МИНИСТЕРСТВО науки и высшего ОБРАЗОВАНИЯ  
РОссИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ

УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ

(национальный исследовательский университет)»

Институт №3 «Системы управления, информатика и электроэнергетика»

Кафедра № 304 «Вычислительные машины, системы и сети»

Пояснительная записка к курсовой работе

по дисциплине: «Программирование»

на тему: «База данных автомастерской»

Выполнил:

студент группы М3О-211Б-21 Пономарев Н.А.

Проверила:

доцент каф. 304, к.т.н. Дмитриева Е.А.

Москва 2023 г.

Содержание

[Задание 2](#_Toc153786624)

[Постановка задачи 3](#_Toc153786625)

[Краткое описание функций 4](#_Toc153786626)

[Входные данные 5](#_Toc153786627)

[Выходные данные 5](#_Toc153786628)

[Структурные схемы 6](#_Toc153786629)

[Модуль main 6](#_Toc153786630)

[Функция AddCustomer 8](#_Toc153786631)

[Функция SaveCustomer 8](#_Toc153786632)

[Функция SaveToFile 9](#_Toc153786633)

[Функция ReadAndAdd 10](#_Toc153786634)

[Функция Print 11](#_Toc153786635)

[Функция EditCustomer 12](#_Toc153786636)

[Функция DeleteCustomer 13](#_Toc153786637)

[Функция SortByField 15](#_Toc153786638)

[Функция SortCustomers 16](#_Toc153786639)

[Функция SelectionByCity 17](#_Toc153786640)

[Функция DeleteAll 18](#_Toc153786641)

[Описание интерфейса программы 19](#_Toc153786642)

[Код программы 22](#_Toc153786643)

[Результаты работы 35](#_Toc153786644)

## 

## **Задание**

Разработать систему управления базой данных на языке С++.

Тематики базы данных выбираются согласно варианту.

Для хранения данных использовать текстовый файл.

Разработать структуры данных для хранения информации и функции

* для ввода данных,
* выборки значения по заданному условию,
* вывода данных,
* удаления,
* записи новых данных,
* сортировки по определенному пользователем значению.

Обеспечить выдачу запросов по любым полям данных.

Всю обработку данных в главном модуле вести с помощью динамических структур данных (списков, одно- или двусвязных).

Реализовать систему меню.

Вариант 3/15

Схема организации данных: линейный двусвязный список

Тематика работы:

|  |  |
| --- | --- |
| «Автомастерская» | Владелец автомобиля: фамилия, имя, отчество; марка автомобиля, номер автомобиля, номер техпаспорта, телефон, водительское удостоверение, срок и тип ремонта |

# Постановка задачи

В ходе работы необходимо реализовать базу данных автомастерской и разработать систему для управления базой данных.

Требуется обеспечить:

* Выбор способа ввода исходных данных: с клавиатуры или из файла.
* Выбор способа вывода результата: на экран или в файл.
* Систему меню для управление программой.

Для реализации всех заявленных возможностей программы понадобится написать функции:

1. Для ввода базы данных с консоли или из файла по выбору пользователя.
2. Для вывода базы данных на экран или в файл по выбору пользователя.
3. Для редактирования элементов путём ввода с клавиатуры.
4. Для удаления отдельных элементов или всей базы данных.
5. Для сортировки базы данных по выбранному условию.
6. Для выборки элементов по параметру – тип ремонта

## **Краткое описание функций**

Функция для ввода базы данныхимеет два режима работы:

1. Ввод базы данных из файла.
2. Ввод базы данных с консоли.

Для ввода базы данных из файла реализовано чтение файла и последовательное добавление в список для дальнейших действий.

Для ввода базы данных с консоли понадобится заполнить данные о каждом клиенте автомастерской.

Эта функция используется также для добавления нового клиента в любое время.

Функция для вывода базы данныхимеет два режима работы:

* 1. Вывод полной базы в файл.
  2. Вывод полной базы на экран.

Для вывода базы данных в файл понадобится последовательно сохранить информацию о каждом покупателе в файл в виде таблицы.

Для вывода базы данных на экран понадобится вывести на экран каждого покупателя интернет-магазина и его данные в виде таблицы.

Функция, редактирующая элементы в базу данных будет реализована по принципу выбора по ФИО покупателя и редактирования его данных.

Функция удаления элементов из базыданных будет иметь два режима работы:

1. Удаление покупателя по порядковому номеру.
2. Удаление всей базы данных.

Функция для сортировки может сортировать базу данных по возрастанию по следующим параметрам:

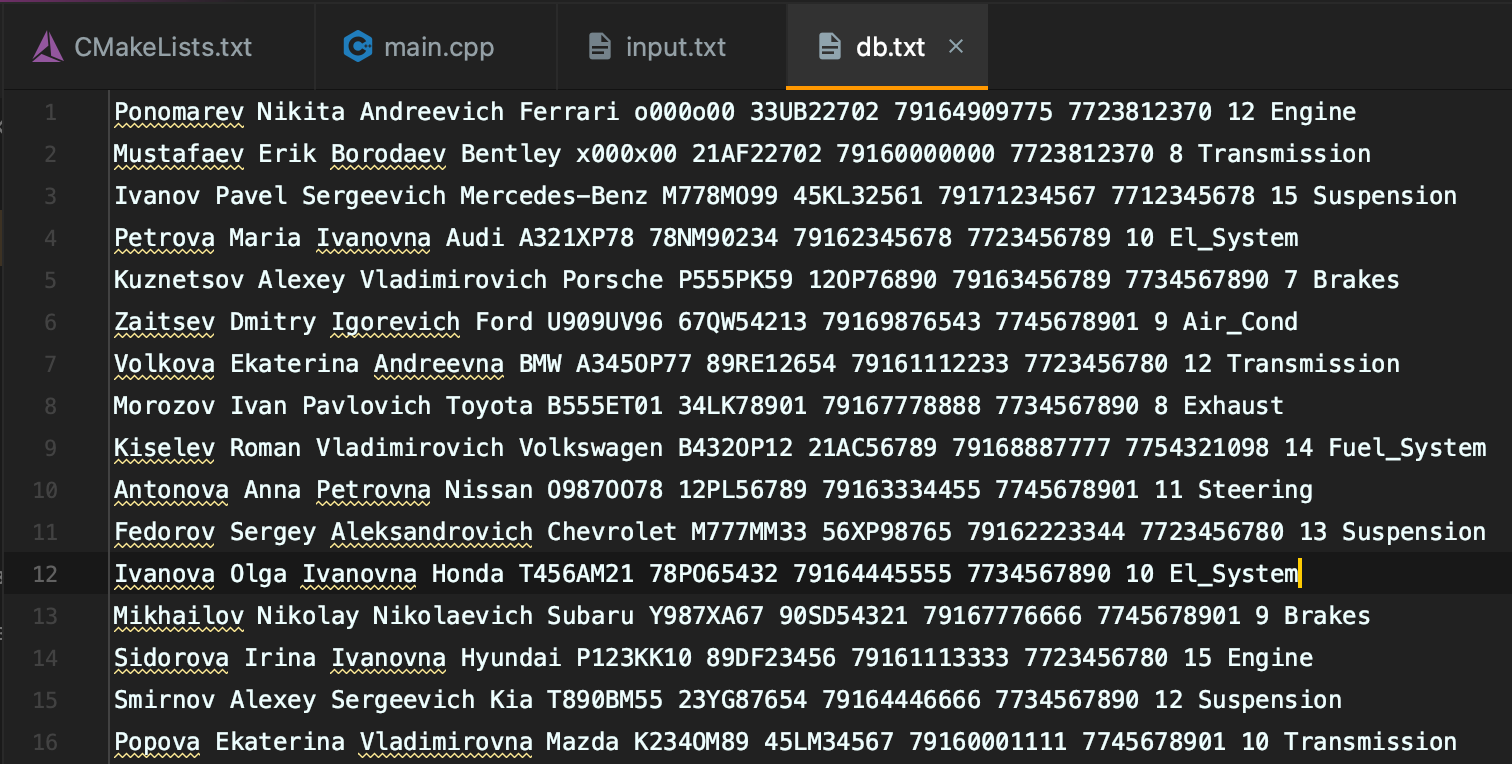
* Фамилия
* Имя
* Отчество
* Модель авто
* Номер авто
* Тех. паспорт
* Контактный телефон
* Права на вождение авто
* Длительность работ
* Тип ремонта

Функция выборки элементов из базыданных по параметру – тип ремонта. В списке будут выбираться покупатели, у которых параметр соответствует заданному, и сразу же выводится на экран.

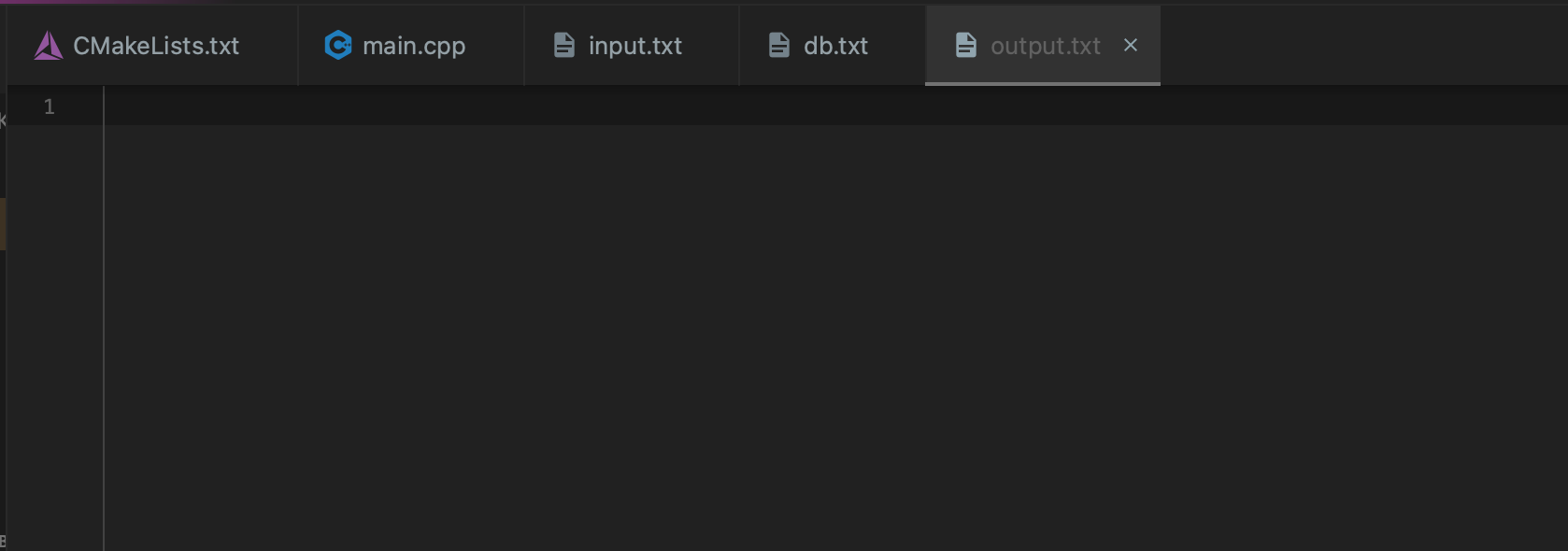
**Входные данные**

Пользователь сам решает, как ввести данные в программу: с помощью клавиатуры или из файла. Выбрать способ ввода он может с помощью меню.

Для примера работы заготовлен следующий файл с базой интернет-магазина



Выходной файл перед запуском программы пуст



**Выходные данные**

Выходные данные выводятся на экран или в файл.

В файл можно сохранить все данные из базы данных.

На экран попадает вся остальная информация: меню, результаты работы функций и сама база данных.

Через клавиатуру и экран пользователь взаимодействует с программой. Все выходные данные отображаются на экране. Например, после сортировки базы пользователь может вывести всю базу на экран для более детального изучения, отредактировать её или удалить информацию о каком-нибудь конкретном пользователе. Все взаимодействия с базой отражаются на экране.

