Феральное государственное бюджетное образовательное учреждение высшего образования

«Санкт-Петербургский государственный университет телекоммуникаций им. проф. М. А. Бонч-Бруевича»

(СПбГУТ)

Кафедра Программной инженерии и вычислительной техники

**Курсовая работа по дисциплине**

**«Объектно-ориентированное программирование»**

**Работу выполнил:**

Студент 2-го курса

Группа ИКПИ-73

Васюхин Никита Александрович

**Приняла:**

Петрова Ольга Борисовна

Санкт-Петербург

2018 уч. г.

СОДЕРЖАНИЕ

|  |  |
| --- | --- |
| 1. Цель работы................................................................................................... | 3 |
| 1. Обоснование выбранного контейнера…………………………………… | 3 |
| 1. Таблица идентификаторов…………………….………………………….. | 4 |
| 1. Поля классов……………….......................................................................... | 7 |
| 1. Расчет кратчайших маршрутов между узлами сети…………………….. | 7 |
| 1. Разработка модулей программы………..………………………………… | 9 |
| 1. Реализация взаимодействия программы с MS Excel……………………. | 29 |
| 1. Результаты тестирования……………………………...………………….. | 30 |
| 1. Инструкция пользователя………………………………………………… 2. Заключение……………………………………………………………….. 3. Список литературы…………………...…………………………. | 31  41  42 |

**Цель работы**

Создать базу данных, работающую на основе контейнера, который следует разработать в процессе выполнения курсовой работы. В качестве прототипа контейнера можно использовать контейнеры vector или list из библиотеки STL. Для работы с контейнером следует предусмотреть итератор. Остальные требования к контейнеру определяются студентом самостоятельно.

Разрабатываемая база данных должна быть создана использованием C++ и библиотеки Qt4. Среда программирования Qt Creator.

В качестве элементов контейнеров должны использоваться объекты классов.

База данных должна обеспечивать выполнение следующих операций

* создание базы данных,
* объединение баз данных,
* добавления и удаления записей,
* редактирования записей,
* просмотра базы данных,
* поиск данных (найти данные, удовлетворяющие заданному критерию),
* запись на диск,
* чтения с диска.

Интерфейсная часть программы должна содержать следующие компоненты:

* окно «ABOUT»,
* главное меню,
* всплывающее меню,
* строку состояния,
* панель инструментов,
* подсказки по командам меню и панели инструментов.

**Вариант 4**

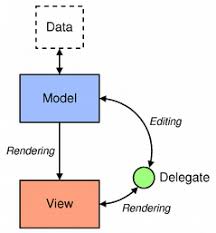
Класс, характеризующий автомобиль.

**Обоснование выбранного контейнера**

В курсовой работе был разработан контейнер на основе двусвязного списка из библиотеки стандартных шаблонов (STL). Он был выбран из-за того, что в контейнере такого типа создание объектов происходит с большой скоростью и без большой потери по времени, так как не происходит никакого копирования, а происходит только выделение памяти. В векторе же мы нуждаемся в постоянном копировании данных. В данной работе список будет работать только с одним классом, разработанным нами.

**Разработка структуры программы**

Разработанная программа функционирует по принципу MVC.



**Таблица идентификаторов**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Файлы** | **Классы** | **Методы класса** | **Функция** | **Назначение** |
| List.h | List | List():head(nullptr),tail(nullptr),Size(0) | Конструктор | Контейнер |
| ~List() | Деструктор |
| void push\_back(Car var); | Добавить элемент в конец |
| void push\_front(Car var); | Добавить элемент в начало |
| void pop\_back() | Убрать элемент с конца |
| void pop\_front() | Убрать элемент с начала |
| Car operator[](int index) const | Перегрузка [] |
| int GetSize() const | Получить кол-во элементов в листе |
| void setCar(int indexr, int indexc, QString data) | Поменять данные в листе |
| void erase(Iterator it); | Удалить элемент |
| Iterator begin() const | Начало списка |
| Iterator end() const | Конец списка |
| void clear(); | Очистить контейнер |
| Node | Node(Car data = Car(), Node \*pNext = nullptr, Node \*pPrev = nullptr) | Конструктор | Ячейка с данными |
| Iterator | Iterator(Node \*head) | Конструктор | Итератор |
| ~Iterator() | Деструктор |
| Car& operator\*() | Перегруженный оператор разыменования |
| Iterator operator++(int) | Перегруженный оператор инкремента |
| Iterator operator--(int) | Перегруженный оператор декремента |
| bool operator ==(Iterator const rhs) const | Перегруженный оператор равенства |
| bool operator !=(Iterator const rhs) const | Перегруженный оператор неравенства |
| Node \*GetHead() | Получить данные |
| Cars.h | Car | Car(); | Конструктор | Класс |
| Car(QString make, QString model, QString power, QString engine, QString year, QString plat) | Конструктор с параметрами |
| Car(const Car &other) | Конструктор копирования |
| ~Car() | Деструктор |
| QString GetMake() | Геттер Make |
| QString GetModel() | Геттер Model |
| QString GetPower() | Геттер Power |
| QString GetEngine() | Геттер Engine |
| QString GetYear() | Геттер Year |
| QString GetPlat() | Геттер Plat |
| void SetMake(QString name) | Сеттер Make |
| void SetModel(QString model) | Сеттер Model |
| void SetPower(QString power) | Сеттер Power |
| void SetEngine(QString engine) | Сеттер Engine |
| void SetYear(QString year) | Сеттер Year |
| void SetPlay(QString plat) | Сеттер Plat |
| QString operator[](int index) const | Перегрузка [] |
| mainwindow.h | MainWindow | explicit MainWindow(QWidget \*parent = 0) | Конструктор | Главное окно программы |
| ~MainWindow(); | Деструктор |
| void contextMenuEvent(QContextMenuEvent \*event) | Создание контекстного меню | События окна |
| void closeEvent(QCloseEvent \*event) | Событие при закрытии программы |
| void receiveData(QString make,QString model, QString power, QString engine, QString year, QString plat) | Получаем информацию с диалога daddcar | Слоты |
| void removeRows() | Удаление строк |
| void getFind(int row,QString str) | Получаем информацию с диалога finddialog |
| void open() | Открыть файл |
| bool saveFile() | Сохранить файл |
| bool saveAs() | Реализация “Сохранить как” |
| bool save() | Реализация “Сохранить” |
| void copy() | Скопировать элемент |
| void paste() | Вставить элемент |
| void cut() | Вырезать элемент |
| void append() | Объединить таблицы |
| void about() | Информация о программе |
| void aboutQt() | Информация о Qt |
| void faq() | Инструкция по программе |
| void nfindpush() | Быстрый поиск “вперед” |
| void pfindpush() | Быстрый поиск “назад” |
| void newFile() | Создать файл |
| void setConnection() | Создание соединений | Методы окна |
| void setStatusBar() | Создание статусбара |
| bool maybeSave() | Реализация “Файл быз изменён,не хотите его сохранитрь?” |
| void Excel() | Запуск макрсово Excel |
| void setCurrentFile(const QString &Name) | Поменять название окна |
| qcarmodel.h | QCarModel : public QAbstarctTableModel | QCarModel( QObject\* parent = 0 ) | Конструктор | Модель данных |
| int rowCount(const QModelIndex &parent) const | Кол-во строк |
| int columnCount(const QModelIndex &parent) const | Кол-во столбцов |
| QVariant headerData( int section, Qt::Orientation orientation, int role ) const | Шапка таблицы |
| QVariant data(const QModelIndex &index, int role) const | Отображение данных в таблице |
| bool setData( const QModelIndex& index, const QVariant& value, int role ) | Редактировать данные в таблице |
| Qt::ItemFlags flags( const QModelIndex& index ) const | Реализация флагов |
| void addCar(const QString& make,const QString& model,const QString& power,const QString& engine, const QString& year, const QString& plat ) | Добавить элемент в таблицу из диалога |
| void remove(int row) | Удалить данные из таблицы |
| void addExcelCar(Car temp) | Добавить данные в табилце из файла |
| void clearTable() | Очистить таблицу |
| daddcar.h | DAddCar: public QDialog | explicit DAddCar(QWidget \*parent = 0) | Конструктор | Диалог добавление данных в таблицу |
| ~DAddCar() | Деструктор |
| void sendData(QString model, QString make, QString power, QString engine, QString year, QString plat) | Отправить данные в главное окно программы | Сигналы диалога |
| void okButtonPush() | Реализация кнопки Ok | Слоты диалога |
| finddialog.h | findDialog : public QDialog | explicit findDialog(QWidget \*parent = 0) | Конструктор | Диалог поиска данных в таблице |
| ~findDialog() | Деструктор |
| void SfindData(int check, QString str) | Отправить данные в главное окно программы | Сигналы диалога |
| void okButtonPush() | Реализация кнопки Ok | Слоты диалога |

**Поля классов**

**List:**

* Node \* head – начало списка
* Node \* tail – конец списка
* int Size– размер списка

**Node:**

* Node \*pNext – указатель на следующий элемент
* Node \*pPrev – указатель на предыдущий элемент

**Iterator:**

* Node \* head – узел итератора

**Car:**

* QString make – Марка машины
* QString model – Модель машины
* QString power – Мощность машины
* QString engine – Двигатель машины
* QString year – Год производства машины
* QString plat – Кузов машины

**MainWindow:**

* Ui::MainWindow \*ui – интерфейс окна
* QCarModel \*t\_model – модель данных
* bool isOpen = false – состояние открытия файла
* bool isSaveAs = false – состояние сохранить как
* Car \*buffer – буффер программы
* QString fileBuffer = "" – буффер для отслеживания файла
* QString findBuffer – буффер направления поиска
* int sizeBuffer – кол-во элементов в буфере программы
* QToolBar \*ToolBar – тулбар программы
* QString filename – имя файла, с которым работает программа
* QAxWidget \*excel – указатель на Excel
* QAxObject \*workbooks – указатель на книги
* QAxObject \*workbook – указатель на директорию откуда грузить книги
* QAxObject \*sheet – указатель на листы
* QAxObject \*cell – указатель на ячейку
* int mRows – кол-во использованные строк в листе Excel
* Car temp – временный объект
* QString data – Данные из ячеек Excel

