоаукцион”, позволяющую обрабатывать файл, содержащий сведения: производитель а/м, марка, год выпуска, цвет, пробег, начальная цена; формировать список автомобилей одной марки, отсортированный по цене; строить диаграмму средней стоимости а/м одной марки в зависимости от года выпуска; выводить список автомашин заданного производителя; отвечать всем требованиям и реализовывать все задания курсовой работы.

Для написания программы с интерфейсом на языке C++ был выбран современный кроссплатформенный фреймворк Qt (версия 5.15.2), предоставляющий удобную работу с визуальными формами, имеющий возможность отрисовки графики. Хорошая документация, известность среди программистов и бесплатность позволили написать программу с удобством и лёгкостью.

Qt позволяет создавать различные типы приложений, в случае программы “Автоаукцион” наиболее предпочтительно создание приложения типа «Qt Widget» – этот тип используется для создания программ на C++ для настольных компьютеров с классическим графическим интерфейсом.

# Постановка задачи

Необходимо создать программу, выдающую записи об автомобилях. Включаются следующие данные:

* Производитель;
* Марка;
* Год выпуска;
* Цвет;
* Пробег;
* Начальная цена.

Программа должна уметь работать с основной БД и пользовательскими файлами; добавлять, редактировать и удалять записи; иметь поиск и сортировку по возрастанию/убыванию каждого критерия таблицы; строить среднегодовую диаграмму цен автомобилей; иметь три вида справки: «Руководство пользователя», «Задание», «О программе».

Из требований необходимо подчеркнуть, что необходимо реализовать проверку правильного чтения и записи в файл (проверка удачного чтения и записи, проверка данных в файле), проверку правильного ввода добавляемых и редактируемых данных, поиск должен производиться даже при неполном вводе данных, пункты меню должны иметь горячие клавиши для быстрого доступа к функциям программы.

# Глобальные переменные

Глобальные переменные отсутствуют, поскольку программа использует классы, и именно в них хранятся переменные, необходимые конкретному окну.

# Переменные модуля main()

QApplication a, MainWindow w – переменные, определённые автоматически при создании нового приложения Qt Widget. Они должны быть созданы для функционирования программы и вывода главного окна.

# Структура записи в файле

Стандартный файл БД (cars.data) и файлы, создаваемые пользователем, идентичны по структуре и представляют из себя перечисление всех данных об автомобиле, разделённых знаком табуляции (\t), причём информация о каждом автомобиле хранится на отдельной строке. Порядок данных в строке идентичен приведённому в задании: производитель, марка, год выпуска, цвет, пробег, начальная цена. Такая запись наглядна и предоставляет возможность использования текста с любыми знаками, кроме знака табуляции (символ \t) и перехода на следующую строку (символ \n).

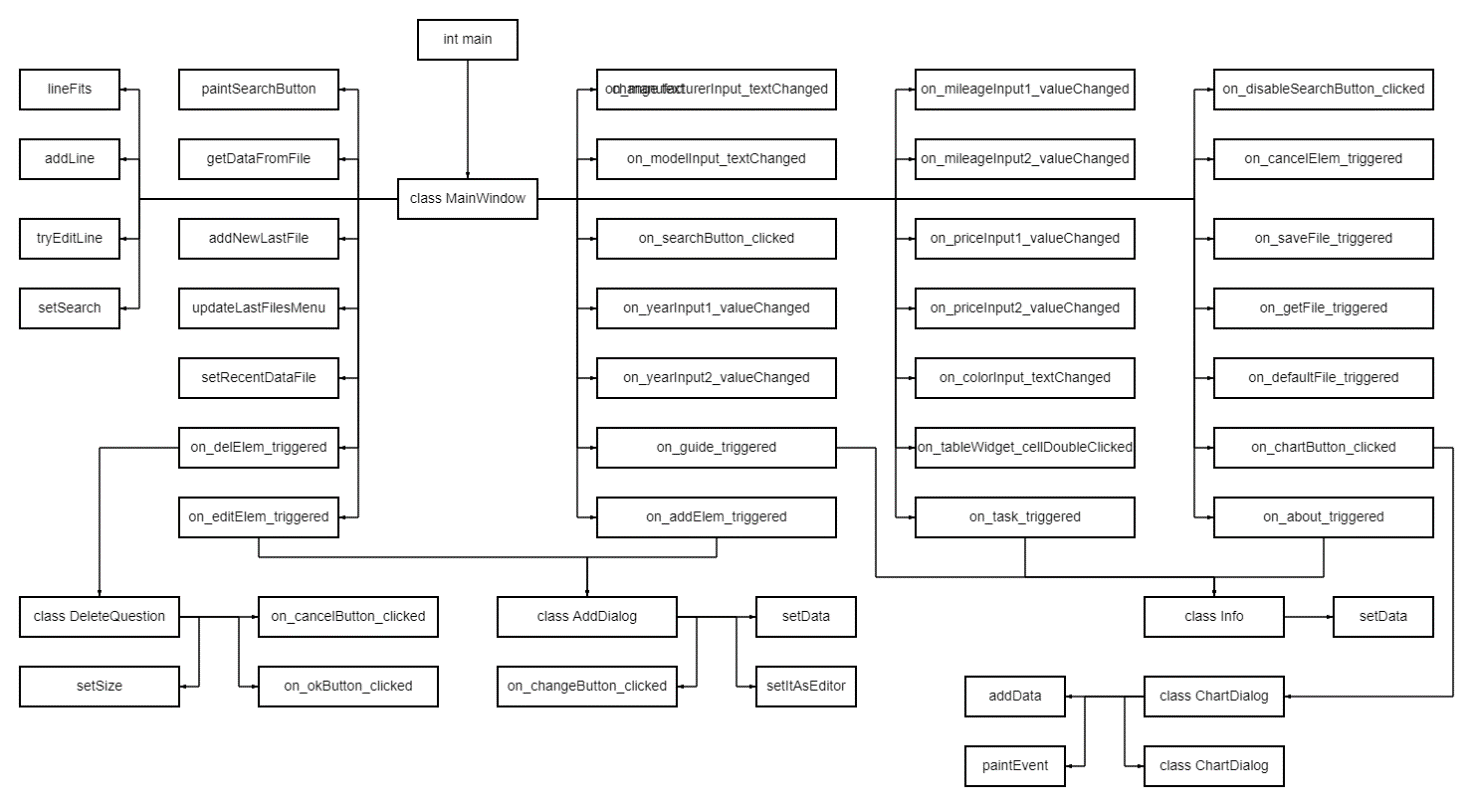
Запись в файл происходит автоматически при добавлении, изменении и удалении данных. Программа устроена таким образом, что изменение файла происходит сразу после добавления, изменения или удаления данных, поэтому проверка изменения файла перед выходом не требуется.

