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

КУРСОВАЯ РАБОТА (ПРОЕКТ)   
ЗАЩИЩЕНА С ОЦЕНКОЙ

РУКОВОДИТЕЛЬ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ст. преподаватель |  |  |  | М.Д. Поляк |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

|  |
| --- |
| ПОЯСНИТЕЛЬНАЯ ЗАПИСКА К КУРСОВОМУ ПРОЕКТУ |
| РАЗРАБОТКА ПРОГРАММЫ «ОБСЛУЖИВАНИЕ КЛИЕНТОВ В БЮРО ПРОКАТА АВТОМОБИЛЕЙ» |
| по дисциплине: СТРУКТУРЫ И АЛГОРИТМЫ ОБРАБОТКИ ДАННЫХ |
|  |
|  |

РАБОТУ ВЫПОЛНИЛ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СТУДЕНТ ГР. № | 4032 |  |  |  | А.А. Поздеев |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург 2022

Оглавление

[Задание на курсовой проект 3](#_Toc104886279)

[Введение 4](#_Toc104886280)

[Алгоритмы и структуры данных 4](#_Toc104886281)

[Описание программы 5](#_Toc104886282)

[Тестирование 13](#_Toc104886283)

[Заключение 25](#_Toc104886284)

[Список использованной литературы 26](#_Toc104886285)

ПРИЛОЖЕНИЕ....………………………………………………….……………………...29

**Задание на курсовой проект**

Задача курсового проекта: разработка информационной системы для заданной предметной области с использованием заданных структур данных и алгоритмов. Тема курсового проекта: «Использование заданных структур данных и алгоритмов при разработке программного обеспечения информационной системы»

* предметная область: обслуживание клиентов в бюро проката автомобилей (907 mod 6 = 1)
* метод хеширования: закрытое хеширование с двойным хешированием (907 mod 4 = 3)
* метод сортировки: быстрая (Хоара) (907 mod 7 = 4)
* вид списка: циклический однонаправленный (907 mod 5 = 2)
* метод обхода дерева: обратный (907 mod 3 = 1)
* алгоритм поиска слова в тексте: прямой (907 mod 2 = 1)

**Введение**

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

* Регистрацию нового клиента;
* Снятие с обслуживания клиента;
* Просмотр всех зарегистрированных клиентов;
* Удаление всех клиентов из системы;
* Поиск клиента по «номеру водительского удостоверения».

Результат поиска: все сведения о найденном клиенте и государственный регистрационный номер автомобиля, который ему выдан (при наличии);

* Поиск клиента по фрагменту ФИО.

Результат поиска – список найденных клиентов с указанием номера водительского удостоверения, ФИО и адреса;

* Регистрация нового автомобиля;
* Удаление автомобиля из системы;
* Просмотр всех зарегистрированных автомобилей;
* Удаление всех автомобилей из системы;
* Поиск автомобиля по «государственному регистрационному номеру».

Результат поиска – все сведения о найденном автомобиле, а также ФИО и номер водительского удостоверения клиента, которому выдан этот автомобиль (при наличии);

* Поиск автомобиля по названию марки автомобиля.

Результаты поиска – список найденных автомобилей с указанием «государственного регистрационного номера», марки, цвета, года выпуска;

* Регистрация отправки автомобиля в ремонт;
* Регистрация прибытия автомобиля из ремонта;
* Регистрация выдачи клиенту автомобиля на прокат;
* Регистрация возврата автомобиля от клиентов.

**Алгоритмы и структуры данных**

Данные о клиенте содержаться в структуре «DATA»:

struct DATA

{

string licenseNum; // Номер водительского удостоверения.

string fullName; // ФИО.

string passportID; // Номер паспорта.

string adress; // Адрес проживания.

/\*Операторы сравнения\*/

bool operator > (const DATA);

bool operator < (const DATA);

bool operator >= (const DATA);

bool operator <= (const DATA);

bool operator == (const DATA);

bool operator != (const DATA);

void clear();

};

Данные о клиентах реализованы с помощью АВЛ-дерева. Его элемент- структура «node»

// Значения показателя сбалансированности узла

const int leftheavy = -1;

const int balanced = 0;

const int rightheavy = 1;

/\*структура для хранения информации о клиенте бюро проката (узел АВЛ дерева)\*/

struct node

{

/\*фромат записи licenseNum: RR AA NNNNNN\*/

/\*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).\*/

struct DATA

{

string licenseNum; // Номер водительского удостоверения.

string fullName; // ФИО.

string passportID; // Номер паспорта.

string adress; // Адрес проживания.

/\*Операторы сравнения\*/

bool operator > (const DATA);

bool operator < (const DATA);

bool operator >= (const DATA);

bool operator <= (const DATA);

bool operator == (const DATA);

bool operator != (const DATA);

void clear();

};

DATA data; //Открытый элемент для доступа к данным клиента

node(node::DATA,

node\* l = nullptr,

node\* r = nullptr,

int h = 0);

node(string licenseNum = "",

string fullName = "",

string passportID = "",

string adress = "",

node\* left = nullptr,

node\* right = nullptr,

//int height,

int height = 1);

private:

/\*Указатели на левого и правого потомка АВЛ дерева\*/

node\* left;

node\* right;

/\*balance- высота правого поддерева минус высота левого поддерева.\*/

//int height;

int height;

/\*Класс дерева должен иметь доступ ко всем полям узла\*/

friend class ClientTree;

};

Само АВЛ дерево реализовано в классе «ClientTree»

/\*Элементы сортируются по номеру водительского удостоверения (licenseNum)\*/

class ClientTree

{

//node\* root;

/\*повороты дерева\*/

node\* srl(node\*\*);

node\* brl(node\*\*);

node\* srr(node\*\*);

node\* brr(node\*\*);

/\*node\* getNewNode(node::DATA val,

node\* l = nullptr,

node\* r = nullptr);\*/

node\* balance(node\*);

int get\_height(node\*);

void height\_upd(node\*);

int bFactor(node\*);

node\* find\_min(node\*&);

node\* remove\_min(node\*&);

/\*Работа с пользователем для ввода информации о новом клиенте\*/

node::DATA createForm(bool\*);

public:

node\* root; //"корень дерева"

RentalList\* listOfRent; //переменна для просмотра списка аренды

ClientTree(RentalList\*);

~ClientTree();

node\* findBy\_licNum(node\*, const string, int\* err);

node\* findBy\_passportID(node\*, const string, int\* err);

void findBy\_SN(node\*, const string, list<node>\*);

node\* remove(node\*, const string, bool&);

bool isCorrect\_licenseNum(string);

bool isCorrect\_passportID(string, int\*);

void display\_tree(node\* R, int l = 0);

/\*Вывод клиента, при поиске по водительскому удостоверению\*/

void display\_elem(node R);

RentalListNode\* display\_elem\_wCar(node R);

/\*вывод ВТОРОЙ ЧАСТИ ДЛЯ ПОИСКА В ТАБЛИЦЕ\*/

void display\_treeScnd(RentalListNode\*);

void display\_set(list<node>);

bool isEmpty();

node\* add(node\*& r, node::DATA n);

void \_dltTree(node\*&);

node\* \_get\_root() { return root; };

void add\_ui();

void display\_ui();

void del\_ui();

void clear\_ui();

};

Данные об автомобилях хранятся в структуре «TableNode»- элементе хэш-таблицы.

#pragma once

#include "Includes.h"

/\*Элемент тхеш-такблицы хранящий информацию об автомобиле.\*/

struct TableNode

{

string car\_number; // «ANNNAA-NN» где A – буква из следующего множества: А,В,Е,К,М,Н,О,Р,С,Т,У,Х; N - цифры

string brand; // Марка машины

string color; // Цвет машины

int yearOfManufacture; // год выпуска

bool status = true; //признак наличия автомобиля

bool repair = false;

bool deleted = false;

TableNode();

TableNode(string refNum,

string brand,

string color,

int yOfManafact,

bool stat= true,

bool del= false);

void clear();

};

Сама хеш-таблица реализована в классе «HashTable»

/\*Организация хеш-таблицы\*/

class HashTable {

const int m = 9; //длина ключа

const int m1 = m+1; //длина ключа +1

const int tour = 6; //кол-во итераций поиска

// константа, используемая во второй хеш-функции (обязательное простое число)

const int PRIME = 7;

/\*Размер хеш таблицы\*/

int \_size;

/\*сама хеш таблица

Хрранит указатели на структуры данных авто.\*/

TableNode\* table;

int keyTo\_int(string);

int hash\_1(string);

int hash\_2(string);

void resize();

bool set\_repair(string inRN, bool rep\_stat);

public:

HashTable(RentalList\*);

~HashTable();

RentalList\* listOfRent;

/\*Очищает данные таблицы\*/

void clear();

/\*Проверяет,находятся ли элементы в таблице\*/

bool isEmpty();

/\*Ввод данных для регистрации авто\*/

TableNode createForm(bool\* retTMen);

/\*Добавлени элемента в таблицу\*/

bool add(TableNode);

/\*Работа с пользователем для регистрации авто\*/

void add\_ui(bool\* retTMen);

/\*Работа с пользователем для удалении авто с площадки\*/

void del\_ui();

/\*Удаление из таблицы\*/

bool del(string);

void clear\_ui();

/\*вывод всей таблицы\*/

void display();

/\*Вывод одного авто, при поиске по рег. номеру\*/

RentalListNode\* display(TableNode elem);

/\*Выыод элемента\*/

void display\_elem(TableNode);

/\*вывод ВТОРОЙ ЧАСТИ ДЛЯ ПОИСДЕРЕВЕ\*/

void display\_hashScnd(RentalListNode\* l);

/\*Вывод списка авто, при поиске по марке\*/

void display(list<TableNode>);

short findBy\_regNum(string);

TableNode findBy\_regNum\_elem(string);

list<TableNode>\* findBy\_brand(string);

/\*Регистрация авто в ремонт\*/

void repair\_login\_ui(RentalList\* r);

/\*Регистрация авто в ремонт\*/

void repair\_logout\_ui(RentalList\* r);

bool isStateRegNumCorrect(string);

};

Данные об аренде авто содержаться в структуре «RentalListNode»- элементе списка:

/\*Элемент списка rentalList\*/

struct RentalListNode

{

RentalListNode(string = "",

string = "",

string = "",

string = "",

RentalListNode\* = nullptr);

/\*stateRegNum совпадает с полем структуры для авто

clientLicenseNum совпадает с полем структуры клиента.\*/

string stateRegNum; //гос. рег. номер

string clientLicenseNum;

string date\_issue; //дата выдачи авто.

string date\_return; //дата возврата авто.

RentalListNode\* pNext;

RentalListNode operator = (const RentalListNode);

bool operator > (const RentalListNode);

bool operator < (const RentalListNode);

bool operator == (const RentalListNode);

bool operator != (const RentalListNode);

bool operator >= (const RentalListNode);

bool operator <= (const RentalListNode);

};

Сам список реализован в классе «RentalList»

/\*Список с информацией о прокате авто\*/

class RentalList

{

/\*хранение данных\*/

RentalListNode\* head; //указатель на голову списка

int \_size; //кол-во элементов в списке. ипользуется в сортировке.

public:

RentalList();

~RentalList();

int getSize() { return \_size; };

bool emplace\_front(string lic

, string regN

, string start

, string end);

bool remove(RentalListNode\*&);

void sort(int, int);

void out();

bool isEmpty() { return (head) ? false : true; };

void add\_ui();

RentalListNode\* findBy\_licNum(string);

RentalListNode\* findBy\_regNum(string);

/\*Вывод шапки таблицы на экран\*/

void display\_firstLine();

/\*Вывод строки с одним элементом на экран.\*/

void display\_elem(RentalListNode\*);

/\*Вывод списка на экран\*/

void display\_all();

void swap(RentalListNode\*, RentalListNode\*);

RentalListNode\* operator [] (int);

};

В ходе написания программы были использованы алгоритмы: сортировка Хоара для односвязного циклического списка, обратный обход дерева, Закрытое хеширование с двойным хешированием, прямой поиск в тексте.

Сортировка Хоара для односвязного списка (в данном случае сортирует элементы по возрастанию, располагая их от головы списка к хвосту).

Суть сортировки:

1. Примерно выбирается средний по значению элемент (между первым, последним и серединным по положению в списке) (далее: средний).
2. Далее происходит независимая последовательная проверка элементов с двух концов списка: проверка, движущаяся от головы к хвосту списка, ищет элементы, большие или равные среднему элементу (если такой найден- сохраняется его номер в списке). Аналогичный поиск происходит по направлению от хвоста к голове списка, только ищется элемент меньший среднего. Проверки происходят до тех пор, пока не «пройдут» друг друга, т.е. проверка от головы к хвосту начнёт проверять элементы на позициях, уже пройденных «встречной» проверкой и наоборот.
3. Если проверки остановились, но не «прошли друг друга», элементы, на которых остановились проверки меняются местами в списке.
4. Если проверки остановились и «прошли друг друга», то если обе проверки дошли до своих концов, то сортировка завершается- список отсортирован. Если правая проверка не дошла до головы, то рекурсивно запускается сортировка участка от головы до участка, на котором она остановилась. Для левой, соответственно: от того места, на котором она остановилась до хвоста списка.

Обратный обход дерева.

При прохождении дерева в обратном порядке сначала рекурсивно посещаются вершины левого поддерева, далее вершины правого поддерева, последним посещается корень.

Закрытое хеширование с двойным хешированием

Для борьбы с коллизиями используется двойное хеширование- хеширование, основанное на использовании второй хеш-функции для расчёта индивидуального отступа от первоначального хеша при возникновении коллизии. При переполнении таблицы она не позволяет записать в себя большее количество элементов.

Прямой поиск в тексте

Происходит поочерёдное сравнение исходного текста с искомым отрывком, до момента полного совпадения символов или до достижения конца текста (текст считается ненайденным, если ни разу не произошло полное совпадение.

**Описание программы**

Программа реализована на языке C++ в виде консольного приложения. В главной функции main() реализовано меню пользователя, в котором каждому действию соответствует определенная цифра. При выборе пункта, вызывается функция отвечающая за данный пункт.

1. Информация о клиентах
   1. Зарегистрировать

void ClientTree::add\_ui()

Функция добавляет информацию о клиенте в дерево.

Для этого пользователю предлагается ввести ФИО клиента, номер лицензии, серию и номер паспорта, адрес проживания. Введенные данные проверяются и записываются в структуру. Если данные не удовлетворяют проверкам, то выводится соответствующее предупреждение и пользователю предлагается попробовать ввести данные еще раз. Если все данные удовлетворяют проверкам, то при помощи вспомогательной функции, запись по номеру и серии паспорта добавляется в дерево, после чего, при необходимости, происходит балансировка дерева.

* 1. Удалить аккаунт

void ClientTree::del\_ui()

Функция удаляет информацию о клиенте из дерева.

Пользователю предлагается ввести номер водительского удостоверения человека, которого необходимо удалить из программы. Если клиента с таким водительским удостоверением не нашлось или этот клиент арендует автомобиль, на экран выводится соответствующее предупреждение, программа возвращается в главное меню. Если введенные данные клиента удовлетворяют проверкам, то данные о клиенте удаляются из дерева, после чего, при необходимости, происходит балансировка дерева.

* 1. Все клиенты

void ClientTree::display\_tree(node\* R, int l)

Функция выводит информацию обо всех зарегистрированных клиентах в форме «дерева». Удалось реализовать с помощью симметричного обхода дерева из-за особенности вывода на экран (консольное приложение).

* 1. Поиск клиента по номеру водительского удостоверения

node\* ClientTree::findBy\_licNum(node\* curRoot, const string value, int\* err)

Функция осуществляет поиск по водительскому удостоверению.

Для запуска необходимо выбрать пункт «найти» и выбрать соответствующий пункт.