**Структурные схемы**

Функция input



Рис. 1 - Структурная схема функции input

## Функция inputFromFile



Рис. 2 - Структурная схема функции inputFromFile

## Функция saveData



Рис. 3 - Структурная схема функции saveData

Функция deleteList



Рис. 4 - Структурная схема функции deleteList

## Функция editNodeByFIO

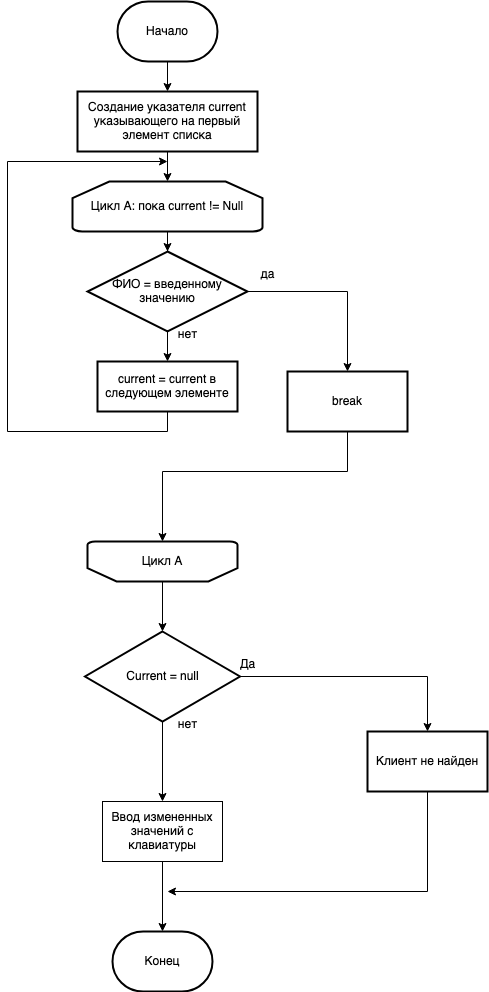


Рис. 5 - Структурная схема функции editNodeByFIO

## Функция deleteNodeByFIO

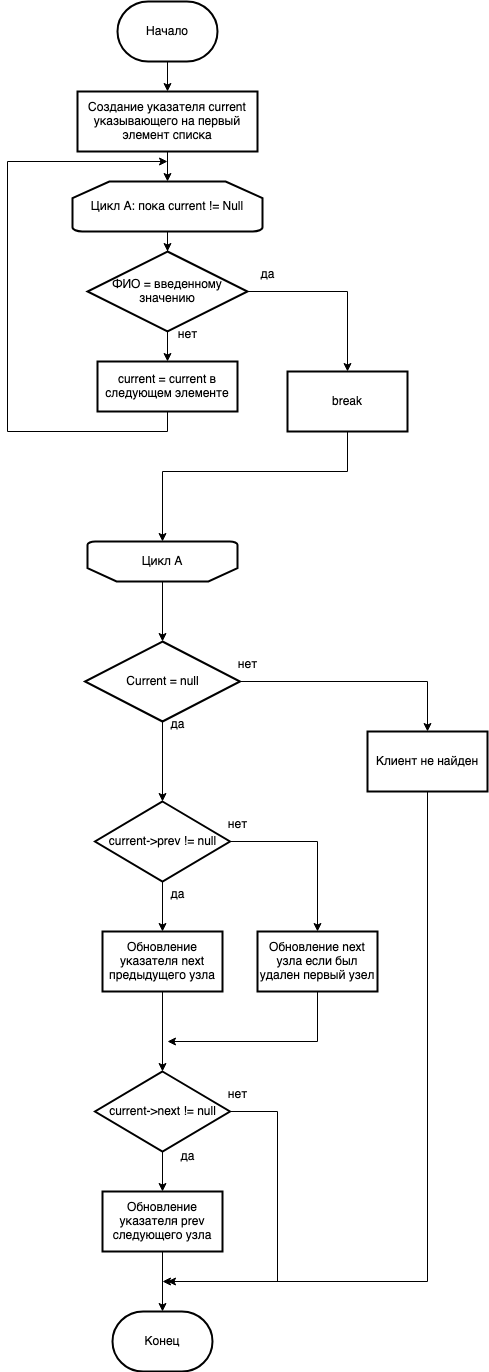


Рис. 6 - Структурная схема функции deleteNodeByFIO

## Функция outputAllListTable

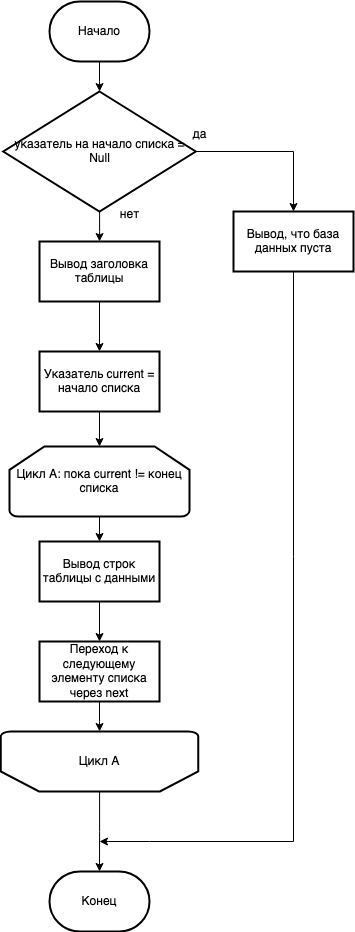


Рис. 7 - Структурная схема функции DeleteCustomer

## Функция outputTableToFile

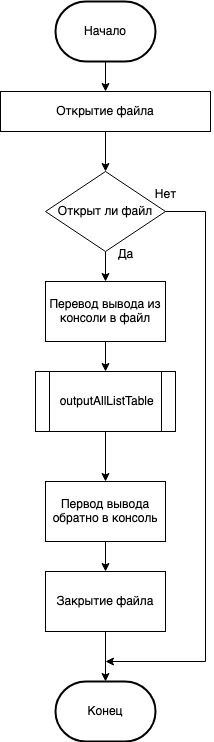


Рис. 8 - Структурная схема функции outputTableToFile

## Функция sortList

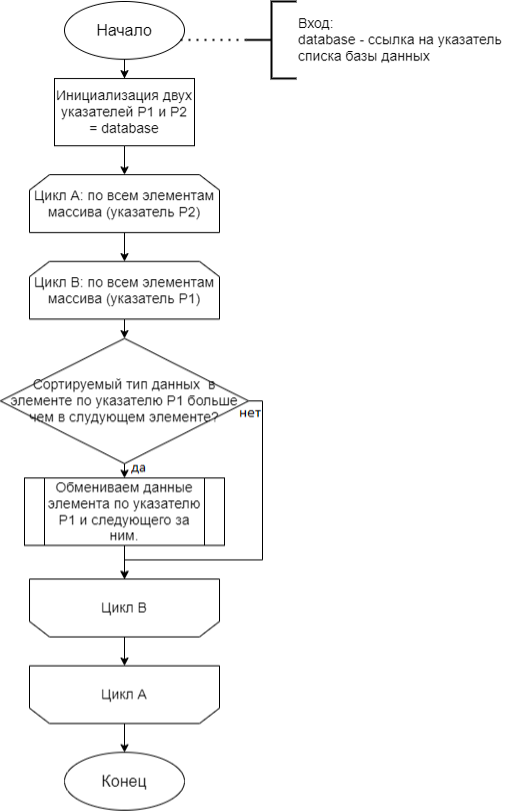


Рис. 9 - Структурная схема функции sortList

## Функция sampleNode

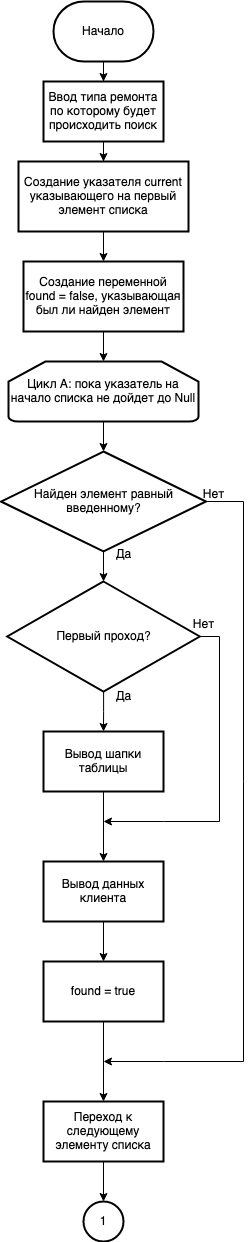




Рис. 10 - Структурная схема функции sampleNode

**Описание интерфейса программы**

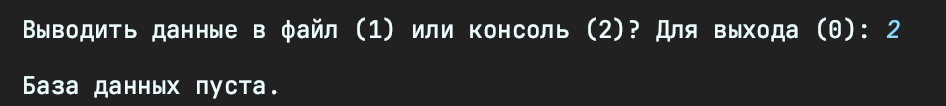
В программе реализована система меню.



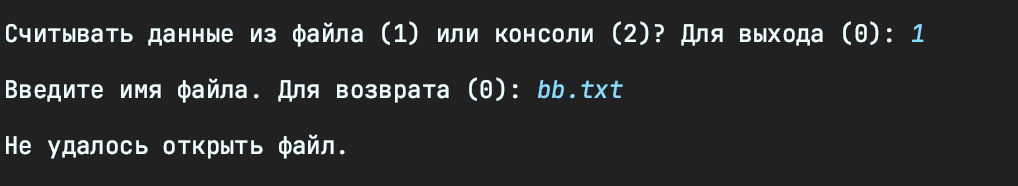
Программа может выполнить все указанные действия. Для этого пользователю требуется ввести номер действия.

Если в процессе работы программы возникают ошибки, будет выведены сообщения об ошибках на экране.

Например, при попытке вывести пустую базу данных на экран, будет выведено следующее сообщение

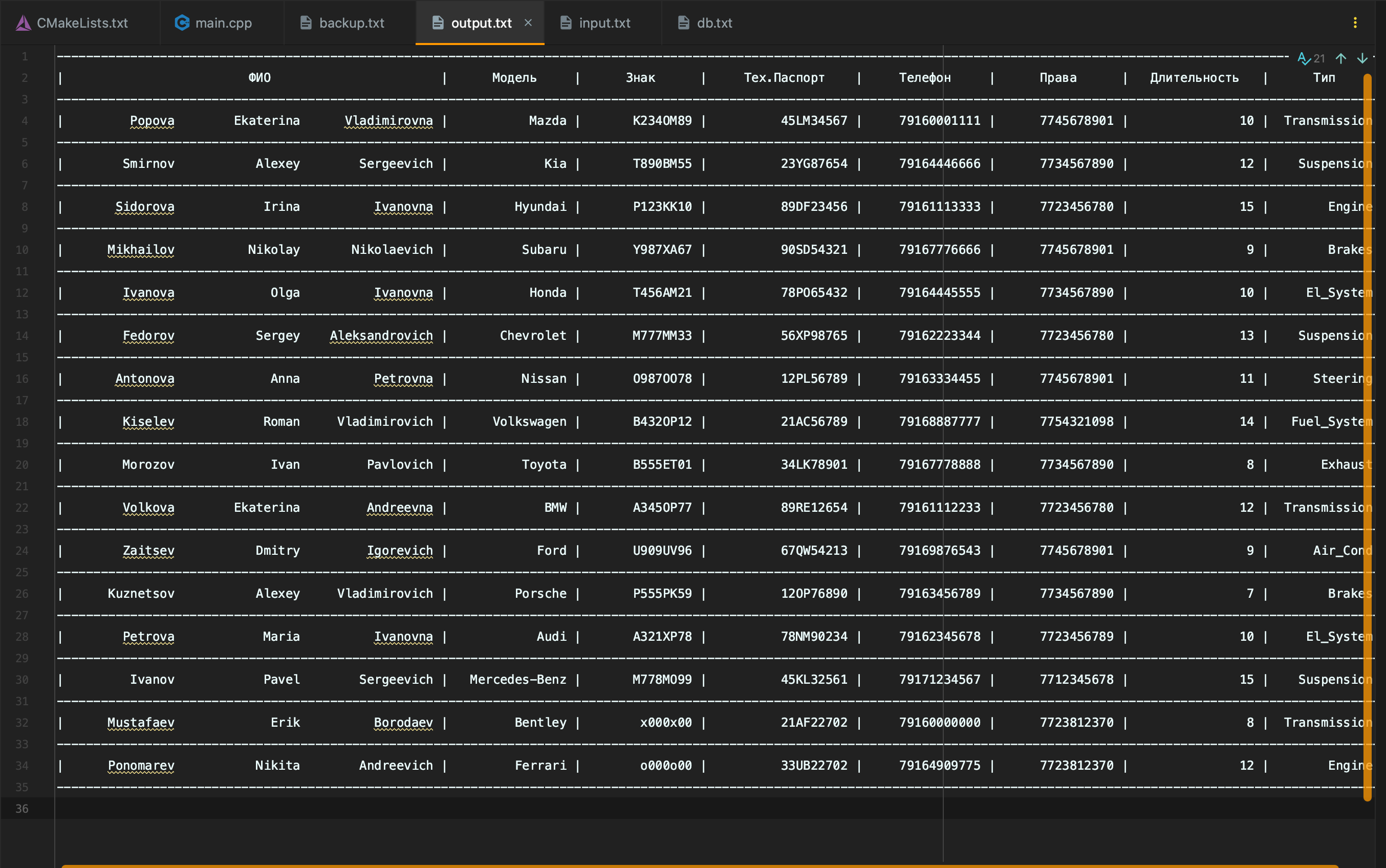


При попытки загрузить данные из некорректного файла, будет выведено следующее сообщение

