МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное образовательное учреждение высшего образования

«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ   
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

ИНСТИТУТ НЕПРЕРЫВНОГО И ДИСТАНЦИОННОГО ОБРАЗОВАНИЯ

|  |
| --- |
| КАФЕДРА компьютерных технологий и программной инженерии |

КУРСОВАЯ РАБОТА (ПРОЕКТ)   
ЗАЩИЩЕНА С ОЦЕНКОЙ

РУКОВОДИТЕЛЬ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

|  |
| --- |
| ПОЯСНИТЕЛЬНАЯ ЗАПИСКА К КУРСОВОЙ РАБОТЕ (ПРОЕКТУ) |
| Использование заданных структур данных и алгоритмов при разработке программного обеспечения информационной системы |
| по дисциплине: структуры и алгоритмы обработки данных |

РАБОТУ ВЫПОЛНИЛ

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| СТУДЕНТ ГР. № | Z9431 | |  |  |  | Андреев Д.И. |
|  | номер группы | |  | подпись, дата |  | инициалы, фамилия |
| Студенческий билет № | | 2019/3781 | |  |  |  |

|  |  |
| --- | --- |
| Шифр ИНДО |  |

Санкт-Петербург 2020

Оглавление

[1. Задание на курсовой проект 3](#_Toc70878095)

[2. Введение 6](#_Toc70878096)

[3. Алгоритмы и структуры данных 7](#_Toc70878097)

[4. Описание программы 15](#_Toc70878098)

[5. Тестирование программы 18](#_Toc70878099)

[6. Заключение 23](#_Toc70878100)

[Список использованной литературы 24](#_Toc70878101)

[Приложение 1 25](#_Toc70878102)

[Приложение 2 40](#_Toc70878103)

1. Задание на курсовой проект

Цель курсового проекта является изучение структур данных и алгоритмов их обработки, а также получение практических навыков их использования при разработке программ. Задачей курсового проекта является разработка информационной системы для заданной предметной области с использованием заданных структур данных и алгоритмов.

Вариант курсового проекта:

* предметная область – обслуживание клиентов в бюро проката автомобилей;
* метод хеширования – закрытое хеширование с линейным опробованием;
* метод сортировки – быстрая (Хоара);
* вид списка – линейный двунаправленный;
* метод обхода дерева – обратный;
* алгоритм поиска в тексте – прямой;

Вариант задание накладывает следующие требования для информационной системы:

* информационная система должна осуществлять ввод, хранение, обработку и вывод данных о клиентах, автомобилях, принадлежащих бюро проката, выдаче на прокат и возврате автомобилей от клиентов;
* данные о каждом клиенте должны содержать номер водительского удостоверения (строка формата «RR AA NNNNNN», где RR – код региона (цифры); AA – серия (буквы из следующего множества: А, В, Е, К, М, Н, О, Р, С, Т, У, Х); NNNNNN – порядковый номер удостоверения (цифры); код, серия и номер отделяются друг от друга пробелами), ФИО (строка), паспортные данные (строка), адрес (строка);
* данные о клиентах должны быть организованны в виде АВЛ-дерева поиска, упорядоченного по «номеру водительского удостоверения»;
* данные о каждом автомобиле должны содержать государственный регистрационный номер (строка формата «ANNNAA-NN», где N –цифра; A – буква из следующего множества: А, В, Е, К, М, Н, О, Р, С, Т, У, Х), марку (строка), цвет (строка), год выпуска (целое), признак наличия (логическое);
* данные об автомобилях должны быть организованны в виде хеш-таблицы, первичным ключом которой является «Государственный регистрационный номер»
* данные о выдаче на прокат или возврате автомобилей от клиентов должны содержать, номер водительского удостоверения (строка, формат которой соответствует аналогичной строке в данных о клиентах), государственный регистрационный номер (строка, формат которой соответствует аналогичной строке в данных об автомобилях), дату выдачи (строка), дату возврата (строка).
* данные о выдаче на прокат или возврате автомобилей от клиентов должны быть организованны в виде списка, который упорядочен по первичному ключу – «Государственный регистрационный номер».
* информационная система «Обслуживание клиентов в бюро проката автомобилей» должна осуществлять следующие операции:
  1. регистрацию нового клиента;
  2. снятие с обслуживания клиента;
  3. просмотр всех зарегистрированных клиентов;
  4. очистку данных о клиентах;
  5. поиск клиента по «номер водительского удостоверения» (результаты поиска – все сведения о найденном клиенте и государственный регистрационный номер автомобиля, который ему выдан);
  6. поиск клиента по фрагментам ФИО или адреса (результаты поиска – список найденных клиентов с указанием номера водительского удостоверения, ФИО и адреса);
  7. добавление нового автомобиля;
  8. удаление сведений об автомобиле;
  9. просмотр всех имеющихся автомобилей;
  10. очистку данных об автомобилях;
  11. поиск автомобиля по «Государственному регистрационному номеру». (результаты поиска – все сведения о найденном автомобиле, а также ФИО и номер водительского удостоверения клиента, которому выдан этот автомобиль);
  12. поиск автомобиля по названию марки автомобиля. (результаты поиска – список найденных автомобилей с указанием «Государственный регистрационный номер», марки, цвета, года выпуска);
  13. регистрацию отправки автомобиля в ремонт;
  14. регистрацию прибытия автомобиля из ремонта;
  15. регистрацию выдачи клиенту автомобиля на прокат;
  16. регистрацию возврата автомобиля от клиентов.
* состав данных о клиенте или автомобиле, выдаваемых при просмотре всех зарегистрированных клиентов или просмотре всех автомобилей, принадлежащих бюро проката должен содержать не менее двух полей.
* поиск клиента по фрагментам ФИО или адреса должен осуществляться путем систематического обхода АВЛ-дерева поиска. При поиске клиента по фрагментам ФИО или адреса могут быть заданы как полное ФИО или адрес, так и их части. Для обнаружения заданного фрагмента в полном ФИО или адресе должен применяться алгоритм поиска слова в тексте.
* регистрация отправки автомобиля на ремонт должна осуществляться только при наличии этого автомобиля (значение поля «Признак наличия» для соответствующего автомобиля имеет значение «Истина»). При этом значение поля «Признак наличия» для соответствующего автомобиля изменяется на значение «Ложь».
* при регистрации прибытия автомобиля из ремонта значение поля «Признак наличия» для соответствующего автомобиля изменяется на значение «Истина».
* регистрация выдачи автомобиля клиенту должна осуществляться только при наличии свободного выдаваемого автомобили (значение поля «Признак наличия» для соответствующего автомобиля имеет значение «Истина»).
* при регистрации выдачи автомобиля клиенту или возврата автомобиля от клиента должно корректироваться значение поля «Признак наличия» для соответствующего автомобиля.
* при снятии с обслуживания клиента должны быть учтены и обработаны ситуации, когда у клиента имеется выданный автомобиль. Аналогичным образом следует поступать и с удалением сведений об автомобилях.

1. Введение

Любая сфера деятельности человека имеет потребность в обработке больших объемов данных. Так как человек не совершенен и при обработке больших объемов информации возникают ошибки, которые могут привести к серьезным последствиям. В следствии этого такие задачи как поставленная курсовым проектом должна быть решена с помощью переноса обработки данных на машину.

Данная задача заключается в структурировании данных и создании стректур для их хранения, а также алгоритмов и правил для доступа к ним, а также их добавления, модификации, дополнения и удаления. Так как человеку для успешного использования таких данных необходимо их понятное представление в задачу так же входит разработка модели представления данных.

1. Алгоритмы и структуры данных

В ходе выполнения курсового проекта были разработаны следующие структуры данных (исходный код приведен в приложении 1):

* Линейный двусвязный список
* Хеш-таблица
* Дерево поиска

Структуры данных были реализованы в виде шаблонных классов для возможности использования их с различными данными, при этом ключом в хеш-таблице и дереве поиска является строка в стиле С, а длина ключа задается как параметр шаблона.

**Описание линейного двусвязного списка**

Данные в линейном двусвязным списке хранятся в шаблонном классе node, который является одним звеном списка. Данный класс хранит указатель на следующий и предыдущий элемент в списке, а также непосредственно данные. Также для линейного двусвязного списка был реализован паттерн итератор для упрощения обхода списка при реализации поставленных задач, а также для разработки универсальных шаблонных функций алгоритмов, определенных в задании курсового проекта. Так как структура данных линейный двусвязный список не имеет произвольного доступа к элементам (в следствии того, что список организован, как последовательность элементов, содержащих указатели на предыдущий и следующий элементы), операции сравнения (operator<, operator<=, operator> и operator>=) представляют собой системный обход списка. Псевдокод для операции operator< представлен ниже (для остальных операторов псевдокод отличается лишь направлением обхода и операторами сравнения):

Входные данные: указатель на данный элемент и на элемент, с которым данный сравнивается.

Если элементы равны

вернуть ЛОЖЬ и завершить выполнение

Создать временный итератор iter, указывающий на данный элемент

Пока итератор iter указывает на существующую память

Если iter равен элементу с которым сравнивается данный

Вернуть ИСТИНА и завершить выполнение

Передвинуть итератор на следующий элемент

Вернуть ЛОЖЬ и завершить выполнение

Как можно увидеть из приведенного выше псевдокода, данный алгоритм имеет временную сложность в худшем варианте, когда данный элемент в начале списка, а сравниваемый элемент в конце – O(n), в лучшем, когда сравниваемый элемент стоит следующим за данным – O(1). Схожим образом построены алгоритмы операторов + и - итератора, а также функции at(int i) и оператора [] линейного двусвязного списка, которые позволяют получить произвольный доступ к элементам списка при реализации алгоритма сортировки, за исключением того, что в данных алгоритмах отсутствуют операции сравнения.

Сам список имеет следующие возможности:

* создание нового списка из уже существующего (при этом последовательно копируются все значения и указатели существующего списка) – временная сложность всегда O(n);
* поиск прямой поиск элемента в списке (функция contains(const T& val)) – временная сложность в лучшем случае, когда искомый элемент первый в списке O(1), в худшем, когда элемента в списке нет – O(n);
* добавление элемента в начало списка (функция append(const T& val)) – временная сложность всегда О(1), так как для вставки нужно только переустановить указатель начала списка на вставленный элемент
* добавление элемента в конец списка
* добавление элемента в произвольное место
* удаление элемента
* очистка списка
* обмен двух элементов (функция swap(iterator left, iterator right)) – функция обменивает местами в списке два элемента на которые указывают итераторы left и right, при этом для самих элементов помять заново не распределяется, изменяются только значения указателей на предыдущий и следующий элемент, элементов, соседних с обмениваемыми. Временная сложность всегда O(1).

Псевдокод операции обмена элементов:

Входные данные: указатели на обмениваемые элементы left и right

Если указатель left – первый элемент в списке

Присвоить указателю на начало списка указатель right

Если элемент right следует в списке за элементом left

Создать временные указатели rTemp и lTemp и присвоить им значения указателя, следующего за right и указателя за которым следует left соответственно

Установить указатели next элементов right left на указатели left и rTemp соответственно

Установить указатели previous элементов left b right на right и lTemp соответственно

Установить указатель previous элемента rTemp на left

Если lTemp существует (элемент left не первый в списке)

Установить его указатель на элемент right

Завершить выполнение

Иначе

Создать временные указатели: llTemp, равный указателю previous элемента left; rrTemp, равный указателю на элемент next элемента right; lrTemp, равный указателю next элемента left; rlTemp, равный указателю previous элемента right

Присвоить указателям next элементов right и left указатели lrTemp и rrTemp соответственно

Присвоить указателям previous элементов right и left указатели llTemp и rlTemp соответственно

Если llTemp существует (элемент left не первый в списке

Присвоить ему значение right

Присвоисть указателям rrTemp, lrTemp, rlTemp соответственно значения указателей previous элементов left и right и указателя next элемента left.

Для реализации поддержания списка в отсортированном виде была реализована шаблонная функция sort, которая вызывается каждый раз после добавления элемента в список. Исходный код функции sort приведен ниже.

template<typename Container, typename Iter, typename Pred>

void sort(Container &cont, Iter begin, Iter end, Pred pred)

{

if(!begin.isValid() || !end.isValid() || !(begin < end))

return;

Iter temp;

Iter left = begin;

Iter right = end;

Iter pivod = begin + (end - begin) / 2;

while (left.isValid() && left < right) {

while(pred(\*left, \*pivod) < 0)

++left;

while(pred(\*pivod, \*right) < 0)

--right;

if(left < right && pred(\*left, \*right) != 0) {

bool swapPivod = left == pivod;

bool swapBegin = left == begin;

bool swapEnd = right == end;

cont.swap(left, right);

temp = left;

left = right;

right = temp;

if(swapPivod)

pivod = left;

if(swapBegin)

begin = left;

if(swapEnd)

end = right;

}

if(pred(\*left, \*right) == 0 && left != right)

++left;

}

if(begin < end) {

sort(cont, begin, pivod, pred);

sort(cont, ++pivod, end, pred);

}

}

Псевдокод алгоритма сортировки:

Входные данные: контейнер cont на котором производится сортировка, указатель на начало промежутка сортировки begin, указатель на конец промежутка сортировки end,

Если end < begin

Завершить выполнение и выйти

Создаем временный указатель temp для дальнейшего обмена указателей через него

Сохраняем указатели на начало и конец промежутка сортировки

Находим указатель pivod, указывающий на элемент относительно которого контейнер будет разбит на подмножества

Пока left < right

Находим слева от pivod элемент больший элемента на который указывает pivod

Находим справа от pivod элемент меньший элемента на который указывает pivod

Если элементы, на которые указывают right и left не равны

Обмениваем элементы, на которые указывают left и right

Обмениваем итераторы left и right для сохранения порядка указателей

Если left был равным pivod до обмена элементов

Делаем pivod равным left

Если left был равным begin до обмена элементов

Делаем begin равным left

Если right был равным end до обмена элементов

Делаем end равным right

Если элементы, на которые указывают left и right равны, а указатели не равны

Увеличиваем left на 1

Если begin меньше end

Рекурсивно вызываем sort с входными данными: cont, begin, pivod, pred

Рекурсивно вызываем sort с входными данными: cont, pivod + 1, end, pred

Как можно увидеть из приведенного кода точка относительно которой происходит разбиение контейнера на подмножества всегда находится в центре контейнера, так как контейнер (в данном случае список) всегда находится в отсортированном состоянии и поэтому средний элемент находится либо в центре контейнера, либо является только что вставленным. Так же для исключения зацикливания алгоритма при равных элементах, на которые указывают указатели left и right, если они равны элемент left сдвигается вправо при этом происходит проверка, что указатели действительно указывают на разные элементы в памяти.

**Описание хеш-таблицы.**

Внутренне хеш-таблица (приложение 1) хранит данные в выделенном в куче массиве. При этом сами данные представлены закрытой структурой хеш-таблицы Data. Структура Data содержит в себе ключ, значение и два флага: empty и deleted, которые разрешают запись элемента по данному адресу. Флаг empty позволяет таблице определить, что данный адрес еще не использовался, а флаг deleted, что данный адрес был очищен. Данные флаги необходимы при правильно добавлении элементов, так как память для таблицы выделяется при ее создании и не удаляется до момента удаления хеш-таблицы и поэтому для организации записи необходимо понимать заполнен ли данный адрес или нет.

Сама хеш-таблица обладает следующими возможностями:

* вставка элемента val с ключом key (строка в стиле C) – временная сложность всегда O(1) так ка таблица индексируется хеш-функцией. Псевдокод функции вставки элементов приведен ниже. Хеш-функция хеш-таблицы основана на кодах символов UNICODE. Данная функция преобразует массив символов в число типа int при помощи объединения (union), и увеличивает его в keyLen раз (размер ключа в байтах). Далее полученный сегмент приводится к минимальной границе, умножается на коэффициент multiCoef и приводится к размеру выделенной памяти таблицы. При изменении размера таблицы создается временный массив равный новому размеру, все элементы текущего массива копируются в новый. Затем память, выделенная под текущий массив, освобождается и распределяется вновь для массива с новым размером. После этого, для сохранения индексирования, для каждого элемента временного массива вызывается функция вставки этого элемента, если он не помечен как deleted и empty.

Входные данные: ключ key значение val

Если фактический размер таблицы превышает выделенный размер более чем на 80%

Расширить таблицу

Создать переменную seg и записать в нее значение хеш-функции сегмента для key

Пока seg меньше выделенного размера таблицы

Если элемент по адресу seg удален или пуст

Сохраняем значение val по данному адресу

Сохраняем значение key по данному адресу

Устанавливаем флаги delete и empty в false

Увеличиваем фактический размер на 1

Завершаем выполнение и выходим

Иначе если ключ по адресу seg равен key

Перезаписываем значение по адресу seg на val

Завершаем выполнение и выходим

Иначе

Вызываем функцию линейного опробования

Увеличиваем размер таблицы

Рекурсивно вызываем функцию вставки с входными параметрами: key, val

* удаление элемента с ключом key – временная сложность всекда O(1). Псевдокод представлен ниже.

Входные данные: ключ key

Создать переменную seg и записать в нее значение хеш-функци сегмента для key

Пока seg меньше выделенного размера таблицы

Если ключ по адресу seg равен key

Устанавливаем флаг deleted в true

Уменьшаем фактический размер на 1

Завершаем выполнение и выходим

Иначе

Вызываем функцию линейного опробования

* очистка таблицы – временная сложность всегда O(1), так как при очистке память, выделенная под массив данных таблицы освобождается и резервируется заново.
* операторы доступа к константным и не константным значениям – временная сложность всегда O(1). Так как функции доступа возвращают значение по ссылке, обращение к хеш-таблице по несуществующему ключу, возвращает значени первого элемента таблицы. Псевдокод представлен ниже.

Входные данные: ключ key

Создать переменную seg и записать в нее значение хеш-функци сегмента для key

Пока seg меньше выделенного размера таблицы

Если ключ по адресу seg равен key

Вернуть значение по адресу key

Иначе

Вызвать функцию линейного опробования

Вернуть первый элемент таблицы

* возврат ключа по значению - временная сложность O(m) в худшем случае, если элемента нет в таблице и O(1) в лучшем, когда элемент находится в начале таблицы, (m – максимальное количество элементов в таблице в данный момент). Данная функция обходит массив элементов таблицы и, при совпадении значений, возвращает найденный ключ. Если ключ не найден, возвращает символ начала строки;
* поиск ключа в таблице – временная сложность такая же, как и для функции возврата ключа по значению. Данная функция обходит массив элементов таблицы и, если ключ найден останавливает работу и возвращает true;
* возврат списка ключей и значений таблицы – временная сложность всегда O(m), где m – максимальное количество элементов в таблице в данный момент. Данные функции обходят все элементы таблицы и, если не установлены флаги deleted и empty, сохраняют ключи или значения в список.

**Описание дерева поиска**

Каждый элемент дерева поиска хранится в структуре Node, которая представляет собой узел дерева. Данная структура содержит сами данные, ключ с ними ассоциированный, высота в дереве на которой находится данный узел, правого и левого потомка.

Так как высота дерева не превышает log2((n+1)-1), n – количество элементов дерева, алгоритм обхода выполнен в рекурсивном виде, и, как следствие структура содержит методы для обратного обхода дерева. Так как для представлений данных необходимы все значения дерева, а также задание указывает выполнять поиск по элементу в древе (должна быть реализована фильтрация) алгоритм обхода дерева представлен в виде двух перегруженных функций: первая принимает на вход ссылку на список и заполняет его всеми значениями дерева, производя обратный обход, другая принимает на вход помимо ссылки на список значение для поиска и функцию – предикат для сравнения значений и возвращает список значений, при сравнении которых с переданным предикат вернул true. Псевдокод обхода для получения всех значений дерева представлен ниже.

Входные данные: линейный двунаправленный список storage

Если правый потомок существует

Вызвать его функцию обхода с параметром storage

Если левый потомок существует

Вызвать его функцию обхода с параметром storage

Иначе

Сохранить значение данного узла в storage

Структура Node также содержит методы для выполнения балансировки дерева и поворота узлов. Балансировка производится каждый раз после вставки элементов в дерево.

Вставка происходит рекурсивно: сначала переданный ключ сравнивается с дочерними, далее происходит рекурсивный вызов функции вставки для выбранного дочернего элемента, если он существует, если же дочернего элемента не существует, происходит вставка элемента и балансировка дерева.

Также структура Node производит удаление узла. Как и при вставке, после удаления узла происходит балансировка дерева.

Сама структура бинарного дерева реализует следующий функции:

* вставка и удаление элементов – временная сложность в среднем O(log2(n)) в худшем O(n). Производится рекурсивным вызовом соответствующих функций узла корня дерева;
* константные и не константные функции доступа к элементам – временная сложность такая же, как и у функции вставки и удаления. Производится не рекурсивным спуском по дереву от корневого узла;
* получение всех значений дерева – временная сложность всегда О(n), так как нужно обойти все элементы дерева. Выполняется алгоритмом обратного обхода дерева;
* функция поиска – временная сложность и алгоритм работы такой же как и в предыдущем пункте;
* функция очистки дерева – выполняется за O(n), так как необходимо удалить каждый из узлов;
* функция поиска ключа – временная сложность такая же, как и у функций удаления и вставке. Работает также, как и функции доступа к элементам.

1. Описание программы

Для решения задачи курсового проекта было создано консольное приложение с псевдографикой. За основу взята архитектурная модель MVVM. Диаграмма классов приложения представлена на рис. 1.