Для этого пользователю предлагается ввести номер и серию водительского удостоверения. Происходит стандартный для АВЛ-дерева алгоритм поиска. Если пользователь с таким водительским удостоверением не найден, то выводится соответствующее оповещение. Если пользователь с таким водительским удостоверением найден, то в виде таблицы выводятся данные о пользователе.

* 1. Поиск клиента по фрагменту ФИО

void ClientTree::findBy\_SN(node\* curRoot, const string value, list<node>\* res)

Функция осуществляет поиск по фрагменту ФИО при помощи прямого поиска.

Пользователю предлагается ввести фрагмент искомого ФИО клиента, после чего происходит обратный обход всего дерева. Для ФИО каждого клиента проводится прямой поиск по фрагменту текста. Поиск доступен только при величине фрагмента более чем 1 символ. Если фрагмент не найден, то выводится соответствующее оповещение. Если клиенты с такими ФИО найдены (или содержат введённый для поиска текст), то их данные выводятся на экран в виде таблицы.

* 1. Очистить дерево клиентов

treeOfClients->clear\_ui();

Функция удаляет информацию обо всех клиентах из дерева.

Для этого, происходит обратный обход всего дерева и удаление каждого элемента начиная с конца дерева до корня.

* 1. Вывести данные паспортов в виде дерева

void Show\_tree(Tree\* R, int l)

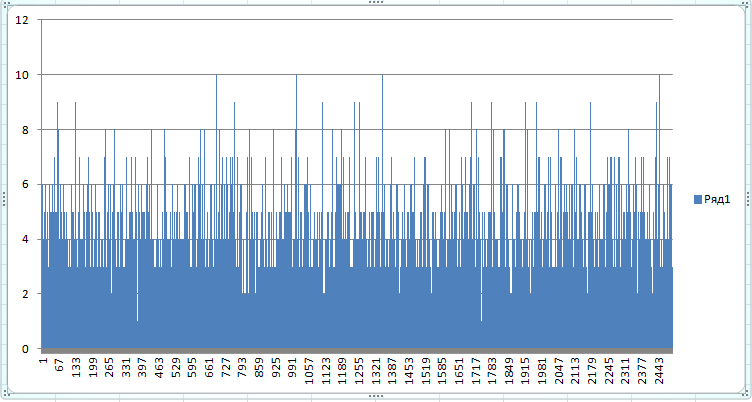
Функция выводит данные паспортов всех клиентов в виде дерева. Для этого при помощи рекурсии происходит обход дерева, начиная с его корня.

1. Информация об автомобилях, принадлежащих бюро
   1. Зарегистрировать авто

void HashTable::add\_ui(bool\* returnToMenu)

Функция просит пользователя ввести регистрационный номер авто, его марку, цвет и год производства. Если автомобиль с таким номером уже зарегистрирован, выводится соответствующее сообщение, иначе происходит добавление ссылки на структуру с данными об авто в хеш-таблицу по рег.номеру авто. Если таблица считается переполненной (6 раз провалилась попытка вставить эленет), выводится сообщение о переполнении таблиц и добавление не происходит.

Номер в таблице вычисляется по первой хеш-функции. Её гистограмма коллизий показана на рис. 1. Если место занято, то шаг до следующего номера вычисляется по второй хеш-функции (в этом заключается суть закрытого двойного хеширования).

****

1. гистограмма коллизий
   1. Удаление автомобиля

void HashTable::del\_ui()

Функция удаляет информацию об автомобиле из хеш-таблицы.

Для этого пользователю предлагается ввести номер автомобиля (происходит проверка на наличие номера в базе, не сдан ли автомобиль в аренду), если автомобиль не удовлетворяет хотя бы одной из проверок, то удаление не происходит, выводится соответствующее предупреждение. Если введенный номер автомобиля удовлетворяют проверкам, то данные об автомобиле удаляются из линейного списка и хеш-таблицы (в хеш-таблице ставится метка удаления, на месте удаленного элемента).

* 1. Просмотр всех имеющихся автомобилей

void HashTable::display()

Функция выводит на экран регистрационные номера авто и соответствующие им номера в хеш- таблице.

* 1. Поиск автомобиля по регистрационному номеру

TableNode HashTable::findBy\_regNum\_elem(string k)

Необходимо выбрать пункт «найти», а затем выбрать соответствующий пункт для поиска.

Функция осуществляет поиск автомобиля по номеру в хеш-таблице. Пользователю предлагается ввести номер искомого автомобиля, после чего происходит преобразование номера при помощи хэш-функции и переход к элементу хеш-таблицы по полученному хешу. В случае перехода к метке удаления, происходит преобразование при помощи второй хеш-функции до тех пор, пока функция не наткнется на пустое поле без метки удаления, либо на поле с соответствующим номером автомобиля. Если номер не найден, то выводится соответствующее оповещение. Если номер совпал, то на экран выводится вся информация о найденном автомобиле + информация об арендующем клиенте (при наличии).

* 1. Поиск автомобиля по марке

void FindCar\_brand(int len)

Функция осуществляет поиск автомобилей по фрагменту марки.

Пользователю предлагается ввести фрагмент марки искомых автомобилей, после чего происходит проход по всей хеш-таблицы и сравнение марки в поле с искомой. Если ни одна марка автомобилей не совпала с искомой, то выводится соответствующее оповещение. Если марка совпала, данный автомобиль добавляется во вспомогательный список. После окончания прохода по таблице происходит вывод данного списка на экран (всех данных об авто) или соответствующее сообщение, если список пуст. Вспомогательный список удаляется.

* 1. Очистка данных об автомобилях

void HashTable::clear()

При подтверждении пользователя, старая таблица удаляется, а на её месте создаётся новая, тем самым, убирая все данные и метки удаления из таблицы.

1. Выдача на прокат и возврат автомобиля
   1. Регистрация аренды клиентом авто.

void rentCar\_start(HashTable\* tab, ClientTree\* tree, RentalList\* rental)

Пользователю предлагается ввести номер водительского удостоверения клиента. Если клиент не зарегистрирован в программе или уже арендует авто, то предлагается повторный ввод. Далее запрашивается ввод рег номера авто. Если авто с таким номером не зарегистрировано, или его уже арендуют, или оно находится в ремонте, выводится соответствующее сообщение, происходит повторный запрос ввода. Затем требуется ввести дату начала и окончания аренды.

При выполнении всех условий происходит добавление в список строковых значений: регистрационного номера авто, номера лицензии клиента и сроков аренды. Затем происходит сортировка списка методом Хоара по регистрационному номеру автомобиля в порядке возрастания.

* 1. Регистрация возврата автомобиля от клиентов

void Car\_return(Rent\*& head)

Функция осуществляет возврат автомобиля из проката. Пользователю предлагается ввести номер автомобиля, который необходимо вернуть. Происходит проверка на соответствие формату, в случае несоответствия пользователю предлагается ввести номер повторно. В случае соответствия формату происходит проверка на то, зарегистрирован ли автомобиль в базе, в случае отсутствия выводится соответствующее оповещение. Если же автомобиль зарегистрирован, то происходит проверка на то, находится ли автомобиль в аренде, если автомобиль находится в аренде, то происходит возврат и обновление данных таблицы, иначе выводится соответствующее оповещение. Происходит выход в главное меню.

* 1. Регистрация отправки автомобиля в ремонт

void Send\_for\_repair(Rent\*& head)

Функция осуществляет отправку автомобиля в ремонт. Пользователю предлагается ввести номер автомобиля, который необходимо отправить в ремонт. Происходит проверка на соответствие формату, в случае несоответствия пользователю предлагается ввести номер повторно. В случае соответствия формату происходит проверка на то, зарегистрирован ли автомобиль в базе, в случае отсутствия выводится соответствующее оповещение. Если же автомобиль зарегистрирован, то происходит проверка на то, свободен ли он. Если автомобиль не находится в прокате и уже не сдан в ремонт, то производится отправка в ремонт, в таблицу заносятся обновленные данные, иначе выводится соответствующее оповещение. Происходит выход в главное меню.

* 1. Регистрация принятия автомобиля из ремонта

void rentCar\_end(HashTable\* tab, ClientTree\* tree, RentalList\* rental)

Функция осуществляет возврат автомобиля из ремонта.

Функция запрашивает ввод рег. Номера авто. Если по такому номеру был найден автомобиль в списке аренды, то данная запись оттуда убирается. Иначе, выводится соответствующее сообщение.

* 1. Показать все автомобили в бюро

void RentalList::display\_all()

Функция выводит данные, содержащиеся в списке, в виде таблицы, содержащей поля: паспорт, номер автомобиля, дата выдачи, дата возврата, ремонт.

* 1. Отправить автомобиль в ремонт.

void HashTable::repair\_login\_ui(RentalList\* r)

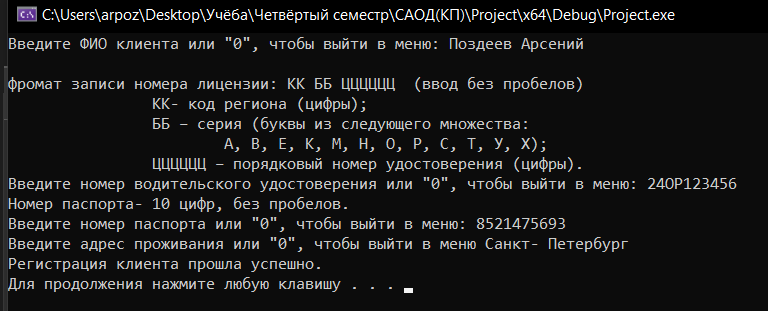
Функция запрашивает ввод регистрационного номера автомобиля для отправки в ремонт. Если такой автомобиль не зарегистрирован в программе или сотсоит в активном договоре аренды, то выводится соответствующее сообщение и автомобиль в ремонт не отправляется. В ином случае, в данных авто ставится пометка о постановке автомобиля в ремонт.

* 1. Забрать автомобиль из ремонта.

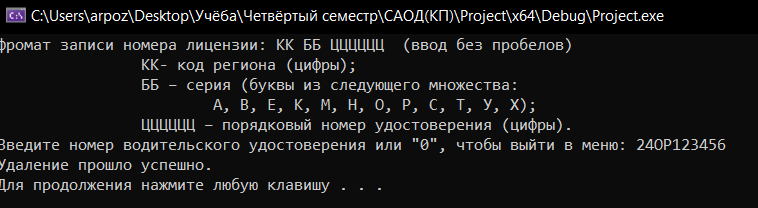
void HashTable::repair\_logout\_ui(RentalList\* r)

Функция запрашивает ввод регистрационного номера автомобиля. Если такой автомобиль не зарегистрирован в программе или не стоит в ремонте, выводится соответствующее сообщение, и программа возвращается в меню. Иначе, в данных авто ставится пометка о ненахождении в ремонте.

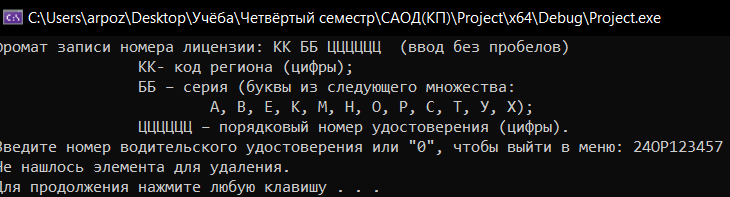
**Тестирование**



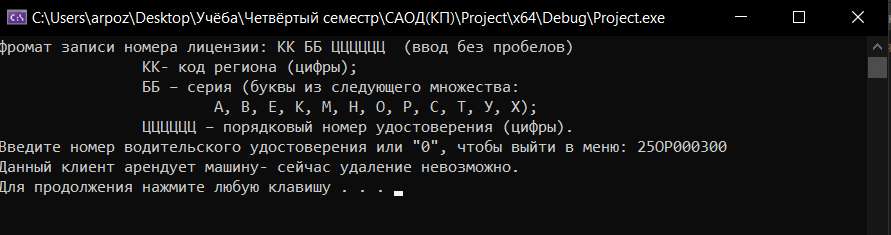
1. регистрацию нового клиента;



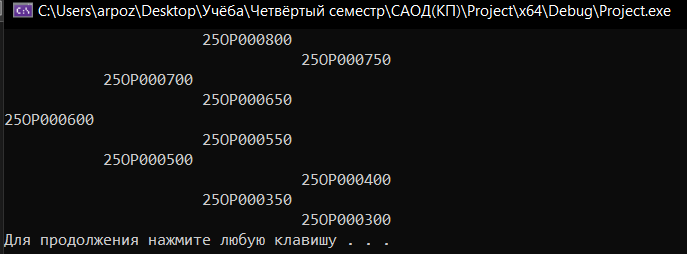
1. снятие с обслуживания клиента;



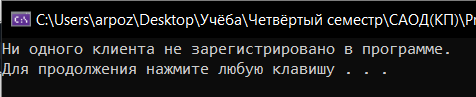
1. снятие с обслуживания несуществующего клиента;



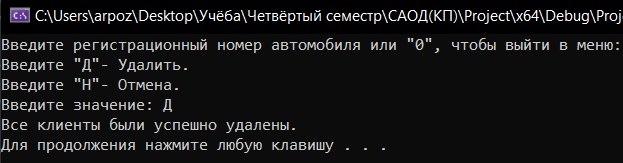
1. снятие с обслуживания клиента, арендовавшего автомобиль;



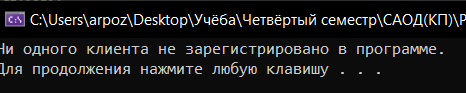
1. просмотр всех зарегистрированных клиентов;



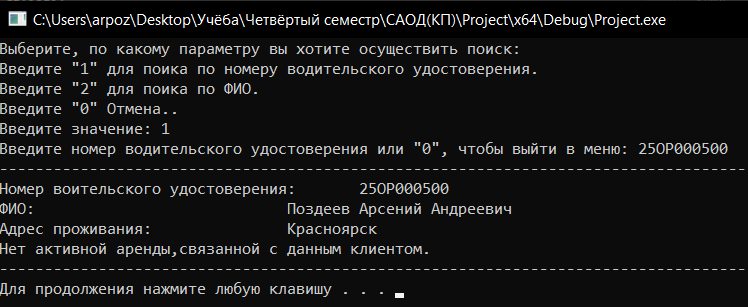
1. просмотр всех зарегистрированных клиентов при пустом списке;



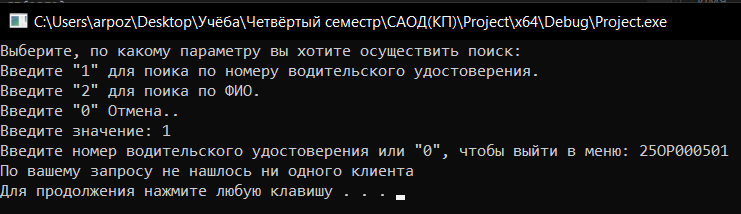
1. очистка данных о клиентах;



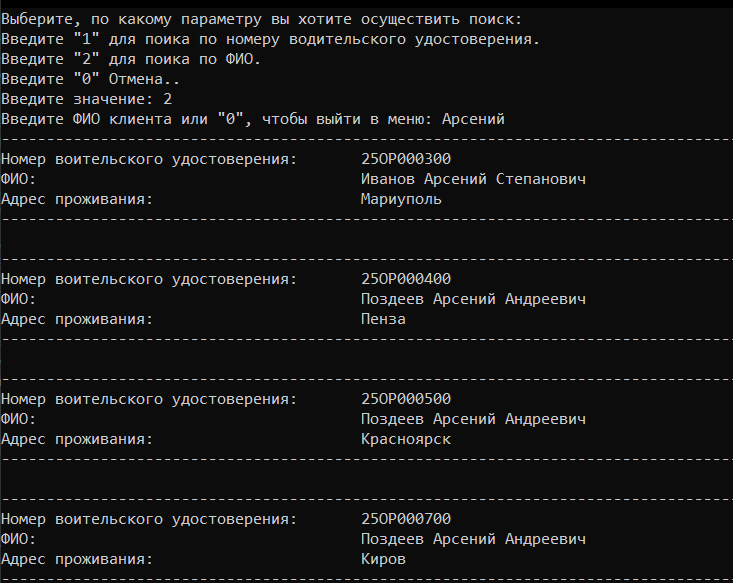
1. очистка данных о клиентах при пустом списке;



