

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ

УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ

(национальный исследовательский университет)»

Факультет №3 «Системы управления, информатика и электроэнергетика»

Кафедра № 304 «Вычислительные машины, системы и сети»

Пояснительная записка к курсовой работе

по дисциплине: «Программирование»

на тему: «Система управления базой данных»

Выполнил:

студент группы М3О-211Б-21

Багиров Э. Р.

Проверила:

Дмитриева Е. А.

Москва 2022 г.

Задание

Разработать систему управления базой данных на языке С++.

Для хранения данных использовать текстовый файл.

Разработать структуры данных для хранения данных и функции для их обработки.

Тематика базы данных: владелец автомобиля (имя, номер автомобиля, номер техпаспорта, дата рождения, телефон, ид отделения гибдд, водительское удостоверение, срок годности удостоверения, штрафы)

Оглавление

[Задание 2](#_Toc125331665)

[Введение 5](#_Toc125331666)

[Глава 1. Хэш-таблицы. Понятие, свойства 6](#_Toc125331667)

[1.1. Определение хэш-таблицы 6](#_Toc125331668)

[1.2. Организация хэш-таблицы в представленной работе 6](#_Toc125331669)

[1.3. Хэш-функция в представленной работе 7](#_Toc125331670)

[1.4. Аналитика хэш-таблицы. 8](#_Toc125331671)

[Глава 2. Алгоритмы работы с хэш-таблицей. 9](#_Toc125331672)

[2.1. Алгоритм хэширования 9](#_Toc125331673)

[2.1.1. Программная реализация хэширования 9](#_Toc125331674)

[2.1.2. Пример работы алгоритма хэширования 9](#_Toc125331675)

[2.2. Алгоритм инициализации таблицы 10](#_Toc125331676)

[2.2.1. Программная реализация инициализации 10](#_Toc125331677)

[2.3. Аналитика хэш-таблицы 10](#_Toc125331678)

[2.3.1. Программная реализация аналитики 11](#_Toc125331679)

[2.3.2. Программная реализация вывода аналитики 11](#_Toc125331680)

[2.3.3. Пример вывода аналитики 11](#_Toc125331681)

[2.4. Алгоритм добавления элемента 11](#_Toc125331682)

[2.4.1. Программная реализация добавления 11](#_Toc125331683)

[2.5. Алгоритм удаления элемента 12](#_Toc125331684)

[2.5.1. Программная реализация удаления 12](#_Toc125331685)

[2.6. Алгоритм перехэширования 13](#_Toc125331686)

[2.6.1. Программная реализация перехэширования 13](#_Toc125331687)

[Глава 3. Рандомизированное дерево. Понятия, свойства 14](#_Toc125331688)

[3.1. Определение рандомизированного дерева 14](#_Toc125331689)

[Глава 4. Алгоритмы работы с деревом 15](#_Toc125331690)

[4.1. Алгоритмы правого и левого поворотов 15](#_Toc125331691)

[4.1.2. Программная реализация поворотов 15](#_Toc125331692)

[4.2. Алгоритм вставки в корень поддерева 15](#_Toc125331693)

[4.2.2. Программная реализация вставки в корень 15](#_Toc125331694)

[4.3. Алгоритм рандомизированной вставки 16](#_Toc125331695)

[4.3.1. Программная реализация рандомизированной вставки 16](#_Toc125331696)

[4.4. Алгоритм удаления 16](#_Toc125331697)

[4.4.1. Программная реализация удаления 16](#_Toc125331698)

[Заключение 18](#_Toc125331699)

[Приложение 1 19](#_Toc125331700)

[Приложение 2 24](#_Toc125331701)

[Приложение 3 42](#_Toc125331702)

# Введение

В данной курсовой работе были применены такие структуры данных, как хэш-таблица, рандомизированное бинарное дерево, связный список. Выбор структуры данных для каждого типа данных был сделан для оптимальной скорости доступа и добавления, и эффективной работы с памятью.

Также рассмотрены алгоритмы работы с хэш-таблицей: добавление элементов, удаление элементов, получение аналитики занятости таблицы, изменение размера таблицы с перехэшированием; алгоритмы работы с рандомизированным бинарным деревом поиска: рандомизированная вставка элемента, удаление элемента, поиск элемента по ид.

Для программной реализации использован язык программирования С++.

# Глава 1. Хэш-таблицы. Понятие, свойства

## 1.1. Определение хэш-таблицы

Хеш-таблица - структура данных, обеспечивающая очень быструю вставку и поиск. Независимо от количества элементов данных вставка и поиск (а иногда и удаление) выполняются за время, близкое к постоянному, О(1).

1. Доступ к элементам хэш-таблицы осуществляется путем хэширования ключа элемента таблицы. После прохождения ключа через функцию хэша, получается число, которому соответствует элемент поиска в массиве данных.
2. Добавление элемента в хэш-таблицу осуществляется путем хэширования ключа элемента таблицы. После прохождения ключа через функцию хэша, получается число, в соответствии которому необходимо поставить новый элемент в массив данных.
3. Расширение (сужение) хэш-таблицы происходит заданием нового массива данных нужного размера, перехэширование ключей каждого элемента старого массива с новым параметром размера таблицы, помещение каждого элемента старой таблицы в новую в соответствии с новым хэшем.

## 1.2. Организация хэш-таблицы в представленной работе

В курсовой работе предложена следующая организация таблицы:

Уникальным ключом владельца автомобиля является его сочетание имени и фамилии. Хэшированию подвергается поле name данных о владельце.

Хэш-таблица с закрытой адресацией. В массиве хранятся ссылки на связные списки владельцев автомобилей.

Diagram

Description automatically generated

## 1.3. Хэш-функция в представленной работе

В курсовой работе предложена следующая хэш-функция:

HugeNumber: Для каждого символа из поля name данных о владельце выполнить: нормализованный код символа умножить на простое число 13, возведенное в степень – позиция символа в слове по модулю 7.

(ascii - 64) \* 13^(pos%7+1)

ArrayIndex: HugeNumber по модулю N, где N – вместимость массива.