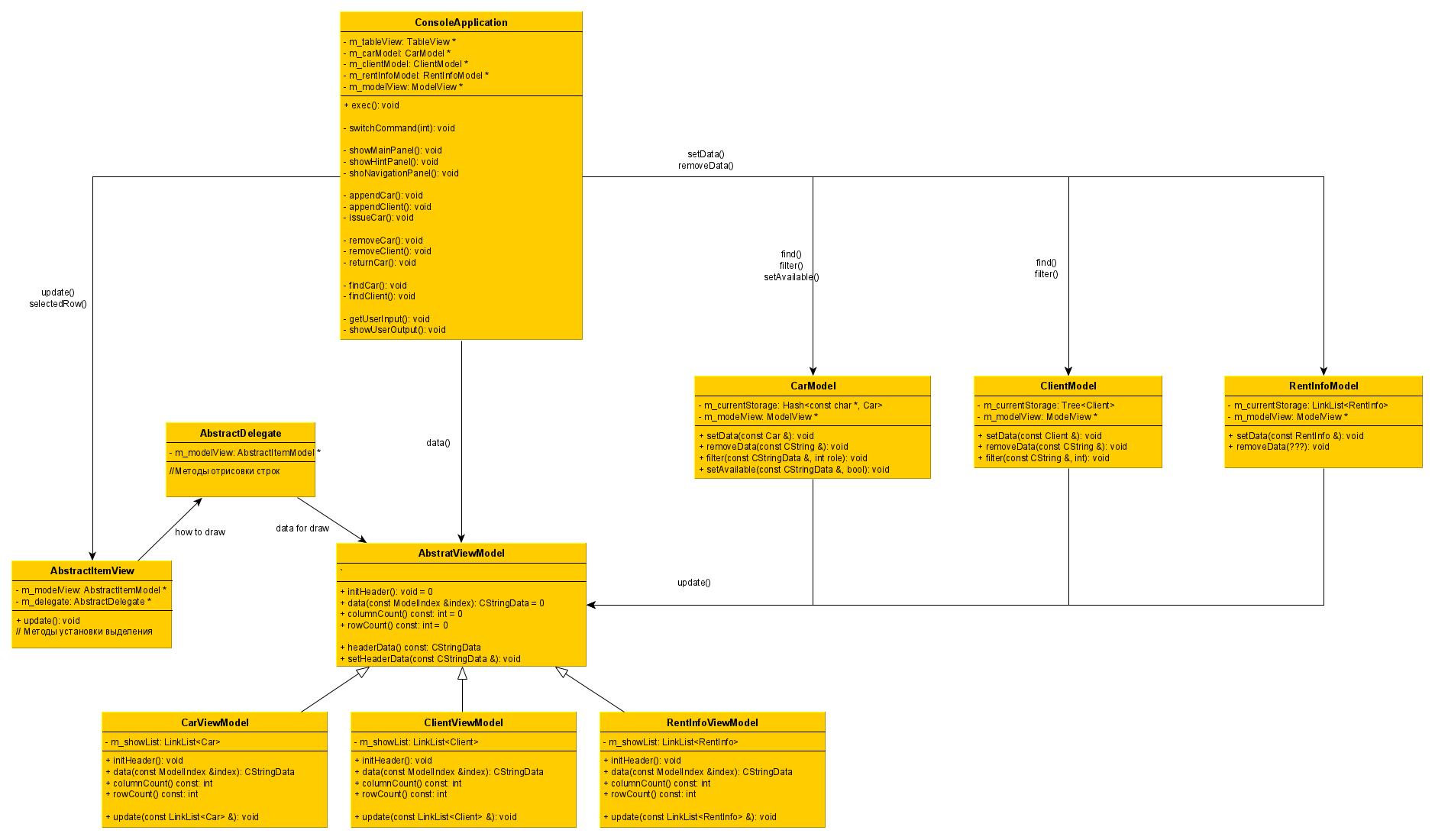


Рис. 1 Диаграмма классов разработанного приложения

Для преставления данных в табличном виде используется класс ListView, который наследует интерфейс класса AbstractItemView. Данный класс содержит логику отрисовки таблицы, но за саму отрисовку не отвечает, а делегирует данные функции классу ListDelegate, который наследует интерфейс AbstractDelegate и на который AbstracItemView содержит указатель, Данный класс содержит логике отрисовки самой таблицы с учетом выделения (которое задается классом AbstractItemView), а также содержит указатель на модель представлений AbstractViewModel, определяющую порядок отрисовки данных.

Сами данные содержатся в трех моделях данных – CarModel, ClientModel и RentInfoModel. При изменении данных в этих моделях, они уведомляют модель AbstractViewModel с которой они ассоциированы о данном изменении. Модель представления в свою очередь забирает все данные, необходимые для представления в виде связного списка. При этом представление об этих изменениях не уведомляется. Это значит, что при изменении данных представление не будет обновлено.

Контролем за обновлением представления, а также за пользовательским вводом занимается класс ConsoleApplication. Данный класс по сути является конечным автоматом, который может находится в одном из нескольких состояний. В каждом из состояний определено какая таблица должна быть отображена, и какие действия с ней в данный момент производятся.

При старте разработанного приложения появляется пользовательское меню (рис. 2).

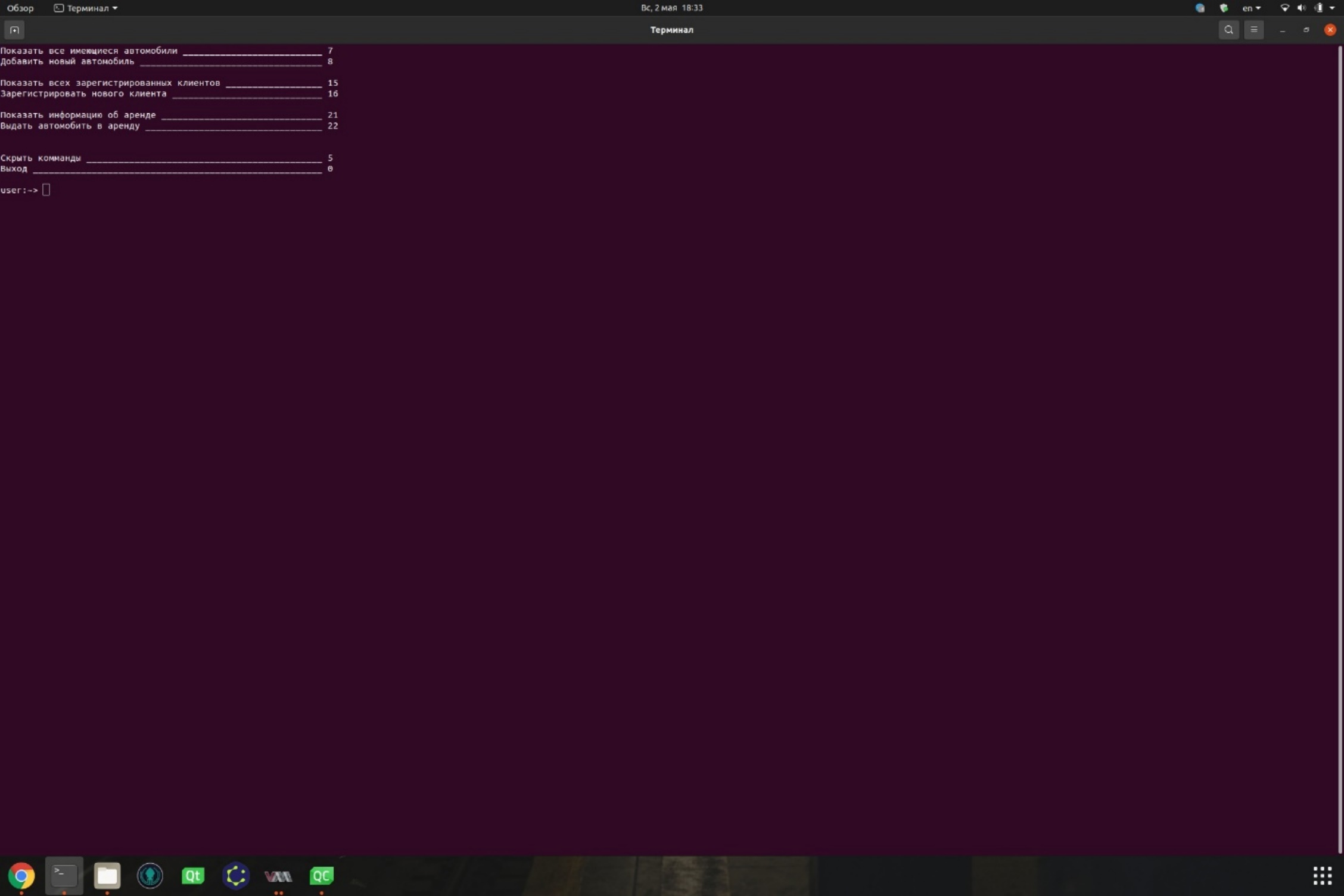


Рис. 2 Стартовое меню разработанного приложения

Меню разделено на пять частей: команды управления таблицей клиентов, автомобилей и информацией об аренде, команды навигации по текущей активной таблице, дополнительные команды отображения меню и таблиц.

Единовременно может быть отображена одна из таблиц, при этом меню будет содержать только команды для данной таблицы (рис. 3, рис.4, рис. 5).

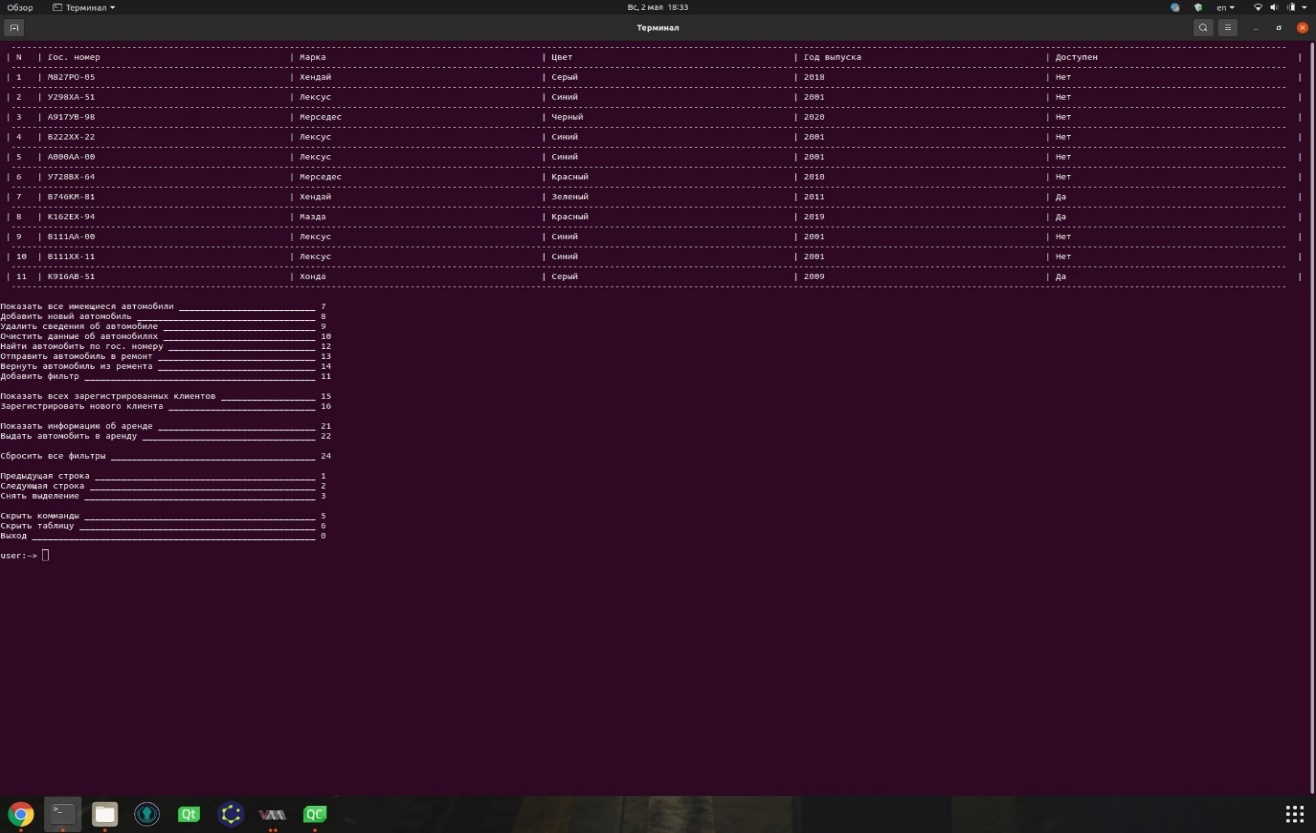


Рис. 3 Пример отображения таблицы информации об аренде

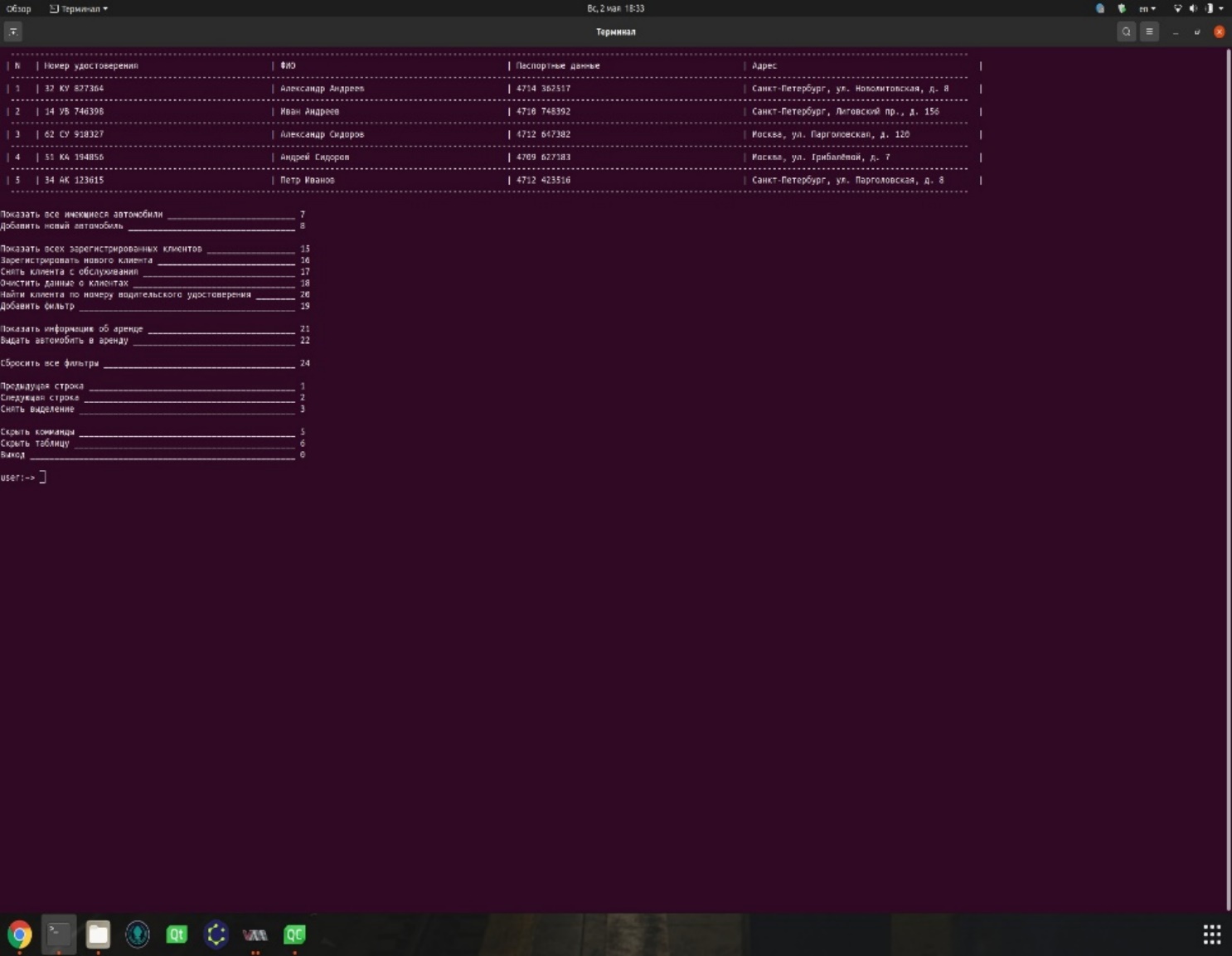


Рис. 4 Пример отображения информации о клиентах

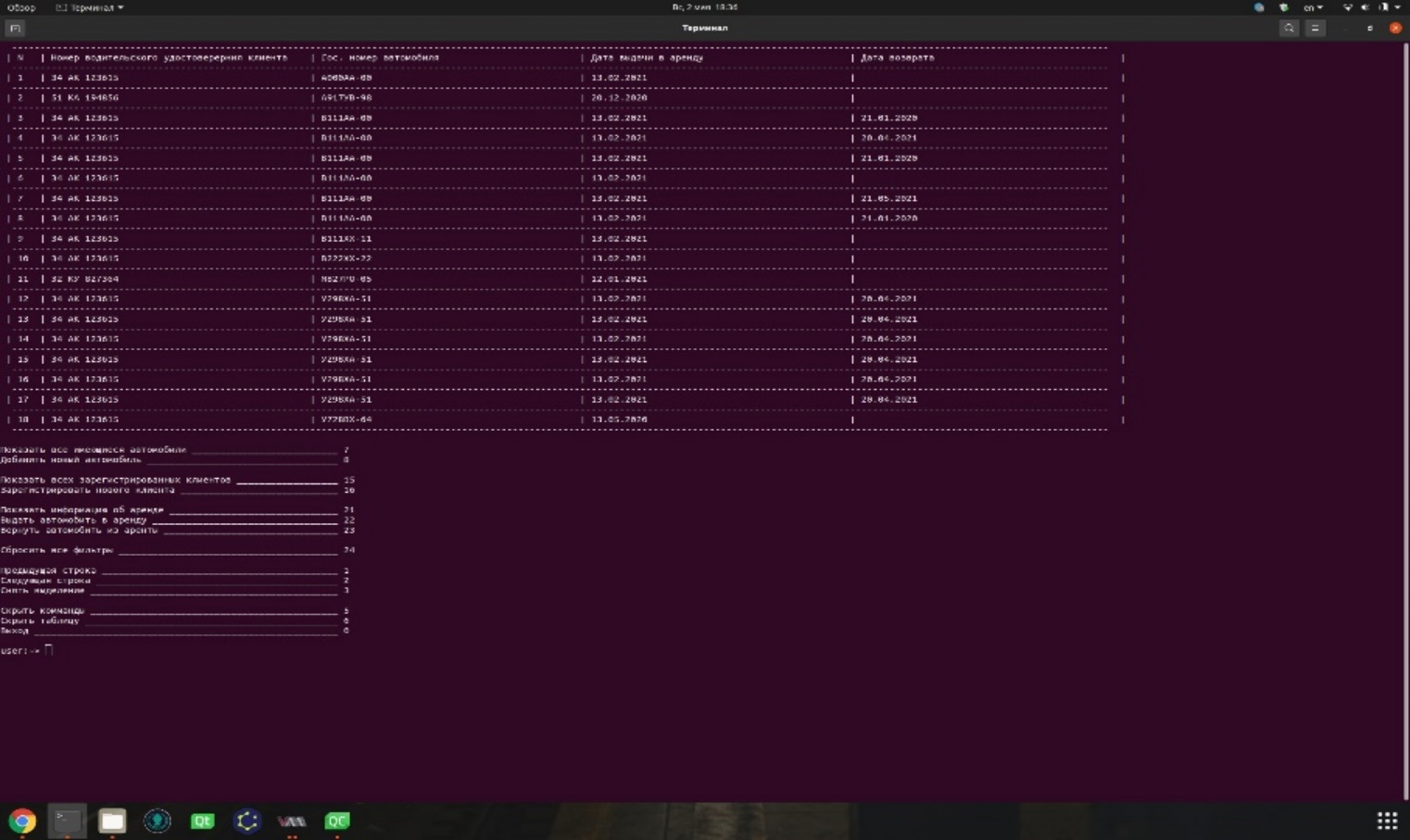


Рис. 5 Пример отображения информации об автомобилях

Исходный код приложения представлен в приложении 2.

