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 ИНЖЕНЕРИИ |

КУРСОВОЙ ПРОЕКТ  
ЗАЩИЩЕН С ОЦЕНКОЙ

РУКОВОДИТЕЛЬ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Старший преподаватель |  |  |  | С. А. Рогачев |
| 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

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| СТУДЕНТ ГР. № | Z9431 | |  | 14,06,2021 |  | А. А. Пайвин |
|  | номер группы | |  | подпись, дата |  | инициалы, фамилия |
| Студенческий билет № | | 2018/4265 | |  |  |  |

Санкт-Петербург 2021

**СОДЕРЖАНИЕ**

[1. Задание на курсовой проект 3](#_Toc74788171)

[2. Введение 7](#_Toc74788172)

[Данные о клиентах 8](#_Toc74788173)

[Данные об автомобилях 8](#_Toc74788174)

[Данные о выдаче на прокат или возврате автомобилей 9](#_Toc74788175)

[3. Алгоритмы и структуры данных 10](#_Toc74788176)

[4. Описание программы 15](#_Toc74788177)

[5. Тестирование программы 17](#_Toc74788178)

[6. Заключение 21](#_Toc74788179)

[Список использованной литературы 22](#_Toc74788180)

[ПРИЛОЖЕНИЕ 1. Текст программы 23](#_Toc74788181)

[Файл saod\_kp.cpp 23](#_Toc74788182)

[Файл cleents.h 68](#_Toc74788183)

[Файл clients.cpp 69](#_Toc74788184)

[Файл cars.h 78](#_Toc74788185)

[Файл cars.cpp 79](#_Toc74788186)

[Файл take\_return.h 90](#_Toc74788187)

[Файл take\_return.cpp 90](#_Toc74788188)

# 1. Задание на курсовой проект

1.1 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.

1.2 Вариант задания

* предметная область: обслуживание клиентов в бюро проката автомобилей
* метод хеширования: закрытое хеширование с линейным опробованием
* метод сортировки: распределением
* вид списка: линейный однонаправленный
* метод обхода дерева: обратный
* алгоритм поиска слова в тексте: прямой

Обслуживание клиентов в бюро проката автомобилей

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.

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 номер водительского удостоверения) определяется студентом самостоятельно.

Данные о клиентах должны быть организованны в виде

АВЛ-дерева поиска, упорядоченного по «номеру водительского удостоверения».

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.

Данные об автомобилях должны быть организованны в виде хеш-таблицы, первичным ключом которой является «Государственный регистрационный номер». Метод хеширования - **закрытое хеширование с линейным опробованием**

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.

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 – **линейный однонаправленный,** метод сортировки - **распределением**

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;

58

– регистрацию отправки автомобиля в ремонт;

– регистрацию прибытия автомобиля из ремонта;

– регистрацию выдачи клиенту автомобиля на прокат;

– регистрацию возврата автомобиля от клиентов.

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.

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

Поиск клиента по фрагментам ФИО или адреса должен

осуществляться путем систематического обхода АВЛ-дерева поиска. Метод обхода - **обратный**. 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 **прямой** алгоритм поиска слова в тексте.

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».

При регистрации прибытия автомобиля из ремонта значение поля «Признак наличия» для соответствующего автомобиля

изменяется на значение «Истина».

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

При регистрации выдачи автомобиля клиенту или возврата автомобиля от клиента должно корректироваться значение

поля «Признак наличия» для соответствующего автомобиля.

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

Во время выполнения курсового проекта необходимо реализовать информационную систему для предметной области «Обслуживание клиентов в бюро проката автомобилей», осуществляющей ввод, хранение, обработку и вывод данных о:

– клиентах;

– автомобилях, принадлежащих бюро проката;

– выдаче на прокат и возврате автомобилей от клиентов.

Создадим консольное приложение.

Для удобства пользованием программой создадим главное меню стремя пунктами: клиенты, автомобили, выдача/возврат. При выборе конкретного раздела будет выходить подменю с доступными операциями над данным разделом

В подменю клиентов будут доступны следующие операции:

* Регистрация нового клиента
* Снятие с обслуживания клиента
* Просмотр всех зарегистрированных клиентов
* Очистка данных о клиентах
* Поиск клиента по номеру водительского удостоверения (Полная информация о клиенте)
* Поиск клиента по фрагментам ФИО или адреса

В подменю автомобилей будут доступны операции:

* Добавление нового автомобиля
* Удаление сведений об автомобиле
* Просмотр всех имеющихся автомобилей
* Очистка данных об автомобилях
* Поиск автомобиля по государственному регистрационному номеру
* Поиск автомобиля по названию марки автомобиля
* Регистрация отправки автомобиля в ремонт
* Регистрация прибытия автомобиля из ремонта

В подменю выдачи и возврата будут доступны операции:

* Регистрация выдачи клиенту автомобиля на прокат
* Регистрация возврата автомобиля от клиента
* Просмотр всех выданных автомобилей

Для удобного ориентирования в проекте разобьем его на 3 файла, для каждого раздела, в каждом из них будут реализованы соответствующие функции данного раздела

Если пользователь программы во время выполнения любой операции не захочет ее завершать, нужно добавить возможность выхода в предыдущее меню. В нашем случае это можно осуществить с помощью ввода цифры 0 вместо данных

### Данные о клиентах

Согласно варианту, данные о клиентах должны быть организованны в виде

АВЛ-дерева поиска, упорядоченного по «номеру водительского удостоверения. При добавлении нового клиента, необходимо проверять всю вводимую информацию на корректный формат. Номер водительского удостоверения должен представлять собой строку формата «RR AA

NNNNNN», ФИО – строку до 50 символов, паспорт должен состоять из 10 цифр без пробела, адрес должен содержать до 150 символов. Также Должна происходить проверка на ввод пустой строки.

В нашей системе не может существовать два одинаковых номера водительского удостоверения клиента, поэтому, перед добавлением нового клиента, сначала проверим, существует ли данный номер водительского удостоверения в системе. Если нет, продолжим ввод информации.

После добавления данных проверим дерево на балансировку, и, в случае необходимости, отбалансируем.

Вывод информации обо всех клиентах реализуем двумя способами

Сначала выведем их списком, а затем, ниже в виде бинарного дерева, чтобы иметь наглядное представление.

При поиске информации о конкретном клиенте, кроме его данных выведем на экран и об автомобилях, которые ему выданы, если таковые имеются

Поиск клиента по фрагментам ФИО или адреса осуществляем обратным обходом бинарного дерева, метод поиска-прямой. Не останавливаем поиск при нахождении первого клиента, ищем все совпадения.

При удалении информации о клиенте, проверяем выдан ли ему автомобиль.

### Данные об автомобилях

Данные об автомобилях должны быть организованны в виде хеш-таблицы, первичным ключом которой является «Государственный регистрационный номер». Метод хеширования - закрытое хеширование с линейным опробованием

При добавлении нового автомобиля также необходимо проверять всю вводимую информацию на корректный формат.

Государственный регистрационный номер должен представлять собой строку формата «ANNNAA-NN», марка - строку до 20 символов, цвет – строку до 20 символов, год выпуска – целое четырехзначное число от 1800.

В нашей системе не может существовать два одинаковых государственных регистрационных номера автомобиля, поэтому, перед добавлением нового автомобиля, сначала проверим, существует ли данный государственный регистрационный номер в системе. Если нет, продолжаем ввод информации.

Создадим изначально хеш-таблицу размерностью в 10 элементов. А при добавлении нового элемента будем считать количество попыток записи. Если было совершено 5 неудачных попыток записи, увеличим размер хеш-таблицы вдвое и рехешируем ее. Наша хеш-функция будет принимать на вход размер таблицы, поэтому позиция элемента будет от него зависеть.

Изначально инициализируем таблицу с пустыми строками в поле государственного регистрационного номера. При удалении автомобиля, в поле будем оставлять 0, чтобы не потерять элементы на случай коллизии.

Ключи, образующие коллизию: «Е769СК-01» и «Х407АК-99», «А390РК-68» и «Т942СУ-20».

При поиске автомобиля, если он кому-то выдан, выведем номер водительского удостоверения этого клиента

При поиске автомобиля по марке, сначала пройдемся по списку выданных автомобилей, так как такой автомобиль вероятнее всего нуждается в поиске. Затем проходим последовательно по хеш-таблице от начала до конца, и выводим найденные автомобили, которые не выданы клиенту. Такой способ предпочтительнее, так как если сразу начать искать по хеш-таблице, поиск нужного автомобиля займет больше времени.

Регистрацию отправки автомобиля в ремонт осуществляем только, если автомобиль в наличии. Аналогично, регистрацию прибытия из ремонта осуществляем только, если автомобиль находился в ремонте.

При удалении информации об автомобиле проверяем, есть ли он в наличии.

### Данные о выдаче на прокат или возврате автомобилей

Данные о выдаче на прокат или возврате автомобилей от

клиентов должны быть организованны в виде линейного однонаправленного списка, который упорядочен по первичному ключу – «Государственный регистрационный номер». Метод сортировки – распределением

При регистрации выдачи автомобиля проводим аналогичные проверки, существует ли данный клиент, существует ли автомобиль с данным номером, и если ли данный автомобиль в наличии, проверяем даты на корректный формат, а также проверяем, чтобы срок выдачи составлял от одного дня до одного года.

Список должен быть отсортирован по государственному регистрационному номеру автомобиля. Правильнее всего было бы добавлять элементы в середину списка, так, чтобы он был сразу отсортирован. Но, так как нужно реализовать сортировку распределением, для демонстрации ее работы добавлять элемент будем в начало списка, а затем вызывать функцию сортировки. При регистрации возврата проверяем, выдан ли данный автомобиль.

При создании объектов каждой структуры выделяется динамическая память, поэтому при выходе из программы нужно обязательно ее очистить

# 3. Алгоритмы и структуры данных

Три основные структуры данных:

struct clients

{

string num\_VU; // номер водительского удостоверения

string FIO; // Фамилия имя отчество

string passport;

string address;

clients\* Left; //указатели на соседние ветви

clients\* Right;

short int h;// высота поддерева

};

struct cars

{

string grn; // гос регистрационный номер

string mark; // марка авто

string color; // цвет

int year; // год производства

bool have;// имеется ли в наличии

bool rem; // в ремонте ли

};

struct take\_return

{

string num\_VU; // номер водительского удостоверения

string grn; // гос регистрационный номер

string mark; // марка авто

string date\_of\_issue; // дата выдачи

string return\_date; // дата возврата

take\_return\* next; // указатель на следующий элемент

};

Функции из раздела с клиентами:

функция сравнения номеров ВУ для поиска нужного расположения листьев:

Сравниваем поэлементно по кодам символов.

bool comparison(string num\_VU, string num\_VU\_root);

Поиск клиента по номеру ВУ:

Движемся начиная от корня дерева.

void search\_cl(clients\* client, string num\_VU, clients\*& search\_client);

Функция добавления клиента без балансировки:

void Add(clients\* add\_client, clients\*& client);

Функция вывода дерева:

Дерево выводится повернутым влево, то есть корень находится слева, сверху - правые ветви, снизу - левые

void print\_client(clients\* client);

Функция вычисления баланс фактора:

Проверка разности высот между двумя ветвями

int bf(clients\* client);

Восстановление правильного значения высоты:

void fixheight(clients\*& client);

Правый поворот:

clients\* rotateright(clients\* client);

Левый поворот:

clients\* rotateleft(clients\* client);

Балансировка:

Балансировка может осуществлять как малые правый и левый повороты, так и большие, в зависимости от ситуации. Большой поворот включает в себя левый и правый поворот сразу, но может быть в разной последовательности.

clients\* balance(clients\* client);

Отбалансировать все дерево:

void balance\_all(clients\*& client);

Добавить клиента с последующей балансировкой:

Так как после добавления элемента может поменяться высота дерева, отбалансируем дерево, если она изменится

void Add\_balance(clients\* add\_client, clients\*& Root, short int& H);

Вывести всех клиентов:

Выведем клиентов симметричным обходом, чтобы они выглядели отсортированно

void print\_all\_clients(clients\* client);

Удалить клиента без проверки на высоту:

clients\* del\_elem(clients\*& client, string num\_VU);

Удалить клиента с проверкой на изменение высоты дерева:

void del\_check\_H(clients\*& Root, string num\_VU, short int& H);

Проверка на допустимые символы:

bool sym\_true(unsigned char sym);

Проверка на корректное значение номера ВУ:

bool check\_num\_VU(string num\_VU);

Проверка на корректное значение номера паспорта

10 цифр без пробела

bool check\_passport(string passport);

Удалить всех клиентов, кроме тех кому выдан автомобиль:

void del\_except\_t\_r(clients\*& Root, short int& H, take\_return\* head);

Вывод информации о клиенте:

void print\_client\_complete(clients\* client);

Поиск фрагмента в строке:

bool contain\_fragment(string str, string fragment);

Поиск по фрагментам фио или адреса:

void search\_fragment(clients\* client, string fragment, bool& header);

Полная очистка динамической памяти бинарного дерева:

void FreeTree(clients\* client);

Функции из раздела с автомобилями:

Поиск автомобиля по ГРН:

bool search\_car(cars\* hash\_table, string grn, cars& search\_car, int table\_size);

Получение позиции автомобиля в хеш-таблице:

int get\_pos(cars\* hash\_table, string grn, int table\_size);

Поиск автомобиля по марке:

Сначала ищем среди выданных автомобилей, а потом в хеш-таблице

void search\_mark(cars\* hash\_table, string mark, int table\_size, take\_return\* head);

Вывод информации об автомобиле:

void print\_car(cars car);

Вывод информации о всех автомобилях:

void print\_cars(cars\* hash\_table, int table\_size, int count\_cars);

Хеш-функция:

int hash\_function(string key, int n, int table\_size);

Рехеширование:

Увеличим размер хеш-таблицы вдвое, и найдем для элементов новые позиции.

void rehash(cars\* (&hash\_table), int& table\_size);

Добавить автомобиль с возможностью рехеширования и расширения таблицы:

void add\_car\_check(cars car, cars\* (&hash\_table), int& table\_size, int& count\_cars);

Добавить автомобиль в хеш-таблицу:

Если автомобиль добавлен до 5 попыток, возвращает true, иначе - false

bool add\_car(cars car, cars\* (&hash\_table), int table\_size);

Проверка на допустимые символы:

bool sym\_car(unsigned char sym);

Проверка на корректность ГРН:

bool check\_grn(string grn);

Проверка года:

bool check\_year(string year, int wYear);

Удаление автомобиля:

void del\_car(cars\* (&hash\_table), string grn, int table\_size, int& count\_cars);

Удаление всех автомобилей, которые есть в наличии:

void del\_car\_have(cars\* (&hash\_table), int& table\_size, int& count\_cars);

Регистрация отправки на ремонт:

void reg\_send\_rem(cars\* hash\_table, string grn, int table\_size);

Регистрация прибытия из ремонта:

void reg\_get\_rem(cars\* hash\_table, string grn, int table\_size);

Регистрация выдачи автомобиля клиенту:

void reg\_take\_car(cars\* hash\_table, string grn, int table\_size);

Регистрация возврата автомобиля от клиента:

void reg\_return\_car(cars\* hash\_table, string grn, int table\_size);

Функции из раздела выдачи и возврата:

Добавить информацию о выдаче:

void add\_t\_r(take\_return\*& head, take\_return\* t\_r, int& kol\_t\_r);

Удаление всей информации о выдаче:

void del\_all\_t\_r(take\_return\*& head, int& kol\_t\_r);

Удаление записи о выдаче автомобиля клиенту:

bool del\_t\_r(take\_return\*& head, string grn, int& kol\_t\_r);

Вывод всей информации о выдаче:

void print\_all\_t\_r(take\_return\* head);

Поиск по ГРН либо ВУ:

take\_return\* search\_tr(take\_return\* head, string grn\_VU);

Просмотр выданных автомобилей у клиента:

void print\_cars\_of\_client(take\_return\* head, string VU);

Сортировка списка:

Посимвольно, начиная справа, указатели на элементы перемещаются по двум двумерным массивам, тем самым происходит сортировка распределением

void sort\_t\_r(take\_return\*& head, int kol\_t\_r);

Проверка на корректное значение даты:

bool correct\_date(string date, long long& unix\_time);

# 4. Описание программы

При запуске программы на экран выводится консольное приложение с главным меню пользователя. Предлагается загрузить тестовые данные для проверки программы. Для этого необходимо ввести «1».