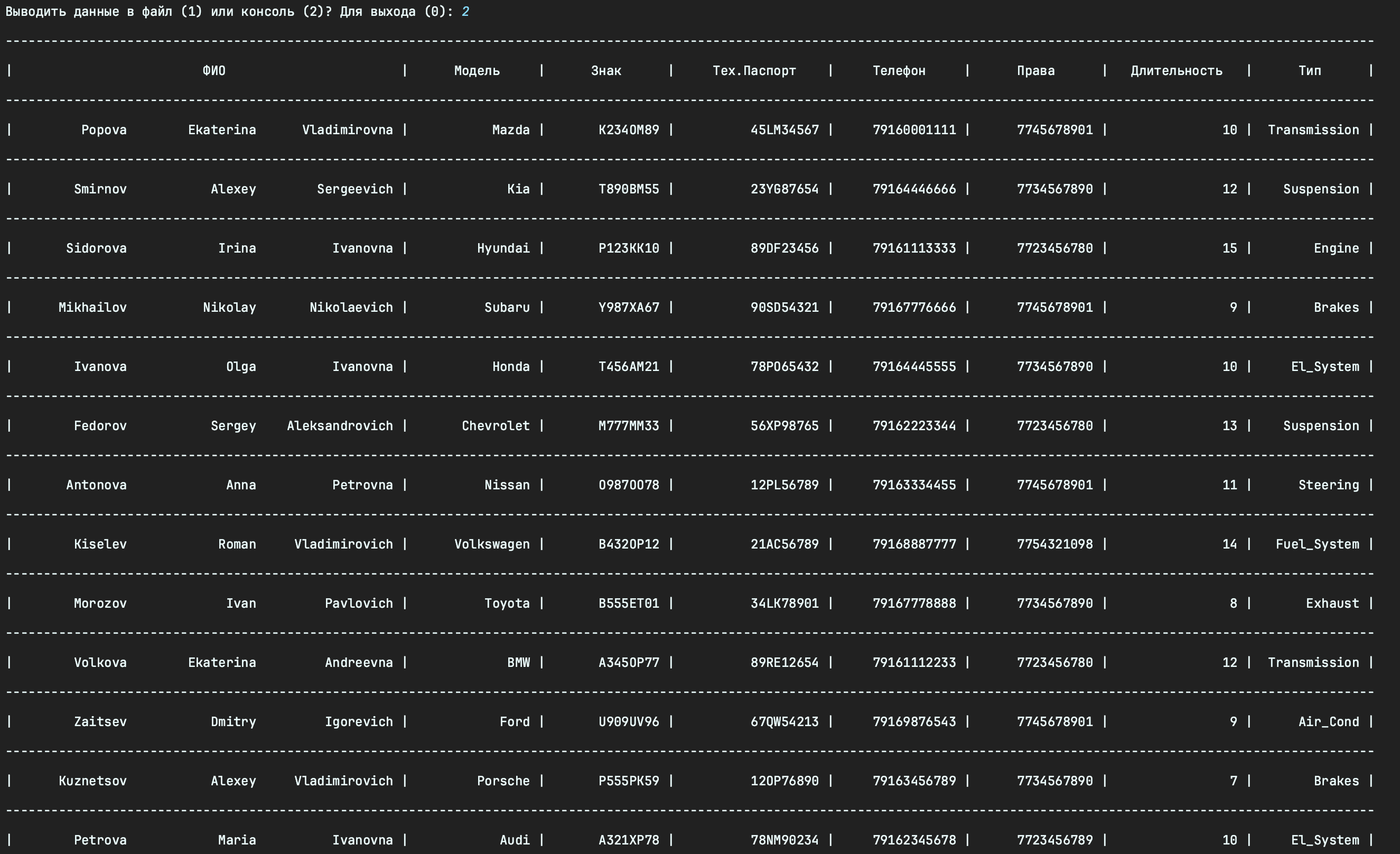


Вывод базы данных реализовывается в двух режимах:

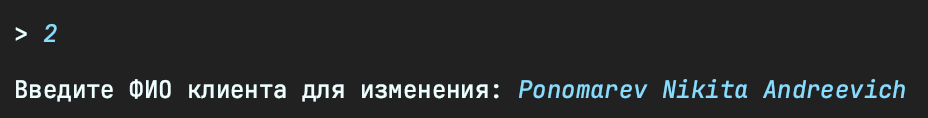
1. В файл



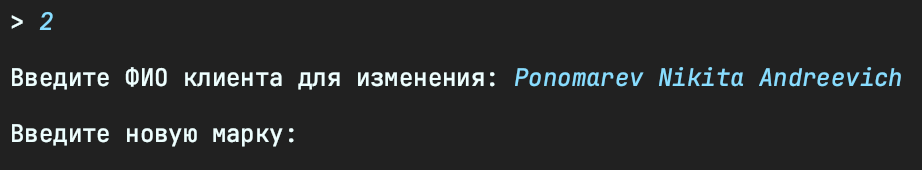
1. Вывод на экран



Для редактирования пользователю необходимо будет ввести ФИО клиента, чьи данные он хочет исправить.



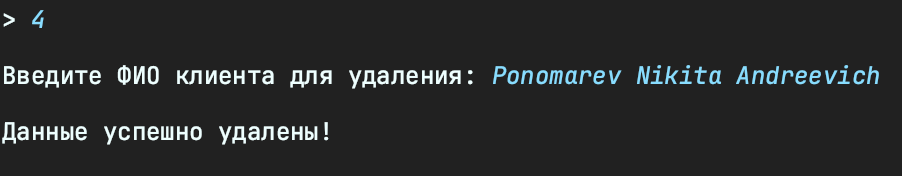
Далее пользователь должен по порядку ввести все данные клиента.



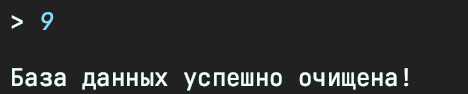
Функция для удаления базы данных есть в двух режимах:

1. Удаление отдельного элемента.

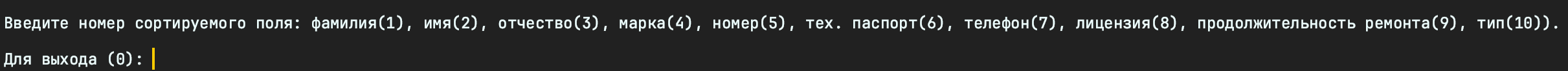
Пользователь выбирает ФИО клиента, которого хочет исключить из списка.



1. Удаление всей базы данных

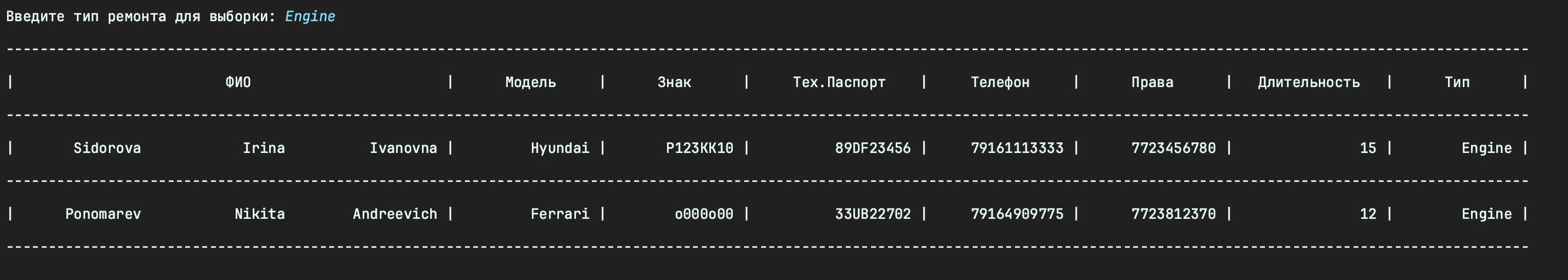


Функция сортировки реализована по следующим параметрам: фамилия, имя, отчество, марка авто, техпаспорт, телефон, права на вождение автомобилем, продолжительность ремонта, тип ремонта.

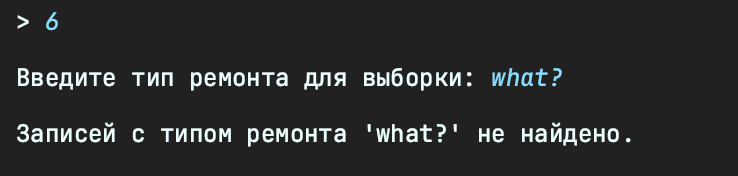


После выбора параметра список сортируется, и пользователь может вызвать действие 5, и будет выведен отсортированный список.

Функция выборки реализована по одному параметру – тип ремонта. После введения типа ремонта выводится список, который прошел выборку.



При вводе типа ремонта, которого нет в базе, будет выведена ошибка.



**Код программы**

#include <iostream>

#include <string>

#include <fstream>

#include <sstream>

#include <iomanip>

using namespace std;

// Структура для хранения ФИО

struct FIO {

string surname; // Фамилия человека

string name; // Имя человека

string middleName; // Отчество человека

};

// Структура для хранения информации о ремонте автомобиля

struct workshopList {

FIO fio; // Объект структуры FIO, представляющий имя человека

string brand; // Марка автомобиля

string plate; // Регистрационный номер автомобиля

string techPassport; // Технический паспорт

string phone; // Контактный телефон

string license; // Лицензия

int duration; // Длительность работы

string type; // Тип работы

workshopList\* next; // Указатель на следующий объект workshopList

workshopList\* prev; // Указатель на предыдущий объект workshopList

workshopList() {

next = nullptr;

prev = nullptr;

}

};

// Функция для ввода данных о ремонте с клавиатуры

void input(workshopList\*\* head);

// Функция для ввода данных о ремонте из файла

void inputFromFile(workshopList\*\* head, const string& filename);

// Функция для сохранения данных о ремонте в файл

void saveData(const workshopList\* head, const string& filename);

// Функция для удаления списка и освобождения памяти

void deleteList(struct workshopList\*\* head);

// Функция для редактирования данных о ремонте по ФИО клиента

void editNodeByFIO(struct workshopList\* head, const string& surname, const string& name, const string& middleName);

// Функция для удаления данных о ремонте по ФИО клиента

void deleteNodeByFIO(struct workshopList\*\* head, const string& surname, const string& name, const string& middleName);

// Функция для вывода всех данных о ремонтах в виде таблицы

void outputAllListTable(struct workshopList\* head);

// Функция для вывода таблицы данных о ремонтах в файл

void outputTableToFile(const workshopList\* head, const string& filename);

// Функция для обмена данными между двумя узлами списка

void swapNodes(workshopList\* node1, workshopList\* node2);

// Функция для сортировки списка по выбранному полю

int sortList(workshopList\*& database);

// Функция для выполнения выборки данных о ремонтах по типу работы

void sampleNode(workshopList\*& database);

// Функция для ввода числового значения в консольном интерфейсе

int ConsoleInterfaceInputVar();

// Функция для ввода числового значения в консольном интерфейсе (вариант из двух)

int ConsoleInterfaceInputVarOfTwo();

// Функция для ввода данных о ремонте через консольный интерфейс

void consoleInterfaceInput(workshopList\*& database);

// Функция для вывода данных о ремонтах через консольный интерфейс

void consoleInterfaceOutput(workshopList\*& database);

// Функция для сохранения данных о ремонтах через консольный интерфейс

void saveDataInterface(workshopList\*& database);

// Основная функция консольного интерфейса программы

void consoleInterface();

void appendNode(workshopList\*& list, const FIO& fio, const string& brand,

const string& plate, const string& techPassport,

const string& phone, const string& license,

int duration, const string& type);

workshopList\* createListByBrand(const workshopList\* database, const string& brand);

// Функция для ввода данных о ремонте с клавиатуры

void input(workshopList\*\* head) {

workshopList\* newNode = new workshopList;

// Ввод данных с клавиатуры

cout << "Введите фамилию: ";

cin >> newNode->fio.surname;

cout << "Введите имя: ";

cin >> newNode->fio.name;

cout << "Введите отчество: ";

cin >> newNode->fio.middleName;

cout << "Введите марку автомобиля: ";

cin >> newNode->brand;

cout << "Введите номер автомобиля: ";

cin >> newNode->plate;

cout << "Введите тех. паспорт автомобиля: ";

cin >> newNode->techPassport;

cout << "Введите номер телефона: ";

cin >> newNode->phone;

cout << "Введите номер прав: ";

cin >> newNode->license;

cout << "Введите продолжительность работ: ";

cin >> newNode->duration;

cout << "Введите тип ремонта(eng): ";

cin >> newNode->type;

// Установка связей для нового узла в списке

newNode->next = (\*head);

newNode->prev = nullptr;

if ((\*head) != nullptr) {

(\*head)->prev = newNode;

}

(\*head) = newNode;

}

// Функция для ввода данных о ремонте из файла

void inputFromFile(workshopList\*\* head, const string& filename) {

ifstream inputFile(filename);

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

cout << "Не удалось открыть файл." << endl;

return;

}

string surname, name, middleName, brand, plate, techPassport, phone, license, type;

int duration;

// Чтение данных из файла

while (inputFile >> surname >> name >> middleName >> brand >> plate >> techPassport >> phone >> license >> duration >> type) {

workshopList\* newNode = new workshopList;

// Заполнение нового узла данными из файла

newNode->fio.surname = surname;

newNode->fio.name = name;

newNode->fio.middleName = middleName;

newNode->brand = brand;

newNode->plate = plate;

newNode->techPassport = techPassport;

newNode->phone = phone;

newNode->license = license;

newNode->duration = duration;

newNode->type = type;

// Установка связей для нового узла в списке

newNode->next = (\*head);

newNode->prev = nullptr;

if ((\*head) != nullptr) {

(\*head)->prev = newNode;

}

(\*head) = newNode;

}

inputFile.close();

}

// Функция для сохранения данных о ремонте в файл

void saveData(const workshopList\* head, const string& filename) {

ofstream outputFile(filename);

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

cout << "Не удалось открыть файл." << endl;

return;

}

const workshopList\* current = head;

// Запись данных в файл

while (current != nullptr) {

outputFile << current->fio.surname << " ";

outputFile << current->fio.name << " ";

outputFile << current->fio.middleName << " ";

outputFile << current->brand << " ";

outputFile << current->plate << " ";

outputFile << current->techPassport << " ";

outputFile << current->phone << " ";

outputFile << current->license << " ";

outputFile << current->duration << " ";

outputFile << current->type << endl;

current = current->next;

}

outputFile.close();

cout << "Данные успешно сохранены в файл: " << filename << endl;

}

// Функция для удаления списка и освобождения выделенной памяти

void deleteList(struct workshopList\*\* head) {

struct workshopList\* temp;