1. Тестирование программы

Для тестирования программы били подготовлены тестовые данные, которые загружаются при старте программы. Тестовые данные приведены в табл.1, табл.2, табл.3.

|  |  |  |  |
| --- | --- | --- | --- |
| Номер вод. удостоверения | Гос. номер авто | Дата выдачи | Дата возврата |
| 34 АК 123615 | В111АА-00 | 13.02.2021 |  |
| 34 АК 123615 | А000АА-00 | 13.02.2021 |  |
| 34 АК 123615 | У298ХА-51 | 13.02.2021 | 20.04.2021 |
| 34 АК 123615 | В111АА-00 | 13.02.2021 | 21.01.2020 |
| 34 АК 123615 | В111АА-00 | 13.02.2021 | 20.04.2021 |
| 34 АК 123615 | В111АА-00 | 13.02.2021 | 21.01.2020 |
| 32 КУ 827364 | М827РО-05 | 12.01.2021 |  |
| 34 АК 123615 | У298ХА-51 | 13.02.2021 | 20.04.2021 |
| 51 КА 194856 | А917УВ-98 | 20.12.2020 |  |
| 34 АК 123615 | В111АА-00 | 13.02.2021 | 21.05.2021 |
| 34 АК 123615 | У728ВХ-64 | 13.05.2020 |  |
| 34 АК 123615 | У298ХА-51 | 13.02.2021 | 20.04.2021 |
| 34 АК 123615 | В111ХХ-11 | 13.02.2021 |  |
| 34 АК 123615 | В111АА-00 | 13.02.2021 | 21.01.2020 |
| 34 АК 123615 | У298ХА-51 | 13.02.2021 | 20.04.2021 |
| 34 АК 123615 | В222ХХ-22 | 13.02.2021 |  |
| 34 АК 123615 | У298ХА-51 | 13.02.2021 | 20.04.2021 |
| 34 АК 123615 | У298ХА-51 | 13.02.2021 | 20.04.2021 |

Табл. 1 Тестовые данные для таблицы RentInfo

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Гос. Номер | Марка | Цвет | Год выпуска | Доступность |
| А917УВ-98 | Мерседес | Черный | 2020 | 0 |
| К916АВ-51 | Хонда | Серый | 2009 | 1 |
| У728ВХ-64 | Мерседес | Красный | 2010 | 0 |
| М827РО-05 | Хендай | Серый | 2018 | 0 |
| К162ЕХ-94 | Мазда | Красный | 2019 | 1 |
| В746КМ-81 | Хендай | Зеленый | 2011 | 1 |
| У298ХА-51 | Лексус | Синий | 2001 | 0 |
| В111АА-00 | Лексус | Синий | 2001 | 0 |
| В111ХХ-11 | Лексус | Синий | 2001 | 0 |
| В222ХХ-22 | Лексус | Синий | 2001 | 0 |
| А000АА-00 | Лексус | Синий | 2001 | 0 |

Табл. 2 Тестовые данные для таблицы Cars

|  |  |  |  |
| --- | --- | --- | --- |
| Номер удостоверения | ФИО | Паспортные данные | Адрес |
| 34 АК 123615 | Петр Иванов | 4712 423516 | Санкт-Петербург, ул. Парголовская, д. 8 |
| 14 УВ 746398 | Иван Андреев | 4710 748392 | Санкт-Петербург, Лиговский пр., д. 156 |
| 51 КА 194856 | Андрей Сидоров | 4709 627183 | Москва, ул. Грибалёвой, д. 7 |
| 62 СУ 918327 | Александр Сидоров | 4712 647382 | Москва, ул. Парголовская, д. 120 |
| 32 КУ 827364 | Александр Андреев | 4714 362517 | Санкт-Петербург, ул. Новолитовская, д. 8 |

Табл. 3 Тестовые данные таблицы Clients

Как можно увидеть на рис. 3, рис. 4 и рис. 5, отображение данных происходит верно. Также, как и указано в задании, таблица RentInfo поддерживается в отсортированном состоянии.

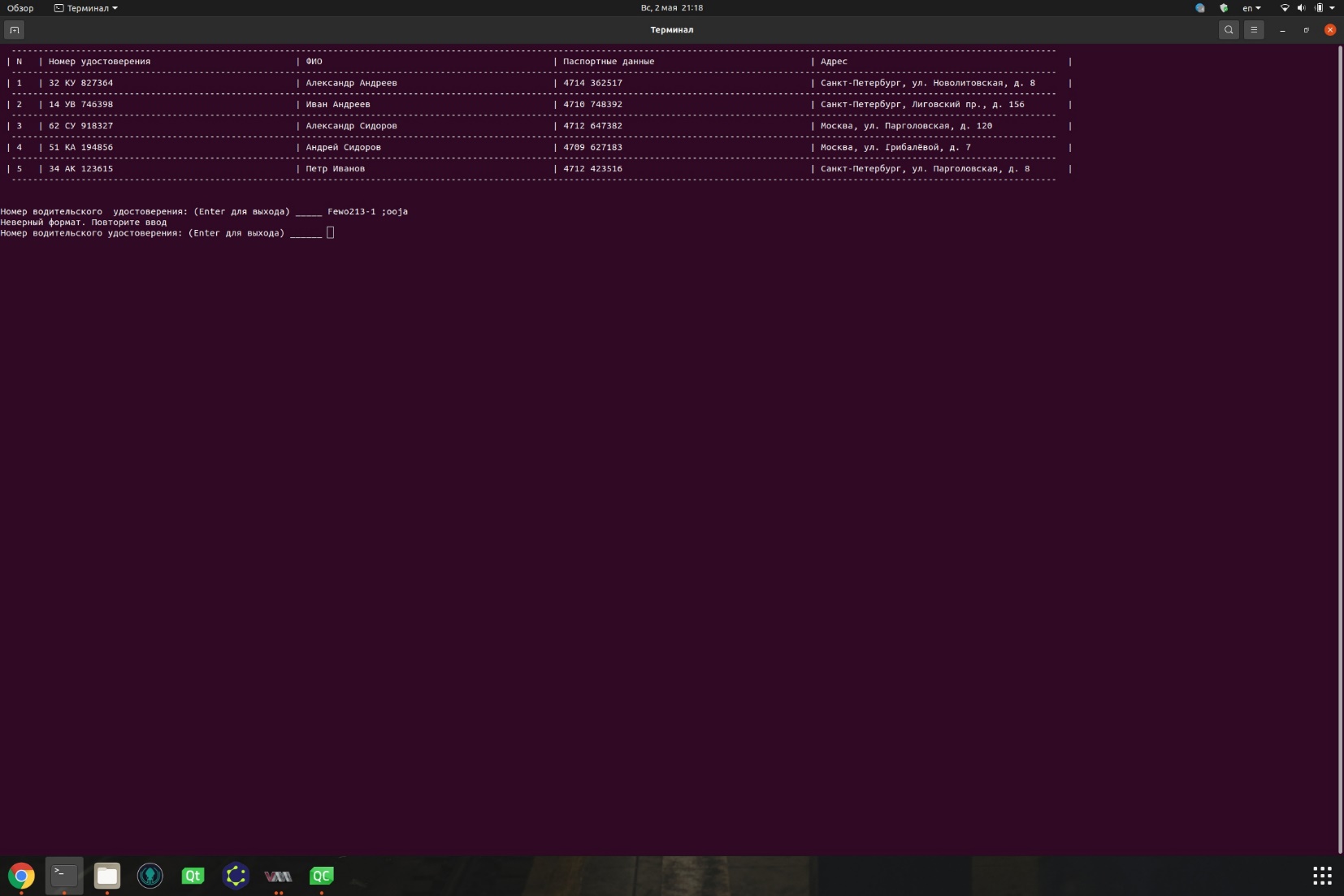
На рис. 6 и рис. 7 можно увидеть, что программа проверяет правильность введенного пользователем номера автомобиля и номера водительского удостоверения

Рис. 6 Проверка правильности введенных пользователем данных в таблице Cars

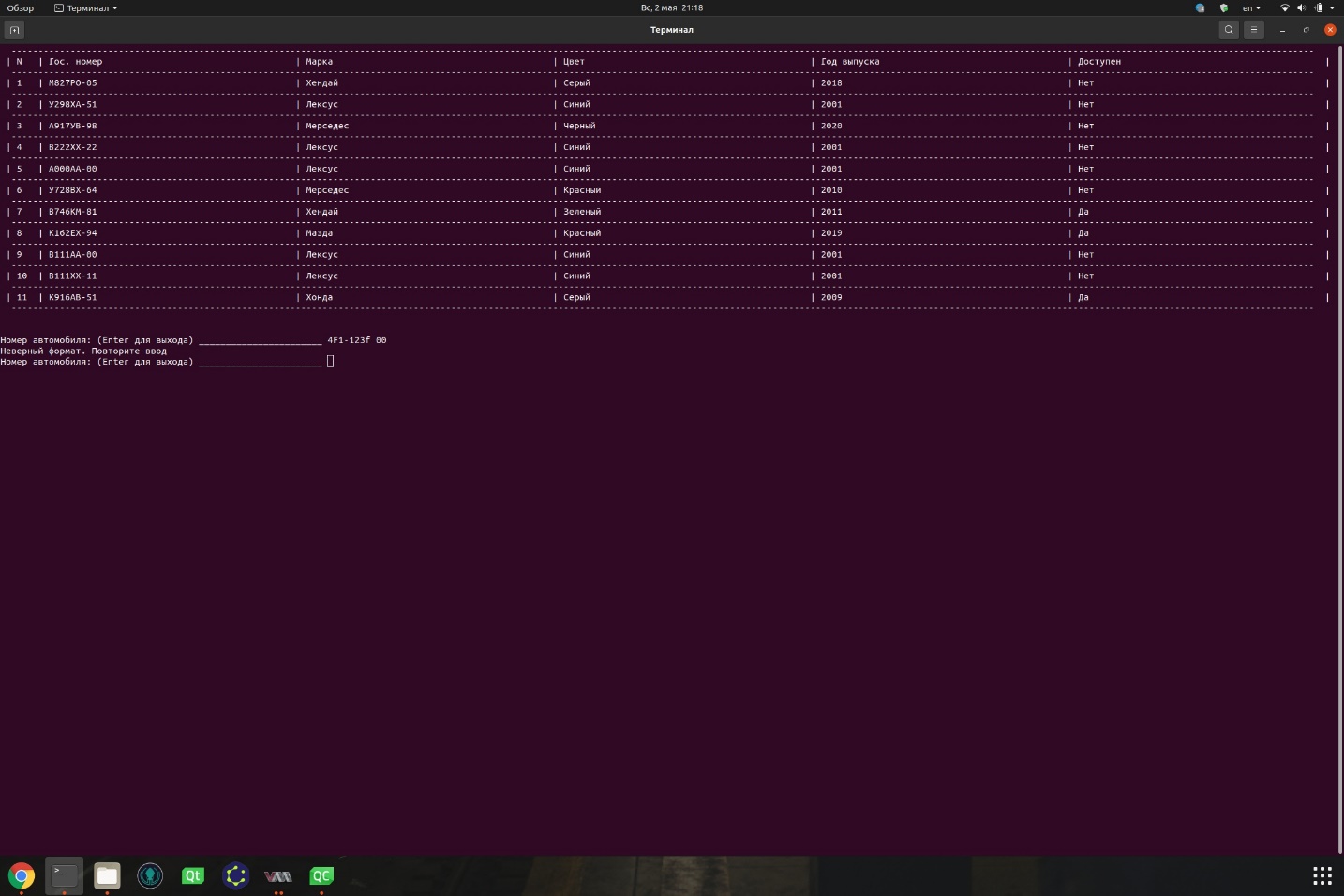


Рис. 7 Проверка правильности введенных пользователем данных в таблице Clients

На рис. 8 приставлен тест функции поиска автомобиля по гос. Номеру.

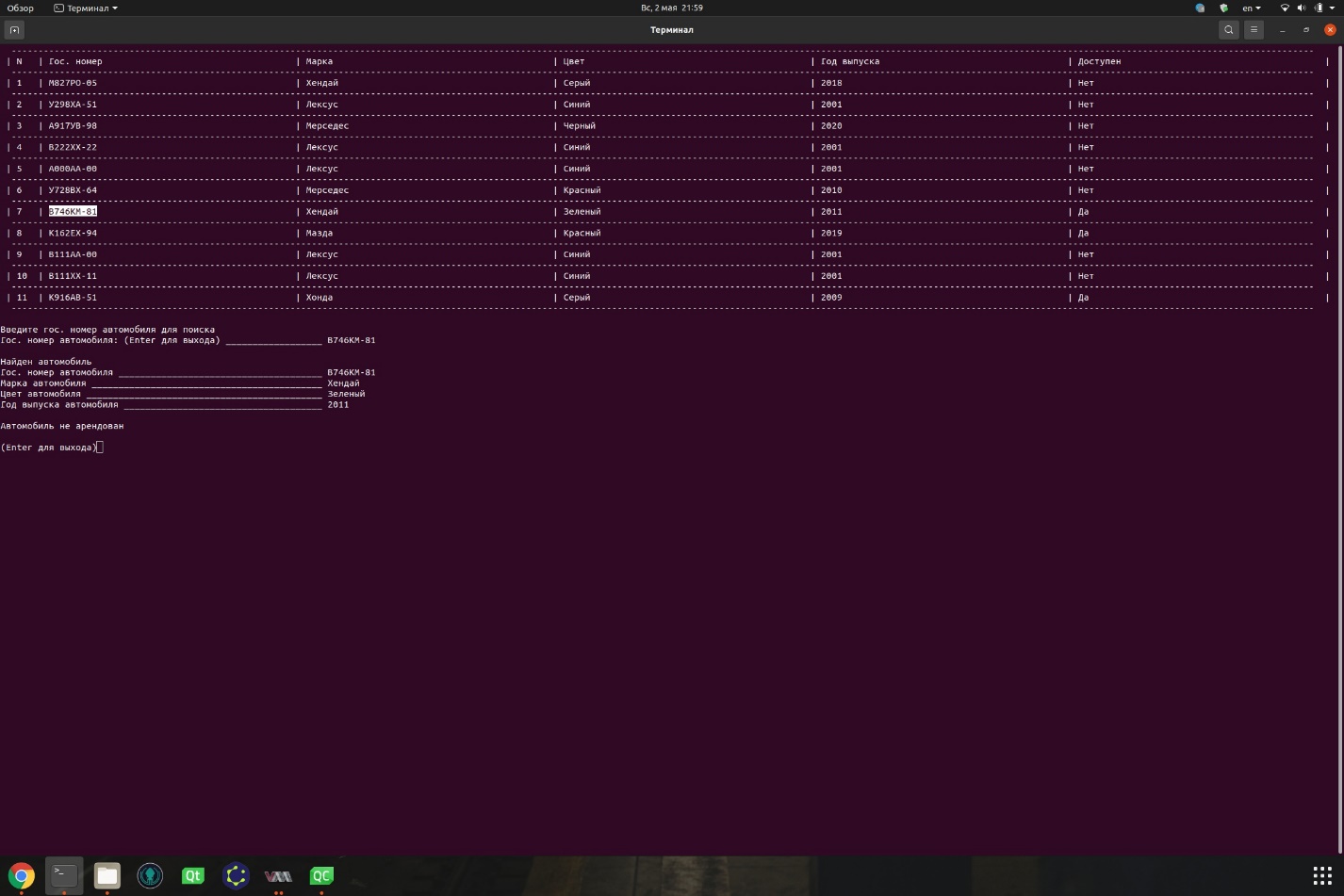


Рис. 8 Поиск автомобиля по гос. номеру

На рис. 9 представлен тест функции фильтрации таблицы автомобилей марке.

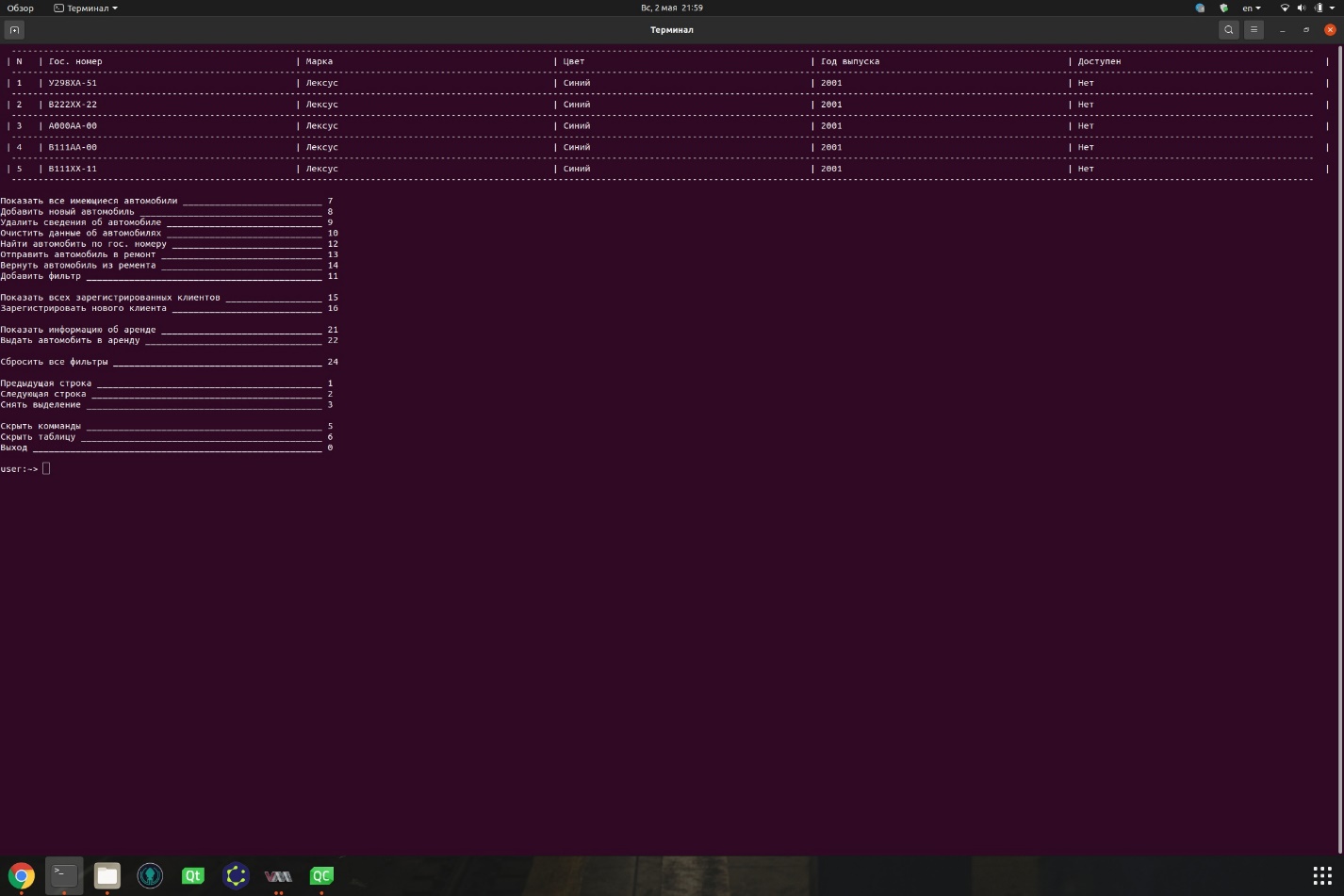


Рис. 9 Фильтрация автомобилей по марке Лексус

На рис. 10 представлен тест удаления клиентов из таблицы. Как можно увидеть, программа не позволяет удальть запись, если клиент имеет арендованные автомобили

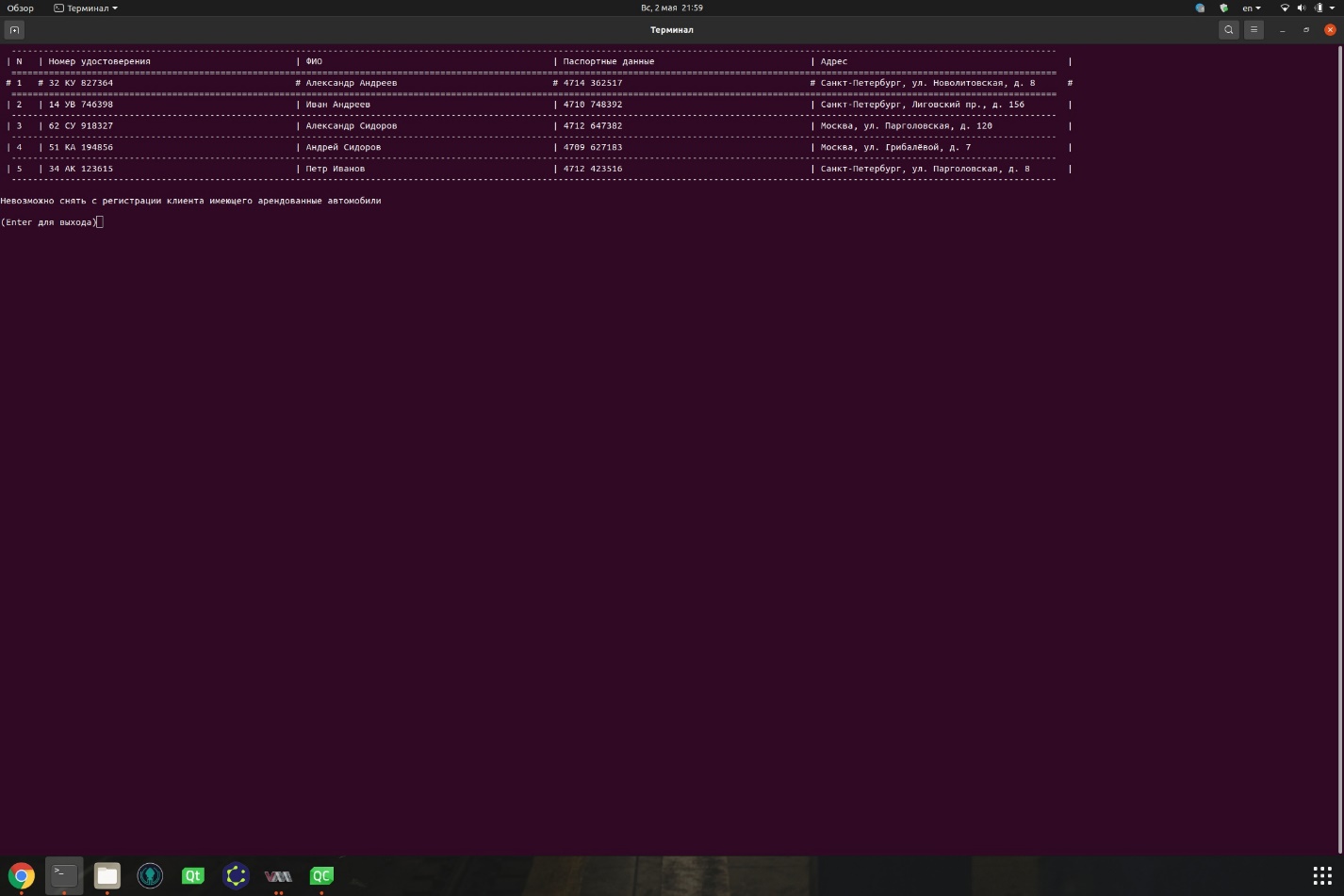


Рис. 10 Удаление клиентов, имеющих автомобиль в аренде

На рис. 11 представлен тест удаления автомобилей из таблицы. Как можно увидеть, программа не позволяет удалить втомобили, недоступные для аренды (уже арендованные или находящиеся в ремонте)