В программе предусмотрена защита от «неправильных» строк, поэтому при получении на вход файла, содержащего лишние или повреждённые данные в строке, в таблице просто не появятся такие записи.

# Схема иерархии классов и модулей

Схема иерархии классов и модулей представлена ниже на рисунке 1.

Рисунок 1 - схема иерархии классов и модулей



# Спецификация модулей

Все функции, не считая int main(int argc, char \*argv[]), содержатся в классах, относящихся к конкретному окну. Далее они будут приводиться в виде var class::function([]), где var – возвращаемый функцией тип данных, class – название класса, function – название функции, [] – входные данные.

Класс MainWindow относится к главному окну программ и содержит в себе все необходимые модули, позволяющие работать с таблицей, обрабатывать поиск, работать с данными и передавать нужные команды другим окнам.

Класс Info относится к информационному окну, позволяет получать данные из главного окна и выводить их на экран. Окно используется при нажатии на кнопки «Руководство пользователя», «Изучить задание» и «О программе».

Класс DeleteQuestion относится к окну с вопросом об удалении, позволяет получать количество удаляемых элементов и подстраивать под это число выводимый текст.

Класс ChartDialog относится к окну построения диаграмм, позволяет получать записи из главного окна и строить диаграмму на их основе.

Класс AddDialog относится к окнам добавления и изменения записи, позволяет в зависимости от выбранного режима добавить или изменить запись в файле, при этом в случае с изменением записи подставляются данные изменяемой записи.

MainWindow::MainWindow(QWidget \*parent), Info::Info(QWidget \*parent), DeleteQuestion::DeleteQuestion(QWidget \*parent), ChartDialog::ChartDialog(QWidget \*parent), AddDialog::AddDialog(QWidget \*parent) – конструкторы классов, автоматически выполняющающие некоторый код при вызове соответствующего её класса.

При вызове MainWindow::MainWindow(QWidget \*parent) отключается возможность изменения размера окна, устанавливается текущий год для поиска по умолчанию, проверяется существование стандартного файла с данными, создаётся стандартный пустой файл при отсутствии, добавляется информация в таблицу, добавляются последние файлы в меню.

При вызове AddDialog::AddDialog(QWidget \*parent) отключается возможность изменения размера окна, убирается ненужная кнопка справки в углу окна, устанавливается текущий год в поле ввода года.

При вызове ChartDialog::ChartDialog(QWidget \*parent) и DeleteQuestion::DeleteQuestion(QWidget \*parent) отключается возможность изменения размера окна, убирается ненужная кнопка справки в углу окна.

При вызове Info::Info(QWidget \*parent) убирается ненужная кнопка справки в углу окна.

Классы также нуждаются в деструкторах MainWindow::~MainWindow(), Info::~Info(), DeleteQuestion::~DeleteQuestion(), ChartDialog::~ChartDialog(), AddDialog::~AddDialog(). Они были созданы автоматически и не были изменены. Они вызываются при удалении класса для очистки ресурсов, появившихся в оперативной памяти во время работы с классом.

bool MainWindow::lineFits(QString line) – проверяет правильность строки данных и то, подходит ли она по выбранным критериям поиска.

void MainWindow::getDataFromFile() – получает данные из файла и обрабатывает их.

void MainWindow::setSearch() – берёт данные из полей ввода и применяет поиск для них.

void MainWindow::addLine(QString line) – добавляет подходящую строку в таблицу, делая запись.

void MainWindow::tryEditLine() – позволяет изменить запись в случае правильного выделения данных.

void MainWindow::paintSearchButton() – изменяет шрифт кнопки «Найти записи» при взаимодействии с полями ввода.

void MainWindow::setRecentDataFile(QString file) – устанавливает file как текущий используемый файл, выводит данные из файла.

void MainWindow::updateLastFilesMenu() – добавляет последние файлы в меню.

void MainWindow::addNewLastFile(QString f) – добавляет файл f в верх файла, хранящего последние файлы, и добавляет его в меню.

void MainWindow::on\_manufacturerInput\_textChanged(), void MainWindow::on\_modelInput\_textChanged(), void MainWindow::on\_colorInput\_textChanged(), void MainWindow::on\_yearInput1\_valueChanged(int), void MainWindow::on\_yearInput2\_valueChanged(int), void MainWindow::on\_mileageInput1\_valueChanged(int), void MainWindow::on\_mileageInput2\_valueChanged(int), void MainWindow::on\_priceInput1\_valueChanged(int), void MainWindow::on\_priceInput2\_valueChanged(int) – функции, вызываемые при изменении данных в полях поиска, все эти функции служат только для изменения шрифта кнопки «Найти записи».

void MainWindow::on\_searchButton\_clicked() – функция, вызываемая при нажатии кнопки «Найти записи», которая применяет поиск и ищет подходящие записи.

void MainWindow::on\_disableSearchButton\_clicked() – функция, вызываемая при нажатии кнопки «Сбросить поиск», которая сбрасывает поиск и выводит все записи.

void MainWindow::on\_chartButton\_clicked() – функция, вызываемая при нажатии кнопки «Диаграмма цен», которая формирует пару массивов, имеющие в себе отсортированные по годам пары год-средняя цена, и передаёт их в класс ChartDialog для отображения диаграммы.

void MainWindow::on\_saveFile\_triggered() – функция, вызываемая при нажатии кнопки «Сохранить выборку в файл» или Ctrl+S на клавиатуре, которая сохраняет данные, выведенные в таблице, в файл.

void MainWindow::on\_getFile\_triggered() – функция, вызываемая при нажатии кнопки «Открыть файл» или Ctrl+O на клавиатуре, которая делает выбранный файл используемым и выводит данные из него.

void MainWindow::on\_defaultFile\_triggered() – функция, вызываемая при нажатии кнопки «Вернуться к стандартной БД» или Ctrl+Q на клавиатуре, которая делает файл по умолчанию используемым и выводит данные из него.

void MainWindow::on\_addElem\_triggered() – функция, вызываемая при нажатии кнопки «Добавить запись» или Ctrl+A на клавиатуре, которая вызывает окно добавления записи и при успешном добавлении обновляет таблицу.

void MainWindow::on\_editElem\_triggered() – функция, вызываемая при нажатии кнопки «Изменить запись» или основной Enter на клавиатуре, которая вызывает окно изменения записи и при успешном изменении обновляет таблицу.

void MainWindow::on\_delElem\_triggered() – функция, вызываемая при нажатии кнопки «Удалить запись(-и)» или Delete на клавиатуре, которая вызывает окно подтверждения удаления и при положительном ответе удаляет записи и обновляет таблицу.