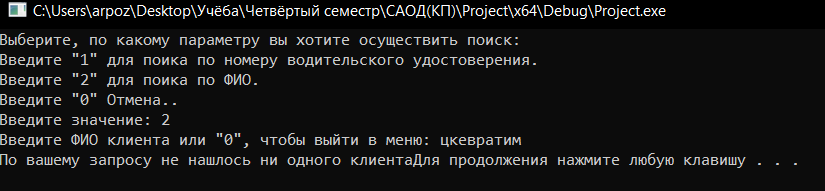
1. поиск клиента по номеру водительского удостоверения;



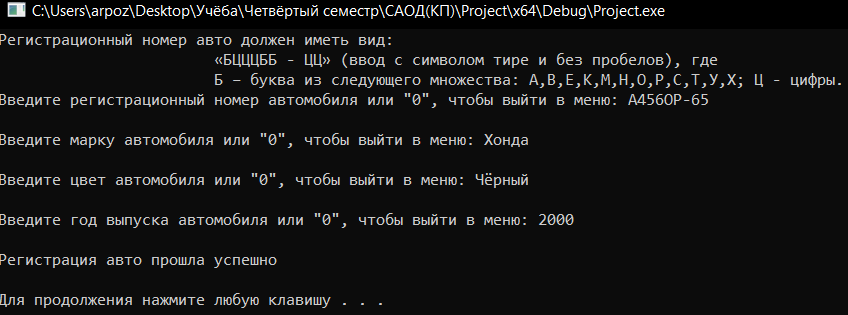
1. поиск клиента по несуществующему номеру водительского удостоверения;



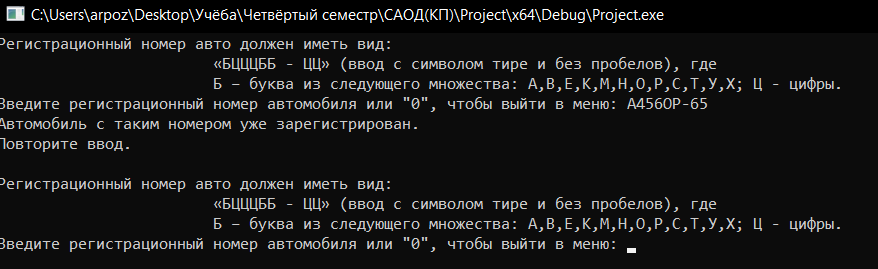
1. поиск клиента по фрагменту ФИО;



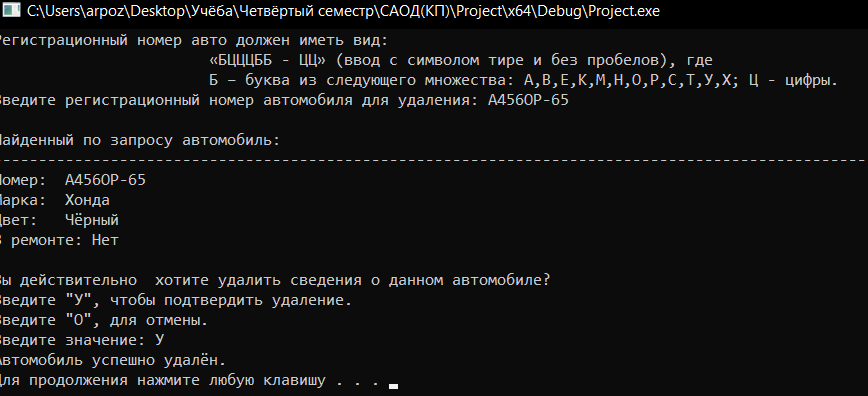
1. поиск клиента по несуществующему в базе фрагменту ФИО;



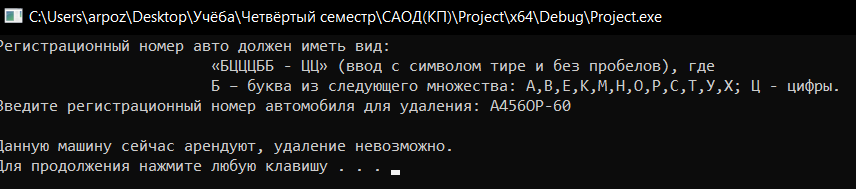
1. добавление нового автомобиля;



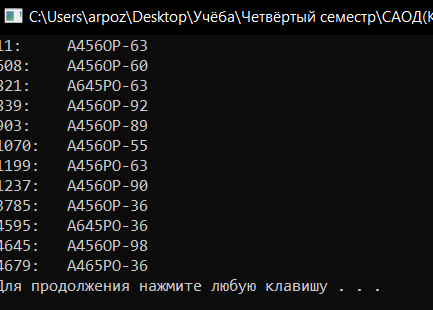
1. добавление уже существующего в базе автомобиля;



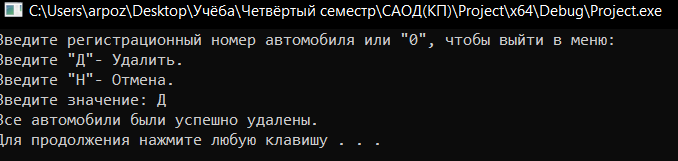
1. удаление сведений об автомобиле;



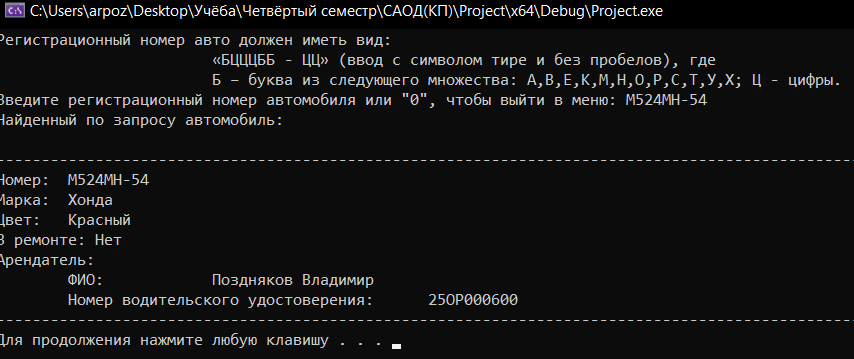
1. удаление сведений об автомобиле, сданном в аренду клиенту



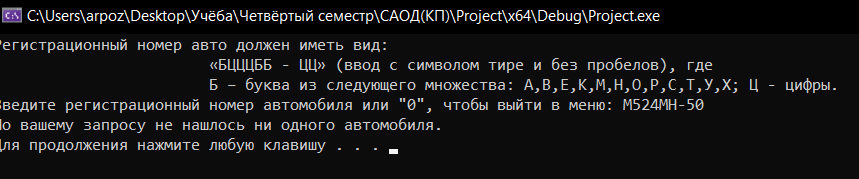
1. просмотр всех имеющихся автомобилей;



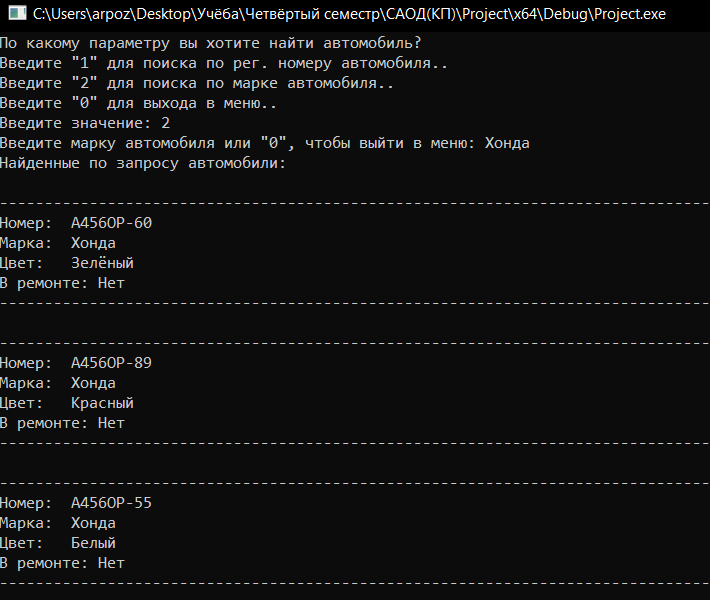
1. очистка данных об автомобилях;



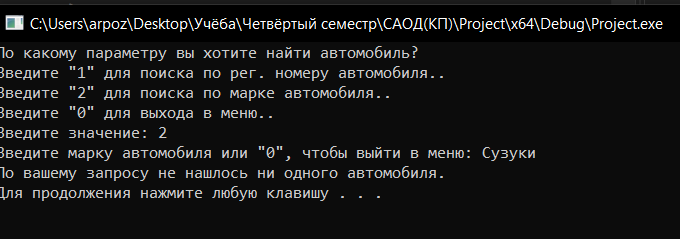
1. поиск автомобиля по «Государственному регистрационному номеру».



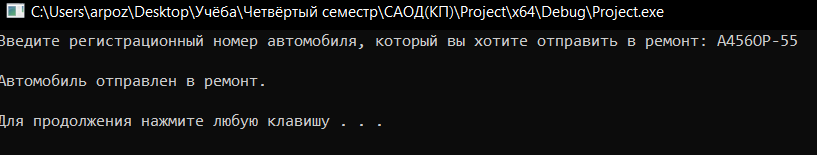
1. поиск автомобиля по «Государственному регистрационному номеру», который не занесен в базу.



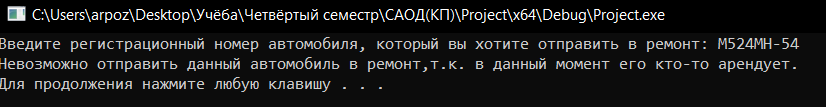
1. поиск автомобиля по названию марки автомобиля.



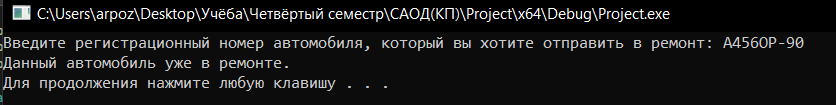
1. поиск автомобиля по названию марки автомобиля, не занесенной в базу.



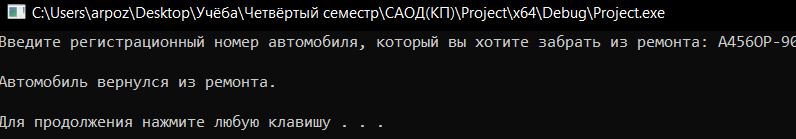
1. регистрация отправки автомобиля в ремонт;



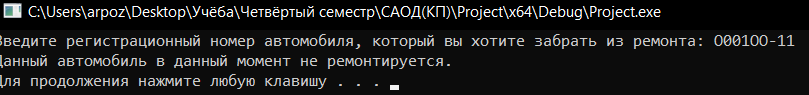
1. регистрация отправки автомобиля, сданного в аренду, в ремонт;



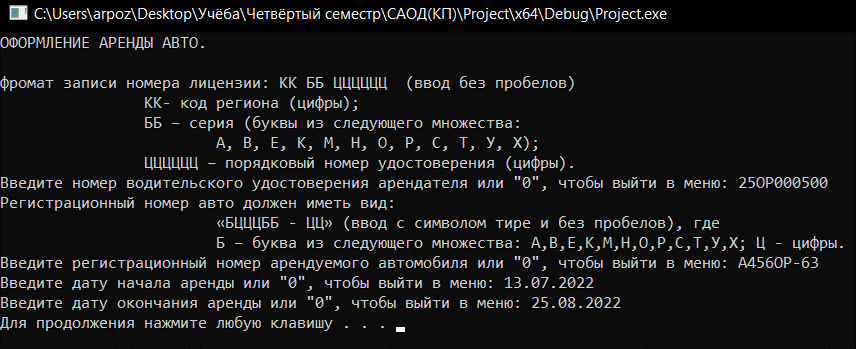
1. регистрация отправки в ремонт автомобиля, уже находящегося там;



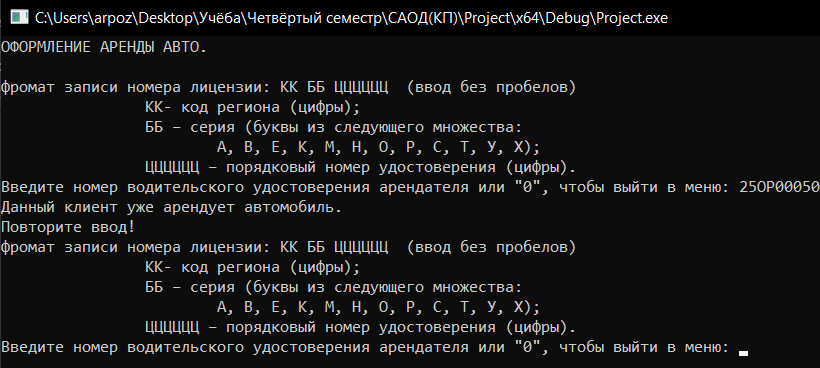
1. регистрация прибытия автомобиля из ремонта;



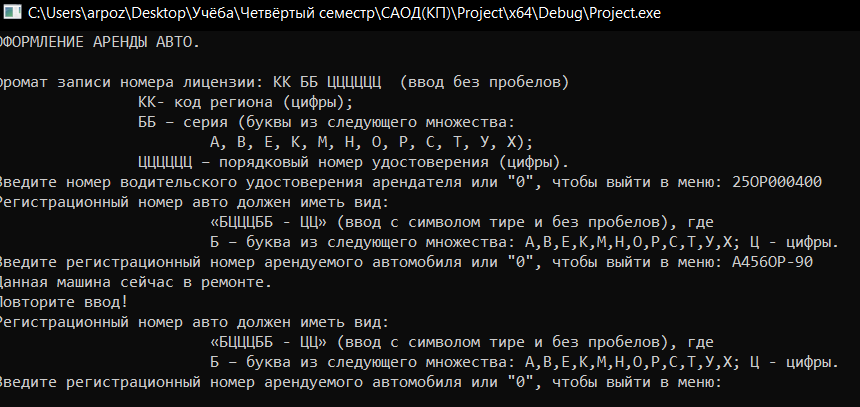
1. регистрация прибытия из ремонта автомобиля, находящегося в аренде;



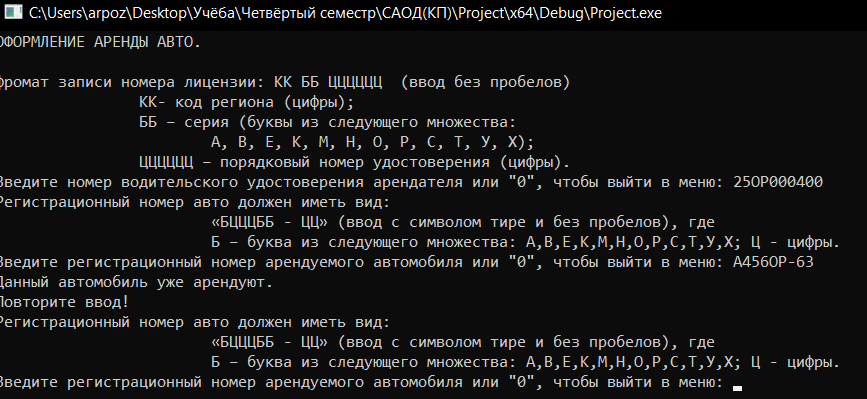
1. регистрация выдачи клиенту автомобиля на прокат;



