МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное образовательное учреждение высшего образования

«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ   
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

КАФЕДРА КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ И ПРОГРАММНОЙ ИНЖНЕРИИ

КУРСОВОЙ ПРОЕКТ  
ЗАЩИЩЕН С ОЦЕНКОЙ

РУКОВОДИТЕЛЬ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| доц., канд. техн. наук |  |  |  | В.А. Матьяш |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

|  |
| --- |
| ПОЯСНИТЕЛЬНАЯ ЗАПИСКА К КУРСОВОМУ ПРОЕКТУ |
| ИСПОЛЬЗОВАНИЕ ЗАДАННЫХ СТРУКТУР ДАННЫХ И АЛГОРИТМОВ ПРИ РАЗРАБОТКЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ИНФОРМАЦИОННОЙ СИСТЕМЫ |
| по дисциплине: СТРУКТУРЫ И АЛГОРИТМЫ ОБРАБОТКИ ДАННЫХ |
|  |
|  |

РАБОТУ ВЫПОЛНИЛ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СТУДЕНТГР. № | 4936 |  |  |  | Е.А. Цыганкова |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург 2021

****Содержание****

[1. Задание на курсовой проект 3](#_Toc69935026)

[2. Введение 6](#_Toc69935027)

[3. Алгоритмы и структуры данных 7](#_Toc69935028)

[4. Описание программы 11](#_Toc69935029)

[5. Тестирование программы 15](#_Toc69935030)

[6. Заключение 32](#_Toc69935031)

[7. Список использованной литературы 33](#_Toc69935032)

[Приложение А. 34](#_Toc69935033)

# 1. Задание на курсовой проект

Цель курсового проектирования: изучение структур данных и алгоритмов их обработки, а также получение практических навыков их использования при разработке программ.

Задача курсового проекта: разработка информационной системы для заданной предметной области с использованием заданных структур данных и алгоритмов.

Вариант №985

− предметная область – «Обслуживание клиентов в бюро проката автомобилей» (985 mod 6 = 1);

− метод хеширования – Закрытое хеширование с линейным опробованием (985 mod 4 = 1);

− метод сортировки – Слиянием (985 mod 7 = 5);

− вид списка – Линейный однонаправленный (985 mod 5 = 0);

− метод обхода дерева – обратный (985 mod 3 = 1);

− алгоритм поиска слова в тексте – прямой (985 mod 2 = 1);

**1.1 Обслуживание клиентов в бюро проката автомобилей**

1.1.1. Информационная система для предметной области «Обслуживание клиентов в бюро проката автомобилей» должна осуществлять ввод, хранение, обработку и вывод данных о:

– клиентах;

– автомобилях, принадлежащих бюро проката;

– выдаче на прокат и возврате автомобилей от клиентов.

1.1.2. Данные о каждом клиенте должны содержать:

– Номер водительского удостоверения – строка формата «RR AA NNNNNN», где RR – код региона (цифры); AA – серия (буквы из следующего множества: А, В, Е, К, М, Н, О, Р, С, Т, У, Х); NNNNNN – порядковый номер удостоверения (цифры). Код, серия и номер отделяются друг от друга пробелами;

– ФИО – строка;

– Паспортные данные – строка;

– Адрес – строка.

Примечание: длина строк (кроме номера водительского удостоверения) определяется студентом самостоятельно.

1.1.3. Данные о клиентах должны быть организованны в виде АВЛ-дерева поиска, упорядоченного по «номеру водительского удостоверения».

1.1.4. Данные о каждом автомобиле должны содержать:

– Государственный регистрационный номер – строка формата «ANNNAA-NN», где N –цифра; A – буква из следующего множества: А, В, Е, К, М, Н, О, Р, С, Т, У, Х;

– Марку – строка;

– Цвет – строка;

– Год выпуска – целое;

– Признак наличия – логическое.

Примечание: длина строк (кроме «Государственного регистрационного номера») определяется студентом самостоятельно.

1.1.5. Данные об автомобилях должны быть организованны в виде хеш-таблицы, первичным ключом которой является «Государственный регистрационный номер». Метод хеширования определяется вариантом задания.

1.1.6. Данные о выдаче на прокат или возврате автомобилей от клиентов должны содержать:

– строку, формат которой соответствует аналогичной строке в данных о клиентах;

– Государственный регистрационный номер – строка, формат которой соответствует аналогичной строке в данных об автомобилях;

– Дату выдачи – строка;

– Дату возврата – строка.

Примечание: наличие в этих данных записи, содержащей в поле «Номер водительского удостоверения» значение X и в поле «Государственный регистрационный номер» значение Y, означает выдачу клиенту с номером водительского удостоверения X автомобиля с государственным регистрационным номером Y. Отсутствие такой записи означает, что клиенту с номером водительского удостоверения X не выдавался автомобиль с номером Y.

1.1.7. Данные о выдаче на прокат или возврате автомобилей от клиентов должны быть организованны в виде списка, который упорядочен по первичному ключу – «Государственный регистрационный номер». Вид списка и метод сортировки определяются вариантом задания.

1.1.8. Информационная система «Обслуживание клиентов в бюро проката автомобилей» должна осуществлять следующие операции:

– регистрацию нового клиента;

– снятие с обслуживания клиента;

– просмотр всех зарегистрированных клиентов;

– очистку данных о клиентах;

– поиск клиента по «номер водительского удостоверения». Результаты поиска – все сведения о найденном клиенте и государственный регистрационный номер автомобиля, который ему выдан;

– поиск клиента по фрагментам ФИО или адреса. Результаты поиска – список найденных клиентов с указанием номера водительского удостоверения, ФИО и адреса;

– добавление нового автомобиля;

– удаление сведений об автомобиле;

– просмотр всех имеющихся автомобилей;

– очистку данных об автомобилях;

– поиск автомобиля по «Государственному регистрационному номеру». Результаты поиска – все сведения о найденном автомобиле, а также ФИО и номер водительского удостоверения клиента, которому выдан этот автомобиль;

– поиск автомобиля по названию марки автомобиля. Результаты поиска – список найденных автомобилей с указанием «Государственный регистрационный номер», марки, цвета, года выпуска;

– регистрацию отправки автомобиля в ремонт;

– регистрацию прибытия автомобиля из ремонта;

– регистрацию выдачи клиенту автомобиля на прокат;

– регистрацию возврата автомобиля от клиентов.

1.1.9. Состав данных о клиенте или автомобиле, выдаваемых при просмотре всех зарегистрированных клиентов или просмотре всех автомобилей, принадлежащих бюро проката, определяется студентом самостоятельно, но должен содержать не менее двух полей.

1.1.10. Метод поиска автомобиля по марке определяется студентом самостоятельно. Выбранный метод необходимо сравнить с альтернативными методами.

1.1.11. Поиск клиента по фрагментам ФИО или адреса должен осуществляться путем систематического обхода АВЛ-дерева поиска. Метод обхода определяется вариантом задания. При поиске клиента по фрагментам ФИО или адреса могут быть заданы как полное ФИО или адрес, так и их части (например, только фамилия клиента без имени и отчества, только название улицы из адреса). Для обнаружения заданного фрагмента в полном ФИО или адресе должен применяться алгоритм поиска слова в тексте, указанный в варианте задания.

1.1.12. Регистрация отправки автомобиля на ремонт должна осуществляться только при наличии этого автомобиля (значение поля «Признак наличия» для соответствующего автомобиля имеет значение «Истина»). При этом значение поля «Признак наличия» для соответствующего автомобиля изменяется на значение «Ложь».

1.1.13. При регистрации прибытия автомобиля из ремонта значение поля «Признак наличия» для соответствующего автомобиля изменяется на значение «Истина».

1.1.14. Регистрация выдачи автомобиля клиенту должна осуществляться только при наличии свободного выдаваемого автомобили (значение поля «Признак наличия» для соответствующего автомобиля имеет значение «Истина»).

1.1.15. При регистрации выдачи автомобиля клиенту или возврата автомобиля от клиента должно корректироваться значение поля «Признак наличия» для соответствующего автомобиля.

1.1.16. При снятии с обслуживания клиента должны быть учтены и обработаны ситуации, когда у клиента имеется выданный автомобиль. Аналогичным образом следует поступать и с удалением сведений об автомобилях.

# 2. Введение

В настоящее время электронным вычислительным машинам (ЭВМ) приходится не только считывать и выполнять определенные алгоритмы, но и хранить значительные объемы информации. При этом к хранимой информации нужно иметь быстрый доступ. Она в некотором смысле представляет собой абстракцию того или иного фрагмента реального мира и состоит из определенного множества данных, относящихся к какому-либо объекту или предметной области.

Современная ЭВМ хранит и обрабатывает только один вид данных (двоичные цифры) и может работать с ними только в соответствии с алгоритмами, которые определяются системой команд центрального процессора. Зачастую данные имеют форму чисел, текстов, символов и более сложных структур типа множеств, массивов, списков и деревьев.

Физическая структура данных отражает представление данных в памяти ЭВМ. Рассмотрение структур данных без учета представления их в памяти ЭВМ обычно называют абстрактным, как и сами структуры. Оперирование абстрактными структурами данных позволяет программисту изменять детали логики работы программы независимо от подробностей того, как эти структуры будут реализованы.

Использование таких абстрактных структур данных и алгоритмов работы с ними позволило реализовать информационную систему «Обслуживание клиентов в бюро проката автомобилей».

# 3. Алгоритмы и структуры данных

**3.1 Структуры**

Данные о клиенте хранятся в структуре Customer, данные об авто хранятся в структуре Auto, о выдаче на прокат – в Rental.

В структуре Tree хранятся данные о содержимом узла дерева, высота, указатель на предка, на меньшего потомка и на большего.

В структуре Keys хранятся данные об авто и логические данные о состоянии ячейки таблицы. Структура SecondKey создана для поиска по вторичным ключам в хеш-таблице, в ней хранятся данные о марке автомобиля и вектор с адресами ячеек, в которых находятся авто с такой маркой.

В структуре List хранятся данные о выдаче на прокат автомобиля, и указатель на следующий элемент списка.

//структура с информацией о клиенте

struct Customer{

char drivenumber[N];//номер вод удостоверения

char name[M];//фио

char passport[P];//паспорт

char address[A];//адрес

};

//структура элемента дерева

struct Tree{

Customer elem; //содержимое узла

int height; //высота узла

Tree\* Prev; //указатель на предка

Tree\* Left; //указатель на меньшего потомка

Tree\* Right; //указатель на большего потомка

};

//структура с информацией об авто

struct Auto{

char regnumber[R];//регистрационный номер

char brand[B];//марка

char color[C];//цвет

int year;//год выпуска

bool exist;//наличие

};

//структура ключа

struct Keys{

bool empty, deleted;

Auto car;

};

//структура вторичного ключа

struct SecondKey {

char brand[B];//марка

vector<unsigned int> Address;//инвертированные индексы

};

//структура проката

struct Rental{

char drivenumber[N];//номер вод уд-ия

char regnumber[R];//регистрационный номер авто

char dateissue[R];//дата выдачи

char dateret[R];//дата возврата

};

//структура списка

struct List {

Rental data;

List\* ptr;

};

**3.2 Закрытое хеширование с линейным опробованием**

Хеширование – это метод случайного упорядочивания элементов, который обеспечивает сокращение времени доступа к данным. При таком алгоритме данные организуются в виде таблицы, но место хранения данных в таблице «вычисляется» при помощи хеш-функции по значению ключа.

При закрытом хешировании в хеш-таблице хранятся непосредственно элементы. При использовании закрытого хеширования для решения коллизий используются алгоритмы повторного хеширования.

Линейное опробование – метод повторного хеширования. Под методом линейного опробования понимается последовательный перебор адресов сегментов таблицы с фиксированным шагом.

**3.3 Сортировка слиянием**

Процедуру сортировки слиянием можно описать следующим образом:

1) если в рассматриваемом массиве один элемент, то он уже отсортирован – алгоритм завершает работу;

2) исходный массив разбивается на k частей;

3) сортируем каждую часть массива;

4) «сливаем» попарно части массива;

5) проделываем пункты 3 и 4 до тех пор, пока все части массива не будут объединены.

Теоретические временные и пространственные сложности данного метода сортировки приведены в таблице 1.

Таблица 1.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Метод сортировки | Характеристики | | | |
| Tmax | Tmid | Tmin | Vmax |
| Слиянием | O(n\*log n) | | | O(n) |

**3.4 Линейный однонаправленный список**

Список – это абстрактная структура данных, элементы которой содержат пользовательские данные и указатели на следующий (иногда и на предыдущий) элемент списка.

В линейных однонаправленных списках каждый элемент содержит в дополнение к полям данных одно поле указателя, который указывает на следующий элемент списка. Если элемент является последним, то указатель на следующий элемент является пустым.

Следует обратить особое внимание на то, что при выполнении любых операций с линейным однонаправленным списком необходимо всегда обеспечивать хранение указателя на первый элемент. Если указатель на первый элемент будет утерян (недоступен), то это приведет к потере данных всего списка.

В качестве недостатка такой организации нужно упомянуть, что осуществлять переходы между элементами списка можно только в одном направлении. Это увеличивает время, затрачиваемое на работу со списком.

**3.5 Обратный метод обхода дерева**

Способ обхода рекурсивно можно определить следующим образом:

1) если дерево Tree является пустым деревом, то в список обхода заносится пустая запись;

2) если дерево Tree состоит из одной вершины, то в список обхода записывается эта вершина;

3) если Tree – дерево с корнем n и поддеревьями Tree1, Tree2, …, Treek, то сначала посещаются в обратном порядке вершины поддерева Tree1, далее последовательно в обратном порядке посещаются вершины поддеревьев Tree2, …, Treek. Последним посещается корень n (рис. 1).

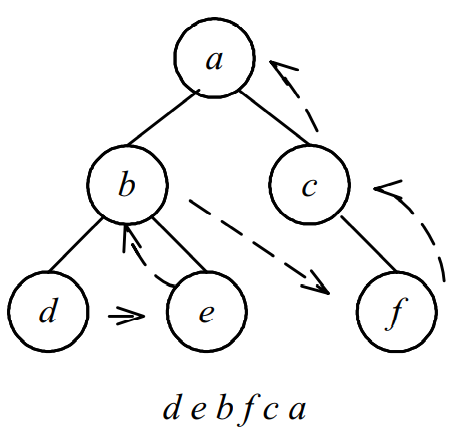


Рис. 1. Обратный обход дерева

**3.6 Прямой алгоритм поиска слова в тексте**

Предполагается, что символы текста и слова будут сравниваться поэлементно. На первом этапе работы алгоритма искомое слово сопоставляется с текстом так, что первый символ слова соотносится с первым символом текста. Далее происходит поэлементное сравнение первого символа слова с первым символом текста, второго символа слова со вторым символом текста и так далее. Если все символы совпадают, то результат работы алгоритма положительный (слово в тексте найдено). Если на каком-либо символе обнаружилось несоответствие, то происходит смещение слова на одну позицию так, что первый символ слова соответствует второму символу текста. Далее повторяется посимвольное сравнение. Смещение слова может повторяться до тех пор, пока не будет обнаружено полное совпадение символов либо пока не будет достигнут конец массива.

Использование данного примитивного алгоритма достаточно эффективно в том случае, если для поиска строки потребуется небольшое количество смещений. К плюсам такого алгоритма можно отнести его понятность, он не требует предварительной подготовки данных. В языках высокого уровня данный алгоритм, подразумевающий сравнение участков памяти, достаточно хорошо оптимизирован.

**3.7 Поиск по вторичным ключам (инвертированные индексы)**

Для хеш-таблицы строится отдельный набор данных, содержащий так называемые инвертированные индексы. Вспомогательный набор содержит для каждого значения вторичного ключа отсортированный список адресов записей таблицы, которые содержат данный ключ.

Поиск осуществляется по вспомогательной структуре достаточно быстро, так как фактически отсутствует необходимость обращения к основной структуре данных. Область памяти, используемая для индексов, является относительно небольшой по сравнению с другими методами организации таблиц.

Недостатками данной системы являются большие затраты времени на составление вспомогательной структуры данных и ее обновление. Причем эти затраты возрастают с увеличение объема базы данных.

Альтернативным методом является организация вспомогательной структуры в виде битовых карт. Основным преимуществом такой организации является очень простая и эффективная организация обработки сложных запросов, которые могут объединять значения ключей различными логическими предикатами, что отсутствует у первого метода. К недостаткам битовых карт следует отнести увеличение длины строки карты пропорционально длине таблицы. При этом заполненность карты единицами уменьшается с увеличением длины файла.

Система инвертированных индексов является чрезвычайно удобной и эффективной при организации поиска в больших таблицах, чего нельзя сказать об альтернативном методе.

# 4. Описание программы

Программа реализована на языке С++ в виде консольного приложения. В главной функции main() реализовано меню пользователя, в котором каждому действию соответствует определенная цифра. Реализованы следующие функции для работы с данными об автомобилях:

– добавление нового автомобиля;

– удаление сведений об автомобиле;

– просмотр имеющихся автомобилей;

– очистка данных об автомобилях;

– поиск автомобиля по государственному регистрационному номеру;

– поиск автомобиля по названию марки;

– регистрация отправки авто в ремонт;

– регистрация прибытия авто из ремонта.

Функции для работы с данными о клиентах:

– регистрация нового клиента;

– снять клиента с обслуживания;

– просмотр клиентов;

– очистка данных о клиентах;

– поиск клиента по номеру водительского удостоверения;

– поиск клиента по фрагменту ФИО или адреса.

Функции для работы с данными о выдаче на прокат:

– регистрация выдачи клиенту авто на прокат;

– регистрация возврата авто от клиента;

– просмотр данных о выданных авто.

Также созданы функции чтения из текстовых файлов начальных данных.

**Добавление нового автомобиля**

Функция выполняет добавление записи в хеш-таблицу. При вводе данных выполняется проверка формата введенных данных. При вводе регистрационного номера происходит проверка соответствию формату корректной записи. Автомобиль с уже существующим номером не будет добавлен в таблицу. Если пользователь ввел неуникальный номер, будет предложено ввести № повторно. При вводе марки, цвета проверяется длина введенной строки, если длина больше заданного допустимого количества символов, пользователю предлагается ввести значения повторно. Год выпуска ограничен на ввод (1980-2021). Если авто удалось добавить (в таблице есть место), на экран выводится соответствующее сообщение.

**Удаление сведений об автомобиле**

Функция удаляет запись, найденную по регистрационному номеру, введенному с клавиатуры, и выводит сообщение «Сведения удалены». Если такая запись не найдена, выводится соответствующее сообщение на экран.

**Просмотр имеющихся автомобилей**

Функция выводит на экран содержимое хеш-таблицы в виде таблицы (таблица 2).

Пример выполнения

Таблица 2.

|  |  |  |
| --- | --- | --- |
| Государственный регистрационный номер  Е777КМ-10  Р444АА-33  В632ТТ-30  Р555РО-66 | Марка  опель  лада  рено  нисан | В наличии  нет  да  нет  да |

**Очистка данных об автомобилях**

Функция удаляет все сведения обо всех автомобилях, кроме тех, что находятся в ремонте или прокате.

**Поиск автомобиля по государственному регистрационному номеру**

Функция осуществляет поиск по государственному регистрационному номеру, введенному с клавиатуры, и выводит запись на экран, также если данный автомобиль находится в прокате, вместе с информацией о нем выводятся номер водительского удостоверения и ФИО клиента, которому выдано авто. Если запись не была обнаружена, выводится сообщение об этом.

**Поиск автомобиля по названию марки**

Функция осуществляет поиск по названию марки авто, введенному с клавиатуры, и выводит все найденные записи на экран. Если авто с такой маркой не обнаружены, выводится сообщение «Авто с такой маркой не найдены».

**Регистрация отправки авто в ремонт**

Функция отправляет авто в ремонт. Для этого необходимо ввести номер, расположенный справа от выведенной таблицы. Если будет выбрано авто, которого нет в наличии, пользователю будет предложено заново ввести номер. Признак наличия у автомобиля, отправленного в ремонт, меняется на значение «Нет в наличии». Выводится сообщение «Авто отправлено в ремонт». Если в таблице нет ни одного свободного автомобиля, выводится сообщение о том, что свободных машин нет.

**Регистрация прибытия авто из ремонта**

Функция регистрирует прибытие авто из ремонта, меняя признак наличия на «В наличии». Для этого необходимо ввести регистрационный номер автомобиля, который следует вернуть. В случае если авто находилось в ремонте, выводится сообщение «Авто прибыло из ремонта», если же авто находится в прокате – «Авто находится в прокате».

**Регистрация нового клиента**

Функция выполняет добавление записи АВЛ-дерево. При вводе данных выполняется проверка формата введенных данных. При вводе номера водительского удостоверения и паспортных данных происходит проверка соответствию формату корректной записи (серия и номер паспорта: NNNN NNNNNN, где N – это цифра). Клиент с уже существующим номером не будет добавлен в дерево. Если пользователь ввел неуникальный номер водительского удостоверения или неуникальные паспортные данные, будет предложено ввести данные повторно. При вводе ФИО, адреса проверяется длина введенной строки, если длина больше заданного допустимого количества символов, пользователю предлагается ввести значения повторно.