**DAddCar:**

* Ui::MainWindow \*ui – интерфейс окна

**findDialog:**

* Ui::MainWindow \*ui – интерфейс окна

**QCarModel:**

* Ui::MainWindow \*ui – интерфейс окна
* List value – контейнер с которым работает модель
* Car f – тип данных с которым работает контейнер
* bool isChanged = false – отслеживание изменения модели

**Разработка модулей программы**

**main.cpp**

#include "mainwindow.h"

#include <QApplication>

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

{

QCoreApplication::setOrganizationName(ORGANIZATION\_NAME);

QCoreApplication::setOrganizationDomain(ORGANIZATION\_DOMAIN);

QCoreApplication::setApplicationName(APPLICATION\_NAME);

QApplication a(argc, argv);

QSplashScreen \*splash =

new QSplashScreen(QPixmap(":images/splash.png"));

splash->showMessage(QObject::tr("Загружается, пожалуйста подождите..."),

Qt::AlignRight | Qt::AlignBottom, Qt::white);

splash->show();

MainWindow w;

w.show();

splash->finish(&w);

delete splash;

return a.exec();

}

**List.h**

#ifndef LIST\_H

#define LIST\_H

#include "Cars.h"

class **List**

{

private:

class **Node** {

public:

Node \* pNext;

Node \* pPrev;

Car data;

**Node**(Car data = Car(), Node \*pNext = nullptr, Node \*pPrev = nullptr) {

this->data = data;

this->pNext = pNext;

this->pPrev = pPrev;

}

};

int Size;

Node \* head;

Node \* tail;

public:

class **Iterator** {

private:

Node \* head;

public:

**Iterator**(Node \*head) {

this->head = head;

}

Iterator operator++(int) { Iterator itr = head;

head = head->pNext;

return itr; }

Iterator operator--(int) {

Iterator it = head;

head = head->pPrev;

return it;

}

Car& operator\*() { return head->data; }

bool operator ==(Iterator const rhs) const {

return head == rhs.head;

}

bool operator !=(Iterator const rhs) const {

return !(\*this == rhs);

}

Node \***GetHead**() {

return head;

}

};

**List**():head(nullptr),tail(nullptr),Size(0){}

~**List**();

void **push\_back**(Car var);

void **pop\_back**();

void **push\_front**(Car var);

void **pop\_front**();

void **clear**();

Iterator **begin**() const { return Iterator(head); }

Iterator **end**() const { return Iterator(tail); }

Car operator[](int index) const {

if (index == 0) {

return head->data;

}

if (index <= Size / 2) {

int count = 0;

Node \*temp = head;

while (count != index) {

temp = temp->pNext;

count++;

}

return temp->data;

}

else {

int count = Size-1;

Node \*temp = tail;

while (count != index) {

temp = temp->pPrev;

count--;

}

return temp->data;

}

}

int **GetSize**() const;

void **setCar**(int indexr, int indexc, QString data);

void **erase**(Iterator it);

};

#endif // LIST\_H

#endif // LIST\_H

**List.cpp**

#include "List.h"

List::~**List**()

{

clear();

}

void List::**push\_front**(Car var)

{

if(Size == 0){

head = tail = new Node(var, head, tail);

Size++;

}

else if (Size == 1) {

head = new Node(var, head, nullptr);

tail->pPrev = head;

tail->pNext = nullptr;

Size++;

}

else {

Node \*temp = head;

head = new Node(var, head, nullptr);

temp->pPrev = head;

Size++;

}

}

void List::**push\_back**(Car var) {

if(Size == 0){

head = tail = new Node(var, head, tail);

Size++;

}

else if (Size == 1) {

tail = new Node(var, nullptr, head);

head->pNext = tail;

head->pPrev = nullptr;

Size++;

}

else

{

Node \*temp = tail;

tail = new Node(var, nullptr, temp);

head->pPrev = tail;

temp->pNext = tail;

Size++;

}

}

void List::**pop\_front**() {

if (Size == 1) {

Node\* temp = head;

delete temp;

Size--;

}

else{

Node \*temp = head;

head = head->pNext;

delete temp;

Size--;

}

}

void List::**pop\_back**() {

if (Size > 2) {

Node \*temp = tail;

tail = tail->pPrev;

delete temp;

Size--;

}

else if (Size == 2) {

Node \*temp = tail;

tail = head;

delete temp;

Size--;

}

else

{

pop\_front();

}

}

void List::**clear**() {

while (Size) {

pop\_front();

}

}

int List::**GetSize**()const {

return Size;

}

void List::**erase**(Iterator it)

{

if (it == tail) {

pop\_back();

}

else if (it == head) {

pop\_front();

}

else {

Node \*next = it.GetHead()->pNext;

Node \*prev = it.GetHead()->pPrev;

delete it.GetHead();

Size--;

prev->pNext = next;

next->pPrev = prev;

}

}

void List::**setCar**(int indexr, int indexc, QString data)

{

if (indexr == 0) {

switch (indexc) {

case 0: head->data.SetMake(data); break;

case 1: head->data.SetModel(data); break;

case 2: head->data.SetPower(data); break;

case 3: head->data.SetEngine(data); break;

case 4: head->data.SetYear(data); break;

case 5: head->data.SetPlay(data); break;

}

return;

}

if (indexr <= Size / 2) {

int count = 0;

Node \*temp = head;

while (count != indexr) {

temp = temp->pNext;

count++;

}

switch (indexc) {

case 0: temp->data.SetMake(data); break;

case 1: temp->data.SetModel(data); break;

case 2: temp->data.SetPower(data); break;

case 3:temp->data.SetEngine(data); break;

case 4: temp->data.SetYear(data); break;

case 5: temp->data.SetPlay(data); break;

}

return;

}

else {

int count = Size - 1;

Node \*temp = tail;

while (count != indexr) {

temp = temp->pPrev;

count--;

}

switch (indexc) {

case 0: temp->data.SetMake(data); break;

case 1: temp->data.SetModel(data); break;

case 2: temp->data.SetPower(data); break;

case 3:temp->data.SetEngine(data); break;

case 4: temp->data.SetYear(data); break;

case 5: temp->data.SetPlay(data); break;

}

return;

}

}

**Cars.h**

#ifndef CARS\_H

#define CARS\_H

#include <iostream>

#include <string>

#include <QString>

using namespace std;

class **Car**

{

public:

**Car**();

**Car**(QString make, QString model, QString power, QString engine, QString year, QString plat) : make(make),model(model),power(power),engine(engine),year(year),plat(plat) {}

**Car**(const Car &other);

// string &operator[](int index);

QString **GetMake**();

QString **GetModel**();

QString **GetPower**();

QString **GetEngine**();

QString **GetYear**();

QString **GetPlat**();

void **SetMake**(QString name);

void **SetModel**(QString model);

void **SetPower**(QString power);

void **SetEngine**(QString engine);

void **SetYear**(QString year);

void **SetPlay**(QString plat);

QString operator[](int index) const;

~**Car**();

private:

QString make;

QString model;

QString power;

QString engine;

QString year;

QString plat;

};

#endif // CARS\_H

**Cars.cpp**

#include "Cars.h"

Car::**Car**()

{

make = "null";

model = "null";

power = "null";

engine = "null";

year = "null";

plat = "null";

}

QString Car::**GetEngine**() {

return engine;

}

Car::**Car**(const Car & other)

{

this->model = other.model;

this->engine = other.engine;

this->make = other.make;

this->year = other.year;

this->plat = other.plat;

this->power = other.power;

}

QString Car::**GetMake**() {

return make;

}

QString Car::**GetModel**(){

return model;

}

QString Car::**GetPower**() {

return power;

}

QString Car::**GetYear**() {

return year;

}

QString Car::**GetPlat**() {

return plat;

}

void Car::**SetEngine**(QString engine) {

this->engine = engine;

}

void Car::**SetMake**(QString name) {

make = name;

}

void Car::**SetModel**(QString model) {

this->model = model;

}

void Car::**SetPower**(QString power) {

this->power = power;

}

void Car::**SetYear**(QString year) {

this->year = year;

}

void Car::**SetPlay**(QString plat) {

this->plat = plat;

}

QString Car::operator[](int index) const

{

switch (index) {

case 0: return make; break;

case 1: return model; break;

case 2: return power; break;

case 3: return engine; break;

case 4: return year; break;

case 5: return plat; break;

}

}

Car::~**Car**()

{

}

**mainwindow.h**

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

#include <QStatusBar>

#include <QCloseEvent>

#include <QLabel>

#include <QMessageBox>

#include <QFileDialog>

#include <QString>

#include <QFile>

#include <QTextStream>

#include <ActiveQt/qaxobject.h>

#include <ActiveQt/qaxwidget.h>

#include <QSplashScreen>

#include "qcarmodel.h"

#include "daddcar.h"

#include "finddialog.h"

#define ORGANIZATION\_NAME "Kolen Inc"

#define ORGANIZATION\_DOMAIN "SPBSUT"

#define APPLICATION\_NAME "CarBase"

class **Spreadsheet**;

class **QCarModel**;

namespace **Ui** {

class **MainWindow**;

}

class **MainWindow** : public QMainWindow

{

Q\_OBJECT

friend class QCarModel;

public:

explicit **MainWindow**(QWidget \*parent = 0);

~***MainWindow***();

protected:

void ***contextMenuEvent***(QContextMenuEvent \*event);

void ***closeEvent***(QCloseEvent \*event);

private slots:

void **receiveData**(QString make,QString model, QString power, QString engine, QString year, QString plat); // взаимодейтсвие с диалогом DAddCar

void **removeRows**();

void **getFind**(int row,QString str);

void **open**();

bool **saveFile**();

bool **saveAs**();

bool **save**();

void **copy**();

void **paste**();

void **cut**();

void **append**();

void **about**();

void **aboutQt**();

void **faq**();

void **nfindpush**();

void **pfindpush**();

void **newFile**();

private:

void **createToolBar**();

void **setConnection**();

void **setStatusBar**();

bool **maybeSave**();

void **Excel**();

void **setCurrentFile**(const QString &Name);

private:

Ui::MainWindow \*ui;