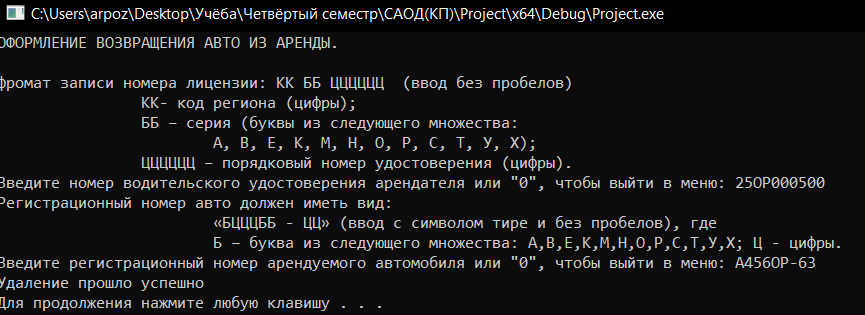
1. регистрация выдачи клиенту второго автомобиля на прокат;



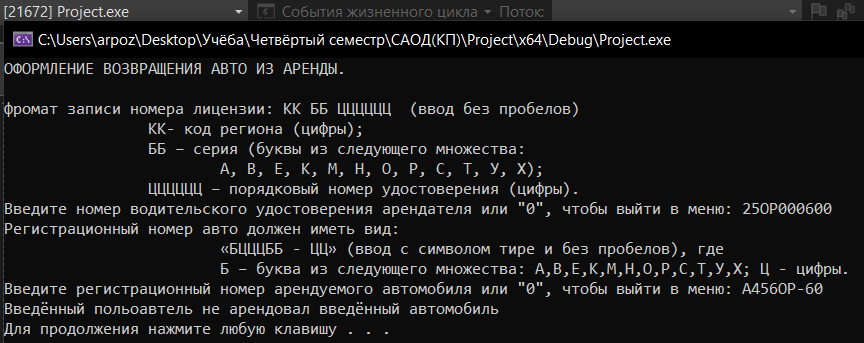
1. регистрация выдачи клиенту на прокат автомобиля, находящегося в ремонте;



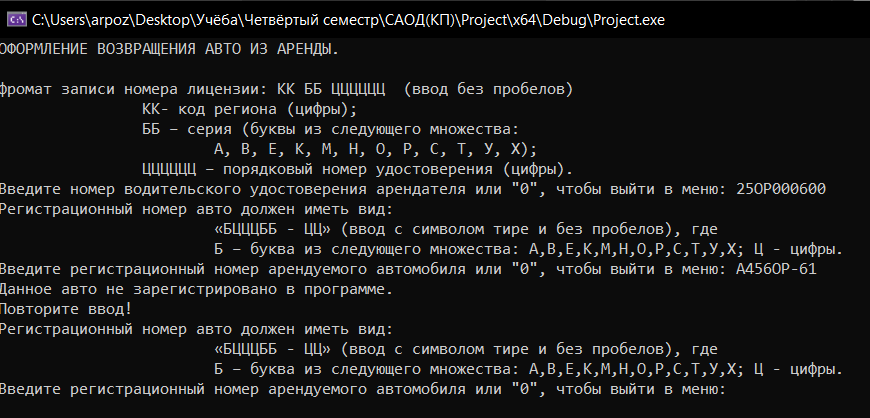
1. регистрация выдачи клиенту на прокат автомобиля, уже сданного в аренду;



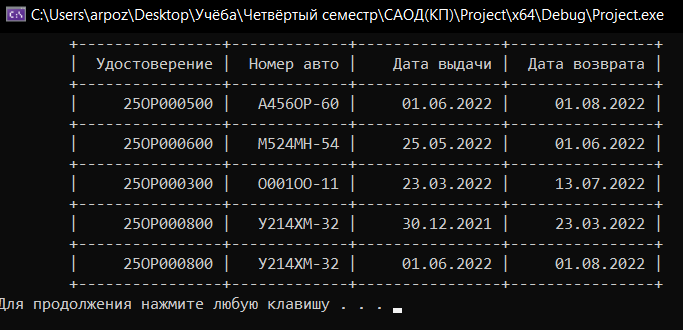
1. регистрацию возврата автомобиля от клиента;



1. регистрацию возврата от клиента не арендованного автомобиля;



1. регистрацию возврата от клиента автомобиля, не занесенного в базу;



1. список автомобилей в бюро

**Заключение**

В процессе выполнения курсового проекта мною была разработана информационная система – «Обслуживание клиентов в бюро проката автомобилей». При написании данного кода я применила знания, полученные на дисциплине «Структуры и алгоритмы обработки данных», а также закрепила их.

**Список использованной литературы**

1. В. А. Матьяш, С. А. Рогачев. Алгоритмы и структуры данных: учеб. пособие СПб.: ГУАП, 2021.
2. Матьяш В. А., Рогачев С. А. Структуры и алгоритмы обработки данных: учеб.-метод. пособие. СПб.: ГУАП, 2017.
3. Ключарев А. А., Матьяш В. А., Щекин С. В. Структуры и алгоритмы обработки данных: учеб. пособие. СПб.: ГУАП, 2004.

**ПРИЛОЖЕНИЕ**

Includes.h

#pragma once

#define \_CRT\_SECURE\_NO\_WARNINGS //т.к. используются ф-ции ctime

#include <iostream>

#include <string>

#include <Windows.h>

#include <list>

#include <sstream>

#include <ctime>

#include <iomanip>

#undef max

using std::cout;

using std::cin;

using std::endl;

using std::string;

using std::pair;

using std::list;

using std::stringstream;

using std::setw;

using std::setfill;

/\*list of pair of strings\*/

typedef list<pair<string, string>> lopos;

/\*Функция возвращает текст ошибки по коду\*/

static string errorName(int index)

{

/\*1-2 цифры: номер группы ошибки

3-4 фицры- номер ошибки\*/

switch (index)

{

/\*10- работа с АВЛ деревом\*/

/\*11- работа с Хэш-таблицей\*/

/\*12- работа со списком\*/

/\*13-Ошибки ввода\*/

case(1001): return "Ни одного клиента не зарегистрировано в программе.";

case(1002): return "По вашему запросу не нашлось ни одного клиента";

case(1003): return "Неверно введено водительское удостоверение";

case(1004): return "Клиент с таким водительским удостоверением уже зарегистрирован.";

case(1005): return "Клиент с такими паспортными данными уже зарегистрирован.";

case(1006): return "Клиент с таким водительским удостовенением в программе не зарегистрирован.";

case(1007): return "В ТАБЛИЦЕ СОДЕРЖАТСЯ НЕКОРРЕКТНЫЕ ДАННЫЕ!";

case(1008): return "Нет активной аренды,связанной с данным клиентом.";

case(1101): return "В программе не зарегистрировано ни одного автомобиля.";

case(1102): return "По вашему запросу не нашлось ни одного автомобиля.";

case(1103): return "Неверно введён регистрационный номер авто.";

case(1104): return "Автомобиль с таким номером уже зарегистрирован.";

case(1105): return "Автомобиль с таким номером не зарегистрирован.";

case(1106): return "Данный автомобиль уже арендуют.";

case(1107): return "Данный автомобиль уже в ремонте.";

case(1108): return "Данный автомобиль в данный момент не ремонтируется.";

case(1109): return "Нет активной аренды,связанной с данным автомобилем.";

case(1201): return "Данный клиент уже арендует автомобиль.";

case(1202): return "Впрограмме недостаточно данных для начала аренды.";

case(1203): return "Данная машина сейчас в ремонте.";

case(1204): return "Невозможно отправить данный автомобиль в ремонт,т.к. в данный момент его кто-то арендует.";

case(1205): return "Ни одна аренда сейчас не действительна.";

case(1206): return "Данную машину сейчас арендуют, удаление невозможно.";

case(1207): return "Данный клиент арендует машину- сейчас удаление невозможно.";

case(1301): return "Вы должны были ввести одно из предложенных значений.";

case(1302): return "Введено некорректное значение.";

case(1303): return "Данное значение не может быть короче 2 символов.";

case(1304): return "Данное значение может состоять только из букв.";

case(1305): return "В середине слова не может стоять заглавная буква.";

case(1306): return "Введите реальное значение года производства автомобиля.";

/\*Ввод ФИО\*/

case(1307): return "ФИО может состоять минимум из трёх букв!";

case(1308): return "В начале слова, после дефиса или пробела в ФИО должна стоять большая буква, но не \'Ъ\',\'Ь\'!";

case(1309): return "ФИО может состоять только из букв русского алфавита и из символов дефиса и(/или) пробела.";

case(1310): return "ФИО не может оканчиваться на что-либо кроме буквы.";

case(1311): return "ФИО должно состоять минимум из двух слов: Имени и фамилии.";

/\*номер паспорта\*/

case(1312): return "Номер паспорта должен состоять из 10 цифр без пробелов!";

default:

return "Неизвестная ошибка";

}

}

/\*Функция возвращает текст предупреждения(/сообщения) по коду\*/

static string warningName(int tag)

{

/\*1-2 цифры: номер группы предупреждения

3-4 фицры- номер прежупреждения\*/

switch (tag)

{

/\*10- работа с АВЛ деревом\*/

/\*11- работа с Хэш-таблицей\*/

/\*12- работа со списком\*/

/\*13-предупредения при ввода ввода\*/

/\*99- общие предупреждения (для взаиможействия с программой).\*/

case(1001): return "фромат записи номера лицензии: КК ББ ЦЦЦЦЦЦ (ввод без пробелов)\n\

КК- код региона (цифры);\n\

ББ – серия (буквы из следующего множества:\n\

А, В, Е, К, М, Н, О, Р, С, Т, У, Х);\n\

ЦЦЦЦЦЦ – порядковый номер удостоверения (цифры).";

case(1002): return "Номер паспорта- 10 цифр, без пробелов.";

case(1101): return "Регистрационный номер авто должен иметь вид:\n\

«БЦЦЦББ - ЦЦ» (ввод с символом тире и без пробелов), где\n\

Б – буква из следующего множества: А,В,Е,К,М,Н,О,Р,С,Т,У,Х; Ц - цифры.";

case(1102): return "Найденный по запросу автомобиль:";

case(1103): return "Найденные по запросу автомобили:";

case(9901): return "Возвращение в меню...";

case(9902): return "Продолжение ввода...";

case(9903): return "Внимание! Вы не до конца ввели данные. При выходе в меню эти данные не сохранятся.";

default: return "Неопознанное предупреждение.";

break;

}

}

/\*Ф-ция, возвращающая строку запроса пользователю.\*/

static string askName(int tag)

{

/\*1-2 цифры: номер группы запроса

3-4 фицры- номер запроса\*/

switch (tag)

{

/\*10- работа с АВЛ деревом\*/

/\*11- работа с Хэш-таблицей\*/

/\*12- работа со списком\*/

/\*13-Мольбы о помощи\*/

case(1001): return "Введите ФИО клиента или \"0\", чтобы выйти в меню:";

case(1002): return "Введите номер водительского удостоверения или \"0\", чтобы выйти в меню:";

case(1003): return "Введите номер паспорта или \"0\", чтобы выйти в меню:";

case(1004): return "Введите адрес проживания или \"0\", чтобы выйти в меню";

case(1101): return "Введите регистрационный номер автомобиля или \"0\", чтобы выйти в меню:";

case(1102): return "Введите марку автомобиля или \"0\", чтобы выйти в меню:";

case(1103): return "Введите цвет автомобиля или \"0\", чтобы выйти в меню:";

/\*Без "Введите, т.к. используется в ф-ции inputAndCheck\_int"\*/

case(1104): return "год выпуска автомобиля или \"0\", чтобы выйти в меню";

case(1105): return "Вы действительно хотите удалить все автомобили, хранящиеся в программе?";

case(1106): return "Введите регистрационный номер автомобиля для удаления:";

case(1107): return "Вы действительно хотите удалить сведения о данном автомобиле?";

case(1108): return "Введите регистрационный номер автомобиля, который вы хотите отправить в ремонт:";

case(1109): return "Введите регистрационный номер автомобиля, который вы хотите забрать из ремонта:";

case(1201): return "Введите номер водительского удостоверения арендателя или \"0\", чтобы выйти в меню:";

case(1202): return "Введите регистрационный номер арендуемого автомобиля или \"0\", чтобы выйти в меню:";

case(1203): return "Введите дату начала аренды или \"0\", чтобы выйти в меню:";

case(1204): return "Введите дату окончания аренды или \"0\", чтобы выйти в меню:";

case(1301): return "Вы уверены, что хотите выйти?";

default:

return "Неизвестный запрос.";

}

}

/\*Возвращает порядковый номер выбранного действия\*/

static short int chooseDoing(string question, lopos variants)

{/\*пары variants- first- что ввести, second- что произойдёт

Это тект для восприятия пользователем.

Не используется контейнери map, т.к. он автоматически сортирует данные,

хранящиеся в variants\*/

string inChoice("\0"); //выбор пользователя.

int counter; //порядковый номер, выбранного варианта.

while (true)

{

cout << question << endl;

for (auto i = variants.begin(); i != variants.end(); i++)

{

cout << "Введите \"" << i->first << "\"" << i->second << ".\n";

}

cout << "Введите значение: ";

cin;

getline(cin, inChoice, '\n');

counter = 0;

/\*Поиск совпадений введённого значения с одним из предложенных\*/

for (auto i = variants.begin(); i != variants.end(); i++, counter++)

{

if (inChoice == i->first)

return counter;

}

cout << errorName(1301) << "\nПовторите ввод!\n";

continue;

}

}

/\*проверка ввода числа с клавиатуры.\*/

static int inputAndCheck\_int(string varName)

{

int result(0);

while (1) { //ввод до корректного значения.

cout << "Введите " << varName << ": ";

/\*Очистка буффера ввода перед запрососом ввода\*/

/\*cin.clear();

cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');\*/

cin >> result;

cout << endl;

/\*корректность ввода (является ли числом).\*/

if (cin.fail() || cin.peek() != '\n') {

cin.clear();

cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');

cout << errorName(1302) << "Введите число ещё раз!\n\n";

continue;

}

break;

}

cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');

return result;

}

/\*Порверка на строку (толкьо из букв)\*/

static bool isStrCorrect(string str,int\* err)

{/\*1. не менее 2 символов

2. Только из букв русского алфавита

3. В центре слова не может стоять заглавная буква.\*/

const string forbidderSymbols = { 'Ь', 'Ъ' }; //запрещённые символы для первой буквы.

if (str.length() < 2)

{

\*err = 1303;

return false;

}

for (int i(0); i < str.length(); i++)

{

/\*Проверка первого символа\*/

if (!(str[0] >= 'А' && str[0] <= 'Я' || str[0] == 'Ё' ||

str[0] >= 'а' && str[0] <= 'я' || str[0] == 'ё') ||

int(forbidderSymbols.find(str[0])) != -1)

{

\*err = 1304;

return false;

}

/\*проверка остальных символов\*/

for (int i(1); i < str.length(); i++)

{

/\*встретилось что-то кроме мал. буквы\*/

if (!(str[i] >= 'а' && str[i] <= 'я' || str[i] == 'ё'))

{

\*err = 1304;

/\*большая буква в середине слова\*/

if (str[i] >= 'А' && str[i] <= 'Я' || str[i] == 'Ё')

{

\*err = 1305;

}

return false;

}

}

}

return true;

}

/\*Корректно ли ФИО\*/

static bool isNameCorrect(string city, int\* error)

{/\*error- возвращаемый код ошибки.\*/

bool isSignBehind(true); //флаг поднимается, если предыдущий символ был девис или пробел.

int spaces(0);

if (city.length() < 3) //если название слишком короткое

{

\*error = 1307;

return false;

}

for (int i(0); i < city.length(); i++) //проверка всех символов.