Доказательство выбора функции хэша:

Для набора уникальных сочетаний Имя Фамилия, которые были использованы в курсовой работе для генерации тестовых данных, были проведены тесты функции хэширования. Тест заключался в хэшировании всего набора сочетаний и сборе статистики (приложение 1) о коллизиях полученных индексов массива. Тест показал, что предложенная функция хэширования обеспечивает минимальное отклонение 70- и 50-процентиля от среднего значения распределения сочетаний Имя Фамилия по значениям индексов массива.

Тест также показал, что значение коллизий значительно возрастает в случаях, когда длина массива есть число, кратное 13. Это следствие выбора функции хэширования. В работе предложен метод избежания этой аномалии: при задании нового массива, если запрошенная длина кратна 13, создать массив длиной (запрошенная длина+1). Такой метод избежания аномалии не должен создавать проблем, так как вместительность массива является полем класса хэш-таблицы, и при любых операциях с таблицей программист может (и это рекомендуется) использовать значение длины массива из поля класса хэш таблицы.

## 1.4. Аналитика хэш-таблицы.

Производительность хэш-таблицы значительно падает при увеличении количества коллизий. С увеличением заполненности таблицы возрастает количество коллизий. Для поддержания производительности хэш-таблицы необходимо своевременно расширять ее. Для этого в работе предложено введение аналитики таблицы – учет количества элементов таблицы по расположению в связных списках: если при добавлении элемент добавляется к существующему связному списку, это считается коллизией; иначе элемент занял пустую ячейку таблицы. Аналитика в любой момент позволяет оценить заполненность таблицы для принятия решения о расширении таблицы и перехэшировании элементов.

# Глава 2. Алгоритмы работы с хэш-таблицей.

## 2.1. Алгоритм хэширования

### 2.1.1. Программная реализация хэширования

Нестатические методы класса вызывают статический метод Hash с аргументами строки для хэширования и емкости массива.

int Hash(string name) {

return Hash(name, capacity);

}

int Hash(CarOwner owner) {

return Hash(owner.getName());

}

static int Hash(string name, int capacity) {

// (ascii - 64) \* 13^(pos%7+1)

size\_t size = name.length();

unsigned int hash = 0;

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

int p = i % 7 + 1;

p = pow(13, p);

int ascii = int(name[i]) - 64;

hash += p \* ascii;

}

return hash % capacity;

}

### 2.1.2. Пример работы алгоритма хэширования

Программа:

setlocale(LC\_ALL, "rus");

string s1 = "Anton Petrov";

string s2 = "Ivan Nikolaev";

int base1 = 12;

int base2 = 244;

cout << "Хэш строки \"" << s1 << "\" при длине массива " << base1

<< " равен " << OwnersHashTable::Hash(s1, base1) << endl;

cout << "Хэш строки \"" << s1 << "\" при длине массива " << base2

<< " равен " << OwnersHashTable::Hash(s1, base2) << endl;

cout << "Хэш строки \"" << s2 << "\" при длине массива " << base1

<< " равен " << OwnersHashTable::Hash(s2, base1) << endl;

cout << "Хэш строки \"" << s2 << "\" при длине массива " << base2

<< " равен " << OwnersHashTable::Hash(s2, base2) << endl;

Вывод:

Хэш строки "Anton Petrov" при длине массива 12 равен 8

Хэш строки "Anton Petrov" при длине массива 244 равен 76

Хэш строки "Ivan Nikolaev" при длине массива 12 равен 3

Хэш строки "Ivan Nikolaev" при длине массива 244 равен 43

## 2.2. Алгоритм инициализации таблицы

Инициализация хэш-таблицы заключается в задании длины массива, в соответствии с которой в дальнейшем будет работать функция хэша, и в инициализации массива с заданной емкостью.

### 2.2.1. Программная реализация инициализации

unsigned int capacity = 100;

OwnersHashTable() {

SetupArray();

}

OwnersHashTable(int capacity) {

if (capacity % 13 == 0) { capacity++; }

int last\_capacity = this->capacity;

this->capacity = capacity;

try {

SetupArray();

}

catch(...) {

this->capacity = last\_capacity;

}

}

void SetupArray() {

if (arr) { throw "there is already an array"; }

arr = new CarOwnersList[capacity];

}

Конструктор класса по умолчанию OwnersHashTable() инициализирует хэш таблицу со значением емкости по умолчанию - 100. Конструктор класса OwnersHashTable(capacity) принимает в качестве аргумента запрашиваемую емкость таблицы. Тут же проверяется кратность запрошенной емкости числу 13 для избежания аномалий коллизий. Функция SetupArray() инициализирует массив, если он еще не инициализирован.

## 2.3. Аналитика хэш-таблицы

Аналитика содержит следующие поля: емкость таблицы (дублирует поле класса таблицы; необходимо для удобного доступа – дублирование позволяет не обращаться к геттеру поля емкости), население (суммарное количество элементов данных в таблице), заполненность ячеек, количество коллизий, соотношение заполненности и соотношение коллизий:

struct HashTableAnalytics {

unsigned int capacity = 0;

unsigned int population = 0;

unsigned int filling = 0;

unsigned int collisions = 0;

double fillingRate = 0; // will be out of date until GetAnalytics() is called

double collisionsRate = 0;// will be out of date until GetAnalytics() is called

};

### 2.3.1. Программная реализация аналитики

Все поля, кроме емкости и соотношений, обновляются при каждом изменении таблицы, как то добавление или удаление элемента. Поэтому для получения аналитики требуется обновить только три поля:

HashTableAnalytics GetAnalytics() {

analytics.capacity = this->capacity;

analytics.fillingRate = (float)analytics.filling / analytics.capacity;

analytics.collisionsRate = (float)analytics.collisions / analytics.population;

return analytics;

}

### 2.3.2. Программная реализация вывода аналитики

Класс ConsoleInterface имеет статический метод для вывода аналитики:

static void OutputHashTableAnalytics(HashTableAnalytics analytics) {

cout << "Capacity:\t" << analytics.capacity

<< "\nPopulation:\t" << analytics.population

<< "\nFilling:\t" << analytics.filling

<< "\nFilling rate:\t" << analytics.fillingRate

<< "\nCollisions:\t" << analytics.collisions

<< "\nCollisions rate:" << analytics.collisionsRate

<< endl;

}

### 2.3.3. Пример вывода аналитики

Приведем пример вывода аналитики заполненной хэш-таблицы:

Capacity: 400

Population: 200

Filling: 151

Filling rate: 0.3775

Collisions: 49

Collisions rate:0.245

## 2.4. Алгоритм добавления элемента

Добавление элемента заключается в получении хэша поля имени владельца автомобиля и вставка владельца в список, который располагается в массиве по индексу хэша. Также добавление изменяет аналитику таблицы.