void MainWindow::on\_cancelElem\_triggered() – функция, вызываемая при нажатии кнопки «Отменить выделение» или Esc на клавиатуре, которая отменяет выделение записей в таблице.

void MainWindow::on\_guide\_triggered() – функция, вызываемая при нажатии кнопки «Руководство пользователя» или F1 на клавиатуре, которая вызывает информационное окно с необходимыми данными из файла.

void MainWindow::on\_task\_triggered() – функция, вызываемая при нажатии кнопки «Задание» или F11 на клавиатуре, которая вызывает информационное окно с необходимыми данными из файла.

void MainWindow::on\_about\_triggered() – функция, вызываемая при нажатии кнопки «О программе» или F12 на клавиатуре, которая вызывает информационное окно с необходимыми данными.

void MainWindow::on\_tableWidget\_cellDoubleClicked() – функция, вызываемая при двойном нажатии по элементу таблицы, которая вызывает действие кнопки «Изменить запись» для удобного изменения записи.

void Info::setData(QString titleText, QString labelText) – функция, устанавливающая название окна как titleText и текстовое поле как labelText для модификации информационного окна под необходимую информацию.

void DeleteQuestion::setSize(uint32\_t s) – функция, изменяющая текстовое поле в случае, если удаляемых записей s больше, чем одна.

void DeleteQuestion::on\_okButton\_clicked() – функция, вызываемая при нажатии кнопки «Да», которая записывает подтверждение и закрывает окно.

void DeleteQuestion::on\_cancelButton\_clicked() – функция, вызываемая при нажатии кнопки «Отмена», которая закрывает окно.

void ChartDialog::addData(QList<uint16\_t> y, QList<uint32\_t> p) – функция, устанавливающая передаваемые отсортированные массивы годов y и средних цен p в переменные класса для последующей отрисовки диаграммы, находит количество полос, максимальную цену, устанавливает подписи и модифицирует слайдер в зависимости от количества полос.

void ChartDialog::paintEvent(QPaintEvent \*e) – функция, вызываемая при открытии окна и его обновлении, которая отрисовывает диаграмму.

void ChartDialog::on\_horizontalSlider\_sliderMoved(int position) – функция, вызываемая при движении слайдера, которая обновляет данные о смещении и обновляет окно графика.

void AddDialog::setItAsEditor(QList<QTableWidgetItem \*> row) – функция, которая делает окно изменяемым запись, устанавливает данные row выбранной записи в поля ввода для удобства.

void AddDialog::setData(QString nowFile, QString defFile) – функция, которая устанавливает данные о текущем nowFile и главном defFile файлах.

void AddDialog::on\_changeButton\_clicked() – функция, вызываемая при нажатии кнопки, которая добавляет или изменяет запись и закрывает окно.

# Таблица тестов и результаты тестирования, результаты работы программы

|  |  |  |
| --- | --- | --- |
| Действие | Ожидание | Результат |
| Открытие программы впервые | Предупреждение об отсутствии стандартной БД, её создание и открытие программы | Результат идентичен ожиданию (рисунки 2, 3) |
| Нажатие на кнопку меню «Добавить запись». Нажатие клавиш Ctrl+A | Добавление записи в отдельном диалоговом окне | Результат идентичен ожиданию (рисунки 4, 5) |
| Нажатие на кнопку меню «Изменить запись». Нажатие клавиши Enter | Изменение записи в отдельном диалоговом окне | Результат идентичен ожиданию (рисунок 6) |
| Нажатие на кнопку меню «Удалить запись». Нажатие клавиши Delete | Удаление записи с предварительным подтверждением | Результат идентичен ожиданию (рисунок 7) |
| Нажатие на кнопку меню «Отменить выделение». Нажатие клавиши Esc | Отмена выделения в таблице | Результат идентичен ожиданию |
| Нажатие по названию колонки | Сортировка записей по выбранному критерию | Результат идентичен ожиданию (рисунок 8) |
| Открытие программы повторно | Открытие программы с выводом стандартной БД | Результат идентичен ожиданию (рисунок 5) |
| Ввод данных поиска | Изменение стиля кнопки «Найти записи» | Результат идентичен ожиданию (рисунок 9) |
| Нажатие на кнопку «Найти записи» | В таблице выведены записи, удовлетворяющие поисковому запросу | Результат идентичен ожиданию (рисунок 10) |
| Нажатие на кнопку «Диаграмма цен» | Открытие дополнительного окна с диаграммой, использующей данные таблицы с учётом текущего поиска | Результат идентичен ожиданию (рисунок 11) |
| Нажатие на кнопку меню «Сохранить выборку в файл». Нажатие клавиш Ctrl+S | Вывод окна выбора папки и названия файла, сохранение, добавление файла в последние | Результат идентичен ожиданию (рисунок 12) |
| Нажатие на кнопку «Сбросить поиск» | В таблице выведены все записи | Результат идентичен ожиданию (рисунок 8) |
| Нажатие на кнопку меню «Открыть файл». Нажатие клавиш Ctrl+O | Вывод окна выбора папки и названия файла, открытие выбранного файла, добавление файла в последние | Результат идентичен ожиданию (рисунок 13) |
| Нажатие на кнопку меню «Вернуться к стандартной БД». Нажатие клавиш Ctrl+Q | Открытие стандартного файла | Результат идентичен ожиданию (рисунок 8) |
| Открытие последнего файла | Открытие выбранного файла | Результат идентичен ожиданию (рисунок 8) |
| Нажатие на кнопку меню «Руководство пользователя». Нажатие клавиши F1 | Вывод информации из файла в новом окне | Результат идентичен ожиданию (рисунок 14) |
| Нажатие на кнопку меню «Изучить задание». Нажатие клавиши F11 | Вывод информации в новом окне | Результат идентичен ожиданию (рисунок 15) |
| Нажатие на кнопку меню «О программе». Нажатие клавиши F12 | Вывод информации в новом окне | Результат идентичен ожиданию (рисунок 16) |