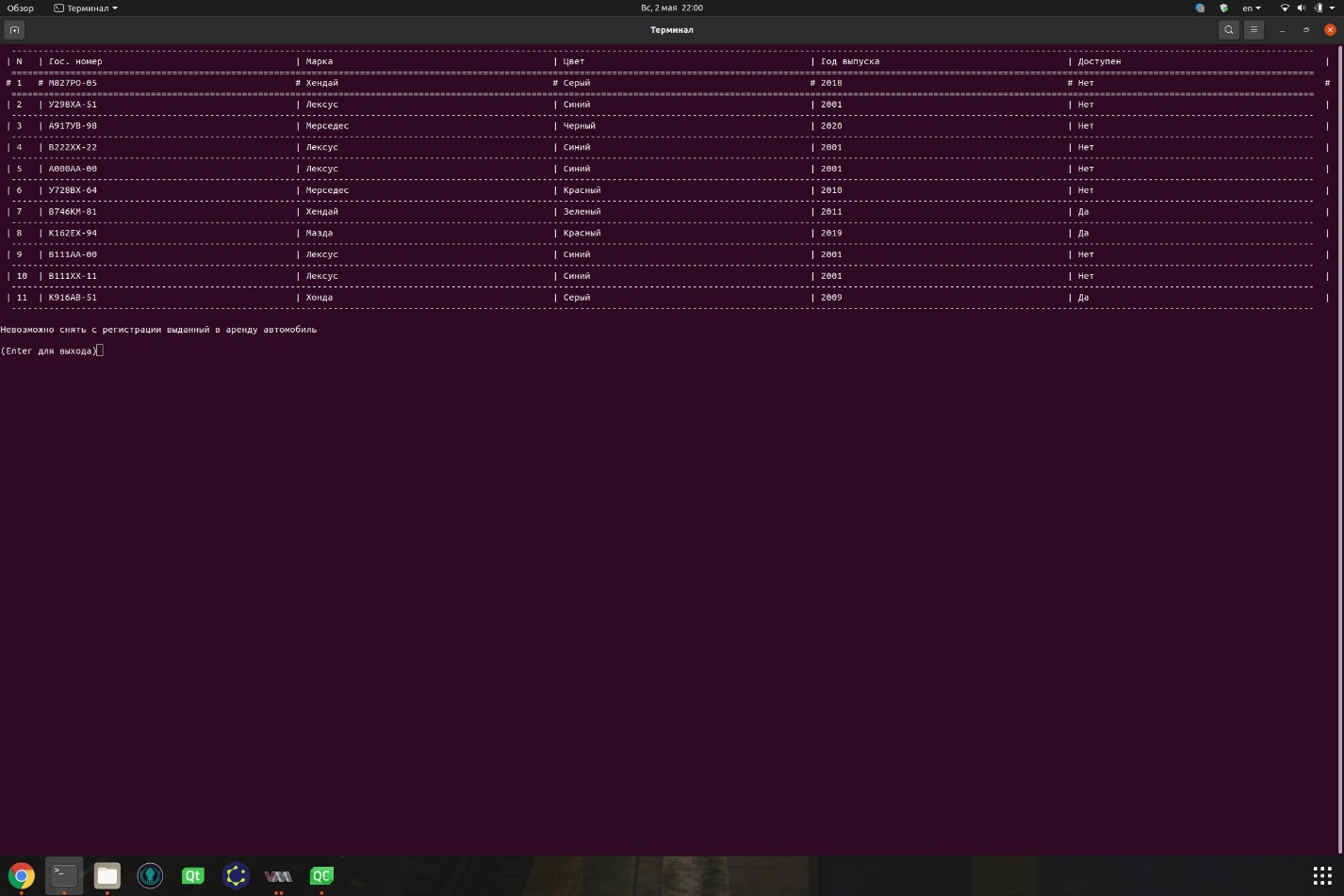


Рис. 11 Удаление недоступных автомобилей

На рис. 12 представлен тест функции поиска клиента по номеру водительского удостоверения.

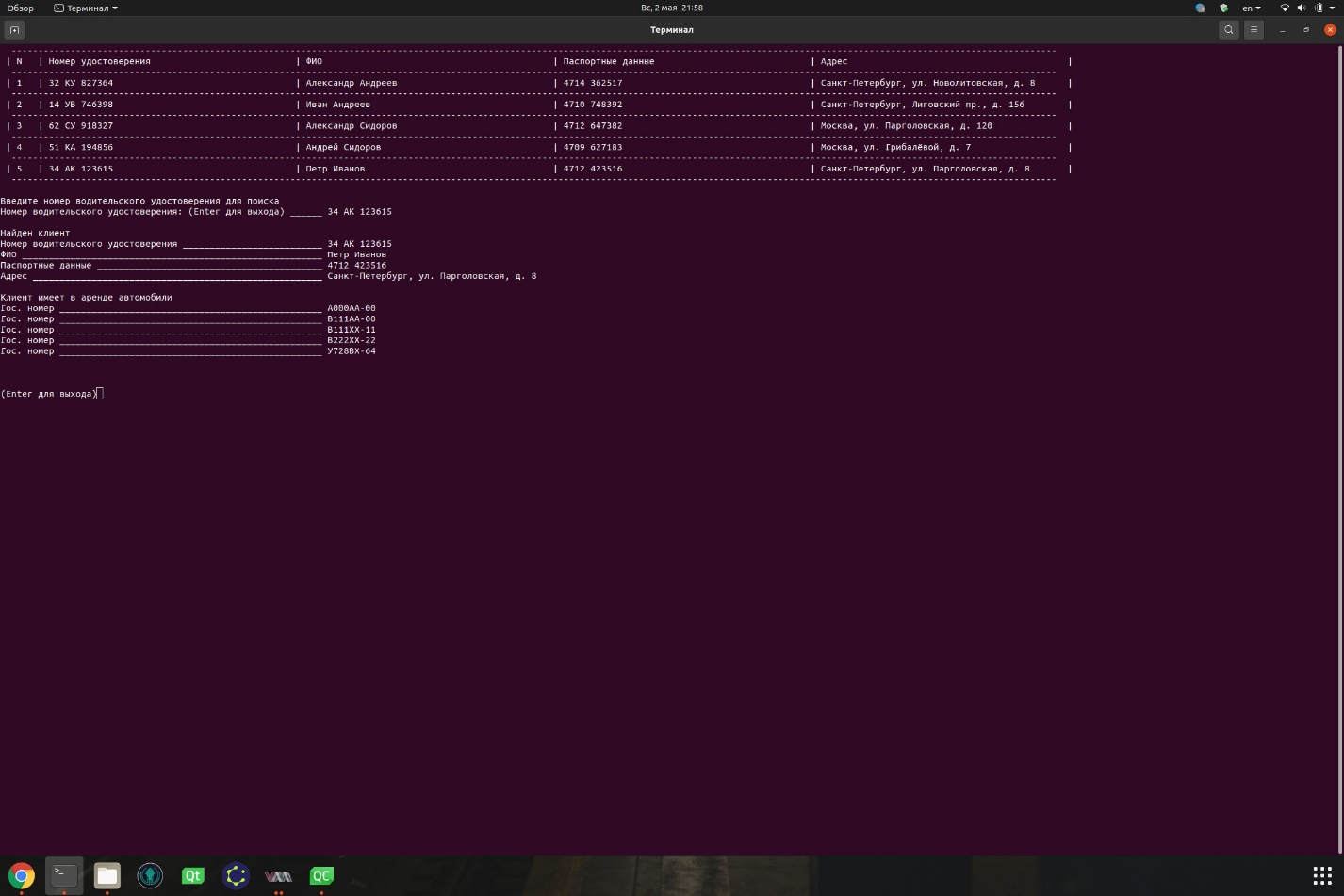


Рис. 12 Поиск клиента по номеру удостоверения

На рис. 13 и рис. 14 представлен тест функции таблицы клиентов по ФИО и адресу.

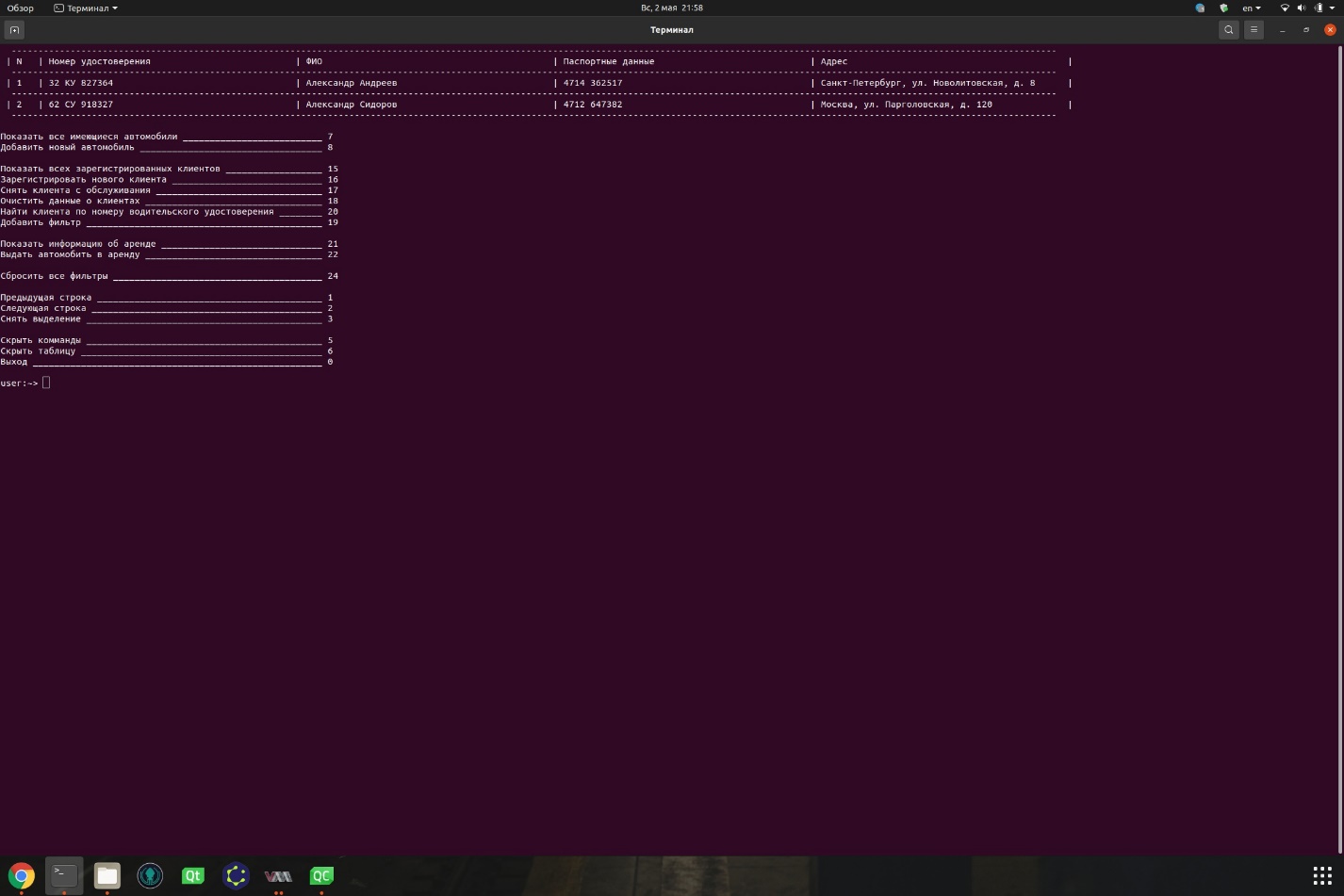


Рис. 13 Фильтрация клиентов по ФИО

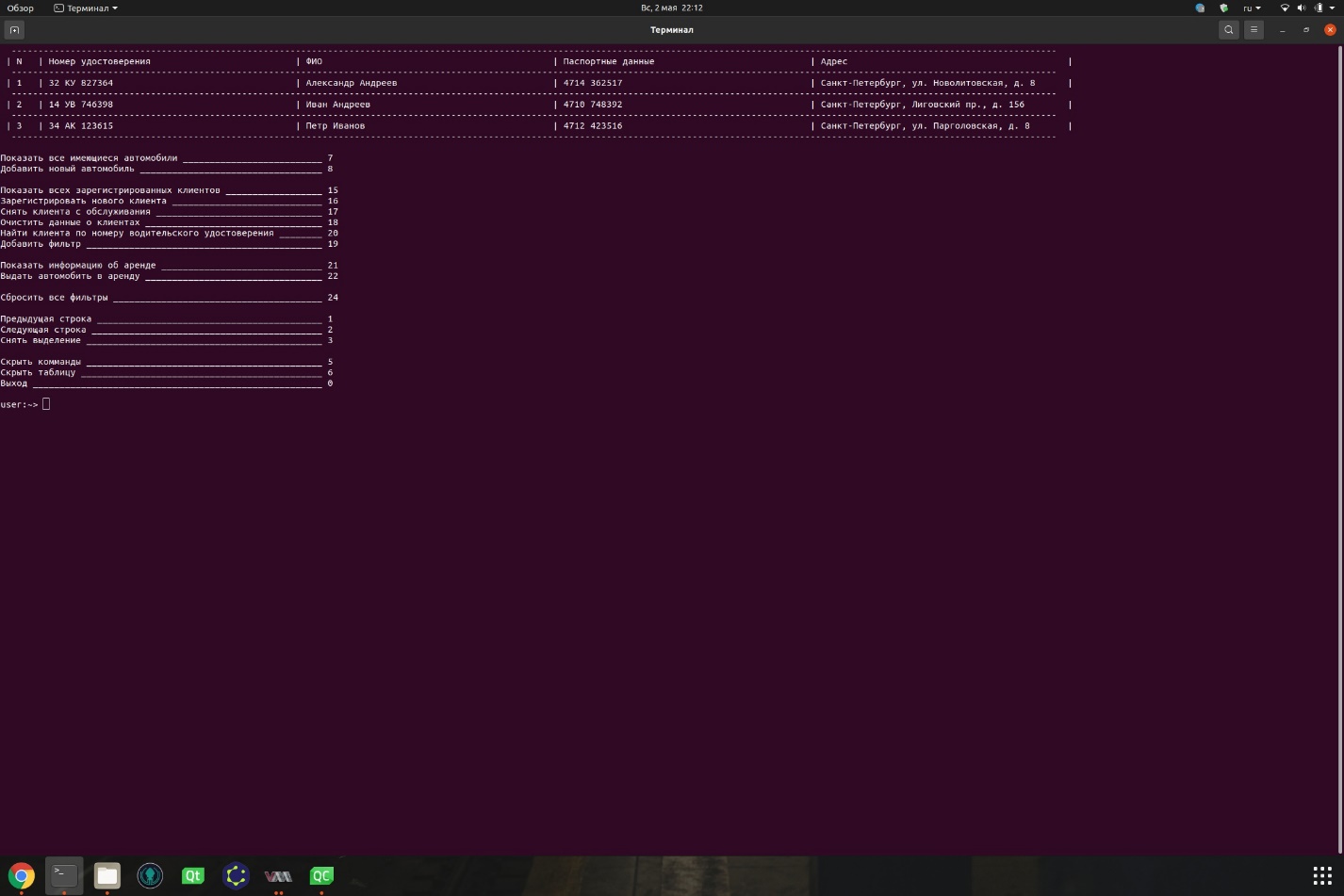


Рис. 14 Фильтрация клиентов по адресу

На рис. 15 показано, что программа проверяет, доступен ли автомобиль для аренды и выдает сообщение о невозможности выдачи в аренду в случае, если он не доступен (уже находится в аренде или в ремонте).

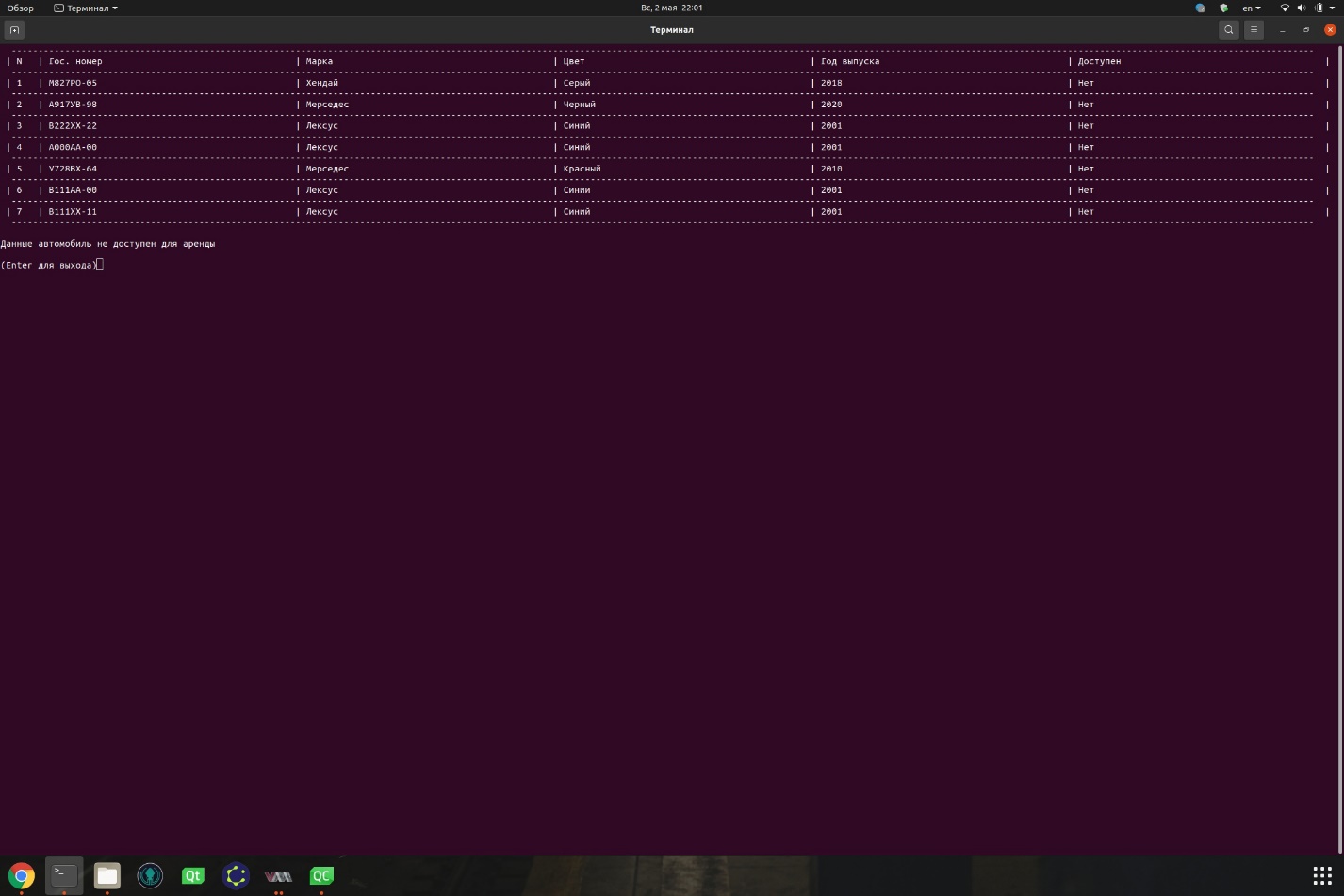


Рис. 15 Выдача в аренду недоступного автомобиля

1. Заключение

В ходе выполнения курсового проекта была разработана информационная система для заданной предметной области «Обслуживание клиентов в бюро проката автомобилей», которая позволяет хранить связанные между собой массивы данных и отображать их в понятном человеку виде, осуществлять поиск, модификацию, сортировку, удаление и фильтрацию данных. Для хранения данных при работе программы разработаны структуры данных двунаправленный связный список, хеш-таблица и бинарное дерево поиска. Для отображения информации был реализован паттерн MVVM, а также создана псевдографика для представления таблиц.

Из достоинств можно выделить:

* понятный пользователю интерфейс, который имеет цикл обновления при изменении данных;
* возможность поиска и фильтрации информации;
* невозможность изменения данных при условиях, которые не позволяют это сделать (например, удаление информации при выданных в аренду автомобилях);
* возможность навигации по таблицам;

Из недостатков можно отметить:

* отсутствие полноценного GUI;
* быстрая сортировка (Хоара) производится на списке, а так как список не имеет произвольного доступа к памяти, это увеличивает временную сложность алгоритма;
* отсутствует функция сохранения изменений при закрытии приложения;

.

Список использованной литературы

1. Ключарев А.А., Матьяш В.А., Щекин С.В. Структуры и алгоритмы обработки данных: Учебное пособие / СПбГУАП. СПб., 2004.
2. Страуструп, Б. Язык программирования C++ специальное издание / Б. Страуструп ; пер.: С. Анисимов, М. Кононов ; ред.: Ф. Андреев, А. Ушаков. - [Б. м.] : Бином-Пресс, 2008. - 1098 с.
3. Кнут, Д. Искусство программирования [в 3 т.]. Т. 1. Основные алгоритмы / Д. Кнут ; ред. Ю. В. Козаченко. - 3-е изд. - М. : Вильямс, 2014. - 720 с.
4. Прата, Стивен Язык программирования С++. Лекции и упражнения, 6-е изд. Пер. с англ. – М.:ООО «И.Д. Вильямс», 2012. – 1248 с.: ил. - Парал. тит. англ.