Далее на экране появляется главное меню. При нажатии на клавиатуре на определенную цифру осуществляется переход в соответствующий раздел, на экран выводится его меню с различными функциями, выбор которых происходит также при нажатии на клавиатуре на определенную цифру.

После выполнения любой функции меню снова выводится на экран. Для выхода из разделов в главное меню или для выхода из программы пользователю необходимо ввести 0. Если пользователь не может или не хочет завершить начатую функцию, он также может выйти, введя 0.

Главное меню:

1 - Клиенты

2 - Автомобили

3 - Выдача и возврат

0 – Выход

При нажатии цифры 1 выходит подменю:

1 - Регистрация нового клиента

2 - Снятие с обслуживания клиента

3 - Просмотр всех зарегистрированных клиентов

4 - Очистка данных о клиентах

5 - Поиск клиента по номеру водительского удостоверения (Полная информация о клиенте)

6 - Поиск клиента по фрагментам ФИО или адреса

0 - Выход в главное меню

При нажатии цифры 2 выходит подменю:

1 - Добавление нового автомобиля

2 - Удаление сведений об автомобиле

3 - Просмотр всех имеющихся автомобилей

4 - Очистка данных об автомобилях

5 - Поиск автомобиля по государственному регистрационному номеру

6 - Поиск автомобиля по названию марки автомобиля

7 - Регистрация отправки автомобиля в ремонт

8 - Регистрация прибытия автомобиля из ремонта

0 - Выход в главное меню

При нажатии цифры 3 выходит подменю:

1 - Регистрация выдачи клиенту автомобиля на прокат

2 - Регистрация возврата автомобиля от клиента

3 - Просмотр всех выданных автомобилей

0 - Выход в главное меню

Листинг программы находится в приложении 1

# 5. Тестирование программы

Запускаем программу, подтверждаем загрузку тестовых данных, введя «1»:

Результат мы можем видеть на рисунках 1-2

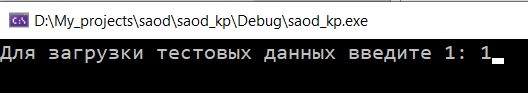


Рисунок 1

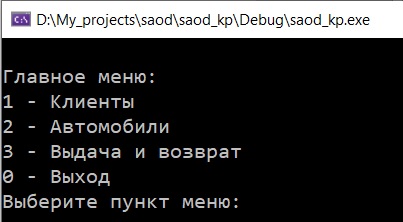


Рисунок 2

Вводим «1» и видим на экране меню клиентов, рисунок 3

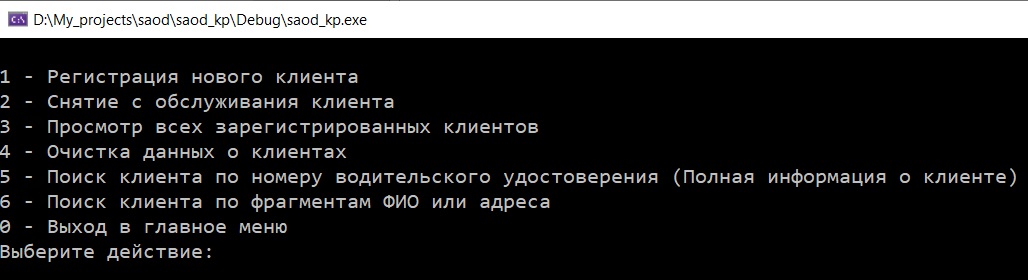


Рисунок 3

Для просмотра всех клиентов введем 3, рис. 4

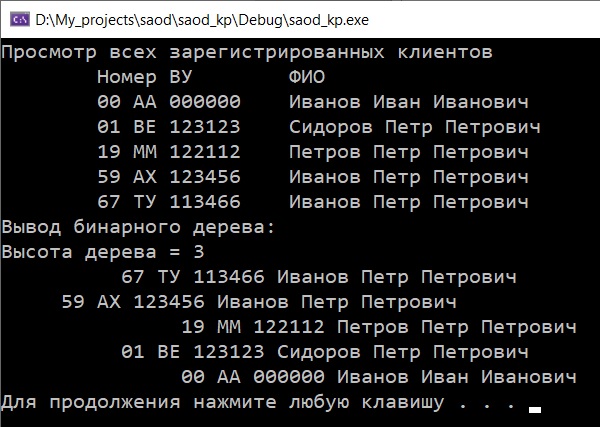


Рисунок 4

Нажимаем любую клавишу для продолжения, выходит меню клиентов, рисунок 3.

Вводим 6 для поиска по фрагментам и вводим «Петр»

Результат видим на рисунке 5

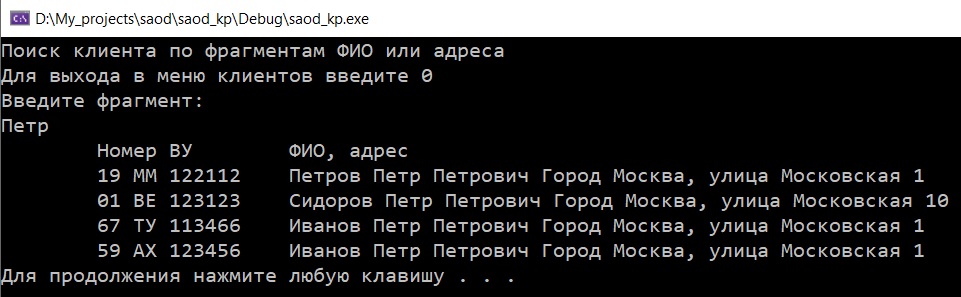


Рисунок 5

Нажимаем любую клавишу для продолжения, затем нажимаем «0» для выхода в главное меню (рис. 2), и переходим в раздел автомобилей, нажав клавишу 2. На рисунке 6 видим меню автомобилей

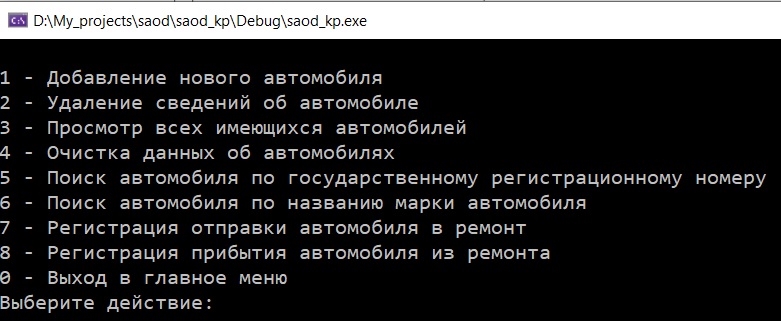


Рисунок 6

Нажимаем клавишу «3», на рисунке 7 видим все имеющиеся автомобили

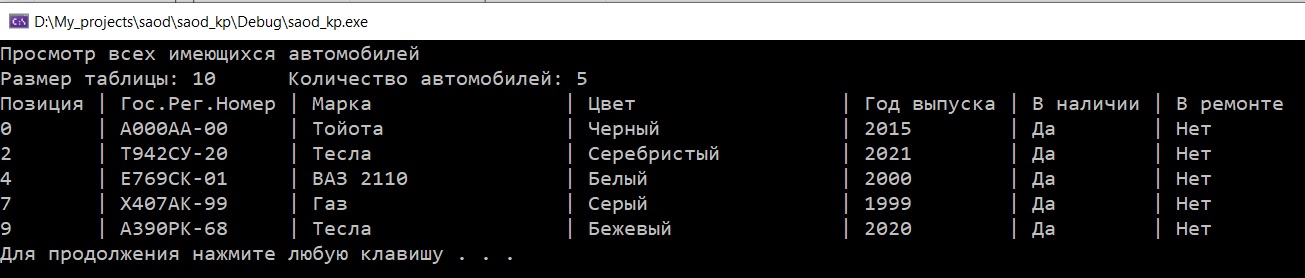


Рисунок 7

Нажимаем любую клавишу для продолжения и выходим в меню автомобилей (рис. 6), поищем автомобиль по названию марки «Тесла»

Результат поиска видим на рисунке 8

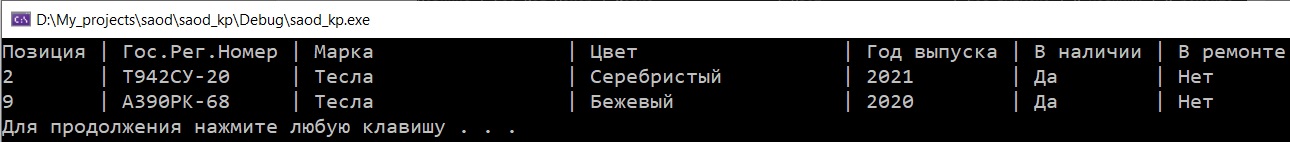


Рисунок 8

Нажимаем любую клавишу для продолжения, затем нажимаем «0» для выхода в главное меню (рис. 2), и переходим в раздел выдачи и возврата, Результат на рисунке 9

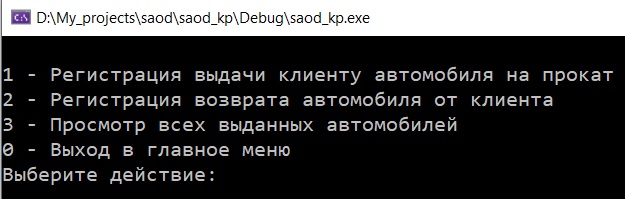


Рисунок 9

Нажимаем «1» для регистрации выдачи автомобиля

Выдадим автомобиль с номером «А000АА-00» клиенту с номером водительского удостоверения «00 АА 000000» автомобиль с 01.06.2021 по 10.06.2021, результат на рисунке10

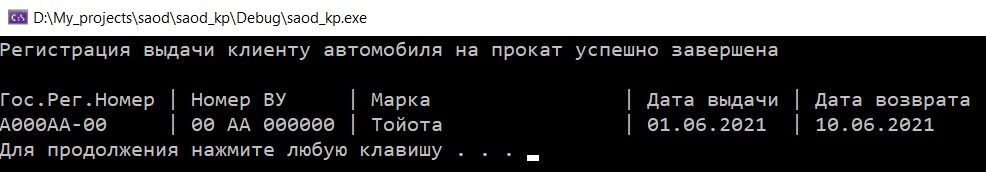


Рисунок 10

Просмотрим информацию об автомобиле с номером «А000АА-00» (рисунок 11)

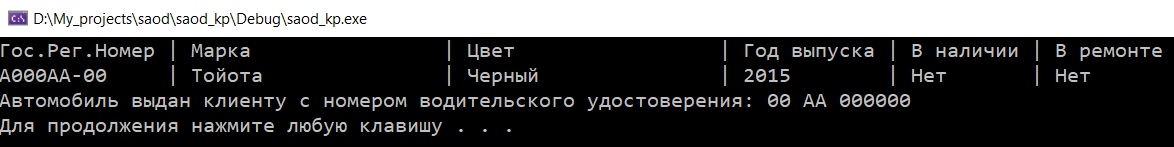


Рисунок 11

Просмотрим информацию о клиенте с номером «00 АА 000000» (рисунок 12)

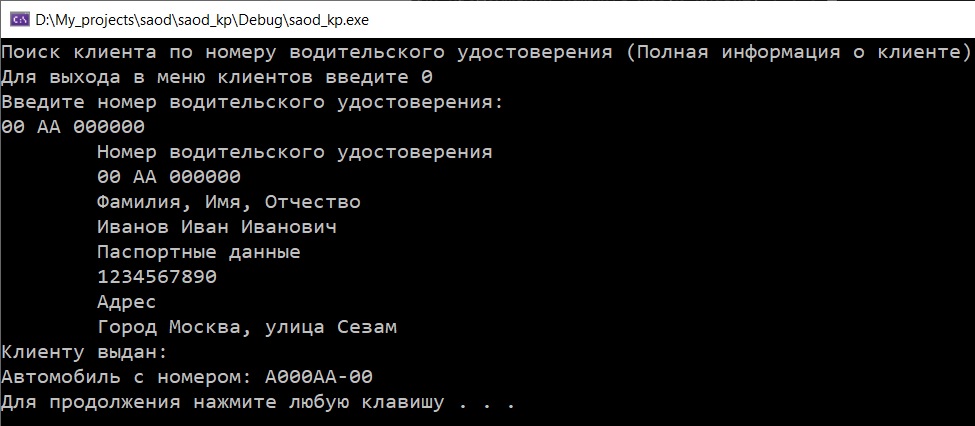


Рисунок 12

Как видим, при поиске информации о клиенте, попутно выходит информация о выданном ему автомобиле, а при поиске информации об автомобиле, выходит информация о клиенте, которому он выдан

# 6. Заключение

В процессе выполнения курсового проекта были изучены структуры данных и алгоритмы их обработки, а также получены практические навыки их использования при разработке программ.

К достоинствам программы можно отнести:

* Программа выполняет поставленную задачу и работает без ошибок
* При ручном вводе или загрузке данные проверяются во избежание ошибок
* Интерфейс удобен и понятен для массового пользователя
* Данные выводятся на экран в виде ровной таблицы, что удобно для чтения, все столбцы подписаны сверху
* Каждый пункт меню реализован в виде отдельной функции, что позволяет использовать их в других проектах и удобно ориентироваться в коде.
* Проект разбит на несколько файлов, в каждом из которых реализованы функции соответствующего подменю.

Недостатков обнаружено не было.