Таблица 1 – таблица тестов

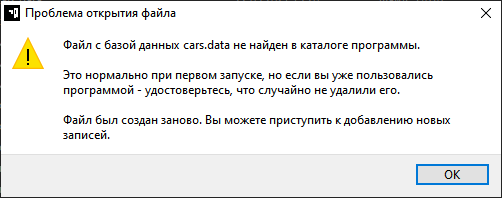


Рисунок 2 – предупреждение об успешном создании нового файла

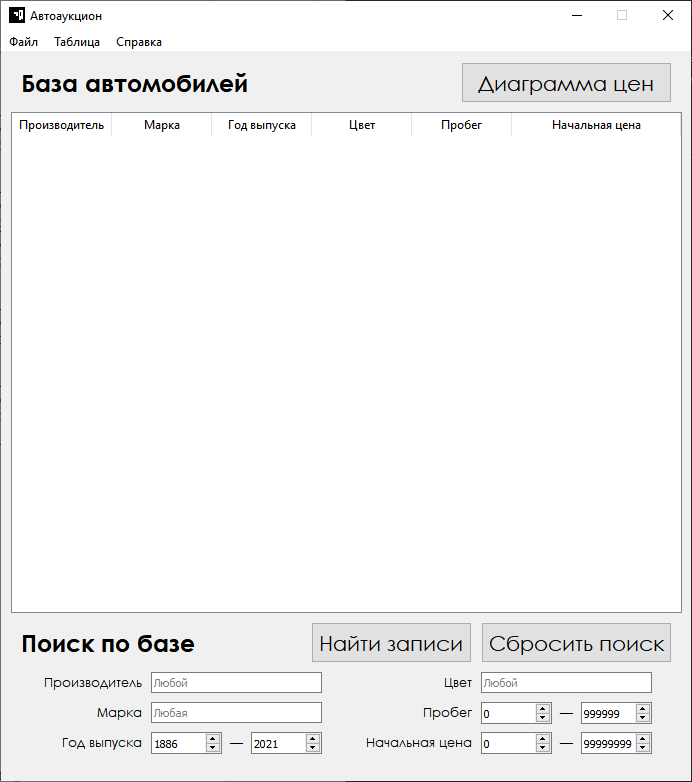


Рисунок 3 – вывод пустого файла

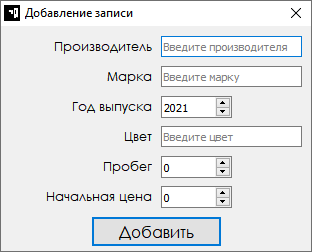


Рисунок 4 – добавление записи

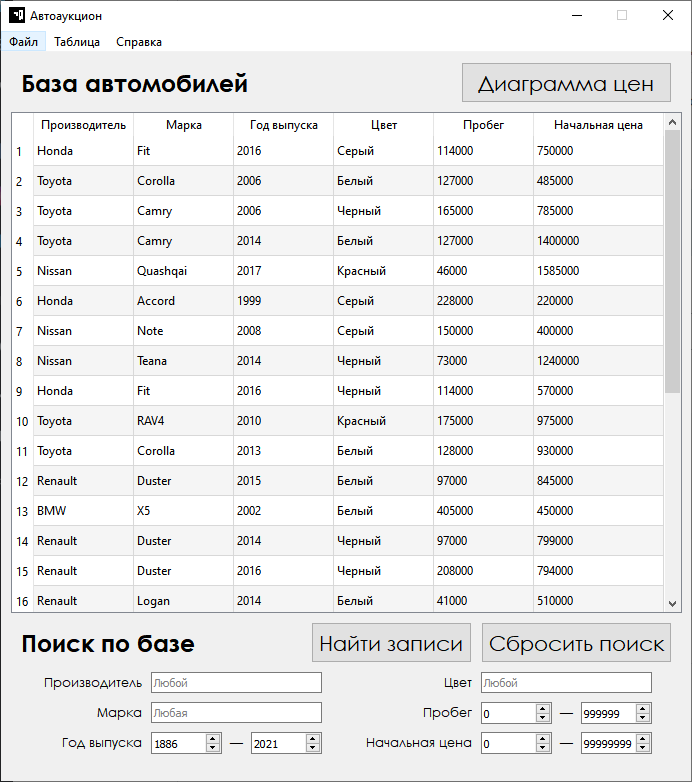


Рисунок 5 – вывод имеющихся в файле записей после добавления, в том числе после повторного открытия программы

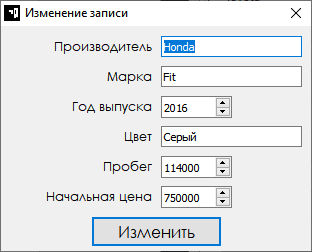


Рисунок 6 – изменение записи

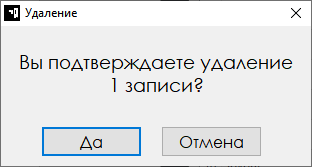


Рисунок 7 – подтверждение удаления одной записи

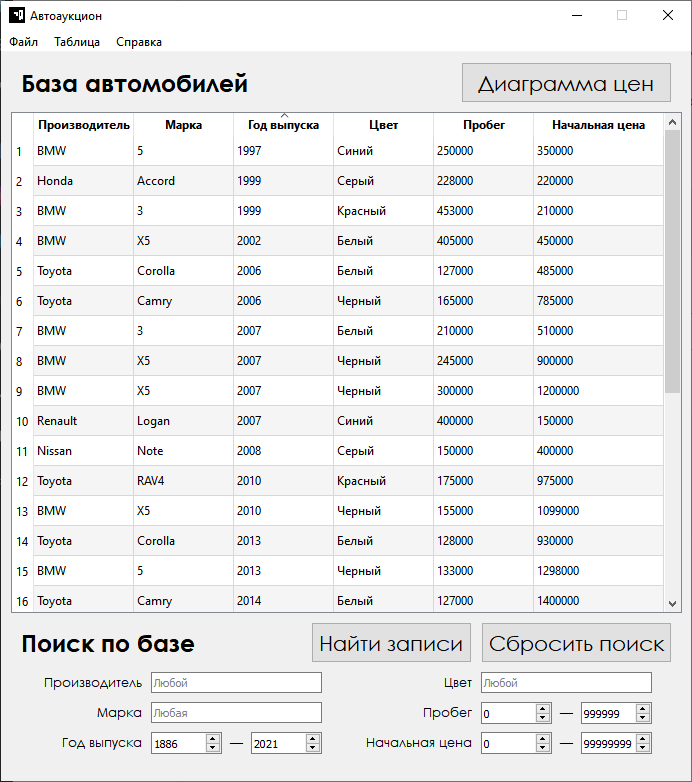


Рисунок 8 – сортировка по году выпуска

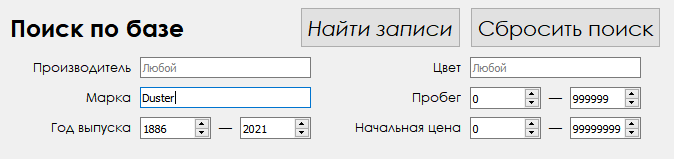


Рисунок 9 – введён запрос, но не нажата кнопка «Найти записи»