{

if (isSignBehind && !((city[i] >= 'А' && city[i] <= 'Я' || city[i] == 'Ё') && city[i] != 'Ъ' && city[i] != 'Ь' || city[i] == ' ')) //если после дефиса/ пробела не стоит б. буква

{

\*error = 1308;

return false;

}

if (!(city[i] >= 'А' && city[i] <= 'Я' || city[i] == 'Ё' || city[i] >= 'а' && city[i] <= 'я' || city[i] == 'ё'

|| city[i] == ' ' || city[i] == '-')) //можно только буквы, девисы и пробелы использовать.

{

\*error = 1309;

return false;

}

(city[i] == ' ' || city[i] == '-') ? (isSignBehind = true) : (isSignBehind = false);

if (city[i] == ' ') spaces++;

}

if (isSignBehind) //если в конце слова не буква, а знак (деф./ проб.)

{

\*error = 1310;

return false;

}

if (spaces > 2 || spaces == 0)

{

\*error = 1311;

return false;

}

return true;

}

/\*Алгоритм прямого поиска по фрагменту текста в строке\*/

static int findInText(string text, string str)

{/\*если фрагмент не был найден- возвращается -1\*/

int pos(0); //позиция первого символа в тексте

bool coincidence(false);

/\*пока последний символ фрагмента не выйдет за пределы текста

или до нахждения полного совпадения\*/

while (pos + str.length() <= text.length())

{

coincidence = false;

/\*проверка каждого символа фрагмента текста\*/

for (int i(0); i < str.length(); i++)

{

if (str[i] == text[pos + i])

coincidence = true;

else

{

coincidence = false;

break;

}

}

/\*нашли фрагмент в тексте\*/

if (coincidence)

return pos;

/\*продолжаем поиск\*/

pos++;

}

/\*Фрагмента в тексте нет\*/

return -1;

}

node.h

#pragma once

#include "Includes.h"

// Значения показателя сбалансированности узла

const int leftheavy = -1;

const int balanced = 0;

const int rightheavy = 1;

/\*структура для хранения информации о клиенте бюро проката (узел АВЛ дерева)\*/

struct node

{

/\*фромат записи licenseNum: RR AA NNNNNN\*/

/\*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).\*/

struct DATA

{

string licenseNum; // Номер водительского удостоверения.

string fullName; // ФИО.

string passportID; // Номер паспорта.

string adress; // Адрес проживания.

/\*Операторы сравнения\*/

bool operator > (const DATA);

bool operator < (const DATA);

bool operator >= (const DATA);

bool operator <= (const DATA);

bool operator == (const DATA);

bool operator != (const DATA);

void clear();

};

DATA data; //Открытый элемент для доступа к данным клиента

node(node::DATA,

node\* l = nullptr,

node\* r = nullptr,

int h = 0);

node(string licenseNum = "",

string fullName = "",

string passportID = "",

string adress = "",

node\* left = nullptr,

node\* right = nullptr,

//int height,

int height = 1);

private:

/\*Указатели на левого и правого потомка АВЛ дерева\*/

node\* left;

node\* right;

/\*balance- высота правого поддерева минус высота левого поддерева.\*/

//int height;

int height;

/\*Класс дерева должен иметь доступ ко всем полям узла\*/

friend class ClientTree;

};

node.cpp

#include "node.h"

node::node(node::DATA value,

node\* l,

node\* r,

int h)

{

this->data = value;

left = l;

right = r;

height = h;

}

node::node(string lic,

string fName,

string pass,

string adr,

node\* l,

node\* r,

int h)

{

/\*Значащие поля\*/

data.licenseNum = lic;

data.fullName = fName;

data.passportID = pass;

data.adress = adr;

/\*указатели на потомков\*/

left = l;

right = r;

/\*баланс фактор\*/

height = 0;

}

bool node::DATA::operator > (const DATA r)

{

return (this->licenseNum > r.licenseNum) ? true : false;

}

bool node::DATA::operator < (const DATA r)

{

return (this->licenseNum < r.licenseNum) ? true : false;

}

bool node::DATA::operator >= (const DATA r)

{

if (this->licenseNum > r.licenseNum ||

this->licenseNum == r.licenseNum)

return true;

return false;

}

bool node::DATA::operator <= (const DATA r)

{

if (this->licenseNum < r.licenseNum ||

this->licenseNum == r.licenseNum)

return true;

return false;

}

bool node::DATA::operator == (const DATA r)

{

return (this->licenseNum == r.licenseNum) ? true : false;

}

bool node::DATA::operator != (const DATA r)

{

return (this->licenseNum != r.licenseNum) ? true : false;

}

void node::DATA::clear()

{

this->licenseNum.clear();

this->fullName.clear();

this->passportID.clear();

this->adress.clear();

}

ClientTree.h

#pragma once

#include "node.h"

#include "RentalList.h"

/\*Элементы сортируются по номеру водительского удостоверения (licenseNum)\*/

class ClientTree

{

//node\* root;

/\*повороты дерева\*/

node\* srl(node\*\*);

node\* brl(node\*\*);

node\* srr(node\*\*);

node\* brr(node\*\*);

/\*node\* getNewNode(node::DATA val,

node\* l = nullptr,

node\* r = nullptr);\*/

node\* balance(node\*);

int get\_height(node\*);

void height\_upd(node\*);

int bFactor(node\*);

node\* find\_min(node\*&);

node\* remove\_min(node\*&);

/\*Работа с пользователем для ввода информации о новом клиенте\*/

node::DATA createForm(bool\*);

public:

node\* root; //"корень дерева"

RentalList\* listOfRent; //переменна для просмотра списка аренды

ClientTree(RentalList\*);

~ClientTree();

node\* findBy\_licNum(node\*, const string, int\* err);

node\* findBy\_passportID(node\*, const string, int\* err);

void findBy\_SN(node\*, const string, list<node>\*);

node\* remove(node\*, const string, bool&);

bool isCorrect\_licenseNum(string);

bool isCorrect\_passportID(string, int\*);

void display\_tree(node\* R, int l = 0);

/\*Вывод клиента, при поиске по водительскому удостоверению\*/

void display\_elem(node R);

RentalListNode\* display\_elem\_wCar(node R);

/\*вывод ВТОРОЙ ЧАСТИ ДЛЯ ПОИСКА В ТАБЛИЦЕ\*/

void display\_treeScnd(RentalListNode\*);

void display\_set(list<node>);

bool isEmpty();

node\* add(node\*& r, node::DATA n);

void \_dltTree(node\*&);

node\* \_get\_root() { return root; };

void add\_ui();

void display\_ui();

void del\_ui();

void clear\_ui();

};

ClientTree.cpp

#include "ClientTree.h"

ClientTree::ClientTree(RentalList\* rl)

{

listOfRent = rl;

root = nullptr;

}

/\*Рекурсивное удаление дерева\*/

void ClientTree::\_dltTree(node\*& p)

{

if (p)

{

\_dltTree(p->left);

\_dltTree(p->right);

delete p;

p = nullptr;

}

}

ClientTree::~ClientTree()

{

if (root)

{

\_dltTree(root);

}

}

/\*Все повороты называются по направлению поворота

левый- против часовой стрелки, правый- по часовой стрелке\*/

/\*Малое левое вращение\*/

node\* ClientTree::srl(node\*\* curRoot)

{

node\* newRoot = (\*curRoot)->right;

(\*curRoot)->right = newRoot->left;

newRoot->left = (\*curRoot);

height\_upd(\*curRoot);

height\_upd(newRoot);

return newRoot;

}

/\*Малое правое вращение\*/

node\* ClientTree::srr(node\*\* curRoot)

{

node\* newRoot = (\*curRoot)->left;

(\*curRoot)->left = newRoot->right;

newRoot->right = (\*curRoot);

height\_upd(\*curRoot);

height\_upd(newRoot);

return newRoot;

}

/\*Большое левое вращение\*/

node\* ClientTree::brl(node\*\* curRoot)

{

node\* newRoot = nullptr;

(\*curRoot)->right = srr(&(\*curRoot)->right);

newRoot = srl(curRoot);

return newRoot;

}

/\*Большое правое вращение\*/

node\* ClientTree::brr(node\*\* curRoot)

{

node\* newRoot = nullptr;

(\*curRoot)->left = srl(&(\*curRoot)->left);

newRoot = srr(curRoot);

return newRoot;

}

/\*Функция, выполняющая необходимый поворот, если это необходимо

Возвращает указатель на новый элемент, находящийся на месте "старого"\*/

node\* ClientTree::balance(node\* p)

{

height\_upd(p);

switch (bFactor(p))

{

/\*Перевес влево - правый поворот\*/

case(-2):

/\*Левове поддерево больше- большой поворот,

правое большое- большой\*/

return (bFactor(p->left) > 0) ? brr(&p) : srr(&p);

/\*Перевес вправо - ллевый поворот\*/

case(2):

/\*правое поддерево больше- большой поворот,

левое большое- большой\*/

return (bFactor(p->right) < 0) ? brl(&p) : srl(&p);

/\*Балансировка не нужна\*/

default:

return p;

}

}

///\*Выделение места под новый узел дерева с заполнением его данными\*/

//node\* ClientTree::getNewNode(node::DATA value, node\* l, node\* r)

//{

// node\* newnode = new node(value, l, r);

//

// /\*произощла ошибка выделения памяти\*/

// if (!newnode)

// {

// std::cerr << "ошибка выделения памяти!" << endl;

// exit(1);

// }

// else

// {

// return newnode;

// }

//}

/\*АВл вставка- алгоритм обхода дерева со вставкой элемента и

конетролем балансировки\*/

node\* ClientTree::add(node\*& curRoot, node::DATA newNode)

{

/\*используется стандартный алгоритм поиска для АВЛ дерева, а не

заданный по варианту, т.к. тогда бы утерялся смысл самого АВЛ-дереева.

Заданный тип обхода дерева используется в другой функции.

В левое поддерево записываются элементы < рассматриваемого

В правое поддерево записываются элементы > рассматриваемого.\*/

/\*curRoot-узел, с которым идёт сравнение в данный момент\*/

//node\* tempP(nullptr);

/\*Нашли место для вставки\*/

if (!curRoot)

{

/\*Инициализируем новый элемент\*/

node\* root = new node(newNode);

curRoot = root;

curRoot->height = 1;

return curRoot;

}

/\*Спускаемся ниже\*/

/\*спуск в левое поддерево\*/

if (newNode < curRoot->data)

add(curRoot->left, newNode);

else

/\*Спуск в правое поддерево\*/

if (newNode > curRoot->data)

add(curRoot->right, newNode);

else

/\*Такой элемент уже был в дереве- ничего не вставляем\*/

return curRoot;

/\*Корректируем узел, при проведении балнсировки\*/

curRoot = balance(curRoot);

return curRoot;

}

///\*рекурсивно вычисляет высоту, на которой находится корень\*/

//int ClientTree::Height(node\* root)

//{

// /\*Обратный обход дерева\*/

// if (!root)

// return 0;

// int hLeft = Height(root->left); //рассчитываем высоту левого поддерева

// int hRight = Height(root->right); //рассчитываем высоту правого поддерева

//

// /\*Возваращаем высоту наибольшего поддерева + 1 за счёт рассматриваемого узла\*/

// return (hLeft > hRight) ? hLeft + 1 : hRight + 1;

//}

/\*Возвращает высоту дерева\*/

int ClientTree::get\_height(node\* p)

{

/\*Обёртка, чтобы работать даже с нулевыми указателями\*/

return p ? p->height : 0;

}

/\*Расскчитывает и возвращает баланс фактор\*/

int ClientTree::bFactor(node\* p)

{

/\*Работает при корректных высотах потомков\*/

return get\_height(p->right) - get\_height(p->left);

}

/\*Обновляет высоту узла\*/

void ClientTree::height\_upd(node\* p)

{

/\*работает с корректными высотами потомков\*/

if (p)

{

int h\_1 = get\_height(p->left);

int h\_2 = get\_height(p->right);

p->height = (h\_1 > h\_2) ? h\_1 + 1 : h\_2 + 1;

}

}

/\*поиск минимального элемента, начиная с переданного узла\*/

node\* ClientTree::find\_min(node\*& curNode)

{

return curNode->left ? find\_min(curNode->left) : curNode;

}

/\*Изымание минимального элемента из дерева без его удаления из памяти\*/

node\* ClientTree::remove\_min(node\*& value)

{

/\*У минимального элемента из потомков может быть только 1 правый или не быть совсем\*/

/\*Нашли минимальный\*/

if (!value->left)

return value->right;

value->left = remove\_min(value->left);

return balance(value);

}

/\*Удаление элемента из дерева\*/

node\* ClientTree::remove(node\* R, const string val, bool& del) {

if (!R) return 0;

if (val < R->data.licenseNum)

R->left = remove(R->left, val, del);

else if (val > R->data.licenseNum)

R->right = remove(R->right, val, del);

else {

node\* l = R->left;

node\* r = R->right;

delete R;

R = nullptr;

del = true;

if (r == NULL) return l;

node\* min = find\_min(r);

min->right = remove\_min(r);

min->left = l;

return balance(min);

}

return balance(R);

}

//node\* ClientTree::remove(node\* root, node::DATA value)

//{

// if (!root)

// return nullptr;

// /\*Рекурсивный поиск\*/

// /\*В лево поддерево\*/

// if (value < root->data)

// root->left = remove(root->left, value);

// /\*В правое поддерево\*/

// else if (value > root->data)

// root->right = remove(root->right, value);

// else //Нашли элемент

// {

// /\*Сохраняем потомков\*/

// node\* l = root->left;

// node\* r = root->right;

// delete root;

//

// if (!r)

// return l;

// node\* min = find\_min(r);

// min->right = remove\_min(r);

// min->left = l;

//

// return balance(min);

// }

// /\*Выполняем баллансировку узла перед выходом из рекурсии\*/

// return balance(root);

//

//}

/\*поиск по водительскому удостовеению\*/

node\* ClientTree::findBy\_licNum(node\* curRoot, const string value, int\* err)

{

node\* res(nullptr), \* temp(nullptr);

/\*err- порядковый номер ошибки\*/

if (!this->root)

{

\*err = 1002;//не нашлось узла

return nullptr;

}

if (!isCorrect\_licenseNum(value))

{

\*err = 1003; //Некорректна строка для поиска

return nullptr;

}

if (curRoot)

{

if (value < curRoot->data.licenseNum)

{

temp = findBy\_licNum(curRoot->left, value, err);

if (temp)

res = temp;

}

if (value > curRoot->data.licenseNum)

{

temp = findBy\_licNum(curRoot->right, value, err);

if (temp)

res = temp;

}

}

if (curRoot)

{

/\*Если нашли подходящий узел- возвращем его и сохраняем в переменную\*/

if (value == curRoot->data.licenseNum)

return curRoot;

}

/\*продолжаем перемещать найденый узел через всю глубину рекурсии\*/

return res;

}

/\*Поиск по номеру паспорта\*/

node\* ClientTree::findBy\_passportID(node\* curRoot, const string value, int\* err)

{

node\* res(nullptr), \* temp(nullptr);

if (curRoot)

{

temp = findBy\_passportID(curRoot->left, value, err);

if (temp)

res = temp;

temp = findBy\_passportID(curRoot->right, value, err);

if (temp)

res = temp;

/\*Если нашли подходящий узел- возвращем его и сохраняем в переменную\*/

if (value == curRoot->data.passportID)

return curRoot;