# Список использованной литературы

*1. Алгоритмы и структуры данных: учеб. пособие / В. А. Матьяш, С. А. Рогачев. – СПб.: ГУАП, 2021. – 71 с*

*2. Ключарев А.А., Матьяш В.А., Щекин С.В. Структуры и алгоритмы обработки данных: Учебное пособие / СПбГУАП. СПб., 2004.*

*3. Колдаев В.Д. Основы алгоритмизации и программирования: Учебное пособие / Колдаев В.Д; Под ред. проф.Л.Г. Гагариной - М.: ИД ФОРУМ, НИЦ ИНФРА-М, 2016. - 416 с.* [*http://znanium.com/catalog.php?bookinfo=537513*](http://znanium.com/catalog.php?bookinfo=537513)

*4. Павловская Т. А. C/C++. Программирование на языке высокого уровня: учебник. СПб. : ПИТЕР, 2007. - 461 с*

*5. Кузин А. В. Чумакова Е. В. Программирование на языке Си. М.: Форум, НИЦ ИНФРА-М, 2015. - 144 с.* [*http://znanium.com/catalog.php?bookinfo=505194*](http://znanium.com/catalog.php?bookinfo=505194)

*6. Страуструп, Б. Язык программирования C++ [Текст] = The C++ Programming Language : специальное издание / Б. Страуструп ; пер.: С. Анисимов, М. Кононов ; ред.: Ф. Андреев, А. Ушаков. - [Б. м.] : Бином-Пресс, 2008. - 1098 с.*

*7. Кнут, Д. Искусство программирования [Текст] = The art of computer programming : [в 3 т.]. Т. 1. Основные алгоритмы / Д. Кнут ; ред. Ю. В. Козаченко. - 3-е изд. - М. : Вильямс, 2014. - 720 с.*

*8. Демидович, Е.М. Основы алгоритмизации и программирования. Язык СИ [Текст] : учебное пособие / Е. М. Демидович. - 2-е изд., испр. и доп. - СПб. : БХВ - Петербург, 2008. - 440 с.*

*9. Вирт, Н Алгоритмы и структуры данных. Новая версия для Оберона + CD [Текст] / Н. Вирт ; пер. Д. Б. Подшивалов. - 2-е изд., испр. - М. : ДМК Пресс, 2012. - 272 с.*

# ПРИЛОЖЕНИЕ 1. Текст программы

### Файл saod\_kp.cpp

using namespace std;

#include <iostream>

#include <Windows.h>

#include <string>

#include "clients.h"

#include "cars.h"

#include "take\_return.h"

struct clients

{

string num\_VU; // номер водительского удостоверения

string FIO; // Фамилия имя отчество

string passport;

string address;

clients\* Left; //указатели на соседние ветви

clients\* Right;

short int h;// высота поддерева

};

struct cars

{

string grn; // гос регистрационный номер

string mark; // марка авто

string color;

int year;

bool have;// имеется ли в наличии

bool rem;

};

struct take\_return

{

string num\_VU; // номер водительского удостоверения

string grn; // гос регистрационный номер

string mark; // марка авто

string date\_of\_issue; // дата выдачи

string return\_date; // дата возврата

take\_return\* next;

};

int menu\_client() {

int num = -1;

bool err\_write = false;

string digit;

cout << endl <<

"1 - Регистрация нового клиента" << endl <<

"2 - Снятие с обслуживания клиента" << endl <<

"3 - Просмотр всех зарегистрированных клиентов" << endl <<

"4 - Очистка данных о клиентах" << endl <<

"5 - Поиск клиента по номеру водительского удостоверения (Полная информация о клиенте)" << endl <<

"6 - Поиск клиента по фрагментам ФИО или адреса" << endl <<

"0 - Выход в главное меню\nВыберите действие: " << endl;

cin.clear();

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

cin >> digit;

int digit\_lenght = digit.length();

for (int i = 0; i < digit\_lenght; i++) {

if (digit[i] < 48 || digit[i] > 57) {

err\_write = true;

}

}

if (!err\_write) {

num = stoi(digit);

}

while (cin.fail() || (num < 0) || (num > 6)) {

cin.clear();

cin.sync();

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

err\_write = false;

cin >> digit;

digit\_lenght = digit.length();

for (int i = 0; i < digit\_lenght; i++) {

if (digit[i] < 48 || digit[i] > 57) {

err\_write = true;

}

}

if (!err\_write) {

num = stoi(digit);

}

}

return num;

}

int menu\_cars() {

int num = -1;

bool err\_write = false;

string digit;

cout << endl <<

"1 - Добавление нового автомобиля" << endl <<

"2 - Удаление сведений об автомобиле" << endl <<

"3 - Просмотр всех имеющихся автомобилей" << endl <<

"4 - Очистка данных об автомобилях" << endl <<

"5 - Поиск автомобиля по государственному регистрационному номеру" << endl <<

"6 - Поиск автомобиля по названию марки автомобиля" << endl <<

"7 - Регистрация отправки автомобиля в ремонт" << endl <<

"8 - Регистрация прибытия автомобиля из ремонта" << endl <<

"0 - Выход в главное меню\nВыберите действие: " << endl;

cin.clear();

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

cin >> digit;

int digit\_lenght = digit.length();

for (int i = 0; i < digit\_lenght; i++) {

if (digit[i] < 48 || digit[i] > 57) {

err\_write = true;

}

}

if (!err\_write) {

num = stoi(digit);

}

while (cin.fail() || (num < 0) || (num > 8)) {

cin.clear();

cin.sync();

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

err\_write = false;

cin >> digit;

digit\_lenght = digit.length();

for (int i = 0; i < digit\_lenght; i++) {

if (digit[i] < 48 || digit[i] > 57) {

err\_write = true;

}

}

if (!err\_write) {

num = stoi(digit);

}

}

return num;

}

int menu\_t\_r() {

int num = -1;

bool err\_write = false;

string digit;

cout << endl <<

"1 - Регистрация выдачи клиенту автомобиля на прокат" << endl <<

"2 - Регистрация возврата автомобиля от клиента" << endl <<

"3 - Просмотр всех выданных автомобилей" << endl <<

"0 - Выход в главное меню\nВыберите действие: " << endl;

cin.clear();

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

cin >> digit;

int digit\_lenght = digit.length();

for (int i = 0; i < digit\_lenght; i++) {

if (digit[i] < 48 || digit[i] > 57) {

err\_write = true;

}

}

if (!err\_write) {

num = stoi(digit);

}

while (cin.fail() || (num < 0) || (num > 3)) {

cin.clear();

cin.sync();

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

err\_write = false;

cin >> digit;

digit\_lenght = digit.length();

for (int i = 0; i < digit\_lenght; i++) {

if (digit[i] < 48 || digit[i] > 57) {

err\_write = true;

}

}

if (!err\_write) {

num = stoi(digit);

}

}

return num;

}

int menu() {

int num = -1;

bool err\_write = false;

string digit;

cout << endl <<

"Главное меню: " << endl <<

"1 - Клиенты" << endl <<

"2 - Автомобили" << endl <<

"3 - Выдача и возврат" << endl <<

"0 - Выход\nВыберите пункт меню: " << endl;

cin.clear();

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

cin >> digit;

int digit\_lenght = digit.length();

for (int i = 0; i < digit\_lenght; i++) {

if (digit[i] < 48 || digit[i] > 57) {

err\_write = true;

}

}

if (!err\_write) {

num = stoi(digit);

}

while (cin.fail() || (num < 0) || (num > 3)) {

cin.clear();

cin.sync();

cout << "Ошибка ввода, выберите пункт меню: ";

err\_write = false;

cin >> digit;

digit\_lenght = digit.length();

for (int i = 0; i < digit\_lenght; i++) {

if (digit[i] < 48 || digit[i] > 57) {

err\_write = true;

}

}

if (!err\_write) {

num = stoi(digit);

}

}

return num;

}

void download\_test(clients\* client, clients\*& Root, short int& H, cars car, cars\* (&hash\_table), int& table\_size, int& count\_cars) {

client->num\_VU = "00 АА 000000";

client->FIO = "Иванов Иван Иванович";

client->passport = "1234567890";

client->address = "Город Москва, улица Сезам";

client->Left = NULL;

client->Right = NULL;

client->h = 1;

Add\_balance(client, Root, H);

client->num\_VU = "59 АХ 123456";

client->FIO = "Иванов Петр Петрович";

client->passport = "1231231231";

client->address = "Город Москва, улица Московская 1";

Add\_balance(client, Root, H);

client->num\_VU = "67 ТУ 113466";

client->FIO = "Иванов Петр Петрович";

client->passport = "1231231231";

client->address = "Город Москва, улица Московская 1";

Add\_balance(client, Root, H);

client->num\_VU = "01 ВЕ 123123";

client->FIO = "Сидоров Петр Петрович";

client->passport = "2233445566";

client->address = "Город Москва, улица Московская 10";

Add\_balance(client, Root, H);

client->num\_VU = "19 ММ 122112";

client->FIO = "Петров Петр Петрович";

client->passport = "1231231231";

client->address = "Город Москва, улица Московская 1";

Add\_balance(client, Root, H);

car.grn = "Е769СК-01";

car.mark = "ВАЗ 2110";

car.color = "Белый";

car.year = 2000;

car.have = true;

car.rem = false;

add\_car\_check(car, hash\_table, table\_size, count\_cars);

car.grn = "Х407АК-99";

car.mark = "Газ";

car.color = "Серый";

car.year = 1999;

add\_car\_check(car, hash\_table, table\_size, count\_cars);

car.grn = "А390РК-68";

car.mark = "Тесла";

car.color = "Бежевый";

car.year = 2020;

add\_car\_check(car, hash\_table, table\_size, count\_cars);

car.grn = "Т942СУ-20";

car.mark = "Тесла";

car.color = "Серебристый";

car.year = 2021;

add\_car\_check(car, hash\_table, table\_size, count\_cars);

car.grn = "А000АА-00";

car.mark = "Тойота";

car.color = "Черный";

car.year = 2015;

add\_car\_check(car, hash\_table, table\_size, count\_cars);

}

int main()

{

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

setlocale(LC\_ALL, "rus");

clients\* Root = NULL;

clients\* client = new clients;

clients\* search\_client = NULL; // указатель для поиска

short int H = 0;// высота дерева клиентов

string num\_VU = "";

string grn = "";

string fragment = "";

cars car;

int table\_size = 10, count\_cars = 0; // размер таблицы и общее количество автомобилей

int kol\_t\_r = 0; // количество выданных автомобилей

cars\* hash\_table = new cars[table\_size];

for (int i = 0; i < table\_size; i++) {

hash\_table[i].grn = "";

}

take\_return\* head = NULL;// указатель на начало списка

take\_return\* search\_t\_r = NULL; // указатель для поиска

take\_return\* t\_r = new take\_return;

string test = "";

cout << "Для загрузки тестовых данных введите 1: ";

getline(cin, test);

if (test == "1") download\_test(client, Root, H, car, hash\_table, table\_size, count\_cars);

system("cls"); // очистка консоли

int n = menu(); //меню пользователя

while (n != 0) {

switch (n) {

//клиенты

case 1: {

system("cls"); // очистка консоли

int n\_cl = menu\_client();

while (n\_cl != 0) {

switch (n\_cl) {

// добавить

case 1: {

system("cls"); // очистка консоли

cout << "Регистрация нового клиента" << endl;

cout << "Введите номер водительского удостоверения" << endl;

cout << "строка формата \"RR AA NNNNNN\", где RR – код региона(цифры)" << endl;

cout << "AA – серия(буквы из следующего множества : А, В, Е, К, М, Н, О, Р, С, Т, У, Х)" << endl;

cout << "NNNNNN – порядковый номер удостоверения(цифры)" << endl;

cout << "Код, серия и номер отделяются друг от друга пробелами " << endl;

cout << "Для выхода в меню клиентов введите 0 " << endl;

cout << "Введите номер водительского удостоверения: " << endl;

cin.clear();

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

getline(cin, num\_VU);

if (num\_VU == "0") {

system("cls");

n\_cl = menu\_client();

break;

}

search\_client = NULL;

bool check\_have\_VU = check\_num\_VU(num\_VU);

if (check\_have\_VU) {

search\_cl(Root, num\_VU, search\_client);

}

while ((!check\_have\_VU || search\_client) && num\_VU != "0") {

system("cls");

cout << "Ошибка ввода!" << endl;

if (search\_client) {

cout << "Клиент с данным номером водительского удостоверения уже зарегистрирован!" << endl;

}

cout << "Регистрация нового клиента" << endl;

cout << "Введите номер водительского удостоверения" << endl;

cout << "строка формата \"RR AA NNNNNN\", где RR – код региона(цифры)" << endl;

cout << "AA – серия(буквы из следующего множества : А, В, Е, К, М, Н, О, Р, С, Т, У, Х)" << endl;

cout << "NNNNNN – порядковый номер удостоверения(цифры)" << endl;

cout << "Код, серия и номер отделяются друг от друга пробелами " << endl;

cout << "Для выхода в меню клиентов введите 0 " << endl;

cout << "Введите номер водительского удостоверения: " << endl;

cin.clear();

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

getline(cin, num\_VU);

search\_client = NULL;

check\_have\_VU = check\_num\_VU(num\_VU);

if (check\_have\_VU) {

search\_cl(Root, num\_VU, search\_client);

}

}

if (num\_VU == "0") {

system("cls");

n\_cl = menu\_client();

break;

}

client->num\_VU = num\_VU;

system("cls");

cout << "Для выхода в меню клиентов введите 0 " << endl;

cout << "Номер водительского удостоверения: " << num\_VU << endl;

cout << "Введите ФИО клиента: " << endl;

cin.clear();

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

getline(cin, client->FIO);

if (client->FIO == "0") {

system("cls");

n\_cl = menu\_client();

break;

}

while ((client->FIO.length() > 50 || client->FIO.length() == 0) && client->FIO != "0") {

system("cls");

cout << "Ошибка ввода!" << endl;

cout << "ФИО не может быть пустым и должно содержать до 50 символов" << endl;

cout << "Для выхода в меню клиентов введите 0 " << endl;

cout << "Номер водительского удостоверения: " << num\_VU << endl;

cout << "Введите ФИО клиента: " << endl;

cin.clear();

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

getline(cin, client->FIO);

}

if (client->FIO == "0") {

system("cls");

n\_cl = menu\_client();

break;

}

system("cls");

cout << "Для выхода в меню клиентов введите 0 " << endl;

cout << "Номер водительского удостоверения: " << num\_VU << endl;

cout << "ФИО клиента: " << client->FIO << endl;

cout << "Введите паспортные данные клиента, 10 цифр без пробела: " << endl;

cin.clear();

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

getline(cin, client->passport);

if (client->passport == "0") {

system("cls");

n\_cl = menu\_client();

break;

}

while (!check\_passport(client->passport) && client->passport != "0") {

system("cls");

cout << "Ошибка ввода!" << endl;

cout << "Для выхода в меню клиентов введите 0 " << endl;

cout << "Номер водительского удостоверения: " << num\_VU << endl;

cout << "ФИО клиента: " << client->FIO << endl;

cout << "Введите паспортные данные клиента, 10 цифр без пробела: " << endl;

cin.clear();

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

getline(cin, client->passport);

}

if (client->passport == "0") {

system("cls");

n\_cl = menu\_client();

break;

}

system("cls");

cout << "Для выхода в меню клиентов введите 0 " << endl;

cout << "Номер водительского удостоверения: " << num\_VU << endl;

cout << "ФИО клиента: " << client->FIO << endl;

cout << "Паспортные данные клиента: " << client->passport << endl;

cout << "Введите адрес клиента: " << endl;

cin.clear();

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

getline(cin, client->address);

if (client->address == "0") {

system("cls");

n\_cl = menu\_client();

break;

}

while ((client->address.length() > 150 || client->address.length() == 0) && client->address != "0") {

system("cls");

cout << "Ошибка ввода!" << endl;

cout << "Адрес клиента не может быть пустым и должен содержать до 150 символов: " << endl;

cout << "Для выхода в меню клиентов введите 0 " << endl;

cout << "Номер водительского удостоверения: " << num\_VU << endl;

cout << "ФИО клиента: " << client->FIO << endl;

cout << "Паспортные данные клиента: " << client->passport << endl;

cout << "Введите адрес клиента: " << endl;

cin.clear();

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

getline(cin, client->address);

}

if (client->address == "0") {

system("cls");

n\_cl = menu\_client();

break;

}

client->Left = NULL;

client->Right = NULL;

client->h = 1;

Add\_balance(client, Root, H);

system("cls");

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

cout << "Просмотр всех зарегистрированных клиентов" << endl;

cout << "\tНомер ВУ\tФИО" << endl;

print\_all\_clients(Root);

cout << "Вывод бинарного дерева: " << endl;

cout << "Высота дерева = " << H << endl;

print\_client(Root);

system("pause");

system("cls");

n\_cl = menu\_client();

break;

}

// удалить

case 2: {

system("cls"); // очистка консоли

if (H == 0) {

cout << "База клиентов пуста" << endl;

system("pause");

system("cls");

n\_cl = menu\_client();

break;

}

else {

cout << "\tНомер ВУ\tФИО" << endl;

print\_all\_clients(Root);

cout << "Снятие с обслуживания клиента" << endl;

cout << "Для выхода в меню клиентов введите 0 " << endl;

cout << "Введите номер водительского удостоверения: " << endl;

cin.clear();

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

getline(cin, num\_VU);

if (num\_VU == "0") {

system("cls");

n\_cl = menu\_client();

break;

}

search\_client = NULL;

bool check\_have\_VU = check\_num\_VU(num\_VU);

if (check\_have\_VU) {

search\_cl(Root, num\_VU, search\_client);

}

while ((!check\_have\_VU || !search\_client) && num\_VU != "0") {

system("cls"); // очистка консоли

cout << "Ошибка ввода!" << endl;

if (!search\_client && check\_have\_VU) {

cout << "Клиент с данным номером водительского удостоверения не найден!" << endl;