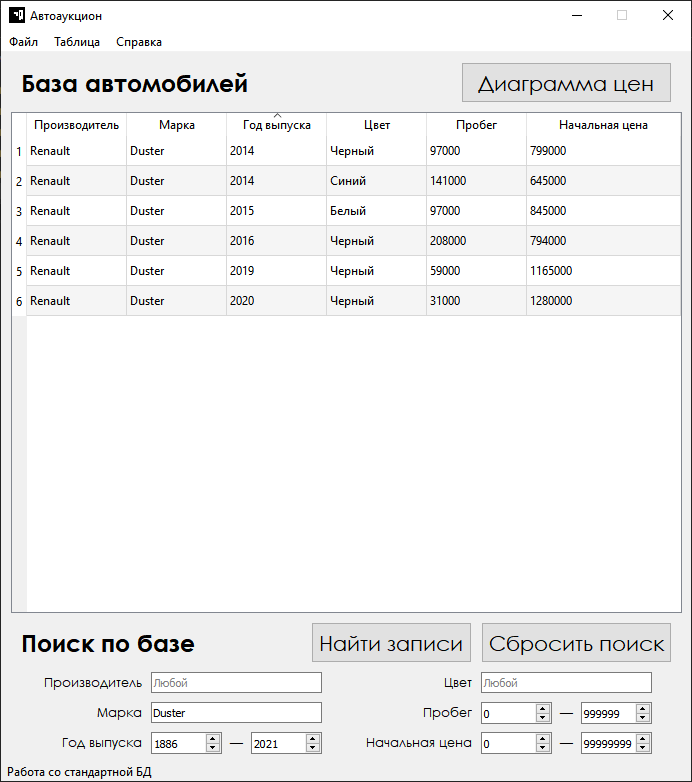


Рисунок 10 – произведён поиск

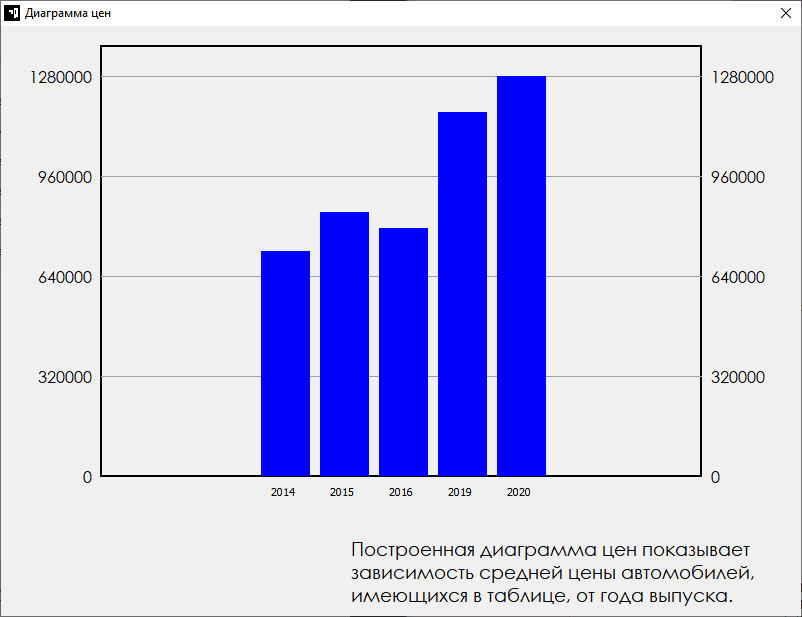


Рисунок 11 – среднегодовая диаграмма цен выборки

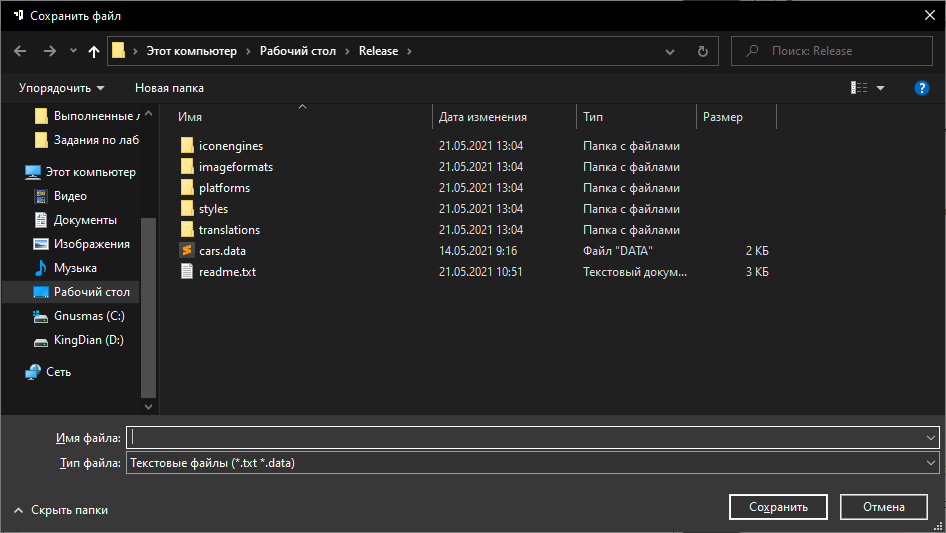


Рисунок 12 – создание файла

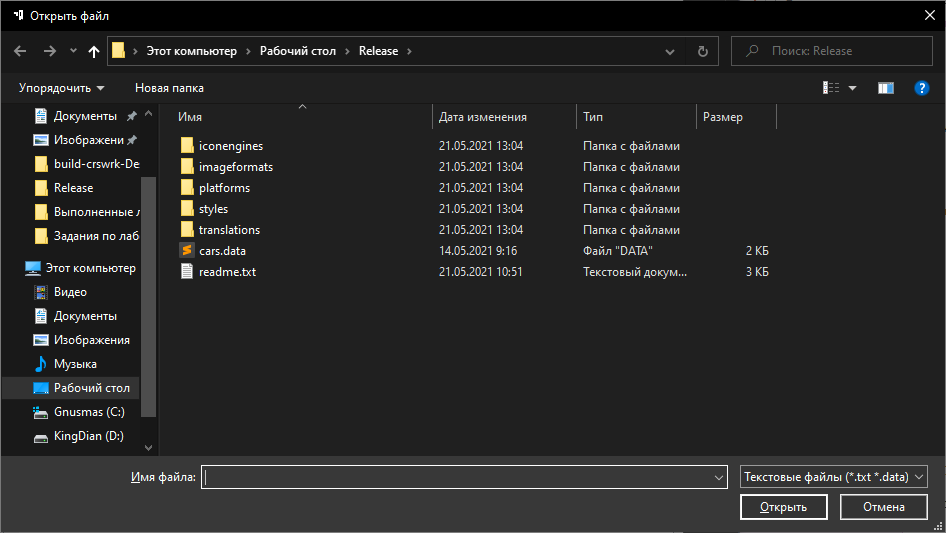


Рисунок 13 – выбор файла

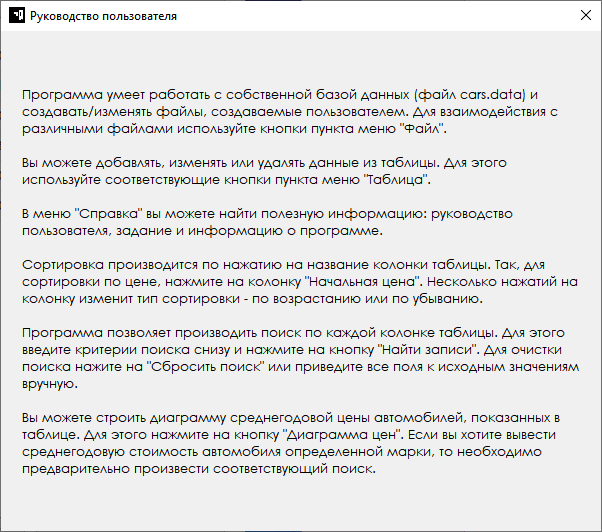


Рисунок 14 – руководство пользователя

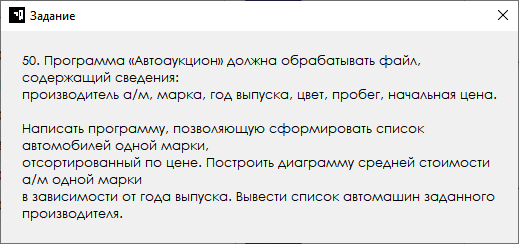


Рисунок 15 – вывод задания

censored

Рисунок 16 – информация о программе

По результатам работы и тестирования программы можно убедиться, что всё работает и все критерии задания соблюдены.

# Руководство для пользователя

Программа умеет работать с собственной базой данных (файл cars.data) и создавать/изменять файлы, создаваемые пользователем. Для взаимодействия с различными файлами используйте кнопки пункта меню "Файл".

Вы можете добавлять, изменять или удалять данные из таблицы. Для этого используйте соответствующие кнопки пункта меню "Таблица".

В меню "Справка" вы можете найти полезную информацию: руководство пользователя, задание и информацию о программе.

Сортировка производится по нажатию на название колонки таблицы. Так, для сортировки по цене, нажмите на колонку "Начальная цена". Несколько нажатий на колонку изменит тип сортировки - по возрастанию или по убыванию.

Программа позволяет производить поиск по каждой колонке таблицы. Для этого введите критерии поиска снизу и нажмите на кнопку "Найти записи". Для очистки поиска нажмите на "Сбросить поиск" или приведите все поля к исходным значениям вручную.

Вы можете строить диаграмму среднегодовой цены автомобилей, показанных в таблице. Для этого нажмите на кнопку "Диаграмма цен". Если вы хотите вывести среднегодовую стоимость автомобиля определенной марки, то необходимо предварительно произвести соответствующий поиск.

# Исходный код

Файл main.cpp:

/\* Задание 50

\*

\* Программу создал Ярков М.Д.

\* Группа ИСТб-20-3

\* 20.05.2021

\*/

#include "mainwindow.h"

#include <QApplication>

int main(int argc, char \*argv[]) {

QApplication a(argc, argv);

MainWindow w;

w.show();

return a.exec();

}

Файл adddialog.h:

#pragma once

#include <QDialog>

#include <QTableWidget>

namespace Ui { class addDialog; }

class AddDialog : public QDialog {

Q\_OBJECT

public:

explicit AddDialog(QWidget \*parent = nullptr);

~AddDialog();

void setData(QString nowFile, QString defFile);