/\*продолжаем перемещать найденый узел через всю глубину рекурсии\*/

}

return res;

}

/\*Поиск по ФИО\*/

void ClientTree::findBy\_SN(node\* curRoot, const string value, list<node>\* res)

{

if (curRoot)

{

findBy\_SN(curRoot->left, value, res);

findBy\_SN(curRoot->right, value, res);

if (findInText(curRoot->data.fullName, value) != -1)

res->push\_back(\*curRoot);

}

}

/\*Проверка на правильность записи номера водительского удостоверения\*/

bool ClientTree::isCorrect\_licenseNum(string str)

{/\*фромат записи licenseNum: RR AA NNNNNN\*/

/\*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).\*/

/\*Разрешённые символы\*/

const string allowedCheracters = "АВЕКМНОРСТУХ-";

if (str.length() != 10)

return false;

if (!isdigit(str[0]) || !isdigit(str[1]))

return false;

if (int(allowedCheracters.find(str[2])) == -1

|| int(allowedCheracters.find(str[3])) == -1)

return false;

/\*проверка наличия только разрешённых символов\*/

for (int i(4); i < str.length(); i++)

if (!isdigit(str[i]))

return false;

return true;

}

/\*Проверка на правильность номера паспорта\*/

bool ClientTree::isCorrect\_passportID(string num, int\* err)

{/\*10 цифр без пробелов\*/

if (num.length() != 10)

{

\*err = 1312;

return false;

}

for (int i(0); i < num.length(); i++)

{

if (!isdigit(num[i]))

{

\*err = 1312;

return false;

}

}

return true;

}

/\*Вывод дерева на экран\*/

void ClientTree::display\_tree(node\* R, int l) {

if (R)

{

display\_tree(R->right, l + 11);

for (int i = 1; i <= l; i++) cout << " ";

cout << R->data.licenseNum << endl; //выводит номера водительского удостоверения

display\_tree(R->left, l + 11);

}

}

void ClientTree::display\_ui()

{

if(root) display\_tree(this->root);

else cout << errorName(1001) << endl;

}

/\*Вывод клиента, при поиске по водительскому удостоверению.\*/

RentalListNode\* ClientTree::display\_elem\_wCar(node elem)

{

RentalListNode\* foundElem(nullptr);

string regNum(""); //номер авто.

cout << setfill('-') << setw(120) << '-' << setfill(' ') << endl;

cout << "Номер воительского удостоверения:\t" << elem.data.licenseNum << endl

<< "ФИО:\t\t\t\t" << elem.data.fullName << endl

<< "Адрес проживания:\t\t" << elem.data.adress<< endl;

foundElem = listOfRent->findBy\_licNum(elem.data.licenseNum);

return foundElem;

}

/\*вывод ВТОРОЙ ЧАСТИ ДЛЯ ПОИСКА В ТАБЛИЦЕ\*/

void ClientTree::display\_treeScnd(RentalListNode\* l)

{

node\* client(nullptr);

int err;

if (l)

{

/\*ищем арендателя в дереве\*/

client = findBy\_licNum(root, l->clientLicenseNum, &err);

/\*его нет (а аренда, связанная с ним есть => ошибка)\*/

if (!client)

{

cout << errorName(1007) << endl;

exit(1);

}

cout << "Арендатель:" << endl

<< "\tФИО:\t\t" << client->data.fullName << endl

<< "\tНомер водительского удостоверения:\t" << client->data.licenseNum << endl;

}

else

cout << errorName(1109) << endl;

cout << setfill('-') << setw(120) << '-' << setfill(' ') << endl;

}

/\*Вывод клиента, при поиске по марке.\*/

void ClientTree::display\_elem(node elem)

{

cout << setfill('-') << setw(120) << '-' << setfill(' ') << endl;

cout << "Номер воительского удостоверения:\t" << elem.data.licenseNum << endl

<< "ФИО:\t\t\t\t\t" << elem.data.fullName << endl

<< "Адрес проживания:\t\t\t" << elem.data.adress << endl;

cout << setfill('-') << setw(120) << '-' << setfill(' ') << endl;

}

/\*Вывод списка клиентов, при поиске по ФИО.\*/

void ClientTree::display\_set(list<node> set)

{

list<node>::iterator it;

if (!set.empty())

{

for (it = set.begin(); it != set.end(); it++)

{

display\_elem(\*it);

cout << endl;

}

}

}

/\*Есть ли элементы в дереве\*/

bool ClientTree::isEmpty()

{

return (root) ? false : true;

}

/\*Ввод информации о новом клиенте\*/

node::DATA ClientTree::createForm(bool\* returnToMenu)

{

node::DATA person;

node\* tempP(nullptr);

int errorKey(0); // код ошибки для вывода внутри этой ф-ции.

short int choice; //номер пункта, выбранного пользователем в ситуации выбора.

lopos variants;

\*returnToMenu = false;

/\*ФИО\*/

while (!\*returnToMenu)

{

cout << askName(1001) << ' ';

getline(cin, person.fullName, '\n');

/\*Запросили выход\*/

if (person.fullName == "0")

{

\*returnToMenu = true;

cout << warningName(9901) << endl;

}

/\*Введи ФИО\*/

else

{

if (!isNameCorrect(person.fullName, &errorKey))

{

cout << errorName(errorKey) << "\nПовторите ввод!\n\n";

continue;

}

cout << endl;

}

break;

}

/\*Водительское удостоверение\*/

while (!\*returnToMenu)

{

cout << warningName(1001) << endl

<< askName(1002) << ' ';

getline(cin, person.licenseNum, '\n');

/\*Запросил выход\*/

if (person.licenseNum == "0")

{

variants.emplace\_back("Д", ", чтобы выйти в меню");

variants.emplace\_back("Н", ", чтобы продолжить ввод");

choice = chooseDoing(warningName(9903), variants);

variants.clear(); //очистка списка вариантов

if (!choice) //Решил выйти в меню.

{

\*returnToMenu = true;

cout << warningName(9901) << "\n\n";

}

else

{

\*returnToMenu = false;

cout << warningName(9902) << endl << endl;

continue;

}

}

/\*Ввели номер водит. удостов.\*/

else

{

if (!isCorrect\_licenseNum(person.licenseNum))

{

cout << errorName(1302) << "\nПовторите ввод.\n\n";

continue;

}

/\*Проверка на уникальность\*/

if (this->findBy\_licNum(this->root, person.licenseNum, &errorKey))

{

cout << errorName(1004) << "\nПовторите ввод.\n\n";

continue;

}

}

break;

}

/\*номер паспорта\*/

while (!\*returnToMenu)

{

cout << warningName(1002) << endl

<< askName(1003) << ' ';

getline(cin, person.passportID, '\n');

/\*Запросил выход\*/

if (person.passportID == "0")

{

variants.emplace\_back("Д", ", чтобы выйти в меню");

variants.emplace\_back("Н", ", чтобы продолжить ввод");

choice = chooseDoing(warningName(9903), variants);

variants.clear(); //очистка списка вариантов

if (!choice) //Решил выйти в меню.

{

\*returnToMenu = true;

cout << warningName(9901) << "\n\n";

}

else

{

\*returnToMenu = false;

cout << warningName(9902) << endl << endl;

continue;

}

}

/\*Ввели номер паспорта\*/

else

{

if (!isCorrect\_passportID(person.passportID, &errorKey))

{

cout << errorName(errorKey) << "\nПовторите ввод.\n\n";

continue;

}

else

{

/\*Проверка на уникальность\*/

if (this->findBy\_passportID(this->root, person.passportID, &errorKey))

{

cout << errorName(1005) << "\nПовторите ввод.\n\n";

continue;

}

}

}

break;

}

/\*Адрес проживания\*/

while (!\*returnToMenu)

{

cout << askName(1004) << ' ';

getline(cin, person.adress, '\n');

/\*Запросил выход\*/

if (person.adress == "0")

{

variants.emplace\_back("Д", ", чтобы выйти в меню");

variants.emplace\_back("Н", ", чтобы продолжить ввод");

choice = chooseDoing(warningName(9903), variants);

variants.clear(); //очистка списка вариантов

if (!choice) //Решил выйти в меню.

{

\*returnToMenu = true;

cout << warningName(9901) << "\n\n";

}

else

{

\*returnToMenu = false;

cout << warningName(9902) << endl << endl;

continue;

}

}

break;

}

if (\*returnToMenu)

person.clear();

return person;

}

/\*Работа с пользователем для регистрации клиента компании\*/

void ClientTree::add\_ui()

{

bool returnTMenu(false);

node::DATA newPerson = createForm(&returnTMenu);

if (!returnTMenu)

{

add(this->root, newPerson);

cout << "Регистрация клиента прошла успешно." << endl;

}

}

/\*Работа с пользователем для удаления пользователя\*/

void ClientTree::del\_ui()

{

int plug;

bool deleted(false);

if (this->root)

{

string strTDel;

cout << warningName(1001) << endl

<< askName(1002) << ' ';

getline(cin, strTDel, '\n');

/\*Запросили выход\*/

if (strTDel == "0")

cout << warningName(9901) << endl;

else

if (isCorrect\_licenseNum(strTDel))

{

if (findBy\_licNum(root, strTDel, &plug))

{

if (listOfRent->findBy\_licNum(strTDel))

{

cout << errorName(1207) << endl;

return;

}

else

{

this->root = remove(this->root, strTDel, deleted);

if (deleted)

cout << "Удаление прошло успешно." << endl;

else

cout << "Не нашлось элемента для удаления." << endl;

}

}

else

cout << "Не нашлось элемента для удаления." << endl;

}

else

cout << "Удаление не было произведено" << endl;

}

else

cout << errorName(1001) << endl;

}

void ClientTree::clear\_ui()

{

lopos variants;

int choice;

if (!this->isEmpty())

{

//variants.clear();

variants.emplace\_back("Д", "- Удалить");

variants.emplace\_back("Н", "- Отмена");

choice = chooseDoing(askName(1101), variants);

variants.clear();

if (choice)

{

cout << warningName(9901) << endl;

}

else

{

this->\_dltTree(this->root);

cout << "Все клиенты были успешно удалены.\n";

}

}

else

cout << errorName(1101) << endl;

}

TableNode.h

#pragma once

#include "Includes.h"

/\*Элемент тхеш-такблицы хранящий информацию об автомобиле.\*/

struct TableNode

{

string car\_number; // «ANNNAA-NN» где A – буква из следующего множества: А,В,Е,К,М,Н,О,Р,С,Т,У,Х; N - цифры

string brand; // Марка машины

string color; // Цвет машины

int yearOfManufacture; // год выпуска

bool status = true; //признак наличия автомобиля

bool repair = false;

bool deleted = false;

TableNode();

TableNode(string refNum,

string brand,

string color,

int yOfManafact,

bool stat= true,

bool del= false);

void clear();

};

TableNode.cpp

#include "TableNode.h"

TableNode::TableNode()

{

car\_number.clear();

brand.clear();

yearOfManufacture = -1;

color.clear();

status = false;

repair = false;

deleted = false;

}

TableNode::TableNode(string cN,

string b,

string c,

int y,

bool s,

bool d)

{

car\_number = cN;

brand = b;

color = c;

yearOfManufacture = y;

status = s;

repair = false;

deleted = d;

}

/\*Очистка полей структуры\*/

void TableNode::clear()

{

this->car\_number.clear();

this->brand.clear();

this->color.clear();

this->yearOfManufacture = -1;

this->status = false;

this->repair = false;

this->deleted = false;

}

HashTable.h

#pragma once

/\*Хеш-таблица автомобилей компании- арендодателя\*/

#include "TableNode.h"

#include "RentalList.h"

#include <cstring>

#include <iomanip>

#include <fstream>

#include <windows.h>

#undef max

/\*Организация хеш-таблицы\*/

class HashTable {

const int m = 9; //длина ключа

const int m1 = m+1; //длина ключа +1

const int tour = 6; //кол-во итераций поиска

// константа, используемая во второй хеш-функции (обязательное простое число)

const int PRIME = 7;

/\*Размер хеш таблицы\*/

int \_size;

/\*сама хеш таблица

Хрранит указатели на структуры данных авто.\*/

TableNode\* table;

int keyTo\_int(string);

int hash\_1(string);

int hash\_2(string);

void resize();

bool set\_repair(string inRN, bool rep\_stat);

public:

HashTable(RentalList\*);

~HashTable();

RentalList\* listOfRent;

/\*Очищает данные таблицы\*/

void clear();

/\*Проверяет,находятся ли элементы в таблице\*/

bool isEmpty();

/\*Ввод данных для регистрации авто\*/

TableNode createForm(bool\* retTMen);

/\*Добавлени элемента в таблицу\*/

bool add(TableNode);

/\*Работа с пользователем для регистрации авто\*/

void add\_ui(bool\* retTMen);

/\*Работа с пользователем для удалении авто с площадки\*/

void del\_ui();

/\*Удаление из таблицы\*/

bool del(string);

void clear\_ui();

/\*вывод всей таблицы\*/

void display();

/\*Вывод одного авто, при поиске по рег. номеру\*/

RentalListNode\* display(TableNode elem);

/\*Выыод элемента\*/

void display\_elem(TableNode);

/\*вывод ВТОРОЙ ЧАСТИ ДЛЯ ПОИСДЕРЕВЕ\*/

void display\_hashScnd(RentalListNode\* l);

/\*Вывод списка авто, при поиске по марке\*/

void display(list<TableNode>);

short findBy\_regNum(string);

TableNode findBy\_regNum\_elem(string);

list<TableNode>\* findBy\_brand(string);

/\*Регистрация авто в ремонт\*/

void repair\_login\_ui(RentalList\* r);

/\*Регистрация авто в ремонт\*/

void repair\_logout\_ui(RentalList\* r);

bool isStateRegNumCorrect(string);

};

HashTable.cpp

#include "HashTable.h"

HashTable::HashTable(RentalList\* rl)

{

listOfRent = rl;

\_size = 5000;

table = new TableNode [\_size];

}

HashTable::~HashTable()

{

if (table)

delete[] table;

}

/\*Перевод ключа в число\*/

int HashTable::keyTo\_int(string key) {

/\*Сумма кодов символов (символ "-" не считается)\*/

int res(0);

/\*Складываем номера символов\*/

/\*До симола "-"\*/

for (int i(0); i < 7; i++)

res += (int)key[i];

/\*после символа "-"\*/

res += (int)key[7] + (int)key[8];

return res;

}

///\*Первая хеш-функция\*/

//int HashTable::hash\_1(string key\_char) {

// long index = 0;

// for (int i = 0; i < m1; i++) {

// index += (int)key\_char[i] \* (int)key\_char[i];

// }

// index %= \_size;

// return index;

//}

/\*Первая хеш-функция\*/

int HashTable::hash\_1(string key)

{

long index = 0;

for (int i(0); i < key.length(); i++)

{

index += key[i] \* pow(PRIME, i);

}

return index;

}

/\*Вторая хеш-функция\*/

int HashTable::hash\_2(string key\_char) // вторая хеш-ф-ция

{

int key = keyTo\_int(key\_char);

return (PRIME - (key % PRIME));

}

/\*Добавить элемент в хеш-таблицу\*/

bool HashTable::add(TableNode car) {

int index1, index2, newIndex;

index1 = hash\_1(car.car\_number);

index2 = hash\_2(car.car\_number);

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

newIndex = (index1 + i \* index2) % \_size;

/\*нашли элемент\*/

if ((table[newIndex].car\_number == "\0") || (table[newIndex].deleted)) {

table[newIndex] = car;

return true;

}

}

return false;

/\*resize();

return add(key);\*/

}

/\*Проверяет строку на соответствие норме рег. номера авто.\*/

bool HashTable::isStateRegNumCorrect(string str)