}

cout << "Снятие с обслуживания клиента" << endl;

cout << "Введите номер водительского удостоверения";

cout << "строка формата RR AA NNNNNN, где RR – код региона(цифры)" << endl;

cout << "AA – серия(буквы из следующего множества : А, В, Е, К, М, Н, О, Р, С, Т, У, Х)" << endl;

cout << "NNNNNN – порядковый номер удостоверения(цифры)" << endl;

cout << "Код, серия и номер отделяются друг от друга пробелами " << endl;

cout << "\tНомер ВУ\tФИО" << endl;

print\_all\_clients(Root);

cout << "Для выхода в меню клиентов введите 0 " << endl;

cout << "Введите номер водительского удостоверения: " << endl;

cin.clear();

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

getline(cin, num\_VU);

search\_client = NULL;

check\_have\_VU = check\_num\_VU(num\_VU);

if (check\_have\_VU) {

search\_cl(Root, num\_VU, search\_client);

}

}

if (num\_VU == "0") {

system("cls");

n\_cl = menu\_client();

break;

}

search\_t\_r = search\_tr(head, num\_VU);

if (search\_t\_r) {

cout << "Клиент не вернул: " << endl;

print\_cars\_of\_client(head, num\_VU);

}

else {

del\_check\_H(Root, num\_VU, H);

system("cls");

cout << "Клиент с номером водительского удостоверения: " << num\_VU << " снят с обслуживания!" << endl;

if (H == 0) {

cout << "База клиентов пуста" << endl;

}

else {

cout << "Просмотр всех зарегистрированных клиентов" << endl;

cout << "\tНомер ВУ\tФИО" << endl;

print\_all\_clients(Root);

cout << "Вывод бинарного дерева: " << endl;

cout << "Высота дерева = " << H << endl;

print\_client(Root);

}

}

system("pause");

system("cls");

n\_cl = menu\_client();

break;

}

}

// просмотр всех

case 3: {

system("cls"); // очистка консоли

if (H == 0) {

cout << "База клиентов пуста" << endl;

system("pause");

system("cls");

n\_cl = menu\_client();

break;

}

else {

cout << "Просмотр всех зарегистрированных клиентов" << endl;

cout << "\tНомер ВУ\tФИО" << endl;

print\_all\_clients(Root);

cout << "Вывод бинарного дерева: " << endl;

cout << "Высота дерева = " << H << endl;

print\_client(Root);

system("pause");

system("cls");

n\_cl = menu\_client();

break;

}

}

// очистка всех

case 4: {

system("cls"); // очистка консоли

if (H == 0) {

cout << "База клиентов пуста" << endl;

system("pause");

system("cls");

n\_cl = menu\_client();

break;

}

else {

string check = "";

if (head) {

cout << "Существуют клиенты, которым выдан автомобиль: " << endl;

print\_all\_t\_r(head);

cout << "Подтвердите выбранное действие, введите 1" << endl;

cout << "Для выхода в меню клиентов введите 0 " << endl;

cin.clear();

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

getline(cin, check);

if (check == "0") {

system("cls");

n\_cl = menu\_client();

break;

}

if (check == "1") {

del\_except\_t\_r(Root, H, head);

cout << "Данные об остальных клиентах очищены" << endl;

}

}

else {

cout << "Подтвердите выбранное действие, введите 1" << endl;

cout << "Для выхода в меню клиентов введите 0 " << endl;

cin.clear();

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

getline(cin, check);

if (check == "0") {

system("cls");

n\_cl = menu\_client();

break;

}

if (check == "1") {

del\_except\_t\_r(Root, H, head);

cout << "Данные о клиентах очищены" << endl;

}

}

system("pause");

system("cls");

n\_cl = menu\_client();

break;

}

}

//Поиск клиента по номеру водительского удостоверения (Полная информация о клиенте)

case 5: {

system("cls"); // очистка консоли

if (H == 0) {

cout << "База клиентов пуста" << endl;

system("pause");

system("cls");

n\_cl = menu\_client();

break;

}

else {

cout << "Поиск клиента по номеру водительского удостоверения (Полная информация о клиенте)" << endl;

cout << "Для выхода в меню клиентов введите 0 " << endl;

cout << "Введите номер водительского удостоверения: " << endl;

cin.clear();

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

getline(cin, num\_VU);

if (num\_VU == "0") {

system("cls");

n\_cl = menu\_client();

break;

}

search\_client = NULL;

bool check\_have\_VU = check\_num\_VU(num\_VU);

if (check\_have\_VU) {

search\_cl(Root, num\_VU, search\_client);

}

while ((!check\_have\_VU || !search\_client) && num\_VU != "0") {

system("cls"); // очистка консоли

if (!check\_have\_VU) cout << "Ошибка ввода!" << endl;

if (!search\_client && check\_have\_VU) {

cout << "Клиент с данным номером водительского удостоверения не найден!" << endl;

}

cout << "Поиск клиента по номеру водительского удостоверения (Полная информация о клиенте)" << endl;

cout << "Для выхода в меню клиентов введите 0 " << endl;

cout << "Введите номер водительского удостоверения: ";

cout << "строка формата RR AA NNNNNN, где RR – код региона(цифры)" << endl;

cout << "AA – серия(буквы из следующего множества : А, В, Е, К, М, Н, О, Р, С, Т, У, Х)" << endl;

cout << "NNNNNN – порядковый номер удостоверения(цифры)" << endl;

cout << "Код, серия и номер отделяются друг от друга пробелами " << endl;

cout << "Для выхода в меню клиентов введите 0 " << endl;

cout << "Введите номер водительского удостоверения: " << endl;

cin.clear();

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

getline(cin, num\_VU);

search\_client = NULL;

check\_have\_VU = check\_num\_VU(num\_VU);

if (check\_have\_VU) {

search\_cl(Root, num\_VU, search\_client);

}

}

if (num\_VU == "0") {

system("cls");

n\_cl = menu\_client();

break;

}

print\_client\_complete(search\_client);

search\_t\_r = search\_tr(head, num\_VU);

if (search\_t\_r) {

cout << "Клиенту выдан: " << endl;

print\_cars\_of\_client(head, num\_VU);

}

system("pause");

system("cls");

n\_cl = menu\_client();

break;

}

}

// Поиск клиента по фрагментам ФИО или адреса

case 6: {

system("cls"); // очистка консоли

if (H == 0) {

cout << "База клиентов пуста" << endl;

system("pause");

system("cls");

n\_cl = menu\_client();

break;

}

else {

cout << "Поиск клиента по фрагментам ФИО или адреса" << endl;

cout << "Для выхода в меню клиентов введите 0 " << endl;

cout << "Введите фрагмент: " << endl;

cin.clear();

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

getline(cin, fragment);

if (fragment == "0") {

system("cls");

n\_cl = menu\_client();

break;

}

while ((fragment.length() > 30 || fragment.length() == 0) && fragment != "0") {

system("cls"); // очистка консоли

cout << "Ошибка ввода!" << endl;

cout << "Фрагмент не может быть пустым и должен содержать до 30 символов" << endl;

cout << "Для выхода в меню автомобилей введите 0 " << endl;

cout << "Введите фрагмент: " << endl;

cin.clear();

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

getline(cin, fragment);

}

if (fragment == "0") {

system("cls");

n\_cl = menu\_client();

break;

}

bool header = false;

search\_fragment(Root, fragment, header);

if (!header) {

cout << "Совпадений не найдено" << endl;

}

system("pause");

system("cls");

n\_cl = menu\_client();

break;

}

}

}

}

system("cls");

n = menu(); //главное меню

break;

}

// автомобили

case 2: {

system("cls"); // очистка консоли

int n\_car = menu\_cars();

while (n\_car != 0) {

switch (n\_car) {

// Добавление нового автомобиля

case 1: {

system("cls"); // очистка консоли

cout << "Добавление нового автомобиля" << endl;

cout << "Введите государственный регистрационный номер" << endl;

cout << "строка формата \"ANNNAA-NN\", где N –цифра" << endl;

cout << "A – буква из следующего множества : А, В, Е, К, М, Н, О, Р, С, Т, У, Х" << endl;

cout << "Для выхода в меню автомобилей введите 0 " << endl;

cout << "Введите государственный регистрационный номер: " << endl;

cin.clear();

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

getline(cin, grn);

if (grn == "0") {

system("cls");

n\_car = menu\_cars();

break;

};

bool check\_num\_car = check\_grn(grn);

bool check\_have\_car = false;

if (check\_num\_car) {

check\_have\_car = search\_car(hash\_table, grn, car, table\_size);

}

while ((check\_have\_car || !check\_num\_car) && grn != "0") {

system("cls");

cout << "Ошибка ввода!" << endl;

if (check\_have\_car) {

cout << "Автомобиль с данным государственным регистрационным номером уже зарегистрирован!" << endl;

}

cout << "Добавление нового автомобиля" << endl;

cout << "Введите государственный регистрационный номер" << endl;

cout << "строка формата \"ANNNAA-NN\", где N –цифра" << endl;

cout << "A – буква из следующего множества : А, В, Е, К, М, Н, О, Р, С, Т, У, Х" << endl;

cout << "Для выхода в меню автомобилей введите 0 " << endl;

cout << "Введите государственный регистрационный номер: " << endl;

cin.clear();

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

getline(cin, grn);

check\_have\_car = false;

check\_num\_car = check\_grn(grn);

if (check\_num\_car) {

check\_have\_car = search\_car(hash\_table, grn, car, table\_size);

}

}

if (grn == "0") {

system("cls");

n\_car = menu\_cars();

break;

}

car.grn = grn;

system("cls"); // очистка консоли

cout << "Для выхода в меню автомобилей введите 0 " << endl;

cout << "Государственный регистрационный номер: " << car.grn << endl;

cout << "Введите марку автомобиля: " << endl;

cin.clear();

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

getline(cin, car.mark);

if (car.mark == "0") {

system("cls");

n\_car = menu\_cars();

break;

}

while ((car.mark.length() > 20 || car.mark.length() == 0) && car.mark != "0") {

system("cls"); // очистка консоли

cout << "Ошибка ввода!" << endl;

cout << "Марка не может быть пустой и должна содержать до 20 символов" << endl;

cout << "Для выхода в меню автомобилей введите 0 " << endl;

cout << "Государственный регистрационный номер: " << car.grn << endl;

cout << "Введите марку автомобиля: " << endl;

cin.clear();

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

getline(cin, car.mark);

}

if (car.mark == "0") {

system("cls");

n\_car = menu\_cars();

break;

}

system("cls"); // очистка консоли

cout << "Для выхода в меню автомобилей введите 0 " << endl;

cout << "Государственный регистрационный номер: " << car.grn << endl;

cout << "Марка автомобиля: " << car.mark << endl;

cout << "Введите цвет автомобиля: " << endl;

cin.clear();

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

getline(cin, car.color);

if (car.color == "0") {

system("cls");

n\_car = menu\_cars();

break;

}

while ((car.color.length() > 20 || car.color.length() == 0) && car.color != "0") {

system("cls"); // очистка консоли

cout << "Ошибка ввода!" << endl;

cout << "Цвет не может быть пустым и должен содержать до 20 символов" << endl;

cout << "Для выхода в меню клиентов введите 0 " << endl;

cout << "Государственный регистрационный номер: " << car.grn << endl;

cout << "Марка автомобиля: " << car.mark << endl;

cout << "Введите цвет автомобиля: " << endl;

cin.clear();

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

getline(cin, car.color);

}

if (car.color == "0") {

system("cls");

n\_car = menu\_cars();

break;

}

system("cls"); // очистка консоли

cout << "Для выхода в меню клиентов введите 0 " << endl;

cout << "Государственный регистрационный номер: " << car.grn << endl;

cout << "Марка автомобиля: " << car.mark << endl;

cout << "Цвет автомобиля: " << car.color << endl;

string year = "";

SYSTEMTIME st;

GetSystemTime(&st);

int wYear = st.wYear;

cout << "Введите год выпуска автомобиля: " << endl;

cin.clear();

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

getline(cin, year);

if (year == "0") {

system("cls");

n\_car = menu\_cars();

break;

}

bool true\_year = check\_year(year, wYear);

while (!true\_year && year != "0") {

system("cls"); // очистка консоли

cout << "Ошибка ввода!" << endl;

cout << "Год может принимать значение от 1800 до " << wYear << endl;

cout << "Для выхода в меню клиентов введите 0 " << endl;

cout << "Государственный регистрационный номер: " << car.grn << endl;

cout << "Марка автомобиля: " << car.mark << endl;

cout << "Цвет автомобиля: " << car.color << endl;

cout << "Введите год выпуска автомобиля: " << endl;

cin.clear();

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

getline(cin, year);

true\_year = check\_year(year, wYear);

}

if (year == "0") {

system("cls");

n\_car = menu\_cars();

break;

}

car.year = stoi(year);

car.have = true;

car.rem = false;

add\_car\_check(car, hash\_table, table\_size, count\_cars);

system("cls");

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

print\_cars(hash\_table, table\_size, count\_cars);

system("pause");

system("cls");

n\_car = menu\_cars();

break;

}

// Удаление сведений об автомобиле

case 2: {

system("cls"); // очистка консоли

if (count\_cars == 0) {

cout << "База автомобилей пуста" << endl;

system("pause");

system("cls");

n\_car = menu\_cars();

break;

}

else {

cout << "Удаление сведений об автомобиле" << endl;

cout << "Для выхода в меню автомобилей введите 0" << endl;

cout << "Введите государственный регистрационный номер: ";

cin.clear();

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

getline(cin, grn);

if (grn == "0") {

system("cls");

n\_car = menu\_cars();

break;

}

bool check\_num\_car = check\_grn(grn);

bool check\_have\_car = false;

if (check\_num\_car) {

check\_have\_car = search\_car(hash\_table, grn, car, table\_size);

}

while ((!check\_have\_car || !check\_num\_car) && grn != "0") {

system("cls"); // очистка консоли

cout << "Ошибка ввода!" << endl;

if (!check\_have\_car && check\_num\_car) {

cout << "Автомобиль с данным государственным регистрационным номером не найден!" << endl;

}

cout << "Удаление сведений об автомобиле" << endl;

cout << "Введите государственный регистрационный номер" << endl;

cout << "строка формата \"ANNNAA-NN\", где N –цифра" << endl;

cout << "A – буква из следующего множества : А, В, Е, К, М, Н, О, Р, С, Т, У, Х" << endl;

cout << "Для выхода в меню автомобилей введите 0" << endl;

cout << "Введите государственный регистрационный номер: ";

cin.clear();

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

getline(cin, grn);

check\_have\_car = false;

check\_num\_car = check\_grn(grn);

if (check\_num\_car) {

check\_have\_car = search\_car(hash\_table, grn, car, table\_size);

}

}

if (grn == "0") {

system("cls");

n\_car = menu\_cars();

break;

}

system("cls");

if (!car.have) {

cout << "Данного автомобиля нет в наличии" << endl;

if (car.rem) {

cout << "Автомобиль в ремонте" << endl;

}

search\_t\_r = search\_tr(head, grn);

if (search\_t\_r) {

cout << "Автомобиль выдан клиенту с номером водительского удостоверения: " << search\_t\_r->num\_VU << endl;

}

}

else {

del\_car(hash\_table, grn, table\_size, count\_cars);

cout << "Сведения об втомобиле с государственным регистрационным номером: " << grn << " удалены!" << endl;

}

system("pause");

system("cls");

n\_car = menu\_cars();

break;

}

}

// Просмотр всех имеющихся автомобилей

case 3: {

system("cls"); // очистка консоли

if (count\_cars == 0) {

cout << "База автомобилей пуста" << endl;

system("pause");

system("cls");

n\_car = menu\_cars();

break;