void setItAsEditor(QList<QTableWidgetItem \*> row);

bool successful = false;

QString nowDataFile, defaultDataFile;

private slots:

void on\_changeButton\_clicked();

private:

Ui::addDialog \*ui;

bool addMode = true;

QString manufacturer, model, color;

uint16\_t year;

uint32\_t mileage, price;

};

Файл adddialog.cpp:

#include "adddialog.h"

#include "ui\_adddialog.h"

#include <QFile>

#include <QDebug>

#include <QMessageBox>

#include <QDate>

AddDialog::AddDialog(QWidget \*parent) : QDialog(parent), ui(new Ui::addDialog) {

ui->setupUi(this);

this->setWindowFlags(Qt::Dialog | Qt::MSWindowsFixedSizeDialogHint); // отключение возможности изменять размер окна

this->setWindowFlags(this->windowFlags() & ~Qt::WindowContextHelpButtonHint); // отключение ?

ui->yearInput1->setValue(QDate::currentDate().year()); // установка текущего года

}

AddDialog::~AddDialog() { delete ui; }

void AddDialog::setData(QString nowFile, QString defFile) { // установка сведений об открытом файле

nowDataFile = nowFile;

defaultDataFile = defFile;

ui->manufacturerInput->setFocus();

}

void AddDialog::setItAsEditor(QList<QTableWidgetItem \*> row) { // сделать окно изменяющим запись

manufacturer = row[0]->text();

model = row[1]->text();

year = row[2]->text().toUInt();

color = row[3]->text();

mileage = row[4]->text().toUInt();

price = row[5]->text().toUInt();

ui->manufacturerInput->setText(manufacturer);

ui->modelInput->setText(model);

ui->colorInput->setText(color);

ui->yearInput1->setValue(year);

ui->mileageInput1->setValue(mileage);

ui->priceInput1->setValue(price);

addMode = false;

this->setWindowTitle("Изменение записи");

ui->changeButton->setText("Изменить");

}

void AddDialog::on\_changeButton\_clicked() { // обработка данных при нажатии на кнопку

QString line = ui->manufacturerInput->text() + '\t' + ui->modelInput->text() + '\t' + ui->yearInput1->text() + '\t' + ui->colorInput->text() + '\t' + ui->mileageInput1->text() + '\t' + ui->priceInput1->text();

QFile file(nowDataFile); // открытие файла

if (line.count('\n') == 0 and line.count('\t') == 5 and ui->manufacturerInput->text() != "" and ui->modelInput->text() != "" and ui->colorInput->text() != "") {

if (file.exists() and file.open(QIODevice::ReadOnly | QIODevice::Text)) {

// получение исходных данных

QString fileData = file.readAll();

file.close();