**Снять клиента с обслуживания**

Функция удаляет запись, найденную по номеру водительского удостоверения, введенному с клавиатуры, и выводит сообщение «Клиент удален». Если такая запись не найдена, выводится соответствующее сообщение на экран.

**Просмотр клиентов**

Функция выводит на экран содержимое АВЛ-дерева в виде таблицы (таблица 3).

Пример выполнения

Таблица 3.

|  |  |
| --- | --- |
| Номер водительского удостоверения | ФИО |
| 33 АТ 000000  11 АТ 000000  00 АТ 000000  22 АТ 000000  44 АТ 000000 | Цыганкова Екатерина Александровна  Герасимова Анастасия Дмитриевна  Петрова Анна Григорьевна  Симонов Олег Игоревич  Касторов Максим Алексеевич |

**Очистка данных о клиентах**

Функция удаляет все сведения обо всех клиентах, в том случае если нет ни одного клиента, которому в данный момент выдана машина. В противном случае выводится сообщение о том, что данные удалить невозможно.

**Поиск клиента по номеру водительского удостоверения**

Функция осуществляет поиск по номеру водительского удостоверения, введенному с клавиатуры, и выводит запись на экран, также если у клиента есть выданный на прокат автомобиль, вместе с информацией о клиенте выводится государственный регистрационный номер авто, которое ему выдано. Если запись не была обнаружена, выводится сообщение об этом.

**Поиск клиента по фрагменту ФИО или адреса**

Функция осуществляет поиск по фрагменту ФИО или адреса, введенному с клавиатуры, и выводит все найденные записи на экран. Для этого необходимо сначала выбрать поле, по которому будет выполняться поиск (ФИО или адрес). Если клиенты с таким фрагментом в ФИО или адресе не обнаружены, программа выводит главное меню на экран.

**Регистрация выдачи клиенту авто на прокат**

Функция осуществляет добавление записи в список проката авто. При регистрации авто на прокат необходимо ввести порядковый номер автомобиля в представленном списке, далее ввести порядковый номер клиента в представленном списке, ввести дату выдачи. После выводится сообщение «Регистрация прошла успешно». При выборе автомобиля, которого нет в наличии, программа выведет предупреждение о том, что это невозможно и предложит ввести номер повторно. Присутствуют проверки на корректный ввод даты выдачи в формате (дд.мм.гг): дата должна быть записана цифрами, соответствовать календарному году, год должен находиться в пределах от (20)00 до (20)21. Если нет ни одного автомобиля в наличии, выводится сообщение об этом.

**Регистрация возврата авто от клиента**

Функция осуществляет запись даты возврата автомобиля от клиента в список проката. Для этого необходимо ввести порядковый номер записи в выпадающем списке, которую требуется отредактировать. Далее ввести дату возврата. Присутствует проверка на то, чтобы дата возврата была позже даты выдачи.

**Просмотр данных о выданных авто**

Функция выводит на экран содержимое списка проката в виде таблицы (таблица 4).

Пример выполнения

Таблица 4.

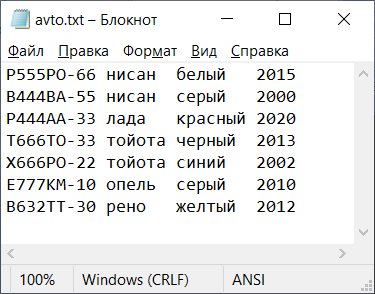
|  |  |  |  |
| --- | --- | --- | --- |
| Водительское удостоверение | Регистрационный номер | Дата выдачи | Дата возврата |
| 33 АТ 000000  22 АТ 000000  33 АТ 000000  00 АТ 000000  22 АТ 000000 | В444ВА-55  В632ТТ-30  Е777КМ-10  Р444АА-33  Т666ТО-33 | 20.05.21  10.02.21  15.01.21  20.02.20  12.03.20 | 21.05.21  21.02.20  20.04.20 |

Листинг программы приведен в приложении А.

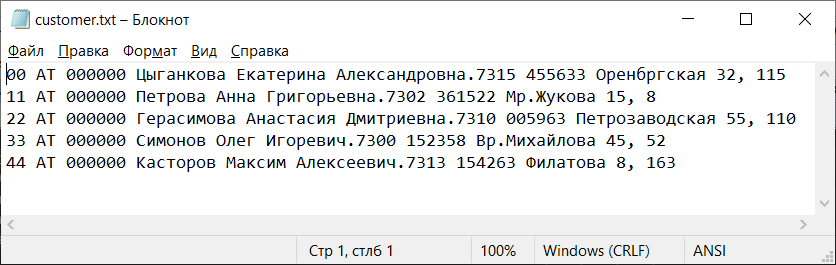
# 5. Тестирование программы

**Исходные данные для тестовых прогонов программы**

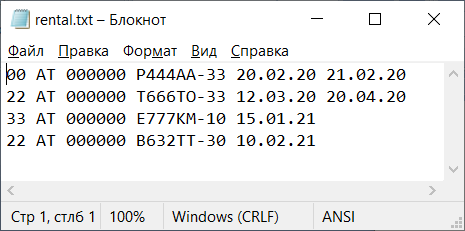
Файл с данными об автомобилях:



Файл с данными о клиентах:

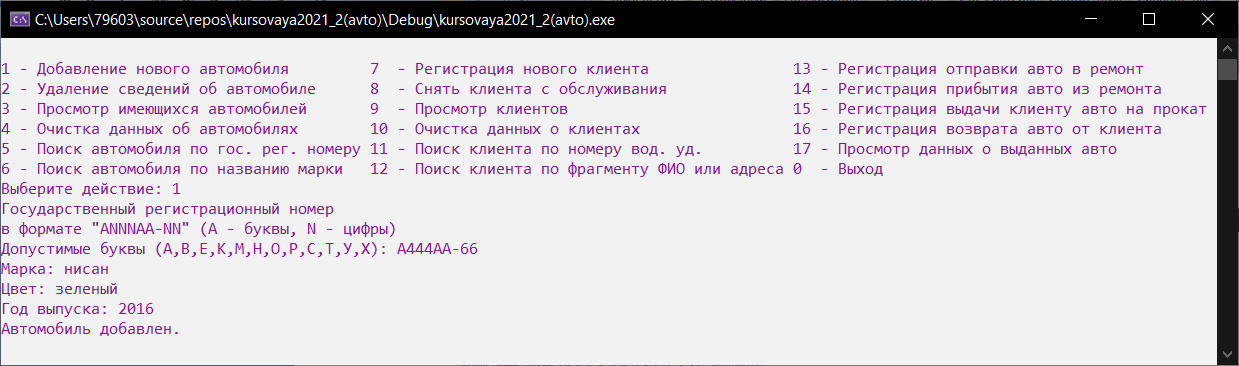


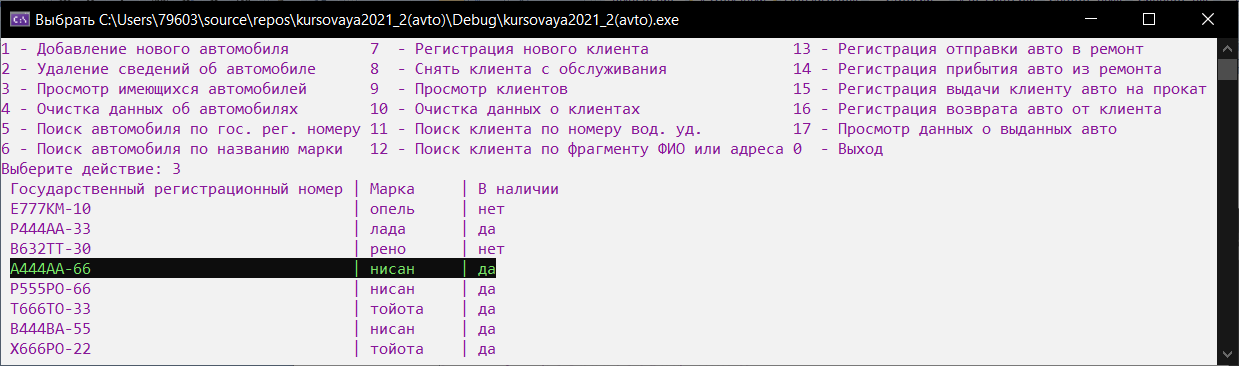
Файл с данными о прокате авто:



**Функции для работы с данными об автомобилях**

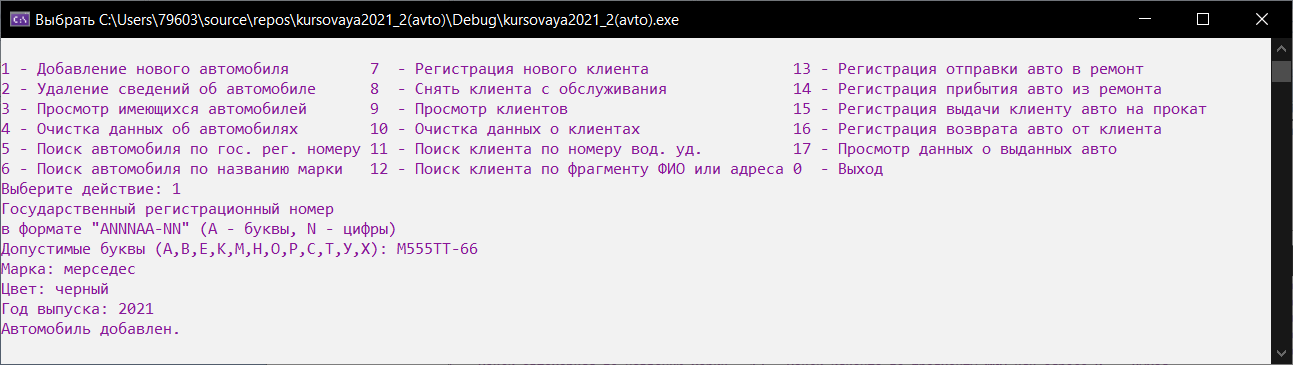
*Добавление нового автомобиля*

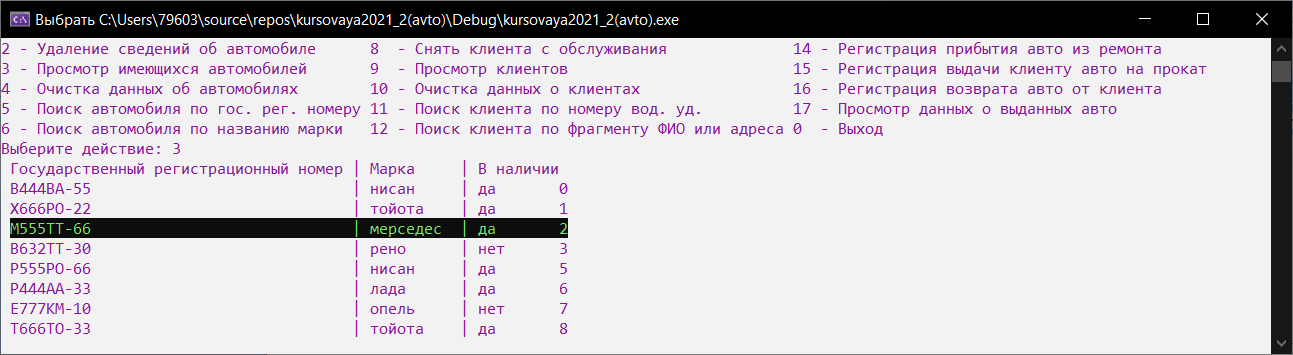




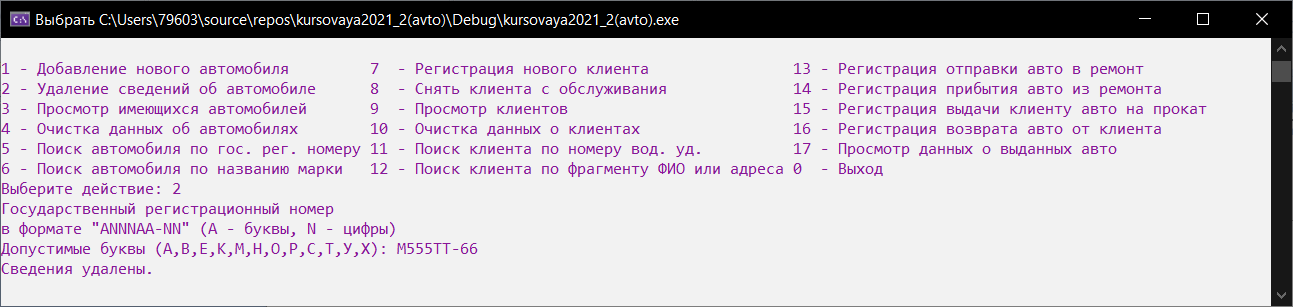
Добавление двух автомобилей с одинаковым ключом. Уменьшим размер таблицы до 9 сегментов и выведем индексы массива.

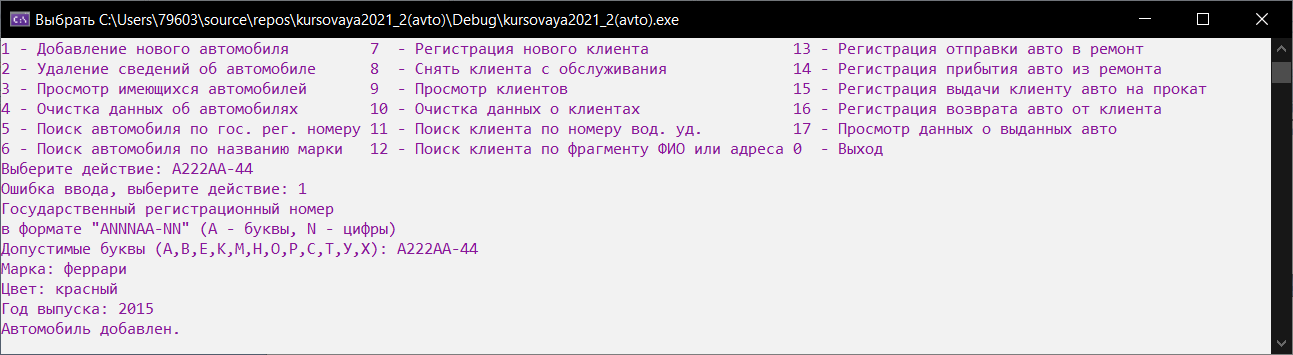
Добавим машину с номером М555ТТ-66 и убедимся в том, что программа сработала верно.



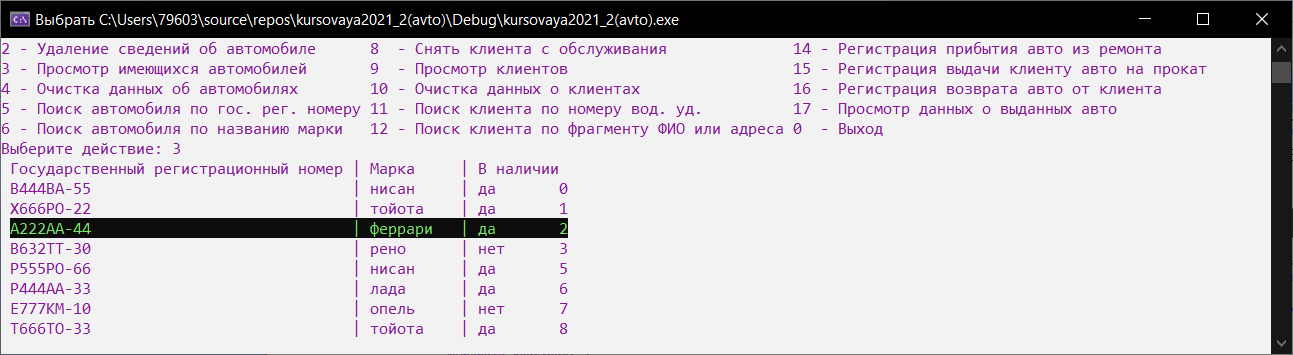


Запись попала в сегмент с индексом 2. Удалим автомобиль и добавим новый с номером А222АА-44.

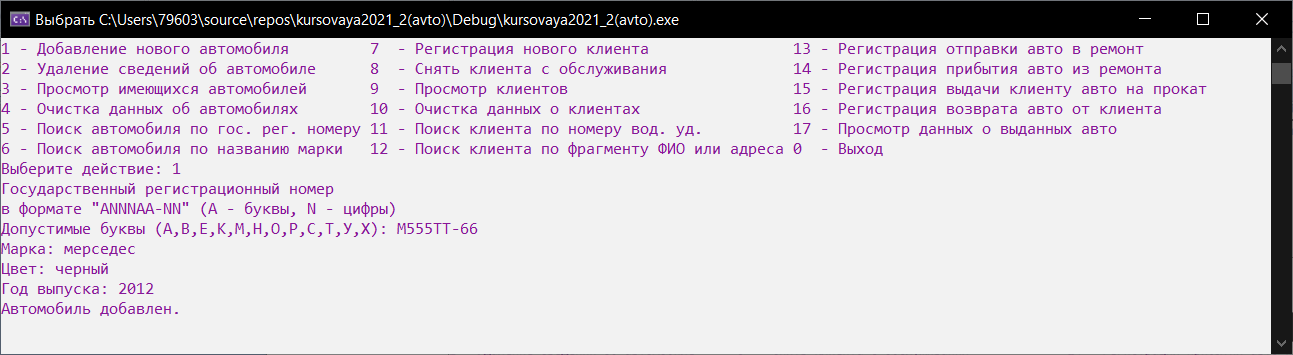




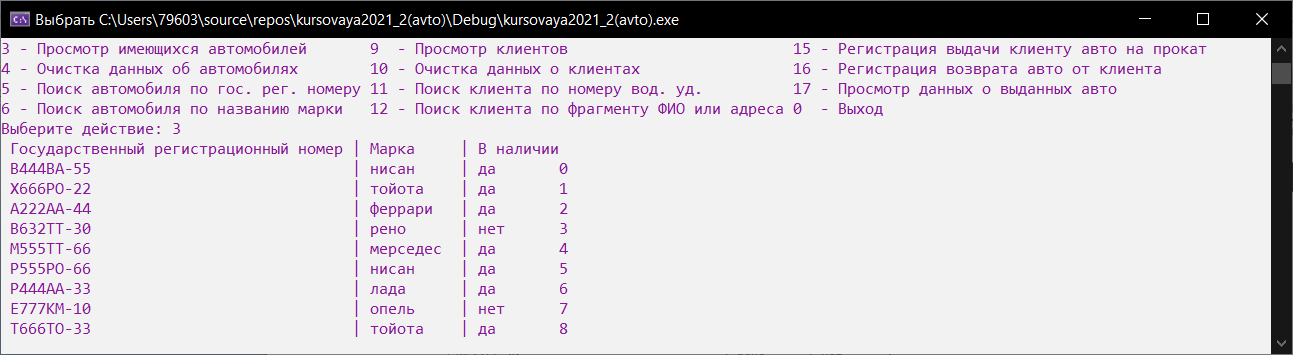
Автомобиль с номером А222АА-44 попал в сегмент с индексом 2.



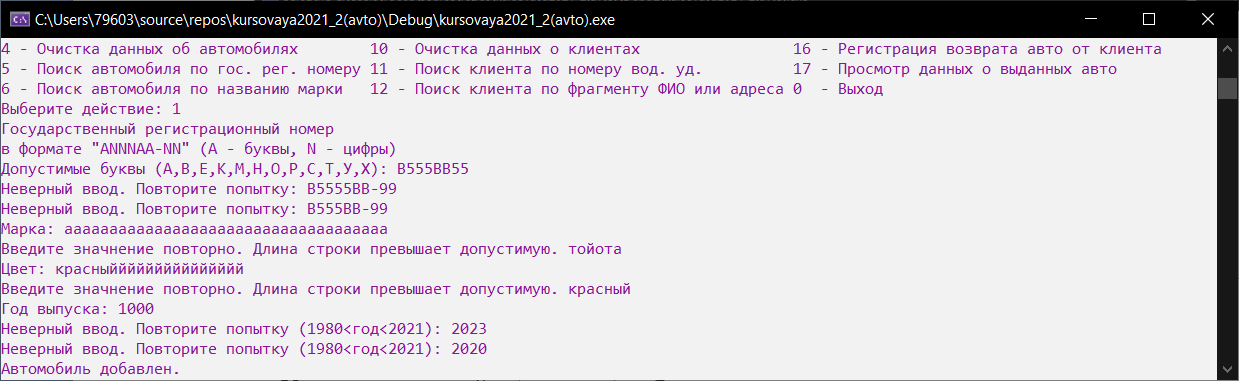
Снова добавим автомобиль с номером М555ТТ-66.