}

else {

cout << "Просмотр всех имеющихся автомобилей" << endl;

print\_cars(hash\_table, table\_size, count\_cars);

system("pause");

system("cls");

n\_car = menu\_cars();

break;

}

}

//Очистка данных об автомобилях

case 4: {

system("cls"); // очистка консоли

if (count\_cars == 0) {

cout << "База автомобилей пуста" << endl;

system("pause");

system("cls");

n\_car = menu\_cars();

break;

}

else {

string check = "";

cout << "Очистка данных об автомобилях" << endl;

cout << "Подтвердите выбранное действие, введите 1" << endl;

cout << "Для выхода в меню автомобилей введите 0 " << endl;

cin.clear();

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

getline(cin, check);

if (check == "0") {

system("cls");

n\_car = menu\_cars();

break;

}

if (check == "1") {

del\_car\_have(hash\_table, table\_size, count\_cars);

if (count\_cars) {

cout << "Данные очищены кроме автомобилей, которых нет в наличии" << endl;

}

else cout << "Данные об автомобилях полностью очищены" << endl;

}

system("pause");

system("cls");

n\_car = menu\_cars();

break;

}

}

//Поиск автомобиля по государственному регистрационному номеру

case 5:{

system("cls"); // очистка консоли

if (count\_cars == 0) {

cout << "База автомобилей пуста" << endl;

system("pause");

system("cls");

n\_car = menu\_cars();

break;

}

else {

cout << "Поиск автомобиля по государственному регистрационному номеру" << endl;

cout << "Для выхода в меню автомобилей введите 0" << endl;

cout << "Введите государственный регистрационный номер: ";

cin.clear();

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

getline(cin, grn);

if (grn == "0") {

system("cls");

n\_car = menu\_cars();

break;

}

bool check\_num\_car = check\_grn(grn);

bool check\_have\_car = false;

if (check\_num\_car) {

check\_have\_car = search\_car(hash\_table, grn, car, table\_size);

}

while ((!check\_have\_car || !check\_num\_car) && grn != "0") {

system("cls"); // очистка консоли

if (!check\_num\_car) cout << "Ошибка ввода!" << endl;

if (!check\_have\_car && check\_num\_car) {

cout << "Автомобиль с данным государственным регистрационным номером не найден!" << endl;

}

cout << "Поиск автомобиля по государственному регистрационному номеру" << endl;

cout << "Введите государственный регистрационный номер" << endl;

cout << "строка формата \"ANNNAA-NN\", где N –цифра" << endl;

cout << "A – буква из следующего множества : А, В, Е, К, М, Н, О, Р, С, Т, У, Х" << endl;

cout << "Для выхода в меню автомобилей введите 0" << endl;

cout << "Введите государственный регистрационный номер: ";

cin.clear();

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

getline(cin, grn);

check\_have\_car = false;

check\_num\_car = check\_grn(grn);

if (check\_num\_car) {

check\_have\_car = search\_car(hash\_table, grn, car, table\_size);

}

}

if (grn == "0") {

system("cls");

n\_car = menu\_cars();

break;

}

system("cls"); // очистка консоли

print\_car(car);

search\_t\_r = search\_tr(head, grn);

if (search\_t\_r) {

cout << "Автомобиль выдан клиенту с номером водительского удостоверения: " << search\_t\_r->num\_VU << endl;

}

system("pause");

system("cls");

n\_car = menu\_cars();

break;

}

}

//Поиск автомобиля по названию марки автомобиля

case 6: {

system("cls"); // очистка консоли

if (count\_cars == 0) {

cout << "База автомобилей пуста" << endl;

system("pause");

system("cls");

n\_car = menu\_cars();

break;

}

else {

string mark = "";

cout << "Поиск автомобиля по названию марки автомобиля" << endl;

cout << "Для выхода в меню автомобилей введите 0" << endl;

cout << "Введите марку автомобиля: ";

cin.clear();

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

getline(cin, mark);

if (mark == "0") {

system("cls");

n\_car = menu\_cars();

break;

}

while ((mark.length() > 30 || mark.length() == 0) && mark != "0") {

cout << "Ошибка ввода!" << endl;

cout << "Марка не может быть пустой и должна содержать до 30 символов" << endl;

cout << "Для выхода в меню автомобилей введите 0 " << endl;

cout << "Введите марку автомобиля: " << endl;

cin.clear();

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

getline(cin, mark);

}

if (mark == "0") {

system("cls");

n\_car = menu\_cars();

break;

}

system("cls"); // очистка консоли

search\_mark(hash\_table, mark, table\_size, head);

search\_t\_r = search\_tr(head, grn);

if (search\_t\_r) {

cout << "Автомобиль выдан клиенту с номером водительского удостоверения: " << search\_t\_r->num\_VU << endl;

}

system("pause");

system("cls");

n\_car = menu\_cars();

break;

}

}

//Регистрация отправки автомобиля в ремонт

case 7: {

system("cls"); // очистка консоли

if (count\_cars == 0) {

cout << "База автомобилей пуста" << endl;

system("pause");

system("cls");

n\_car = menu\_cars();

break;

}

else {

cout << "Регистрация отправки автомобиля в ремонт" << endl;

cout << "Для выхода в меню автомобилей введите 0" << endl;

cout << "Введите государственный регистрационный номер: ";

cin.clear();

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

getline(cin, grn);

if (grn == "0") {

system("cls");

n\_car = menu\_cars();

break;

}

bool check\_num\_car = check\_grn(grn);

bool check\_have\_car = false;

if (check\_num\_car) {

check\_have\_car = search\_car(hash\_table, grn, car, table\_size);

}

while ((!check\_have\_car || !check\_num\_car) && grn != "0") {

system("cls"); // очистка консоли

if (!check\_num\_car) cout << "Ошибка ввода!" << endl;

if (!check\_have\_car && check\_num\_car) {

cout << "Автомобиль с данным государственным регистрационным номером не найден!" << endl;

}

cout << "Регистрация отправки автомобиля в ремонт" << endl;

cout << "Введите государственный регистрационный номер" << endl;

cout << "строка формата \"ANNNAA-NN\", где N –цифра" << endl;

cout << "A – буква из следующего множества : А, В, Е, К, М, Н, О, Р, С, Т, У, Х" << endl;

cout << "Для выхода в меню автомобилей введите 0" << endl;

cout << "Введите государственный регистрационный номер: ";

cin.clear();

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

getline(cin, grn);

check\_have\_car = false;

check\_num\_car = check\_grn(grn);

if (check\_num\_car) {

check\_have\_car = search\_car(hash\_table, grn, car, table\_size);

}

}

if (grn == "0") {

system("cls");

n\_car = menu\_cars();

break;

}

reg\_send\_rem(hash\_table, grn, table\_size);

system("pause");

system("cls");

n\_car = menu\_cars();

break;

}

}

//Регистрация прибытия автомобиля из ремонта

case 8: {

system("cls"); // очистка консоли

if (count\_cars == 0) {

cout << "База автомобилей пуста" << endl;

system("pause");

system("cls");

n\_car = menu\_cars();

break;

}

else {

cout << "Регистрация прибытия автомобиля из ремонта" << endl;

cout << "Для выхода в меню автомобилей введите 0" << endl;

cout << "Введите государственный регистрационный номер: ";

cin.clear();

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

getline(cin, grn);

if (grn == "0") {

system("cls");

n\_car = menu\_cars();

break;

}

bool check\_num\_car = check\_grn(grn);

bool check\_have\_car = false;

if (check\_num\_car) {

check\_have\_car = search\_car(hash\_table, grn, car, table\_size);

}

while ((!check\_have\_car || !check\_num\_car) && grn != "0") {

system("cls"); // очистка консоли

if (!check\_num\_car) cout << "Ошибка ввода!" << endl;

if (!check\_have\_car && check\_num\_car) {

cout << "Автомобиль с данным государственным регистрационным номером не найден!" << endl;

}

cout << "Регистрация прибытия автомобиля из ремонта" << endl;

cout << "Введите государственный регистрационный номер" << endl;

cout << "строка формата \"ANNNAA-NN\", где N –цифра" << endl;

cout << "A – буква из следующего множества : А, В, Е, К, М, Н, О, Р, С, Т, У, Х" << endl;

cout << "Для выхода в меню автомобилей введите 0" << endl;

cout << "Введите государственный регистрационный номер: ";

cin.clear();

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

getline(cin, grn);

check\_have\_car = false;

check\_num\_car = check\_grn(grn);

if (check\_num\_car) {

check\_have\_car = search\_car(hash\_table, grn, car, table\_size);

}

}

if (grn == "0") {

system("cls");

n\_car = menu\_cars();

break;

}

reg\_get\_rem(hash\_table, grn, table\_size);

system("pause");

system("cls");

n\_car = menu\_cars();

break;

}

}

}

}

system("cls");

n = menu(); //главное меню

break;

}

// выдача и возврат

case 3: {

system("cls"); // очистка консоли

int n\_t\_r = menu\_t\_r();

while (n\_t\_r != 0) {

switch (n\_t\_r) {

//выдача

case 1: {

system("cls"); // очистка консоли

if (H == 0) {

cout << "База клиентов пуста" << endl;

system("pause");

system("cls");

n\_t\_r = menu\_t\_r();

break;

}

if (count\_cars == 0) {

cout << "База автомобилей пуста" << endl;

system("pause");

system("cls");

n\_t\_r = menu\_t\_r();

break;

}

cout << "Регистрация выдачи клиенту автомобиля на прокат" << endl;

cout << "Для выхода в меню выдачи и возврата введите 0" << endl;

cout << "Введите государственный регистрационный номер автомобиля: ";

cin.clear();

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

getline(cin, grn);

if (grn == "0") {

system("cls");

n\_t\_r = menu\_t\_r();

break;

}

bool check\_num\_car = check\_grn(grn);

bool check\_have\_car = false;

if (check\_num\_car) {

check\_have\_car = search\_car(hash\_table, grn, car, table\_size);

}

while ((!check\_have\_car || !check\_num\_car) && grn != "0") {

system("cls"); // очистка консоли

cout << "Ошибка ввода!" << endl;

if (!check\_have\_car && check\_num\_car) {

cout << "Автомобиль с данным государственным регистрационным номером не найден!" << endl;

}

cout << "Регистрация выдачи клиенту автомобиля на прокат" << endl;

cout << "Введите государственный регистрационный номер автомобиля" << endl;

cout << "строка формата \"ANNNAA-NN\", где N –цифра" << endl;

cout << "A – буква из следующего множества : А, В, Е, К, М, Н, О, Р, С, Т, У, Х" << endl;

cout << "Для выхода в меню выдачи и возврата введите 0" << endl;

cout << "Введите государственный регистрационный номер автомобиля: ";

cin.clear();

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

getline(cin, grn);

check\_have\_car = false;

check\_num\_car = check\_grn(grn);

if (check\_num\_car) {

check\_have\_car = search\_car(hash\_table, grn, car, table\_size);

}

}

if (grn == "0") {

system("cls");

n\_t\_r = menu\_t\_r();

break;

}

system("cls");

if (!car.have) {

cout << "Данного автомобиля нет в наличии" << endl;

if (car.rem) {

cout << "Автомобиль в ремонте" << endl;

}

search\_t\_r = search\_tr(head, grn);

if (search\_t\_r) {

cout << "Данный втомобиль уже выдан клиенту с номером водительского удостоверения: " << search\_t\_r->num\_VU << endl;

}

system("pause");

system("cls");

n\_t\_r = menu\_t\_r();

break;

}

t\_r->grn = grn;

system("cls"); // очистка консоли

cout << "Регистрация выдачи клиенту автомобиля на прокат" << endl;

cout << "Для выхода в меню выдачи и возврата введите 0" << endl;

cout << "Государственный регистрационный номер автомобиля: " << grn << endl;

cout << "Введите номер водительского удостоверения клиента: " << endl;

cin.clear();

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

getline(cin, num\_VU);

if (num\_VU == "0") {

system("cls");

n\_t\_r = menu\_t\_r();

break;

}

search\_client = NULL;

bool check\_have\_VU = check\_num\_VU(num\_VU);

if (check\_have\_VU) {

search\_cl(Root, num\_VU, search\_client);

}

while ((!check\_have\_VU || !search\_client) && num\_VU != "0") {

system("cls"); // очистка консоли

cout << "Ошибка ввода!" << endl;

if (!search\_client && check\_have\_VU) {

cout << "Клиент с данным номером водительского удостоверения не найден!" << endl;

}

cout << "Регистрация выдачи клиенту автомобиля на прокат" << endl;

cout << "Введите номер водительского удостоверения клиента";

cout << "строка формата RR AA NNNNNN, где RR – код региона(цифры)" << endl;

cout << "AA – серия(буквы из следующего множества : А, В, Е, К, М, Н, О, Р, С, Т, У, Х)" << endl;

cout << "NNNNNN – порядковый номер удостоверения(цифры)" << endl;

cout << "Код, серия и номер отделяются друг от друга пробелами " << endl;

cout << "\tНомер ВУ\tФИО" << endl;

print\_all\_clients(Root);

cout << "Для выхода в меню клиентов введите 0 " << endl;

cout << "Введите номер водительского удостоверения клиента: " << endl;

cin.clear();

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

getline(cin, num\_VU);

search\_client = NULL;

check\_have\_VU = check\_num\_VU(num\_VU);

if (check\_have\_VU) {

search\_cl(Root, num\_VU, search\_client);

}

}

if (num\_VU == "0") {

system("cls");

n\_t\_r = menu\_t\_r();

break;

}

t\_r->num\_VU = num\_VU;

system("cls"); // очистка консоли

string date = "";

cout << "Регистрация выдачи клиенту автомобиля на прокат" << endl;

cout << "Для выхода в меню выдачи и возврата введите 0" << endl;

cout << "Государственный регистрационный номер автомобиля: " << grn << endl;

cout << "Номер водительского удостоверения клиента:" << num\_VU << endl;

cout << "Введите дату выдачи в формате ДД.ММ.ГГГГ: ";

cin.clear();

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

cin >> date;

if (date == "0") {

system("cls");

n\_t\_r = menu\_t\_r();

break;

}

long long unix\_time\_begin = 0;

while ((cin.fail() || !correct\_date(date, unix\_time\_begin)) && date != "0") {

system("cls"); // очистка консоли

cin.clear();

cin.sync();

cout << "Ошибка ввода!" << endl;

cout << "Некорректная дата" << endl;

cout << "Регистрация выдачи клиенту автомобиля на прокат" << endl;

cout << "Для выхода в меню выдачи и возврата введите 0" << endl;

cout << "Государственный регистрационный номер автомобиля: " << grn << endl;

cout << "Номер водительского удостоверения клиента:" << num\_VU << endl;

cout << "Введите дату выдачи в формате ДД.ММ.ГГГГ: ";

cin >> date;

}

if (date == "0") {

system("cls");

n\_t\_r = menu\_t\_r();

break;

}

t\_r->date\_of\_issue = date;

system("cls"); // очистка консоли

cout << "Регистрация выдачи клиенту автомобиля на прокат" << endl;

cout << "Для выхода в меню выдачи и возврата введите 0" << endl;

cout << "Государственный регистрационный номер автомобиля: " << grn << endl;

cout << "Номер водительского удостоверения клиента:" << num\_VU << endl;

cout << "Дата выдачи: " << t\_r->date\_of\_issue << endl;

cout << "Автомобиль выдается на срок от одних суток до года" << endl;

cout << "Введите дату возврата в формате ДД.ММ.ГГГГ: ";

cin.clear();

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

cin >> date;

if (date == "0") {

system("cls");

n\_t\_r = menu\_t\_r();

break;