// запись новых данных

if (!file.open(QIODevice::WriteOnly | QIODevice::Text)) {

if (nowDataFile == defaultDataFile) QMessageBox::warning(this, "Проблема изменения файла", "Возникла проблема при обращении к БД. Запись не была изменена. Повторите попытку.");

else QMessageBox::warning(this, "Проблема изменения файла", "Возникла проблема при обращении к файлу " + nowDataFile + ". Запись не была изменена. Повторите попытку");

return;

}

QTextStream writeStream(&file);

writeStream.setCodec("UTF-8");

if (addMode) {

writeStream << fileData << line << '\n';

}

else {

QString prevLine = manufacturer + '\t' + model + '\t' + QString::number(year) + '\t' + color + '\t' + QString::number(mileage) + '\t' + QString::number(price);

writeStream << fileData.replace(fileData.indexOf(prevLine), prevLine.size(), line);

}

file.close();

// выход из окна

successful = true;

close();

}

else {

if (nowDataFile == defaultDataFile) QMessageBox::warning(this, "Проблема открытия файла", "Возникла проблема при обращении к БД. Записи не были получены, изменение невозможно. Повторите попытку.");

else QMessageBox::warning(this, "Проблема открытия файла", "Возникла проблема при обращении к файлу " + nowDataFile + ". Записи не были получены, изменение невозможно. Повторите попытку.");

}

}

else QMessageBox::warning(this, "Ошибка введения данных", "Вы не ввели данные в поле, вставили знак перехода на следующую строку или знак табуляции. Это могло произойти при копировании из другого источника. Добавьте данные или удалите эти знаки и повторите попытку.");

}

Файл info.h:

#pragma once

#include <QDialog>

namespace Ui { class Info; }

class Info : public QDialog {

Q\_OBJECT

public:

explicit Info(QWidget \*parent = nullptr);

~Info();

void setData(QString titleText, QString labelText);

private:

Ui::Info \*ui;

};

Файл info.cpp:

#include "info.h"

#include "ui\_info.h"

Info::Info(QWidget \*parent) : QDialog(parent), ui(new Ui::Info) {

ui->setupUi(this);

this->setWindowFlags(this->windowFlags() & ~Qt::WindowContextHelpButtonHint); // отключение ?

}

Info::~Info() { delete ui; }

void Info::setData(QString titleText, QString labelText) { // установка текста

this->setWindowTitle(titleText);

ui->infoLabel->setText(labelText);

}

Файл deletequestion.h:

#pragma once

#include <QDialog>

namespace Ui { class DeleteQuestion; }

class DeleteQuestion : public QDialog {

Q\_OBJECT

public:

explicit DeleteQuestion(QWidget \*parent = nullptr);

~DeleteQuestion();

bool successful = false;

void setSize(uint32\_t s);

private slots:

void on\_okButton\_clicked();

void on\_cancelButton\_clicked();

private:

Ui::DeleteQuestion \*ui;

};

Файл deletequestion.cpp:

#include "deletequestion.h"

#include "ui\_deletequestion.h"

DeleteQuestion::DeleteQuestion(QWidget \*parent) : QDialog(parent), ui(new Ui::DeleteQuestion) {

ui->setupUi(this);

this->setWindowFlags(Qt::Dialog | Qt::MSWindowsFixedSizeDialogHint); // отключение возможности изменять размер окна

this->setWindowFlags(this->windowFlags() & ~Qt::WindowContextHelpButtonHint); // отключение ?

}

DeleteQuestion::~DeleteQuestion() { delete ui; }

void DeleteQuestion::setSize(uint32\_t s) { // изменение текста

if (s > 1) ui->label->setText("Вы подтверждаете удаление\n" + QString::number(s) + " записей?");

}

void DeleteQuestion::on\_okButton\_clicked() { successful = true; close(); } // выход из окна при нажатии "Да"

void DeleteQuestion::on\_cancelButton\_clicked() { close(); } // выход из окна при нажатии "Отмена"

Файл chartdialog.h:

#pragma once

#include <QDialog>

namespace Ui { class ChartDialog; }

class ChartDialog : public QDialog {

Q\_OBJECT

public:

explicit ChartDialog(QWidget \*parent = nullptr);

~ChartDialog();

void addData(QList<uint16\_t> years, QList<uint32\_t> prices);

private slots:

void on\_horizontalSlider\_sliderMoved(int position);

private:

Ui::ChartDialog \*ui;

void paintEvent(QPaintEvent \*e);

QList<uint16\_t> years;

QList<uint32\_t> prices;

uint32\_t maxPrice;

uint16\_t l, slider = 0;

};

Файл chartdialog.cpp:

#include "chartdialog.h"

#include "ui\_chartdialog.h"

#include <QPainter>

#include <QDebug>

ChartDialog::ChartDialog(QWidget \*parent) : QDialog(parent), ui(new Ui::ChartDialog) {

ui->setupUi(this);

this->setWindowFlags(Qt::Dialog | Qt::MSWindowsFixedSizeDialogHint); // отключение возможности изменять размер окна

this->setWindowFlags(this->windowFlags() & ~Qt::WindowContextHelpButtonHint); // отключение ?

}

ChartDialog::~ChartDialog() { delete ui; }

void ChartDialog::addData(QList<uint16\_t> y, QList<uint32\_t> p) { // добавление переданных данных, установка необходимых значений

years = y;

prices = p;

l = y.length();

// установка подписей цены слева и справа

maxPrice = \*std::max\_element(p.begin(), p.end());

ui->L4->setText(QString::number(maxPrice)); ui->R4->setText(QString::number(maxPrice));

ui->L3->setText(QString::number(maxPrice/4\*3)); ui->R3->setText(QString::number(maxPrice/4\*3));

ui->L2->setText(QString::number(maxPrice/2)); ui->R2->setText(QString::number(maxPrice/2));

ui->L1->setText(QString::number(maxPrice/4)); ui->R1->setText(QString::number(maxPrice/4));

if (l < 11) { // отключение слайдера, если все данные входят

ui->horizontalSlider->setVisible(false);

ui->label\_2->setVisible(false);

} // изменение максимума слайдера, если не все данные входят

else ui->horizontalSlider->setMaximum(l-10);

}

void ChartDialog::paintEvent(QPaintEvent \*e) { // отрисовка диаграммы после открытия окна или после движения слайдера

Q\_UNUSED(e); QPainter qp(this);

// строим коробку

QPen boxLine(Qt::black, 2, Qt::SolidLine); qp.setPen(boxLine);

qp.drawLine(100, 20, 700, 20);

qp.drawLine(100, 20, 100, 450);

qp.drawLine(700, 20, 700, 450);

qp.drawLine(100, 450, 700, 450);

// строим дополнительные линии

QPen additionalLine(Qt::gray, 1, Qt::SolidLine); qp.setPen(additionalLine);

qp.drawLine(100, 50, 700, 50);

qp.drawLine(100, 150, 700, 150);

qp.drawLine(100, 250, 700, 250);

qp.drawLine(100, 350, 700, 350);