Приложение 1

Исходные коды структур данных

Линейный двусвязный список

#ifndef LINK\_LIST

#define LINK\_LIST

#include "global.h"

#include <iostream>

template<typename T> class LinkList;

template<typename Node> class ListIterator;

template<typename T>

class node

{

public:

using Value = T;

node() : m\_data(nullptr), m\_next(this), m\_previous(this) {}

node(Value \*data, node \*next, node \*prev)

: m\_data(data)

, m\_next(next)

, m\_previous(prev) {}

~node() { if(m\_data) delete m\_data; m\_data = nullptr; }

node \*previous() const { return m\_previous; }

void setPrevious(node \*prev) { m\_previous = prev; }

node \*next() const { return m\_next; }

void setNext(node \*next) { m\_next = next; }

Value \*data() { return m\_data; }

void setData(Value \*data) { m\_data = data; }

private:

Value \*m\_data;

node \*m\_next;

node \*m\_previous;

};

template<typename Node>

class ListIterator

{

using Value = typename Node::Value;

public:

ListIterator() { m\_ptr = nullptr; }

ListIterator(const ListIterator &other) { m\_ptr = other.m\_ptr; }

explicit ListIterator(Node\* node) : m\_ptr(node) {}

ListIterator &operator=(const ListIterator &other)

{

if(this == &other)

return \*this;

m\_ptr = other.m\_ptr;

return \*this;

}

Value& operator\*() { return \*m\_ptr->data(); }

const Value &operator\*() const { return \*m\_ptr->data(); }

Value \*operator->() { return m\_ptr->data(); }

bool operator==(const ListIterator &rightIter) const { return m\_ptr == rightIter.m\_ptr; }

bool operator!=(const ListIterator &rightIter) const { return m\_ptr != rightIter.m\_ptr; }

bool operator!() const { return m\_ptr == nullptr; }

operator Value () { return \*m\_ptr->data(); }

ListIterator operator++(int)

{

assert(m\_ptr != nullptr);

ListIterator temp = \*this;

++(\*this);

return temp;

}

ListIterator &operator++()

{

assert(m\_ptr != nullptr);

m\_ptr = m\_ptr->next();

return \*this;

}

ListIterator operator--(int)

{

assert(m\_ptr != nullptr);

ListIterator temp = \*this;

--(\*this);

return temp;

}

ListIterator &operator--()

{

assert(m\_ptr != nullptr);

m\_ptr = m\_ptr->previous();

return \*this;

}

ListIterator operator+(int i)

{

ListIterator temp = \*this;

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

assert(m\_ptr != nullptr);

++temp;

}

return temp;

}

ListIterator operator-(int i)

{

ListIterator temp = \*this;

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

assert(m\_ptr != nullptr);

--temp;

}

return temp;

}

ListIterator &operator+=(int i)

{

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

assert(m\_ptr != nullptr);

m\_ptr = m\_ptr->next();

}

return \*this;

}

ListIterator &operator-=(int i)

{

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

assert(m\_ptr != nullptr);

m\_ptr = m\_ptr->previous();

}

}

int operator-(const ListIterator &right)

{

assert(!(right > \*this));

ListIterator iter = right;

int count = 0;

while(iter != \*this) {

++count;

++iter;

}

return count;

}

bool operator<(const ListIterator &right) const

{

assert(right.\_\_getNode() != nullptr);

if(\*this == right)

return false;

ListIterator iter(m\_ptr);

while(iter.m\_ptr != nullptr) {

if(iter == right)

return true;

++iter;

}

return false;

}

bool operator>(const ListIterator &left) const

{

return !(\*this < left) && \*this != left;

}

bool isValid() const { return m\_ptr != nullptr && m\_ptr->data() != nullptr; }

Node \*\_\_getNode() const { return m\_ptr; }

private:

Node \*m\_ptr;

};

template<typename T>

class LinkList

{

typedef node<T> Node;

public:

typedef ListIterator<Node> iterator;

LinkList() : m\_begin(new Node(new T(), nullptr, nullptr)), m\_end(m\_begin), m\_size(0) {}

LinkList(const LinkList<T>& li)

{

for(LinkList<T>::iterator iter = li.begin(); iter != li.end(); ++iter)

append(\*iter);

}

~LinkList()

{

clear();

delete m\_begin;

}

int size() const { return m\_size; }

bool isEmpty() const { return m\_begin == m\_end; }

iterator begin() const { return iterator(m\_begin); }

iterator end() const { return iterator(m\_end);}

LinkList<T>& operator=(const LinkList<T>& li)

{

if (this == &li)

return \*this;

clear();

for(LinkList<T>::iterator it = li.begin(); it != li.end(); ++it)

append(\*it);

return \*this;

}

T& operator[](int i)

{

assert(i < m\_size);

if(!(i < m\_size))

return \*m\_end->data();

Node \*pNode = m\_begin;

for(int j = 0; i != j; ++j)

pNode = pNode->next();

return \*pNode->data();

}

const T& operator[](int i) const

{

assert(i < m\_size);

if(!(i < m\_size))

return \*m\_end->data();

Node \*pNode = m\_begin;

for(int j = 0; i != j; ++j)

pNode = pNode->next();

return \*pNode->data();

}

bool contains(const T &val) const

{

for(LinkList<T>::iterator it = begin(); it != end(); ++it)

if(\*it == val)

return true;

return false;

}

void append(const T &val)

{

if(isEmpty()) {

\*m\_begin->data() = val;

m\_end = new Node(new T(), nullptr, m\_begin);

m\_begin->setNext(m\_end);

}

else {

\*m\_end->data() = val;

Node \*newEnd = new Node(new T(), nullptr, m\_end);

m\_end->setNext(newEnd);

m\_end = newEnd;

}

++m\_size;

}

void prepend(const T &val)

{

if(isEmpty()) {

append(val);

return;

}

else {

Node \*newBegin = new Node(new T(val), m\_begin, nullptr);

m\_begin->setPrevious(newBegin);

m\_begin = newBegin;

}

++m\_size;

}

iterator insert(const T& val, iterator before)

{

assert(before.\_\_getNode());

if(!before.\_\_getNode())

return iterator();

else if(before.\_\_getNode() == m\_begin) {

prepend(val);

return iterator(m\_begin);

}

else if(before.\_\_getNode() == m\_end) {

append(val);

return iterator(m\_end->previous());

}

Node \*next = before.\_\_getNode();

Node \*prev = next->previous();

Node \*newNode = new Node(new T(val), next, prev);

next->setPrevious(newNode);

prev->setNext(newNode);

++m\_size;

return iterator(newNode);

}

iterator erase(iterator delIter)

{

assert(delIter.isValid());

if(!delIter.isValid())

return iterator();

else if(delIter.\_\_getNode() == m\_begin) {

Node \*newBegin = m\_begin->next();

newBegin->setPrevious(nullptr);

delete m\_begin;

m\_begin = newBegin;

--m\_size;

return iterator(m\_begin);

}

else if(delIter.\_\_getNode() == m\_end)

return delIter;

Node \*delNode = delIter.\_\_getNode();

Node \*prev = delNode->previous();

Node \*next = delNode->next();

prev->setNext(next);

next->setPrevious(prev);

delete delNode;

--m\_size;

return iterator(next);

}

void clear()

{

while(!isEmpty())

erase(begin());

}

void swap(iterator left, iterator right)

{

if(left == right)

return;

Node \*leftNode = left.\_\_getNode();

Node \*rightNode = right.\_\_getNode();

assert(right.\_\_getNode() != nullptr && left.\_\_getNode() != nullptr && right.\_\_getNode() != m\_end);

if(leftNode == m\_begin)

m\_begin = rightNode;

if(leftNode->next() == rightNode) {

Node \*rTemp = rightNode->next();

Node \*lTemp = leftNode->previous();

rightNode->setNext(leftNode);

leftNode->setNext(rTemp);

leftNode->setPrevious(rightNode);

rightNode->setPrevious(lTemp);

rTemp->setPrevious(leftNode);

if(lTemp)

lTemp->setNext(rightNode);

return;

}

Node \*llTemp = leftNode->previous();

Node \*rrTemp = rightNode->next();

Node \*lrTemp = leftNode->next();

Node \*rlTemp = rightNode->previous();

rightNode->setNext(lrTemp);

leftNode->setNext(rrTemp);

rightNode->setPrevious(llTemp);

leftNode->setPrevious(rlTemp);

if(llTemp)

llTemp->setNext(rightNode);

rrTemp->setPrevious(leftNode);

lrTemp->setPrevious(rightNode);

rlTemp->setNext(leftNode);

}

private:

Node \*m\_begin;

Node \*m\_end;

int m\_size;

};

#endif //LINK\_LIST

Хеш-таблица

#ifndef HASH\_H

#define HASH\_H

#include <cstring>

#include <cmath>

#include "global.h"

#include "linklist.h"

template<typename Val, int KeyLen>

class Hash

{

static const int minHashFunc;

static const int maxHashFunc;

static const int linearStep;

static const int initCapacity;

union key\_helper {

char c[KeyLen];

int n;

};

struct Data {

explicit Data(Val val = Val())

: m\_data(val)

, m\_key(new char[KeyLen])

, m\_empty(true)

, m\_deleted(false) {}

~Data() { delete [] m\_key; }

Val m\_data;

char\* m\_key;

bool m\_empty;

bool m\_deleted;

};

public:

Hash()

: m\_size(0)

, m\_capacity(initCapacity)

, m\_multiCoef(static\_cast<int>(m\_capacity / (maxHashFunc - minHashFunc)))

, m\_elements(new Data[m\_capacity]) {}

~Hash() { delete[] m\_elements; }

void insert(const char\* key, const Val& value);

void erase(const char\* key);

void clear();

Val &operator[](const char\* key);

const Val &operator[](const char \*key) const;

LinkList<char \*> keys() const;

LinkList<Val> values() const;

const char \*key(const Val &val) const;

bool hasKey(const char \*key) const;

int size() const;

int capacity() const;

private:

void resize(int newSize);

int hashFunction(const char\* key) const;

void linearTesting(int tryNum, int& seg) const;

private:

int m\_size;

int m\_capacity;

int m\_multiCoef;

Data\* m\_elements;

};

template<typename Val, int KeyLen>

const int Hash<Val, KeyLen>::minHashFunc = 240;

template<typename Val, int KeyLen>

const int Hash<Val, KeyLen>::maxHashFunc = 357;

template<typename Val, int KeyLen>

const int Hash<Val, KeyLen>::linearStep = 2;

template<typename Val, int KeyLen>

const int Hash<Val, KeyLen>::initCapacity = 500;

template<typename Val, int KeyLen>

void Hash<Val, KeyLen>::insert(const char \*key, const Val &value)

{

if(m\_size > m\_capacity \* 0.8)

resize(static\_cast<int>(m\_capacity \* 1.5));

int seg = hashFunction(key);

for(int i = 0; seg < m\_capacity; ++i) {

if(m\_elements[seg].m\_empty || m\_elements[seg].m\_deleted) {

m\_elements[seg].m\_data = value;

std::strncpy(m\_elements[seg].m\_key, key, KeyLen);

m\_elements[seg].m\_deleted = false;

m\_elements[seg].m\_empty = false;

++m\_size;

return;

}

else if(std::strncmp(m\_elements[seg].m\_key, key, KeyLen) == 0) {

m\_elements[seg].m\_data = value;

return;

}

else

linearTesting(i, seg);

}

resize(static\_cast<int>(m\_capacity \* 1.5));

insert(key, value);

}

template<typename Val, int KeyLen>

void Hash<Val, KeyLen>::erase(const char \*key)

{

int seg = hashFunction(key);

for(int i = 0; seg < m\_capacity; ++i) {

if(std::strncmp(key, m\_elements[seg].m\_key, KeyLen) == 0) {

m\_elements[seg].m\_deleted = true;

--m\_size;

return;

}

else

linearTesting(i, seg);

}

}

template<typename Val, int KeyLen>

void Hash<Val, KeyLen>::clear()

{

m\_capacity = initCapacity;

delete[] m\_elements;

m\_elements = new Data[m\_capacity];

m\_multiCoef = static\_cast<int>(m\_capacity / (maxHashFunc - minHashFunc));

}

template<typename Val, int KeyLen>

Val &Hash<Val, KeyLen>::operator[](const char \*key)

{

assert(hasKey(key));

int seg = hashFunction(key);

for (int i = 0; seg < m\_capacity; ++i) {

if (std::strncmp(m\_elements[seg].m\_key, key, KeyLen) == 0)

return m\_elements[seg].m\_data;

else {

linearTesting(i, seg);

continue;

}

}

return m\_elements[0].m\_data;

}

template<typename Val, int KeyLen>

const Val &Hash<Val, KeyLen>::operator[](const char \*key) const

{

assert(hasKey(key));

int seg = hashFunction(key);

for (int i = 0; seg < m\_capacity; ++i) {

if (std::strncmp(m\_elements[seg].m\_key, key, KeyLen) == 0)

return m\_elements[seg].m\_data;

else {

linearTesting(i, seg);

continue;

}

}

return m\_elements[0].m\_data;

}

template<typename Val, int KeyLen>

LinkList<char \*> Hash<Val, KeyLen>::keys() const

{

LinkList<char \*> keyList;

for(int i = 0; i < m\_capacity; ++i) {

if(!m\_elements[i].m\_empty && !m\_elements[i].m\_deleted)

keyList.append(m\_elements[i].m\_key);

}

return keyList;

}

template<typename Val, int KeyLen>

LinkList<Val> Hash<Val, KeyLen>::values() const

{

LinkList<Val> keyList;

for(int i = 0; i < m\_capacity; ++i) {

if(!m\_elements[i].m\_empty && !m\_elements[i].m\_deleted) {

keyList.append(m\_elements[i].m\_data);

}

}

return keyList;

}

template<typename Val, int KeyLen>

const char \*Hash<Val, KeyLen>::key(const Val &val) const

{

for(int i = 0; i < m\_capacity; ++i) {

if(val == m\_elements[i].m\_data)

return m\_elements[i].m\_key;

}

return "\0";

}

template<typename Val, int KeyLen>

bool Hash<Val, KeyLen>::hasKey(const char \*key) const

{

int seg = hashFunction(key);

for(int i = 0; seg < m\_capacity; ++i) {

if(m\_elements[i].m\_empty || m\_elements[i].m\_deleted)

continue;

else if(std::strncmp(key, m\_elements[seg].m\_key, KeyLen) == 0)

return true;

else

linearTesting(i, seg);

}

return false;

}

template<typename Val, int KeyLen>

int Hash<Val, KeyLen>::size() const

{

return m\_size;

}

template<typename Val, int KeyLen>

int Hash<Val, KeyLen>::capacity() const

{

return m\_capacity;

}

template<typename Val, int KeyLen>

void Hash<Val, KeyLen>::resize(int newSize)

{

Data \*temp = new Data[newSize];

for(int i = 0; i < m\_capacity; ++i) {

std::strncpy(temp[i].m\_key, m\_elements[i].m\_key, KeyLen);

temp[i].m\_data = m\_elements[i].m\_data;

}

delete[] m\_elements;

m\_elements = new Data[newSize]{Data()};

m\_capacity = newSize;

m\_size = 0;

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

if(!m\_elements[i].m\_empty && !m\_elements[i].m\_deleted) {

insert(temp[i].m\_key, temp[i].m\_data);

}

}

delete[] temp;

}

template<typename Val, int KeyLen>

int Hash<Val, KeyLen>::hashFunction(const char \*key) const

{

int seg = 0;

key\_helper helper;

std::strncpy(helper.c, key, KeyLen);

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

seg += helper.n;

}

return std::abs(((seg - minHashFunc) \* m\_multiCoef) % m\_capacity);

}

template<typename Val, int KeyLen>

void Hash<Val, KeyLen>::linearTesting(int tryNum, int &seg) const

{

seg += linearStep \* tryNum + tryNum % 2 + 1;

}

#endif // HASH\_H

Дерево поиска

#ifndef SEARCHTREE\_H

#define SEARCHTREE\_H

#include <cstring>

#include "global.h"

#include "linklist.h"

;

template<typename Val, int KeyLen>

class Tree {

using cmpFunc = bool(\*)(const Val &, const Val &);

struct Node {

Node(const Val &data = Val(), const char \*key = nullptr, Node \*right = nullptr, Node \*left = nullptr)

: m\_data(data)

, m\_key(new char[KeyLen])

, m\_height(1)

, m\_rightChild(right)

, m\_leftChild(left) { if(key) std::strncpy(m\_key, key, KeyLen); }

~Node() { delete[] m\_key; }

int balanceFactor() const

{

int leftHeight = m\_leftChild ? m\_leftChild->m\_height : 0;

int rightHeight = m\_rightChild ? m\_rightChild->m\_height : 0;

return rightHeight - leftHeight;

}

void fixHeight()

{

int leftHeight = m\_leftChild ? m\_leftChild->m\_height : 0;

int rightHeight = m\_rightChild ? m\_rightChild->m\_height : 0;

m\_height = (leftHeight > rightHeight ? leftHeight : rightHeight) + 1;

}

Node \*rotateLeft()

{

Node \*rotNode = m\_rightChild;

m\_rightChild = rotNode->m\_leftChild;

rotNode->m\_leftChild = this;

fixHeight();

rotNode->fixHeight();

return rotNode;

}

Node \*rotateRight()

{

Node \*rotNode = m\_leftChild;

m\_leftChild = rotNode->m\_rightChild;

rotNode->m\_rightChild = this;

fixHeight();

rotNode->fixHeight();

return rotNode;

}

Node \*balance()

{

fixHeight();

if(balanceFactor() == 2) {

if(m\_rightChild->balanceFactor() < 0)

m\_rightChild = m\_rightChild->rotateRight();

return rotateLeft();

}

else if(balanceFactor() == -2) {

if(m\_leftChild->balanceFactor() > 0)

m\_leftChild = m\_leftChild->rotateLeft();

return rotateRight();

}

return this;

}

Node \*findMinNode()

{

return m\_leftChild ? m\_leftChild->findMinNode() : this;

};

Node \*removeMinNode()

{

if(!m\_leftChild)

return m\_rightChild;

m\_leftChild = m\_leftChild->removeMinNode();

return balance();

}

Node \*insertNode(const char \*key, const Val &val)

{

if(std::strncmp(key, m\_key, KeyLen) < 0) {

if(m\_leftChild)

m\_leftChild = m\_leftChild->insertNode(key, val);

else

m\_leftChild = new Node(val, key);

}

else if(std::strncmp(key, m\_key, KeyLen) > 0) {

if(m\_rightChild)

m\_rightChild = m\_rightChild->insertNode(key, val);

else

m\_rightChild = new Node(val, key);

}

return balance();

}

Node \*eraceNode(const char \*key)

{

if(std::strncmp(key, m\_key, KeyLen) < 0) {

if(m\_leftChild != nullptr)

m\_leftChild = m\_leftChild->eraceNode(key);

// else

// return node;

}

else if(std::strncmp(key, m\_key, KeyLen) > 0) {

if(m\_rightChild != nullptr)

m\_rightChild = m\_rightChild->eraceNode(key);

// else

// return node;

}

else {

Node \*leftNode = m\_leftChild;

Node \*rightNode = m\_rightChild;

delete this;

if(!rightNode) return leftNode;

Node \*minNode = rightNode->findMinNode();

minNode->m\_rightChild = rightNode->removeMinNode();

minNode->m\_leftChild = leftNode;

return minNode->balance();

}

return balance();

}

void storing(LinkList<Val> &storage)

{

if(m\_leftChild)

m\_leftChild->storing(storage);

if(m\_rightChild)

m\_rightChild->storing(storage);

storage.append(m\_data);

}

void storing(const Val &val, cmpFunc predicat, LinkList<Val> &storage)

{

if(m\_leftChild)

m\_leftChild->storing(val, predicat, storage);

if(m\_rightChild)

m\_rightChild->storing(val, predicat, storage);

if(predicat(val, m\_data))

storage.append(m\_data);

}

Val m\_data;

char \*m\_key;

int m\_height;

Node \*m\_rightChild;

Node \*m\_leftChild;

};

public:

Tree() : m\_rootNode(nullptr), m\_size(0) {}

~Tree() { clear(); };

const Val &operator[](const char \*key) const;

Val &operator[](const char \*key);

Val value(const char \*key) const;

LinkList<Val> values() const;

bool hasKey(const char \*key) const;

void insert(const char \*key, const Val &val);

void erase(const char \*key);

void clear();

LinkList<Val> find(const Val &searchVal, cmpFunc predicat);

int size() const { return m\_size; }

private:

Node \*m\_rootNode;

int m\_size;

};

template<typename Val, int KeyLen>

Val &Tree<Val, KeyLen>::operator[](const char \*key)

{

Node \*searchNode = m\_rootNode;

while(searchNode && std::strncmp(key, searchNode->m\_key, KeyLen) != 0) {

if(std::strncmp(key, searchNode->m\_key, KeyLen) > 0) {

if(!searchNode->m\_rightChild)

searchNode = nullptr;

else

searchNode = searchNode->m\_rightChild;

}

else {

if(!searchNode->m\_leftChild)

searchNode = nullptr;

else

searchNode = searchNode->m\_leftChild;

}

}

if(searchNode)

return searchNode->m\_data;

else {

assert(searchNode);

return m\_rootNode->m\_data;

}

}

template<typename Val, int KeyLen>

const Val &Tree<Val, KeyLen>::operator[](const char \*key) const

{

Node \*searchNode = m\_rootNode;

while(searchNode && std::strncmp(key, searchNode->m\_key, KeyLen) != 0) {

if(std::strncmp(key, searchNode->m\_key, KeyLen) > 0) {

if(!searchNode->m\_rightChild)

searchNode = nullptr;

else

searchNode = searchNode->m\_rightChild;