### 2.4.1. Программная реализация добавления

void InsertToTable(CarOwner\* owner) {

if (owner->getName() == "") throw "wrong name - can't hash";

analytics.population++;

int index = Hash(\*owner);

if (arr[index].InsertToList(owner)) {

analytics.collisions++;

}

else {

analytics.filling += 1;

}

}

## 2.5. Алгоритм удаления элемента

Необходимо найти хэш ключа удаляемого владельца, исключить владельца из списка, расположенного в массиве по адресу хэша.

### 2.5.1. Программная реализация удаления

void DeleteFromTable(string name) {

int index = Hash(name);

char result = arr[index].DeleteFromList(name);

if (result == 1) {

analytics.population--;

analytics.filling--;

}

else if (result == 2) {

analytics.collisions--;

analytics.population--;

}

}

char DeleteFromList(string name) {

///

/// 0 - didn't delete

/// 1 - did delete, nothing else

/// 2 - did delete, list contains something else

///

CarOwner\* tmp = firstOwner;

// empty list

if (!tmp) {

return 0;

}

// element for deleting is first

else if (tmp->getName() == name) {

tmp = firstOwner->getNextOwner();

delete firstOwner;

firstOwner = tmp;

if (tmp) { return 2; }

return 1;

}

// list contains only one element

else if (!tmp->getNextOwner()) {

return 0;

}

else {

while (tmp->getNextOwner()) {

if (tmp->getNextOwner()->getName() == name) {

CarOwner\* buff = tmp->getNextOwner()->getNextOwner();

delete tmp->getNextOwner();

tmp->setNextOwner(buff);

if (buff) { return 2; }

return 1;

}

else {

tmp = tmp->getNextOwner();

}

}

}

return 0;

}

Метод класса хэш таблицы DeleteFromTable получает хэш ключа удаления и обращается к методу класса списка, который расположен по индексу – полученному хэшу – метод списка ищет и исключает владельца по ключу и результат (владелец найден или нет, есть ли в списке владельцы помимо удаляемого) возвращается в метод класса таблицы, которая в зависимости от результата изменяет аналитику.

## 2.6. Алгоритм перехэширования

Перехэширование необходимо совершить, если рейтинг заполнения хэш-таблицы более чем 0.8 или рейтинг коллизий превышает 0.4, или если рейтинг заполнения меньше 0.1.

Перехэширование заключается в инициализации новой хэш-таблицы с размером, равным двойному количеству популяции старой хэш-таблицы. Далее, для каждого элемента старой таблицы вычисляется новый хэш и элементы вносятся в новую таблицу в соответствии с новым хэшем.

### 2.6.1. Программная реализация перехэширования

void Rehash(int newCapacity = 0) {

if (newCapacity < 0) throw "wrong capacity";

if (!newCapacity) newCapacity = GetAnalytics().population \* 2;

OwnersHashTable ht = OwnersHashTable(newCapacity);

if (arr) {

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

CarOwner\* owner = arr[i].firstOwner;

while (owner) {

ht.InsertToTable(owner);

owner = owner->getNextOwner();

}

}

DeleteArray();

}

this->arr = ht.getArray();

this->capacity = ht.getCapacity();

}

const double fillingRateUpperLimit = 0.8;

const double collisionsRateUpperlimit = 0.4;

const double fillingRateLowerLimit = 0.1;

bool RehashPending() {

HashTableAnalytics an = GetAnalytics();

if (an.fillingRate > fillingRateUpperLimit ||

an.collisionsRate > collisionsRateUpperlimit ||

an.fillingRate < fillingRateLowerLimit) {

return true;

}

return false;

}

Функция RehashPending показывает, необходимо ли провести перехэширование.

Функиця Rehash проводит перехэширование.

# Глава 3. Рандомизированное дерево. Понятия, свойства

## 3.1. Определение рандомизированного дерева

Рандомизированное дерево – частично сбалансированное самобалансирующееся бинарное дерево. Частичная балансировка достигается рандомизированной вставкой в корень при поиске места вставки элемента, вероятность вставки в корень увеличивается при каждом переходе вниз по дереву.

Такой метод балансировки позволяет получить сложность доступа к элементу, добавления и удаления O(logn) в среднем и O(n) в худшем случае.

# Глава 4. Алгоритмы работы с деревом

## 4.1. Алгоритмы правого и левого поворотов

Повороты необходимы для переорганизации дерева при рандомизированной вставке в корень поддерева.

### 4.1.2. Программная реализация поворотов

TicketsTreeElement\* RotateTreeRight(TicketsTreeElement\* tree)

{

TicketsTreeElement\* q = tree->leftChild;

if (!q) return tree;

tree->leftChild = q->rightChild;

q->rightChild = tree;

q->size = tree->size;

FixTreeSize(tree);

return q;

}

TicketsTreeElement\* RotateTreeLeft(TicketsTreeElement\* tree)

{

TicketsTreeElement\* p = tree->rightChild;

if (!p) return tree;

tree->rightChild = p->leftChild;

p->leftChild = tree;

p->size = tree->size;

FixTreeSize(tree);

return p;

}

RotateTreeRight осуществляет правый поворот вокруг ноды tree.