// Освобождение памяти для каждого узла в списке

while (\*head != NULL) {

temp = \*head;

\*head = (\*head)->next;

free(temp);

}

}

// Функция для удаления узла по ФИО из списка

void deleteNodeByFIO(workshopList\*\* head, const string& surname, const string& name, const string& middleName) {

workshopList\* current = \*head;

// Поиск узла по ФИО

while (current != nullptr) {

if (current->fio.surname == surname && current->fio.name == name && current->fio.middleName == middleName) {

break;

}

current = current->next;

}

// Если узел не найден

if (current == nullptr) {

cout << "Клиент с ФИО(" << surname << ", " << name << ", " << middleName << ") не найден." << endl;

return;

}

// Обновление указателей соседних узлов

if (current->prev != nullptr) {

current->prev->next = current->next;

} else {

\*head = current->next; // Если удаляется первый узел

}

if (current->next != nullptr) {

current->next->prev = current->prev;

}

cout << "Данные успешно удалены!" << endl;

// Освобождение памяти для удаляемого узла

delete current;

}

// Функция для редактирования узла по ФИО в списке

void editNodeByFIO(workshopList\* head, const string& surname, const string& name, const string& middleName) {

workshopList\* current = head;

// Поиск узла по ФИО

while (current != nullptr) {

if (current->fio.surname == surname && current->fio.name == name && current->fio.middleName == middleName) {

break;

}

current = current->next;

}

// Если узел не найден

if (current == nullptr) {

cout << "Клиент с ФИО(" << surname << ", " << name << ", " << middleName << ") не найден." << endl;

return;

}

// Редактирование данных узла

cout << "Введите новую марку: ";

cin >> current->brand;

cout << "Введите новый номер: ";

cin >> current->plate;

cout << "Введите новый тех.паспорт: ";

cin >> current->techPassport;

cout << "Введите новый номер телефона: ";

cin >> current->phone;

cout << "Введите новый номер прав: ";

cin >> current->license;

cout << "Введите новую продолжительность ремонта: ";

cin >> current->duration;

cout << "Введите новый тип ремонта(eng): ";

cin >> current->type;

}

// Функция для вывода списка в табличном формате на экран

void outputAllListTable(workshopList\* head) {

if (head == nullptr) {

cout << "База данных пуста." << endl;

return;

}

// Вывод заголовка таблицы

cout << "------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------" << endl;

cout << "| ФИО | Модель | Знак | Тех.Паспорт | Телефон | Права | Длительность | Тип |" << endl;

cout << "------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------" << endl;

workshopList\* current = head;

// Вывод данных каждого узла в таблице

while (current != nullptr) {

cout << "| ";

cout << setw(14) << current->fio.surname << " ";

cout << setw(14) << current->fio.name << " ";

cout << setw(15) << current->fio.middleName << " | ";

cout << setw(15) << current->brand << " | ";

cout << setw(14) << current->plate << " | ";

cout << setw(18) << current->techPassport << " | ";

cout << setw(15) << current->phone << " | ";

cout << setw(15) << current->license << " | ";

cout << setw(16) << current->duration << " | ";

cout << setw(13) << current->type;

cout << " |" << endl;

current = current->next;

cout << "------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------" << endl;

}

}

// Функция для вывода таблицы в файл

void outputTableToFile(const workshopList\* head, const string& filename) {

ofstream outputFile(filename);

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

cout << "Не удалось открыть файл." << endl;

return;

}

// Create a stringstream to capture the output of outputAllListTable

stringstream ss;

streambuf\* oldCoutBuffer = cout.rdbuf(ss.rdbuf());

// Call outputAllListTable to generate the tabular output

outputAllListTable(const\_cast<workshopList\*>(head));

// Restore the cout buffer

cout.rdbuf(oldCoutBuffer);

// Write the captured output to the file

outputFile << ss.str();

outputFile.close();

cout << "Данные успешно сохранены в файл: " << filename << endl;

}

// Функция для обмена данными между двумя узлами

void swapNodes(workshopList\* node1, workshopList\* node2) {

if (node1 == node2) {

// Узлы идентичны, обмен не требуется

return;

}

// Обмен данными узлов

swap(node1->fio, node2->fio);

swap(node1->brand, node2->brand);

swap(node1->plate, node2->plate);

swap(node1->techPassport, node2->techPassport);

swap(node1->phone, node2->phone);

swap(node1->license, node2->license);

swap(node1->duration, node2->duration);

swap(node1->type, node2->type);

}

// Функция для сортировки списка по заданному полю

int sortList(workshopList\*& database) {

workshopList\* node1 = database; // Указатель для сравнений в пузырьковой сортировке

workshopList\* node2 = database; // Дополнительный указатель для работы цикла сортировки

backk:

int answ; // Ответ пользователя

cout << "Введите номер сортируемого поля: фамилия(1), имя(2), отчество(3), марка(4), номер(5), тех. паспорт(6), телефон(7), лицензия(8), продолжительность ремонта(9), тип(10)).\nДля выхода (0): ";

// Проверка правильности ввода ответа

if (!(cin >> answ) or answ < 0 or answ > 10) {

cout << "Введена неверная команда!" << endl;

goto backk;

}

// Сортировка списка в зависимости от выбранного поля

switch (answ) {

case 0:

return 1;

break;

case 1: // Сортировка по фамилии

for (node2 = database; node2; node2 = node2->next) {

for (node1 = database; node1->next; node1 = node1->next) {

if (node1->fio.surname > node1->next->fio.surname) {

swapNodes(node1, node1->next);

}

}

}

break;

case 2: // Сортировка по имени

for (node2 = database; node2; node2 = node2->next) {

for (node1 = database; node1->next; node1 = node1->next) {

if (node1->fio.name > node1->next->fio.name) {

swapNodes(node1, node1->next);

}

}

}

break;

case 3: // Сортировка по отчеству

for (node2 = database; node2; node2 = node2->next) {

for (node1 = database; node1->next; node1 = node1->next) {

if (node1->fio.middleName > node1->next->fio.middleName) {

swapNodes(node1, node1->next);

}

}

}

break;

case 4: // Сортировка по марке

for (node2 = database; node2; node2 = node2->next) {

for (node1 = database; node1->next; node1 = node1->next) {

if (node1->brand > node1->next->brand) {

swapNodes(node1, node1->next);

}

}

}

break;

case 5: // Сортировка по номеру

for (node2 = database; node2; node2 = node2->next) {

for (node1 = database; node1->next; node1 = node1->next) {

if (node1->plate > node1->next->plate) {

swapNodes(node1, node1->next);

}

}

}

break;

case 6: // Сортировка по тех. паспорту

for (node2 = database; node2; node2 = node2->next) {

for (node1 = database; node1->next; node1 = node1->next) {

if (node1->techPassport > node1->next->techPassport) {

swapNodes(node1, node1->next);

}

}

}

break;

case 7: // Сортировка по телефону

for (node2 = database; node2; node2 = node2->next) {

for (node1 = database; node1->next; node1 = node1->next) {

if (node1->phone > node1->next->phone) {

swapNodes(node1, node1->next);

}

}

}

break;

case 8://сортировка по цене

for (node2 = database; node2; node2 = node2->next)

{

for (node1 = database; node1->next; node1 = node1->next)

{

if (node1->license > node1->next->license)

{

swapNodes(node1, node1->next);

}

}

}

break;

case 9://сортировка по цене

for (node2 = database; node2; node2 = node2->next)

{

for (node1 = database; node1->next; node1 = node1->next)

{

if (node1->duration> node1->next->duration)

{

swapNodes(node1, node1->next);

}

}

}

break;

case 10://сортировка по цене

for (node2 = database; node2; node2 = node2->next)

{

for (node1 = database; node1->next; node1 = node1->next)

{

if (node1->type > node1->next->type)

{

swapNodes(node1, node1->next);

}

}

}

break;

}

cout << "Сортировка произведена успешно!" << endl;

return 1;

}

void sampleNode(workshopList\*& database) {

string type;

cout << "Введите тип ремонта для выборки: ";

cin >> type;

workshopList\* current = database;

bool found = false;

int once = 1;

// Проход по списку данных и вывод записей с заданным типом ремонта

while (current != nullptr) {

if (current->type == type) {

if (once == 1) {

// Вывод заголовка таблицы при первом совпадении

cout << "------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------" << endl;

cout << "| ФИО | Модель | Знак | Тех.Паспорт | Телефон | Права | Длительность | Тип |" << endl;

cout << "------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------" << endl;

}

// Вывод данных о клиенте

cout << "| ";

cout << setw(14) << current->fio.surname << " ";

cout << setw(14) << current->fio.name << " ";

cout << setw(15) << current->fio.middleName << " | ";

cout << setw(15) << current->brand << " | ";

cout << setw(14) << current->plate << " | ";

cout << setw(18) << current->techPassport << " | ";

cout << setw(15) << current->phone << " | ";

cout << setw(15) << current->license << " | ";

cout << setw(16) << current->duration << " | ";

cout << setw(13) << current->type;

cout << " |" << endl;

// Вывод разделительной линии после каждой записи

cout << "------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------" << endl;

found = true;

once = 0;

}

current = current->next;

}

// Если не найдено записей с заданным типом ремонта

if (!found) {

cout << "Записей с типом ремонта '" << type << "' не найдено." << endl;

}

}

int ConsoleInterfaceInputVar() {

int variant;

cin >> variant;

// Проверка на корректность ввода варианта

while (variant != 1 && variant != 2 && variant != 3 && variant != 4 && variant != 5 && variant != 6 && variant != 7 && variant != 8 && variant != 9 && variant != 10 && variant != 11 && variant != 0) {

cout << "Неправильный ввод. Попробуйте еще раз: ";

cin >> variant;

}

return variant;

}

int ConsoleInterfaceInputVarOfTwo() {

int variant;

cin >> variant;

// Проверка на корректность ввода варианта (должен быть 0, 1 или 2)

while (variant != 1 && variant != 2 && variant != 0) {

cout << "Неправильный ввод. Попробуйте еще раз: ";

cin >> variant;

}

return variant;

}

void consoleInterfaceInput(workshopList \*&database) {

chooseAnsw:

int answer;

string FNAME;

// Предложение выбора источника данных (файл или консоль)

cout << "Считывать данные из файла (1) или консоли (2)? Для выхода (0): ";

answer = ConsoleInterfaceInputVarOfTwo();

switch (answer) {

case 1:

cout << "Введите имя файла. Для возврата (0): ";

cin >> FNAME;

// Возврат к выбору, если пользователь ввел 0

if (FNAME == "0") goto chooseAnsw;

inputFromFile(&database, FNAME);

cout << "Данные успешно считаны из файла." << endl;

break;

case 2:

input(&database);

break;

case 0:

break;

}

}

void consoleInterfaceOutput(workshopList \*&database){

chooseAnsw:

int answer;

string FNAME;

// Предложение выбора места вывода данных (файл или консоль)

cout << "Выводить данные в файл (1) или консоль (2)? Для выхода (0): ";

answer = ConsoleInterfaceInputVarOfTwo();

switch (answer) {

case 1:

cout << "Введите имя файла. Для возврата (0): ";

cin >> FNAME;

// Возврат к выбору, если пользователь ввел 0

if (FNAME == "0") goto chooseAnsw;

outputTableToFile(database, FNAME);

break;

case 2:

outputAllListTable(database);

break;

case 0:

break;

}

}

void saveDataInterface(workshopList \*&database){

start:

string FNAME;

// Предложение ввести имя файла для сохранения данных

cout << "Введите имя файла в котором хотите сохранить данные. Для возврата (0): ";

cin >> FNAME;

// Возврат к выбору, если пользователь ввел 0

if (FNAME == "0") goto start;

saveData(database, FNAME);

}

void appendNode(workshopList\*& list, const FIO& fio, const string& brand,

const string& plate, const string& techPassport,

const string& phone, const string& license,

int duration, const string& type) {

// Создание нового узла

workshopList\* newNode = new workshopList;

newNode->fio = fio;

newNode->brand = brand;

newNode->plate = plate;

newNode->techPassport = techPassport;

newNode->phone = phone;

newNode->license = license;

newNode->duration = duration;

newNode->type = type;

newNode->next = nullptr;

// Добавление узла в конец списка

if (list == nullptr) {

list = newNode;

} else {

workshopList\* current = list;

while (current->next != nullptr) {

current = current->next;

}

current->next = newNode;

}

}

workshopList\* createListByBrand(const workshopList\* database, const string& brand) {

workshopList\* newList = nullptr; // Инициализация нового списка

// Проход по существующему списку

const workshopList\* current = database;

while (current != nullptr) {

// Проверка, соответствует ли текущий узел выбранной марке

if (current->brand == brand) {

// Добавление узла в новый список

appendNode(newList, current->fio, current->brand, current->plate,

current->techPassport, current->phone, current->license,

current->duration, current->type);

}

current = current->next;

}

return newList; // Возвращение нового списка

}

void operationsUnderCarsList(workshopList\* cars){

int variant; // Переменная для хранения выбора пользователя

string sur, nam, mid; // Переменные для хранения ввода ФИО клиента

string typeCar;

do {

// Отображение меню выбора действий

cout << "\nSecond database!!!!!!";

cout << "\nЧто вы хотите сделать?\n"

<< "1. Ввести данные\n"

<< "2. Изменить данные\n"

<< "3. Добавить данные\n"

<< "4. Удалить данные\n"

<< "5. Вывести данные\n"