}

else {

if(!searchNode->m\_leftChild)

searchNode = nullptr;

else

searchNode = searchNode->m\_leftChild;

}

}

if(searchNode)

return searchNode->m\_data;

else {

assert(searchNode);

return m\_rootNode->m\_data;

}

}

template<typename Val, int KeyLen>

Val Tree<Val, KeyLen>::value(const char \*key) const

{

if(!m\_rootNode)

return Val();

Node \*searchNode = m\_rootNode;

while(searchNode) {

if(std::strncmp(key, searchNode->m\_key, KeyLen) > 0)

searchNode = searchNode->m\_rightChild;

else if(std::strncmp(key, searchNode->m\_key, KeyLen) < 0)

searchNode = searchNode->m\_leftChild;

else

return searchNode->m\_data;

}

assert(searchNode);

return Val();

}

template<typename Val, int KeyLen>

LinkList<Val> Tree<Val, KeyLen>::values() const

{

LinkList<Val> storage;

if(!m\_rootNode)

return storage;

m\_rootNode->storing(storage);

return storage;

}

template<typename Val, int KeyLen>

bool Tree<Val, KeyLen>::hasKey(const char \*key) const

{

if(!m\_rootNode)

return false;

Node \*searchNode = m\_rootNode;

while(searchNode) {

if(std::strncmp(key, searchNode->m\_key, KeyLen) > 0)

searchNode = searchNode->m\_rightChild;

else if(std::strncmp(key, searchNode->m\_key, KeyLen) < 0)

searchNode = searchNode->m\_leftChild;

else

return true;

}

return false;

}

template<typename Val, int KeyLen>

void Tree<Val, KeyLen>::insert(const char \*key, const Val &val)

{

++m\_size;

if(!m\_rootNode) {

m\_rootNode = new Node(val, key);

return;

}

m\_rootNode = m\_rootNode->insertNode(key, val);

}

template<typename Val, int KeyLen>

void Tree<Val, KeyLen>::erase(const char \*key)

{

--m\_size;

if(!m\_rootNode)

return;

m\_rootNode = m\_rootNode->eraceNode(key);

}

template<typename Val, int KeyLen>

void Tree<Val, KeyLen>::clear()

{

while (m\_rootNode != nullptr) {

erase(m\_rootNode->m\_key);

}

}

template<typename Val, int KeyLen>

LinkList<Val> Tree<Val, KeyLen>::find(const Val &searchVal, cmpFunc predicat)

{

LinkList<Val> storage;

if(!m\_rootNode)

return storage;

m\_rootNode->storing(searchVal, predicat, storage);

return storage;

}

#endif // SEARCHTREE\_H

# Приложение 2

Исходный код разработанного приложения

Файл ConsoleApplication.h

#ifndef COREAPPLICATION\_H

#define COREAPPLICATION\_H

class **AbstractItemModel**;

class **CarsViewModel**;

class **ClientsViewModel**;

class **RentInfoViewModel**;

class **CarsModel**;

class **ClientsModel**;

class **RentInfoModel**;

class **ListView**;

class **ListDelegate**;

class **ConsoleApplication**

{

enum **CurrentModelType** {

NoModel,

CarsModelType,

ClientsModelType,

RentInfoModelType

};

enum **UserCommand** {

Exit = 0,

SelectNext,

SelectPrevious,

Unselect,

ShowHint,

HideHint,

HideTables,

ShowAllCars,

AppendCar,

RemoveCar,

ClearCars,

FilterCar,

FindCar,

SendToRepair,

ReturnFromRepair,

ShowAllClients,

AppendClient,

RemoveClient,

ClearClients,

FilterClients,

FindClient,

ShowRentInfo,

IssueCar,

ReturnCar,

ClearAllFilters

};

public:

**ConsoleApplication**();

~**ConsoleApplication**();

void **load**(const char \*carFile, const char \*clientFile, const char \*rentFile);

void **exec**();

private:

void **switchCommand**(int command);

void **showHintPanel**();

void **showMainPanel**();

void **showNavigationPanel**();

void **showModel**(CurrentModelType type);

void **appendCar**();

void **appendClient**();

bool **removeCar**();

bool **removeClient**();

void **findCar**();

void **findClient**();

void **filterCar**();

void **filterClient**();

void **clearCars**();

void **clearClients**();

void **setCarAvailable**(bool available);

void **issueCar**();

void **returnCar**();

void **clearFilters**();

private:

void **update**();

bool **checkClientLicense**(const char \*key);

bool **checkCarNumber**(const char \*key);

bool **getUserInput**(const char \*title, int titleMaxLen, char \*storage, int maxStoreLen);

bool **getUserInput**(const char \*title, int titleMaxLen, int &storage);

void **showUserOutput**(const char \*title, int titleMaxLen, const char \*data);

void **showUserOutput**(const char \*title, int titleMaxLen, int data);

void **showEscapeMsg**(const char \*msg);

bool **carIsRented**(const char \*number);

bool **clientHasCar**(const char \*number);

AbstractItemModel \***modelByType**(CurrentModelType type);

private:

CarsModel \*m\_carsModel;

ClientsModel \*m\_clientsModel;

RentInfoModel \*m\_rentInfoModel;

CarsViewModel \*m\_carsViewModel;

ClientsViewModel \*m\_clientsViewModel;

RentInfoViewModel \*m\_rentInfoViewModel;

ListView \*m\_listView;

ListDelegate \*m\_listDelegate;

CurrentModelType m\_currentModelType;

bool m\_userInputState;

bool m\_showHintPanel;

bool m\_exit;

};

#endif // COREAPPLICATION\_H

Файл сonsoleApplication.cpp

#include "consoleapplication.h"

#include "models/viewmodels/abstractitemmodel.h"

#include "models/viewmodels/clientsviewmodel.h"

#include "models/viewmodels/carsviewmodel.h"

#include "models/viewmodels/rentinfoviewmodel.h"

#include "models/datamodels/carsmodel.h"

#include "models/datamodels/clientsmodel.h"

#include "models/datamodels/rentinfomodel.h"

#include "views/listview.h"

#include "delegates/listdelegate.h"

#include <limits>

#include <iomanip>

#include <iostream>

#include <fstream>

ConsoleApplication::ConsoleApplication()

: m\_carsModel(new CarsModel)

, m\_clientsModel(new ClientsModel)

, m\_rentInfoModel(new RentInfoModel)

, m\_carsViewModel(new CarsViewModel)

, m\_clientsViewModel(new ClientsViewModel)

, m\_rentInfoViewModel(new RentInfoViewModel)

, m\_listView(new ListView)

, m\_listDelegate(new ListDelegate)

, m\_currentModelType(NoModel)

, m\_userInputState(false)

, m\_showHintPanel(true)

, m\_exit(false)

{

m\_listView->setDelegate(m\_listDelegate);

m\_carsViewModel->initHeader();

m\_clientsViewModel->initHeader();

m\_rentInfoViewModel->initHeader();

m\_clientsModel->setViewModel(m\_clientsViewModel);

m\_carsModel->setViewModel(m\_carsViewModel);

m\_rentInfoModel->setViewModel(m\_rentInfoViewModel);

}

ConsoleApplication::~ConsoleApplication()

{

delete m\_listDelegate;

delete m\_listView;

delete m\_rentInfoViewModel;

delete m\_clientsViewModel;

delete m\_carsViewModel;

delete m\_rentInfoModel;

delete m\_clientsModel;

delete m\_carsModel;

}

void ConsoleApplication::load(const char \*carFile, const char \*clientFile, const char \*rentFile)

{

std::fstream fs;

fs.open(carFile, std::ios\_base::in);

if(fs.is\_open()) {

char num[app\_global::car::numberMaxLen];

char brand[app\_global::car::brandMaxLen];

char color[app\_global::car::colorMaxLen];

int year = -1;

bool available = false;

while (!fs.eof()) {

fs.getline(num, app\_global::car::numberMaxLen, '|');

fs.getline(brand, app\_global::car::brandMaxLen, '|');

fs.getline(color, app\_global::car::colorMaxLen, '|');

fs >> year >> available;

fs.ignore(1, '\n');

if(!checkCarNumber(num))

continue;

Car newCar(num, brand, color, year, available);

m\_carsModel->setData(newCar);

}

}

fs.close();

fs.clear();

fs.open(clientFile, std::ios\_base::in);

if(fs.is\_open()) {

char license[app\_global::client::licenseMaxLen];

char name[app\_global::client::nameMaxLen];

char passport[app\_global::client::passportMaxLen];

char address[app\_global::client::addressMaxLen];

while (!fs.eof()) {

fs.getline(license, app\_global::client::licenseMaxLen, '|');

fs.getline(name, app\_global::client::nameMaxLen, '|');

fs.getline(passport, app\_global::client::passportMaxLen, '|');

fs.getline(address, app\_global::client::addressMaxLen, '\n');

if(!checkClientLicense(license))

continue;

Client newClient(license, name, passport, address);

m\_clientsModel->setData(newClient);

}

}

fs.close();

fs.clear();

fs.open(rentFile, std::ios\_base::in);

if(fs.is\_open()) {

char carData[app\_global::car::numberMaxLen];

char clientData[app\_global::client::licenseMaxLen];

char issueData[app\_global::dateLen];

char returnDate[app\_global::dateLen];

while (!fs.eof()) {

fs.getline(carData, app\_global::car::numberMaxLen, '|');

fs.getline(clientData, app\_global::client::licenseMaxLen, '|');

fs.getline(issueData, app\_global::dateLen, '|');

fs.getline(returnDate, app\_global::dateLen, '\n');

if(!checkClientLicense(clientData) || !checkCarNumber(carData))

continue;

RentInfo rent(clientData, carData, issueData, returnDate);

m\_rentInfoModel->setData(rent);

}

}

fs.close();

fs.clear();

}

void ConsoleApplication::exec()

{

setlocale(LC\_ALL, "");

while (!m\_exit) {

update();

int command = -1;

(std::cin >> command).get();

if(!std::cin) {

std::cin.clear();

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

continue;

}

switchCommand(command);

}

}

void ConsoleApplication::switchCommand(int command)

{

switch(command) {

case Exit:

m\_exit = true;

break;

case SelectNext:

m\_listView->selectNext();

break;

case SelectPrevious:

m\_listView->selectPrevious();

break;

case Unselect:

m\_listView->clearSelection();

break;

case ShowHint:

m\_showHintPanel = true;

break;

case HideHint:

m\_showHintPanel = false;

break;

case HideTables:

showModel(NoModel);

break;

case ShowAllCars:

showModel(CarsModelType);

break;

case AppendCar:

showModel(CarsModelType);

appendCar();

break;

case RemoveCar:

showModel(CarsModelType);

removeCar();

break;

case ClearCars:

showModel(CarsModelType);

clearCars();

break;

case FilterCar:

showModel(CarsModelType);

filterCar();

break;

case FindCar:

showModel(CarsModelType);

findCar();

break;

case SendToRepair:

showModel(CarsModelType);

setCarAvailable(false);

break;

case ReturnFromRepair:

showModel(CarsModelType);

setCarAvailable(true);

break;

case ShowAllClients:

showModel(ClientsModelType);

break;

case AppendClient:

showModel(ClientsModelType);

appendClient();

break;

case RemoveClient:

showModel(ClientsModelType);

removeClient();

break;

case ClearClients:

showModel(ClientsModelType);

clearClients();

break;

case FilterClients:

showModel(ClientsModelType);

filterClient();

break;

case FindClient:

showModel(ClientsModelType);

findClient();

break;

case ShowRentInfo:

showModel(RentInfoModelType);

break;

case IssueCar:

showModel(RentInfoModelType);

issueCar();

break;

case ReturnCar:

showModel(RentInfoModelType);

returnCar();

break;

case ClearAllFilters:

clearFilters();

break;

default:

break;

}

}

void ConsoleApplication::showHintPanel()

{

showUserOutput("\rПоказать все имеющиеся автомобили ", 61, ShowAllCars);

showUserOutput("Добавить новый автомобиль ", 60, AppendCar);

if(m\_currentModelType == CarsModelType) {

showUserOutput("Удалить сведения об автомобиле ", 60, RemoveCar);

showUserOutput("Очистить данные об автомобилях ", 60, ClearCars);

showUserOutput("Найти автомобить по гос. номеру ", 60, FindCar);

showUserOutput("Отправить автомобиль в ремонт ", 60, SendToRepair);

showUserOutput("Вернуть автомобиль из ремента ", 60, ReturnFromRepair);

showUserOutput("Добавить фильтр ", 60, FilterCar);

}

std::cout << std::endl;

showUserOutput("Показать всех зарегистрированных клиентов ", 60, ShowAllClients);

showUserOutput("Зарегистрировать нового клиента ", 60, AppendClient);

if(m\_currentModelType == ClientsModelType) {

showUserOutput("Снять клиента с обслуживания ", 60, RemoveClient);

showUserOutput("Очистить данные о клиентах ", 60, ClearClients);

showUserOutput("Найти клиента по номеру водительского удостоверения ", 60, FindClient);

showUserOutput("Добавить фильтр ", 60, FilterClients);

}

std::cout << std::endl;

showUserOutput("Показать информацию об аренде ", 60, ShowRentInfo);

showUserOutput("Выдать автомобить в аренду ", 60, IssueCar);

if(m\_currentModelType == RentInfoModelType)

showUserOutput("Вернуть автомобить из аренты ", 60, ReturnCar);

std::cout << std::endl;

if(m\_currentModelType != NoModel)

showUserOutput("Сбросить все фильтры ", 60, ClearAllFilters);

std::cout << std::endl;

}

void ConsoleApplication::showMainPanel()

{

if(!m\_showHintPanel)

showUserOutput("\rПоказать комманды ", 61, ShowHint);

else

showUserOutput("Скрыть комманды ", 60, HideHint);

if(m\_currentModelType != NoModel)

showUserOutput("Скрыть таблицу ", 60, HideTables);

showUserOutput("Выход ", 60, Exit);

std::cout << std::endl;

std::cout << "\ruser:~> ";

}

void ConsoleApplication::showNavigationPanel()

{

if(m\_currentModelType == NoModel)

return;

showUserOutput("\rПредыдущая строка ", 61, SelectNext);

showUserOutput("Следующая строка ", 60, SelectPrevious);

showUserOutput("Снять выделение ", 60, Unselect);

std::cout << std::endl;

}

void ConsoleApplication::showModel(CurrentModelType type)

{

m\_currentModelType = type;

AbstractItemModel \*model = modelByType(type);

if(!model) {

m\_listView->setModel(nullptr);

return;

}

if(m\_listView->model() != model) {

m\_listDelegate->setFieldWidth(app\_global::numberOfLetters(model->headerData().data()));

m\_listView->setModel(model);

}

}

void ConsoleApplication::appendCar()

{

m\_userInputState = true;

update();

char numb[app\_global::car::numberMaxLen];

char brand[app\_global::car::brandMaxLen];

char color[app\_global::car::colorMaxLen];

int year = 0;

std::cout << std::endl;

if(!getUserInput("Номер автомобиля: ", 60, numb, app\_global::car::numberMaxLen)) {

m\_userInputState = false;

return;

}

while (!checkCarNumber(numb)) {

std::cout << "Неверный формат. Повторите ввод\n";

if(!getUserInput("Номер автомобиля: ", 60, numb, app\_global::car::numberMaxLen)) {

m\_userInputState = false;

return;

}

}

if(!getUserInput("Марка автомобиля: ", 60, brand, app\_global::car::brandMaxLen)) {

m\_userInputState = false;

return;

}

if(!getUserInput("Цвет автомобиля: ", 60, color, app\_global::car::colorMaxLen)) {

m\_userInputState = false;

return;

}

getUserInput("Год выпуска автомобиля: ", 60, year);

Car newCar(numb, brand, color, year);

m\_carsModel->setData(newCar);

m\_userInputState = false;

}

void ConsoleApplication::appendClient()

{

m\_userInputState = true;

update();

char license[app\_global::client::licenseMaxLen];

char name[app\_global::client::nameMaxLen];

char passport[app\_global::client::passportMaxLen];

char address[app\_global::client::addressMaxLen];

std::cout << std::endl;

if(!getUserInput("Номер водительского удостоверения: ", 60, license, app\_global::client::licenseMaxLen)) {

m\_userInputState = false;

return;

}

while(!checkClientLicense(license)) {

std::cout << "Неверный формат. Повторите ввод\n";

if(!getUserInput("Номер водительского удостоверения: ", 60, license, app\_global::client::licenseMaxLen)) {

m\_userInputState = false;

return;

}

}

if(!getUserInput("ФИО клиента: ", 60, name, app\_global::client::nameMaxLen)) {

m\_userInputState = false;

return;

}

if(!getUserInput("Паспортные данные: ", 60, passport, app\_global::client::passportMaxLen)) {

m\_userInputState = false;

return;

}

if(!getUserInput("Адрес: ", 60, address, app\_global::client::addressMaxLen)) {

m\_userInputState = false;

return;

}

Client newClient(license, name, passport, address);

m\_clientsModel->setData(newClient);

m\_userInputState = false;

}

bool ConsoleApplication::removeCar()

{

if(m\_listView->selectedItems().isEmpty())

return true;

m\_userInputState = true;

update();

int rowForDel = m\_listView->selectedItems().begin()->row();

CStringData deleteData = m\_carsViewModel->data(ModelIndex(rowForDel, CarsModel::Number));

if(carIsRented(deleteData.data())) {

showEscapeMsg("Невозможно снять с регистрации выданный в аренду автомобиль");

m\_userInputState = false;

return false;

}

m\_carsModel->removeData(deleteData);

if(rowForDel > m\_carsViewModel->rowCount())

m\_listView->clearSelection();

m\_userInputState = false;

return true;

}

bool ConsoleApplication::removeClient()

{

if(m\_listView->selectedItems().isEmpty())

return true;

m\_userInputState = true;

update();

int rowForDel = m\_listView->selectedItems().begin()->row();

CStringData deleteData = m\_clientsViewModel->data(ModelIndex(rowForDel, ClientsModel::License));

if(clientHasCar(deleteData.data())) {

showEscapeMsg("Невозможно снять с регистрации клиента имеющего арендованные автомобили");

m\_userInputState = false;

return false;

}

m\_clientsModel->removeData(deleteData);

if(rowForDel > m\_clientsViewModel->rowCount())

m\_listView->clearSelection();

m\_userInputState = false;

return true;

}

void ConsoleApplication::findCar()

{

m\_userInputState = true;

update();

char key[app\_global::client::licenseMaxLen] = "\0";

std::cout << "Введите гос. номер автомобиля для поиска\n";

if(!getUserInput("Гос. номер автомобиля: ", 60, key, app\_global::car::numberMaxLen)) {

m\_userInputState = false;

return;

}

if(!checkCarNumber(key)) {

std::cout << "Неверный формат. Повторите ввод: \n";

if(!getUserInput("Гос. номер автомобиля: ", 60, key, app\_global::car::numberMaxLen)) {

m\_userInputState = false;

return;

}

}

std::cout << std::endl;

Car foundCar = m\_carsModel->getData(CStringData(key, app\_global::car::numberMaxLen));

if(!checkCarNumber(foundCar.number()))

std::cout << "Автомобиля с таким гос. номером не зарегестрирован\n";

else {

std::cout << "Найден автомобиль\n";

showUserOutput("Гос. номер автомобиля ", 60, foundCar.number());

showUserOutput("Марка автомобиля ", 60, foundCar.brand());

showUserOutput("Цвет автомобиля ", 60, foundCar.color());

showUserOutput("Год выпуска автомобиля ", 60, foundCar.year());

std::cout << std::endl;

if(!carIsRented(foundCar.number())) {

showEscapeMsg("Автомобиль не арендован");

m\_userInputState = false;

return;

}

LinkList<RentInfo> rents = m\_rentInfoModel->getData(CStringData(foundCar.number(),

app\_global::car::numberMaxLen),

RentInfoModel::CarRole);

for(LinkList<RentInfo>::iterator iter = rents.begin(); iter != rents.end(); ++iter) {

if(std::strlen(iter->returnDate()) != 0 || !checkCarNumber(foundCar.number()))

continue;

Client foundClient = m\_clientsModel->getData(CStringData(iter->clientData(),

app\_global::client::licenseMaxLen));

if(!checkClientLicense(foundClient.license()))

continue;

showUserOutput("Номер водительского удостоверения клиента ", 60, foundClient.license());

showUserOutput("ФИО ", 60, foundClient.name());

}

}

showEscapeMsg("\n");

m\_userInputState = false;

}

void ConsoleApplication::findClient()

{

m\_userInputState = true;

update();

char key[app\_global::client::licenseMaxLen] = "\0";

std::cout << "Введите номер водительского удостоверения для поиска\n";

if(!getUserInput("Номер водительского удостоверения: ", 60, key, app\_global::client::licenseMaxLen)) {

m\_userInputState = false;

return;

}

if(!checkClientLicense(key)) {

std::cout << "Неверный формат. Повторите ввод: \n";

if(!getUserInput("Номер водительского удостоверения: ", 60, key, app\_global::client::licenseMaxLen)) {

m\_userInputState = false;

return;

}

}

std::cout << std::endl;

Client foundClient = m\_clientsModel->getData(CStringData(key, app\_global::client::licenseMaxLen));

if(!checkClientLicense(foundClient.license()))

std::cout << "Клиент с таким номером водительского удостоверения не зарегистирован\n";

else {

std::cout << "Найден клиент\n";

showUserOutput("Номер водительского удостоверения ", 60, foundClient.license());

showUserOutput("ФИО ", 60, foundClient.name());

showUserOutput("Паспортные данные ", 60, foundClient.passport());

showUserOutput("Адрес ", 60, foundClient.address());

std::cout << std::endl;

if(!clientHasCar(foundClient.license())) {

showEscapeMsg("Клиент не имеет арендованых автомобилей");

m\_userInputState = false;

return;