}

long long unix\_time\_end = 0;

bool check\_date = correct\_date(date, unix\_time\_end);

while ((cin.fail() || !check\_date || unix\_time\_begin + 86400 > unix\_time\_end || unix\_time\_begin + 31536000 < unix\_time\_end) && date != "0") {

cin.clear();

cin.sync();

system("cls"); // очистка консоли

cout << "Ошибка ввода!" << endl;

cout << "Некорректная дата" << endl;

cout << "Автомобиль выдается на срок от одних суток до года!" << endl;

cout << "Регистрация выдачи клиенту автомобиля на прокат" << endl;

cout << "Для выхода в меню выдачи и возврата введите 0" << endl;

cout << "Государственный регистрационный номер автомобиля: " << grn << endl;

cout << "Номер водительского удостоверения клиента:" << num\_VU << endl;

cout << "Дата выдачи: " << t\_r->date\_of\_issue << endl;

cout << "Введите дату возврата в формате ДД.ММ.ГГГГ: ";

cin >> date;

check\_date = correct\_date(date, unix\_time\_end);

}

if (date == "0") {

system("cls");

n\_t\_r = menu\_t\_r();

break;

}

t\_r->return\_date = date;

reg\_take\_car(hash\_table, grn, table\_size); // зарегистрировать в хеш таблице

t\_r->mark = car.mark;

add\_t\_r(head, t\_r, kol\_t\_r);

system("cls");

cout << "Регистрация выдачи клиенту автомобиля на прокат успешно завершена" << endl;

print\_all\_t\_r(head);

system("pause");

system("cls");

n\_t\_r = menu\_t\_r();

break;

}

//возврат

case 2: {

system("cls"); // очистка консоли

if (H == 0) {

cout << "База клиентов пуста" << endl;

system("pause");

system("cls");

n\_t\_r = menu\_t\_r();

break;

}

if (count\_cars == 0) {

cout << "База автомобилей пуста" << endl;

system("pause");

system("cls");

n\_t\_r = menu\_t\_r();

break;

}

if (!head) {

cout << "Отсутствуют выданные автомобили" << endl;

system("pause");

system("cls");

n\_t\_r = menu\_t\_r();

break;

}

cout << "Регистрация возврата автомобиля от клиента" << endl;

cout << "Для выхода в меню выдачи и возврата введите 0" << endl;

cout << "Введите государственный регистрационный номер автомобиля: ";

cin.clear();

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

getline(cin, grn);

if (grn == "0") {

system("cls");

n\_t\_r = menu\_t\_r();

break;

}

bool check\_num\_car = check\_grn(grn);

bool check\_have\_car = false;

if (check\_num\_car) {

check\_have\_car = search\_car(hash\_table, grn, car, table\_size);

}

while ((!check\_have\_car || !check\_num\_car) && grn != "0") {

system("cls"); // очистка консоли

cout << "Ошибка ввода!" << endl;

if (!check\_have\_car && check\_num\_car) {

cout << "Автомобиль с данным государственным регистрационным номером не найден!" << endl;

}

cout << "Регистрация выдачи клиенту автомобиля на прокат" << endl;

cout << "Введите государственный регистрационный номер автомобиля" << endl;

cout << "строка формата \"ANNNAA-NN\", где N –цифра" << endl;

cout << "A – буква из следующего множества : А, В, Е, К, М, Н, О, Р, С, Т, У, Х" << endl;

cout << "Для выхода в меню выдачи и возврата введите 0" << endl;

cout << "Введите государственный регистрационный номер автомобиля: ";

cin.clear();

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

getline(cin, grn);

check\_have\_car = false;

check\_num\_car = check\_grn(grn);

if (check\_num\_car) {

check\_have\_car = search\_car(hash\_table, grn, car, table\_size);

}

}

if (grn == "0") {

system("cls");

n\_t\_r = menu\_t\_r();

break;

}

system("cls");

if (car.have) {

cout << "Данный автомобиль имеется в наличии" << endl;

}

else {

if (car.rem) {

cout << "Автомобиль в ремонте" << endl;

}

else {

search\_t\_r = search\_tr(head, grn);

if (search\_t\_r) {

del\_t\_r(head, grn, kol\_t\_r);

reg\_return\_car(hash\_table, grn, table\_size); // зарегистрировать в хеш таблице

print\_all\_t\_r(head);

}

}

}

cout << "Регистрация возврата автомобиля от клиента успешно завершена" << endl;

system("pause");

system("cls");

n\_t\_r = menu\_t\_r();

break;

}

// просмотр всех выданных авто

case 3: {

system("cls"); // очистка консоли

if (count\_cars == 0) {

cout << "База автомобилей пуста" << endl;

system("pause");

system("cls");

n\_t\_r = menu\_t\_r();

break;

}

else if (head) {

cout << "Просмотр всех выданных автомобилей" << endl;

print\_all\_t\_r(head);

}

else {

cout << "Отсутствуют выданнные автомобили" << endl;

}

system("pause");

system("cls");

n\_t\_r = menu\_t\_r();

break;

}

}

}

system("cls");

n = menu(); //главное меню

break;

}

}

}

del\_all\_t\_r(head, kol\_t\_r);

delete[] hash\_table;

delete client;

delete t\_r;

FreeTree(Root);

return 0;

}

### Файл cleents.h

#pragma once

#include <iostream>

#include <string>

using namespace std;

struct clients;

struct take\_return;

// функция сравнения номеров ВУ

bool comparison(string num\_VU, string num\_VU\_root);

// поиск клиента по номеру ВУ

void search\_cl(clients\* client, string num\_VU, clients\*& search\_client);

//Функция добавления клиента без балансировки

void Add(clients\* add\_client, clients\*& client);

//Функция вывода дерева

void print\_client(clients\* client);

// функция вычисления баланс фактора

int bf(clients\* client);

// восстановление правильного значения высоты

void fixheight(clients\*& client);

// правый поворот

clients\* rotateright(clients\* client);

// левый поворот

clients\* rotateleft(clients\* client);

//балансировка

clients\* balance(clients\* client);

// отбалансировать все дерево

void balance\_all(clients\*& client);

// добавить клиента с последующей балансировкой

void Add\_balance(clients\* add\_client, clients\*& Root, short int& H);

// вывести всех клиентов

void print\_all\_clients(clients\* client);

// удалить клиента без проверки на высоту

clients\* del\_elem(clients\*& client, string num\_VU);

// удалить клиента с проверкой на изменение высоты дерева

void del\_check\_H(clients\*& Root, string num\_VU, short int& H);

// проверка на допустимые символы

bool sym\_true(unsigned char sym);

// проверка на корректное значение номера ВУ

bool check\_num\_VU(string num\_VU);

// проверка на корректное значение номера паспорта

bool check\_passport(string passport);

// удалить всех клиентов, кроме тех кому выдан автомобиль

void del\_except\_t\_r(clients\*& Root, short int& H, take\_return\* head);

// вывод информации о клиенте

void print\_client\_complete(clients\* client);

// поиск фрагмента в строке

bool contain\_fragment(string str, string fragment);

// поиск по фрагментам фио или адреса

void search\_fragment(clients\* client, string fragment, bool& header);

// полная очистка динамической памяти бинарного дерева

void FreeTree(clients\* client);

### Файл clients.cpp

#include "clients.h"

#include "take\_return.h"

struct clients

{

string num\_VU; // номер водительского удостоверения

string FIO; // Фамилия имя отчество

string passport;

string address;

clients\* Left; //УКАЗАТЕЛИ на соседние ветви

clients\* Right;

short int h;// высота поддерева

};

struct take\_return

{

string num\_VU; // номер водительского удостоверения

string grn; // гос регистрационный номер

string mark; // марка авто

string date\_of\_issue; // дата выдачи

string return\_date; // дата возврата

take\_return\* next;

};

// функция сравнения номеров ВУ

bool comparison(string num\_VU, string num\_VU\_root) {

bool right = false; // если направо 1 если налево 0

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

if (i != 2 && i != 5) {

if (num\_VU[i] > num\_VU\_root[i]) {

right = true;

break;

}

else if (num\_VU[i] < num\_VU\_root[i]) {

break;

}

}

}

return right;

}

void search\_cl(clients\* client, string num\_VU, clients\*& search\_client) {

if (client) {

if (client->num\_VU == num\_VU) {

search\_client = client;

return;

}

else if (comparison(num\_VU, client->num\_VU)) { //Если оно больше того, что в этой ветке

if (client->Right) {

search\_cl(client->Right, num\_VU, search\_client);

}

else {

search\_client = NULL;

return;

}

}

else

{

if (client->Left) {

search\_cl(client->Left, num\_VU, search\_client);

}

else {

search\_client = NULL;

return;

}

}

}

else return;

}

//Функция внесения данных

void Add(clients\* add\_client, clients\*& client)

{

//Если ветки не существует

if (!client)

{ //создадим ее и зададим в нее данные

client = new clients;

client->num\_VU = add\_client->num\_VU;

client->FIO = add\_client->FIO;

client->passport = add\_client->passport;

client->address = add\_client->address;

client->Left = add\_client->Left;

client->Right = add\_client->Right;

client->h = add\_client->h;

return;

}

else //Иначе сверим вносимое

if (comparison(add\_client->num\_VU, client->num\_VU))

{ //Если оно больше того, что в этой ветке - добавим вправо

Add(add\_client, client->Right);

if (client->Right) {

if (client->Right->h == client->h) client->h++;

}

}

else if (add\_client->num\_VU == client->num\_VU) {

return;

}

else

{ //Иначе в ветку слева

Add(add\_client, client->Left);

if (client->Left) {

if (client->Left->h == client->h) client->h++;

}

};

}

//Функция вывода дерева

int tabs = 0; //Для создания отступов

void print\_client(clients\* client)

{

if (!client) return; //Если ветки не существует - выходим. Выводить нечего

tabs += 5; //Иначе увеличим счетчик рекурсивно вызванных процедур

//Который будет считать нам отступы для красивого вывода

print\_client(client->Right);//Выведем ветку и ее подветки справа

for (int i = 0; i < tabs; i++) cout << " "; //Потом отступы

cout << client->num\_VU << " " << client->FIO << endl; //Данные этой ветки

print\_client(client->Left); //Выведем ветку и ее подветки слева

tabs -= 5; //После уменьшим кол-во отступов

return;

}

// функция вычисления баланс фактора

int bf(clients\* client)

{

short int h\_right = 0, h\_left = 0;

if (client->Right) {

h\_right = client->Right->h;

}

if (client->Left) {

h\_left = client->Left->h;

}

return h\_right - h\_left;

}

// восстановление правильного значения высоты

void fixheight(clients\*& client)

{

short int h\_right = 0, h\_left = 0;

if (client->Right) {

h\_right = client->Right->h;

}

if (client->Left) {

h\_left = client->Left->h;

}

if (h\_right > h\_left) {

client->h = h\_right + 1;

}

else {

client->h = h\_left + 1;

}

}

clients\* rotateright(clients\* client) // правый поворот

{

clients\* q = client->Left;

client->Left = q->Right;

q->Right = client;

fixheight(client);

fixheight(q);

return q;

}

clients\* rotateleft(clients\* client) // левый поворот

{

clients\* p = client->Right;

client->Right = p->Left;

p->Left = client;

fixheight(client);

fixheight(p);

return p;

}

clients\* balance(clients\* client) // балансировка дерева

{

if (client) {

fixheight(client);

if (bf(client) >= 2)

{

if (bf(client->Right) < 0)

client->Right = rotateright(client->Right);

return rotateleft(client);

}

if (bf(client) <= -2)

{

if (bf(client->Left) > 0)

client->Left = rotateleft(client->Left);

return rotateright(client);

}

}

return client; // балансировка не нужна

}

void balance\_all(clients\*& client) {

if (!client) return; //Если дерева нет, выходим

balance\_all(client->Left); //Обошли левое поддерево

balance\_all(client->Right); //Обошли правое поддерево

client = balance(client);

}

void Add\_balance(clients\* add\_client, clients\*& Root, short int& H) {

Add(add\_client, Root);

if (Root->h != H) {

balance\_all(Root);

H = Root->h;

}

}

void print\_all\_clients(clients\* client) {

if (!client) return; //Если дерева нет, выходим

print\_all\_clients(client->Left); //Обошли левое поддерево

cout << "\t" << client->num\_VU << "\t" << client->FIO << endl;

print\_all\_clients(client->Right); //Обошли правое поддерево

}

clients\* del\_elem(clients\*& client, string num\_VU) {

if (!client)

return client;

if (num\_VU == client->num\_VU) {

clients\* tmp;

if (client->Right == NULL)

tmp = client->Left;

else {

clients\* ptr = client->Right;

if (ptr->Left == NULL) {

ptr->Left = client->Left;

tmp = ptr;

}

else {

clients\* pmin = ptr->Left;

while (pmin->Left != NULL) {

ptr = pmin;

pmin = ptr->Left;

}

ptr->Left = pmin->Right;

ptr = balance(ptr);

pmin->Left = client->Left;

pmin->Right = client->Right;

tmp = pmin;

}

}

delete client;

tmp = balance(tmp);

return tmp;

}

else if (comparison(num\_VU, client->num\_VU)) {

client->Right = del\_elem(client->Right, num\_VU);

}

else {

client->Left = del\_elem(client->Left, num\_VU);

}

client = balance(client);

return client;

}

void del\_check\_H(clients\*& Root, string num\_VU, short int& H) {

Root = del\_elem(Root, num\_VU);

if (Root) {

if (Root->h != H) {

balance\_all(Root);

H = Root->h;

}

}

else {

H = 0;

}

}

bool sym\_true(unsigned char sym ) {

if (sym == 192 || sym == 194 || sym == 197 || sym == 202 ||

sym == 204 || sym == 205 || sym == 206 || sym == 208 ||

sym == 209 || sym == 210 || sym == 211 || sym == 213) return true;

else return false;

}

bool check\_num\_VU(string num\_VU) {

if (num\_VU.length() == 12 && num\_VU[0] >= 48 && num\_VU[0] <= 57 &&

num\_VU[1] >= 48 && num\_VU[1] <= 57 &&

num\_VU[2] == ' ' &&

sym\_true(num\_VU[3]) &&

sym\_true(num\_VU[4]) &&

num\_VU[5] == ' ' &&

num\_VU[6] >= 48 && num\_VU[6] <= 57 &&

num\_VU[7] >= 48 && num\_VU[7] <= 57 &&

num\_VU[8] >= 48 && num\_VU[8] <= 57 &&

num\_VU[9] >= 48 && num\_VU[9] <= 57 &&

num\_VU[10] >= 48 && num\_VU[10] <= 57 &&

num\_VU[11] >= 48 && num\_VU[11] <= 57) return true;

else return false;

}

bool check\_passport(string passport) {

bool check = true;

if (passport.length() == 10) {

for (int i = 0; i < 10; i++) {

if (passport[i] < 48 || passport[i] > 57) {

check = false;

break;

}

}

}

else {

check = false;

}

return check;

}

void del\_except\_t\_r(clients\*& Root, short int& H, take\_return\* head)

{

take\_return\* node = head;

if (!head) {

FreeTree(Root);

Root = NULL;

H = 0;

}

else {

clients\* new\_root = NULL;

clients\* search\_client;

short int H2 = 0;

while (node) {

search\_cl(Root, node->num\_VU, search\_client);

Add\_balance(search\_client, new\_root, H2);

node = node->next;

}

FreeTree(Root);

Root = new\_root;

H = H2;

}

}

void print\_client\_complete(clients\* client){

cout << "\tНомер водительского удостоверения" << endl;

cout << "\t" << client->num\_VU << endl;

cout << "\tФамилия, Имя, Отчество" << endl;

cout << "\t" << client->FIO << endl;

cout << "\tПаспортные данные" << endl;

cout << "\t" << client->passport << endl;

cout << "\tАдрес" << endl;

cout << "\t" << client->address << endl;

}

bool contain\_fragment(string str, string fragment){

int len\_str = str.length(), len\_fragment = fragment.length();

int len\_fr = 0;

if (len\_str >= len\_fragment) {

for (int i = 0; i <= len\_str - len\_fragment; i++) {

len\_fr = 0;

for (int f = 0; f < len\_fragment; f++) {

if (str[i + f] == fragment[f]) {

len\_fr += 1;

if (len\_fr == len\_fragment) {

return true;