QCarModel \*t\_model;

bool isOpen = false;

bool isSaveAs = false;

Car \*buffer;

QString fileBuffer = "";

QString findBuffer;

int sizeBuffer;

QToolBar \*ToolBar;

//QStatusBar statusBar;

QString fileName;

// Всё что ниже нужно для работы с файлами Excel

QAxWidget \*excel;

QAxObject \*workbooks;

QAxObject \*workbook;

QAxObject \*sheet;

QAxObject \*cell;

int mRows;

Car temp;

QString data;

};

#endif // MAINWINDOW\_H

**mainwindow.cpp**

#include "mainwindow.h"

#include "ui\_mainwindow.h"

MainWindow::**MainWindow**(QWidget \*parent) :

QMainWindow(parent),

ui(new Ui::MainWindow)

{

setConnection();

}

MainWindow::~***MainWindow***()

{

delete ui;

}

void MainWindow::**setConnection**(){

buffer = new Car;

t\_model = new QCarModel;

ui->setupUi(this);

ui->tableView->*setModel*(t\_model);

ui->tableView->setSelectionBehavior(QAbstractItemView::SelectRows);

ui->tableView->horizontalHeader()->setSectionResizeMode(QHeaderView::Stretch);

ui->tableView->setColumnWidth(2, 250);

ui->tableView->setColumnWidth(4, 150);

DAddCar \*daddcar = new DAddCar();

findDialog \*fndD = new findDialog();

setStatusBar();

QString path("/buffer");

QDir dir;

if (!dir.exists(path))

dir.mkpath(path);

QFile file("c://buffer//buffer.xlsx");

file.*open*(QIODevice::ReadWrite); // Or QIODevice::ReadWrite

file.*close*();

connect(ui->about,SIGNAL(triggered()),this,SLOT(about()));

connect(ui->aboutQt,SIGNAL(triggered()),this,SLOT(aboutQt()));

connect(ui->FAQ,SIGNAL(triggered()),this,SLOT(faq()));

connect(ui->add,SIGNAL(triggered()),daddcar,SLOT(show()));

connect(ui->copy,SIGNAL(triggered()),this,SLOT(copy()));

connect(ui->cut,SIGNAL(triggered()),this,SLOT(cut()));

connect(ui->find,SIGNAL(triggered()),fndD,SLOT(show()));

connect(ui->open,SIGNAL(triggered()),this,SLOT(open()));

connect(ui->paste,SIGNAL(triggered()),this,SLOT(paste()));

connect(ui->save,SIGNAL(triggered()),this,SLOT(save()));

connect(daddcar,SIGNAL(sendData(QString,QString,QString,QString,QString,QString)),this,SLOT(receiveData(QString,QString,QString,QString,QString,QString)));

connect(fndD,SIGNAL(SfindData(int,QString)),this,SLOT(getFind(int,QString)));

connect(ui->findNext,SIGNAL(triggered()),this,SLOT(nfindpush()));

connect(ui->findPrev,SIGNAL(triggered()),this,SLOT(pfindpush()));

connect(ui->new1,SIGNAL(triggered()),this,SLOT(newFile()));

connect(ui->append,SIGNAL(triggered()),this,SLOT(append()));

connect(ui->saveAs,SIGNAL(triggered()),this,SLOT(saveAs()));

connect(ui->tableView, SIGNAL(customContextMenuRequested(QPoint)), this, SLOT(slotCustomMenuRequested(QPoint)));

setCurrentFile("");

Excel();

}

void MainWindow::**Excel**(){ // Данная функция нужна, чтобы оптимизировать открытие файла, вместо того, чтобы использовать поток, мы открываем эксель при создании главного окна

excel = new QAxWidget("Excel.Application");

excel->setProperty("DisplayAlerts", "0");

workbooks = excel->querySubObject("WorkBooks");/\* querySubObject возвращает указатель

workbooks->dynamicCall("Add");

workbook=excel->querySubObject("ActiveWorkBook");

workbook->dynamicCall("SaveAs (const QString&)", "c:\\buffer\\buffer.xlsx");

workbook->dynamicCall("Close (Boolean)", false);

workbook->*clear*();

delete workbook;

workbook = NULL;

workbook = workbooks->querySubObject("Open (const QString&)","c:\\buffer\\buffer.xlsx");

}

void MainWindow::**receiveData**(QString make,QString model, QString power, QString engine, QString year, QString plat ){

t\_model->addCar(make,model,power,engine,year,plat);

}

void MainWindow::**nfindpush**(){

emit getFind(1,findBuffer);

}

void MainWindow::**pfindpush**(){

emit getFind(2,findBuffer);

}

void MainWindow::**removeRows**(){

QModelIndexList indexes = ui->tableView->selectionModel()->selectedRows();

int countRow = indexes.count();

if(countRow!=0){

QModelIndex deleteindex = ui->tableView->selectionModel()->selectedRows().first();

QModelIndex deleteindex2 = ui->tableView->selectionModel()->selectedRows().last();

if(deleteindex.row() > deleteindex2.row()) deleteindex = deleteindex2;

while(countRow>0){

t\_model->remove( deleteindex.row());

countRow--;

}

}

}

void MainWindow::**copy**(){

QModelIndexList indexes=ui->tableView->selectionModel()->selectedRows();

sizeBuffer = indexes.count();

if(sizeBuffer!=0){

delete buffer;

buffer = new Car[sizeBuffer];

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

buffer[i] = t\_model->value[indexes.at(i).row()];

}

statusBar()->showMessage("Данные занесены в буффер");

}

}

void MainWindow::**paste**(){

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

t\_model->addExcelCar(buffer[i]);

}

statusBar()->showMessage("Данные успешно добавлены");

}

void MainWindow::**cut**(){

copy();

removeRows();

statusBar()->showMessage("Данные занесены в буффер");

}

void MainWindow::**getFind**(int row, QString str){ // row == 0(findAll), 1(findNext), 2(findPrev) ; str(text,which should will be has find);

findBuffer = str;

bool isFind = false;

int k;

if(ui->tableView->selectionModel()->hasSelection())

k = ui->tableView->selectionModel()->selectedRows().last().row();

else k = -1;

if(row == 0){

for(int i=0;i<5;i++){

for(int x=0;x<=t\_model->*rowCount*(QModelIndex());x++){

QModelIndex ind = t\_model->*index*(x,i);

if(ind.data().toString()==findBuffer){

isFind = true;

ui->tableView->*scrollTo*(ind);

ui->tableView->selectRow(ind.row());

break;

}

}

}

}

else if(row == 1){

for(int x=k+1;x<=t\_model->*rowCount*(QModelIndex());x++){

if(isFind == true) break;

for(int i=0;i<5;i++){

QModelIndex ind = t\_model->*index*(x,i);

if(ind.data().toString()==findBuffer){

isFind = true;

ui->tableView->*scrollTo*(ind);

ui->tableView->selectRow(ind.row());

break;

}

}

}

}

else{

for(int x=k-1;x>=0;x--){

if(isFind == true) break;

for(int i=0;i<5;i++){

QModelIndex ind = t\_model->*index*(x,i);

if(ind.data().toString()==findBuffer){

isFind = true;

ui->tableView->*scrollTo*(ind);

ui->tableView->selectRow(ind.row());

break;

}

}

}

}

if(isFind == false){

QMessageBox::information(this,"Информация","'"+str+"' не найдено");

}

}

void MainWindow::**newFile**(){

if(t\_model->isChanged == true) {

maybeSave();

}

fileBuffer="";

t\_model->clearTable();

t\_model->isChanged=false;

statusBar()->showMessage("Новый файл успешно создан");

}

void MainWindow::**open**(){

fileName = QFileDialog::getOpenFileName( this,QString("Открыть файл"), QString(),QString("MS Excel (\*.xlsx)"));

if(!fileName.isEmpty()){

if(t\_model->isChanged == true) {

bool s = maybeSave();

if(s==false) return;

}

if(isOpen==false){

setCurrentFile(fileName);

fileBuffer=fileName;

t\_model->clearTable();

workbook = workbooks->querySubObject("Open (const QString&)", fileName);

sheet = workbook->querySubObject("Worksheets(int)", 1);

QAxObject \*usedRange = sheet->querySubObject("UsedRange");

QAxObject \*usedRows = usedRange->querySubObject("Rows");

QAxObject \*usedCols = usedRange->querySubObject("Columns");

mRows = usedRows->property("Count").toInt();

if(mRows != 0) {// Если файл не пуст, то считать данные

for(int i = 2;i<=mRows+1;i++){ // Начинаем с 2 индекса, так как 1 строка отвечает за хедер

for(int j = 1; j<6; j++){

cell = sheet->querySubObject("Cells(QVariant,QVariant)", i, j);

data = cell->property("Value").toString();

switch(j){

case 1: temp.SetMake(data);

case 2: temp.SetModel(data);

case 3: temp.SetPower(data);

case 4: temp.SetEngine(data);

case 5: temp.SetYear(data);

case 6: temp.SetPlay(data); break;

}

}

t\_model->addExcelCar(temp);

}

cell->*clear*();

delete cell;

cell = NULL;

}

delete usedCols;

delete usedRows;

delete usedRange;

isOpen = true;

statusBar()->showMessage("Файл успешно открыт");

}

else{

fileBuffer=fileName;

setCurrentFile(fileName);

t\_model->clearTable();

sheet->*clear*();

delete sheet;

sheet = NULL;

workbook->*clear*();

delete workbook;

workbook = NULL;

workbook = workbooks->querySubObject("Open (const QString&)", fileName);

sheet = workbook->querySubObject("Worksheets(int)", 1);

QAxObject \*usedRange = sheet->querySubObject("UsedRange");

QAxObject \*usedRows = usedRange->querySubObject("Rows");

QAxObject \*usedCols = usedRange->querySubObject("Columns");

mRows = usedRows->property("Count").toInt();

if(mRows != 0) {// Если файл не пуст, то считать данные

for(int i = 2;i<=mRows+1;i++){ // Начинаем с 2 индекса, так как 1 строка отвечает за хедер

for(int j = 1; j<6; j++){

cell = sheet->querySubObject("Cells(QVariant,QVariant)", i, j);

data = cell->property("Value").toString();

switch(j){

case 1: temp.SetMake(data);

case 2: temp.SetModel(data);

case 3: temp.SetPower(data);

case 4: temp.SetEngine(data);

case 5: temp.SetYear(data);

case 6: temp.SetPlay(data); break;