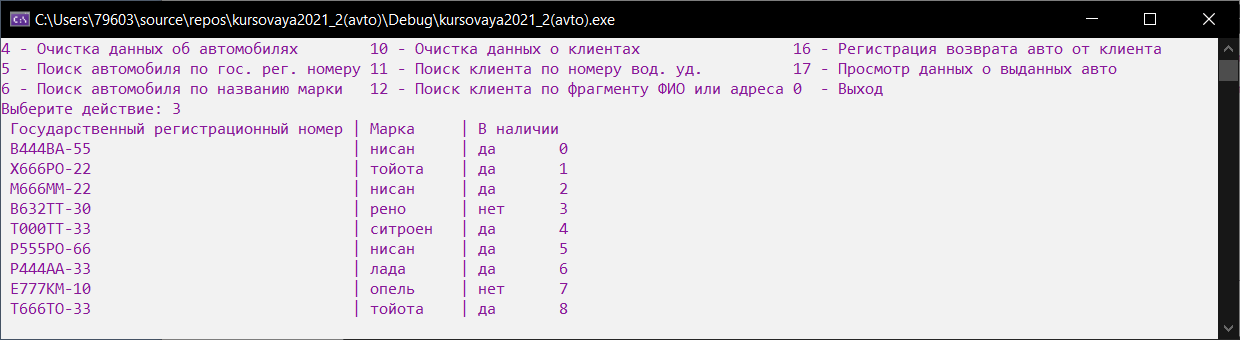
Оба автомобиля находятся в таблице.

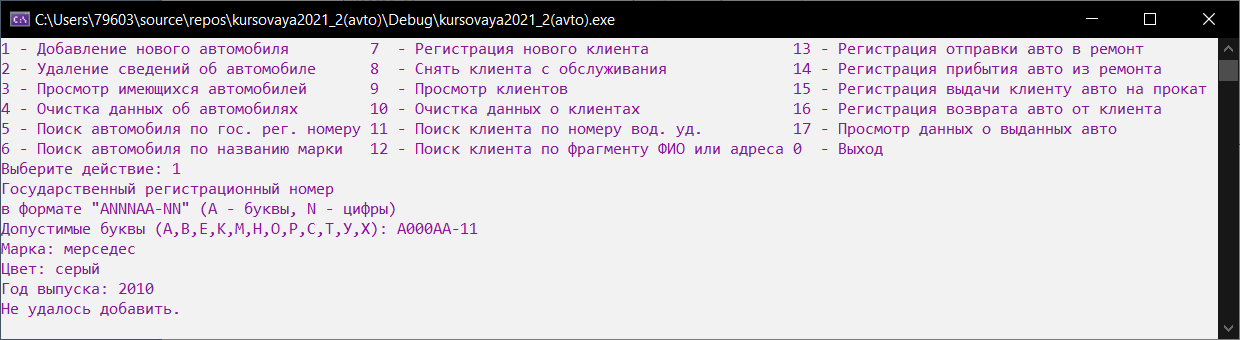


Проверки на ввод:

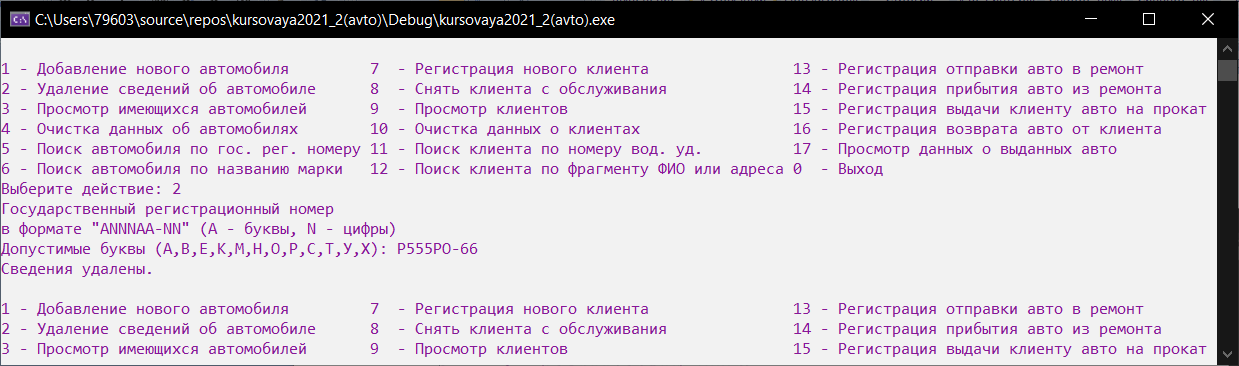


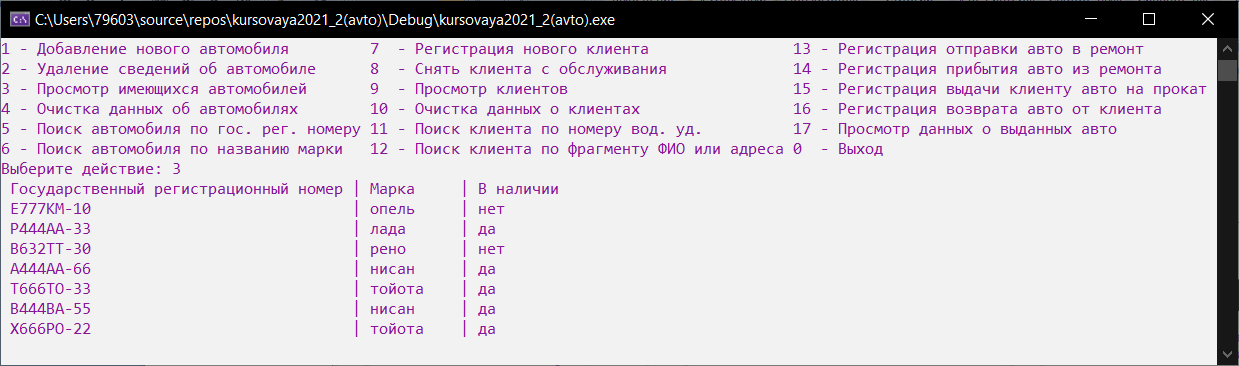
Добавление автомобиля в заполненную таблицу:



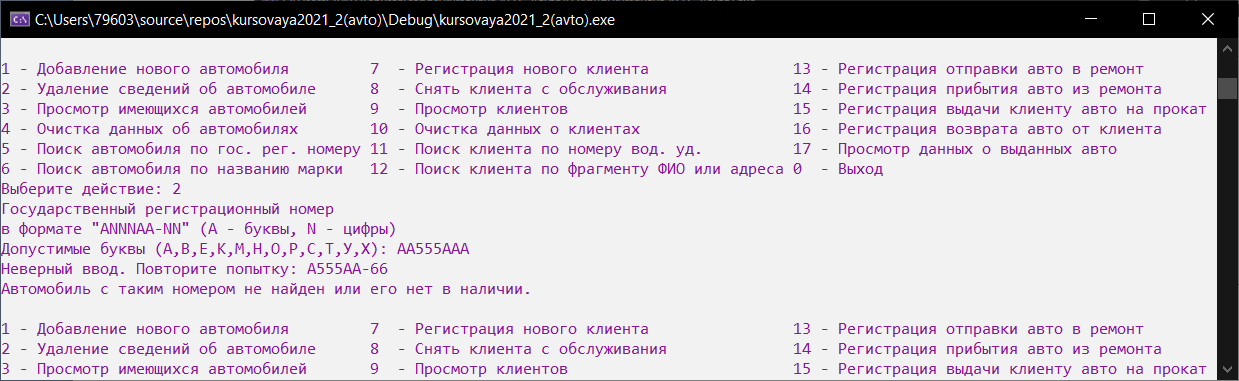


*Удаление сведений об автомобиле*

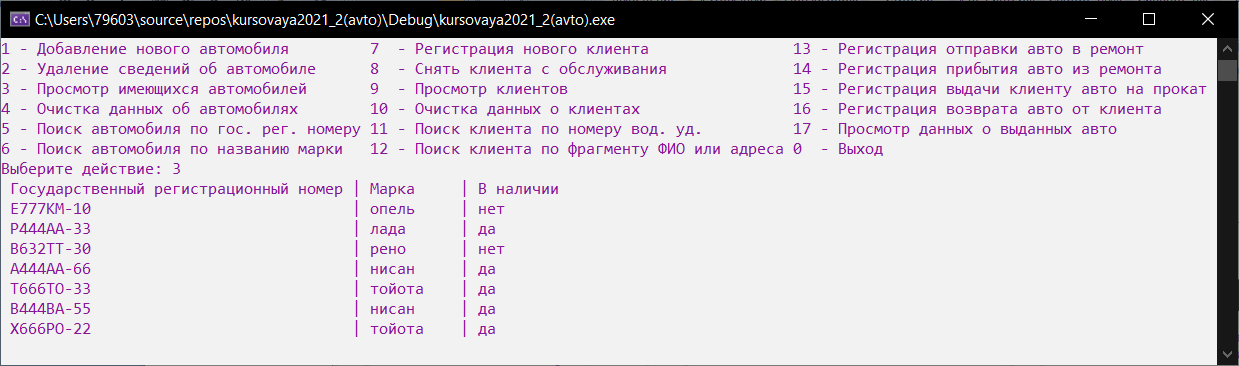




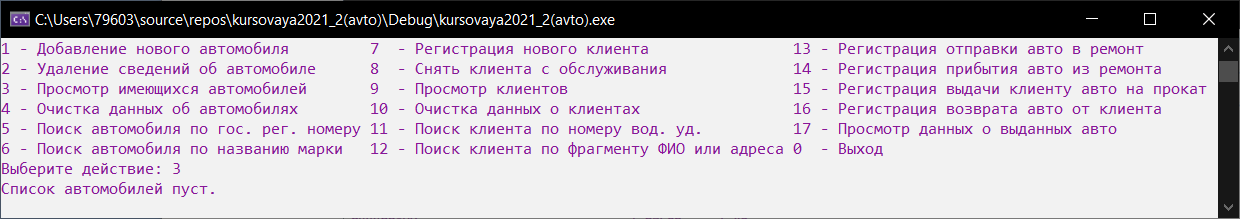
Проверка на ввод (некорректный ввод, несуществующий номер):



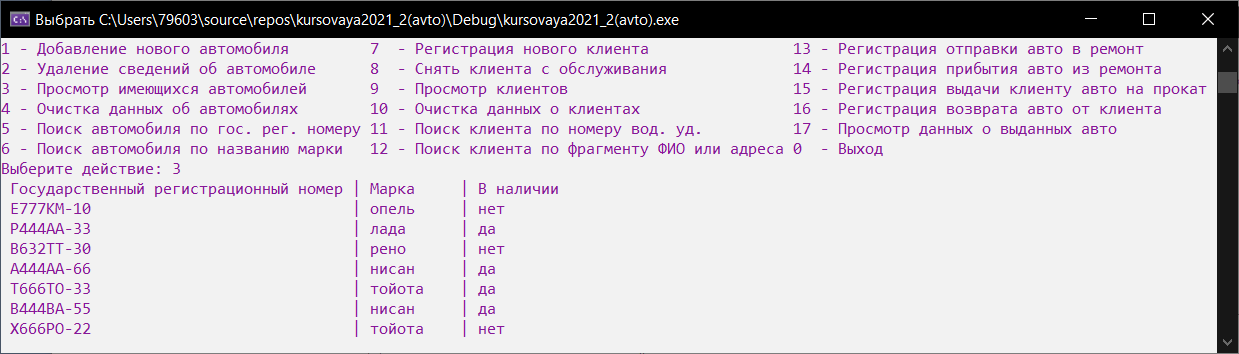
*Просмотр имеющихся автомобилей*

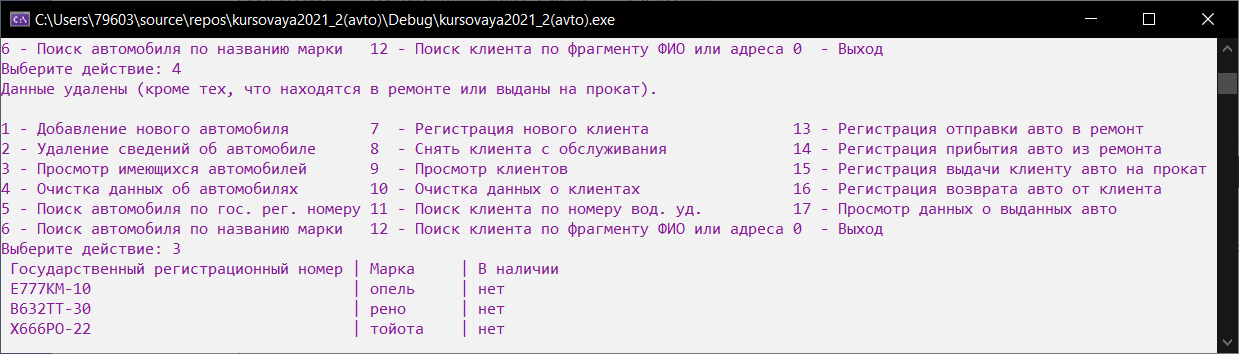


Вызов функции при пустой таблице:

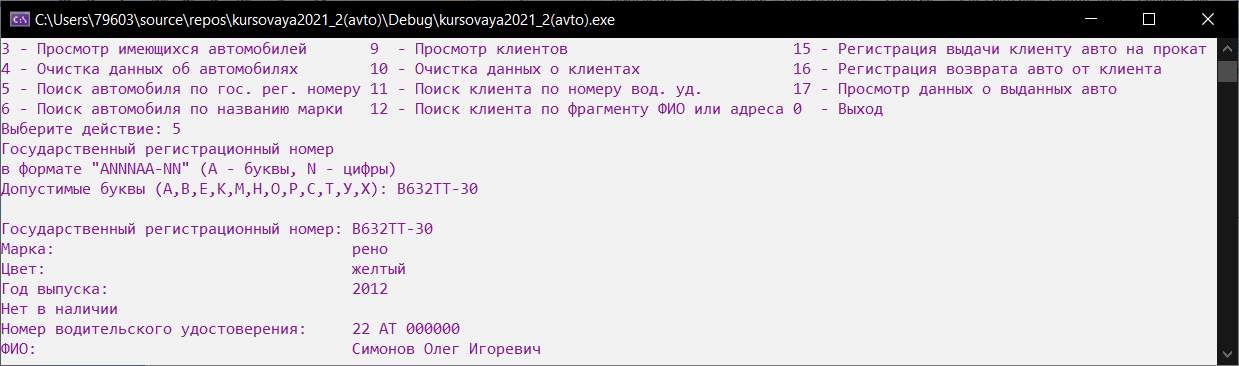


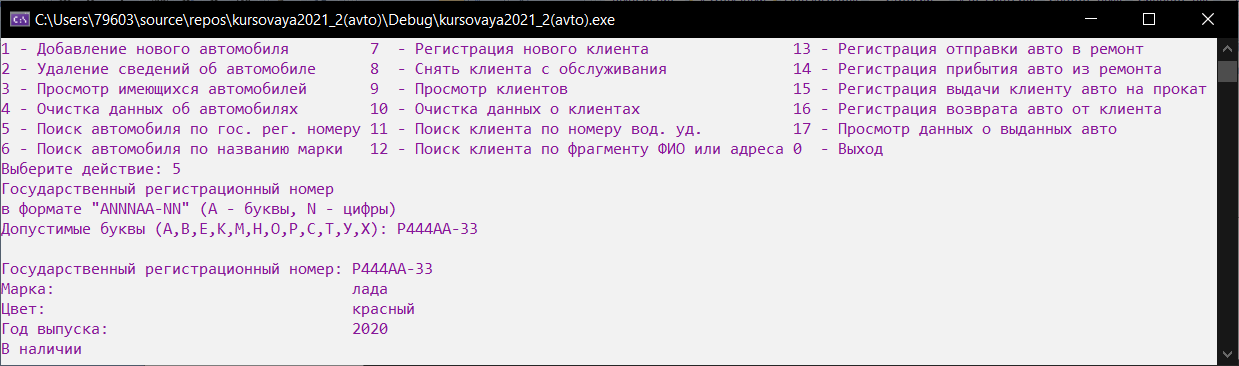
*Очистка данных об автомобилях*



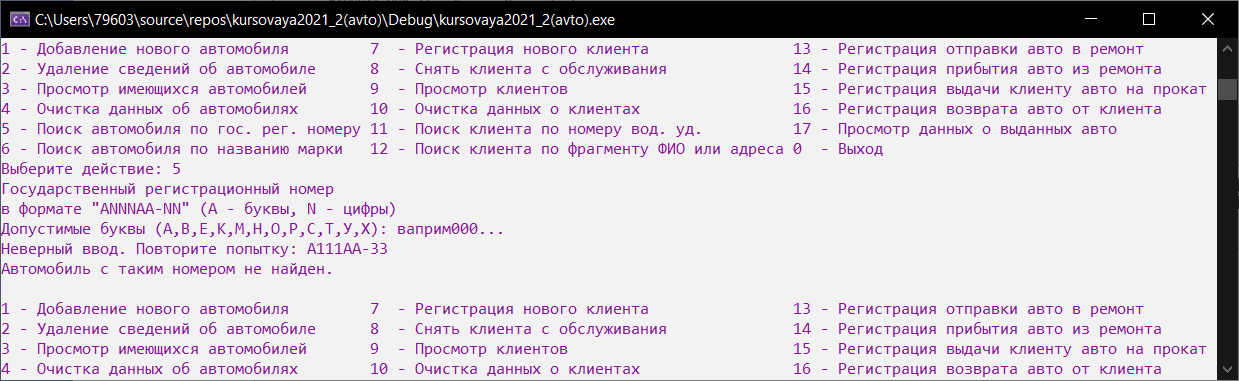


*Поиск автомобиля по государственному регистрационному номеру*

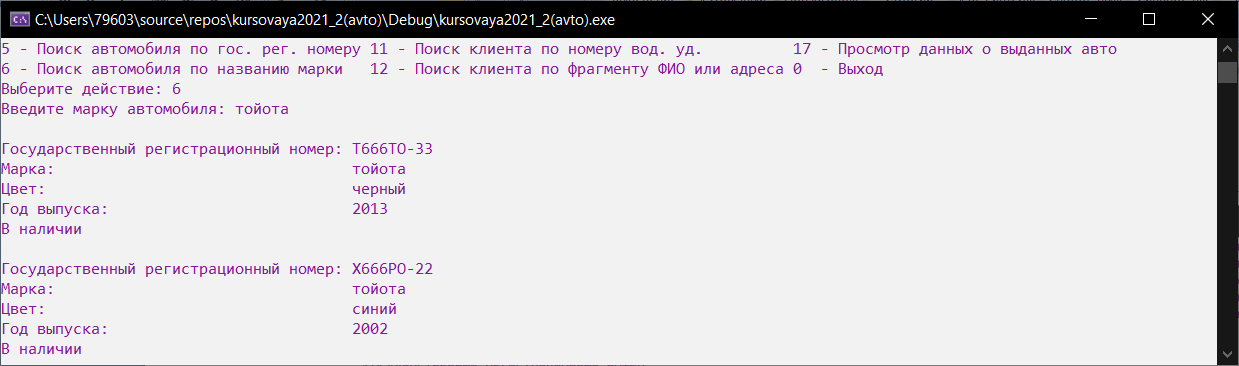




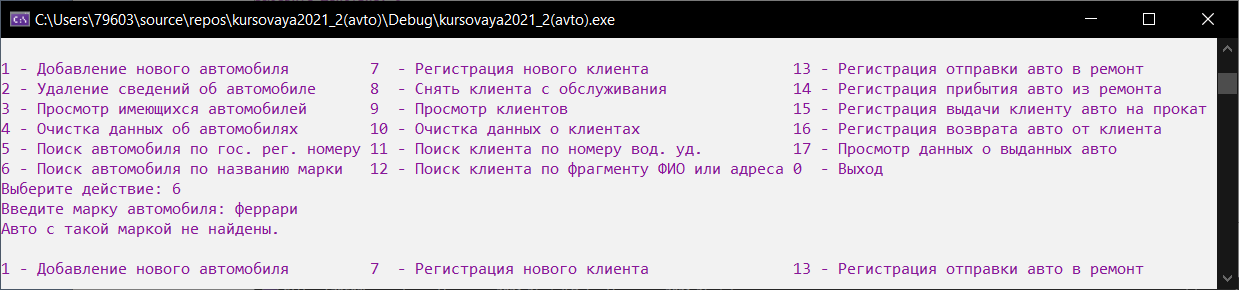
Проверка на ввод некорректных символов, ввод несуществующего номера:



*Поиск автомобиля по названию марки*

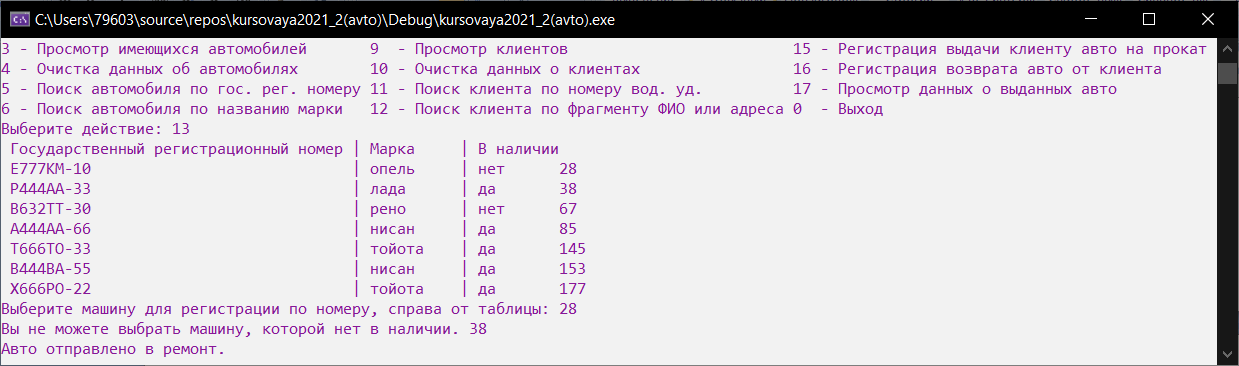


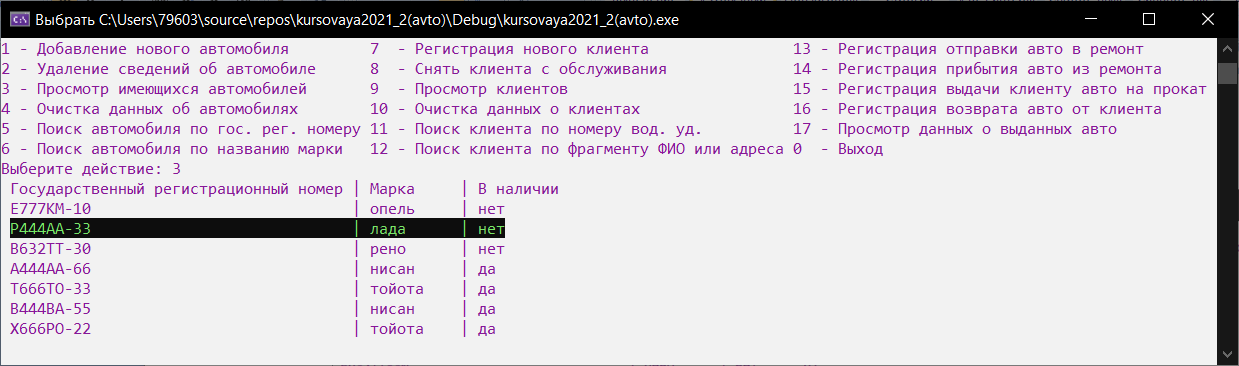
Ввод несуществующей марки:



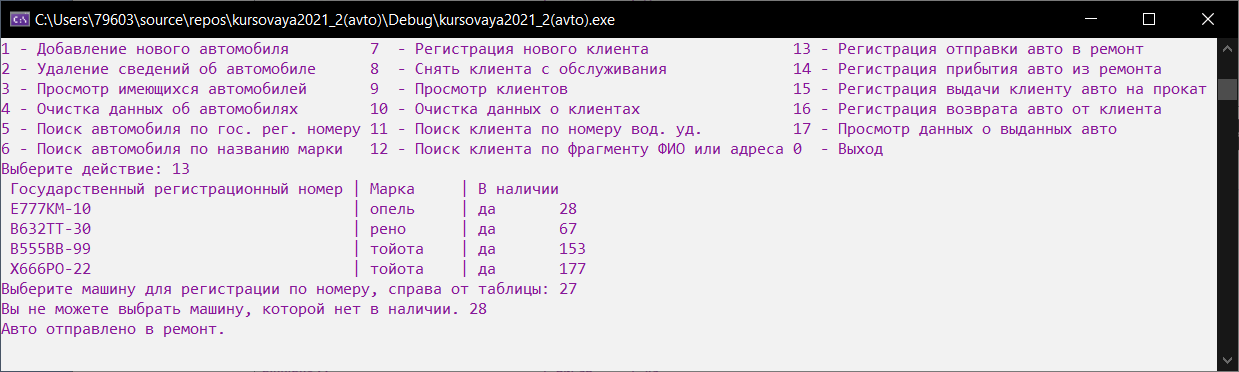
*Регистрация отправки авто в ремонт*

Выбор автомобиля, которого нет в наличии, затем корректный ввод:

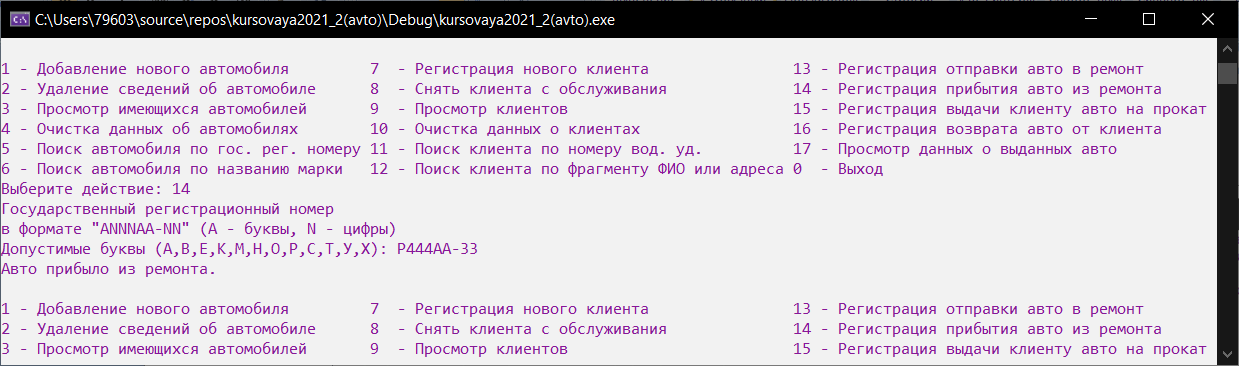


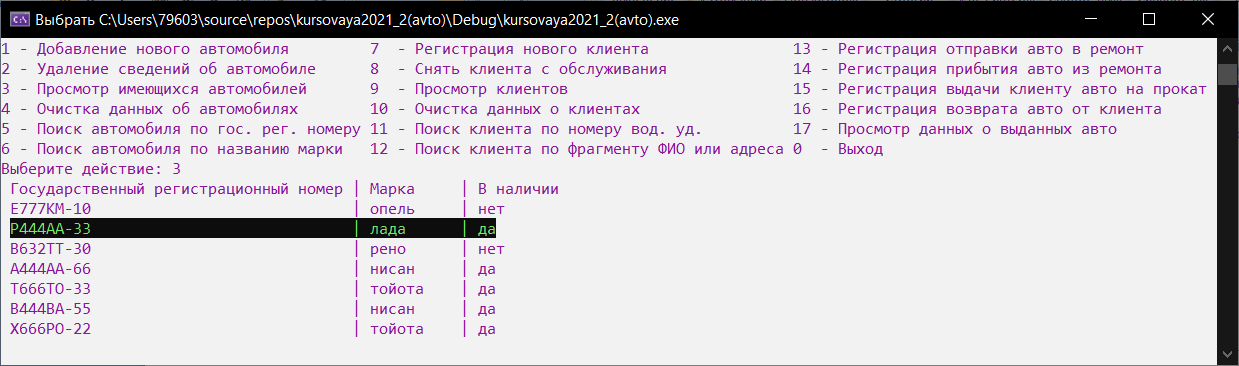


Выбор автомобиля, которого нет в таблице:

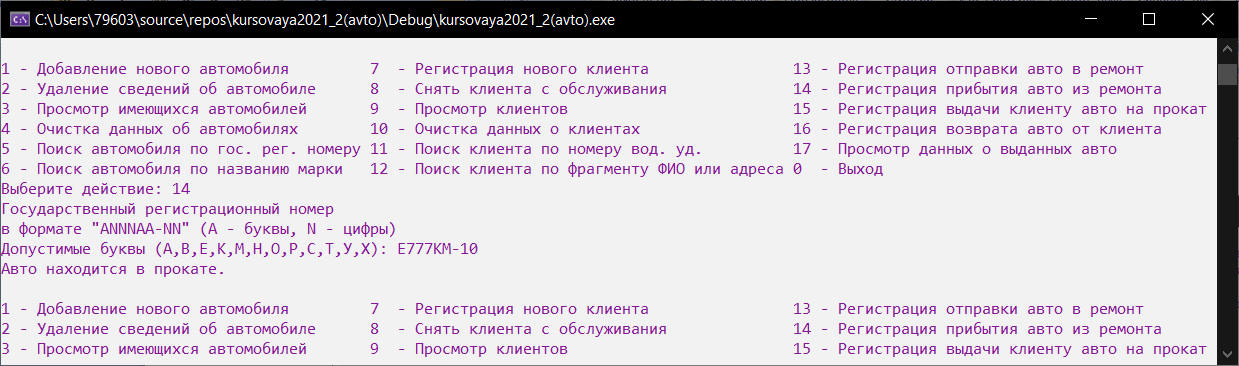


*Регистрация прибытия авто из ремонта*

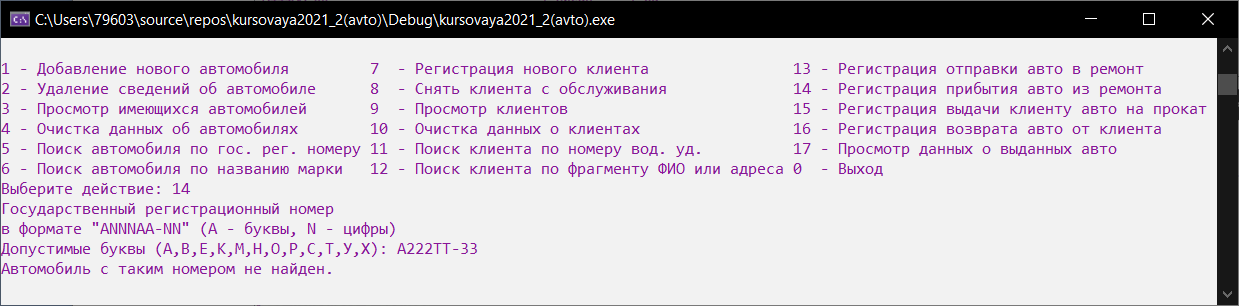
****

****

Выбор авто, которое находится в прокате:

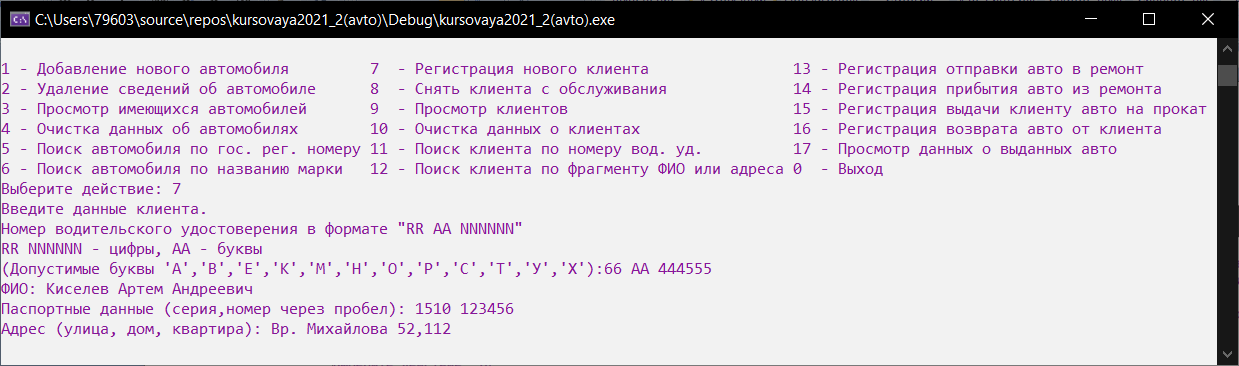
****

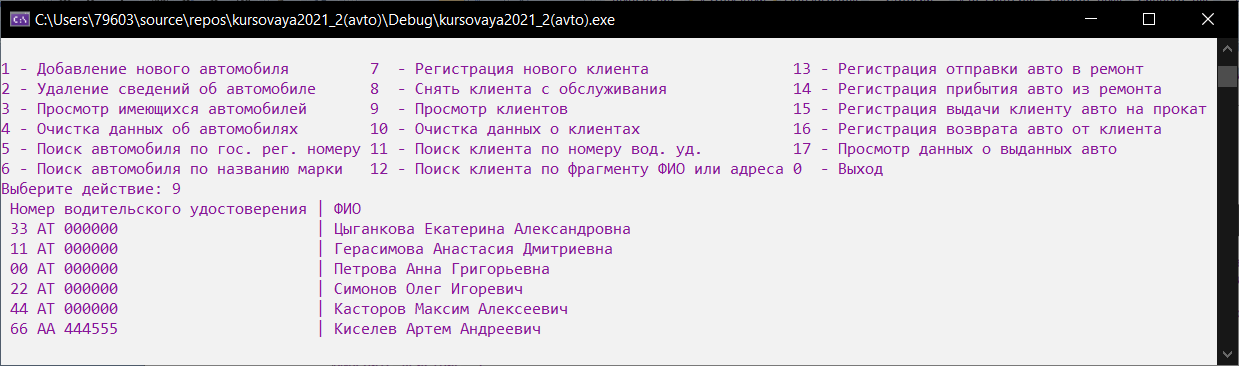
Ввод несуществующего номера:



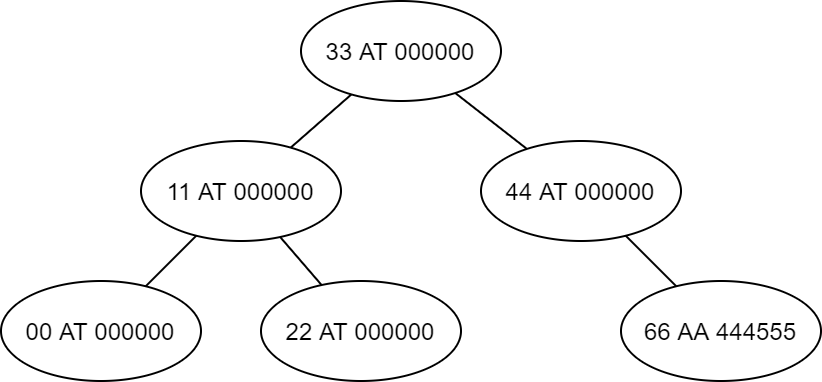
**Функции для работы с данными о клиентах:**

*Регистрация нового клиента*

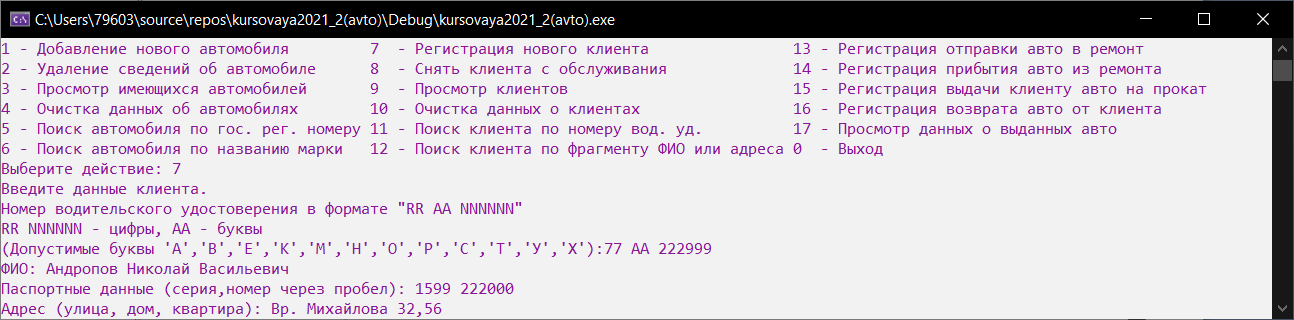


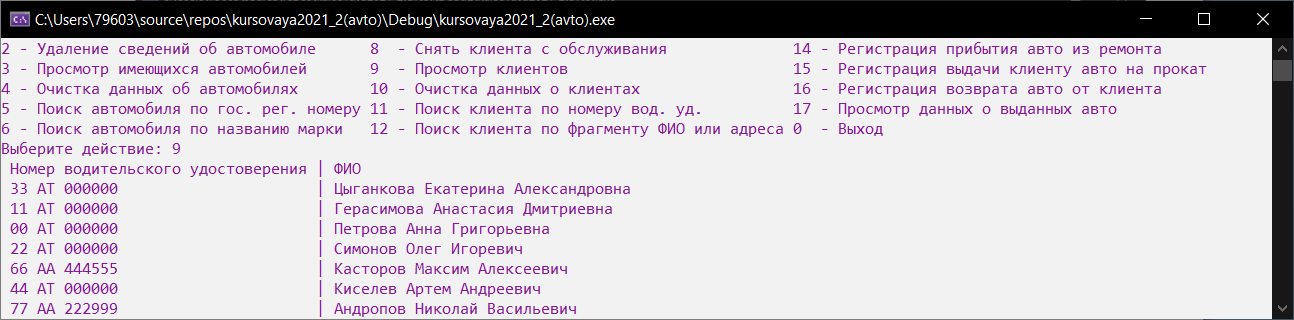


Данные о клиентах расположены в АВЛ-дереве поиска следующим образом:

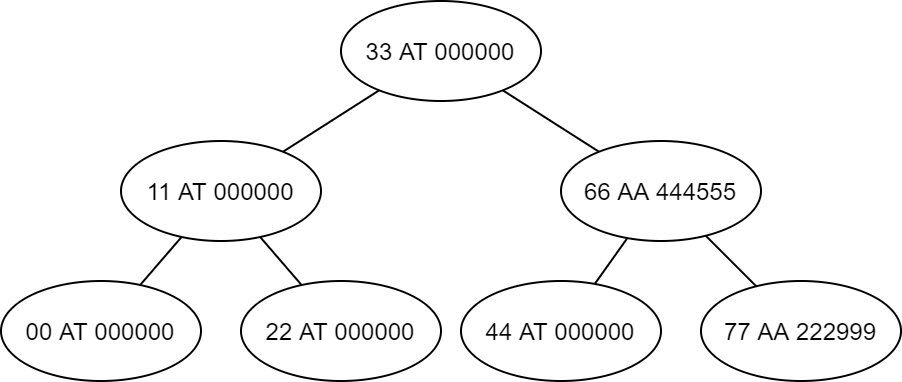


Проверим балансировку дерева, добавив еще одного клиента:

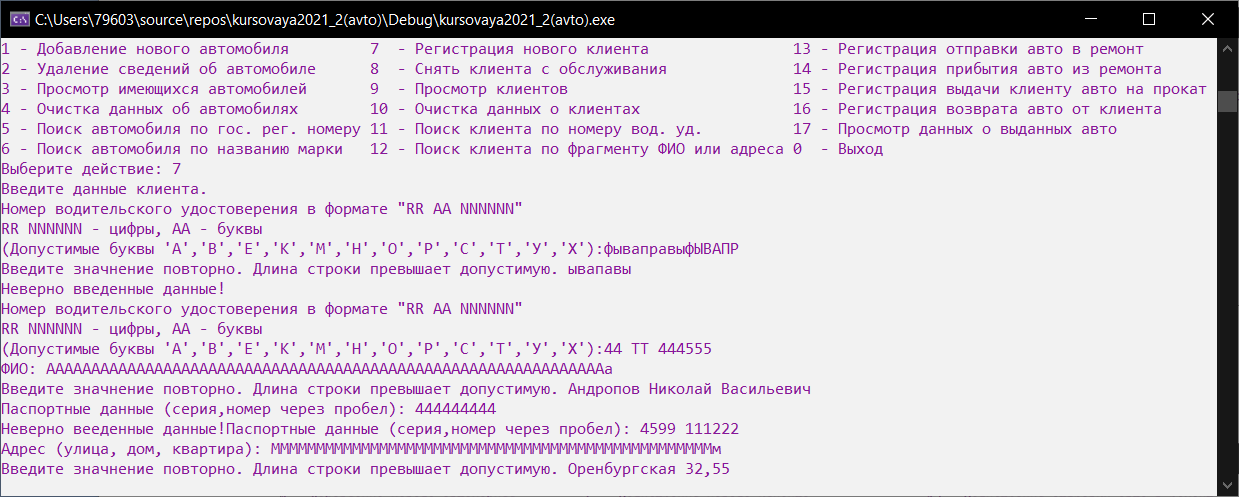




Вид дерева изменился:

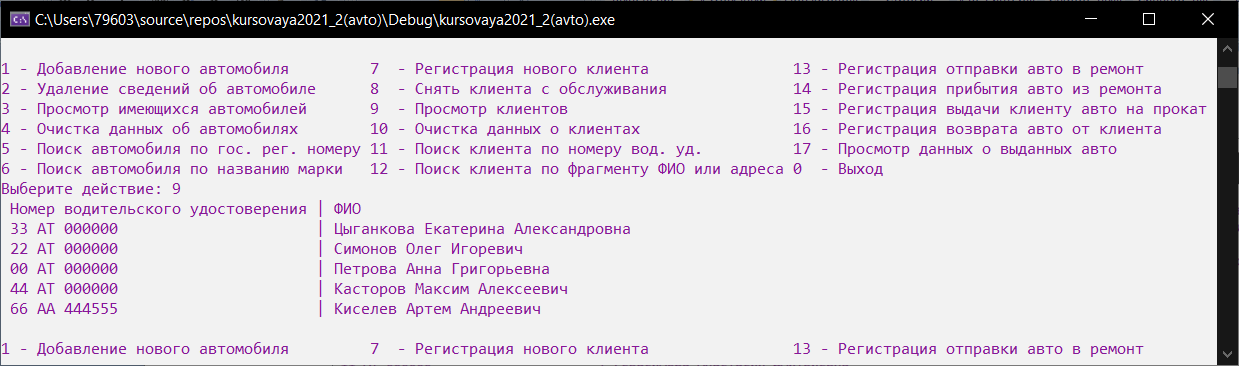


Проверка на ввод:

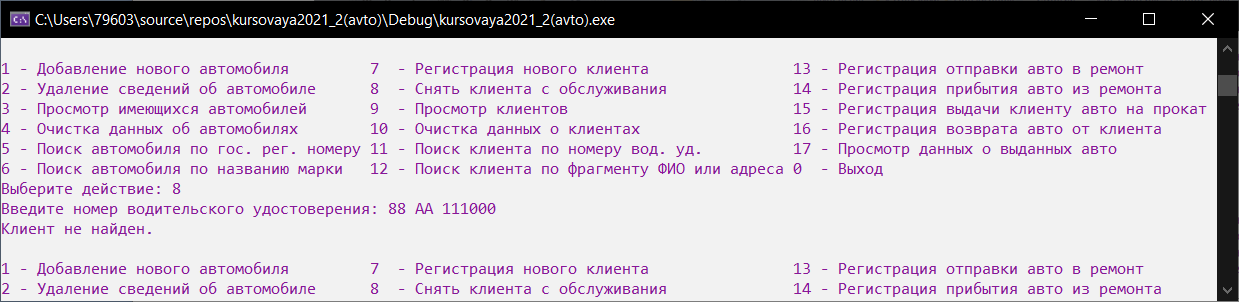


*Снять клиента с обслуживания*

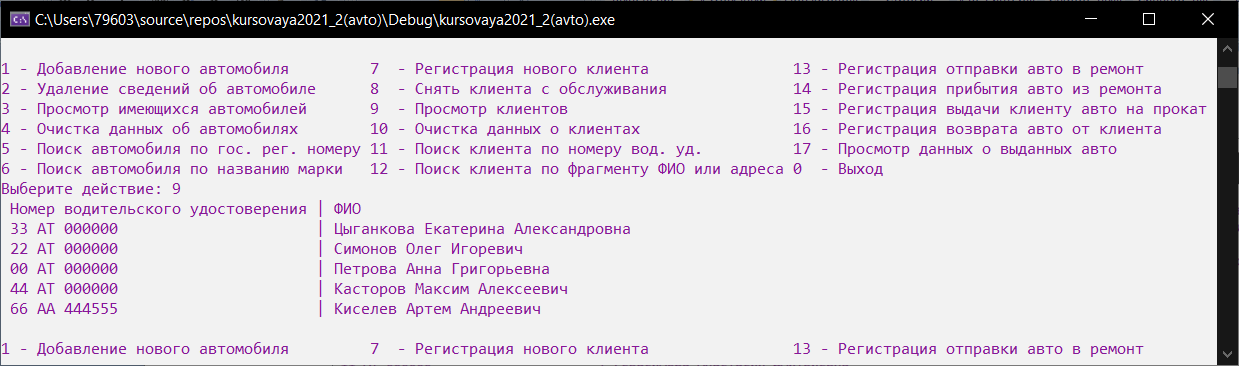




Ввод несуществующего номера:

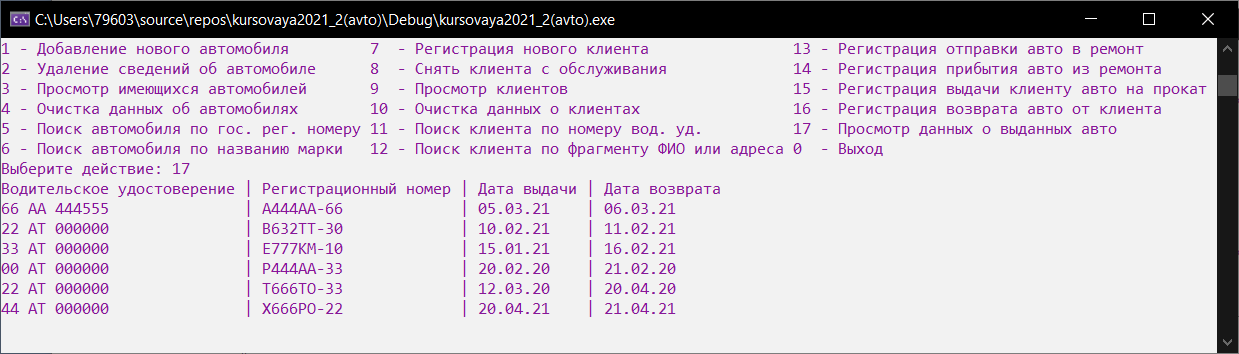


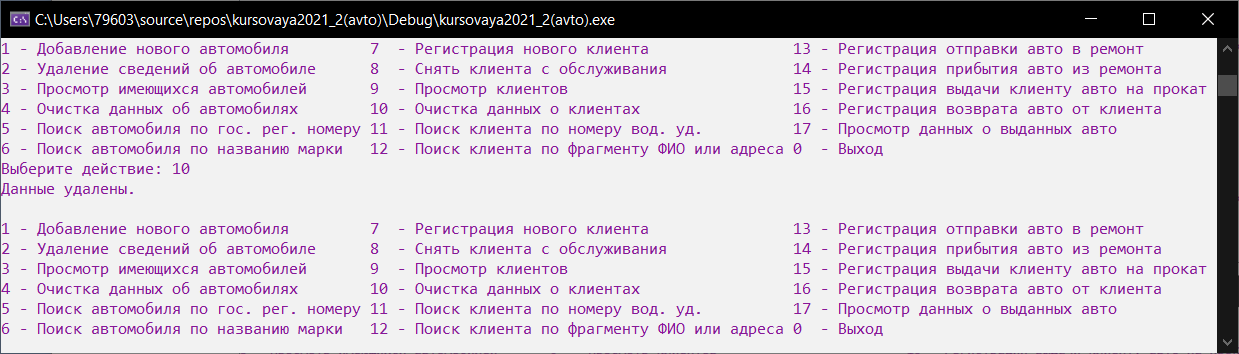
*Просмотр клиентов*

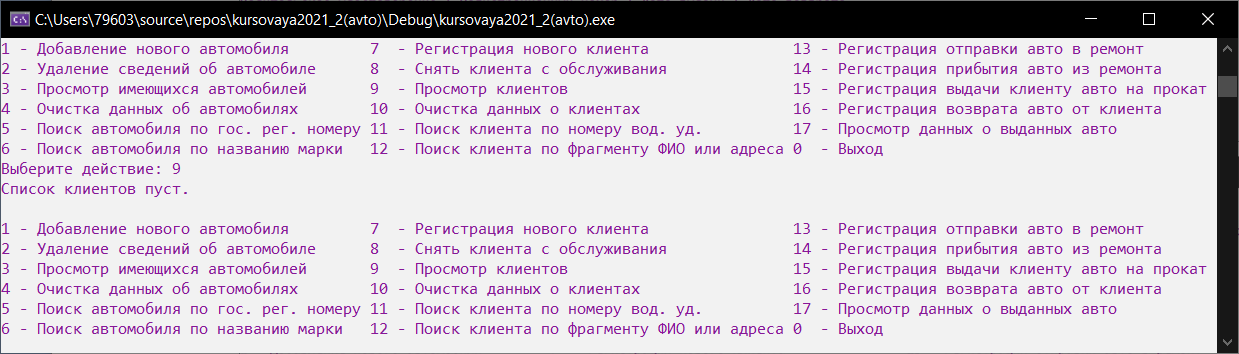


*Очистка данных о клиентах*

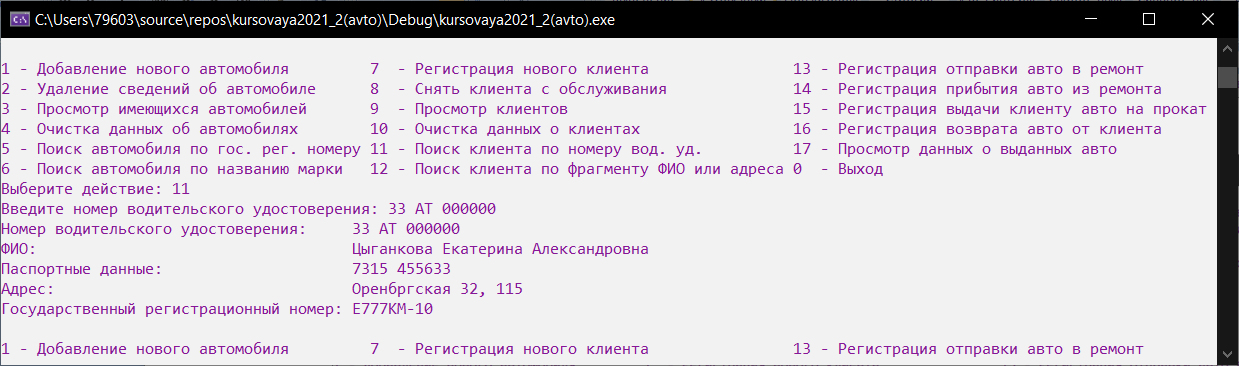


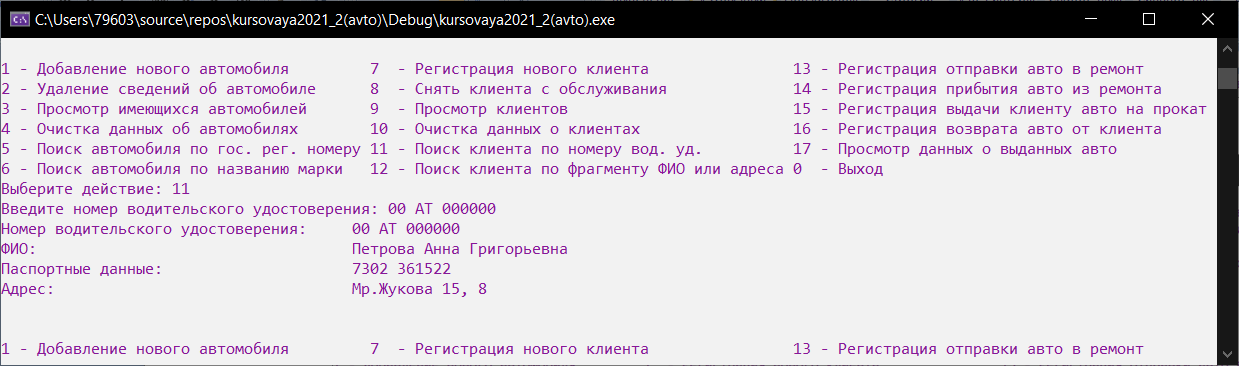




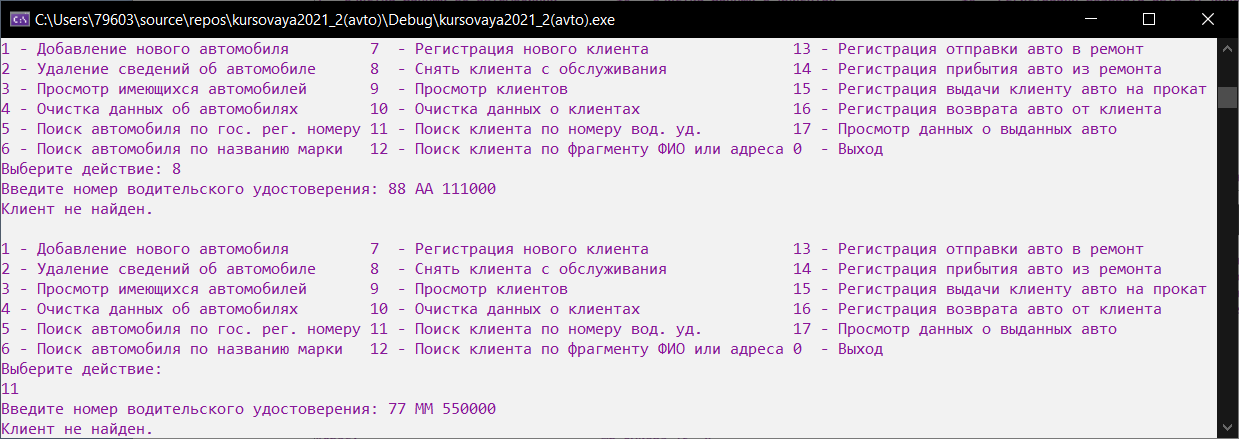


*Поиск клиента по номеру водительского удостоверения*

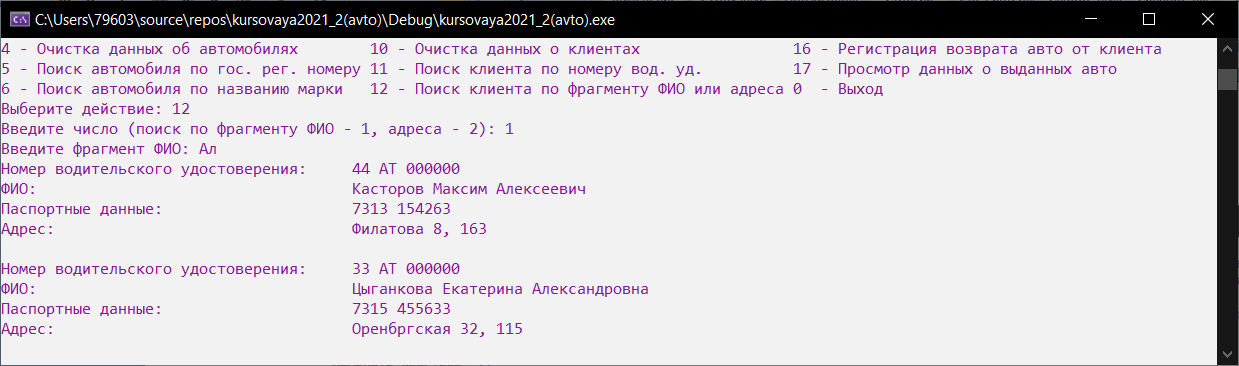


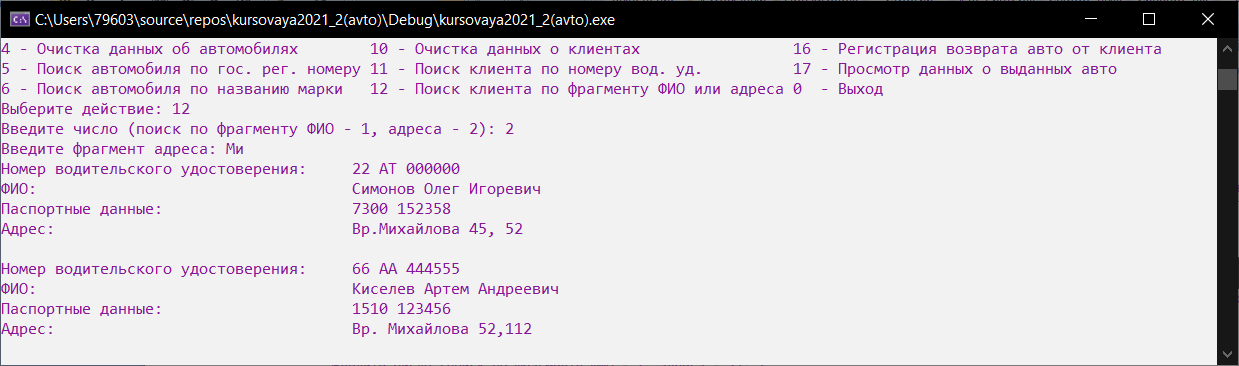


Ввод несуществующего номера:

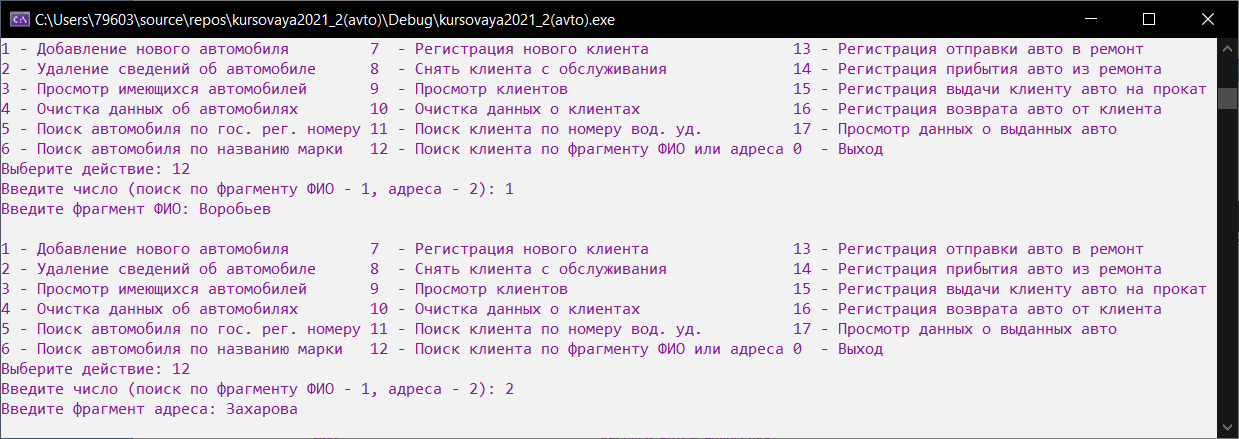


*Поиск клиента по фрагменту ФИО или адреса*

****

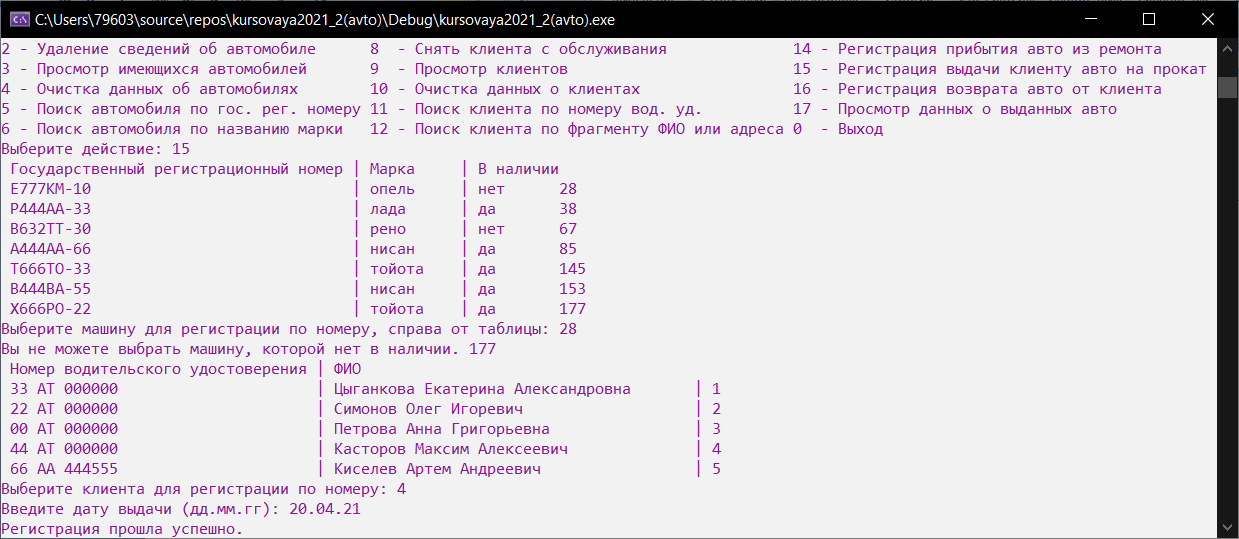
****

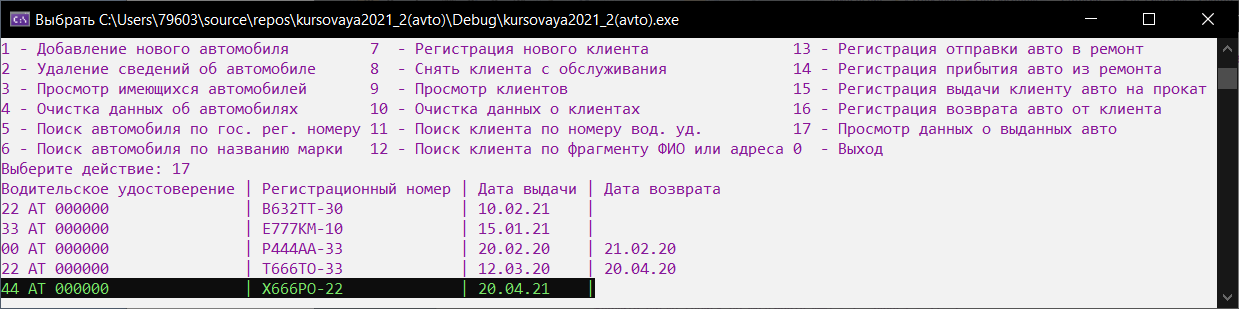
Ввод несуществующих фрагментов (результат: вывод главного меню):