}

LinkList<RentInfo> rents = m\_rentInfoModel->getData(CStringData(foundClient.license(),

app\_global::client::licenseMaxLen),

RentInfoModel::ClientRole);

std::cout << "Клиент имеет в аренде автомобили\n";

for(LinkList<RentInfo>::iterator iter = rents.begin(); iter != rents.end(); ++iter) {

if(std::strlen(iter->returnDate()) != 0)

continue;

Car foundCar = m\_carsModel->getData(CStringData(iter->carData(),

app\_global::car::numberMaxLen));

if(checkCarNumber(foundCar.number()))

showUserOutput("Гос. номер ", 60, foundCar.number());

}

}

showEscapeMsg("\n");

m\_userInputState = false;

}

void ConsoleApplication::filterCar()

{

m\_userInputState = true;

update();

char key[app\_global::car::brandMaxLen] = "\0";

std::cout << "Введите матрку автомобиля для фильткации результатов\n";

if(!getUserInput("Марка автомобиля: ", 60, key, app\_global::car::brandMaxLen)) {

m\_userInputState = false;

return;

}

std::cout << std::endl;

m\_carsModel->filter(CStringData(key, app\_global::car::brandMaxLen), CarsModel::Brand);

m\_userInputState = false;

}

void ConsoleApplication::filterClient()

{

m\_userInputState = true;

update();

int role = ClientsModel::UndefinedRole;

std::cout << std::endl;

showUserOutput("ФИО ", 60, ClientsModel::Name);

showUserOutput("Адрес ", 60, ClientsModel::Address);

std::cout << std::endl;

getUserInput("Веберете колонку для фильтрации ", 60, role);

update();

char \*key = nullptr;

int keyLen = 0;

switch(role) {

case ClientsModel::Name:

keyLen = app\_global::client::nameMaxLen;

key = new char[keyLen];

std::cout << "Введите ФИО клиента для фильткации результатов\n";

if(!getUserInput("ФИО клиента: ", 60, key, keyLen)) {

m\_userInputState = false;

return;

}

break;

case ClientsModel::Address:

keyLen = app\_global::client::addressMaxLen;

key = new char[keyLen];

std::cout << "Введите адрес клиента для фильткации результатов\n";

if(!getUserInput("Адрес клиента: ", 60, key, keyLen)) {

m\_userInputState = false;

return;

}

break;

default:

m\_userInputState = false;

return;;

}

m\_clientsModel->filter(CStringData(key, keyLen), role);

delete [] key;

m\_userInputState = false;

}

void ConsoleApplication::clearCars()

{

m\_userInputState = true;

update();

m\_listView->clearSelection();

m\_listView->selectNext();

while(m\_listView->selectedItems().begin()->row() < m\_carsViewModel->rowCount()) {

if(!removeCar())

m\_listView->selectNext();

update();

if(m\_listView->selectedItems().isEmpty())

break;

}

m\_userInputState = false;

}

void ConsoleApplication::clearClients()

{

m\_userInputState = true;

update();

m\_listView->clearSelection();

m\_listView->selectNext();

while(m\_listView->selectedItems().begin()->row() < m\_carsViewModel->rowCount()) {

if(!removeClient())

m\_listView->selectNext();

update();

if(m\_listView->selectedItems().isEmpty())

break;

}

m\_userInputState = false;

}

void ConsoleApplication::setCarAvailable(bool available)

{

if(m\_listView->selectedItems().isEmpty())

return;

m\_userInputState = true;

update();

int selectedRow = m\_listView->selectedItems().begin()->row();

CStringData number = m\_carsViewModel->data(ModelIndex(selectedRow, CarsModel::Number));

bool isCarAvailable = m\_carsModel->getData(number).available();

if(!m\_carsModel->hasData(number))

showEscapeMsg("Автомобиль с таким номером не зарегистрирован");

else if(carIsRented(number.data()))

showEscapeMsg("Данный автомобиль арендован");

else if(available && isCarAvailable)

showEscapeMsg("Нозможно вернуть из ремонта автомобиль доступный для аренды");

else if(!available && !isCarAvailable)

showEscapeMsg("Нозможно отправить в ремонт автомобиль уже находящийся в ремонте");

else

m\_carsModel->setAvailable(number, available);

m\_userInputState = false;

}

void ConsoleApplication::issueCar()

{

m\_userInputState = true;

char carKey[app\_global::car::numberMaxLen] = "\0";

char clientKey[app\_global::client::licenseMaxLen] = "\0";

char issueDate[app\_global::client::licenseMaxLen] = "\0";

showModel(CarsModelType);

update();

if(!getUserInput("Гос. номер автомобиля: ", 60, carKey, app\_global::car::numberMaxLen)) {

m\_userInputState = false;

return;

}

while(!m\_carsModel->hasData(CStringData(carKey, app\_global::car::numberMaxLen))) {

std::cout << "Автомобиль с таким гос номером не зарегистрирован. Повторите ввод\n";

if(!getUserInput("Гос. номер автомобиля: ", 60, carKey, app\_global::car::numberMaxLen)) {

m\_userInputState = false;

return;

}

}

if(!m\_carsModel->getData(CStringData(carKey, app\_global::car::numberMaxLen)).available()) {

update();

showEscapeMsg("Данные автомобиль не доступен для аренды");

m\_userInputState = false;

return;

}

showModel(ClientsModelType);

update();

if(!getUserInput("Номер водительского удостоверения: ", 60, clientKey, app\_global::client::licenseMaxLen)) {

m\_userInputState = false;

return;

}

while(!m\_clientsModel->hasData(CStringData(clientKey, app\_global::client::licenseMaxLen))){

std::cout << "Клиент с таким номером водительского удостоверения не зарегистрирован. Повторите ввод\n";

if(!getUserInput("Номер водительского удостоверения: ", 60, clientKey, app\_global::client::licenseMaxLen)) {

m\_userInputState = false;

return;

}

}

showModel(RentInfoModelType);

update();

if(!getUserInput("Дата выдачи: ", 60, issueDate, app\_global::dateLen)) {

m\_userInputState = false;

return;

}

RentInfo rent(clientKey, carKey, issueDate);

m\_rentInfoModel->setData(rent);

m\_carsModel->setAvailable(CStringData(carKey, app\_global::car::numberMaxLen), false);

m\_userInputState = false;

}

void ConsoleApplication::returnCar()

{

m\_userInputState = true;

update();

char carKey[app\_global::car::numberMaxLen] = "\0";

char returnDate[app\_global::dateLen] = "\0";

if(!getUserInput("Гос. номер автомобиля: ", 60, carKey, app\_global::car::numberMaxLen)) {

m\_userInputState = false;

return;

}

while(!m\_carsModel->hasData(CStringData(carKey, app\_global::car::numberMaxLen))

|| !m\_rentInfoModel->hasData(CStringData(carKey, app\_global::car::numberMaxLen), RentInfoModel::CarRole)) {

std::cout << "Автомобиль с таким гос номером не выдавался в аренду. Повторите ввод\n";

if(!getUserInput("Гос. номер автомобиля: ", 60, carKey, app\_global::car::numberMaxLen)) {

m\_userInputState = false;

return;

}

}

if(!getUserInput("Дата возврата автомобиля: ", 60, returnDate, app\_global::dateLen)) {

m\_userInputState = false;

return;

}

LinkList<RentInfo> rents = m\_rentInfoModel->getData(CStringData(carKey, app\_global::car::numberMaxLen), RentInfoModel::CarRole);

for(LinkList<RentInfo>::iterator iter = rents.begin(); iter != rents.end(); ++iter) {

if(std::strlen(iter->returnDate()) != 0)

continue;

m\_carsModel->setAvailable(CStringData(iter->carData(), app\_global::car::numberMaxLen), true);

iter->setReturnDate(returnDate);

m\_rentInfoModel->removeData(CStringData(carKey, app\_global::car::numberMaxLen));

m\_rentInfoModel->setData(\*iter);

}

m\_userInputState = false;

}

void ConsoleApplication::clearFilters()

{

switch (m\_currentModelType) {

case CarsModelType:

m\_carsModel->filter(CStringData(), CarsModel::UndefinedRole);

break;

case ClientsModelType:

m\_clientsModel->filter(CStringData(), ClientsModel::UndefinedRole);

break;

default:

break;

}

}

void ConsoleApplication::update()

{

app\_global::clearConsole();

if(m\_currentModelType != NoModel) {

m\_listView->update();

std::cout << std::endl;

}

if(!m\_userInputState) {

if(m\_showHintPanel)

showHintPanel();

showNavigationPanel();

showMainPanel();

}

}

bool ConsoleApplication::checkClientLicense(const char \*key)

{

if(app\_global::numberOfLetters(key) != 12)

return false;

const char \*keyPtr = key;

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

switch (i) {

case 3:

case 4:{

char simbol[2] = "\0";

simbol[0] = \*keyPtr++;

simbol[1] = \*keyPtr++;

if((simbol[0] == "А"[0] && simbol[1] == "А"[1])

|| (simbol[0] == "В"[0] && simbol[1] == "В"[1])

|| (simbol[0] == "Е"[0] && simbol[1] == "Е"[1])

|| (simbol[0] == "К"[0] && simbol[1] == "К"[1])

|| (simbol[0] == "М"[0] && simbol[1] == "М"[1])

|| (simbol[0] == "Н"[0] && simbol[1] == "Н"[1])

|| (simbol[0] == "О"[0] && simbol[1] == "О"[1])

|| (simbol[0] == "Р"[0] && simbol[1] == "Р"[1])

|| (simbol[0] == "С"[0] && simbol[1] == "С"[1])

|| (simbol[0] == "Т"[0] && simbol[1] == "Т"[1])

|| (simbol[0] == "У"[0] && simbol[1] == "У"[1])

|| (simbol[0] == "Х"[0] && simbol[1] == "Х"[1]))

break;

return false;

}

case 0:

case 1:

case 6:

case 7:

case 8:

case 9:

case 10:

case 11:{

int simbolCode = static\_cast<int>(\*keyPtr++);

if(simbolCode < 48 || simbolCode > 57)

return false;

break;

}

case 2:

case 5:

if(\*keyPtr++ != ' ')

return false;

break;

default:

std::cout << "wrong char number\n";

return false;

}

}

return true;

}

bool ConsoleApplication::checkCarNumber(const char \*key)

{

if(app\_global::numberOfLetters(key) != 9)

return false;

const char \*keyPtr = key;

for(int i = 0; i < 9; ++i) {

switch (i) {

case 0:

case 4:

case 5: {

char simbol[2] = "\0";

simbol[0] = \*keyPtr++;

simbol[1] = \*keyPtr++;

if((simbol[0] == "А"[0] && simbol[1] == "А"[1])

|| (simbol[0] == "В"[0] && simbol[1] == "В"[1])

|| (simbol[0] == "Е"[0] && simbol[1] == "Е"[1])

|| (simbol[0] == "К"[0] && simbol[1] == "К"[1])

|| (simbol[0] == "М"[0] && simbol[1] == "М"[1])

|| (simbol[0] == "Н"[0] && simbol[1] == "Н"[1])

|| (simbol[0] == "О"[0] && simbol[1] == "О"[1])

|| (simbol[0] == "Р"[0] && simbol[1] == "Р"[1])

|| (simbol[0] == "С"[0] && simbol[1] == "С"[1])

|| (simbol[0] == "Т"[0] && simbol[1] == "Т"[1])

|| (simbol[0] == "У"[0] && simbol[1] == "У"[1])

|| (simbol[0] == "Х"[0] && simbol[1] == "Х"[1]))

break;

return false;

}

case 1:

case 2:

case 3:

case 7:

case 8: {

int simbolCode = static\_cast<int>(\*keyPtr++);

if(simbolCode < 48 || simbolCode > 57)

return false;

break;

}

case 6:

if(\*keyPtr++ != '-')

return false;

break;

default:

std::cout << "wrong char number\n";

return false;

}

}

return true;

}

bool ConsoleApplication::getUserInput(const char \*title, int titleMaxLen, char \*storage, int maxStoreLen)

{

std::cout.setf(std::ios\_base::left, std::ios\_base::adjustfield);

std::cout << title

<< std::setw(app\_global::realFilledStringSize("(Enter для выхода) ", titleMaxLen - app\_global::numberOfLetters(title)))

<< std::setfill('\_') << "(Enter для выхода) " << " ";

std::cin.getline(storage, maxStoreLen, '\n');

if(!std::cin){

std::cin.clear();

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

}

std::cout.unsetf(std::ios\_base::adjustfield);

if(std::strlen(storage) == 0)

return false;

return true;

}

bool ConsoleApplication::getUserInput(const char \*title, int titleMaxLen, int &storage)

{

std::cout.setf(std::ios\_base::left, std::ios\_base::adjustfield);

std::cout << std::setw(app\_global::realFilledStringSize(title, titleMaxLen))

<< std::setfill('\_') << title << " ";

(std::cin >> storage).get();

while(!std::cin) {

std::cin.clear();

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

std::cout << "Введено некорректное значение. Повторите ввод (0 для выхода): ";

(std::cin >> storage).get();

if(storage == 0)

return false;

}

return true;

}

void ConsoleApplication::showUserOutput(const char \*title, int titleMaxLen, const char \*data)

{

std::cout.setf(std::ios\_base::left, std::ios\_base::adjustfield);

std::cout << std::setw(app\_global::realFilledStringSize(title, titleMaxLen))

<< std::setfill('\_') << title << ' ' << data << '\n';

std::cout.unsetf(std::ios\_base::adjustfield);

}

void ConsoleApplication::showUserOutput(const char \*title, int titleMaxLen, int data)

{

std::cout.setf(std::ios\_base::left, std::ios\_base::adjustfield);

std::cout << std::setw(app\_global::realFilledStringSize(title, titleMaxLen))

<< std::setfill('\_') << title << ' ' << data << '\n';

std::cout.unsetf(std::ios\_base::adjustfield);

}

void ConsoleApplication::showEscapeMsg(const char \*msg)

{

std::cout << msg << std::endl << std::endl << "(Enter для выхода)";

std::cin.get();

}

bool ConsoleApplication::carIsRented(const char \*number)

{

bool rented = false;

if(!checkCarNumber(number))

return rented;

if(!m\_carsModel->hasData(CStringData(number, app\_global::car::numberMaxLen)))

return rented;

if(!m\_rentInfoModel->hasData(CStringData(number, app\_global::car::numberMaxLen),

RentInfoModel::CarRole))

return rented;

Car foundCar = m\_carsModel->getData(CStringData(number, app\_global::car::numberMaxLen));

if(!foundCar.available()) {

LinkList<RentInfo> rents = m\_rentInfoModel->getData(CStringData(number, app\_global::car::numberMaxLen),

RentInfoModel::CarRole);

if(rents.isEmpty())

return rented;

for(LinkList<RentInfo>::iterator iter = rents.begin(); iter != rents.end(); ++iter) {

if(std::strlen(iter->returnDate()) == 0)

rented = true;

}

}

return rented;

}

bool ConsoleApplication::clientHasCar(const char \*number)

{

bool hasRent = false;

if(!checkClientLicense(number))

return hasRent;

if(!m\_clientsModel->hasData(CStringData(number, app\_global::client::licenseMaxLen)))

return hasRent;

if(!m\_rentInfoModel->hasData(CStringData(number, app\_global::client::licenseMaxLen),

RentInfoModel::ClientRole))

return hasRent;

LinkList<RentInfo> rents = m\_rentInfoModel->getData(CStringData(number, app\_global::client::licenseMaxLen),

RentInfoModel::ClientRole);

if(!rents.isEmpty())

hasRent = true;

return hasRent;

}

AbstractItemModel \*ConsoleApplication::modelByType(ConsoleApplication::CurrentModelType type)

{

switch (type) {

case NoModel:

return nullptr;

case CarsModelType:

return m\_carsViewModel;

case ClientsModelType:

return m\_clientsViewModel;

case RentInfoModelType:

return m\_rentInfoViewModel;

break;

default:

return nullptr;

}

}

Файл AbstractItemView.h

#ifndef ABSTRACTITEMVIEW\_H

#define ABSTRACTITEMVIEW\_H

#include "../app\_core/modelindex.h"

#include "../app\_core/linklist.h"

class AbstractItemModel;

class AbstractDelegate;

class AbstractItemView

{

public:

AbstractItemView();

virtual ~AbstractItemView();

virtual void update() = 0;

virtual void editItem(const char \*newData, int dataLen);

virtual void selectItem(int row, int column);

virtual void selectRow(int row);

virtual void selectColumn(int column);

AbstractDelegate \*delegate() const;

void setDelegate(AbstractDelegate \*delegate);

AbstractItemModel \*model() const;

void setModel(AbstractItemModel \*model);

const LinkList<ModelIndex> &selectedItems() const;

void clearSelection();

private:

AbstractItemModel \*m\_model;

AbstractDelegate \*m\_delegate;

LinkList<ModelIndex> m\_selectedItems;

};

#endif // ABSTRACTITEMVIEW\_H

Файл AbstractItemView.cpp

#include "abstractitemview.h"

#include "../models/viewmodels/abstractitemmodel.h"

#include "../delegates/abstractdelegate.h"

#include "../app\_core/cstringdata.h"

AbstractItemView::**AbstractItemView**()

: m\_model(nullptr)

, m\_delegate(nullptr)

{

}

AbstractItemView::~***AbstractItemView***()

{

}

void AbstractItemView::***editItem***(const char \*newData, int dataLen)

{

if(!m\_delegate || !m\_model)

return;

const ModelIndex &currentIndex = m\_delegate->currentIndex();

if(!currentIndex.isValid()

|| !currentIndex.isMulti())

return;

// m\_model->setData(CStringData(newData, dataLen), currentIndex);

}

void AbstractItemView::***selectItem***(int row, int column)

{

if(!m\_delegate)

return;

ModelIndex selectedIndex(row, column);

if(!selectedIndex.isValid())

return;

m\_selectedItems.append(selectedIndex);

}

void AbstractItemView::***selectRow***(int row)

{

*selectItem*(row, -1);

}

void AbstractItemView::***selectColumn***(int column)

{

*selectItem*(column, -1);

}

AbstractDelegate \*AbstractItemView::**delegate**() const

{

return m\_delegate;

}

void AbstractItemView::**setDelegate**(AbstractDelegate \*delegate)

{

if(m\_delegate == delegate)

return;

m\_delegate = delegate;

m\_delegate->setModel(*m\_model*);

}

AbstractItemModel \*AbstractItemView::**model**() const

{

return m\_model;

}

void AbstractItemView::**setModel**(AbstractItemModel \*model)

{

if(m\_model == model)

return;

m\_model = model;

m\_delegate->setModel(*m\_model*);

m\_selectedItems.clear();

}

const LinkList<ModelIndex> &AbstractItemView::**selectedItems**() const

{

return m\_selectedItems;

}

void AbstractItemView::**clearSelection**()

{

m\_selectedItems.clear();

}

Файл ListView.h

#ifndef TABLEVIEW\_H

#define TABLEVIEW\_H

#include "abstractitemview.h"

class ListView : public AbstractItemView

{

public:

ListView();

~ListView();

public:

virtual void update() override;

virtual void selectItem(int row, int column) override;

void selectNext();

void selectPrevious();

};

#endif // TABLEVIEW\_H

Файл ListView.cpp

#include "listview.h"

#include "../models/viewmodels/abstractitemmodel.h"

#include "../delegates/abstractdelegate.h"

#include <iostream>

ListView::**ListView**()

: AbstractItemView()

{

}

ListView::~***ListView***()

{

}

void ListView::***update***()

{

int rowCount = model()->*rowCount*();

int columtCount = model()->*columnCount*();

bool needBoard = false;

for(int i = 0; i < rowCount \* 2; ++i) {

if(i % 2 == 0)

needBoard = true;

else

needBoard = false;

ModelIndex index(i / 2, 0);

bool selectVertical = selectedItems().contains(index);

bool selectHorizontal = selectedItems().contains(index)

|| selectedItems().contains(ModelIndex(index.row() - 1, 0));

if(needBoard)

delegate()->*drawHorizontalBoard*(selectHorizontal);

else {

delegate()->*drawVerticalBorad*(selectVertical);

if(index.row() == 0) {

delegate()->setCurrentIndex(index);

delegate()->*drawData*();

delegate()->*drawVerticalBorad*(selectVertical);

}

else {

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

delegate()->setCurrentIndex(ModelIndex(index.row(), j));

delegate()->*drawData*();

delegate()->*drawVerticalBorad*(selectVertical);

}

}

}

delegate()->*drawNextLine*();

}

delegate()->*drawHorizontalBoard*(selectedItems().contains(ModelIndex(rowCount - 1, 0)));

delegate()->*drawNextLine*();

}

void ListView::***selectItem***(int row, int column)

{

if(!model())

return;

if(row == 0)

return;

if(!(row < model()->*rowCount*())) {

clearSelection();

return;

}

clearSelection();

AbstractItemView::selectItem(row, column);

}

void ListView::**selectNext**()

{

if(!model())

return;

if(selectedItems().isEmpty()) {

*selectItem*(1, 0);

return;

}

else if(selectedItems().size() > 1)

return;

*selectItem*(selectedItems().begin()->row() + 1, 0);

}

void ListView::**selectPrevious**()

{

if(!model())

return;

if(selectedItems().isEmpty()) {

*selectItem*(model()->*rowCount*() - 1, 0);

return;

}

else if(selectedItems().size() > 1)

return;

*selectItem*(selectedItems().begin()->row() - 1, 0);