}

}

t\_model->addExcelCar(temp);

}

cell->*clear*();

delete cell;

cell = NULL;

}

delete usedCols;

delete usedRows;

delete usedRange;

t\_model->isChanged=false;

statusBar()->showMessage("Файл успешно открыт");

}

}

else{

statusBar()->showMessage("Ошибка в открытии файла");

}

}

void MainWindow::**append**(){

fileName = QFileDialog::getOpenFileName( this,QString("Открыть файл"), QString(),QString("MS Excel (\*.xlsx)"));

if(!fileName.isEmpty()){

if(fileBuffer!=nullptr){

sheet->*clear*();

delete sheet;

sheet = NULL;

workbook->*clear*();

delete workbook;

workbook = NULL;

}

workbook = workbooks->querySubObject("Open (const QString&)", fileName);

sheet = workbook->querySubObject("Worksheets(int)", 1);

QAxObject \*usedRange = sheet->querySubObject("UsedRange");

QAxObject \*usedRows = usedRange->querySubObject("Rows");

QAxObject \*usedCols = usedRange->querySubObject("Columns");

mRows = usedRows->property("Count").toInt();

if(mRows != 0) {// Если файл не пуст, то считать данные

for(int i = 2;i<=mRows+1;i++){ // Начинаем с 2 индекса, так как 1 строка отвечает за хедер

for(int j = 1; j<6; j++){

cell = sheet->querySubObject("Cells(QVariant,QVariant)", i, j);

data = cell->property("Value").toString();

switch(j){

case 1: temp.SetMake(data);

case 2: temp.SetModel(data);

case 3: temp.SetPower(data);

case 4: temp.SetEngine(data);

case 5: temp.SetYear(data);

case 6: temp.SetPlay(data); break;

}

}

t\_model->addExcelCar(temp);

}

cell->*clear*();

delete cell;

cell = NULL;

}

delete usedCols;

delete usedRows;

delete usedRange;

t\_model->isChanged=true;

statusBar()->showMessage("Таблицы успешно объединены");

sheet->*clear*();

delete sheet;

sheet = NULL;

workbook->*clear*();

delete workbook;

workbook = NULL;

fileName=fileBuffer;

if(fileBuffer!=nullptr){

// восстанавливаем работу с прошлым файлом, если она была

workbook = workbooks->querySubObject("Open (const QString&)", fileName);

sheet = workbook->querySubObject("Worksheets(int)", 1);

}

}

else

{

fileName=fileBuffer;

statusBar()->showMessage("Ошибка в открытии файла");

}

}

void MainWindow::***closeEvent***(QCloseEvent \*event){

if(maybeSave()){

event->accept();

}

else{

event->ignore();

return;

}

if(fileName!=nullptr){

sheet->*clear*();

delete sheet;

sheet = NULL;

workbook->dynamicCall("Save");

workbook->*clear*();

delete workbook;

workbook = NULL;

workbooks->dynamicCall( "Close" );

workbooks->*clear*();

delete workbooks;

workbooks = NULL;

excel->dynamicCall( "Quit" );

delete excel;

excel = NULL;

}

}

bool MainWindow::**saveFile**(){

if(!fileName.isEmpty()){

if(isSaveAs == true){

if(isOpen==false){

// excel = new QAxWidget("Excel.Application",this);

// excel->setProperty("Visible", false);

// excel->setProperty("DisplayAlerts", "0");

}

else{

sheet->*clear*();

delete sheet;

sheet = NULL;

workbook->*clear*();

delete workbook;

workbook = NULL;

workbooks->dynamicCall( "Close" );

workbooks->*clear*();

delete workbooks;

workbooks = NULL;

}

fileBuffer=fileName;

fileName.replace(QRegExp("/"), "\\");

// workbooks = excel->querySubObject("WorkBooks");

workbooks->dynamicCall("Add");

workbook=excel->querySubObject("ActiveWorkBook");

sheet = workbook->querySubObject("WorkSheets");

workbook->dynamicCall("SaveAs (const QString&)", fileName);

workbook->dynamicCall("Close (Boolean)", false);

// excel->setProperty("DisplayAlerts",1);

// excel->dynamicCall("Quit(void)");

workbook = workbooks->querySubObject("Open (const QString&)", fileName);

sheet = workbook->querySubObject("Worksheets(int)", 1);

isSaveAs=false;

t\_model->isChanged=false;

}

List::Iterator it = t\_model->value.begin();

if( it.GetHead()!=nullptr ){ // делаем проверку, если были добавлены элементы в лист, то заносим данные в файл

if(mRows>t\_model->value.GetSize()){ // Данная проверка нужна для того, чтобы проверить удалили ли мы что-то из таблицы, которую использовали

// Если да, то сначала мы заносим новые данные, а затем стираем старые вторым циклом

for(int i = 2; i<t\_model->value.GetSize()+2;i++){

for(int j = 1; j<7; j++){

cell = sheet->querySubObject("Cells(QVariant,QVariant)", i, j);

cell->setProperty("Value",t\_model->value[i-2][j-1]);

}

}

for(int i = t\_model->value.GetSize()+2;i<mRows+2;i++){

for(int j =1; j<7; j++){

cell = sheet->querySubObject("Cells(QVariant,QVariant)", i, j);

cell->setProperty("Value"," ");

}

}

}

else{

for(int i = 2; i<t\_model->value.GetSize()+2;i++){

for(int j = 1; j<7; j++){

cell = sheet->querySubObject("Cells(QVariant,QVariant)", i, j);

cell->setProperty("Value",t\_model->value[i-2][j-1]);

}

}

}

cell->*clear*();

delete cell;

cell = NULL;

}

t\_model->isChanged=false;

statusBar()->showMessage("Файл успешно сохранён");

return true;

}

else{

return false;

}

}

bool MainWindow::**saveAs**(){

fileName = QFileDialog::getSaveFileName(this,QString("Открыть файл"), QString(),QString("MS Excel (\*.xlsx)") );

fileBuffer=fileName;

setCurrentFile(fileName);

if(fileName.isEmpty())

return false;

isSaveAs=true;

return saveFile();

}

void MainWindow::**setStatusBar**()

{

#ifndef QT\_NO\_STATUSBAR

( void )statusBar();

#endif

}

bool MainWindow::**maybeSave**(){

if (t\_model->isChanged==true) {

int ret = QMessageBox::warning(this, tr("CarBase"),

tr("The document has been modified.\n"

"Do you want to save your changes?"),

QMessageBox::Yes | QMessageBox::Default,

QMessageBox::No,

QMessageBox::Cancel | QMessageBox::Escape);

if (ret == QMessageBox::Yes)

return save();

else if (ret == QMessageBox::Cancel)

return false;

} return true;

}

bool MainWindow::**save**(){

if( fileName == nullptr){

return saveAs();

}

else{

saveFile();

}

}

void MainWindow::**setCurrentFile**(const QString &Name){

fileName=Name;

// t\_model->isChanged=false;

QString shownName;

if(fileName==nullptr){

shownName = tr("untitled.xlsx");}

else{

shownName = QFileInfo(fileName).fileName();

}

setWindowTitle(tr("%1[\*]--%2").arg(shownName).arg(tr("CarBase")));

}

void MainWindow::***contextMenuEvent***(QContextMenuEvent \*event){

QMenu menu(this);

menu.addAction("Вырезать",this,SLOT(cut()));

menu.addAction("Копировать",this,SLOT(copy()));

menu.addSeparator();

menu.addAction("Вставить",this,SLOT(paste()));

menu.addAction("Удалить", this, SLOT(removeRows()));

menu.addSeparator();

menu.addAction("Exit", qApp, SLOT(qApp->exit()));

menu.exec(event->globalPos());

}

void MainWindow::**about**(){

QMessageBox::about(this,"О программе","Данная программа создана по заданию курсового проекта.\nСоздать базу данных, работающую на основе контейнера, который следует разработать в процессе выполнения курсовой работы. В качестве прототипа контейнера можно использовать контейнеры vector или list из библиотеки STL. Для работы с контейнером следует предусмотреть итератор. Остальные требования к контейнеру определяются студентом самостоятельно.Разрабатываемая база данных должна быть создана использованием C++ и библиотеки Qt4. Среда программировании Qt Creater.");

}

void MainWindow::**aboutQt**(){

QMessageBox::aboutQt(this,"О Qt");

}

void MainWindow::**faq**(){

QMessageBox::information(this,"FAQ","Данная программа выполняет функцию базы данных.\n База может работать с программой MS Excel(начиная от 2007).\n Во время работы с программой желательно закрыть Excel, в избежании критических ошибок.\n Программа создаёт дополнительный файл для работы на диске C в папке 'buffer'.\n Если нашли ошибки в работе программы отправляйте их на почту kolen322@yandex.ru");

}

**daddcar.h**

#ifndef DADDCAR\_H

#define DADDCAR\_H

#include <QDialog>

namespace **Ui** {

class **DAddCar**;

}

class **DAddCar** : public QDialog

{

Q\_OBJECT

friend class MainWindow;

public:

explicit **DAddCar**(QWidget \*parent = 0);

~***DAddCar***();

signals:

void **sendData**(QString model, QString make, QString power, QString engine, QString year, QString plat);

private slots:

void **on\_okButton\_toggled**(bool checked);

void **okButtonPush**();

private:

Ui::DAddCar \*ui;

};

#endif // DADDCAR\_H

**daddcar.cpp**

#include "daddcar.h"

#include "ui\_daddcar.h"

DAddCar::**DAddCar**(QWidget \*parent) :

QDialog(parent),

ui(new Ui::DAddCar)

{

ui->setupUi(this);

connect(ui->okButton,SIGNAL(clicked()),this,SLOT(okButtonPush()));

}

DAddCar::~***DAddCar***()

{

delete ui;

}

void DAddCar::**on\_okButton\_toggled**(bool checked)