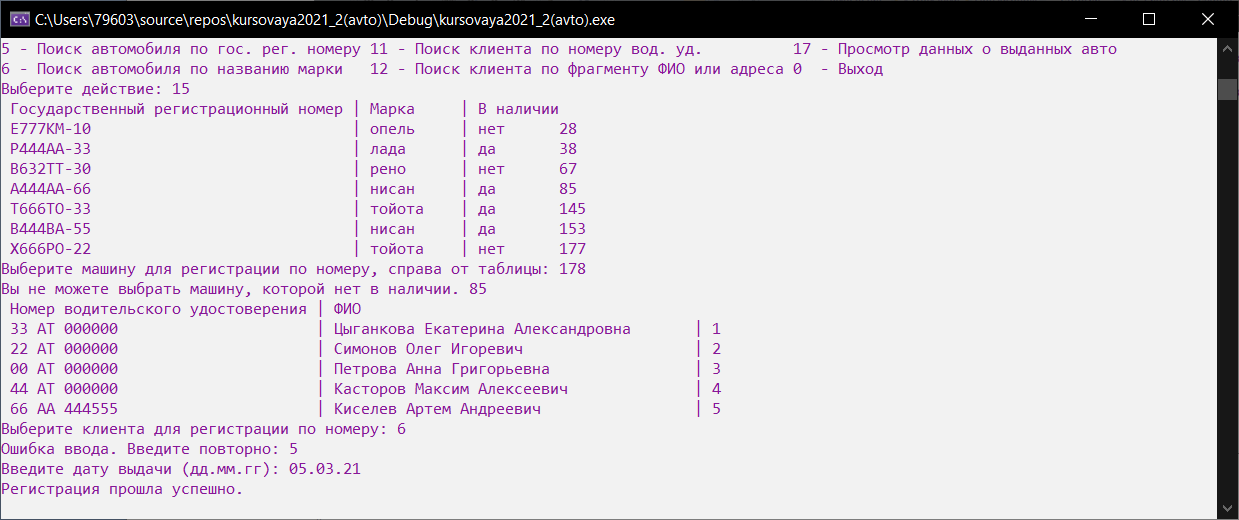


**Функции для работы с данными о выдаче на прокат:**

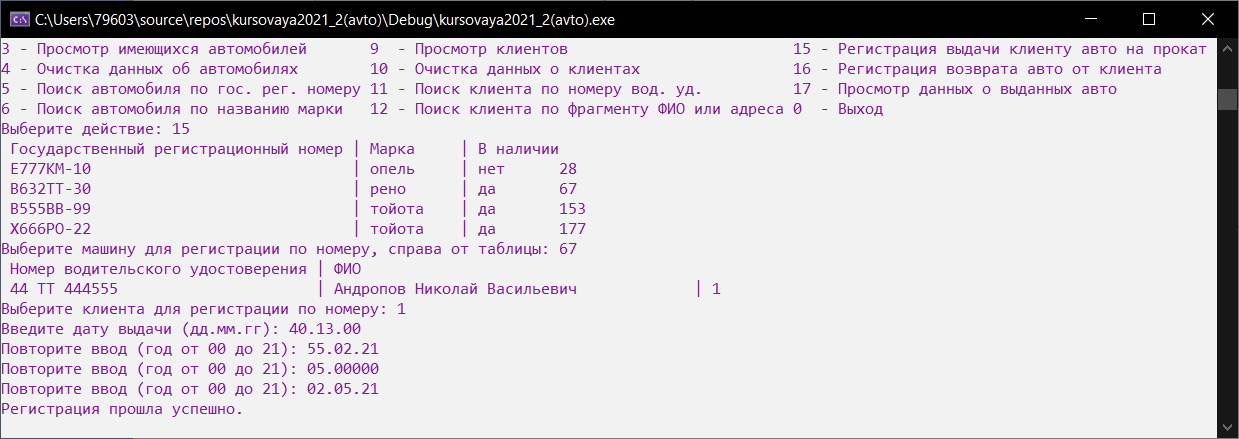
*Регистрация выдачи клиенту авто на прокат*



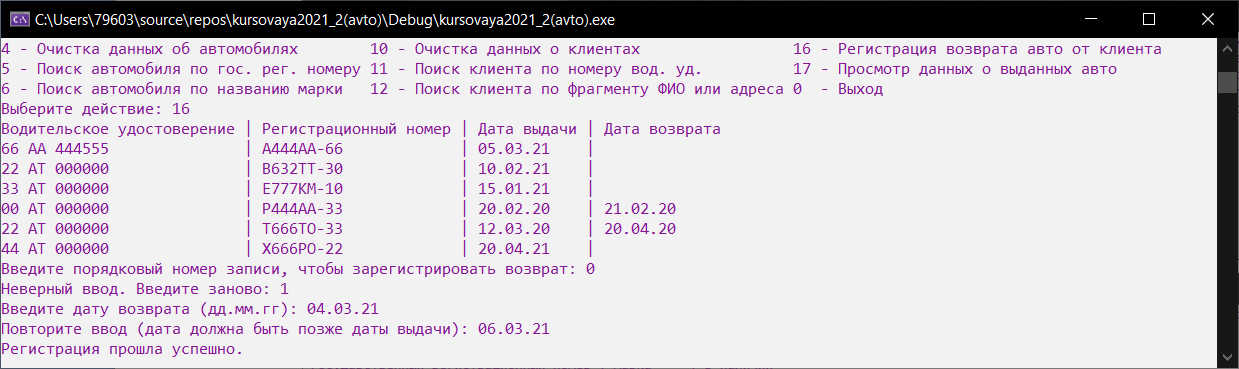




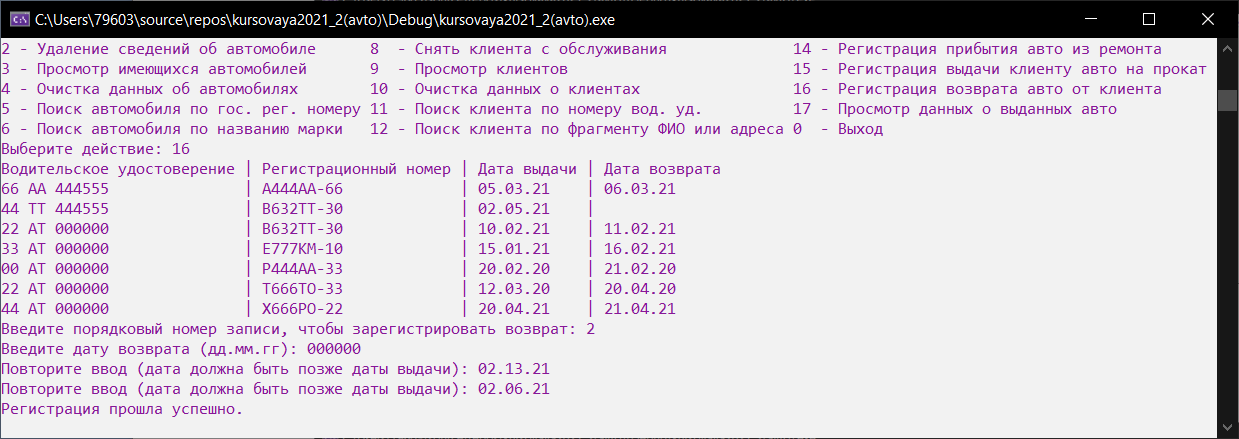
Проверка на корректный ввод даты:



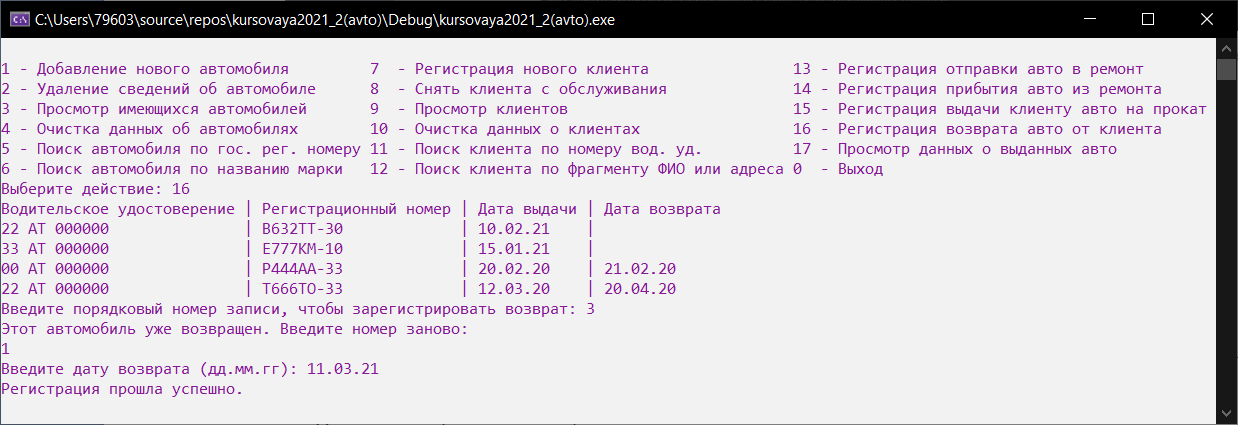
*Регистрация возврата авто от клиента*



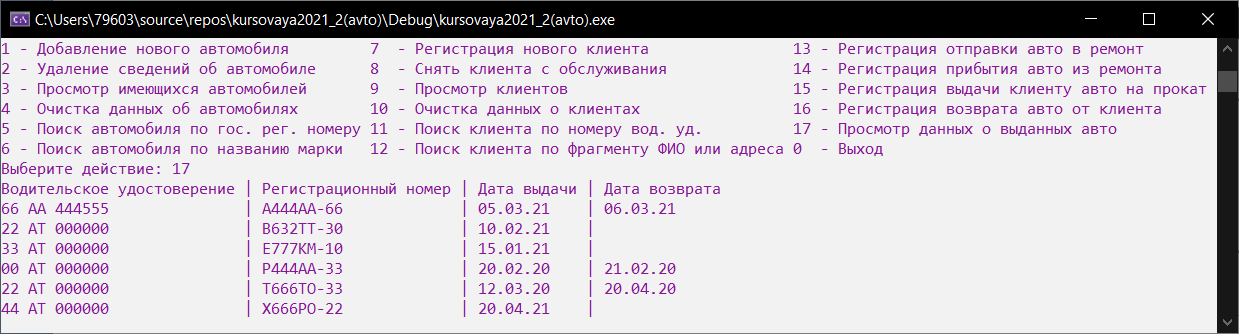
Проверка на корректный ввод даты:



Выбор возврата автомобиля, который уже был возвращен:



*Просмотр данных о выданных авто*



# 6. Заключение

В процессе выполнения курсового проекта была разработана программа «Обслуживание клиентов в бюро проката автомобилей» (приложение А). Мною были закреплены теоретические и практические знания, полученные на лекционных, лабораторных занятиях по курсу «Структуры и алгоритмы обработки данных», а также получены навыки разработки, отладки и тестирования программ на алгоритмических языках программирования. Программа позволяет управлять данными о клиентах, автомобилях и прокате авто. Приложение обладает необходимыми функциями для обслуживания клиентов в бюро проката автомобилей.

Достоинства программы:

* в программе осуществляются различные проверки ввода данных;
* данная программа проста и понятна в использовании;
* принимает как латиницу, так и кириллицу для ввода данных, кроме государственного регистрационного номера и номера водительского удостоверения.

Недостатки программы:

* для загрузки данных из файла информация о клиентах, автомобилях, прокате авто должна быть записана корректно;
* очищение данных о клиентах производится только в том случае, если ни у одного клиента нет автомобиля, выданного на прокат;
* отсутствует нумерация списка проката при регистрации возврата автомобиля от клиента.

# 7. Список использованной литературы

1. Матьяш В.А., Рогачев С.А. Алгоритмы и структуры данных: Учебное пособие / СПбГУАП. СПб., 2021.
2. Ключарев А.А., Матьяш В.А., Щекин С.В. Структуры и алгоритмы обработки данных: Учебное пособие / СПбГУАП. СПб., 2004.
3. Герберт Шилдт С++ для начинающих: Самоучитель / ЭКОМ, 2013.

# Приложение А.

// kursovaya2021\_2(avto).cpp : Этот файл содержит функцию "main". Здесь начинается и заканчивается выполнение программы.

//

#include <windows.h>

#include <fstream>

#include <iostream>

#include <vector>

#include <array>

#include"assert.h"

#include <algorithm>

#include <conio.h>

using namespace std;

const int N = 13;

const int M = 40;

const int P = 12;

const int A = 25;

const int NumOfSegments = 9;

int StepOfLinearSampling = 2;

int MaxCase = 200;

const int R = 10;

const int B = 10;

const int C = 10;

char letters[] = {'А','В','Е','К','М','Н','О','Р','С','Т','У','Х'};//12

//структура с информацией о клиенте

struct Customer{

char drivenumber[N];//номер вод удостоверения

char name[M];//фио

char passport[P];//паспорт

char address[A];//адрес

};

//структура элемента дерева

struct Tree{

Customer elem; //содержимое узла

int height; //высота узла

Tree\* Prev; //указатель на предка

Tree\* Left; //указатель на меньшего потомка

Tree\* Right; //указатель на большего потомка

};

//структура с информацией об авто

struct Auto{

char regnumber[R];//регистрационный номер

char brand[B];//марка

char color[C];//цвет

int year;//год выпуска

bool exist;//наличие

};

//структура ключа

struct Keys{

bool empty, deleted;

Auto car;

};

//структура вторичного ключа

struct SecondKey {

char brand[B];//марка

vector<unsigned int> Address;//инвертированные индексы

};

//структура проката

struct Rental{

char drivenumber[N];//номер вод уд-ия

char regnumber[R];//решистрационный номер авто

char dateissue[R];//дата выдачи

char dateret[R];//дата возврата

};

//структура списка

struct List {

Rental data;

List\* ptr;

};

//////РАБОТА С ДЕРЕВОМ //////////////////////////////////////////////////////////////////////////////

void LeftRotate(Tree\*& a);

void RightRotate(Tree\*& a);

void BigLeftRotate(Tree\*& a, int CurBalance);

void BigRightRotate(Tree\*& a, int CurBalance);

bool FindListElem(List\* begin, char str[], List\*& cur);

//Вычисление высоты дерева

int GetHeight(const Tree\* MainTree){

int l, r, h = 0;

if (MainTree != NULL) {

l = GetHeight(MainTree->Left);

r = GetHeight(MainTree->Right);

h = ((l > r) ? l : r) + 1;

}

return h;

}

//повороты балансировки

void LeftRotate(Tree\*& a){

Tree\* b;

char val\_a[N], val\_b[N];

b = a->Left;

strcpy\_s(val\_a, a->elem.drivenumber); strcpy\_s(val\_b, b->elem.drivenumber);

strcpy\_s(a->elem.drivenumber, val\_b);

strcpy\_s(b->elem.drivenumber, val\_a);

a->Left = b->Left; //поддерево R

if (a->Left)

a->Left->Prev = a;

b->Left = b->Right;//поддерево Q

if (b->Left)

b->Left->Prev = b;

b->Right = a->Right;//поддерево P

if (b->Right)

b->Right->Prev = b;

a->Right = b;

if (a->Right)

a->Right->Prev = a;

}

void BigLeftRotate(Tree\*& a, int CurBalance){

Tree\* b, \* c;

char val\_a[N], val\_b[N], val\_c[N];

b = a->Left;

c = b->Right;

strcpy\_s(val\_a, a->elem.drivenumber);

strcpy\_s(val\_b, b->elem.drivenumber);

strcpy\_s(val\_c, c->elem.drivenumber);

strcpy\_s(a->elem.drivenumber, val\_c);

strcpy\_s(c->elem.drivenumber, val\_a);

b->Right = c->Left; //поддерево R

if (b->Right)

b->Right->Prev = b;

c->Left = c->Right; //поддерево Q

if (c->Left)

c->Left->Prev = c;

c->Right = a->Right; //поддерево P

if (c->Right)

c->Right->Prev = c;

a->Right = c;

// -2 <= a ->Balance <= 2

if ((GetHeight(a->Right) - GetHeight(a->Left)) == -2) {

b = a->Left;

if ((GetHeight(b->Right) - GetHeight(b->Left)) == 1) {

BigLeftRotate(a, CurBalance);

CurBalance = -1;

}

else if ((GetHeight(b->Right) - GetHeight(b->Left)) == 0) {

LeftRotate(a);

CurBalance = 0;

}

else {// b ->Balance = 1

LeftRotate(a);

CurBalance = -1;

}

}

else {

if ((GetHeight(a->Right) - GetHeight(a->Left)) == 2) {

b = a->Right;

if ((GetHeight(b->Right) - GetHeight(b->Left)) == -1) {

BigRightRotate(a, CurBalance);

CurBalance = -1;

}

else if ((GetHeight(b->Right) - GetHeight(b->Left)) == 0) {

RightRotate(a);

CurBalance = 0;

}

else {// b->Balance = -1

RightRotate(a);

CurBalance = -1;

}

}

else { //-2 < a->Balance < 2, ничего делать не надо

CurBalance = 0;

}

}

}

void RightRotate(Tree\*& a){

//Балансировка двоичного дерева

//малое правое вращение поддерева с корнем a

Tree\* b;

char val\_a[N], val\_b[N];

b = a->Right;

strcpy\_s(val\_a, a->elem.drivenumber); strcpy\_s(val\_b, b->elem.drivenumber);

strcpy\_s(a->elem.drivenumber, val\_b);

strcpy\_s(b->elem.drivenumber, val\_a);

a->Right = b->Right; //поддерево R

if (a->Right)

a->Right->Prev = a;

b->Right = b->Left;//поддерево Q

if (b->Right)

b->Right->Prev = b;

b->Left = a->Left; //поддерево P

if (b->Left)

b->Left->Prev = b;

a->Left = b;

if (a->Left)

a->Left->Prev = a;

}

void BigRightRotate(Tree\*& a, int CurBalance){

Tree\* b, \* c;

char val\_a[N], val\_b[N], val\_c[N];

b = a->Right;

c = b->Left;

strcpy\_s(val\_a, a->elem.drivenumber);

strcpy\_s(val\_b, b->elem.drivenumber);

strcpy\_s(val\_c, c->elem.drivenumber);

strcpy\_s(a->elem.drivenumber, val\_c);

strcpy\_s(c->elem.drivenumber, val\_a);

b->Left = c->Right; //поддерево R

if (b->Left)

b->Left->Prev = b;

c->Right = c->Left; //поддерево Q

if (c->Right)

c->Right->Prev = c;

c->Left = a->Left; //поддерево P

if (c->Left)

c->Left->Prev = c;

a->Left = c;

// -2 <= a ->Balance <= 2

if ((GetHeight(a->Right) - GetHeight(a->Left)) == 2) {

b = a->Right;

if ((GetHeight(b->Right) - GetHeight(b->Left)) == -1) {

BigRightRotate(a, CurBalance);

CurBalance = -1;

}

else if ((GetHeight(b->Right) - GetHeight(b->Left)) == 0) {

RightRotate(a);

CurBalance = 0;

}

else {// b ->Balance = 1

RightRotate(a);

CurBalance = -1;

}

}

else {

if ((GetHeight(a->Right) - GetHeight(a->Left)) == -2) {

b = a->Left;

if ((GetHeight(b->Right) - GetHeight(b->Left)) == 1) {

BigLeftRotate(a, CurBalance);

CurBalance = -1;

}

else if ((GetHeight(b->Right) - GetHeight(b->Left)) == 0) {

LeftRotate(a);

CurBalance = 0;

}

else {// b->Balance = -1

LeftRotate(a);

CurBalance = -1;

}

}

else { //-2 < a->Balance < 2, ничего делать не надо

CurBalance = 0;

}

}

}

//Функция сбалансирования

void BalanceTree(Tree\*& Root, Tree\*& NewElem){

Tree\* MainTree = NewElem;

while (true)

{

MainTree->height = max(GetHeight(MainTree->Left), GetHeight(MainTree->Right)) + 1;

if (GetHeight(MainTree->Right) - GetHeight(MainTree->Left) == 2)

{

if (GetHeight(MainTree->Right->Right) - GetHeight(MainTree->Right->Left) == 0)

RightRotate(MainTree);

else if (GetHeight(MainTree->Right->Right) - GetHeight(MainTree->Right->Left) == 1)

RightRotate(MainTree);

else

BigRightRotate(MainTree, ((GetHeight(MainTree->Right) - GetHeight(MainTree->Left))));

}

else if (GetHeight(MainTree->Right) - GetHeight(MainTree->Left) == -2)

{

if (GetHeight(MainTree->Left->Right) - GetHeight(MainTree->Left->Left) == 0)

LeftRotate(MainTree);

else if (GetHeight(MainTree->Left->Right) - GetHeight(MainTree->Left->Left) == -1)

LeftRotate(MainTree);

else

BigLeftRotate(MainTree, ((GetHeight(MainTree->Right) - GetHeight(MainTree->Left))));

}

if (MainTree->Prev)

MainTree = MainTree->Prev;

else break;

}

Root = MainTree;

}

//Добавление элемента в дерево

void AddTreeElem(Tree\*& Root, Tree\* NewElem, int& numOfElem) {

Tree\* MainTree = Root;

numOfElem++;

if (!Root) {

Root = new Tree;

strcpy\_s(Root->elem.drivenumber, NewElem->elem.drivenumber);

strcpy\_s(Root->elem.name, NewElem->elem.name);

strcpy\_s(Root->elem.passport, NewElem->elem.passport);

strcpy\_s(Root->elem.address, NewElem->elem.address);