RotateTreeLeft осуществляет левый поворот вокруг ноды tree.

## 4.2. Алгоритм вставки в корень поддерева

Вставка в корень поддерева происходит по результату рандомизации. Вставка в корень поддерева заключается в присоединении поддерева к новому элементу дерева и повороте нового поддерева в одну сторону в зависимости от значения поддеревьев.

### 4.2.2. Программная реализация вставки в корень

TicketsTreeElement\* TreeInsertToRoot(TicketsTreeElement\* tree, Ticket t)

{

if (!tree) return new TicketsTreeElement(t);

if (t.getId() < tree->ticket.getId())

{

tree->leftChild = TreeInsertToRoot(tree->leftChild, t);

return RotateTreeRight(tree);

}

else

{

tree->rightChild = TreeInsertToRoot(tree->rightChild, t);

return RotateTreeLeft(tree);

}

}

## 4.3. Алгоритм рандомизированной вставки

Любой новый элемент может стать корнем текущего поддерева с вероятностью 1/(n+1) , где n - размер поддерева, а с вероятностью 1-1/(n+1) происходит рекурсивная рандомизированная вставка в правое или левое поддерево данного поддерева.

### 4.3.1. Программная реализация рандомизированной вставки

TicketsTreeElement\* TreeRandomisedInsert(TicketsTreeElement\* p, Ticket t)

{

if (!p) return new TicketsTreeElement(t);

if (rand() % (p->size + 1) == 0 && p->ticket.getId() >= 0)

return TreeInsertToRoot(p, t);

if (p->ticket.getId() > t.getId())

p->leftChild = TreeRandomisedInsert(p->leftChild, t);

else

p->rightChild = TreeRandomisedInsert(p->rightChild, t);

FixTreeSize(p);

return p;

}

Пример рандомизированной вставки:

Два запуска программы с одинаковым тестовым набором данных привели к разным деревьям:

A picture containing text

Description automatically generated A picture containing text

Description automatically generated

## 4.4. Алгоритм удаления

Удаление происходит путем рекурсивного поиска элемента по ключу поиска в поддеревьях. В случае нахождения элемент исключается, а поддеревья элемента объединяются.

### 4.4.1. Программная реализация удаления

TicketsTreeElement\* TreeRemoveNode(TicketsTreeElement\* p, int id)

{

if (!p) return p;

if (p->ticket.getId() == id)

{

TicketsTreeElement\* q = TreeJoinNodes(p->leftChild, p->rightChild);

delete p;

return q;

}

else if (id < p->ticket.getId())

p->leftChild = TreeRemoveNode(p->leftChild, id);

else

p->rightChild = TreeRemoveNode(p->rightChild, id);

return p;

}

TicketsTreeElement\* TreeJoinNodes(TicketsTreeElement\* p, TicketsTreeElement\* q)

{

if (!p) return q;

if (!q) return p;

if (rand() % (p->size + q->size) < p->size)

{

p->rightChild = TreeJoinNodes(p->rightChild, q);

FixTreeSize(p);

return p;

}

else

{

q->leftChild = TreeJoinNodes(p, q->leftChild);

FixTreeSize(q);

return q;

}

}

# Заключение

В данной курсовой работе были рассмотрена реализация и алгоритмы работы с хэш-таблицы с закрытой адресацией, реализация и алгоритмы работы с рандомизированным деревом.

# Приложение 1

Тест функции хэширования.

Набор уникальных сочетаний Имя Фамилия содержит 984 строки. Проведен тест функции хэширования для каждого размера хэш-таблицы в диапазоне [1; 2n+1], где n=984. Графики распределений (70-процентиль, 50-процентиль, среднее) коллизий от размера массива представлены на рисунках п1.1, п1.2.

Проведен тест функции хэширования для каждого размера хэш-таблицы [1; 2n+1], где n=984, с предложенным способом избежания аномалии – коррекции длины хэш-таблицы. Графики распределений (70-процентиль, 50-процентиль, среднее) коллизий от размера массива представлены на рисунках п1.3, п1.4.

Рисунок п1.1. График распределений коллизий от размера хэш-таблицы

Graphical user interface, text, application, email

Description automatically generated

Рисунок п1.2. График распределений коллизий от размера хэш-таблицы, масштаб

Chart, histogram

Description automatically generated

Рисунок п1.3. График распределений коллизий от размера хэш-таблицы с коррекцией размера

Graphical user interface, application

Description automatically generated

Рисунок п1.4. График распределений коллизий от размера хэш-таблицы с коррекцией размера, масштаб

Graphical user interface, application

Description automatically generated

# Приложение 2

Код программы:

#include <iostream>

#include <string>

#include <sstream>

#include <fstream>

using namespace std;

string GetTimeAsString(string formatString, tm theTime)

{

formatString += '\a'; //force at least one character in the result

string buffer;

buffer.resize(formatString.size());

int len = strftime(&buffer[0], buffer.size(), formatString.c\_str(), &theTime);

while (len == 0) {

buffer.resize(buffer.size() \* 2);

len = strftime(&buffer[0], buffer.size(), formatString.c\_str(), &theTime);

}

buffer.resize(len - 1); //remove that trailing '\a'

return buffer;

}

struct TicketsListElement

{

int ticketId;

TicketsListElement\* nextElement = NULL;

};

struct TicketsList {

TicketsListElement\* firstElement = NULL;

// default constructor

TicketsList() {};

TicketsList(int data[], int n) {

GenerateTicketsList(data, n);

}

void DeleteList() {

while (firstElement) {

TicketsListElement\* tmp = firstElement->nextElement;

delete firstElement;

firstElement = tmp;

cout << (tmp == NULL ? 1 : 0) << endl;

}

firstElement = NULL;

}

void GenerateTicketsList(int data[], int n) {

if (!data || !n) { return; }

if (n < 1) {

firstElement = NULL;

return;

}

TicketsListElement\* element = new TicketsListElement{ data[0] };

for (int i = 1; i < n; i++) {

element = new TicketsListElement{ data[i], element };