{

}

void DAddCar::**okButtonPush**(){

emit sendData(ui->lineMake->text(),ui->lineModel->text(),ui->linePower->text(),ui->lineEngine->text(),ui->lineYear->text(),ui->linePlat->text());

ui->lineEngine->clear();

ui->lineMake->clear();

ui->lineModel->clear();

ui->linePower->clear();

ui->lineYear->clear();

ui->linePlat->clear();

}

**finddialog.h**

#ifndef FINDDIALOG\_H

#define FINDDIALOG\_H

#include <QDialog>

#include "qcarmodel.h"

namespace **Ui** {

class **findDialog**;

}

class **findDialog** : public QDialog

{

Q\_OBJECT

public:

explicit **findDialog**(QWidget \*parent = 0);

~***findDialog***();

signals:

void **SfindData**(int check, QString str);

private slots:

void **okButtonPush**();

private:

Ui::findDialog \*ui;

};

#endif // FINDDIALOG\_H

**finddialog.cpp**

#include "finddialog.h"

#include "ui\_finddialog.h"

findDialog::**findDialog**(QWidget \*parent) :

QDialog(parent),

ui(new Ui::findDialog)

{

ui->setupUi(this);

connect(ui->okButton,SIGNAL(clicked()),this,SLOT(okButtonPush()));

}

findDialog::~***findDialog***()

{

delete ui;

}

void findDialog::**okButtonPush**(){

if(ui->allText->isChecked()){

emit SfindData(0,ui->findEdit->text());

}

else if(ui->nextText->isChecked()){

emit SfindData(1,ui->findEdit->text());

}

else {

emit SfindData(2,ui->findEdit->text());

}

}

**qcarmodel.h**

#ifndef QCARMODEL\_H

#define QCARMODEL\_H

#include "Cars.h"

#include "List.h"

#include "daddcar.h"

#include <QAbstractTableModel>

#include <QVariant>

#include <QString>

#include <QItemSelectionModel>

#include <QModelIndexList>

class **QCarModel** : public QAbstractTableModel

{

Q\_OBJECT

friend class MainWindow;

public:

**QCarModel**( QObject\* parent = 0 );

int ***rowCount***(const QModelIndex &parent) const;

int ***columnCount***(const QModelIndex &parent) const;

QVariant ***headerData***( int section, Qt::Orientation orientation, int role ) const;

QVariant ***data***(const QModelIndex &index, int role) const;

bool ***setData***( const QModelIndex& index, const QVariant& value, int role );

Qt::ItemFlags ***flags***( const QModelIndex& index ) const;

void **addCar**(const QString& make,const QString& model,const QString& power,const QString& engine, const QString& year, const QString& plat );

void **remove**(int row);

void **addExcelCar**(Car temp);

void **clearTable**();

private:

List value;

Car f;

bool isChanged = false;

signals:

public slots:

};

#endif // QCARMODEL\_H

**qcarmodel.cpp**

#include "qcarmodel.h"

QCarModel::**QCarModel**( QObject\* parent ) : QAbstractTableModel( parent ) {

}

int QCarModel::***rowCount***(const QModelIndex &parent) const {

value.GetSize();

}

int QCarModel::***columnCount***(const QModelIndex &parent) const{

return 6;

}

QVariant QCarModel::***headerData***(int section, Qt::Orientation orientation, int role) const{

if( role != Qt::DisplayRole ) {

return QVariant();

}

if( orientation == Qt::Vertical ) {

return section;

}

switch( section ) {

case 0:

return trUtf8( "Производитель" );

case 1:

return trUtf8( "Модель" );

case 2:

return trUtf8( "Мощность двигателя" );

case 3:

return trUtf8( "Двигатель" );

case 4:

return trUtf8( "Год производства");

case 5:

return trUtf8( "Кузов");

}

return QVariant();

}

QVariant QCarModel::***data***(const QModelIndex &index, int role) const

{

if(!index.isValid() || value.GetSize() <= index.row() ||

( role != Qt::DisplayRole && role != Qt::EditRole )

) {

return QVariant();

}

return QVariant(value[index.row()][index.column()]);

}

void QCarModel::**addCar**(const QString &make, const QString &model, const QString &power, const QString &engine, const QString &year, const QString &plat){

Car temp;

temp.SetMake(make);

temp.SetModel(model);

temp.SetPower(power);

temp.SetEngine(engine);

temp.SetYear(year);

temp.SetPlay(plat);

int row = value.GetSize();

beginInsertRows( QModelIndex(), row, row );

value.push\_back(temp);

endInsertRows();

isChanged=true;

}

void QCarModel::**addExcelCar**(Car temp){

int row = value.GetSize();

beginInsertRows(QModelIndex(),row,row);

value.push\_back(temp);

endInsertRows();

}

void QCarModel::**clearTable**(){

int k = value.GetSize();

while(k>0){

remove(0);

k--;

}

}

void QCarModel::**remove**(int row) {

int rowselect = 0;

List::Iterator itr = value.begin();

do{

if(rowselect == row){

beginRemoveRows(QModelIndex(),row,row);

value.erase(itr);

endRemoveRows();

break;

}

else{

itr++;

rowselect++;

}

}while(itr!=nullptr);

isChanged=true;

}

Qt::ItemFlags QCarModel::***flags***( const QModelIndex& index ) const {

Qt::ItemFlags flags = QAbstractTableModel::flags( index );

if( index.isValid() ) {

flags |= Qt::ItemIsEditable;

}

return flags;

}

bool QCarModel::***setData***( const QModelIndex& index, const QVariant& value1, int role ) {

if( !index.isValid() || role != Qt::EditRole ) {

return false;

}

value.setCar(index.row(),index.column(),value1.toString());

emit dataChanged( index, index );

isChanged=true;

return true;

}

**Реализация взаимодействия программы с MS Excel**

Взаимодействие программы с Excel стало возможно благодаря каркасу ActiveQt.

Поддержка Qt библиотекой ActiveX и COM, позволяет Qt/Windows программистам:

* Получать доступ и использовать элементы управления ActiveX и объекты COM предоставляемые любым из ActiveX серверов в их Qt приложении.
* Делать доступными Qt приложения, как COM серверы, с любым числом Qt объектов и виджетов, как COM объекты и элементы управления ActiveX.

В данной программе я использовал модуль QAxContainer.

Модуль QAxContainer - это статическая библиотека, реализующая производные от

QObject и QWidget классы QAxObject и QAxWidget, которые реализованы как

контейнеры для COM объектов и элементы упраления ActiveX.

Чтобы подключить библиотеку,вам необходимо в .pro файле (в каталоге с вашим

проектом),дописать LIBS += -lqaxcontainer

При помощи макросов мы открываем Excel, нужную нам книгу, и можем как

считывать данные из таблицы, так и записать в таблицу. Во время реализации

данной задачи я столкнулся с проблемой оптимазиции, т.к открытие excel

требует больших ресурсов компьютера, и в момент открытия файла, программа

блокировалось на несколько секунд. Для оптимизации работы программы, я

создал отдельный метод, в котором открывается Excel с файлом buffer.xlsx всё эт

о происходит в момент загрузочного экрана программы. Тестировании работы про

граммы проводилось с MS Excel 2012 года. Более подробную информацию о рабо

те с макросами вы можете найти на сайте:

<http://www.wiki.crossplatform.ru/index.php/Работа_с_MS_Office_с_помощью_Activ>e

Qt

Нужно внимательно следить за создаваемыми объектами и указателями, возвращаемыми методом querySubObject(). Они не удаляются автоматически, нужно вызывать delete вручную. В противном случае будет эффективно расходоваться память, и после нескольких тысяч вызовов метода querySubObject() ваша программа и эксель в сумме займут всю память, но это полбеды - обращение к одной ячейке будет занимать секунду. Из документации Qt: Returns a pointer to a QAxObject wrapping the COM object provided by the method or property name, passing passing the parameters var1, var1, var2, var3, var4, var5, var6, var7 and var8. The returned QAxObject is a child of this object (which is either of type QAxObject or QAxWidget), and is deleted when this object is deleted. It is however safe to delete the returned object yourself, and you should do so when you iterate over lists of subobjects.

COM enabled applications usually have an object model publishing certain elements of the application as dispatch interfaces. Use this method to navigate the hierarchy of the object model, e.g.

**Результаты тестирования**

Произведено тестирование с методами:

* Добавления/Удаления/Вырезания элементов в контейнере
* Создания файла в приложении
* Открытия файла в приложении
* Сохранения файла в приложении
* Закрытия файла в приложении
* Поиска элемента в таблице
* Добавления элементов из другого файла

Итог: все методы работают корректно. Все тесты были пройдены успешно.