Root->height = 1;

Root->Prev = NULL;

Root->Left = NULL;

Root->Right = NULL;

MainTree = Root;

}

else {

while (true){

if (strcmp(NewElem->elem.drivenumber, MainTree->elem.drivenumber) < 0){

if (MainTree->Left)

MainTree = MainTree->Left;

else{

MainTree->Left = NewElem;

NewElem->Prev = MainTree;

break;

}

}

else {

if (MainTree->Right)

MainTree = MainTree->Right;

else {

MainTree->Right = NewElem;

NewElem->Prev = MainTree;

break;

}

}

}

BalanceTree(Root, MainTree);

}

}

//Функция удаления элемента

void DeleteTreeElem(Tree\*& Root, char drivenumber[], int& numOfElem)

{

Tree\* MainTree = Root;

Tree\* TmpElem = NULL;

Tree\* OneMoreTmp = NULL;

int counter = 0;

while (true)

{

if (MainTree)

{

if (!strcmp(drivenumber, MainTree->elem.drivenumber))

{

numOfElem--;

if (MainTree->Prev || GetHeight(MainTree) > 1)

{

OneMoreTmp = MainTree;

if (MainTree->Left || MainTree->Right)

{

if (MainTree->Right)

MainTree = MainTree->Right;

TmpElem = OneMoreTmp->Prev;

while (MainTree->Left)

MainTree = MainTree->Left;

if (MainTree->Prev)

{

if (MainTree->Prev->Right != MainTree)

MainTree->Prev->Left = NULL;

}

if (TmpElem)

{

if (TmpElem->Right)

{

if (TmpElem->Right == OneMoreTmp)

TmpElem->Right = MainTree;

else

TmpElem->Left = MainTree;

}

else

TmpElem->Left = MainTree;

}

if (MainTree->Prev != OneMoreTmp)

TmpElem = MainTree->Prev;

else

TmpElem = MainTree;

MainTree->Prev = OneMoreTmp->Prev;

if (OneMoreTmp->Right != MainTree)

{

MainTree->Right = OneMoreTmp->Right;

if (MainTree->Right)

MainTree->Right->Prev = MainTree;

}

;

if (OneMoreTmp->Left != MainTree)

{

MainTree->Left = OneMoreTmp->Left;

if (MainTree->Left)

MainTree->Left->Prev = MainTree;

}

else

MainTree->Left = NULL;

BalanceTree(Root, TmpElem);

}

else

{

TmpElem = MainTree->Prev;

if (TmpElem)

{

if (TmpElem->Right)

{

if (TmpElem->Right == MainTree)

TmpElem->Right = NULL;

else

TmpElem->Left = NULL;

}

else

TmpElem->Left = NULL;

}

BalanceTree(Root, TmpElem);

}

cout << "Клиент удален." << endl;

return;

}

else

{

Root = NULL;

cout << "Список пуст." << endl;

return;

}

}

else if (strcmp(drivenumber, MainTree->elem.drivenumber) > 0)

MainTree = MainTree->Right;

else

MainTree = MainTree->Left;

}

else

{

cout << "Клиент не найден." << endl;

return;

}

}

}

//Вывод на экран

void ShowTree(Tree\*& tree, bool i, int& n){ // Обход в прямом порядке

//Посетили узел

cout << ' ' << tree->elem.drivenumber << " | " << tree->elem.name;

if (i) {

for (int s = 0; s < (M - strlen(tree->elem.name)); s++)

cout << ' ';

cout <<"| "<< n;

}

if (tree->Left) {

n++;

cout << endl;

ShowTree(tree->Left,i,n); //Обошли левое поддерево

}

if (tree->Right) {

n++;

cout << endl;

ShowTree(tree->Right,i,n); //Обошли правое поддерево

}

}

//Функция поиска по номеру вод уд

Customer\* FindTreeElem(char drivenumber[], Tree\*& MainTree){

Tree\* Current;

Customer\* temp = NULL;

if (MainTree) {// Дерево не пустое

Current = MainTree;

int i = strcmp(Current->elem.drivenumber, drivenumber);

if (i == 0) // Вершина найдена

temp = &Current->elem;

else if (i > 0) // Ищем в левом поддереве

temp = FindTreeElem(drivenumber, Current->Left);

else // Ищем в правом поддереве

temp = FindTreeElem(drivenumber, Current->Right);

}

return temp;

}

//Вывод на экран одного элемента

void ShowElem(Customer\* Elem){

cout << "Номер водительского удостоверения: " << Elem->drivenumber << endl;

cout << "ФИО: " << Elem->name << endl;

cout << "Паспортные данные: " << Elem->passport << endl;

cout << "Адрес: " << Elem->address << endl;

}

//Вывод на экран данных о клиенте + авто

void ShowElemAuto\_Cust(Tree\* MainTree,char drivenumber[], List\* begin, List\* cur){

Customer\* c = FindTreeElem(drivenumber, MainTree);

ShowElem(c);

if (FindListElem(begin, drivenumber, cur))

cout << "Государственный регистрационный номер: "<<cur->data.regnumber;

cout << endl;

}

//Функция поиска по порядковому номеру

Customer\* FindCustTree(Tree\*& MainTree,int& n, int& i) {

Tree\* Current;

Customer\* temp =NULL;

if (MainTree) {// Дерево не пустое

Current = MainTree;

// Вершина найдена

if (i == n)

temp = &Current->elem;

if (Current->Left && i<n) { // Ищем в левом поддереве

i++;

temp = FindCustTree(Current->Left, n, i);

}

if (Current->Right && i<n){ // Ищем в правом поддереве

i++;

temp = FindCustTree(Current->Right, n, i);

}

}

return temp;

}

//проверка на уникальность паспорта

bool PasUnic(char passport[], Tree\*& MainTree){

Tree\* Current;

if (MainTree != NULL){

Current = MainTree;

PasUnic(passport, Current->Left);

PasUnic(passport, Current->Right);

int i = strcmp(passport, Current->elem.passport);

if (i == 0)

return 0;

}

return 1;

}

//Проверка длины строки

void CheckLength(char str[], const int n){

while (cin.fail()){

cin.clear();

cin.ignore(cin.rdbuf()->in\_avail());

cout << "Введите значнение повторно. Длина строки превышает допустимую. ";

cin.getline(str, n);

}

}

//Ввод данных о клиенте

bool InputDataCust(Tree\*& Elem, Customer& x, Tree\*& MyTree){

unsigned int count;

cout << "Введите данные клиента.\n";

int l = 0;

while (l == 0) {

l = 1;

cout << "Номер водительского удостоверения в формате \"RR AA NNNNNN\"" << endl <<

"RR NNNNNN - цифры, АА - буквы" << endl <<

"(Допустимые буквы 'А','В','Е','К','М','Н','О','Р','С','Т','У','Х'):";

cin.ignore(cin.rdbuf()->in\_avail());

cin.getline(x.drivenumber, N);

CheckLength(x.drivenumber, N);

count = 0;

for (int i = 0; i < 12; i++) {

for (int j = 3; j < 5; j++) {

if (x.drivenumber[j] == letters[i])

count++;

}

}

if (x.drivenumber[0] < 48 || x.drivenumber[0]>57 || x.drivenumber[1] < 48 || x.drivenumber[1]>57 || x.drivenumber[6] < 48 || x.drivenumber[7]>57 || x.drivenumber[7] < 48 || x.drivenumber[7]>57 || x.drivenumber[8] < 48 || x.drivenumber[8]>57 || x.drivenumber[9] < 48 || x.drivenumber[9]>57 || x.drivenumber[10] < 48 || x.drivenumber[10]>57 || x.drivenumber[11] < 48 || x.drivenumber[11]>57 || count != 2) {

cout << "Неверно введенные данные!" << endl;

l = 0;

}

}

if (FindTreeElem(x.drivenumber, MyTree)) {

cout << "Клиент с таким номером уже существует.\n";

return 1;

}

strcpy\_s(Elem->elem.drivenumber, x.drivenumber);

cout << "ФИО: ";

cin.ignore(cin.rdbuf()->in\_avail());

cin.getline(x.name, M);

CheckLength(x.name, M);

strcpy\_s(Elem->elem.name, x.name);

l = 0;

while (l == 0) {

l = 1;

cout << "Паспортные данные (серия,номер через пробел): ";

cin.ignore(cin.rdbuf()->in\_avail());

cin.getline(x.passport, P);

CheckLength(x.passport, P);

if (x.passport[0] < 48 || x.passport[1]>57 || x.passport[2] < 48 || x.passport[2]>57 || x.passport[3] < 48 || x.passport[3]>57 || x.passport[4] != 32 || x.passport[5] < 48 || x.passport[5]>57 || x.passport[6] < 48 || x.passport[6]>57 || x.passport[7] < 48 || x.passport[7]>57 || x.passport[8] < 48 || x.passport[8]>57 || x.passport[9] < 48 || x.passport[9]>57 || x.passport[10] < 48 || x.passport[10]>57) {

cout << "Неверно вееденные данные!";

l = 0;

}

}

//проверка на уникальные паспортные данные

if (!PasUnic(x.passport, MyTree)) {

cout << "Клиент с такими паспортными данными уже существует.\n";

return 1;

}

strcpy\_s(Elem->elem.passport, x.passport);

cout << "Адрес (улица, дом, квартира): ";

cin.ignore(cin.rdbuf()->in\_avail());

cin.getline(x.address, A);

CheckLength(x.address, A);

strcpy\_s(Elem->elem.address, x.address);

return 0;

}

//поиск в тексте по фио

bool TextSearchFIO(char str[], Tree\*& MainTree){

Tree\* Cur; bool find = false;

if (MainTree == NULL)

return find;

else {

Cur = MainTree;

TextSearchFIO(str, Cur->Left);

TextSearchFIO(str, Cur->Right);

int strl, substrl;

strl = strlen(Cur->elem.name); substrl = strlen(str); //длина строки и подстроки

if (strl != 0 && substrl != 0) {

for (int i = 0; i < strl - substrl + 1; i++) { //Проход по строке

for (int j = 0; j < substrl; j++) { //Проход по подстроке

if (str[j] != Cur->elem.name[i + j])

{

break;

}

else if (j == substrl - 1) /\*Если это последний символ в подстроке, то строка найдена\*/

{

ShowElem(&Cur->elem); cout << endl;

find = true;

break;

}

}

}

}

return find;

}

}

//поиск в тексте по адресу

bool TextSearchAdr(char str[], Tree\*& MainTree){

Tree\* Cur; bool find = false;

if (MainTree == NULL)

return find;

else {

Cur = MainTree;

TextSearchAdr(str, Cur->Left);

TextSearchAdr(str, Cur->Right);

int strl, substrl;

strl = strlen(Cur->elem.address); substrl = strlen(str); //длина строки и подстроки

if (strl != 0 && substrl != 0) {

for (int i = 0; i < strl - substrl + 1; i++) { //Проход по строке

for (int j = 0; j < substrl; j++) { //Проход по подстроке

if (str[j] != Cur->elem.address[i + j])

{

break;

}

else if (j == substrl - 1) /\*Если это последний символ в подстроке, то строка найдена\*/

{

ShowElem(&Cur->elem); cout << endl;

find = true;

break;

}

}

}

}

return find;

}

}

//очистка дерева

void Clean(Tree\*& MainTree) {

if (MainTree != NULL) {

Clean(MainTree->Left);

Clean(MainTree->Right);

delete MainTree;

MainTree = NULL;

}

}

//////РАБОТА С ХЕШ-ТАБЛИЦЕЙ //////////////////////////////////////////////////////////////////////////////

//хеш-функция

int H(Auto x){

int hashNum = 0;

for (int j = 0; j < R; j++) {

if (j)

hashNum = hashNum + (j \* 3) \* (int)x.regnumber[j] \* (int)x.regnumber[j];

else hashNum = (int)x.regnumber[j] \* (int)x.regnumber[j];

}

return(hashNum);

}

//очищение таблицы

void Clear\_HTableClose(Keys MassiveOfSegments[]){

unsigned int IndexSeg;

for (IndexSeg = 0; IndexSeg < NumOfSegments; IndexSeg++){

MassiveOfSegments[IndexSeg].empty = true;

MassiveOfSegments[IndexSeg].deleted = false;

for (int j = 0; j < R; j++)

MassiveOfSegments[IndexSeg].car.regnumber[j] = 0;

for (int j = 0; j < B; j++)

MassiveOfSegments[IndexSeg].car.brand[j]=0;

for (int j = 0; j < C; j++)

MassiveOfSegments[IndexSeg].car.color[j]=0;

MassiveOfSegments[IndexSeg].car.exist = false;

MassiveOfSegments[IndexSeg].car.year = 0;

}

}

//поиск по рег номеру

bool Find\_HTableClose(Auto x, Keys MassiveOfSegments[], unsigned int& Index){

int i = 0;

int flag;

do {

flag = 0;

Index = ((H(x) + StepOfLinearSampling \* i) % NumOfSegments + (H(x) + StepOfLinearSampling \* i) / NumOfSegments) % NumOfSegments;

i++;

for (int j = 0; j < R; j++){

if (MassiveOfSegments[Index].car.regnumber[j] == x.regnumber[j])

flag++;

}

if (flag == R)

return 1;

} while (MassiveOfSegments[Index].empty == false && i <= MaxCase);//пока не нашел совпадение

//не нашли пустое место (дальше уже нет)

//или долго ищем

return 0;

}

//добавление элемента

bool Add\_HTableClose(Auto x, Keys MassiveOfSegments[], unsigned int& Index, vector<SecondKey>& InvInd){

int i, flag(0);

if (!Find\_HTableClose(x, MassiveOfSegments, Index)){//если элемент не найден

i = 0;

do {

Index = ((H(x) + StepOfLinearSampling \* i) % NumOfSegments + (H(x) + StepOfLinearSampling \* i) / NumOfSegments) % NumOfSegments;

i++;

if (MassiveOfSegments[Index].empty || MassiveOfSegments[Index].deleted)

break;

} while (i <= MaxCase);

if (MassiveOfSegments[Index].empty || MassiveOfSegments[Index].deleted)

{ //заполнение структуры авто

for (int j = 0; j < R; j++)

MassiveOfSegments[Index].car.regnumber[j] = x.regnumber[j];

strcpy\_s(MassiveOfSegments[Index].car.brand, x.brand);

strcpy\_s(MassiveOfSegments[Index].car.color, x.color);

MassiveOfSegments[Index].car.year= x.year;

MassiveOfSegments[Index].car.exist = true;

//добавление в список вторичных ключей марки автомобиля и индекса

bool ad = 0;

if (!InvInd.empty()){//список не пустой

for (int m = 0;m<InvInd.size();m++) {//ищем совпадения

if (strcmp(InvInd.at(m).brand, x.brand) == 0) {

InvInd.at(m).Address.push\_back(Index);//добавили адрес

sort(InvInd.at(m).Address.begin(), InvInd.at(m).Address.end());//сортировка по возрастанию адресов

ad = 1;

}

}

}

if(!ad || InvInd.empty()){//если не нашли или список пуст

SecondKey y;

strcpy\_s(y.brand, x.brand);

y.Address.push\_back(Index);

InvInd.push\_back(y);

//сортировка по возрастанию адресов

sort(InvInd[InvInd.size() - 1].Address.begin(), InvInd[InvInd.size() - 1].Address.end());

}

//заполнение структуры ключа

MassiveOfSegments[Index].empty = false;

MassiveOfSegments[Index].deleted = false;

return 1;//добавлен

}

else

return 0;

}

else

return 0;

}

//удаление элемента

bool Del\_HTableClose(Auto x, Keys MassiveOfSegments[], vector<SecondKey>& InvInd){

unsigned int Index;

if (Find\_HTableClose(x, MassiveOfSegments, Index) && MassiveOfSegments[Index].car.exist){//поиск записи

MassiveOfSegments[Index].deleted = true;

for (int j = 0; j < R; j++)

MassiveOfSegments[Index].car.regnumber[j] = 0;

for (int j = 0; j < B; j++)

MassiveOfSegments[Index].car.brand[j] =0;

for (int j = 0; j < C; j++)

MassiveOfSegments[Index].car.color[j]= 0;

MassiveOfSegments[Index].car.exist = false;

MassiveOfSegments[Index].car.year = 0;

//корректировка структуры со вторичными ключами

for (int m = 0;m<InvInd.size();m++) {

if (strcmp(InvInd.at(m).brand, x.brand) == 0) {//ищем совпадения по марке

for (int n = 0; n < InvInd.at(m).Address.size(); n++) {

if (InvInd.at(m).Address.at(n) == Index)//ищем совпадения по адресу

InvInd.at(m).Address.erase(InvInd.at(m).Address.begin()+n);//удалили адрес

}

if (InvInd.at(m).Address.empty()) //авто с такой маркой больше нет

InvInd.erase(InvInd.begin() + m);//удаляем марку

}

}

return 1;

}

else

return 0;

}

//вывод таблицы

void showMassive(Keys MassiveOfSegments[], bool ind){

cout << " Государственный регистрационный номер | Марка | В наличии\n";

for (unsigned int i = 0; i < NumOfSegments; i++){ //показ всех непустых записей

if ( MassiveOfSegments[i].car.regnumber[0]!=0) {

cout << " ";

for(int j=0;j<R;j++)

cout << MassiveOfSegments[i].car.regnumber[j];

cout <<" | "<< MassiveOfSegments[i].car.brand;

for(int s=0;s<(B-strlen(MassiveOfSegments[i].car.brand));s++)

cout<< " ";

if (MassiveOfSegments[i].car.exist)

cout << "| да ";

else

cout << "| нет";

if (ind)

cout <<" "<< i << endl;

else

cout << endl;

}

}

}

//вывод всей информации об одном авто

void ShowElem(Keys MassiveOfSegments[], unsigned int Index ){

if (!MassiveOfSegments[Index].empty) {

cout << "\nГосударственный регистрационный номер: ";

cout << MassiveOfSegments[Index].car.regnumber;

cout << endl;

cout << "Марка: " << MassiveOfSegments[Index].car.brand << endl;

cout << "Цвет: " << MassiveOfSegments[Index].car.color << endl;

cout << "Год выпуска: " << MassiveOfSegments[Index].car.year << endl;

if (MassiveOfSegments[Index].car.exist)

cout << "В наличии\n";

else

cout << "Нет в наличии\n";

}

}

//вывод всей информации об одном авто + клиент

void ShowElemAuto\_Cust(Keys MassiveOfSegments[], unsigned int Index, List\* begin, Tree\* MainTree){

ShowElem(MassiveOfSegments, Index);

List\* cust = NULL;

if (FindListElem(begin, MassiveOfSegments[Index].car.regnumber, cust)) {

Customer c = \*FindTreeElem(cust->data.drivenumber, MainTree);

cout << "Номер водительского удостоверения: " << c.drivenumber<< endl;

cout << "ФИО: " << c.name << endl;

}

}

//ввод данных об авто

void InputDataAuto(Auto& x, bool indicator, Keys MassiveOfSegments[],unsigned int& Index){

char rnum[R];//регистрационный номер

int l (0),k;

cout << "Государственный регистрационный номер" << endl <<

"в формате \"ANNNAA-NN\" (A - буквы, N - цифры)" << endl <<

"Допустимые буквы (А,В,Е,К,М,Н,О,Р,С,Т,У,Х): ";

while (l == 0 || cin.fail()) {

l = 1; k = 0;

//ввод

cin.clear();

cin.ignore(cin.rdbuf()->in\_avail());

cin.getline(rnum,R);

//считаем количество совпавших символов с допустимыми

vector<int> m = { 0,4,5 };

for (int i : m) {

for (int j = 0; j < 12; j++) {

if ((int)rnum[i] == (int)letters[j])

k++;

}

}

//проверка символов

if (((int)rnum[6]!=196 && (int)rnum[6] !=45) || (int)rnum[1] < 48 || (int)rnum[1]>57 || (int)rnum[2] < 48 || (int)rnum[2]>57 || (int)rnum[3] < 48 || (int)rnum[3]>57 || (int)rnum[7] < 48 || (int)rnum[7]>57 || (int)rnum[8] < 48 || (int)rnum[8]>57 || k!=3) {

cout << "Неверный ввод. Повторите попытку: ";

l = 0;

}

}

strcpy\_s(x.regnumber,rnum);

//проверка на уникальность и остальные данные не нужны

if (!indicator)

return;

//проверка на уникальность

if (Find\_HTableClose(x, MassiveOfSegments, Index)) {

cout << "Авто с таким номером уже существует.\n";

return;

}

char brand[B];

cout << "Марка: ";

cin.ignore(cin.rdbuf()->in\_avail());

cin.getline(brand, B);

//проверка на длину строки

CheckLength(brand,B);

strcpy\_s(x.brand, brand);

char color[C];

cout << "Цвет: ";

cin.ignore(cin.rdbuf()->in\_avail());

cin.getline(color, C);

//проверка на длину строки

CheckLength(color, C);

strcpy\_s(x.color, color);

int year;

cout << "Год выпуска: ";

cin.ignore(cin.rdbuf()->in\_avail());

cin>>year;

while (cin.fail() || year>2021||year<1980){

cin.clear();

cin.ignore(cin.rdbuf()->in\_avail());

cout << "Неверный ввод. Повторите попытку (1980<год<2021): ";

cin >> year;