}

firstElement = element;

}

void AddTicket(int id) {

TicketsListElement\* newTicket = new TicketsListElement{ id, firstElement };

firstElement = newTicket;

}

};

class CarOwner

{

string name = "";

tm birthday;

string phoneNumber = "";

unsigned int drivingLicence = 0;

tm drivingLicenceExpiry;

string carLicencePlate = "";

string techPassportId = "";

int gibddId = 0;

TicketsList ticketsList = TicketsList(NULL, NULL);

CarOwner\* nextOwner = NULL;

const string standardStringFormat = "%d.%m.%y";

public:

// default constructor

CarOwner() { };

// simple constructor for tests

CarOwner(string name) {

this->name = name;

}

// class constructor

CarOwner(string name, tm birthday, string phoneNumber, unsigned int drivingLicence,

tm drivingLicenceExpiry, string carLicencePlate, string techPassportId,

int gibddId, int\* tickets = NULL, int ticketsAmount = NULL)

{

this->name = name;

this->birthday = birthday;

this->phoneNumber = phoneNumber;

this->drivingLicence = drivingLicence;

this->drivingLicenceExpiry = drivingLicenceExpiry;

this->carLicencePlate = carLicencePlate;

this->techPassportId = techPassportId;

this->gibddId = gibddId;

this->ticketsList = TicketsList(tickets, ticketsAmount);

}

//// GETTERS

// getter for name

string getName() { return name; };

// getter for birthday

tm getBirthday() { return birthday; };

// getter for birthday as string

string getBirthdayStr() { return GetTimeAsString(standardStringFormat, birthday); }

// getter for phoneNumber

string getPhoneNumber() { return phoneNumber; };

// getter for drivingLicence

unsigned int getDrivingLicence() { return drivingLicence; };

// getter for drivingLicenceExpiry

tm getDrivingLicenceExpiry() { return drivingLicenceExpiry; };

string getDrivingLicenceExpiryStr() { return GetTimeAsString(standardStringFormat, drivingLicenceExpiry); }

// getter for carLicencePlate

string getCarLicencePlate() { return carLicencePlate; };

// getter for techPassportId

string getTechPassportId() { return techPassportId; };

// getter for gibddId

int getGibddId() { return gibddId; };

// getter for ticketsList

TicketsList getTicketsList() { return ticketsList; };

// getter for ticketsList

CarOwner\* getNextOwner() { return nextOwner; };

//// SETTERS

// setter for nextOwner

void setNextOwner(CarOwner\* owner) { nextOwner = owner; }

// setter for birthday

void setBirthday(tm bday) { birthday = bday; }

// setter for drivingLicence

void setDrivingLicence(int id) { drivingLicence = id; }

// setter for drivingLicenceExpiry

void setDrivingLicenceExpiry(tm timestamp) { drivingLicenceExpiry = timestamp; }

// setter for gibddId

void setGibddId(int id) { gibddId = id; }

// setter for name

void setName(string name) { this->name = name; }

// setter for phoneNumber

void setPhoneNumber(string phoneNumber) { this->phoneNumber = phoneNumber; }

// setter for carLicencePlate

void setCarLicencePlate(string carLicencePlate) { this->carLicencePlate = carLicencePlate; }

// setter for techPassport

void setTechPassportId(string techPassportId) { this->techPassportId = techPassportId; }

void addTicket(int ticketId) { this->ticketsList.AddTicket(ticketId); }

unsigned int GetLength() {

if (nextOwner) { return nextOwner->GetLength() + 1; }

else return 1;

}

};

istream& operator>>(istream& is, CarOwner\* owner)

{

string name, phoneNumber, bdD, bdM, bdY, licId,

licED, licEM, licEY, gbddId, tckts, carLicencePlate,

techPassportId;

if (getline(is, name, ';') &&

getline(is, bdD, ';') &&

getline(is, bdM, ';') &&

getline(is, bdY, ';') &&

getline(is, phoneNumber, ';') &&

getline(is, licId, ';') &&

getline(is, licED, ';') &&

getline(is, licEM, ';') &&

getline(is, licEY, ';') &&

getline(is, carLicencePlate, ';') &&

getline(is, techPassportId, ';') &&

getline(is, gbddId, ';') &&

getline(is, tckts)) {

owner->setName(name);

owner->setPhoneNumber(phoneNumber);

owner->setCarLicencePlate(carLicencePlate);

owner->setTechPassportId(techPassportId);

owner->setGibddId(stoi(gbddId));

owner->setBirthday(tm{ 0,0,0,stoi(bdD),stoi(bdM) - 1,stoi(bdY) - 1900 });

owner->setDrivingLicence(stoi(licId));

owner->setDrivingLicenceExpiry(tm{ 0,0,0,stoi(licED),stoi(licEM),stoi(licEY) });

int f;

stringstream ss(tckts);

while ((ss >> f))

owner->addTicket(f);

}

return is;

}

ostream& operator<<(ostream& out, CarOwner\* owner) {

out << "Name:\t\t" << owner->getName() << endl;

out << "Bithday:\t" << owner->getBirthdayStr() << endl;

out << "Phone number:\t" << owner->getPhoneNumber() << endl;

out << "Dr. licence:\t" << owner->getDrivingLicence() << endl;

out << "Dr. l. expiry:\t" << owner->getDrivingLicenceExpiryStr() << endl;

out << "GIBDD id:\t" << owner->getGibddId() << endl;

out << "Licence plate:\t" << owner->getCarLicencePlate() << endl;

out << "Tech passport:\t" << owner->getTechPassportId() << endl;

TicketsList tl = owner->getTicketsList();

if (tl.firstElement) {

out << "Tickets id:\t";

TicketsListElement\* leel = tl.firstElement;

while (leel) {

out << leel->ticketId << " ";

leel = leel->nextElement;

}

}

out << endl;

return out;