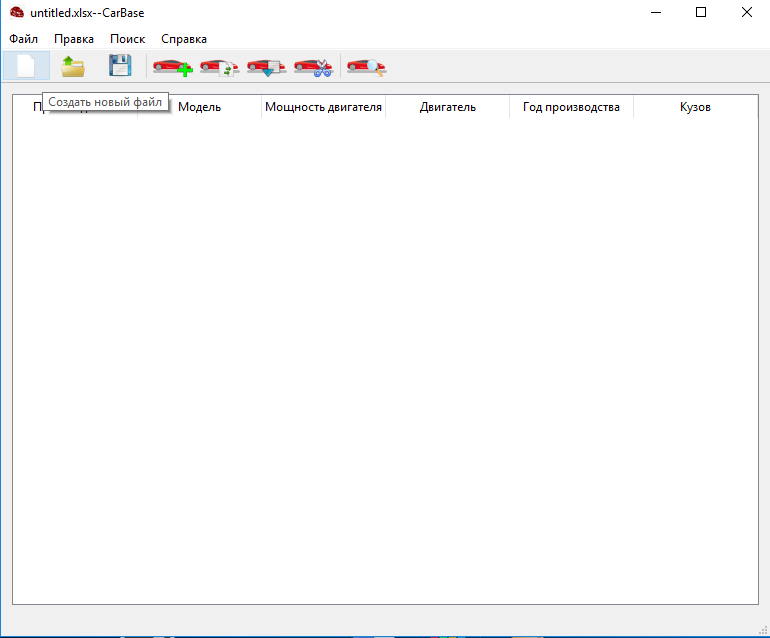
**Инструкция пользователя**

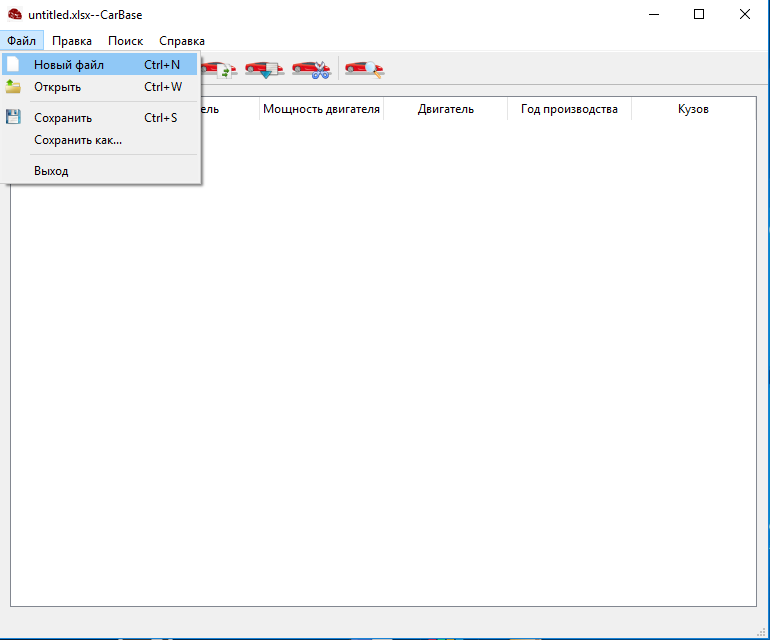
Предупреждение: Не работайте одновременно с программой и Excel, т.к это может вызвать конфликт приложений.