}

}

else break;

}

}

}

return false;

}

void search\_fragment(clients\* client, string fragment, bool& header) {

if (!client) return; //Если дерева нет, выходим

search\_fragment(client->Left, fragment, header); //Обошли левое поддерево

search\_fragment(client->Right, fragment, header); //Обошли правое поддерево

if (contain\_fragment(client->FIO, fragment) || contain\_fragment(client->address, fragment)) {

if (!header) {

cout << "\tНомер ВУ\tФИО, адрес" << endl;

header = true;

}

cout << "\t" << client->num\_VU << "\t" << client->FIO << " " << client->address << endl;

}

}

void FreeTree(clients\* client)

{

if (!client) return;

FreeTree(client->Left);

FreeTree(client->Right);

delete client;

return;

}

### Файл cars.h

#pragma once

#include <iostream>

#include <Windows.h>

#include <string>

using namespace std;

struct cars;

struct take\_return;

// поиск автомобиля по ГРН

bool search\_car(cars\* hash\_table, string grn, cars& search\_car, int table\_size);

// получение позиции автомобиля в хеш таблице

int get\_pos(cars\* hash\_table, string grn, int table\_size);

// поиск автомобиля по марке

void search\_mark(cars\* hash\_table, string mark, int table\_size, take\_return\* head);

// вывод информации об автомобиле

void print\_car(cars car);

// вывод информации о всех автомобилях

void print\_cars(cars\* hash\_table, int table\_size, int count\_cars);

int hash\_function(string key, int n, int table\_size);

// рехеширование

void rehash(cars\* (&hash\_table), int& table\_size);

// добавить автомобиль с возможностью рехеширования и расширения таблицы

void add\_car\_check(cars car, cars\* (&hash\_table), int& table\_size, int& count\_cars);

// добавить автомобиль в хеш таблицу

bool add\_car(cars car, cars\* (&hash\_table), int table\_size);

// проверка на допустимые символы

bool sym\_car(unsigned char sym);

// проверка на корректность ГРН

bool check\_grn(string grn);

// проверка года

bool check\_year(string year, int wYear);

//удаление автомобиля

void del\_car(cars\* (&hash\_table), string grn, int table\_size, int& count\_cars);

//уделение всех автомобилей, которые есть в наличии

void del\_car\_have(cars\* (&hash\_table), int& table\_size, int& count\_cars);

// регистрация отправки на ремонт

void reg\_send\_rem(cars\* hash\_table, string grn, int table\_size);

// реистрация прибытия из ремонта

void reg\_get\_rem(cars\* hash\_table, string grn, int table\_size);

// регистрация выдачи автомобиля клиенту

void reg\_take\_car(cars\* hash\_table, string grn, int table\_size);

// регистрация возврата автомобиля от клиента

void reg\_return\_car(cars\* hash\_table, string grn, int table\_size);

### Файл cars.cpp

#include "cars.h"

struct cars

{

string grn; // гос регистрационный номер

string mark; // марка авто

string color;

int year;

bool have;// имеется ли в наличии

bool rem;

};

struct take\_return

{

string num\_VU; // номер водительского удостоверения

string grn; // гос регистрационный номер

string mark; // марка авто

string date\_of\_issue; // дата выдачи

string return\_date; // дата возврата

take\_return\* next;

};

bool search\_car(cars\* hash\_table, string grn, cars& search\_car, int table\_size) {

int attempt = 0; // номер попытки

int val = 0;

while (attempt < 5) {

val = hash\_function(grn, attempt, table\_size);

if (hash\_table[val].grn == "") {

return false;

}

else if (hash\_table[val].grn == grn) {

search\_car = hash\_table[val];

return true;

}

else {

attempt += 1;

}

}

return false;

}

int get\_pos(cars\* hash\_table, string grn, int table\_size) {

int attempt = 0; // номер попытки

int val = 0;

while (attempt < 5) {

val = hash\_function(grn, attempt, table\_size);

if (hash\_table[val].grn == "") {

return 0;

}

else if (hash\_table[val].grn == grn) {

return val;

}

else {

attempt += 1;

}

}

return 0;

}

void search\_mark(cars\* hash\_table, string mark, int table\_size, take\_return\* head) {

bool header = false;

take\_return\* node = head;

cars srch\_car;

while (node) {

if (node->mark == mark) {

if (!header) {

cout << "Позиция | Гос.Рег.Номер | Марка | Цвет | Год выпуска | В наличии | В ремонте" << endl;

header = true;

}

search\_car(hash\_table, node->grn, srch\_car, table\_size);

printf("%-7d", get\_pos(hash\_table, node->grn, table\_size));

cout << " | ";

printf("%-13s", srch\_car.grn.c\_str());

cout << " | ";

printf("%-20s", srch\_car.mark.c\_str());

cout << " | ";

printf("%-20s", srch\_car.color.c\_str());

cout << " | ";

printf("%-11d", srch\_car.year);

cout << " | ";

if (srch\_car.have) {

printf("%-9s", "Да");

}

else printf("%-9s", "Нет");

cout << " | ";

if (srch\_car.rem) {

cout << "Да" << endl;

}

else cout << "Нет" << endl;

}

node = node->next;

}

for (int i = 0; i < table\_size; i++) {

if (hash\_table[i].mark == mark) {

if (hash\_table[i].have || hash\_table[i].rem) {

if (!header) {

cout << "Позиция | Гос.Рег.Номер | Марка | Цвет | Год выпуска | В наличии | В ремонте" << endl;

header = true;

}

printf("%-7d", i);

cout << " | ";

printf("%-13s", hash\_table[i].grn.c\_str());

cout << " | ";

printf("%-20s", hash\_table[i].mark.c\_str());

cout << " | ";

printf("%-20s", hash\_table[i].color.c\_str());

cout << " | ";

printf("%-11d", hash\_table[i].year);

cout << " | ";

if (hash\_table[i].have) {

printf("%-9s", "Да");

}

else printf("%-9s", "Нет");

cout << " | ";

if (hash\_table[i].rem) {

cout << "Да" << endl;

}

else cout << "Нет" << endl;

}

}

}

if (!header) cout << "Совпадений не найдено" << endl;

}

void print\_car(cars car) {

cout << "Гос.Рег.Номер | Марка | Цвет | Год выпуска | В наличии | В ремонте" << endl;

printf("%-13s", car.grn.c\_str());

cout << " | ";

printf("%-20s", car.mark.c\_str());

cout << " | ";

printf("%-20s", car.color.c\_str());

cout << " | ";

printf("%-11d", car.year);

cout << " | ";

if (car.have) {

printf("%-9s", "Да");

}

else printf("%-9s", "Нет");

cout << " | ";

if (car.rem) {

cout << "Да" << endl;

}

else cout << "Нет" << endl;

}

void print\_cars(cars\* hash\_table, int table\_size, int count\_cars) {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

setlocale(LC\_ALL, "rus");

int kol\_cars = 0;

if (count\_cars > 0) {

cout << "Размер таблицы: " << table\_size << "\tКоличество автомобилей: " << count\_cars << endl;

cout << "Позиция | Гос.Рег.Номер | Марка | Цвет | Год выпуска | В наличии | В ремонте" << endl;

for (int i = 0; i < table\_size; i++) {

if ((hash\_table[i].grn != "" && hash\_table[i].grn != "0")) {

printf("%-7d", i);

cout << " | ";

printf("%-13s", hash\_table[i].grn.c\_str());

cout << " | ";

printf("%-20s", hash\_table[i].mark.c\_str());

cout << " | ";

printf("%-20s", hash\_table[i].color.c\_str());

cout << " | ";

printf("%-11d", hash\_table[i].year);

cout << " | ";

if (hash\_table[i].have) {

printf("%-9s", "Да");

}

else printf("%-9s", "Нет");

cout << " | ";

if (hash\_table[i].rem) {

cout << "Да" << endl;

}

else cout << "Нет" << endl;

kol\_cars += 1;

}

if (kol\_cars == count\_cars) break;

}

}

}

int hash\_function(string key, int n, int table\_size) {

return ((int(unsigned char(key[0])) \* int(unsigned char(key[0]))

+ int(unsigned char(key[1])) \* int(unsigned char(key[1])) \* 2

+ int(unsigned char(key[2])) \* int(unsigned char(key[2])) \* 3

+ int(unsigned char(key[3])) \* int(unsigned char(key[3])) \* 4

+ int(unsigned char(key[4])) \* int(unsigned char(key[4])) \* 5

+ int(unsigned char(key[5])) \* int(unsigned char(key[5])) \* 6

+ int(unsigned char(key[7])) \* int(unsigned char(key[7])) \* 7

+ int(unsigned char(key[8])) \* int(unsigned char(key[8])) \* 8

- 497664) + table\_size/3 \* n) % table\_size;

}

void rehash(cars\* (&hash\_table), int& table\_size) {

cars\* old\_buf = new cars[table\_size];

for (int i = 0; i < table\_size; i++) {

old\_buf[i] = hash\_table[i];

}

int old\_size = table\_size;

bool success = true;

do {

success = true;

table\_size \*= 2;

hash\_table = new cars [table\_size];

for (int i = 0; i < table\_size; i++) {

hash\_table[i].grn = "";

}

for (int i = 0; i < old\_size; i++) {

if (old\_buf[i].grn != "" && old\_buf[i].grn != "0") {

success = add\_car(old\_buf[i], hash\_table, table\_size);

}

if (!success) break;

}

} while (!success);

delete[] old\_buf;

}

void add\_car\_check(cars car, cars\* (&hash\_table), int& table\_size, int& count\_cars) {

bool success = false;

while (!success) {

success = add\_car(car, hash\_table, table\_size);

if (!success) {

rehash(hash\_table, table\_size);

}

}

count\_cars += 1;

}

bool add\_car(cars car, cars\* (&hash\_table), int table\_size) {

int attempt = 0; // номер попытки

int val = 0;

while (attempt < 5) {

val = hash\_function(car.grn, attempt, table\_size);

if (hash\_table[val].grn == "" || hash\_table[val].grn == "0") {

hash\_table[val].grn = car.grn;

hash\_table[val].mark = car.mark;

hash\_table[val].color = car.color;

hash\_table[val].year = car.year;

hash\_table[val].have = car.have;

hash\_table[val].rem = car.rem;

return true;

}

else {

attempt += 1;

}

}

return false;

}

bool sym\_car(unsigned char sym) {

if (sym == 192 || sym == 194 || sym == 197 || sym == 202 ||

sym == 204 || sym == 205 || sym == 206 || sym == 208 ||

sym == 209 || sym == 210 || sym == 211 || sym == 213) return true;

else return false;

}

bool check\_grn(string grn) {

if (grn.length() == 9 &&

sym\_car(grn[0]) &&

grn[1] >= 48 && grn[1] <= 57 &&

grn[2] >= 48 && grn[2] <= 57 &&

grn[3] >= 48 && grn[3] <= 57 &&

sym\_car(grn[4]) &&

sym\_car(grn[5]) &&

grn[6] == '-' &&

grn[7] >= 48 && grn[7] <= 57 &&

grn[8] >= 48 && grn[8] <= 57) return true;

else return false;

}

bool check\_year(string year, int wYear) {

if (year.length() == 4) {

if (year[0] >= 48 && year[0] <= 57 &&

year[1] >= 48 && year[1] <= 57 &&

year[2] >= 48 && year[2] <= 57 &&

year[3] >= 48 && year[3] <= 57) {

int Y = stoi(year);

if (Y >= 1800 && Y <= wYear) return true;

}

}

return false;

}

void del\_car(cars\* (&hash\_table), string grn, int table\_size, int& count\_cars) {

int val = 0;

int attempt = 0; // номер попытки

val = hash\_function(grn, attempt, table\_size);

if (hash\_table[val].grn == "") {

return;

}

else if (hash\_table[val].grn == grn) {

hash\_table[val].grn = "0";

count\_cars -= 1;

}

else {

while (hash\_table[val].grn != grn && attempt < 5) {

attempt++;

val = hash\_function(grn, attempt, table\_size);

}

if (hash\_table[val].grn == grn) {

hash\_table[val].grn = "0";

count\_cars -= 1;

}

}

}

void del\_car\_have(cars\* (&hash\_table), int& table\_size, int& count\_cars) {

cars\* hash\_table\_old = new cars[table\_size];

for (int i = 0; i < table\_size; i++) {

hash\_table\_old[i] = hash\_table[i];

}

int table\_size\_old = table\_size;

table\_size = 10;

count\_cars = 0;

hash\_table = new cars[table\_size];

for (int i = 0; i < table\_size\_old; i++) {

if (!hash\_table\_old[i].have) {

add\_car\_check(hash\_table\_old[i], hash\_table, table\_size, count\_cars);

}

}

delete[] hash\_table\_old;

}

void reg\_send\_rem(cars\* hash\_table, string grn, int table\_size) {

int attempt = 0; // номер попытки

int val = 0;

while (attempt < 5) {

val = hash\_function(grn, attempt, table\_size);

if (hash\_table[val].grn == "") {

return;

}

else if (hash\_table[val].grn == grn) {

if (!hash\_table[val].have) {

if (hash\_table[val].rem) {

system("cls"); // очистка консоли

cout << "Автомобиль с данным регистрационном номером уже в ремонте" << endl;

break;

}

else {

system("cls"); // очистка консоли

cout << "Ошибка регистрации" << endl;

cout << "Автомобиль с данным регистрационным номером выдан клиенту" << endl;

break;

}

}

else {

hash\_table[val].rem = true;

hash\_table[val].have = false;

system("cls"); // очистка консоли

cout << "Регистрация отправки автомобиля в ремонт выполнена успешно" << endl;

break;

}

}

else {

attempt += 1;

}

}

}

void reg\_get\_rem(cars\* hash\_table, string grn, int table\_size) {

int attempt = 0; // номер попытки

int val = 0;

while (attempt < 5) {

val = hash\_function(grn, attempt, table\_size);

if (hash\_table[val].grn == "") {

return;

}

else if (hash\_table[val].grn == grn) {

if (hash\_table[val].have) {

system("cls"); // очистка консоли

cout << "Данный автомобиль не в ремонте, он имеется в наличии" << endl;

break;

}

else {

if (hash\_table[val].rem) {

hash\_table[val].rem = false;

hash\_table[val].have = true;

system("cls"); // очистка консоли

cout << "Регистрация прибытия автомобиля из ремонта выполнена успешно" << endl;

break;

}

else {

system("cls"); // очистка консоли

cout << "Ошибка регистрации прибытия автомобиля из ремонта" << endl;

cout << "Данный автомобиль выдан клиенту" << endl;

break;

}

}

}

else {

attempt += 1;

}

}

}

void reg\_take\_car(cars\* hash\_table, string grn, int table\_size) {

int attempt = 0; // номер попытки

int val = 0;

while (attempt < 5) {

val = hash\_function(grn, attempt, table\_size);

if (hash\_table[val].grn == "") {

return;

}

else if (hash\_table[val].grn == grn) {

if (hash\_table[val].have) {

hash\_table[val].have = false;

break;

}

}

else {

attempt += 1;

}

}

}

void reg\_return\_car(cars\* hash\_table, string grn, int table\_size) {

int attempt = 0; // номер попытки

int val = 0;

while (attempt < 5) {

val = hash\_function(grn, attempt, table\_size);

if (hash\_table[val].grn == "") {

return;

}

else if (hash\_table[val].grn == grn) {

if (!hash\_table[val].have) {

hash\_table[val].have = true;

break;

}

}

else {

attempt += 1;

}

}

}

### Файл take\_return.h

#pragma once.