<< "6. Сделать выборку по типу ремонта\n"

<< "7. Отсортировать по типу\n"

<< "8. Сохранить данные\n"

<< "9. Очистить базу данных\n"

<< "0. Выйти из программы\n"

<< "> ";

variant = ConsoleInterfaceInputVar(); // Получение выбора пользователя

switch(variant){

case 1:

consoleInterfaceInput(cars); // Вызов функции ввода данных

break;

case 2:

cout << "Введите ФИО клиента для изменения: ";

cin >> sur >> nam >> mid;

editNodeByFIO(cars, sur, nam, mid); // Вызов функции изменения данных

break;

case 3:

cout << "Введите данные для добавления:\n";

input(&cars); // Вызов функции добавления данных

cout << "Данные успешно добавлены!" << endl;

break;

case 4:

cout << "Введите ФИО клиента для удаления: ";

cin >> sur >> nam >> mid;

deleteNodeByFIO(&cars, sur, nam, mid); // Вызов функции удаления данных

break;

case 5:

consoleInterfaceOutput(cars); // Вызов функции вывода данных

break;

case 6:

sampleNode(cars); // Вызов функции выборки по типу ремонта

break;

case 7:

sortList(cars); // Вызов функции сортировки данных

break;

case 8:

saveDataInterface(cars); // Вызов функции сохранения данных

break;

case 9:

deleteList(&cars); // Освобождение памяти, выделенной для списка

cout << "База данных успешно очищена!" << endl; // Сообщение об успешной очистке

break;

case 10:

cout << "Wrong input!" << endl;

break;

case 11:

cout << "Wrong input!" << endl;

break;

}

system("clear"); // Очистка экрана для улучшения визуального восприятия

} while (variant != 0);

}

void consoleInterface(){

workshopList \*database = nullptr; // Инициализация указателя на структуру данных

workshopList\* cars = nullptr;

int variant; // Переменная для хранения выбора пользователя

string sur, nam, mid; // Переменные для хранения ввода ФИО клиента

string typeCar;

// Отображение заголовка программы

cout << "8888888b. 888 888 \n"

"888 \"Y88b 888 888 \n"

"888 888 888 888 \n"

"888 888 8888b. 888888 8888b. 88888b. 8888b. .d8888b .d88b. \n"

"888 888 \"88b 888 \"88b 888 \"88b \"88b 88K d8P Y8b \n"

"888 888 .d888888 888 .d888888 888 888 .d888888 \"Y8888b. 88888888 \n"

"888 .d88P 888 888 Y88b. 888 888 888 d88P 888 888 X88 Y8b. \n"

"8888888P\" \"Y888888 \"Y888 \"Y888888 88888P\" \"Y888888 88888P' \"Y8888 \n";

do {

// Отображение меню выбора действий

cout << "\nЧто вы хотите сделать?\n"

<< "1. Ввести данные\n"

<< "2. Изменить данные\n"

<< "3. Добавить данные\n"

<< "4. Удалить данные\n"

<< "5. Вывести данные\n"

<< "6. Сделать выборку по типу ремонта\n"

<< "7. Отсортировать по типу\n"

<< "8. Сохранить данные\n"

<< "9. Очистить базу данных\n"

<< "10. Создать новый список по выборке автомобилей\n"

<< "11. Операции над новым списком с автомобилями\n"

<< "0. Выйти из программы\n"

<< "> ";

variant = ConsoleInterfaceInputVar(); // Получение выбора пользователя

switch(variant){

case 1:

consoleInterfaceInput(database); // Вызов функции ввода данных

break;

case 2:

cout << "Введите ФИО клиента для изменения: ";

cin >> sur >> nam >> mid;

editNodeByFIO(database, sur, nam, mid); // Вызов функции изменения данных

break;

case 3:

cout << "Введите данные для добавления:\n";

input(&database); // Вызов функции добавления данных

cout << "Данные успешно добавлены!" << endl;

break;

case 4:

cout << "Введите ФИО клиента для удаления: ";

cin >> sur >> nam >> mid;

deleteNodeByFIO(&database, sur, nam, mid); // Вызов функции удаления данных

break;

case 5:

consoleInterfaceOutput(database); // Вызов функции вывода данных

break;

case 6:

sampleNode(database); // Вызов функции выборки по типу ремонта

break;

case 7:

sortList(database); // Вызов функции сортировки данных

break;

case 8:

saveDataInterface(database); // Вызов функции сохранения данных

break;

case 9:

deleteList(&database); // Освобождение памяти, выделенной для списка

cout << "База данных успешно очищена!" << endl; // Сообщение об успешной очистке

break;

case 10:

cout << "Введите модель машины для выборки: ";

cin >> typeCar;

cars = createListByBrand(database, typeCar);

break;

case 11:

operationsUnderCarsList(cars);

break;

}

system("clear"); // Очистка экрана для улучшения визуального восприятия

} while (variant != 0);

cout << "До свидания!" << endl;

deleteList(&database); // Освобождение памяти, выделенной для списка

deleteList(&cars);

}

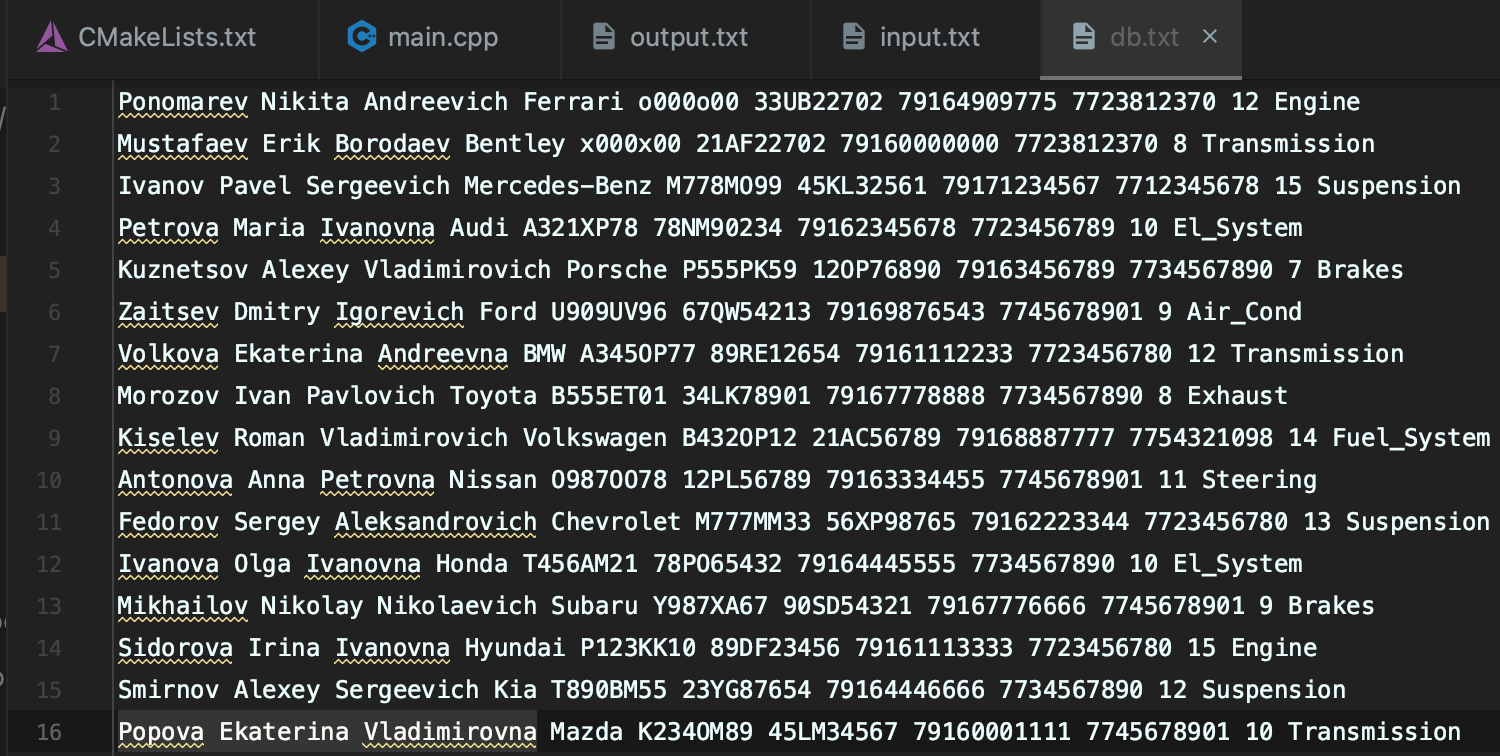
int main() {

consoleInterface();