}

struct CarOwnersList {

CarOwner\* firstOwner = NULL;

bool InsertToList(CarOwner\* owner) {

bool collision = false;

if (firstOwner) { collision = true; }

owner->setNextOwner(firstOwner);

firstOwner = owner;

return collision;

}

void DeleteList() {

while (firstOwner) {

CarOwner\* tmp = firstOwner->getNextOwner();

delete firstOwner;

firstOwner = tmp;

}

}

char DeleteFromList(string name) {

///

/// 0 - didn't delete

/// 1 - did delete, nothing else

/// 2 - did delete, list contains something else

///

CarOwner\* tmp = firstOwner;

// empty list

if (!tmp) {

return 0;

}

// element for deleting is first

else if (tmp->getName() == name) {

tmp = firstOwner->getNextOwner();

delete firstOwner;

firstOwner = tmp;

if (tmp) { return 2; }

return 1;

}

// list contains only one element

else if (!tmp->getNextOwner()) {

return 0;

}

else {

while (tmp->getNextOwner()) {

if (tmp->getNextOwner()->getName() == name) {

CarOwner\* buff = tmp->getNextOwner()->getNextOwner();

delete tmp->getNextOwner();

tmp->setNextOwner(buff);

if (buff) { return 2; }

return 1;

}

else {

tmp = tmp->getNextOwner();

}

}

}

return 0;

}

unsigned int GetLength() {

if (firstOwner) return firstOwner->GetLength();

else return 0;

}

};

class Ticket

{

int id;

tm timestamp;

string carLicencePlate;

bool isPaid;

public:

// default constructor

Ticket() {};

Ticket(int id) {

this->id = id;

}

// class constructor

Ticket(int id, tm timestamp, string carLicencePlate, bool isPaid = false)

{

this->id = id;

this->timestamp = timestamp;

this->carLicencePlate = carLicencePlate;

this->isPaid = isPaid;

}

// getter for id

int getId() { return id; }

// getter for timestamp

tm getTimestamp() { return timestamp; }

string getTimestampStr() {

return GetTimeAsString("%H:%M %d.%m.%y", timestamp);

}

// getter for licencePlate

string getLicencePlate() { return carLicencePlate; }

// getter for isPaid

bool getIsPaid() { return isPaid; }

// setter for name

void setId(int id) { this->id = id; }

// setter for timestamp

void setTimestamp(tm timestamp) { this->timestamp = timestamp; }

// setter for carLicencePlate

void setCarLicencePlate(string carLicencePlate) { this->carLicencePlate = carLicencePlate; }

// setter for isPaid

void setIsPaid(bool isPaid) { this->isPaid = isPaid; }

};

istream& operator>>(istream& is, Ticket\* ticket) {

string id, carLPlate, isPaid, tmMn, tmH, tmD, tmM, tmY;

if (getline(is, id, ';') &&

getline(is, tmMn, ';') &&

getline(is, tmH, ';') &&

getline(is, tmD, ';') &&

getline(is, tmM, ';') &&

getline(is, tmY, ';') &&

getline(is, carLPlate, ';') &&

getline(is, isPaid)) {

ticket->setId(stoi(id));

ticket->setCarLicencePlate(carLPlate);

ticket->setIsPaid((isPaid) == "1" ? true : false);

ticket->setTimestamp(tm{ 0,stoi(tmMn),stoi(tmH),stoi(tmD),stoi(tmM) - 1,stoi(tmY) - 1900 });

}

return is;

}

ostream& operator<<(ostream& out, Ticket\* ticket) {

if (!ticket) { return out; }

out << "Ticket ID:\t" << ticket->getId()

<< "\nLicence pl:\t" << ticket->getLicencePlate()

<< "\nTimestamp:\t" << ticket->getTimestampStr()

<< "\nIs paid:\t" << (ticket->getIsPaid() ? "yes" : "no");

out << endl;

return out;

}

struct TicketsTreeElement // структура для представления узлов дерева

{

Ticket ticket;

int size;

TicketsTreeElement\* rightChild;

TicketsTreeElement\* leftChild;

TicketsTreeElement(Ticket ticket) {

this->ticket = ticket;

rightChild = 0;

leftChild = 0;

size = 1;

}

};

int GetTreeSize(TicketsTreeElement\* p)

{

if (!p) return 0;

return p->size;

}

void FixTreeSize(TicketsTreeElement\* p)

{

p->size = GetTreeSize(p->leftChild) + GetTreeSize(p->rightChild) + 1;

}

TicketsTreeElement\* FindTicket(TicketsTreeElement\* tree, int id)

{

if (!tree) return 0;

if (id == tree->ticket.getId())

return tree;

if (id < tree->ticket.getId())

return FindTicket(tree->leftChild, id);

else

return FindTicket(tree->rightChild, id);

}

TicketsTreeElement\* RotateTreeRight(TicketsTreeElement\* tree)

{

TicketsTreeElement\* q = tree->leftChild;

if (!q) return tree;

tree->leftChild = q->rightChild;

q->rightChild = tree;

q->size = tree->size;

FixTreeSize(tree);

return q;

}

TicketsTreeElement\* RotateTreeLeft(TicketsTreeElement\* tree)

{

TicketsTreeElement\* p = tree->rightChild;

if (!p) return tree;

tree->rightChild = p->leftChild;

p->leftChild = tree;

p->size = tree->size;

FixTreeSize(tree);

return p;

}

TicketsTreeElement\* TreeInsertToRoot(TicketsTreeElement\* tree, Ticket t)

{

if (!tree) return new TicketsTreeElement(t);

if (t.getId() < tree->ticket.getId())

{

tree->leftChild = TreeInsertToRoot(tree->leftChild, t);

return RotateTreeRight(tree);

}

else

{

tree->rightChild = TreeInsertToRoot(tree->rightChild, t);

return RotateTreeLeft(tree);

}

}

TicketsTreeElement\* TreeRandomisedInsert(TicketsTreeElement\* p, Ticket t)