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <iostream>

#include <Windows.h>

#include <string>

using namespace std;

struct take\_return;

// добавить информацию о выдаче

void add\_t\_r(take\_return\*& head, take\_return\* t\_r, int& kol\_t\_r);

// удаление всей информации о выдаче

void del\_all\_t\_r(take\_return\*& head, int& kol\_t\_r);

// удаление записи о выдаче автомобиля клиенту

bool del\_t\_r(take\_return\*& head, string grn, int& kol\_t\_r);

// вывод всей информации о выдаче

void print\_all\_t\_r(take\_return\* head);

// поиск по ГРН либо ВУ

take\_return\* search\_tr(take\_return\* head, string grn\_VU);

// просмотр выданных автомобилей у клиента

void print\_cars\_of\_client(take\_return\* head, string VU);

// сортировка списка

void sort\_t\_r(take\_return\*& head, int kol\_t\_r);

// проверка на корректное значение даты

bool correct\_date(string date, long long& unix\_time);

### Файл take\_return.cpp

#include "take\_return.h"

struct take\_return

{

string num\_VU; // номер водительского удостоверения

string grn; // гос регистрационный номер

string mark; // марка авто

string date\_of\_issue; // дата выдачи

string return\_date; // дата возврата

take\_return\* next;

};

void add\_t\_r(take\_return\*& head, take\_return\* t\_r, int& kol\_t\_r)

{

take\_return\* node = new take\_return;

node->num\_VU = t\_r->num\_VU;

node->grn = t\_r->grn;

node->mark = t\_r->mark;

node->date\_of\_issue = t\_r->date\_of\_issue;

node->return\_date = t\_r->return\_date;

node->next = head;

head = node;

kol\_t\_r++;

sort\_t\_r(head, kol\_t\_r);

}

void del\_all\_t\_r(take\_return\*& head, int& kol\_t\_r)

{

take\_return\* node;

while (head != NULL) {

node = head;

head = head->next;

delete node;

}

kol\_t\_r = 0;

}

bool del\_t\_r(take\_return\*& head, string grn, int& kol\_t\_r) {

take\_return\* node = head;

if (head) {

if (head->grn == grn) {

head = head->next;

delete node;

kol\_t\_r -= 1;

return true;

}

else {

while (node->next != NULL) {

if (node->next->grn == grn) {

take\_return\* del = node->next;

node->next = node->next->next;

delete del;

kol\_t\_r -= 1;

return true;

}

node = node->next;

}

}

}

return false;

}

void print\_all\_t\_r(take\_return\* head) {

cout << endl;

take\_return\* node = head;

if (head) {

cout << "Гос.Рег.Номер | Номер ВУ | Марка | Дата выдачи | Дата возврата" << endl;

}

while (node){

printf("%-13s", node->grn.c\_str());

cout << " | ";

printf("%-12s", node->num\_VU.c\_str());

cout << " | ";

printf("%-20s", node->mark.c\_str());

cout << " | ";

printf("%-11s", node->date\_of\_issue.c\_str());

cout << " | ";

cout << node->return\_date.c\_str();

cout << endl;

node = node->next;

}

}

take\_return\* search\_tr(take\_return\* head, string grn\_VU) {

take\_return\* node = head;

while (node) {

if (node->grn == grn\_VU || node->num\_VU == grn\_VU) {

return node;

}

node = node->next;

}

return NULL;

}

void print\_cars\_of\_client(take\_return\* head, string VU) {

take\_return\* node = head;

while (node) {

if (node->num\_VU == VU) {

cout << "Автомобиль с номером: " << node->grn << endl;

}

node = node->next;

}

}

void sort\_t\_r(take\_return\*& head, int kol\_t\_r) {

take\_return\* node = head;

take\_return\*\*\*\* L = new take\_return \* \*\*[2];

L[0] = new take\_return \* \*[10];

L[1] = new take\_return \* \*[10];

for (int i = 0; i < 10; i++) {

L[0][i] = new take\_return \* [kol\_t\_r];

L[1][i] = new take\_return \* [kol\_t\_r];

for (int j = 0; j < kol\_t\_r; j++) {

L[0][i][j] = NULL;

L[1][i][j] = NULL;

}

}

int x = 0, y = 0, x\_2 = 0, y\_2 = 0; // координаты числа в матрице

while(node) {

y = node->grn[8] - 48;

x = 0;

while (L[0][y][x] != NULL && x < kol\_t\_r) {

x++;

}

L[0][y][x] = node;

node = node->next;

}

/\*

cout << "8" << endl;

//////// вывод матрицы L\_1

cout << endl;

for (int y = 0; y < 10; y++) {

for (int x = 0; x < kol\_t\_r; x++) {

if (L[0][y][x]) {

cout << "\t" << L[0][y][x]->grn;

}

else cout << "\t0\t";

}

cout << endl;

}

cout << endl;

\*/

for (int y = 0; y < 10; y++) {

x = x\_2 = y\_2 = 0;

while (L[0][y][x] != NULL && x < kol\_t\_r) {

y\_2 = L[0][y][x]->grn[7] - 48;

x\_2 = 0;

while (L[1][y\_2][x\_2] != NULL && x\_2 < kol\_t\_r) {

x\_2++;

}

L[1][y\_2][x\_2] = L[0][y][x];

x++;

}

}

/\*

cout << "7" << endl;

////////// вывод матрицы L\_2

cout << endl;

for (int y = 0; y < 10; y++) {

for (int x = 0; x < kol\_t\_r; x++) {

if (L[1][y][x]) {

cout << "\t" << L[1][y][x]->grn;

}

else cout << "\t0\t";

}

cout << endl;

}

cout << endl;

\*/

for (int i = 0; i < 10; i++) { // обнуление матрицы 1

for (int j = 0; j < kol\_t\_r; j++) {

L[0][i][j] = NULL;

}

}

for (int y = 0; y < 10; y++) {

x = x\_2 = y\_2 = 0;

while (L[1][y][x] != NULL && x < kol\_t\_r) {

y\_2 = (unsigned char(L[1][y][x]->grn[5]) - 192) % 10;

x\_2 = 0;

while (L[0][y\_2][x\_2] != NULL && x\_2 < kol\_t\_r) {

x\_2++;

}

L[0][y\_2][x\_2] = L[1][y][x];

x++;

}

}

/\*

cout << "5" << endl;

//////// вывод матрицы L\_1

cout << endl;

for (int y = 0; y < 10; y++) {

for (int x = 0; x < kol\_t\_r; x++) {

if (L[0][y][x]) {

cout << "\t" << L[0][y][x]->grn;

}

else cout << "\t0\t";

}

cout << endl;

}

cout << endl;

\*/

for (int i = 0; i < 10; i++) { // обнуление матрицы 2

for (int j = 0; j < kol\_t\_r; j++) {

L[1][i][j] = NULL;

}

}

for (int y = 0; y < 10; y++) {

x = x\_2 = y\_2 = 0;

while (L[0][y][x] != NULL && x < kol\_t\_r) {

y\_2 = (unsigned char(L[0][y][x]->grn[5]) - 192) / 10;

x\_2 = 0;

while (L[1][y\_2][x\_2] != NULL && x\_2 < kol\_t\_r) {

x\_2++;

}

L[1][y\_2][x\_2] = L[0][y][x];

x++;

}

}

/\*

cout << "5" << endl;

////////// вывод матрицы L\_2

cout << endl;

for (int y = 0; y < 10; y++) {

for (int x = 0; x < kol\_t\_r; x++) {

if (L[1][y][x]) {

cout << "\t" << L[1][y][x]->grn;

}

else cout << "\t0\t";

}

cout << endl;

}

cout << endl;

\*/

for (int i = 0; i < 10; i++) { // обнуление матрицы 1

for (int j = 0; j < kol\_t\_r; j++) {

L[0][i][j] = NULL;

}

}

for (int y = 0; y < 10; y++) {

x = x\_2 = y\_2 = 0;

while (L[1][y][x] != NULL && x < kol\_t\_r) {

y\_2 = (unsigned char(L[1][y][x]->grn[4]) - 192) % 10;

x\_2 = 0;

while (L[0][y\_2][x\_2] != NULL && x\_2 < kol\_t\_r) {

x\_2++;

}

L[0][y\_2][x\_2] = L[1][y][x];

x++;

}

}

/\*

cout << "4" << endl;

//////// вывод матрицы L\_1

cout << endl;

for (int y = 0; y < 10; y++) {

for (int x = 0; x < kol\_t\_r; x++) {

if (L[0][y][x]) {

cout << "\t" << L[0][y][x]->grn;

}

else cout << "\t0\t";

}

cout << endl;

}

cout << endl;

\*/

for (int i = 0; i < 10; i++) { // обнуление матрицы 2

for (int j = 0; j < kol\_t\_r; j++) {

L[1][i][j] = NULL;

}

}

for (int y = 0; y < 10; y++) {

x = x\_2 = y\_2 = 0;

while (L[0][y][x] != NULL && x < kol\_t\_r) {

y\_2 = (unsigned char(L[0][y][x]->grn[4]) - 192) / 10;

x\_2 = 0;

while (L[1][y\_2][x\_2] != NULL && x\_2 < kol\_t\_r) {

x\_2++;

}

L[1][y\_2][x\_2] = L[0][y][x];

x++;

}

}

/\*

cout << "4" << endl;

////////// вывод матрицы L\_2

cout << endl;

for (int y = 0; y < 10; y++) {

for (int x = 0; x < kol\_t\_r; x++) {

if (L[1][y][x]) {

cout << "\t" << L[1][y][x]->grn;

}

else cout << "\t0\t";

}

cout << endl;

}

cout << endl;

\*/

for (int i = 0; i < 10; i++) { // обнуление матрицы 1

for (int j = 0; j < kol\_t\_r; j++) {

L[0][i][j] = NULL;

}

}

for (int y = 0; y < 10; y++) {

x = x\_2 = y\_2 = 0;

while (L[1][y][x] != NULL && x < kol\_t\_r) {

y\_2 = L[1][y][x]->grn[3] - 48;

x\_2 = 0;

while (L[0][y\_2][x\_2] != NULL && x\_2 < kol\_t\_r) {

x\_2++;

}

L[0][y\_2][x\_2] = L[1][y][x];

x++;

}

}

/\*

cout << "3" << endl;

//////// вывод матрицы L\_1

cout << endl;

for (int y = 0; y < 10; y++) {

for (int x = 0; x < kol\_t\_r; x++) {

if (L[0][y][x]) {

cout << "\t" << L[0][y][x]->grn;

}

else cout << "\t0\t";

}

cout << endl;

}

cout << endl;

\*/

for (int i = 0; i < 10; i++) { // обнуление матрицы 2

for (int j = 0; j < kol\_t\_r; j++) {

L[1][i][j] = NULL;

}

}

for (int y = 0; y < 10; y++) {

x = x\_2 = y\_2 = 0;

while (L[0][y][x] != NULL && x < kol\_t\_r) {

y\_2 = L[0][y][x]->grn[2] - 48;

x\_2 = 0;

while (L[1][y\_2][x\_2] != NULL && x\_2 < kol\_t\_r) {

x\_2++;

}

L[1][y\_2][x\_2] = L[0][y][x];

x++;

}

}

/\*

cout << "2" << endl;

////////// вывод матрицы L\_2

cout << endl;

for (int y = 0; y < 10; y++) {

for (int x = 0; x < kol\_t\_r; x++) {

if (L[1][y][x]) {

cout << "\t" << L[1][y][x]->grn;

}

else cout << "\t0\t";

}

cout << endl;

}

cout << endl;

\*/

for (int i = 0; i < 10; i++) { // обнуление матрицы 1

for (int j = 0; j < kol\_t\_r; j++) {

L[0][i][j] = NULL;

}

}

for (int y = 0; y < 10; y++) {

x = x\_2 = y\_2 = 0;

while (L[1][y][x] != NULL && x < kol\_t\_r) {

y\_2 = L[1][y][x]->grn[1] - 48;

x\_2 = 0;

while (L[0][y\_2][x\_2] != NULL && x\_2 < kol\_t\_r) {

x\_2++;

}

L[0][y\_2][x\_2] = L[1][y][x];

x++;

}

}

/\*

cout << "1" << endl;

//////// вывод матрицы L\_1

cout << endl;

for (int y = 0; y < 10; y++) {

for (int x = 0; x < kol\_t\_r; x++) {

if (L[0][y][x]) {

cout << "\t" << L[0][y][x]->grn;

}

else cout << "\t0\t";

}

cout << endl;

}

cout << endl;

\*/

for (int i = 0; i < 10; i++) { // обнуление матрицы 2

for (int j = 0; j < kol\_t\_r; j++) {

L[1][i][j] = NULL;

}

}

for (int y = 0; y < 10; y++) {

x = x\_2 = y\_2 = 0;

while (L[0][y][x] != NULL && x < kol\_t\_r) {

y\_2 = (unsigned char(L[0][y][x]->grn[0]) - 192) % 10;

x\_2 = 0;

while (L[1][y\_2][x\_2] != NULL && x\_2 < kol\_t\_r) {

x\_2++;

}

L[1][y\_2][x\_2] = L[0][y][x];

x++;

}

}

/\*

cout << "0" << endl;

////////// вывод матрицы L\_2

cout << endl;

for (int y = 0; y < 10; y++) {

for (int x = 0; x < kol\_t\_r; x++) {

if (L[1][y][x]) {

cout << "\t" << L[1][y][x]->grn;

}

else cout << "\t0\t";

}

cout << endl;

}

cout << endl;

\*/

for (int i = 0; i < 10; i++) { // обнуление матрицы 1

for (int j = 0; j < kol\_t\_r; j++) {

L[0][i][j] = NULL;

}

}

for (int y = 0; y < 10; y++) {

x = x\_2 = y\_2 = 0;

while (L[1][y][x] != NULL && x < kol\_t\_r) {

y\_2 = (unsigned char(L[1][y][x]->grn[0]) - 192) / 10;

x\_2 = 0;

while (L[0][y\_2][x\_2] != NULL && x\_2 < kol\_t\_r) {

x\_2++;

}

L[0][y\_2][x\_2] = L[1][y][x];

x++;

}

}

/\*

cout << "0" << endl;

//////// вывод матрицы L\_1

cout << endl;

for (int y = 0; y < 10; y++) {

for (int x = 0; x < kol\_t\_r; x++) {

if (L[0][y][x]) {

cout << "\t" << L[0][y][x]->grn;

}

else cout << "\t0\t";

}

cout << endl;

}

cout << endl;

\*/

head = NULL;

for (int y = 9; y >= 0; y--) {

for (int x = kol\_t\_r - 1; x >= 0; x--) {

if (L[0][y][x]) {

L[0][y][x]->next = head;

head = L[0][y][x];

}

}

}

for (int i = 0; i < 10; i++) {

delete[] L[0][i];

delete[] L[1][i];

}

delete[] L[0];

delete[] L[1];

delete[] L;

}

bool correct\_date(string date, long long& unix\_time) {

if (date.length() == 10) {

if (date[0] >= 48 && date[0] <= 51 &&

date[1] >= 48 && date[1] <= 57 &&

date[2] == 46 &&

date[3] >= 48 && date[3] <= 49 &&

date[4] >= 48 && date[4] <= 57 &&

date[5] == 46 &&

date[6] >= 50 && date[6] <= 57 &&

date[7] >= 48 && date[7] <= 57 &&

date[8] >= 48 && date[8] <= 57 &&

date[9] >= 48 && date[9] <= 57)

{

time\_t tm = 0;

struct tm\* dt = localtime(&tm);

int mday, mon, year;

mday = (date[0] - 48) \* 10 + date[1] - 48;

dt->tm\_mday = mday;

mon = (date[3] - 48) \* 10 + date[4] - 49;

dt->tm\_mon = mon;

year = (date[6] - 48) \* 1000 + (date[7] - 48) \* 100 + (date[8] - 48) \* 10 + (date[9] - 48) - 1900;

dt->tm\_year = year;

dt->tm\_hour = 0;

dt->tm\_min = 0;

dt->tm\_sec = 0;

tm = mktime(dt);

unix\_time = tm;

if (year == dt->tm\_year &&

mon == dt->tm\_mon &&

mday == dt->tm\_mday) {

return true;

}

}

}

return false;

}