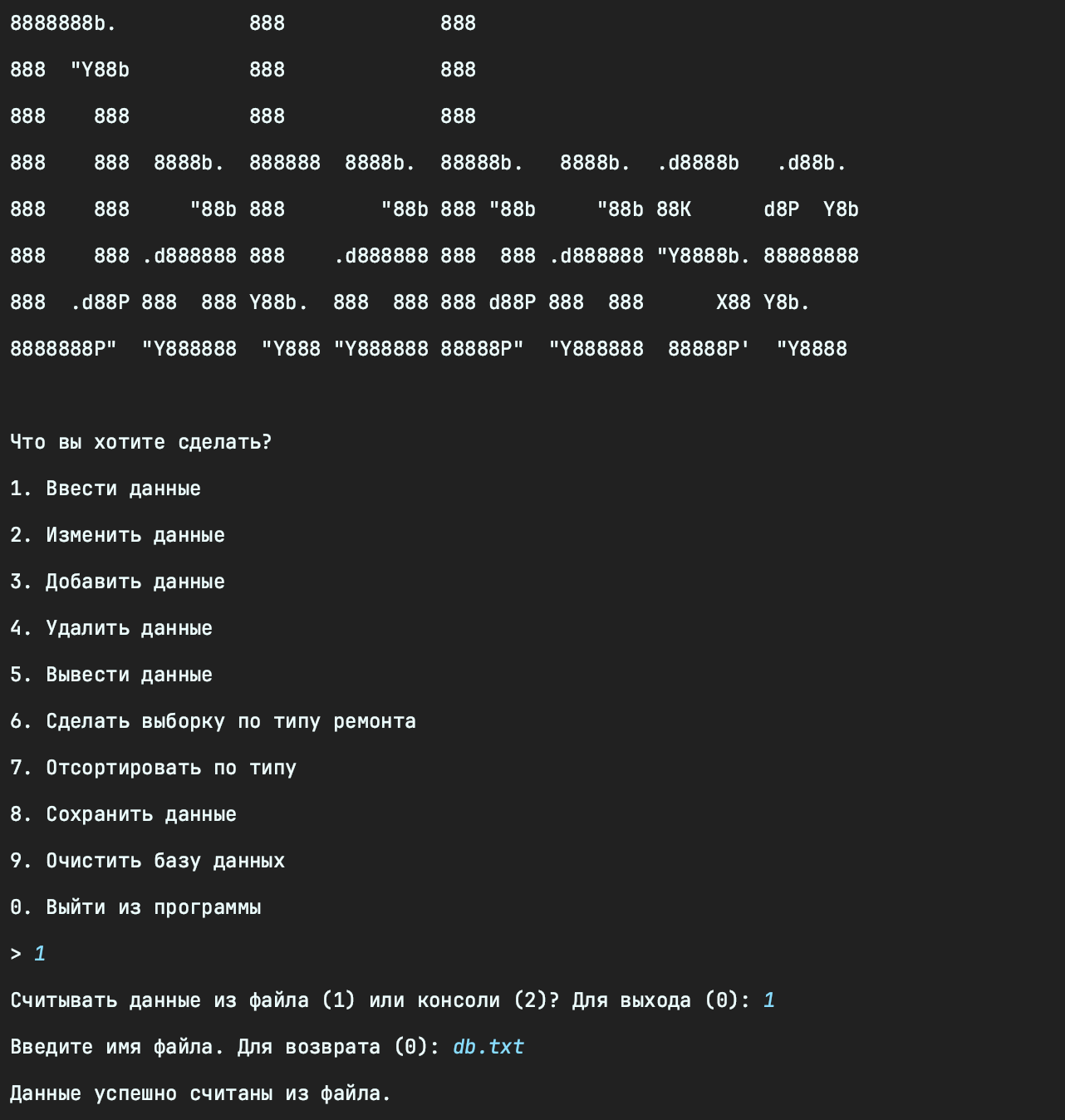
return 0

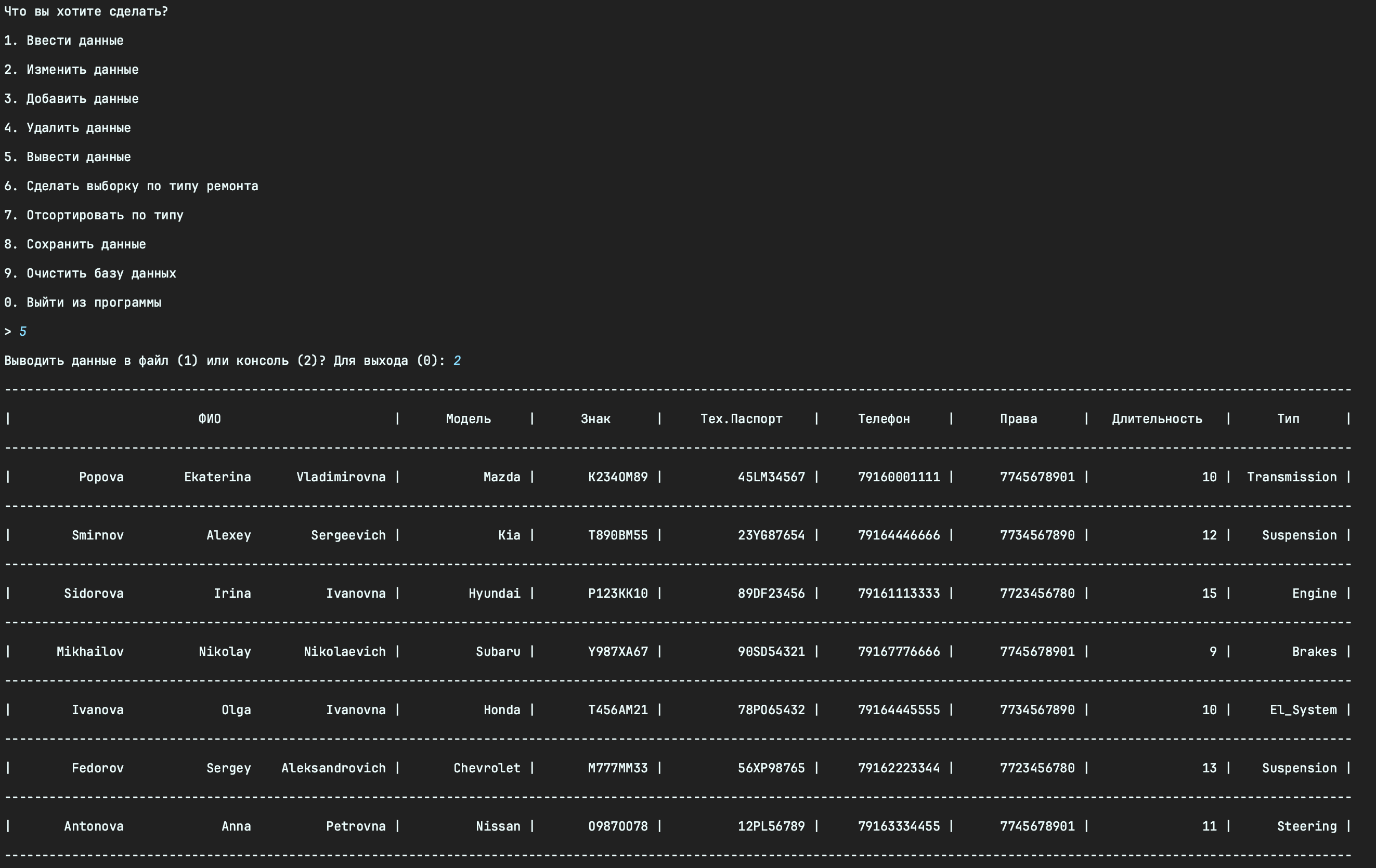
**Результаты работы**

Входной файл:

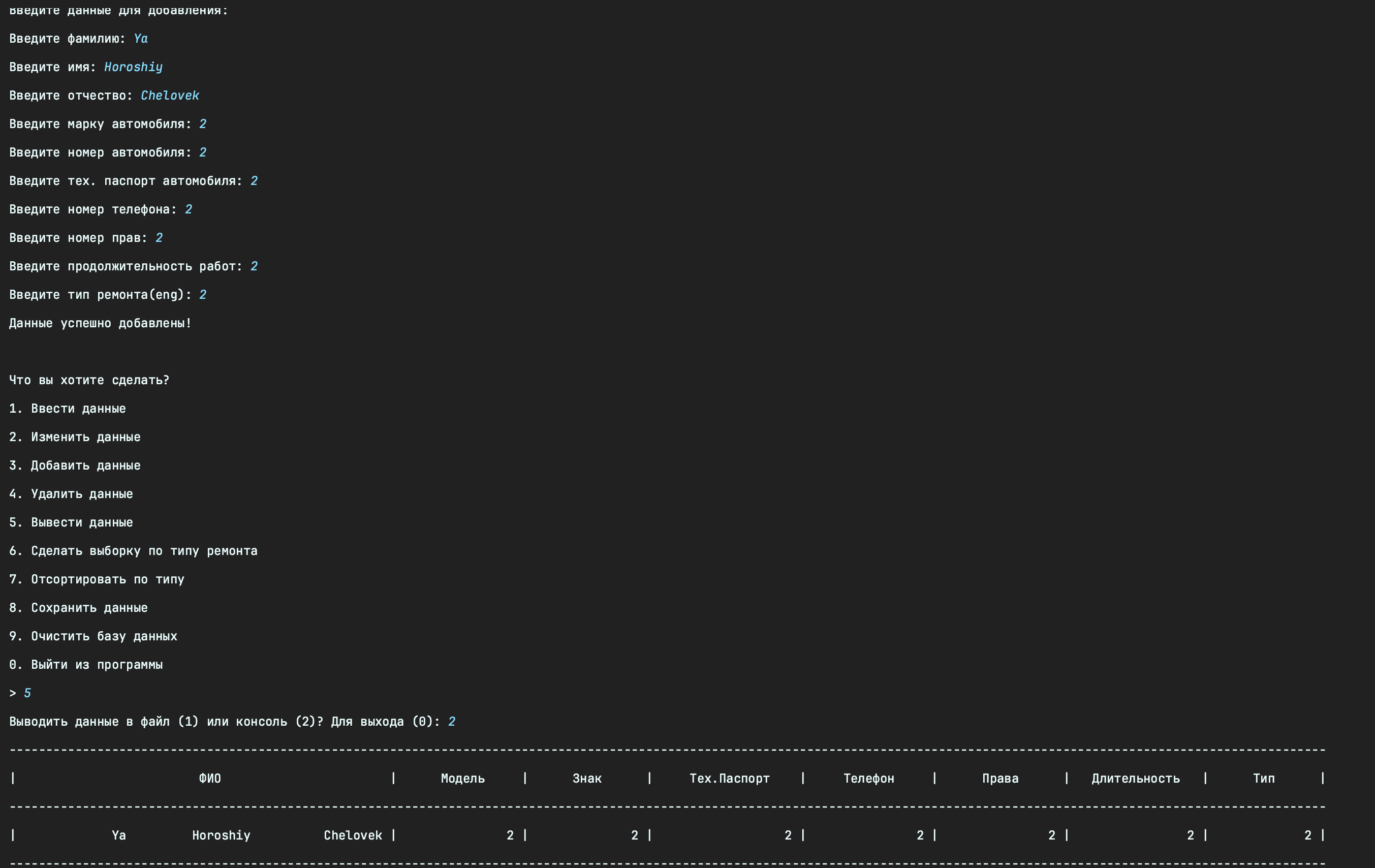


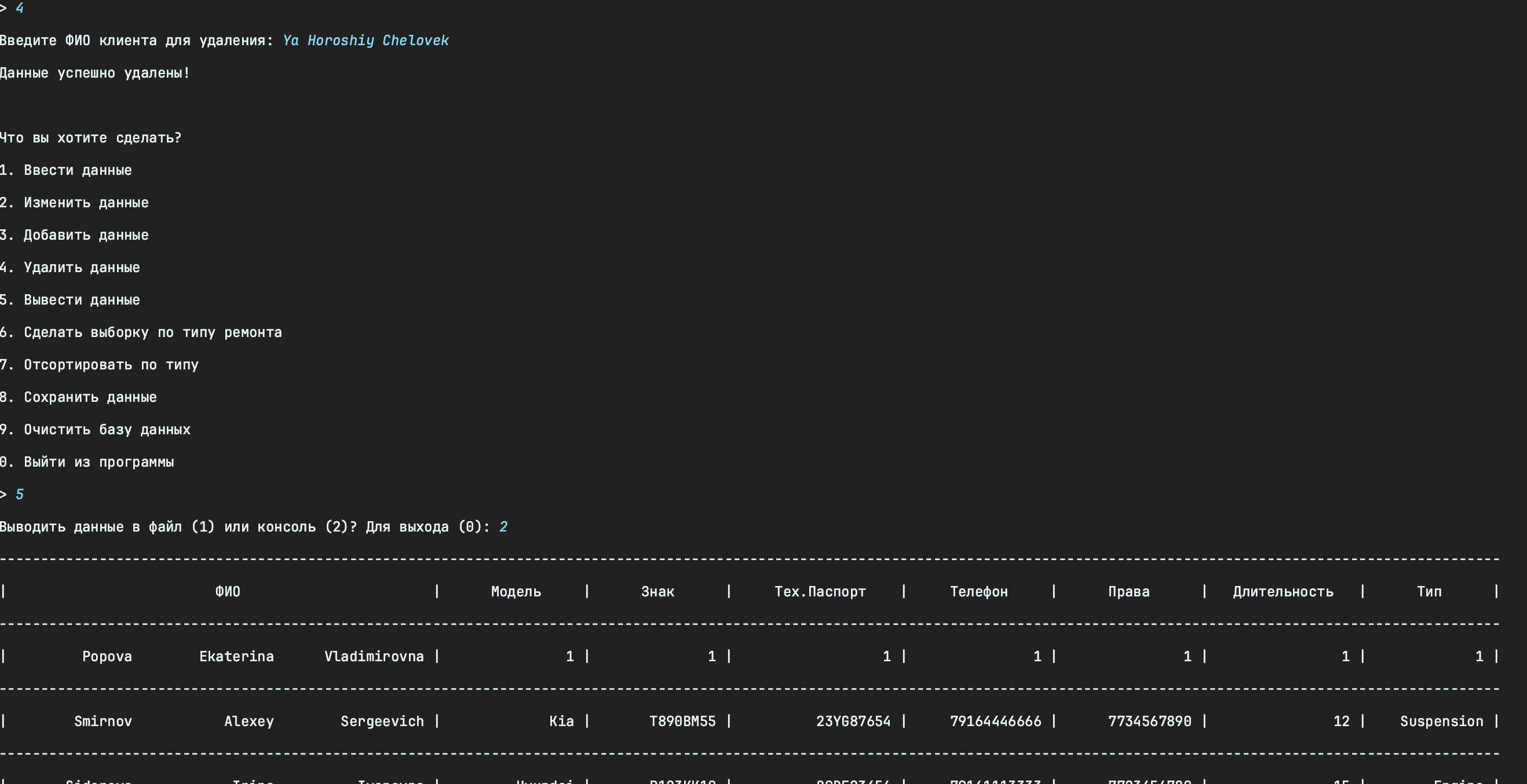
Результаты работы программы:

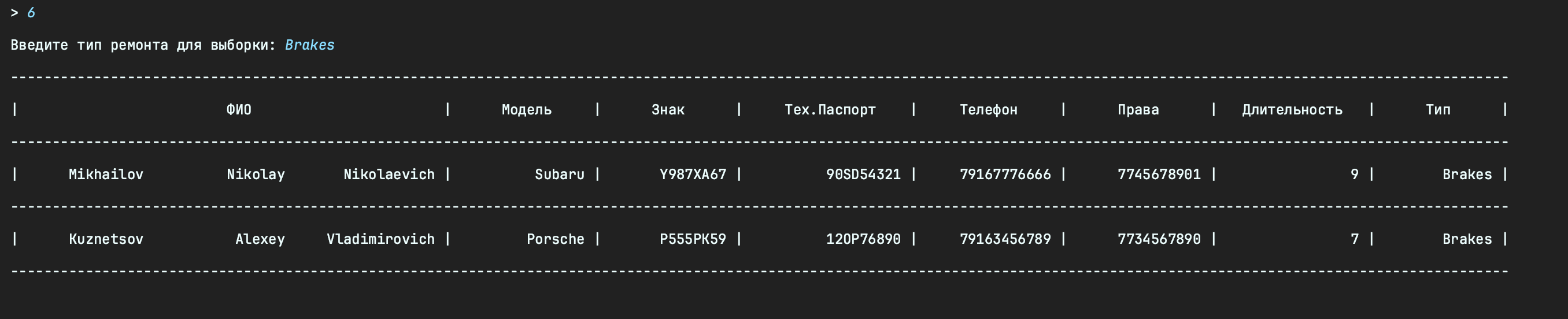


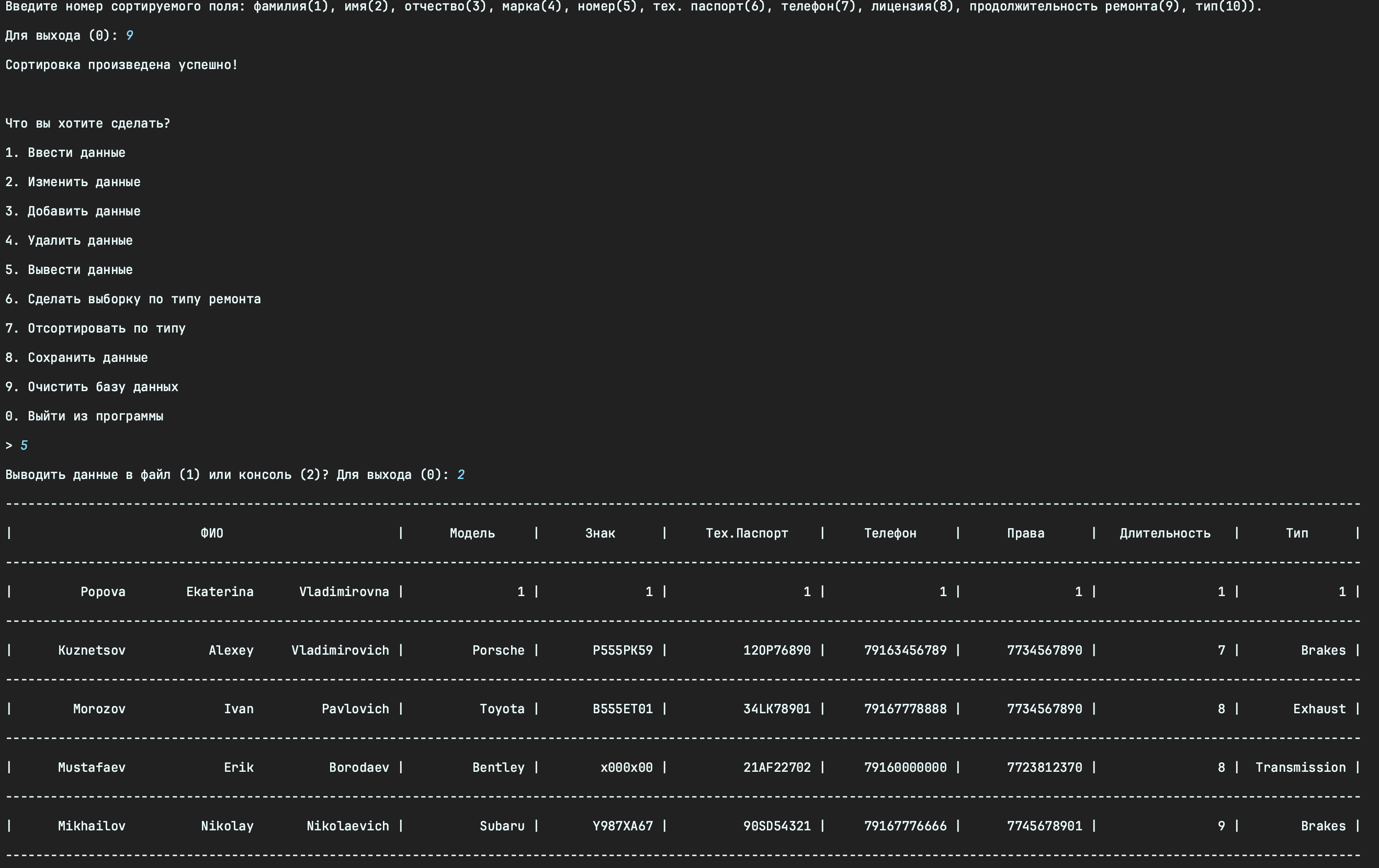




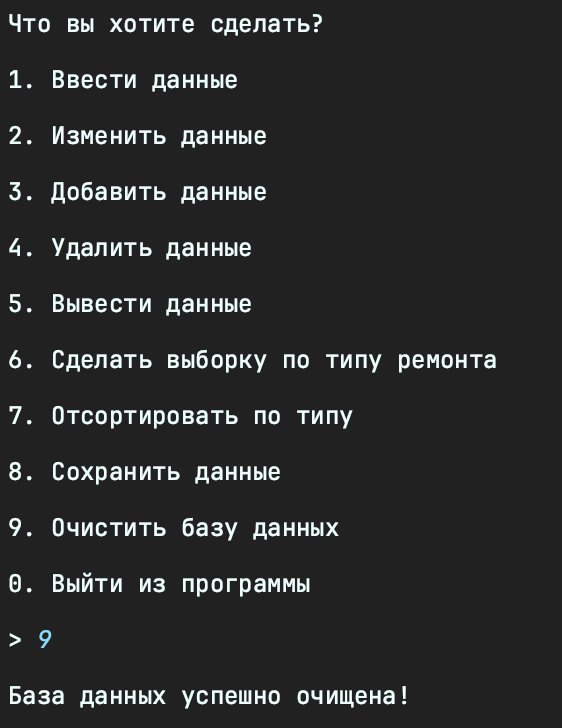


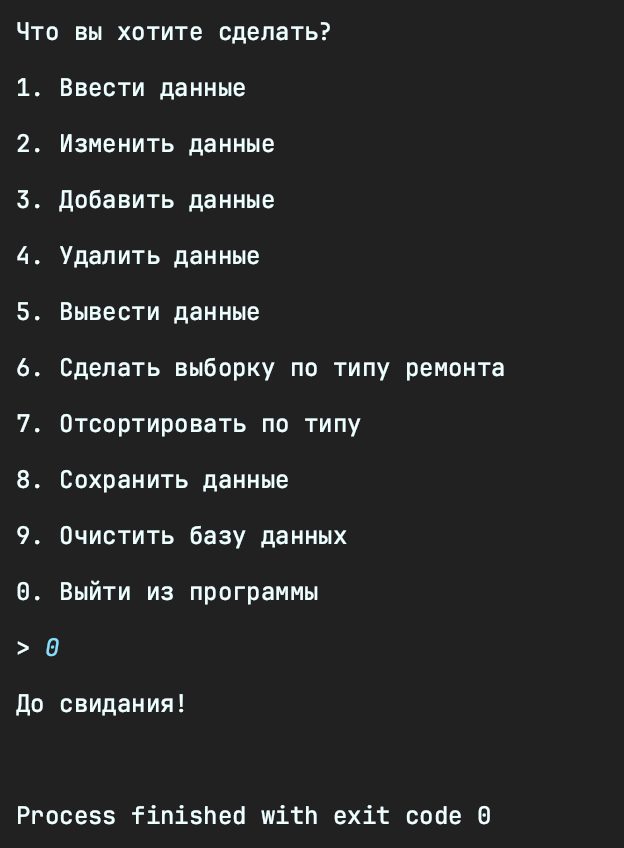












# Дополнительное задание

Задание: реализовать новую функцию программы: создание новой базы данных по выборке автомобилей, над этой базой данных должны быть возможны все те же операции, как и над основной.

## **Код функций, с помощью которых была реализована функция программы:**

void appendNode(workshopList\*& list, const FIO& fio, const string& brand,

const string& plate, const string& techPassport,

const string& phone, const string& license,

int duration, const string& type) {

// Создание нового узла

workshopList\* newNode = new workshopList;

newNode->fio = fio;

newNode->brand = brand;

newNode->plate = plate;

newNode->techPassport = techPassport;

newNode->phone = phone;

newNode->license = license;

newNode->duration = duration;

newNode->type = type;

newNode->next = nullptr;

// Добавление узла в конец списка

if (list == nullptr) {

list = newNode;

} else {

workshopList\* current = list;

while (current->next != nullptr) {

current = current->next;

}

current->next = newNode;

}

}

workshopList\* createListByBrand(const workshopList\* database, const string& brand) {

workshopList\* newList = nullptr; // Инициализация нового списка

// Проход по существующему списку

const workshopList\* current = database;

while (current != nullptr) {

// Проверка, соответствует ли текущий узел выбранной марке

if (current->brand == brand) {

// Добавление узла в новый список

appendNode(newList, current->fio, current->brand, current->plate,

current->techPassport, current->phone, current->license,

current->duration, current->type);

}

current = current->next;

}

return newList; // Возвращение нового списка

}

void operationsUnderCarsList(workshopList\* cars){

int variant; // Переменная для хранения выбора пользователя

string sur, nam, mid; // Переменные для хранения ввода ФИО клиента

string typeCar;

do {

// Отображение меню выбора действий

cout << "\nSecond database!!!!!!";

cout << "\nЧто вы хотите сделать?\n"

<< "1. Ввести данные\n"

<< "2. Изменить данные\n"

<< "3. Добавить данные\n"

<< "4. Удалить данные\n"

<< "5. Вывести данные\n"

<< "6. Сделать выборку по типу ремонта\n"

<< "7. Отсортировать по типу\n"

<< "8. Сохранить данные\n"

<< "9. Очистить базу данных\n"

<< "0. Выйти из программы\n"

<< "> ";

variant = ConsoleInterfaceInputVar(); // Получение выбора пользователя

switch(variant){

case 1:

consoleInterfaceInput(cars); // Вызов функции ввода данных

break;

case 2:

cout << "Введите ФИО клиента для изменения: ";

cin >> sur >> nam >> mid;

editNodeByFIO(cars, sur, nam, mid); // Вызов функции изменения данных

break;

case 3:

cout << "Введите данные для добавления:\n";

input(&cars); // Вызов функции добавления данных

cout << "Данные успешно добавлены!" << endl;

break;

case 4:

cout << "Введите ФИО клиента для удаления: ";

cin >> sur >> nam >> mid;

deleteNodeByFIO(&cars, sur, nam, mid); // Вызов функции удаления данных

break;

case 5:

consoleInterfaceOutput(cars); // Вызов функции вывода данных

break;

case 6:

sampleNode(cars); // Вызов функции выборки по типу ремонта

break;

case 7:

sortList(cars); // Вызов функции сортировки данных

break;

case 8:

saveDataInterface(cars); // Вызов функции сохранения данных

break;

case 9:

deleteList(&cars); // Освобождение памяти, выделенной для списка

cout << "База данных успешно очищена!" << endl; // Сообщение об успешной очистке

break;

case 10:

cout << "Wrong input!" << endl;

break;

case 11:

cout << "Wrong input!" << endl;

break;

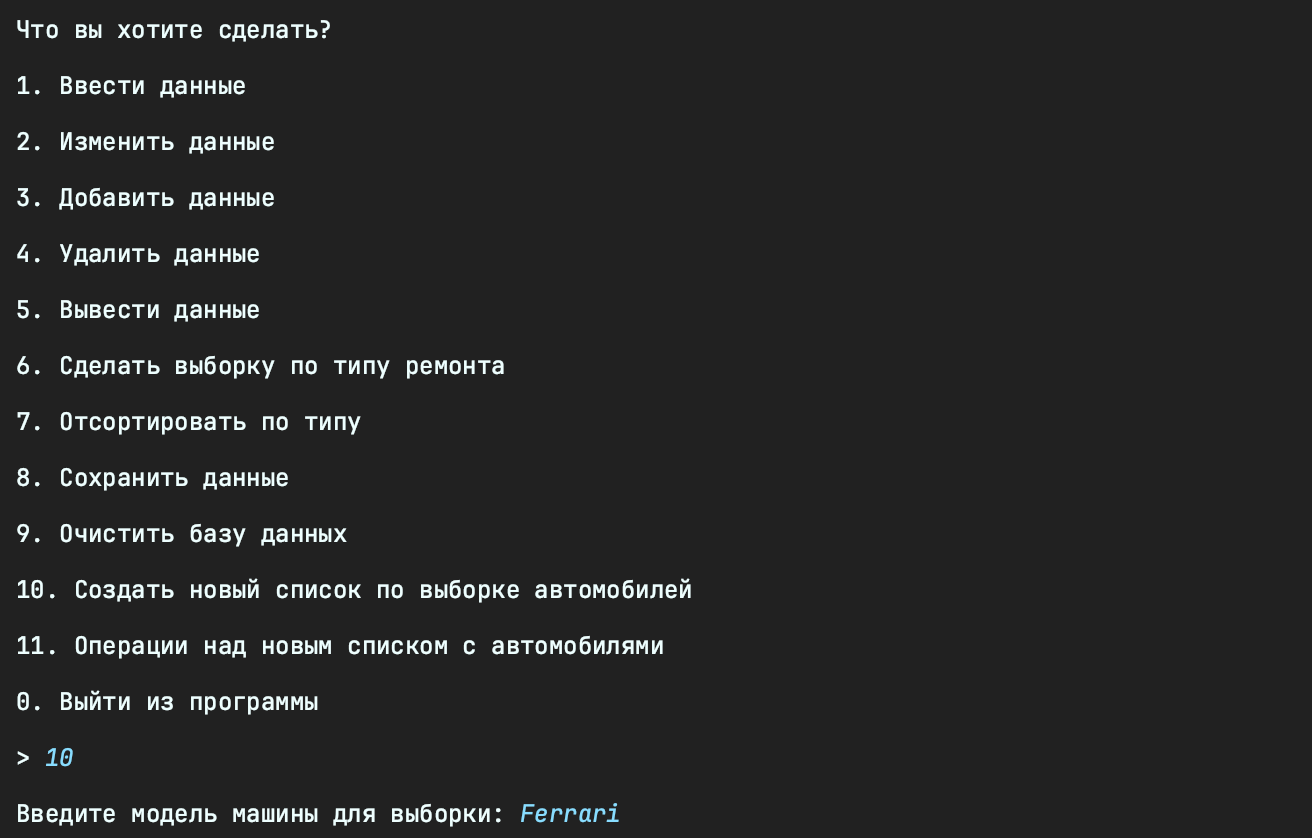
}

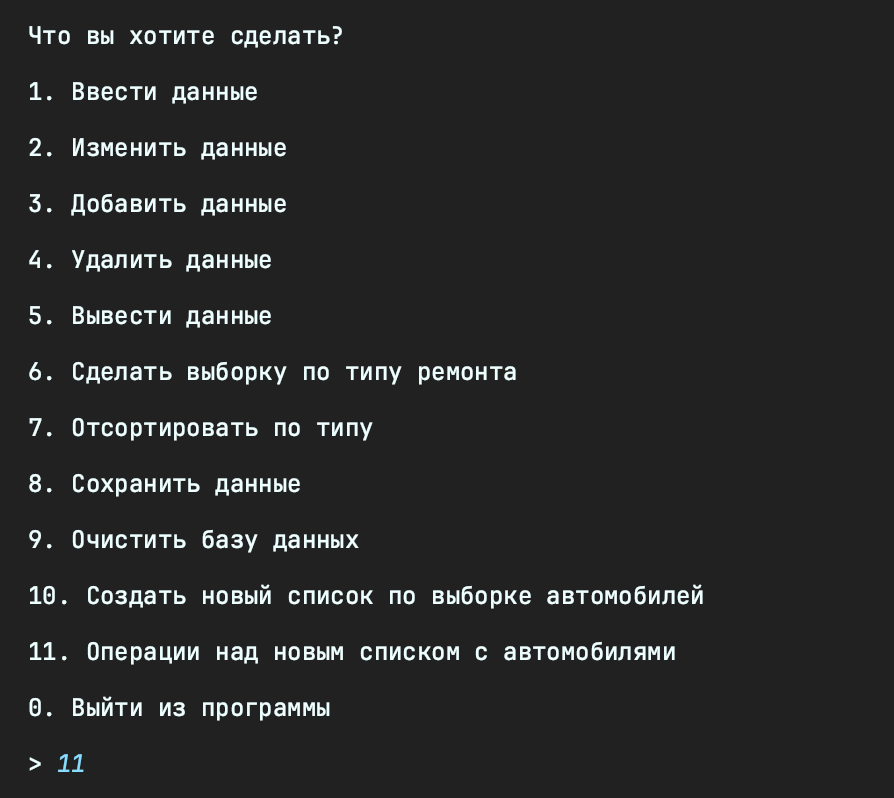
system("clear"); // Очистка экрана для улучшения визуального восприятия