**1. Загрузочный экран**

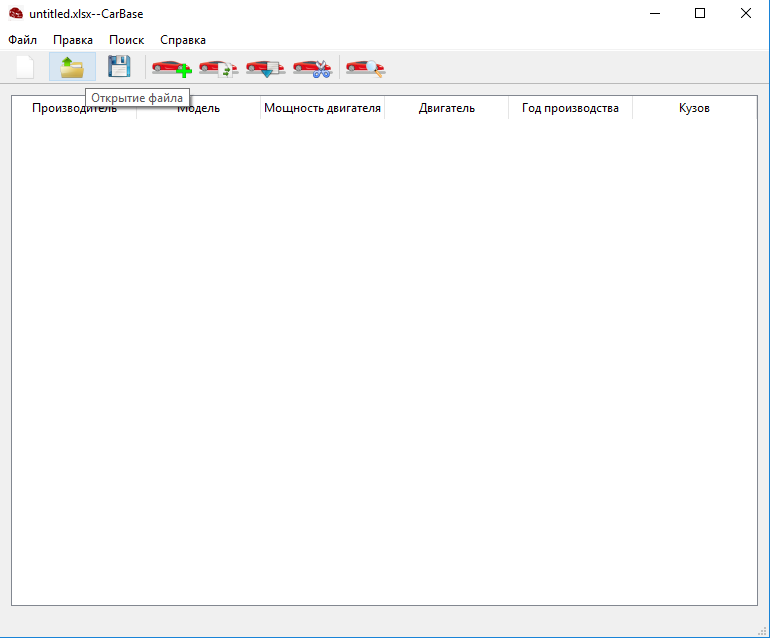


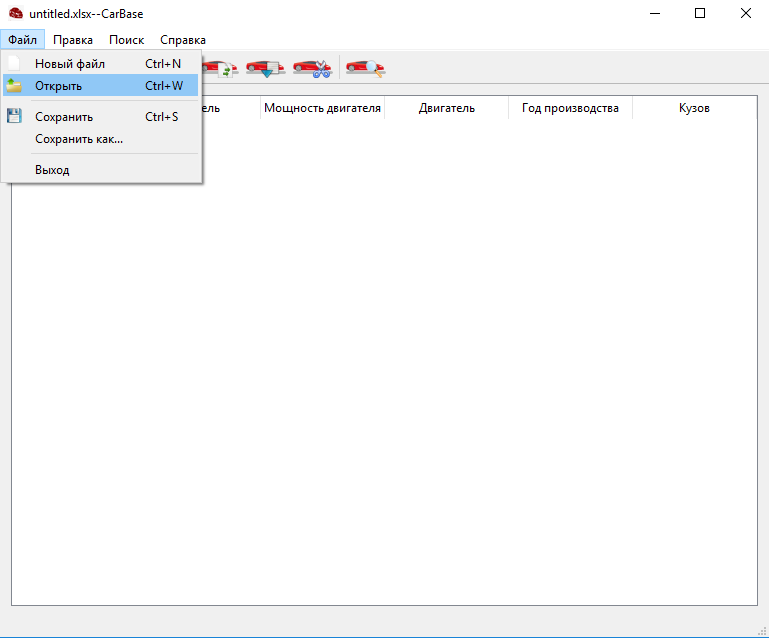
**2. Новый файл**



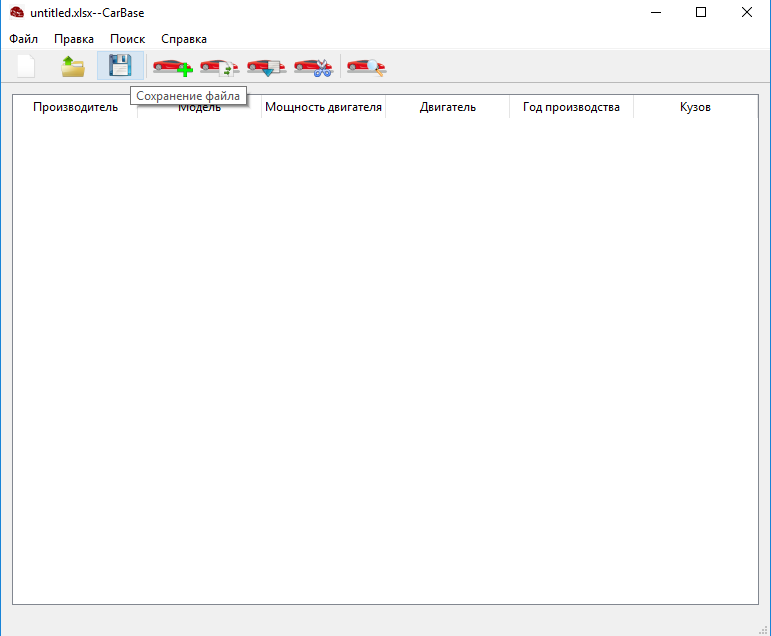


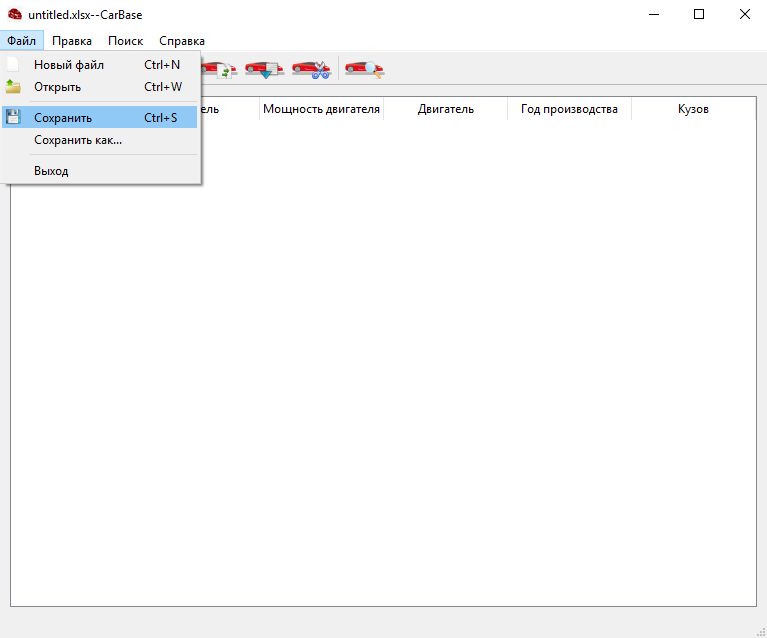
**3. Открыть файл**



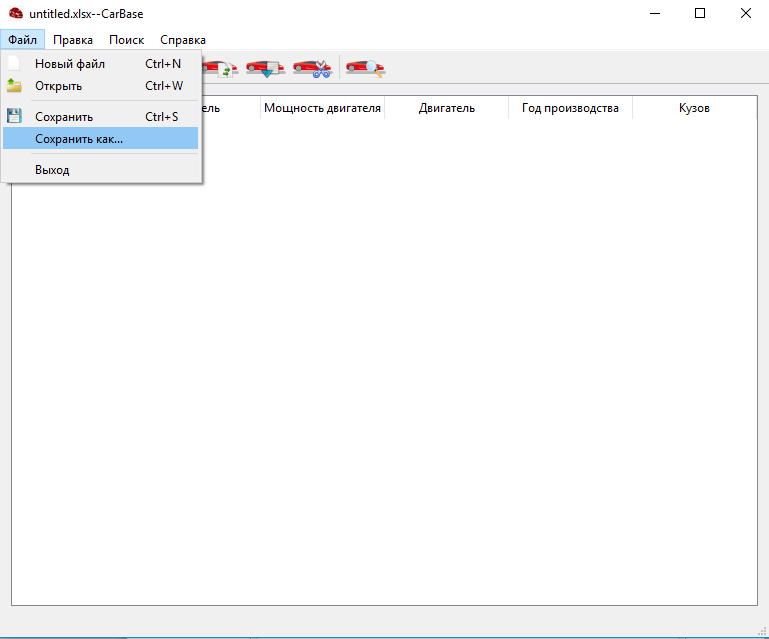


**4. Сохранить файл**

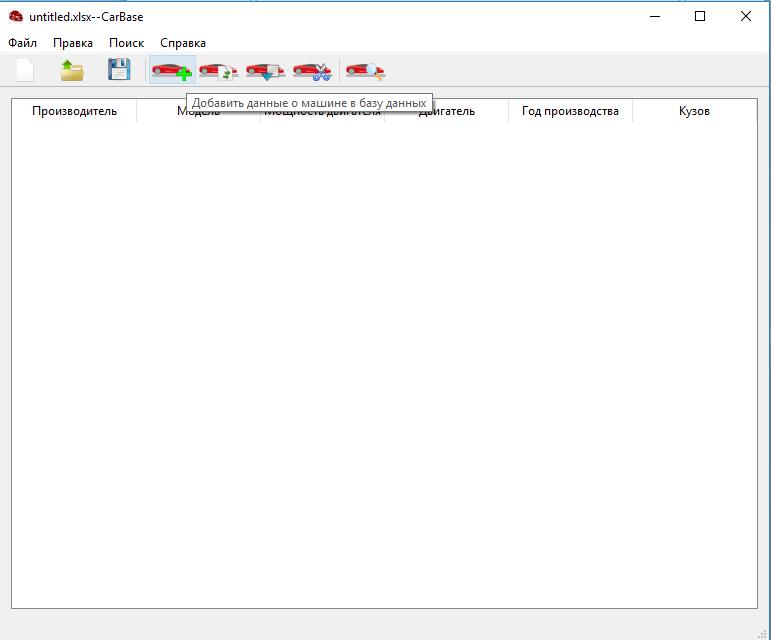


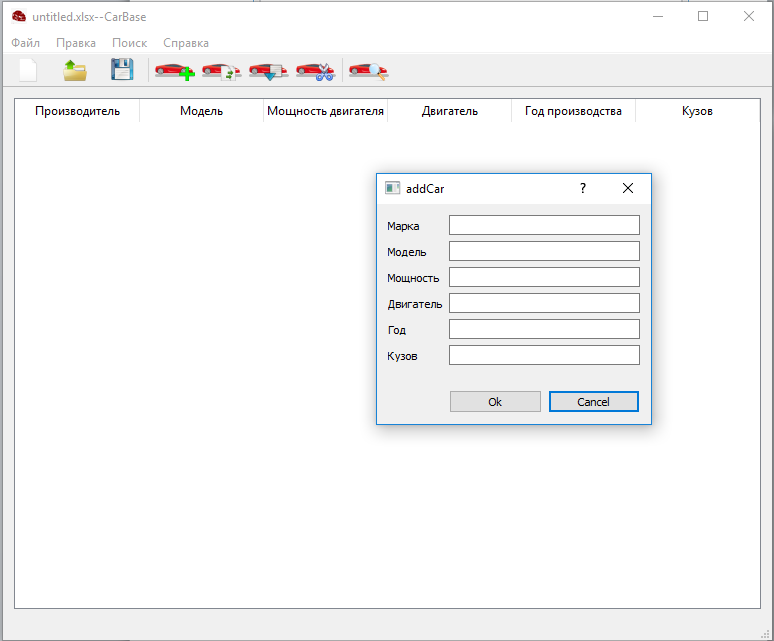


**5. Сохранить как файл**



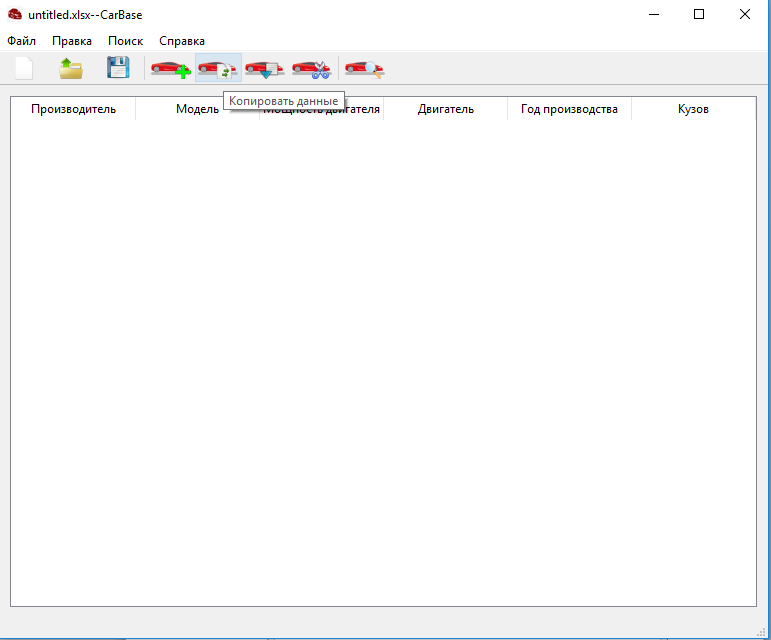
**6. Добавить элемент**



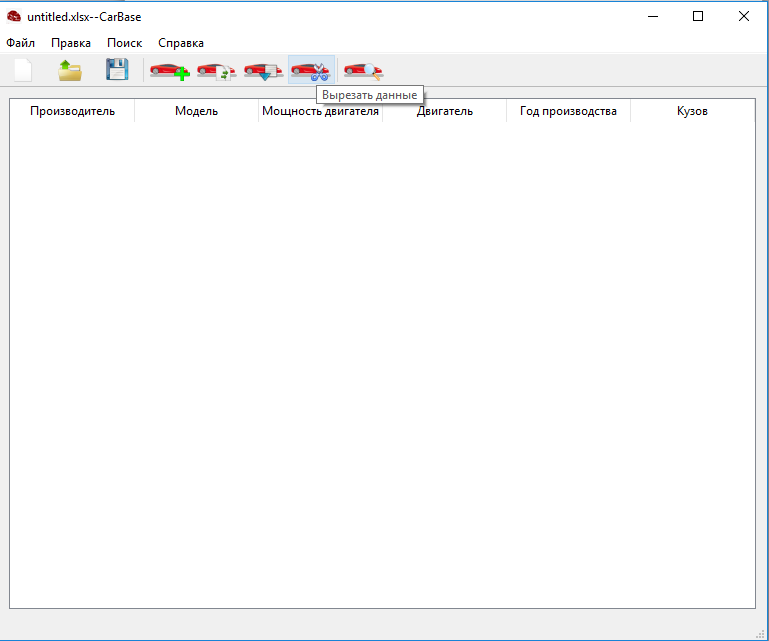


**Ok – добавить, cancel – отмена**

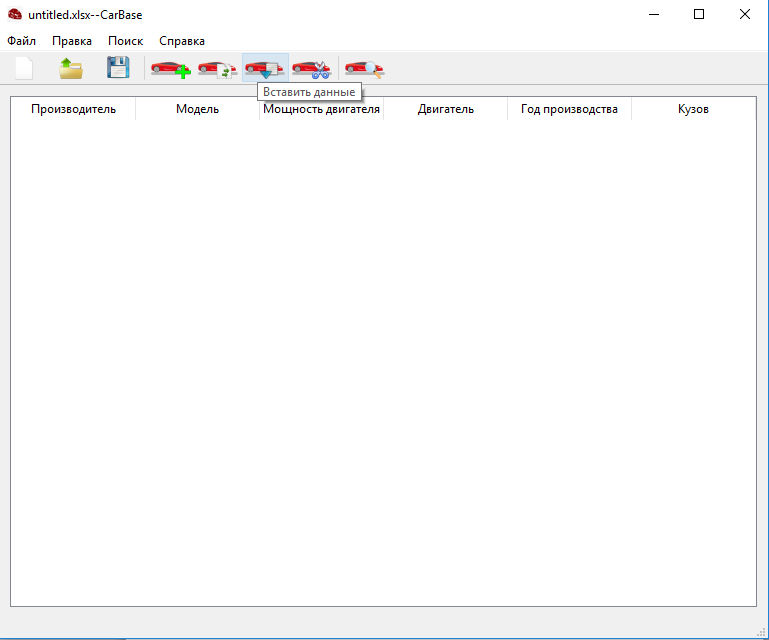
**7. Копировать элемент**



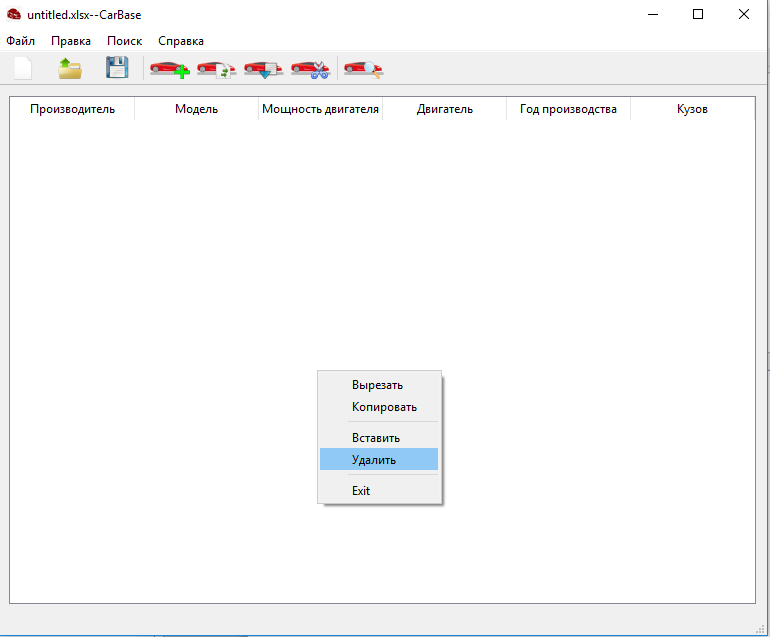
**8. Вырезать элемент**



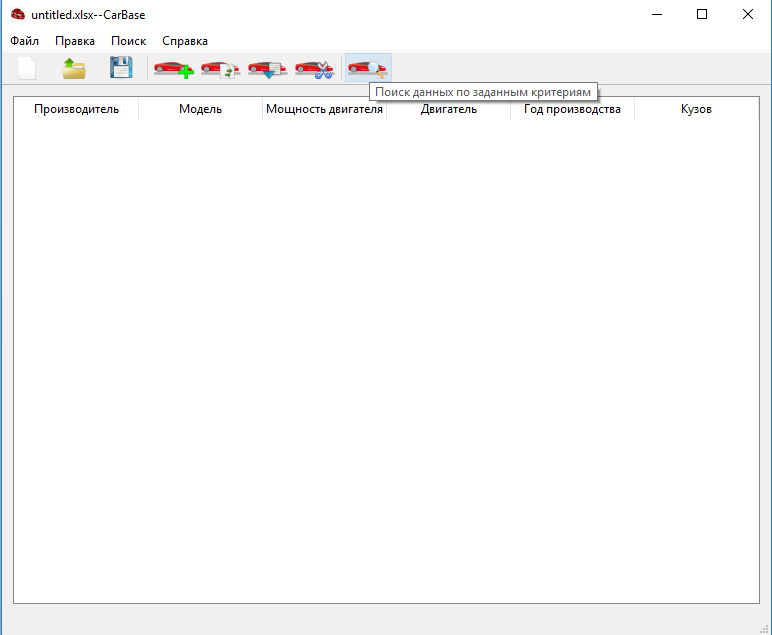
**9. Вставить элемент**

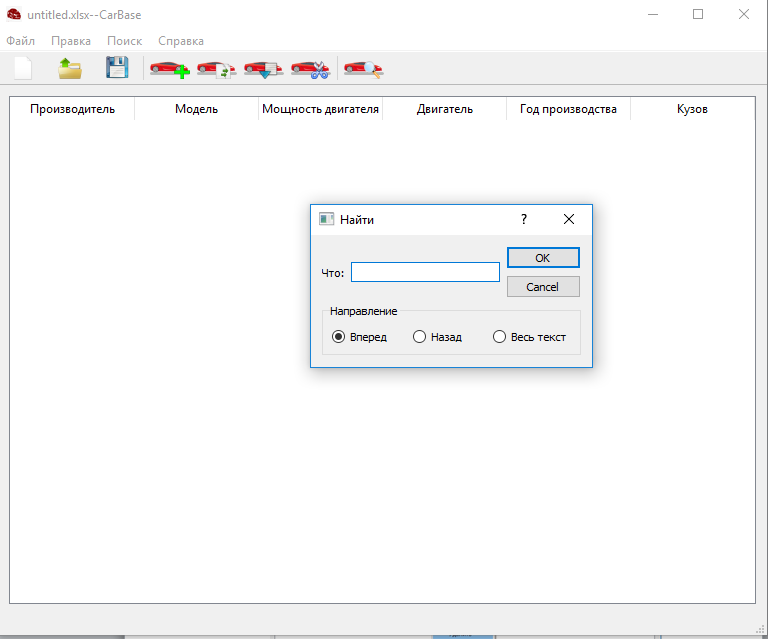


**10.Удалить элемент**



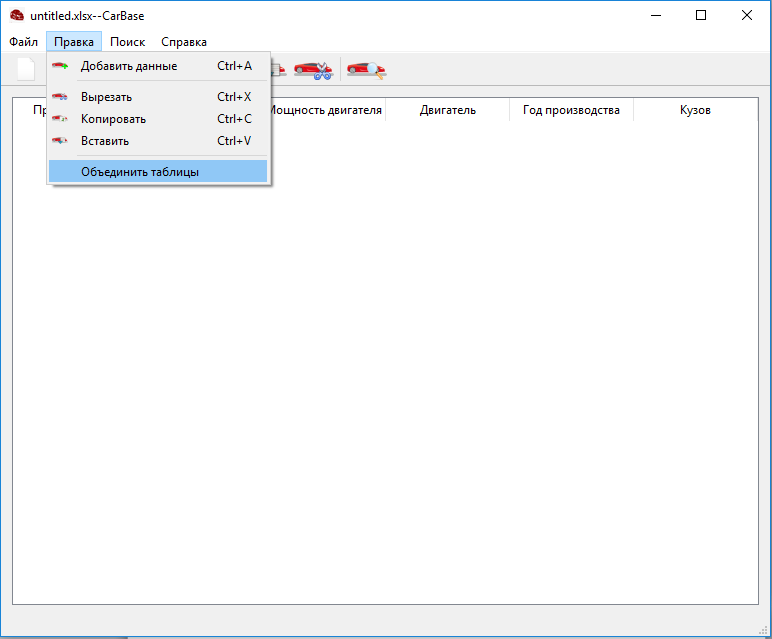
**11. Найти элемент**



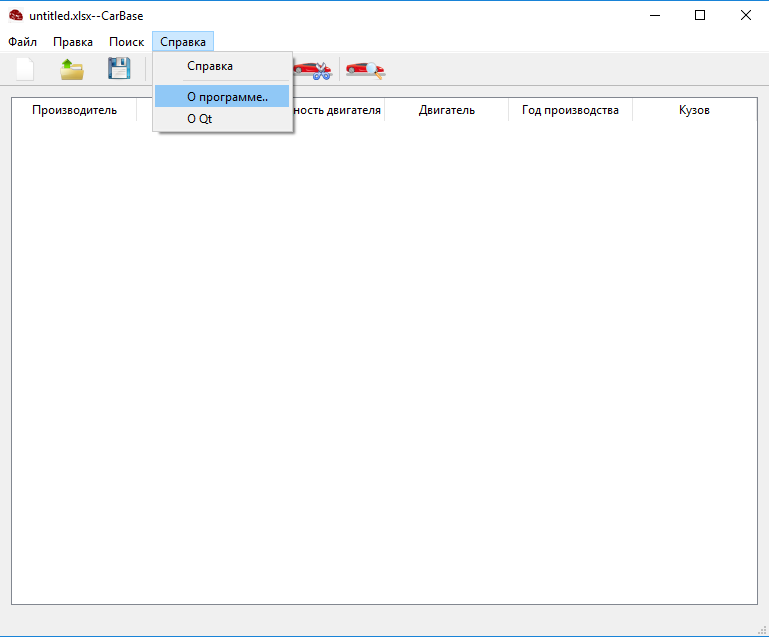


**Ok – добавить, cancel – отмена**

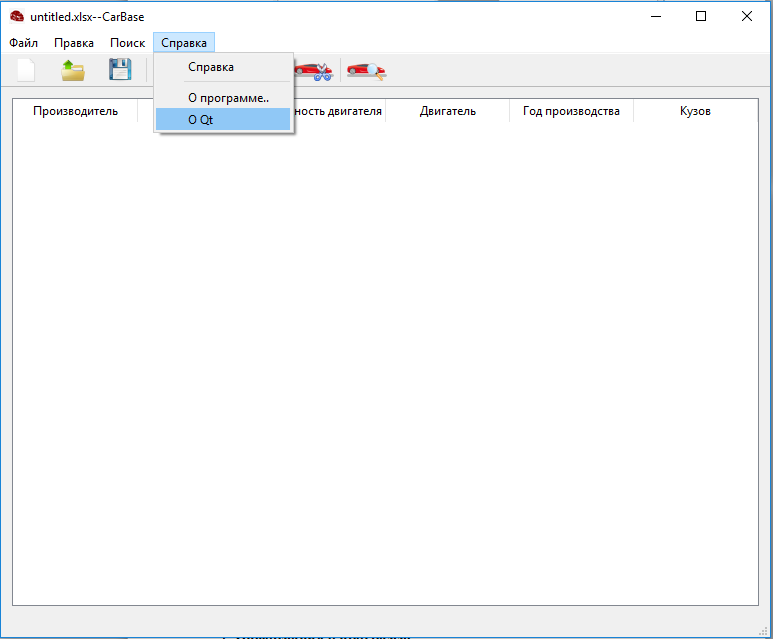
**12. Добавить данные из другой таблицы**



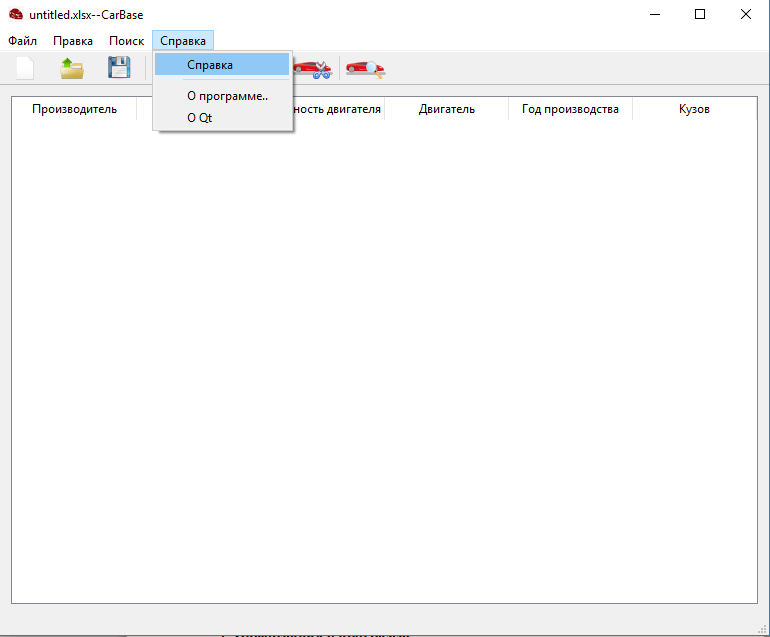
**13. Информация о программе**



**14. Информация о Qt**



**15. Инструкция по пользованию программой**



**Заключение**

Была разработана программа, обеспечивающая выполнение всех функций, объявленных в ТЗ. Также присутствует интерфейс, соответствующий требованиям.

Разработанные классы:

* List – контейнер, имитирующий базу данных
* Car – класс объектов машин
* Mainwindow – класс приложения в Qt

Во время выполнения работы возникало большое количество трудностей. Они разрешались с помощью обращения к официальной документации. Также параллельно изучалась стандартная библиотека шаблонов (STL).

Самая большая сложность заключалась в связки программы и Excel. Пришлось прочитать множество документацией и форумов, прежде чем программа стала работать исправно.

Данная работа дает большое количество опыта в разработке программного обеспечения. Также помогает лучше изучить структуры хранения данных.

**Список литературы**

* Жасмин Бланшет – “Qt4. Программирование GUI на C++”
* <http://www.wiki.crossplatform.ru/index.php/Работа_с_MS_Office_с_помощью_ActiveQt>
* http://doc.qt.io