// строим полоски диаграммы и подписи

QPainter myText(this); myText.setFont(QFont("Helvetica", 8));

if (l < 11) {

uint16\_t step = 295-59/2\*l;

for (uint8\_t i = 0; i < l; i++) {

qp.fillRect(110+step+59\*i, 450, 49, -400\*(1.0\*prices[i]/maxPrice), Qt::blue);

qp.drawText(120+step+59\*i, 470, QString::number(years[i]));

}

}

else {

for (uint8\_t i = 0; i < 10; i++) {

qp.fillRect(110+59\*i, 450, 49, -400\*(1.0\*prices[i+slider]/maxPrice), Qt::blue);

qp.drawText(120+59\*i, 470, QString::number(years[i+slider]));

}

}

}

void ChartDialog::on\_horizontalSlider\_sliderMoved(int position) { // обработка движения слайдера снизу

slider = position;

update();

}

Файл adddialog.h:

#pragma once

#include <QDialog>

#include <QTableWidget>

namespace Ui { class addDialog; }

class AddDialog : public QDialog {

Q\_OBJECT

public:

explicit AddDialog(QWidget \*parent = nullptr);

~AddDialog();

void setData(QString nowFile, QString defFile);

void setItAsEditor(QList<QTableWidgetItem \*> row);

bool successful = false;

QString nowDataFile, defaultDataFile;

private slots:

void on\_changeButton\_clicked();

private:

Ui::addDialog \*ui;

bool addMode = true;

QString manufacturer, model, color;

uint16\_t year;

uint32\_t mileage, price;

};

Файл adddialog.cpp:

#include "adddialog.h"

#include "ui\_adddialog.h"

#include <QFile>

#include <QDebug>

#include <QMessageBox>

#include <QDate>

AddDialog::AddDialog(QWidget \*parent) : QDialog(parent), ui(new Ui::addDialog) {

ui->setupUi(this);

this->setWindowFlags(Qt::Dialog | Qt::MSWindowsFixedSizeDialogHint); // отключение возможности изменять размер окна

this->setWindowFlags(this->windowFlags() & ~Qt::WindowContextHelpButtonHint); // отключение ?

ui->yearInput1->setValue(QDate::currentDate().year()); // установка текущего года

}

AddDialog::~AddDialog() { delete ui; }

void AddDialog::setData(QString nowFile, QString defFile) { // установка сведений об открытом файле

nowDataFile = nowFile;

defaultDataFile = defFile;

ui->manufacturerInput->setFocus();

}

void AddDialog::setItAsEditor(QList<QTableWidgetItem \*> row) { // сделать окно изменяющим запись

manufacturer = row[0]->text();

model = row[1]->text();

year = row[2]->text().toUInt();

color = row[3]->text();

mileage = row[4]->text().toUInt();

price = row[5]->text().toUInt();

ui->manufacturerInput->setText(manufacturer);

ui->modelInput->setText(model);

ui->colorInput->setText(color);

ui->yearInput1->setValue(year);

ui->mileageInput1->setValue(mileage);

ui->priceInput1->setValue(price);

addMode = false;

this->setWindowTitle("Изменение записи");

ui->changeButton->setText("Изменить");

}

void AddDialog::on\_changeButton\_clicked() { // обработка данных при нажатии на кнопку

QString line = ui->manufacturerInput->text() + '\t' + ui->modelInput->text() + '\t' + ui->yearInput1->text() + '\t' + ui->colorInput->text() + '\t' + ui->mileageInput1->text() + '\t' + ui->priceInput1->text();

QFile file(nowDataFile); // открытие файла

if (line.count('\n') == 0 and line.count('\t') == 5 and ui->manufacturerInput->text() != "" and ui->modelInput->text() != "" and ui->colorInput->text() != "") {

if (file.exists() and file.open(QIODevice::ReadOnly | QIODevice::Text)) {

// получение исходных данных

QString fileData = file.readAll();

file.close();

// запись новых данных

if (!file.open(QIODevice::WriteOnly | QIODevice::Text)) {

if (nowDataFile == defaultDataFile) QMessageBox::warning(this, "Проблема изменения файла", "Возникла проблема при обращении к БД. Запись не была изменена. Повторите попытку.");

else QMessageBox::warning(this, "Проблема изменения файла", "Возникла проблема при обращении к файлу " + nowDataFile + ". Запись не была изменена. Повторите попытку");

return;

}

QTextStream writeStream(&file);

writeStream.setCodec("UTF-8");

if (addMode) {

writeStream << fileData << line << '\n';

}

else {

QString prevLine = manufacturer + '\t' + model + '\t' + QString::number(year) + '\t' + color + '\t' + QString::number(mileage) + '\t' + QString::number(price);

writeStream << fileData.replace(fileData.indexOf(prevLine), prevLine.size(), line);

}

file.close();

// выход из окна

successful = true;

close();

}

else {

if (nowDataFile == defaultDataFile) QMessageBox::warning(this, "Проблема открытия файла", "Возникла проблема при обращении к БД. Записи не были получены, изменение невозможно. Повторите попытку.");

else QMessageBox::warning(this, "Проблема открытия файла", "Возникла проблема при обращении к файлу " + nowDataFile + ". Записи не были получены, изменение невозможно. Повторите попытку.");

}

}

else QMessageBox::warning(this, "Ошибка введения данных", "Вы не ввели данные в поле, вставили знак перехода на следующую строку или знак табуляции. Это могло произойти при копировании из другого источника. Добавьте данные или удалите эти знаки и повторите попытку.");

}

Заключение

Разработав программу “Автоаукцион”, я закрепил знания, полученные во время лекций и практик по предмету «Программирование на языке высокого уровня», получил новый опыт при использовании функций и методов разработки, с которыми ещё не сталкивался.

Мне получилось успешно реализовать программу по выбранному заданию, отвечающую всем требованиям. Она может быть скомпилирована под все основные настольные ОС, делая максимальных охват пользователей.

Список использованных источников

1. Документация фреймворка Qt. [Электронный ресурс]. URL: <https://doc.qt.io/qt-5/> (дата обращения: 10.05.2021)
2. Уроки по Qt. 7 урок. QTableWidget и QTimer. [Электронный ресурс]. URL: <https://www.youtube.com/watch?v=AUCP3OSkxg8> (дата обращения: 10.05.2021)
3. Как сделать файл .exe в Qt Creator [Электронный ресурс]. URL: https://coderoad.ru/25570752/Как-сделать-файл-exe-в-Qt-Creator (дата обращения: 17.05.2021)