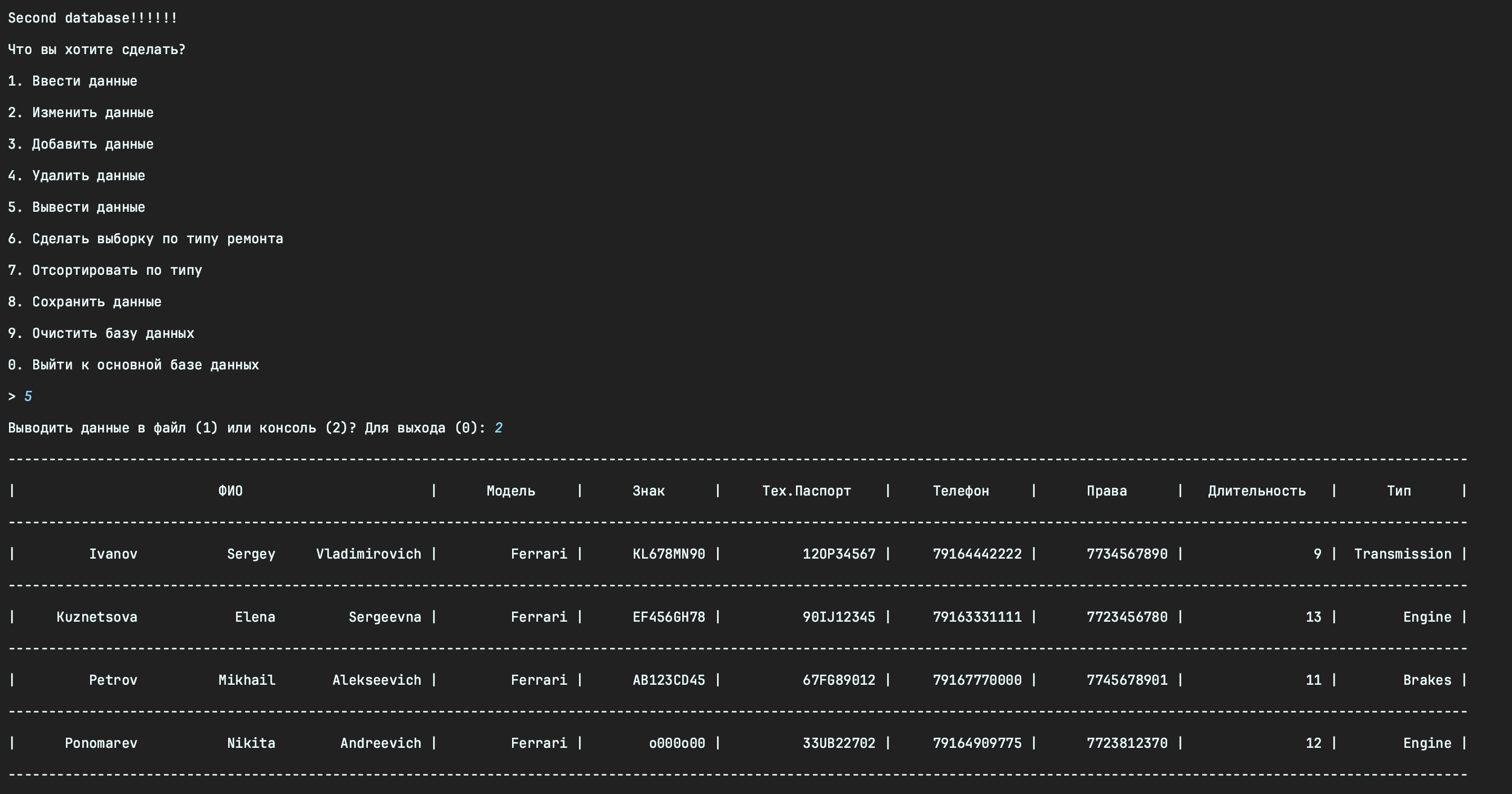
} while (variant != 0);

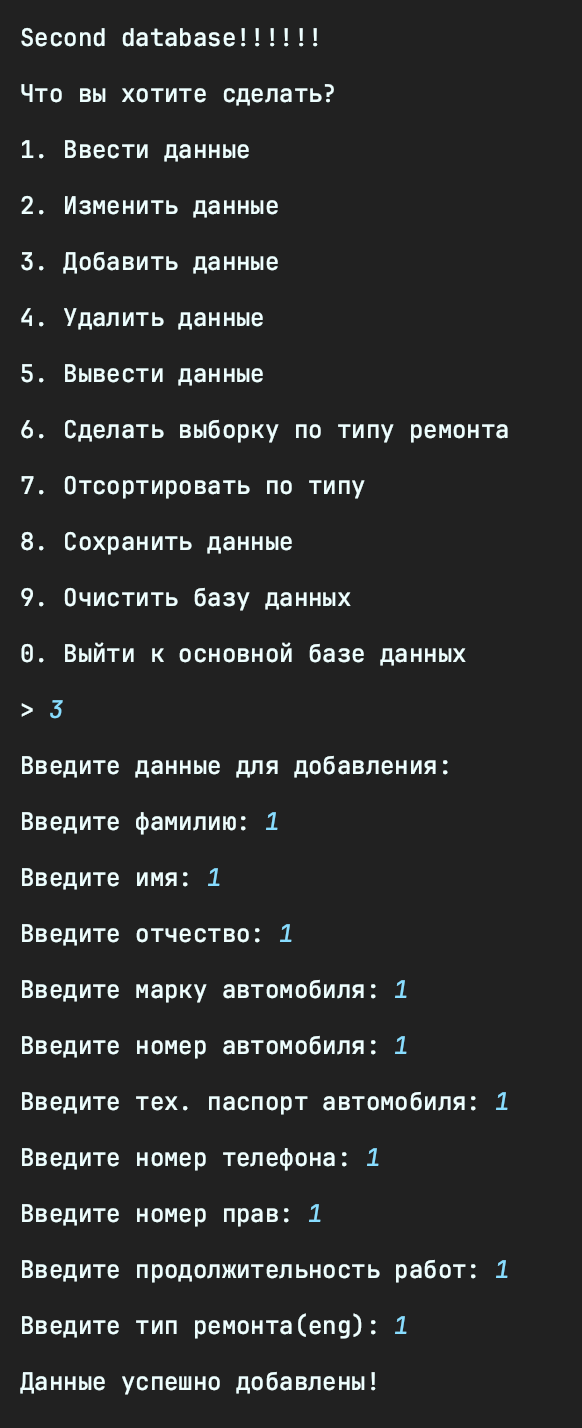
}

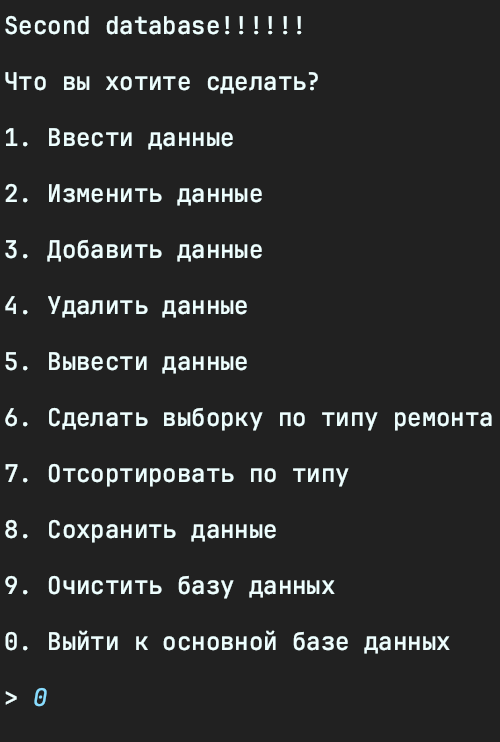
## **Результат работы новой функции программы:**

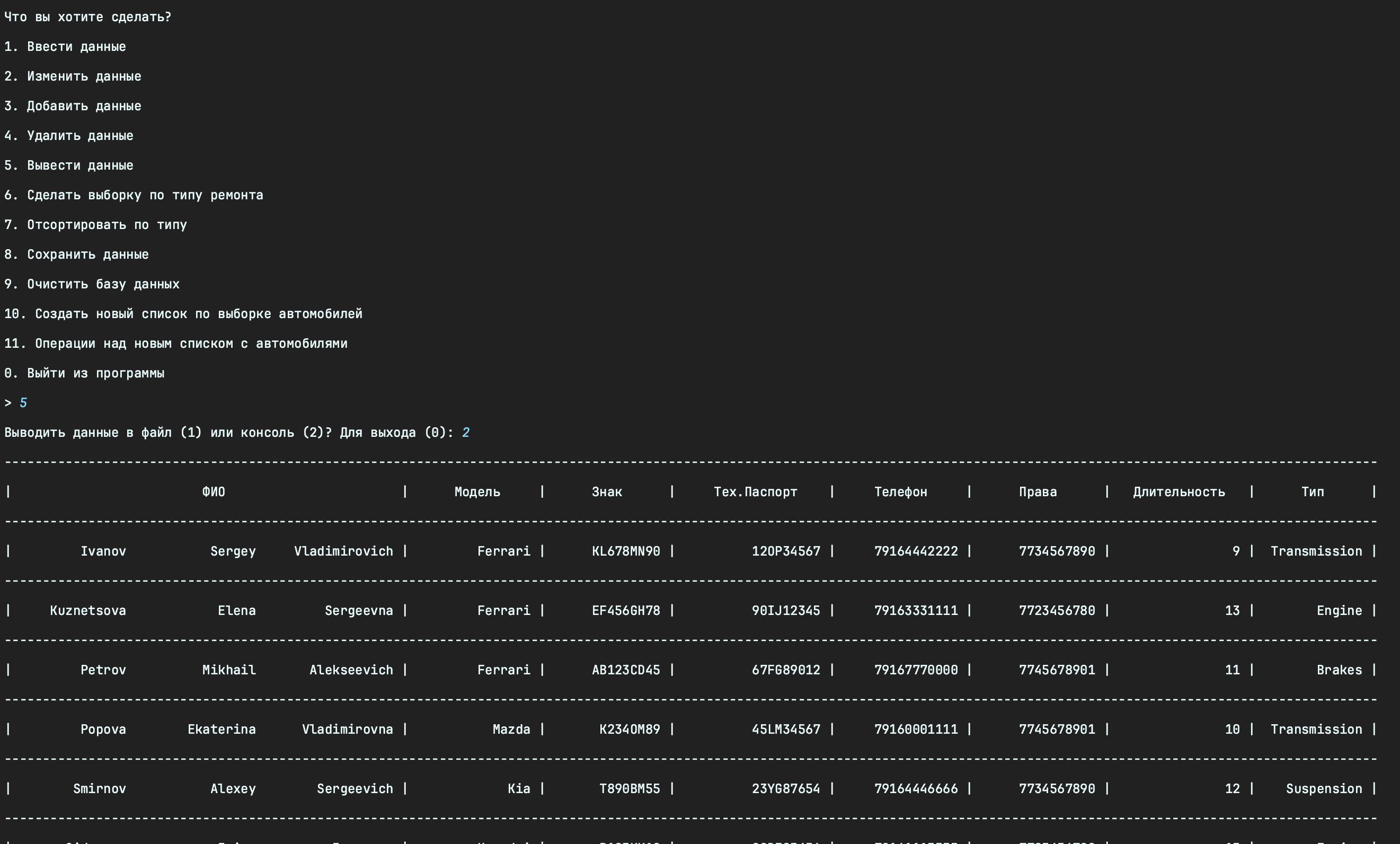












**Вывод**

В ходе работы над курсовым проектом была разработана программа для обработки базы данных автомастерской.

Каждый клиент — это элемент двусвязного линейного списка, который имеет следующие параметры (характеристики): фамилия, имя, отчество, марка авто, техпаспорт, телефон, права на вождение автомобилем, продолжительность ремонта, тип ремонта.

Были реализованы функции ввода, вывода, редактирования, удаления, сортировки и выборки. Каждую функцию пользователь может вызывать с помощью меню.

Была реализована дополнительная задача по созданию новой базы данных по выборке автомобилей, над этой базой данных возможны все те же операции, как и над основной.