{

if (!p) return new TicketsTreeElement(t);

if (rand() % (p->size + 1) == 0 && p->ticket.getId() >= 0)

return TreeInsertToRoot(p, t);

if (p->ticket.getId() > t.getId())

p->leftChild = TreeRandomisedInsert(p->leftChild, t);

else

p->rightChild = TreeRandomisedInsert(p->rightChild, t);

FixTreeSize(p);

return p;

}

TicketsTreeElement\* TreeJoinNodes(TicketsTreeElement\* p, TicketsTreeElement\* q)

{

if (!p) return q;

if (!q) return p;

if (rand() % (p->size + q->size) < p->size)

{

p->rightChild = TreeJoinNodes(p->rightChild, q);

FixTreeSize(p);

return p;

}

else

{

q->leftChild = TreeJoinNodes(p, q->leftChild);

FixTreeSize(q);

return q;

}

}

void EmptyTree(TicketsTreeElement\* p) {

if (p->leftChild) {

EmptyTree(p->leftChild);

delete p->leftChild;

p->leftChild = NULL;

}

if (p->rightChild) {

EmptyTree(p->rightChild);

delete p->rightChild;

p->rightChild = NULL;

}

}

TicketsTreeElement\* TreeRemoveNode(TicketsTreeElement\* p, int id)

{

if (!p) return p;

if (p->ticket.getId() == id)

{

TicketsTreeElement\* q = TreeJoinNodes(p->leftChild, p->rightChild);

delete p;

return q;

}

else if (id < p->ticket.getId())

p->leftChild = TreeRemoveNode(p->leftChild, id);

else

p->rightChild = TreeRemoveNode(p->rightChild, id);

return p;

}

struct Trunk

{

Trunk\* prev;

string str;

Trunk(Trunk\* prev, string str)

{

this->prev = prev;

this->str = str;

}

};

void ShowTrunks(Trunk\* p)

{

if (p == nullptr) {

return;

}

ShowTrunks(p->prev);

cout << p->str;

}

void PrintTree(TicketsTreeElement\* root, Trunk\* prev = NULL, bool isLeft = false)

{

if (root == NULL) {

return;

}

string prev\_str = " ";

Trunk\* trunk = new Trunk(prev, prev\_str);

PrintTree(root->rightChild, trunk, true);

if (!prev) {

trunk->str = "---";

}

else if (isLeft)

{

trunk->str = ".---";

prev\_str = " |";

}

else {

trunk->str = "`---";

prev->str = prev\_str;

}

ShowTrunks(trunk);

if (root->ticket.getId() < 0) {

cout << " root" << endl;

}

else {

cout << " " << root->ticket.getId() << endl;

}

if (prev) {

prev->str = prev\_str;

}

trunk->str = " |";

PrintTree(root->leftChild, trunk, false);

}

struct HashTableAnalytics {

unsigned int capacity = 0;

unsigned int population = 0;

unsigned int filling = 0;

unsigned int collisions = 0;

double fillingRate = 0; // will be out of date until GetAnalytics() is called

double collisionsRate = 0; // will be out of date until GetAnalytics() is called

};

class OwnersHashTable {

unsigned int capacity = 100;

CarOwnersList\* arr = NULL;

HashTableAnalytics analytics = HashTableAnalytics();

const double fillingRateUpperLimit = 0.8;

const double collisionsRateUpperlimit = 0.4;

const double fillingRateLowerLimit = 0.1;

public:

OwnersHashTable() {

SetupArray();

}

OwnersHashTable(int capacity) {

if (capacity % 13 == 0) { capacity++; }

int last\_capacity = this->capacity;

this->capacity = capacity;

try {

SetupArray();

}

catch (...) {

this->capacity = last\_capacity;

}

}

void SetupArray() {

if (arr) { throw "there is already an array"; }

arr = new CarOwnersList[capacity];

}

void DeleteArray(bool deleteLists = false) {

if (deleteLists) {

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

arr[i].DeleteList();

}

}

delete arr;

arr = NULL;

analytics = HashTableAnalytics();

}

void InsertToTable(CarOwner\* owner) {

if (owner->getName() == "") throw "wrong name - can't hash";

analytics.population++;

int index = Hash(\*owner);

if (arr[index].InsertToList(owner)) {

analytics.collisions++;

}

else {

analytics.filling += 1;

}

}

void DeleteFromTable(string name) {

int index = Hash(name);

char result = arr[index].DeleteFromList(name);

if (result == 1) {

analytics.population--;

analytics.filling--;

}

else if (result == 2) {

analytics.collisions--;

analytics.population--;

}

}

CarOwnersList GetOwnersByIndex(int index) {

if (index < 0 || index >= capacity) { throw "wrong index"; }

return arr[index];

}

int Hash(string name) {

return Hash(name, capacity);

}

int Hash(CarOwner owner) {

return Hash(owner.getName());

}

static int Hash(string name, int capacity) {

// (ascii - 64) \* 13^(pos%7+1)

size\_t size = name.length();

unsigned int hash = 0;

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

int p = i % 7 + 1;

p = pow(13, p);

int ascii = int(name[i]) - 64;

hash += p \* ascii;

}

return hash % capacity;

}

unsigned int getCapacity() { return capacity; }

HashTableAnalytics GetAnalytics() {

analytics.capacity = this->capacity;

analytics.fillingRate = (float)analytics.filling / analytics.capacity;

analytics.collisionsRate = (float)analytics.collisions / analytics.population;

return analytics;

}

void Rehash(int newCapacity = 0) {

if (newCapacity < 0) throw "wrong capacity";

if (!newCapacity) newCapacity = GetAnalytics().population \* 2;

OwnersHashTable ht = OwnersHashTable(newCapacity);

if (arr) {

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

CarOwner\* owner = arr[i].firstOwner;

while (owner) {

ht.InsertToTable(owner);

owner = owner->getNextOwner();

}

}

DeleteArray();

}

this->arr = ht.getArray();

this->capacity = ht.getCapacity();

}

bool RehashPending() {

HashTableAnalytics an = GetAnalytics();

if (an.fillingRate > fillingRateUpperLimit ||

an.collisionsRate > collisionsRateUpperlimit ||

an.fillingRate < fillingRateLowerLimit) {

return true;