{/\*ANNNAA-NN\*/

const string allowedCheracters = "АВЕКМНОРСТУХ"; //разреш. символы

const int m = 9; //длина ключа

if (str.length() != m)

return false;

/\*Проверка на буквы\*/

if (int(allowedCheracters.find(str[0])) == -1

|| int(allowedCheracters.find(str[4])) == -1

|| int(allowedCheracters.find(str[5])) == -1

|| str[6] != '-')

return false;

/\*проверка на числа\*/

for (int i(1); i < 3; i++)

if (!isdigit(str[i]))

return false;

if (!isdigit(str[7])

|| !isdigit(str[8]))

return false;

return true;

}

/\*Поиск элемента по ключу\*/

short HashTable::findBy\_regNum(string k) {

int index1, index2, newIndex;

index1 = hash\_1(k);

index2 = hash\_2(k);

/\*проходим таблицу tour раз\*/

for (unsigned long i(0); i < \_size \* tour; i++) {

newIndex = (index1 + i \* index2) % \_size;

/\*пустая клетка- элемнета нет\*/

if (table[newIndex].car\_number == "\0")

return -1;

/\*метака удаления- идём дальше\*/

if (table[newIndex].deleted)

continue;

/\*нашли элемент- возвращаем\*/

if (table[newIndex].car\_number == k)

return newIndex;

}

/\*прошли всё и не нашли\*/

return -1;

}

/\*Поиск элемента по ключу\*/

TableNode HashTable::findBy\_regNum\_elem(string k) {

TableNode emptyNode;

int index1, index2, newIndex;

index1 = hash\_1(k);

index2 = hash\_2(k);

/\*проходим таблицу tour раз\*/

for (unsigned long i(0); i < \_size \* tour; i++) {

newIndex = (index1 + i \* index2) % \_size;

/\*пустая клетка- элемнета нет\*/

if (table[newIndex].car\_number == "\0")

return emptyNode;

/\*метака удаления- идём дальше\*/

if (table[newIndex].deleted)

continue;

/\*нашли элемент- возвращаем\*/

if (table[newIndex].car\_number == k)

return table[newIndex];

}

/\*прошли всё и не нашли\*/

return emptyNode;

}

list<TableNode>\* HashTable::findBy\_brand(string k)

{

list<TableNode>\* result = new list<TableNode>;

result->clear();

/\*Проходим таблицу\*/

for (int i(0); i < \_size; i++)

if (table[i].car\_number != "\0")

if (findInText(table[i].brand, k) != -1)

result->emplace\_back(table[i]);

return result;

}

/\*Удалчение элемента из таблицы\*/

bool HashTable::del(string k) {

short index = findBy\_regNum(k);

if (index == -1)

return false;

table[index].deleted = true;

table[index].car\_number = "\0";

table[index].brand.clear();

table[index].yearOfManufacture = 0;

table[index].color.clear();

table[index].status = false;

return true;

}

/\*Вывести таблицу на экран\*/

void HashTable::display() {

if (this->isEmpty())

cout << errorName(1101) << "\n\n";

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

{

//cout << i << ": ";

bool is\_deleted = false;

if ((table[i].car\_number == "d") && (table[i].car\_number == "\0"))

is\_deleted = true;

else

is\_deleted = false;

/\*if (table[i].car\_number == "\0" || is\_deleted)

{

cout << "-\n";

}

else if (table[i].car\_number == "\0")

{

cout << "-\n";

}

else

cout << table[i].car\_number << endl;\*/

if (table[i].car\_number != "\0")

cout << i << ":\t" << table[i].car\_number << endl;

}

}

/\*Вывод элемента\*/

void HashTable::display\_elem(TableNode elem)

{

cout << setfill('-') << setw(120) << '-' << setfill(' ') << endl;

cout << "Номер:\t" << elem.car\_number << endl

<< "Марка: \t" << elem.brand << endl

<< "Цвет:\t" << elem.color << endl

<< "В ремонте: " << ((elem.repair) ? "Да" : "Нет") << endl;

cout << setfill('-') << setw(120) << '-' << setfill(' ') << endl;

}

/\*Вывод одного авто, при поиске по рег. номеру\*/

RentalListNode\* HashTable::display(TableNode elem)

{/\*Возвращает номер вод. удостов. ждя дальнейшей обработки\*/

RentalListNode\* foundElem(nullptr);

string licensNum(""); //номер водит. ужостов. арендателя

int err(0);

cout << setfill('-') << setw(120) << '-' << setfill(' ') << endl;

cout << "Номер:\t" << elem.car\_number << endl

<< "Марка: \t" << elem.brand << endl

<< "Цвет:\t" << elem.color << endl

<< "В ремонте: " << ((elem.repair) ? "Да" : "Нет") << endl;

foundElem = listOfRent->findBy\_regNum(elem.car\_number);

return foundElem;

}

/\*вывод ВТОРОЙ ЧАСТИ ДЛЯ ПОИСДЕРЕВЕ\*/

void HashTable::display\_hashScnd(RentalListNode\* l)

{

TableNode car;

if (l)

{

/\*ищем арендателя в дереве\*/

car = findBy\_regNum\_elem(l->stateRegNum);

/\*его нет\*/

if (car.car\_number == "\0")

{

cout << errorName(1007) << endl;

exit(1);

}

cout << "Аренда:" << endl

<< "\tРег. номер авто.:\t\t" << car.car\_number << endl;

}

else

cout << errorName(1008) << endl;

cout << setfill('-') << setw(120) << '-' << setfill(' ') << endl;

}

/\*Поиск по марке авто\*/

void HashTable::display(list<TableNode> lst)

{

/\*вывод шапки\*/

/\*вывод элемнетов списка в форме таблицы\*/

list<TableNode>::iterator it;

for (it = lst.begin(); it != lst.end(); it++)

{

display\_elem(\*it);

cout << endl;

}

}

/\*Очищает таблицу\*/

void HashTable::clear()

{

TableNode\* a = new TableNode;

if (table)

for (int i(0); i < \_size; i++)

if(!listOfRent->findBy\_regNum(table[i].car\_number))

table[i].clear();

}

/\*ПРоверяет, находятся ли элементы в таблице\*/

bool HashTable::isEmpty()

{

for (int i(0); i < \_size; i++)

if (table[i].car\_number != "\0")

return false;

return true;

}

/\*Увеличивает размер хэш-таблицы в 2 раза

и переносит данные в расширенную таблицу.\*/

void HashTable::resize()

{

int prevSize(\_size);

\_size \*= 2;

TableNode\* newTable = new TableNode[\_size];

//void swap(table, newTable);

/\*переносим данные из старой таблицы в новую\*/

for (int i(0); i < prevSize; i++)

{

if (newTable[i].car\_number != "\0")

add(newTable[i]);

}

}

/\*Ввод данных для регистрации авто\*/

TableNode HashTable::createForm(bool\* returnToMenu)

{

TableNode newCar;

int errorKey(0); // код ошибки для вывода внутри этой ф-ции.

short int choice; //номер пункта, выбранного пользователем в ситуации выбора.

lopos variants;

/\*Для получения текущего года\*/

time\_t seconds = time(nullptr);

tm\* timeInfo = localtime(&seconds);

unsigned int curYear(1900 + timeInfo->tm\_year);

\*returnToMenu = false;

/\*Ввод регистрационного номера.

номер- УНИКАЛЬНЫЙ\*/

while (!\*returnToMenu)

{

cout << warningName(1101) << endl

<< askName(1101) << ' ';

getline(cin, newCar.car\_number, '\n');

/\*Запросили выход\*/

if (newCar.car\_number == "0")

{

\*returnToMenu = true;

cout << warningName(9901) << endl;

}

/\*Был введён номер.\*/

else

{

if (!isStateRegNumCorrect(newCar.car\_number))

{

cout << errorName(1103) << "\nПовторите ввод!\n\n";

continue;

}

/\*Проверка на уникальность\*/

if (this->findBy\_regNum(newCar.car\_number) != -1)

{

cout << errorName(1104) << "\nПовторите ввод.\n\n";

continue;

}

cout << endl;

break;

}

}

/\*Ввод марки.\*/

while (!\*returnToMenu)

{

cout << askName(1102) << ' ';

getline(cin, newCar.brand, '\n');

/\*Запросили выход\*/

if (newCar.brand == "0")

{

/\*Варианты для выбора\*/

variants.emplace\_back("Д", ", чтобы выйти в меню");

variants.emplace\_back("Н", ", чтобы продолжить ввод");

choice = chooseDoing(warningName(9903), variants);

variants.clear(); //очистка списка вариантов

if (!choice) //Решил выйти в меню.

{

\*returnToMenu = true;

cout << warningName(9901) << "\n\n";

}

else

{

\*returnToMenu = false;

cout << warningName(9902) << endl << endl;

continue;

}

}

/\*Была введена марка.\*/

else

{

if (!isStrCorrect(newCar.brand, &errorKey))

{

cout << errorName(errorKey) << "\nПовторите ввод!\n\n";

continue;

}

cout << endl;

break;

}

}

/\*Ввод цвета.\*/

while (!\*returnToMenu)

{

cout << askName(1103) << ' ';

getline(cin, newCar.color, '\n');

/\*Запросили выход\*/

if (newCar.color == "0")

{

/\*Варианты для выбора\*/

variants.emplace\_back("Д", ", чтобы выйти в меню");

variants.emplace\_back("Н", ", чтобы продолжить ввод");

choice = chooseDoing(warningName(9903), variants);

variants.clear(); //очистка списка вариантов

if (!choice) //Решил выйти в меню.

{

\*returnToMenu = true;

cout << warningName(9901) << "\n\n";

}

else

{

\*returnToMenu = false;

cout << warningName(9902) << endl << endl;

continue;

}

}

/\*Был введён номер.\*/

else

{

if (!isStrCorrect(newCar.color, &errorKey))

{

cout << errorName(errorKey) << "\nПовторите ввод!\n\n";

continue;

}

cout << endl;

break;

}

}

/\*Ввод года выпуска.\*/

while (!\*returnToMenu)

{

newCar.yearOfManufacture = inputAndCheck\_int(askName(1104));

if (newCar.yearOfManufacture == 0)

{

/\*Варианты для выбора\*/

variants.emplace\_back("Д", ", чтобы выйти в меню");

variants.emplace\_back("Н", ", чтобы продолжить ввод");

choice = chooseDoing(warningName(9903), variants);

variants.clear(); //очистка списка вариантов

if (!choice) //Решил выйти в меню.

{

\*returnToMenu = true;

cout << warningName(9901) << "\n\n";

}

else

{

\*returnToMenu = false;

cout << warningName(9902) << endl << endl;

continue;

}

}

/\*Был введён год производства\*/

else

{

/\*Если год производства <0 или если больше текущего года.\*/

if (newCar.yearOfManufacture < 0

|| newCar.yearOfManufacture > curYear)

{

cout << errorName(1306) << "\nПовторите ввод!\n\n";

continue;

}

break;

}

}

if (\*returnToMenu)

newCar.clear();

else

{

newCar.status = true;

newCar.deleted = false;

}

return newCar;

}

/\*Рабта с пользователем для регистрации авто\*/

void HashTable::add\_ui(bool\* returnToMenu)

{

TableNode car;

\*returnToMenu = false;

car = createForm(returnToMenu);

if (!\*returnToMenu)

if (this->add(car))

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

else

cout << "База данных переполнена: не удалось добавить маршрут.\n\n";

}

/\*Работа с пользователем для удалении авто с площадки\*/

void HashTable::del\_ui()

{

lopos variants;

short choice, index;

string numTDel("\0"); //Номер авто для удаления.

if (this->isEmpty())

cout << errorName(1101) << "\n\n";

else

{

cout << warningName(1101) << endl

<< askName(1106) << ' ';

getline(cin, numTDel, '\n');

cout << endl;

if (isStateRegNumCorrect(numTDel))

{

index = this->findBy\_regNum(numTDel);

if (index != -1)

{

/\*сейчас арендует\*/

if (listOfRent->findBy\_regNum(numTDel))

{

cout << errorName(1206) << endl;

return;

}

cout << warningName(1102) << endl;

display(table[index]);

cout << endl;

variants.emplace\_back("У", ", чтобы подтвердить удаление");

variants.emplace\_back("О", ", для отмены");

choice = chooseDoing(askName(1107), variants);

if (!choice)

{

if (this->del(numTDel))

cout << "Автомобиль успешно удалён.";

else

cout << "Ошибка удаления.";

cout << endl;

}

else

{

cout << warningName(9901) << endl;

}

}

else

cout << errorName(1102) << "\n\n"; //такого авто нет в таблице

}

else

cout << errorName(1103) << endl;

}

}

/\*Работа с пользователем для очистки таблицы\*/

void HashTable::clear\_ui()

{

lopos variants;

int choice;

if (!this->isEmpty())

{

//variants.clear();

variants.emplace\_back("Д", "- Удалить");

variants.emplace\_back("Н", "- Отмена");

choice = chooseDoing(askName(1101), variants);

variants.clear();

if (choice)

{

cout << warningName(9901) << endl;

}

else

{

this->clear();

cout << "Все автомобили были успешно удалены.\n";

}

}

else

cout << errorName(1101) << endl;

}

/\*сеттер поля repair\*/

bool HashTable::set\_repair(string inRN, bool rep\_stat)

{

int index = findBy\_regNum(inRN);

if (index != -1)

{

table[index].repair = rep\_stat;

return true;

}

return false;

}

/\*отправить авто в ремонт\*/

void HashTable::repair\_login\_ui(RentalList\* r)

{

TableNode\* helpPtr(nullptr);

int index(0);

string value("");

cout << askName(1108) << ' ';

getline(cin, value, '\n');

/\*Корректное лизначение рег. номера\*/

if (isStateRegNumCorrect(value))

{

/\*такой автомобиль не зарегистрирован\*/

if ((index = findBy\_regNum(value)) == -1)

cout << errorName(1105) << endl;

else

{

/\*Авто сейчас арендуют\*/

if (r->findBy\_regNum(value))

cout << errorName(1204) << endl;

/\*авто уже в ремонте\*/

else if (this->table[index].repair)

cout << errorName(1107) << endl;

else

{

this->table[index].repair = true;

cout << "\nАвтомобиль отправлен в ремонт.\n\n";

}

}

}

else //некорректный ввод

cout << errorName(1302) << endl;

}

/\*Забрать авто из ремонта\*/

void HashTable::repair\_logout\_ui(RentalList\* r)

{

TableNode\* helpPtr(nullptr);

int index(0);

string value("");

cout << askName(1109) << ' ';

getline(cin, value, '\n');

/\*Корректное лизначение рег. номера\*/

if (isStateRegNumCorrect(value))

{

/\*такой автомобиль не зарегистрирован\*/

if ((index = findBy\_regNum(value)) == -1)

cout << errorName(1105) << endl;

else

{

/\*авто и так не в ремонте\*/

if (!this->table[index].repair)

cout << errorName(1108) << endl;

else

{

this->table[index].repair = false;

cout << "\nАвтомобиль вернулся из ремонта.\n\n";

}

}

}

else //некорректный ввод

cout << errorName(1302) << endl;

}

RentalListNode.h

#pragma once

#include "Includes.h"

/\*stateRegNum- гос. рег. ном. авто.

Формат: ANNNAA-NN», где N –цифра; A – буква из следующего множества:

А, В, Е, К, М, Н, О, Р, С, Т, У, Х;\*/

/\*Элемент списка rentalList\*/

struct RentalListNode

{

RentalListNode(string = "",

string = "",

string = "",

string = "",

RentalListNode\* = nullptr);

/\*stateRegNum совпадает с полем структуры для авто

clientLicenseNum совпадает с полем структуры клиента.\*/

string stateRegNum; //гос. рег. номер

string clientLicenseNum;

string date\_issue; //дата выдачи авто.

string date\_return; //дата возврата авто.