}

Файл AbstractViewModel.h

#ifndef ABSTRACTITEMMODEL\_H

#define ABSTRACTITEMMODEL\_H

#include "../../app\_core/modelindex.h"

#include "../../app\_core/cstringdata.h"

class **AbstractItemModel**

{

public:

**AbstractItemModel**();

virtual ~***AbstractItemModel***();

virtual void ***initHeader***() = 0;

virtual CStringData ***data***(const ModelIndex &index) const = 0;

virtual int ***columnCount***() const = 0;

virtual int ***rowCount***() const = 0;

CStringData **headerData**() const;

void **setHeaderData**(const CStringData &data);

private:

CStringData m\_header;

};

#endif // ABSTRACTITEMMODEL\_H

Файл AbstractViewModel.cpp

#include "abstractitemmodel.h"

AbstractItemModel::**AbstractItemModel**()

: m\_header()

{

}

AbstractItemModel::~***AbstractItemModel***()

{

}

CStringData AbstractItemModel::**headerData**() const

{

return m\_header;

}

void AbstractItemModel::**setHeaderData**(const CStringData &data)

{

m\_header = data;

}

Файл CarViewModel.h

#ifndef CARSVIEWMODEL\_H

#define CARSVIEWMODEL\_H

#include "abstractitemmodel.h"

#include "../../app\_core/linklist.h"

#include "../../items.h"

class CarsViewModel : public AbstractItemModel

{

public:

CarsViewModel();

~CarsViewModel();

virtual void initHeader() override;

virtual CStringData data(const ModelIndex &index) const override;

virtual int columnCount() const override;

virtual int rowCount() const override;

void update(const LinkList<Car> &newStorage);

private:

LinkList<Car> m\_showList;

};

#endif // CARSVIEWMODEL\_H

Файл CarViewModel.cpp

#include "carsviewmodel.h"

#include "../../app\_core/cstringdata.h"

#include <cstring>

CarsViewModel::CarsViewModel()

: AbstractItemModel()

, m\_showList()

{

}

CarsViewModel::~CarsViewModel()

{

}

void CarsViewModel::initHeader()

{

char n[100] = "N";

char numb[100] = "Гос. номер";

char brand[100] = "Марка";

char color[100] = "Цвет";

char year[100] = "Год выпуска";

char available[100] = "Доступен";

char header[600] = "\0";

std::strncat(header, n, 100);

for(int i = 0; i < 3 - app\_global::numberOfLetters(n); ++i)

std::strncat(header, " ", 1);

std::strncat(header, " | ", 3);

std::strncat(header, numb, 100);

for(int i = 0; i < 45 - app\_global::numberOfLetters(numb); ++i)

std::strncat(header, " ", 1);

std::strncat(header, " | ", 3);

std::strncat(header, brand, 100);

for(int i = 0; i < 45 - app\_global::numberOfLetters(brand); ++i)

std::strncat(header, " ", 1);

std::strncat(header, " | ", 3);

std::strncat(header, color, 100);

for(int i = 0; i < 45 - app\_global::numberOfLetters(color); ++i)

std::strncat(header, " ", 1);

std::strncat(header, " | ", 3);

std::strncat(header, year, 100);

for(int i = 0; i < 45 - app\_global::numberOfLetters(year); ++i)

std::strncat(header, " ", 1);

std::strncat(header, " | ", 3);

std::strncat(header, available, 100);

for(int i = 0; i < 45 - app\_global::numberOfLetters(available); ++i)

std::strncat(header, " ", 1);

setHeaderData(CStringData(header, 600));

}

CStringData CarsViewModel::data(const ModelIndex &index) const

{

if(m\_showList.size() < index.row() - 1

|| !index.isValid())

return CStringData();

if(index.column() > columnCount())

return CStringData();

int realRow = index.row() - 1;

if(realRow < 0)

return headerData();

switch (index.column()) {

case 0:

return CStringData(index.row());

case 1:

return CStringData(m\_showList[realRow].number(), app\_global::car::numberMaxLen);

case 2:

return CStringData(m\_showList[realRow].brand(), app\_global::car::brandMaxLen);

case 3:

return CStringData(m\_showList[realRow].color(), app\_global::car::colorMaxLen);

case 4:

return CStringData(m\_showList[realRow].year());

case 5: {

if(m\_showList[realRow].available())

return CStringData("Да", app\_global::car::availableMaxLen);

else

return CStringData("Нет", app\_global::car::availableMaxLen);

}

default:

return CStringData();

}

return CStringData();

}

int CarsViewModel::columnCount() const

{

return 6;

}

int CarsViewModel::rowCount() const

{

return m\_showList.size() + 1;

}

void CarsViewModel::update(const LinkList<Car> &newStorage)

{

m\_showList = newStorage;

}

Файл CarModel.h

#ifndef CARMODEL\_H

#define CARMODEL\_H

#include "../../items.h"

#include "../../global.h"

#include "../../app\_core/hash.h"

#include "../../app\_core/cstringdata.h"

class CarsViewModel;

class CarsModel

{

public:

enum CarRoles {

UndefinedRole = -1,

Number = 1,

Brand,

Color,

Year

};

CarsModel();

~CarsModel();

void setData(const Car &data);

Car getData(const CStringData &key) const;

bool hasData(const CStringData &key) const;

void removeData(const CStringData &key);

void clearData();

void filter(const CStringData &key, int role) const;

void setAvailable(const CStringData &key, bool available);

void setViewModel(CarsViewModel \*model);

private:

void notifyViewModel(const LinkList<Car> &newStorage) const;

private:

Hash<Car, app\_global::car::numberMaxLen> m\_currentStorage;

CarsViewModel \*m\_viewModel;

};

#endif // CARMODEL\_H

Файл CarModel.cpp

#include "carsmodel.h"

#include "../viewmodels/carsviewmodel.h"

CarsModel::CarsModel()

: m\_currentStorage()

, m\_viewModel(nullptr)

{

}

CarsModel::~CarsModel()

{

}

void CarsModel::setData(const Car &data)

{

m\_currentStorage.insert(data.number(), data);

notifyViewModel(m\_currentStorage.values());

}

Car CarsModel::getData(const CStringData &key) const

{

if(!m\_currentStorage.hasKey(key.data()))

return Car();

return m\_currentStorage[key.data()];

}

bool CarsModel::hasData(const CStringData &key) const

{

return m\_currentStorage.hasKey(key.data());

}

void CarsModel::removeData(const CStringData &key)

{

if(!m\_currentStorage.hasKey(key.data()))

return;

m\_currentStorage.erase(key.data());

notifyViewModel(m\_currentStorage.values());

}

void CarsModel::clearData()

{

m\_currentStorage.clear();

notifyViewModel(m\_currentStorage.values());

}

void CarsModel::filter(const CStringData &key, int role) const

{

(void)key;

LinkList<Car> storage;

switch (role) {

case Brand: {

LinkList<Car> cars = m\_currentStorage.values();

LinkList<Car>::iterator iter = cars.begin();

while(iter != cars.end()) {

iter = app\_global::find\_if(iter, cars.end(), [&](const Car &left) {

return std::strncmp(left.brand(), key.data(), app\_global::car::brandMaxLen) == 0;

});

if(iter != cars.end())

storage.append(\*iter++);

}

break;

}

case UndefinedRole:

storage = m\_currentStorage.values();

break;

default:

return;

}

notifyViewModel(storage);

}

void CarsModel::setAvailable(const CStringData &key, bool available)

{

if(!m\_currentStorage.hasKey(key.data()))

return;

m\_currentStorage[key.data()].setAvailable(available);

notifyViewModel(m\_currentStorage.values());

}

void CarsModel::setViewModel(CarsViewModel \*model)

{

m\_viewModel = model;

notifyViewModel(m\_currentStorage.values());

}

void CarsModel::notifyViewModel(const LinkList<Car> &newStorage) const

{

if(m\_viewModel)

m\_viewModel->update(newStorage);

}

Файл ClientModel.h

#ifndef CLIENTSMODEL\_H

#define CLIENTSMODEL\_H

#include "../../global.h"

#include "../../items.h"

#include "../../app\_core/cstringdata.h"

#include "../../app\_core/searchtree.h"

class ClientsViewModel;

class ClientsModel

{

public:

enum ClientRoles {

UndefinedRole = -1,

License = 1,

Name,

PassportData,

Address

};

ClientsModel();

~ClientsModel();

void setData(const Client &data);

Client getData(const CStringData &key) const;

bool hasData(const CStringData &key) const;

void removeData(const CStringData &key);

void clearData();

void filter(const CStringData &key, int role);

void setViewModel(ClientsViewModel \*model);

private:

void notifyViewModel(const LinkList<Client> &newStorage) const;

private:

Tree<Client, app\_global::client::licenseMaxLen> m\_currentStorage;

ClientsViewModel \*m\_viewModel;

};

#endif // CLIENTSMODEL\_H

Файл ClientModel.cpp

#include "clientsmodel.h"

#include "../viewmodels/clientsviewmodel.h"

ClientsModel::ClientsModel()

: m\_currentStorage()

, m\_viewModel(nullptr)

{

}

ClientsModel::~ClientsModel()

{

}

void ClientsModel::setData(const Client &data)

{

m\_currentStorage.insert(data.license(), data);

notifyViewModel(m\_currentStorage.values());

}

Client ClientsModel::getData(const CStringData &key) const

{

if(!m\_currentStorage.hasKey(key.data()))

return Client();

return m\_currentStorage[key.data()];

}

bool ClientsModel::hasData(const CStringData &key) const

{

return m\_currentStorage.hasKey(key.data());

}

void ClientsModel::removeData(const CStringData &key)

{

if(!m\_currentStorage.hasKey(key.data()))

return;

m\_currentStorage.erase(key.data());

notifyViewModel(m\_currentStorage.values());

}

void ClientsModel::clearData()

{

m\_currentStorage.clear();

notifyViewModel(m\_currentStorage.values());

}

void ClientsModel::filter(const CStringData &key, int role)

{

LinkList<Client> storage;

Client searchVal;

switch (role) {

case Name:

searchVal.setName(key.data());

storage = m\_currentStorage.find(searchVal, [](const Client &left, const Client &right) {

return app\_global::textSearch(left.name(), right.name());

});

break;

case Address:

searchVal.setAddress(key.data());

storage = m\_currentStorage.find(searchVal, [](const Client &left, const Client &right) {

return app\_global::textSearch(left.address(), right.address());

});

break;

case UndefinedRole:

storage = m\_currentStorage.values();

break;

default:

break;

}

notifyViewModel(storage);

}

void ClientsModel::setViewModel(ClientsViewModel \*model)

{

m\_viewModel = model;

notifyViewModel(m\_currentStorage.values());

}

void ClientsModel::notifyViewModel(const LinkList<Client> &newStorage) const

{

if(m\_viewModel)

m\_viewModel->update(newStorage);

}

Файл ClientViewModel.h

#ifndef CLIENTSVIEWMODEL\_H

#define CLIENTSVIEWMODEL\_H

#include "abstractitemmodel.h"

#include "../../app\_core/linklist.h"

#include "../../items.h"

class **ClientsViewModel** : public AbstractItemModel

{

public:

**ClientsViewModel**();

~***ClientsViewModel***();

virtual void ***initHeader***() override;

virtual CStringData ***data***(const ModelIndex &index) const override;

virtual int ***columnCount***() const override;

virtual int ***rowCount***() const override;

void **update**(const LinkList<Client> &newStorage);

private:

LinkList<Client> m\_showList;

};

#endif // CLIENTSVIEWMODEL\_H

Файл ClientViewModel.cpp

#include "clientsviewmodel.h"

#include "../../app\_core/cstringdata.h"

#include <cstring>

ClientsViewModel::**ClientsViewModel**()

: AbstractItemModel()

, m\_showList()

{

}

ClientsViewModel::~***ClientsViewModel***()

{

}

void ClientsViewModel::***initHeader***()

{

char n[100] = "N";

char licence[100] = "Номер удостоверения";

char name[100] = "ФИО";

char passport[100] = "Паспортные данные";

char address[100] = "Адрес";

char header[600] = "\0";

std::strncat(*header*, n, 100);

for(int i = 0; i < 3 - app\_global::numberOfLetters(n); ++i)

std::strncat(*header*, " ", 1);

std::strncat(*header*, " | ", 3);

std::strncat(*header*, licence, 100);

for(int i = 0; i < 45 - app\_global::numberOfLetters(licence); ++i)

std::strncat(*header*, " ", 1);

std::strncat(*header*, " | ", 3);

std::strncat(*header*, name, 100);

for(int i = 0; i < 45 - app\_global::numberOfLetters(name); ++i)

std::strncat(*header*, " ", 1);

std::strncat(*header*, " | ", 3);

std::strncat(*header*, passport, 100);

for(int i = 0; i < 45 - app\_global::numberOfLetters(passport); ++i)

std::strncat(*header*, " ", 1);

std::strncat(*header*, " | ", 3);

std::strncat(*header*, address, 100);

for(int i = 0; i < 45 - app\_global::numberOfLetters(address); ++i)

std::strncat(*header*, " ", 1);

setHeaderData(CStringData(header, 600));

}

CStringData ClientsViewModel::***data***(const ModelIndex &index) const

{

if(m\_showList.size() < index.row() - 1

|| !index.isValid())

return CStringData();

if(index.column() > *columnCount*())

return CStringData();

int realRow = index.row() - 1;

if(realRow < 0)

return headerData();

switch (index.column()) {

case 0:

return CStringData(index.row());

case 1:

return CStringData(m\_showList[realRow].license(), app\_global::client::licenseMaxLen);

case 2:

return CStringData(m\_showList[realRow].name(), app\_global::client::nameMaxLen);

case 3:

return CStringData(m\_showList[realRow].passport(), app\_global::client::passportMaxLen);

case 4:

return CStringData(m\_showList[realRow].address(), app\_global::client::addressMaxLen);

default:

return CStringData();

}

return CStringData();

}

int ClientsViewModel::***columnCount***() const

{

return 5;

}

int ClientsViewModel::***rowCount***() const

{

return m\_showList.size() + 1;

}

void ClientsViewModel::**update**(const LinkList<Client> &newStorage)

{

m\_showList = newStorage;

}

Файл RentInfoModel.h

#ifndef RENTINFOMODEL\_H

#define RENTINFOMODEL\_H

#include "../../items.h"

#include "../../global.h"

#include "../../app\_core/hash.h"

#include "../../app\_core/cstringdata.h"

class RentInfoViewModel;

class RentInfoModel

{

public:

enum RentInfoRoles {

UndefinedRole = -1,

CarRole = 1,

ClientRole,

IssueDateRole,

ReturnDateRole

};

RentInfoModel();

~RentInfoModel();

void setData(const RentInfo &data);

LinkList<RentInfo> getData(const CStringData &key, int keyRole) const;

bool hasData(const CStringData &key, int keyRole) const;

void removeData(const CStringData &carKey);

void clearData();

void filter(const CStringData &key, int role);

void setViewModel(RentInfoViewModel \*model);

private:

void notifyViewModel(const LinkList<RentInfo> &newStorage);

private:

LinkList<RentInfo> m\_currentStorage;

RentInfoViewModel \*m\_viewModel;

};

#endif // RENTINFOMODEL\_H

Файл RentInfoModel.cpp

#include "rentinfomodel.h"

#include "../viewmodels/rentinfoviewmodel.h"

RentInfoModel::**RentInfoModel**()

: m\_currentStorage()

, m\_viewModel(nullptr)

{

}

RentInfoModel::~**RentInfoModel**()

{

}

void RentInfoModel::**setData**(const RentInfo &data)

{

m\_currentStorage.append(data);

app\_global::sort(*m\_currentStorage*, m\_currentStorage.begin(), --m\_currentStorage.end(),

[&](const RentInfo &left, const RentInfo &right){

return (std::strncmp(left.carData(), right.carData(), app\_global::car::numberMaxLen));

});

notifyViewModel(m\_currentStorage);

}

LinkList<RentInfo> RentInfoModel::**getData**(const CStringData &key, int keyRole) const

{

LinkList<RentInfo> storage;

LinkList<RentInfo>::iterator iter = m\_currentStorage.begin();

while(iter != m\_currentStorage.end()) {

iter = app\_global::find\_if(iter, m\_currentStorage.end(), [&](const RentInfo &info) {

switch (keyRole) {

case CarRole:

return std::strncmp(info.carData(), key.data(), key.length()) == 0;

case ClientRole:

return std::strncmp(info.clientData(), key.data(), key.length()) == 0;

case IssueDateRole:

return false;

case ReturnDateRole:

return false;

default:

return false;

}

return false;

});

if(iter != m\_currentStorage.end())

storage.append(\*iter++);

}

return storage;

}

bool RentInfoModel::**hasData**(const CStringData &key, int keyRole) const

{

LinkList<RentInfo>::iterator iter = app\_global::find\_if(m\_currentStorage.begin(), m\_currentStorage.end(), [&](const RentInfo &info) {

switch (keyRole) {

case CarRole:

return std::strncmp(info.carData(), key.data(), key.length()) == 0;

case ClientRole:

return std::strncmp(info.clientData(), key.data(), key.length()) == 0;

case IssueDateRole:

return std::strncmp(info.issueDate(), key.data(), key.length()) == 0;

case ReturnDateRole:

return std::strncmp(info.returnDate(), key.data(), key.length()) == 0;

default:

return false;

}

return false;

});

if(iter != m\_currentStorage.end())

return true;

return false;

}

void RentInfoModel::**removeData**(const CStringData &carKey)

{

LinkList<RentInfo>::iterator iter = app\_global::find\_if(m\_currentStorage.begin(), m\_currentStorage.end(), [&](const RentInfo &info) {

return std::strncmp(info.carData(), carKey.data(), carKey.length()) == 0;

});

if(iter == m\_currentStorage.end())

return;

m\_currentStorage.erase(iter);

}

void RentInfoModel::**clearData**()

{

m\_currentStorage.clear();

notifyViewModel(m\_currentStorage);

}

void RentInfoModel::**filter**(const CStringData &key, int role)

{

(void)key;

(void)role;

}

void RentInfoModel::**setViewModel**(RentInfoViewModel \*model)

{

m\_viewModel = model;

}

void RentInfoModel::**notifyViewModel**(const LinkList<RentInfo> &newStorage)

{

if(m\_viewModel)

m\_viewModel->update(newStorage);

}

Файл RentIndoViewModel.h

#ifndef RENTINFOVIEWMODEL\_H

#define RENTINFOVIEWMODEL\_H

#include "abstractitemmodel.h"

#include "../../app\_core/linklist.h"

#include "../../items.h"

class RentInfoViewModel : public AbstractItemModel

{

public:

RentInfoViewModel();

~RentInfoViewModel();

virtual void initHeader() override;

virtual CStringData data(const ModelIndex &index) const override;

virtual int columnCount() const override;

virtual int rowCount() const override;

void update(const LinkList<RentInfo> &newStorage);

private:

LinkList<RentInfo> m\_showList;

};

#endif // RENTINFOVIEWMODEL\_H

Файл RentIndoViewModel.cpp

#include "rentinfoviewmodel.h"

#include "../../app\_core/cstringdata.h"

#include <cstring>

RentInfoViewModel::**RentInfoViewModel**()

: AbstractItemModel()

, m\_showList()

{

}

RentInfoViewModel::~***RentInfoViewModel***()

{

}

void RentInfoViewModel::***initHeader***()

{

char n[100] = "N";

char licence[100] = "Номер водительского удостоверерния клиента";

char number[100] = "Гос. номер автомобиля";

char issueDate[100] = "Дата выдачи в аренду";

char returnDate[100] = "Дата возврата";

char header[600] = "\0";

std::strncat(*header*, n, 100);

for(int i = 0; i < 3 - app\_global::numberOfLetters(n); ++i)

std::strncat(*header*, " ", 1);

std::strncat(*header*, " | ", 3);

std::strncat(*header*, licence, 100);

for(int i = 0; i < 45 - app\_global::numberOfLetters(licence); ++i)

std::strncat(*header*, " ", 1);

std::strncat(*header*, " | ", 3);

std::strncat(*header*, number, 100);

for(int i = 0; i < 45 - app\_global::numberOfLetters(number); ++i)

std::strncat(*header*, " ", 1);

std::strncat(*header*, " | ", 3);

std::strncat(*header*, issueDate, 100);

for(int i = 0; i < 45 - app\_global::numberOfLetters(issueDate); ++i)

std::strncat(*header*, " ", 1);

std::strncat(*header*, " | ", 3);

std::strncat(*header*, returnDate, 100);

for(int i = 0; i < 45 - app\_global::numberOfLetters(returnDate); ++i)

std::strncat(*header*, " ", 1);

setHeaderData(CStringData(header, 600));

}

CStringData RentInfoViewModel::***data***(const ModelIndex &index) const

{

if(m\_showList.size() < index.row() - 1

|| !index.isValid())

return CStringData();

if(index.column() > *columnCount*())

return CStringData();

int realRow = index.row() - 1;

if(realRow < 0)

return headerData();

switch (index.column()) {

case 0:

return CStringData(index.row());

case 1:

return CStringData(m\_showList[realRow].clientData(), app\_global::client::licenseMaxLen);

case 2:

return CStringData(m\_showList[realRow].carData(), app\_global::car::numberMaxLen);

case 3:

return CStringData(m\_showList[realRow].issueDate(), app\_global::dateLen);

case 4:

return CStringData(m\_showList[realRow].returnDate(), app\_global::dateLen);

default:

return CStringData();

}

return CStringData();

}

int RentInfoViewModel::***columnCount***() const

{

return 5;

}

int RentInfoViewModel::***rowCount***() const

{

return m\_showList.size() + 1;

}

void RentInfoViewModel::**update**(const LinkList<RentInfo> &newStorage)

{

m\_showList = newStorage;

}