}

x.year=year;

x.exist = true;

}

//вывод всех авто одной марки

void FindBrand(const vector <SecondKey>& InvertInd, char marka[], Keys MassiveOfSegments[]){

bool find = false;

for (int i = 0;i<size(InvertInd);i++) {

if (strcmp(InvertInd[i].brand, marka) == 0) { //нашли марку

find = true;

for (unsigned int j : InvertInd[i].Address)

ShowElem(MassiveOfSegments, j);

}

}

if(!find)

cout << "Авто с такой маркой не найдены.\n";

}

//////РАБОТА СО СПИСКОМ//////////////////////////////////////////////////////////////////////////////

// Добавление элемента в конец списка

void AddListElem(List\*& begin, List\*& cur, Rental elem,int& numofrecord){

List\* p = new List;

p->data = elem;

//проверка, является ли список пустым

if (begin == NULL){

p->ptr = NULL;

begin = p;

}

else{

p->ptr = cur->ptr;

cur->ptr = p;

}

cur = p;

numofrecord++;

}

//вывод списка проката

void ShowList(List\* begin){

List\* p = begin;

cout << "Водительское удостоверение | Регистрационный номер | Дата выдачи | Дата возврата\n";

while (p != NULL){

cout << p->data.drivenumber;

cout << " | ";

cout << p->data.regnumber;

cout<< " | ";

cout << p->data.dateissue;

for (int i = 0; i < R-strlen(p->data.dateissue); i++)

cout << ' ';

cout << " | "<<p->data.dateret<<endl;

p = p->ptr;

}

}

//поиск в списке записей без даты возврата

bool FindListElem(List\* begin, char str[], List\*& cur){

List\* p = begin;

bool find = false;

while (p != NULL){

if ((!strcmp(p->data.regnumber, str)|| !strcmp(p->data.drivenumber, str)) && (p->data.dateret[0]==32||p->data.dateret[0]==0)){

find = true; cur = p;

return find;

}

p = p->ptr;

}

return find;

}

//поиск в списке

bool FindListElem(List\* begin){

List\* p = begin;

bool find = false;

while (p != NULL){

if ((p->data.dateret[0] == 32 || p->data.dateret[0] == 0)) {

find = true;

return find;

}

p = p->ptr;

}

return find;

}

//проверка ввода даты

bool IsRange(int value, int min, int max){

if (value < min) return 0;

if (value > max) return 0;

return 1;

}

bool CheckDate(char date[]) {

vector<int> ch = {0,1,3,4,6,7};

for (int i : ch){

if (date[i]<48||date[i]>57)

return 0;

}

if (date[2] != 56 && date[2] != '.' || date[5] != 56 && date[5] != '.')

return 0;

char year[] = { date[6],date[7] };

char month[] = { date[3],date[4] };

char day[] = { date[0],date[1] };

int year\_c = atoi(year);

int month\_c = atoi(month);

int day\_c = atoi(day);

if (year\_c < 0 || year\_c > 21)

return 0;

switch (month\_c) {

case 1: //JANUARY

case 3: //MARCH

case 5: //MAY

case 7: //JULY

case 8: //AUGUST

case 10: //OCTOBER

case 12: //DECEMBER

return IsRange(day\_c, 1, 31);

case 2:

if (year\_c % 4 == 0)

return IsRange(day\_c, 1, 29);

else

return IsRange(day\_c, 1, 28);

case 4: //APRIL

case 6: //JUNE

case 9: //SEPTEMBER

case 11: //NOVEMBER

return IsRange(day\_c, 1, 30);

default:

return 0;

}

return 1;

}

bool CheckDate\_2(char dateissue[], char date[]) {

char year[] = { date[6],date[7] };

char month[] = { date[3],date[4] };

char day[] = { date[0],date[1] };

int year\_2 = atoi(year);

int month\_2 = atoi(month);

int day\_2 = atoi(day);

year[0] = dateissue[6];year[1] = dateissue[7];

month[0] = dateissue[3];month[1] = dateissue[4];

day[0] = dateissue[0];day[1] = dateissue[1];

int year\_1 = atoi(year);

int month\_1 = atoi(month);

int day\_1 = atoi(day);

if (year\_1 > year\_2)

return 0;

else if (year\_1 == year\_2)

if (month\_1 > month\_2)

return 0;

else if (month\_1 == month\_2)

if (day\_1 > day\_2)

return 0;

return 1;

}

//редактирование списка при возврате машины

bool EditList(List\*& begin, int NumOfrecord, Auto& x){

List\* p = begin; int s(0);

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

if (p->data.dateret[0] == 0 || p->data.dateret[0] == ' ')

s++;

p = p->ptr;

}

if (s < 1) {

cout << "Возвращать нечего.\n";

return 0;

}

ShowList(begin);

cout << "Введите порядковый номер записи, чтобы зарегистрировать возврат: ";

int num;bool l(0);

while (l==0){

l = 1;

p = begin;

cin >> num;

if (cin.fail() || num<1 || num > NumOfrecord) {

cin.clear();

cout << "Неверный ввод. Введите заново: ";

l = 0;

}

else {

for (int i = 1; i < num; i++)

p = p->ptr;

if (p->data.dateret[0] != ' ' && p->data.dateret[0] != 0){

cout << "Этот автомобиль уже возвращен. Введите номер заново: \n";

l = 0;

}

}

}

char date[R];

cout << "Введите дату возврата (дд.мм.гг): ";

cin.ignore(cin.rdbuf()->in\_avail());

cin.getline(date, R);

//проверка на символы //проверка на дату (позже)

while (!CheckDate(date) || !CheckDate\_2(p->data.dateissue,date)){

cout << "Повторите ввод (дата должна быть позже даты выдачи): ";

cin.ignore(cin.rdbuf()->in\_avail());

cin.getline(date, R);

}

strcpy\_s(p->data.dateret, date);

strcpy\_s(x.regnumber, p->data.regnumber);

return 1;

}

//вывод списка для регистрации авто на прокат

bool Registration(unsigned int& num, Keys MassiveOfSegments[]) {

int s(0);

for (int j = 0; j < NumOfSegments; j++)

if (MassiveOfSegments[j].car.exist)

s++;

if (s < 1) {

cout << "Свободных машин нет.\n";

return 0;

}

showMassive(MassiveOfSegments, 1);

cout << "Выберите машину для регистрации по номеру, справа от таблицы: ";

bool l(0);

while (l == 0) {

l = 1;

cin >> num;

if (num > NumOfSegments || num < 0 || cin.fail()) {

cout << "Ошибка ввода. Повторите ввод: ";

l = 0;

}

else if (!MassiveOfSegments[num].car.exist) {

cout << "Вы не можете выбрать машину, которой нет в наличии. ";

l = 0;

}

}

return 1;

}

//ввод данных при регистрации проката

bool InputDataRental(Rental& y, Keys MassiveOfSegments[], Tree\* MainTree, int& numOfElem){

unsigned int num;

if (!Registration(num, MassiveOfSegments))

return 0;

int n, n2(1);Customer\* x;

cout << " Номер водительского удостоверения | ФИО" << endl;

ShowTree(MainTree, 1,n2);

cout << "\nВыберите клиента для регистрации по номеру: ";

cin >> n;

while (n<1 || n>numOfElem || cin.fail()){

cout << "Ошибка ввода. Введите повторно: ";

cin.clear();

cin >> n;

}

int i(1);

x = FindCustTree(MainTree, n,i);

strcpy\_s(y.drivenumber, x->drivenumber);

strcpy\_s(y.regnumber, MassiveOfSegments[num].car.regnumber);

for (int i = 0; i < R; i++)

y.dateret[i] = 0;

cout << "Введите дату выдачи (дд.мм.гг): ";

char date[R];

cin.ignore(cin.rdbuf()->in\_avail());

cin.getline(date, R);

//проверка на длину строки

CheckLength(date, R);

//проверка даты

while (!CheckDate(date) || cin.fail()){

cout << "Повторите ввод (год от 00 до 21): ";

cin.ignore(cin.rdbuf()->in\_avail());

cin.getline(date, R);

}

strcpy\_s(y.dateissue, date);

MassiveOfSegments[num].car.exist = false;

return 1;

}

//сортировка слиянием

List\* merge\_lists(List\* list1, List\* list2){

assert(list2 != nullptr);

List\* list, \*\* pnext = &list;

while (list1 != nullptr){

if (strcmp(list1->data.regnumber, list2->data.regnumber)==1)

swap(list1, list2);

\*pnext = list1;

pnext = &list1->ptr;

list1 = \*pnext;

}

\*pnext = list2;

return list;

}

List\* merge\_sort\_list(List\* list){

array<List\*, 32> hooks = {};

auto it\_hook\_end = hooks.begin();

while (list != nullptr){

List\* next = list->ptr;

list->ptr = nullptr;

auto it\_hook = hooks.begin();

for (; it\_hook != it\_hook\_end && \*it\_hook != nullptr; ++it\_hook){

list = merge\_lists(list, \*it\_hook);

\*it\_hook = nullptr;

}

if (it\_hook == it\_hook\_end)

if (it\_hook < hooks.end())

it\_hook = it\_hook\_end++;

else

--it\_hook;

assert(\*it\_hook == nullptr);

\*it\_hook = list;

list = next;

}

for (auto it\_hook = hooks.begin(); it\_hook != it\_hook\_end; ++it\_hook)

if (\*it\_hook != nullptr)

list = merge\_lists(list, \*it\_hook);

return list;

}

//загрузка из файла (авто)

void FillAvto(Keys MassiveOfSegments[], ifstream& f, unsigned int Index, vector<SecondKey>& InvInd) {

Auto x;

char r[R];

char b[B];

char c[C];

int y;

while (!f.eof()) {

f >> r;

f >> b;

f >> c;

f >> y;

strcpy\_s(x.regnumber, r);

strcpy\_s(x.brand, b);

strcpy\_s(x.color, c);

x.year = y;

Add\_HTableClose(x,MassiveOfSegments,Index,InvInd);

}

}

//загрузка из файла (прокат)

void FillRental(List\*& begin, List\*& cur, ifstream& f, int& NumOfrecord, Keys MassiveOfSegments[]) {

Rental y; Auto x;

unsigned int ind;

char n[N];

char r[R];

char dateiss[R];

char dateret[R];

while (!f.eof()) {

f.get(n, N);

f.get();

f >> r;

f >> dateiss;

f.get();

f.get(dateret, R);

f.get();

strcpy\_s(y.drivenumber, n);

strcpy\_s(y.regnumber, r);

strcpy\_s(x.regnumber, r);

strcpy\_s(y.dateissue, dateiss);

strcpy\_s(y.dateret, dateret);

AddListElem(begin,cur,y,NumOfrecord);

if (dateret[1] == ' ') {

Find\_HTableClose(x, MassiveOfSegments, ind);

MassiveOfSegments[ind].car.exist = false;

}

begin = merge\_sort\_list(begin);

}

}

//загрузка из файла (клиенты)

void FillCustomer(ifstream& f, Tree\*& MainTree, int& numOfElem){

Customer x;

while (!f.eof()) {

Tree\* Elem = new Tree;

Elem->Left = NULL;

Elem->Right = NULL;

Elem->Prev = NULL;

Elem->height = 0;

f.get(x.drivenumber, N);

f.get();

f.get(x.name, M, '.');

f.get();

f.get(x.passport, P);

f.get();

f.getline(x.address, A, '\n');

strcpy\_s(Elem->elem.drivenumber, x.drivenumber);

strcpy\_s(Elem->elem.name, x.name);

strcpy\_s(Elem->elem.passport, x.passport);

strcpy\_s(Elem->elem.address, x.address);

AddTreeElem(MainTree, Elem, numOfElem);

}

}

int main()

{

system("color f5");

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

Keys MassiveOfSegments[NumOfSegments];//хеш-таблица

vector<SecondKey> InvertInd;//список инвертированных индексов для поиска по вторичному ключу

Clear\_HTableClose(MassiveOfSegments);//очищение таблицы

Customer z;//значение элемента дерева

Tree\* MyTree = NULL;

Tree\* Elem;

int numOfElem(0);

Auto x;//вводимая структура авто

unsigned int Index(0);//адрес элемента

Rental y;//вводимая структура проката

int NumOfrecord(0);//количество записей в списке

List\* head = NULL;

List\* cur = NULL;

//загрузка данных из файла

ifstream avto;

avto.open("C:\\Users\\79603\\Documents\\учеба\\САОД\\курсовая2021\\avto.txt", ios\_base::in);

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

cout << "Ошибка при открытии файла 1!" << endl;

exit(1);

}

ifstream rental;

rental.open("C:\\Users\\79603\\Documents\\учеба\\САОД\\курсовая2021\\rental.txt", ios\_base::in);

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

cout << "Ошибка при открытии файла 2!" << endl;

exit(1);

}

ifstream customer;

customer.open("c:\\Users\\79603\\Documents\\учеба\\САОД\\курсовая2021\\customer.txt", ios\_base::in);

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

cout << "Ошибка при открытии файла!" << endl;

exit(1);

}

FillCustomer(customer, MyTree,numOfElem);

FillAvto(MassiveOfSegments,avto,Index,InvertInd);

FillRental(head, cur, rental,NumOfrecord, MassiveOfSegments);

int n = -1;

// Меню пользователя

while (n != 0) {

cout << endl <<

"1 - Добавление нового автомобиля " << "7 - Регистрация нового клиента "<<"13 - Регистрация отправки авто в ремонт" <<endl<<

"2 - Удаление сведений об автомобиле " << "8 - Снять клиента с обслуживания "<<"14 - Регистрация прибытия авто из ремонта"<<endl<<

"3 - Просмотр имеющихся автомобилей " << "9 - Просмотр клиентов "<<"15 - Регистрация выдачи клиенту авто на прокат" << endl <<

"4 - Очистка данных об автомобилях " << "10 - Очистка данных о клиентах "<<"16 - Регистрация возврата авто от клиента"<<endl <<

"5 - Поиск автомобиля по гос. рег. номеру " << "11 - Поиск клиента по номеру вод. уд. "<<"17 - Просмотр данных о выданных авто "<< endl <<

"6 - Поиск автомобиля по названию марки " << "12 - Поиск клиента по фрагменту ФИО или адреса "<<"0 - Выход\nВыберите действие: ";

cin >> n;

while ((n < 0) || (n > 17) || cin.fail()) {

cout << "Ошибка ввода, выберите действие: ";

cin.clear();

cin.ignore(cin.rdbuf()->in\_avail());

cin >> n;

}

switch (n){

case 1: {//добавление нового авто

InputDataAuto(x, 1, MassiveOfSegments, Index);//ввод данных об авто

if (Add\_HTableClose(x, MassiveOfSegments, Index, InvertInd))//добавление записи в таблицу

cout << "Автомобиль добавлен.\n";

else

cout << "Не удалось добавить.\n";

break;

}

case 2: {//удаление сведений об автомобиле

InputDataAuto(x, 0,MassiveOfSegments,Index);//ввод рег номера

if (Del\_HTableClose(x, MassiveOfSegments, InvertInd))//если авто найдено и в наличии удаляем

cout << "Сведения удалены.\n";

else

cout << "Автомобиль с таким номером не найден или его нет в наличии.\n";

break;

}

case 3: {//просмотр имеющихся авто

int n(0);

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

if (MassiveOfSegments[i].car.regnumber[0] == 0)

n++;

}

if (n != NumOfSegments)

showMassive(MassiveOfSegments, 1);

else

cout << "Список автомобилей пуст.\n";

break;

}

case 4: {//очитска данных об авто

for (int i = 0; i < NumOfSegments; i++)

if (MassiveOfSegments[i].car.exist == true)//удаление всех элементов кроме тех что в прокате

Del\_HTableClose(MassiveOfSegments[i].car, MassiveOfSegments, InvertInd);

cout << "Данные удалены (кроме тех, что находятся в ремонте или выданы на прокат).\n";

break;

}

case 5: {//поиск по гос номеру

InputDataAuto(x, 0,MassiveOfSegments,Index);//ввод номера

if (Find\_HTableClose(x, MassiveOfSegments, Index))

ShowElemAuto\_Cust(MassiveOfSegments, Index,head, MyTree);

else

cout << "Автомобиль с таким номером не найден.\n";

break;

}

case 6: {//Поиск автомобиля по названию марки

char marka[B];

cout << "Введите марку автомобиля: ";

cin.ignore(cin.rdbuf()->in\_avail());

cin.getline(marka,B);

FindBrand(InvertInd,marka, MassiveOfSegments);

break;

}

case 13: {//Регистрация отправки авто в ремонт

if (!Registration(Index, MassiveOfSegments))//вывод списка авто

break;

else {//если есть свободные машины - меняем пизнак наличия

cout << "Авто отправлено в ремонт.\n";

MassiveOfSegments[Index].car.exist = false;

}

break;

}

case 14: {//Регистрация прибытия авто из ремонта

InputDataAuto(x, 0,MassiveOfSegments,Index);//ввод рег номера

if (Find\_HTableClose(x, MassiveOfSegments, Index))

{

if (MassiveOfSegments[Index].car.exist == false) {//авто нет в наличии

if (!FindListElem(head,MassiveOfSegments[Index].car.regnumber, cur)) {//авто нет в списке проката

cout << "Авто прибыло из ремонта.\n";

MassiveOfSegments[Index].car.exist = true;//меняем признак наличия

}

else

cout << "Авто находится в прокате.\n";

}

else

cout << "Автомобиль не был в ремонте (доступен для проката).\n";

}

else

cout << "Автомобиль с таким номером не найден.\n";

break;

}

case 15: {//Регистрация выдачи клиенту авто на прокат

if (InputDataRental(y, MassiveOfSegments,MyTree, numOfElem)) {//ввод данных

AddListElem(head, cur, y, NumOfrecord);//добавление элемента в список

head = merge\_sort\_list(head);//сортировка по рег номеру

cout <<"Регистрация прошла успешно.\n";

break;

}

else

break;

}

case 16: {//Регистрация возврата авто от клиента

if (!head) {

cout << "Список пуст.\n";

break;

}

else if (EditList(head, NumOfrecord, x)) {//есть есть что возвращать

Find\_HTableClose(x, MassiveOfSegments, Index);//ищем машину с таким номером

MassiveOfSegments[Index].car.exist = true;//меняем признак наличия

cout << "Регистрация прошла успешно.\n";

break;

}

else

break;

}

case 17: {//просмотр данных о выданных авто

if (head)

ShowList(head);

else

cout << "Список пуст.\n";

break;

}

case 7: {//регистрация нового клиента

Elem = new Tree;

Elem->Left = NULL;

Elem->Right = NULL;

Elem->Prev = NULL;

Elem->height = 0;

if (InputDataCust(Elem, z, MyTree))//ввод данных

break;

else

AddTreeElem(MyTree, Elem, numOfElem);//добавление элемента в дерево

break;

}

case 8: {//снятие с обслуживания клиента

cout << "Введите номер водительского удостоверения: ";

cin.ignore(cin.rdbuf()->in\_avail());

cin.getline(z.drivenumber, N);//ввод номера вод уд-ия

if (!FindListElem(head, z.drivenumber,cur)) //если клиент существует и у него нет машины

DeleteTreeElem(MyTree, z.drivenumber, numOfElem);

else

cout << "У клиента есть машина.\n";

break;

}

case 9: {//просмотр клиентов

if (MyTree) {

int n(0);

cout << " Номер водительского удостоверения | ФИО" << endl;

ShowTree(MyTree,0,n);

cout << endl;

}

else

cout << "Список клиентов пуст." << endl;

break;

}

case 10: {//очистка данных о клиентах

//только если нет машины

if(!FindListElem(head)){

Clean(MyTree);

cout << "Данные удалены.\n";

}

else

cout<<"Невозможно удалить, есть клиенты с машинами.\n";

break;

}

case 11: {//поиск по номеру

Customer\* Current = NULL;

cout << "Введите номер водительского удостоверения: ";

cin.ignore(cin.rdbuf()->in\_avail());

cin.getline(z.drivenumber, N);

if (FindTreeElem(z.drivenumber, MyTree))

ShowElemAuto\_Cust(MyTree, z.drivenumber, head, cur);

else

cout << "Клиент не найден.\n";

break;

}

case 12: {//поиск по фрагментам фио или адреса

int c;

cout << "Введите число (поиск по фрагменту ФИО - 1, адреса - 2): ";

cin >> c;

while ((c < 1) || (c > 2)) {

cout << "Ошибка ввода, повторите действие: ";

cin.clear();

cin.ignore(cin.rdbuf()->in\_avail());

cin >> c;

}

if (c == 1) {

char FIO[M];

cout << "Введите фрагмент ФИО: ";

cin.ignore(cin.rdbuf()->in\_avail());

cin.getline(FIO, M);

TextSearchFIO(FIO, MyTree);

}

else if (c == 2) {

char adr[A];

cout << "Введите фрагмент адреса: ";

cin.ignore(cin.rdbuf()->in\_avail());

cin.getline(adr, A);

TextSearchAdr(adr, MyTree);

}

break;

}

case 0: {

exit(0);

break; }

}

}

return 0;

}