RentalListNode\* pNext;

RentalListNode operator = (const RentalListNode);

bool operator > (const RentalListNode);

bool operator < (const RentalListNode);

bool operator == (const RentalListNode);

bool operator != (const RentalListNode);

bool operator >= (const RentalListNode);

bool operator <= (const RentalListNode);

};

RentalListNode.cpp

#include "RentalListNode.h"

RentalListNode::RentalListNode(string inCLN,

string inSRN,

string inD\_i,

string inD\_r,

RentalListNode\* newPNext)

{/\*при создании объекта rentaList в \_Node ещё нечего загружать,

поэтому значениями по умолчанию избавляюсь от мусора.\*/

this->stateRegNum = inSRN;

this->clientLicenseNum = inCLN;

this->date\_issue = inD\_i;

this->date\_return = inD\_r;

this->pNext = newPNext;

}

RentalListNode RentalListNode::operator = (const RentalListNode rightOperand)

{

/\*RentalListNode helpOperator;

helpOperator.stateRegNum = rightOperand.stateRegNum;

helpOperator.clientLicenseNum = rightOperand.clientLicenseNum;

helpOperator.date\_issue = rightOperand.date\_issue;

helpOperator.date\_return = rightOperand.date\_return;

return helpOperator;\*/

this->stateRegNum = rightOperand.stateRegNum;

this->clientLicenseNum = rightOperand.clientLicenseNum;

this->date\_issue = rightOperand.date\_issue;

this->date\_return = rightOperand.date\_return;

return \*this;

}

bool RentalListNode::operator > (RentalListNode rightOperand)

{

/\*Посимвольное сравнение кодов\*/

for (int i(0); i < this->stateRegNum.length(); i++)

{

if (this->stateRegNum[i] > rightOperand.stateRegNum[i])

return true;

if (this->stateRegNum[i] < rightOperand.stateRegNum[i])

return false;

}

return false;

}

bool RentalListNode::operator < (RentalListNode rightOperand)

{

/\*Посимвольное сравнение кодов\*/

for (int i(0); i < this->stateRegNum.length(); i++)

{

if (this->stateRegNum[i] < rightOperand.stateRegNum[i])

return true;

if (this->stateRegNum[i] > rightOperand.stateRegNum[i])

return false;

}

return false;

}

bool RentalListNode::operator == (RentalListNode rightOperand)

{

/\*Посимвольное сравнение кодов\*/

for (int i(0); i < this->stateRegNum.length(); i++)

{

if (this->stateRegNum[i] != rightOperand.stateRegNum[i])

return false;

}

return true;

}

bool RentalListNode::operator != (RentalListNode rightOperand)

{

/\*Посимвольное сравнение кодов\*/

for (int i(0); i < this->stateRegNum.length(); i++)

{

if (this->stateRegNum[i] != rightOperand.stateRegNum[i])

return true;

}

return false;

}

bool RentalListNode::operator >= (RentalListNode rightOperand)

{

if (\*this > rightOperand ||

\*this == rightOperand)

return true;

else

return false;

}

bool RentalListNode::operator <= (RentalListNode rightOperand)

{

if (\*this < rightOperand ||

\*this == rightOperand)

return true;

else

return false;

}

RentalList.h

#include "RentalListNode.h"

RentalListNode::RentalListNode(string inCLN,

string inSRN,

string inD\_i,

string inD\_r,

RentalListNode\* newPNext)

{/\*при создании объекта rentaList в \_Node ещё нечего загружать,

поэтому значениями по умолчанию избавляюсь от мусора.\*/

this->stateRegNum = inSRN;

this->clientLicenseNum = inCLN;

this->date\_issue = inD\_i;

this->date\_return = inD\_r;

this->pNext = newPNext;

}

RentalListNode RentalListNode::operator = (const RentalListNode rightOperand)

{

/\*RentalListNode helpOperator;

helpOperator.stateRegNum = rightOperand.stateRegNum;

helpOperator.clientLicenseNum = rightOperand.clientLicenseNum;

helpOperator.date\_issue = rightOperand.date\_issue;

helpOperator.date\_return = rightOperand.date\_return;

return helpOperator;\*/

this->stateRegNum = rightOperand.stateRegNum;

this->clientLicenseNum = rightOperand.clientLicenseNum;

this->date\_issue = rightOperand.date\_issue;

this->date\_return = rightOperand.date\_return;

return \*this;

}

bool RentalListNode::operator > (RentalListNode rightOperand)

{

/\*Посимвольное сравнение кодов\*/

for (int i(0); i < this->stateRegNum.length(); i++)

{

if (this->stateRegNum[i] > rightOperand.stateRegNum[i])

return true;

if (this->stateRegNum[i] < rightOperand.stateRegNum[i])

return false;

}

return false;

}

bool RentalListNode::operator < (RentalListNode rightOperand)

{

/\*Посимвольное сравнение кодов\*/

for (int i(0); i < this->stateRegNum.length(); i++)

{

if (this->stateRegNum[i] < rightOperand.stateRegNum[i])

return true;

if (this->stateRegNum[i] > rightOperand.stateRegNum[i])

return false;

}

return false;

}

bool RentalListNode::operator == (RentalListNode rightOperand)

{

/\*Посимвольное сравнение кодов\*/

for (int i(0); i < this->stateRegNum.length(); i++)

{

if (this->stateRegNum[i] != rightOperand.stateRegNum[i])

return false;

}

return true;

}

bool RentalListNode::operator != (RentalListNode rightOperand)

{

/\*Посимвольное сравнение кодов\*/

for (int i(0); i < this->stateRegNum.length(); i++)

{

if (this->stateRegNum[i] != rightOperand.stateRegNum[i])

return true;

}

return false;

}

bool RentalListNode::operator >= (RentalListNode rightOperand)

{

if (\*this > rightOperand ||

\*this == rightOperand)

return true;

else

return false;

}

bool RentalListNode::operator <= (RentalListNode rightOperand)

{

if (\*this < rightOperand ||

\*this == rightOperand)

return true;

else

return false;

}

RentalList.cpp

#include "RentalList.h"

RentalList::RentalList()

{/\*t, ct- хэш таблица и список, с которыми работает список\*/

this->head = nullptr;

this->\_size = 0;

}

RentalList::~RentalList()

{

RentalListNode\* viewPtr(head), \* helpPtr(nullptr);

/\*разбиваем линейную связь\*/

if (head)

{

while (viewPtr->pNext != head)

viewPtr = viewPtr->pNext;

viewPtr->pNext = nullptr;

}

viewPtr = head;

while (viewPtr)

{

helpPtr = viewPtr;

viewPtr = viewPtr->pNext;

delete helpPtr;

helpPtr = nullptr;

}

}

///\*Меняет местами данные в объектах\*/

//void RentalList::swap(RentalListNode\*& p1, RentalListNode\*& p2)

//{

// RentalListNode\* tempP;

// tempP = p2;

// p2 = p1;

// p1 = tempP;

//}

void RentalList::swap(RentalListNode\* p1, RentalListNode\* p2)

{

RentalListNode hNode;

hNode = \*p1;

\*p1 = \*p2;

\*p2 = hNode;

}

///\*быстрая ортировка\*/

//void RentalList::quicksort(int\* mas, int first, int last, int\* comp, int\* perm)

//{ //first, last- индексы первого и последнего элменета сортировки.

// //comp, perm- кол-во сравнений и перестановок.

// //\*comp = \*perm = 0; //зануляю счётчики

// int mid, count; //mid-средний по значению элемент, count- вспомогательная переменная.

// int l = first, r = last;

// mid = min(max(mas[l], mas[r]), mas[(l + r) / 2]); //средний из первого, последнего и среднего по положению элемнетов.

// \*comp += 2; //т.к. для нахождения среднего значения требуется 2 сравнения элементов.

// do

// { /\*ищем элементы, подходящие для перестановки местами\*/

// // >= среднему- вправо, <= среднему- влево (= повторяется, т.к. l и r смотрят независимо друг от друга).

// while (mas[l] < mid)

// {

// l++;

// (\*comp)++; //сравнение

// }

// while (mas[r] > mid)

// {

// r--;

// (\*comp)++; //сравнение

// }

// //(\*comp)++;

// if (l <= r) //если не l и r не прошли мимо друг друга, то производим замену.

// {

// //(\*comp)++;

// count = mas[l];

// mas[l] = mas[r];

// mas[r] = count;

// (\*perm)++; //произошла перестановка.

// /\*отходим от переставленных элементов\*/

// l++;

// r--;

// }

// //else

//

// } while (l < r); //пока l и r в своих половинках (не прошли друг друга).

// /\*если не прошли весь массив.\*/

// if (first < r) //сортируем левую часть.

// {

// //(\*comp)++;

// quicksort(mas, first, r, comp, perm);

// }

// if (l < last) //сортируем правую часть.

// {

// //(\*comp)++;

// quicksort(mas, l, last, comp, perm);

// }

//}

/\*Сортировка списка методом Хоара\*/

void RentalList::sort(int first, int last)

//сортировка по возрастанию

{ //first, last- индексы первого и последнего элменета сортировки.

RentalListNode\* mid = new RentalListNode, \* hv = new RentalListNode;//средний по значению элемент, hv- вспомогательное значение

RentalListNode\* tmpMin, \* tmpMax;

int l = first, r = last; //левые и правые вспомог. индексы

/\*среднее по значению\*/

tmpMax = (\*this->operator[](l) > \*this->operator[](r)) ? this->operator[](l) : this->operator[](r);

mid = (\*tmpMax < \*this->operator[]((l + r) / 2)) ? tmpMax : this->operator[]((l + r) / 2);

do

{ /\*ищем элементы, подходящие для перестановки местами\*/

// >= среднему- вправо, <= среднему- влево (= повторяется, т.к. l и r смотрят независимо друг от друга).

while (\*this->operator[](l) < \*mid)

{

l++;

}

while (\*this->operator[](r) > \*mid)

{

r--;

}

if (l <= r) //если не l и r не прошли мимо друг друга, то производим замену.

{

///\*RentalListNode\* p\_1, \* p\_2;

//p\_1 = this->operator[](l);

//p\_2 = this->operator[](r);

//hv = this->operator[](r);\*/

//\*hv = \*this->operator[](l);

//\*this->operator[](l) = \*this->operator[](r);

//\*this->operator[](r) = \*hv;

///\*отходим от переставленных элементов\*/

swap(this->operator[](l), this->operator[](r));

l++;

r--;

}

} while (l < r); //пока l и r в своих половинках (не прошли друг друга).

/\*если не прошли весь список.\*/

if (first < r) //сортируем левую часть.

sort(first, r);

if (l < last) //сортируем правую часть.

sort(l, last);

}

/\*добавление элемента списка в начало\*/

bool RentalList::emplace\_front(string inCLN

, string inSRN

, string inD\_i

, string inD\_r)

{/\*добавление только в начало, т.к. это быстрее, при учёте, что

после каждого добавления будет производиться сортировка списка (по заданию).\*/

/\*Машина с таким номером не зарегистрирована\*/

/\*Клинент с таким вод. удостоверением не зарегистрирован.\*/

/\*добавляем первый элемент в список\*/

if (!head)

{

head = new RentalListNode(inCLN, inSRN, inD\_i, inD\_r);

head->pNext = head; //замыкаем на себя.

}

else

{

RentalListNode\* newNode = new RentalListNode(inCLN, inSRN, inD\_i, inD\_r, head);

/\*поиск последнего элемента списка для зациклилвания\*/

RentalListNode\* lastNode(head);

while (lastNode->pNext != head)

{

lastNode = lastNode->pNext;

}

lastNode->pNext = newNode;

head = newNode;

}

/\*заполняем поля данными\*/

head->clientLicenseNum = inCLN;

head->stateRegNum = inSRN;

head->date\_issue = inD\_i;

head->date\_return = inD\_r;

this->\_size++;

return true;

}

/\*удаляет элемент списка по указателю\*/

bool RentalList::remove(RentalListNode\*& elem)

{

RentalListNode\* viewPtr(nullptr),\* helpPtr(nullptr);

if (elem)

{

/\*удаляем голову\*/

if (elem == head)

{

if (head->pNext)

{

/\*перенаправляем последний элемент на второй\*/

viewPtr = head;

while (viewPtr->pNext != head)

{

viewPtr = viewPtr->pNext;

}

viewPtr->pNext = head->pNext;

delete elem;

elem = nullptr;

head = viewPtr->pNext;

}

else

{

delete head;

head = nullptr;

}

\_size--;

return true;

}

else /\*Удаляем хвостовой элемент\*/

if (elem->pNext == head)

{

viewPtr = head;

while (viewPtr->pNext != elem)

viewPtr = viewPtr->pNext;

viewPtr->pNext = elem->pNext;

delete elem;

elem = nullptr;

\_size--;

return true;

}

/\*середнинный элемент\*/

else

{

viewPtr = head;

while (viewPtr->pNext != elem)

viewPtr = viewPtr->pNext;

viewPtr->pNext = elem->pNext;

delete elem;

elem = nullptr;

\_size--;

}

}

return false;

}

/\*Вывод информации об анкете на экран.\*/

void RentalList::out()

{

for (int i(0); i < \_size; i++)

{

cout << (\*this)[i]->stateRegNum << " ";

}

cout << endl;

}

/\*Поиск по номеру водит. удостоверения\*/

RentalListNode\* RentalList::findBy\_licNum(string value)

{

if (head)

{

RentalListNode\* viewPtr = this->head;

do

{

if (value == viewPtr->clientLicenseNum)

return viewPtr;

viewPtr = viewPtr->pNext;

} while (viewPtr != this->head);

}

return nullptr;

}

/\*Поиск по номеру водит. удостоверения\*/

RentalListNode\* RentalList::findBy\_regNum(string value)

{

if (head)

{

RentalListNode\* viewPtr = this->head;

do

{

if (value == viewPtr->stateRegNum)

return viewPtr;

viewPtr = viewPtr->pNext;

} while (viewPtr != this->head);

}

return nullptr;

}

/\*Возвращает элемент списка по его порядковому номеру\*/

RentalListNode\* RentalList::operator[](int index)

{

int curIndex(0);

RentalListNode\* curNode(head);

if (curNode)

{

do

{

if (curIndex == index)

return curNode;

curNode = curNode->pNext;

curIndex++;

} while (curNode != head);

}

/\*Были пройдены все элементы => элемент по номеру не найден\*/

return nullptr;

}

/\*\*/

void RentalList::display\_elem(RentalListNode\* p)

{

cout << "\t|" << setw(15) << p->clientLicenseNum << setw(2) << " |" << setw(12) << p->stateRegNum << setw(2) << " |" << setw(15) << p->date\_issue << setw(2) << " |" << setw(15) << p->date\_return << setw(2) << " |" << endl;

cout << "\t+----------------+-------------+----------------+----------------+" << endl;

}

/\*Вывод шапки таблицы\*/

void RentalList::display\_firstLine()

{

cout << "\t+----------------+-------------+----------------+----------------+" << endl;

cout << "\t|" << setw(2) << ' ' << "Удостоверение" << setw(1) << " |" << setw(12) << "Номер авто" << setw(2) << " |" << setw(15) << "Дата выдачи" << setw(2) << " |" << setw(15) << "Дата возврата" << setw(2) << " |" << endl;

cout << "\t+----------------+-------------+----------------+----------------+" << endl;

}

/\*Вывод всего списка\*/

void RentalList::display\_all()

{

RentalListNode\* viewPtr(this->head);

if (!viewPtr)

cout << errorName(1205) << endl;

else

{

display\_firstLine();

do

{

display\_elem(viewPtr);

viewPtr = viewPtr->pNext;

} while (viewPtr != this->head);

}

}