}

return false;

}

CarOwnersList\* getArray() { return arr; }

};

class ConsoleInterface {

public:

static void OutputTicketsList(TicketsList list) {

TicketsListElement\* element = list.firstElement;

while (element) {

cout << element->ticketId << " ";

element = element->nextElement;

}

cout << endl;

}

static void OutputCarOwnersList(CarOwnersList list) {

CarOwner\* element = list.firstOwner;

while (element) {

cout << endl;

cout << element;

element = element->getNextOwner();

}

cout << endl;

}

static void OutputHashTable(OwnersHashTable table) {

int capacity = table.getCapacity();

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

cout << "\n-------------------------------------------------\n\nHash=" << i << endl;

OutputCarOwnersList(table.GetOwnersByIndex(i));

cout << "\n";

}

}

static void OutputCarOwner(CarOwner owner) {

cout << &owner;

}

static void OutputTicketFromTree(TicketsTreeElement\* tree, int ticketId) {

Ticket t = FindTicket(tree, ticketId)->ticket;

cout << "id:\t" << t.getId()

<< "\nplate:\t" << t.getLicencePlate()

<< "\ntime:\t" << t.getTimestampStr()

<< "\npaid?\t" << (t.getIsPaid() ? "yes" : "no")

<< endl;

cout << endl;

}

static void OutputHashTableAnalytics(HashTableAnalytics analytics) {

cout << "Capacity:\t" << analytics.capacity

<< "\nPopulation:\t" << analytics.population

<< "\nFilling:\t" << analytics.filling

<< "\nFilling rate:\t" << analytics.fillingRate

<< "\nCollisions:\t" << analytics.collisions

<< "\nCollisions rate:" << analytics.collisionsRate

<< endl;

}

};

class ConsoleInterfaceMenu {

enum UserScenarios { Initial, Menu };

public:

static void InitialMenu() {

srand(time(0));

setlocale(LC\_ALL, "rus");

ifstream fn;

int fileLength;

TicketsTreeElement\* ticketsTree = new TicketsTreeElement(-1);

OwnersHashTable hashTable = OwnersHashTable();

bool initialised = false;

int scenario;

do {

if (initialised) { // список задан

system("cls");

cout

<< "\nЧто вы хотите сделать?\n"

<< "1. Вывести хэш-таблицу на экран\n"

<< "2. Вывести аналитику таблицы на экран\n"

<< "3. Вывести дерево штрафов на экран\n"

<< "4. Найти в владельца хэш-таблице\n"

<< "5. Добавить владельца в хэш-таблицу\n"

<< "6. Удалить владельца из хэш-таблицы\n"

<< "7. Очистить хэш-таблицу\n"

<< "8. Найти штраф\n"

<< "9. Добавить штраф\n"

<< "10. Удалить штраф\n"

<< "11.Очистить штрафы\n"

<< "12.Выйти\n"

<< "> ";

scenario = InputUsersChoice(Menu);

switch (scenario) {

case 1: {

ConsoleInterface::OutputHashTable(hashTable);

system("pause");

break;

}

case 2: {

ConsoleInterface::OutputHashTableAnalytics(hashTable.GetAnalytics());

system("pause");

break;

}

case 3: {

PrintTree(ticketsTree);

system("pause");

break;

}

case 4: {

bool found = false;

string name;

cout << "Введите имя для поиска > ";

cin >> name;

CarOwner\* co = hashTable.GetOwnersByIndex(hashTable.Hash(name)).firstOwner;

while (co) {

if (co->getName() == name) {

found = true;

cout << "Найдена запись!" << endl << co;

break;

}

co = co->getNextOwner();

}

if (!found) { cout << "Запись с таким именем не найдена.\n"; }

system("pause");

break;

}

case 5: {

cout << "Введите строку информации о водителе в соответствии со стандартом:" << endl;

cout << "name; birthday.day; birthday.month; birthday.year; phoneN; licenceId; licenceExp.day;"

<< "licenceExp.month; licenceExp.year; licencePlate; techPassport; gibddId; tickets" << endl;

CarOwner\* co = new CarOwner();

cin >> co;

hashTable.InsertToTable(co);

cout << "Добавлена запись:\n" << co << endl;

system("pause");

break;

}

case 6: {

string name;

cout << "Введите имя для удаления из таблицы > ";

cin >> name;

hashTable.DeleteFromTable(name);

cout << "Вхождение было удалено\n";

system("pause");

break;

}

case 7: {

hashTable.DeleteArray(true);

cout << "Таблица владельцев была очищена\n";

system("pause");

break;

}

case 8: {

int id;

cout << "Введите ИД штрафа для поиска > ";

cin >> id;

TicketsTreeElement\* t = FindTicket(ticketsTree, id);

if (!t) {

cout << "Элемент не найден\n";

}

else {

cout << "Элемент найден:\n"

<< &(t->ticket);

}

system("pause");

break;

}

case 9: {

cout << "Введите строку информации о штрафе в соответствии со стандартом:" << endl;

cout << "id; timestamp.minute; timestamp.hour; timestamp.day; timestamp.month;"

<< " timestamp.year; licencePlate; isPaid" << endl;

Ticket\* t = new Ticket();

cin >> t;

TreeRandomisedInsert(ticketsTree, \*t);

delete t;

system("pause");

break;

}

case 10: {

int id;

cout << "Введите ИД штрафа для удаления > ";

cin >> id;

TicketsTreeElement\* f = FindTicket(ticketsTree, id);

if (!f) {

cout << "Штраф не найден" << endl;

}

else {

cout << "Удаляется штраф:\n" << &(f->ticket) << endl;

TreeRemoveNode(ticketsTree, id);

cout << "Удаление завершено\n";

}

system("pause");

break;

}

case 11: {

EmptyTree(ticketsTree);

system("pause");

break;

}

}

}

else { // список не задан

system("cls");

cout << "\tЧто вы хотите сделать?\n"

<< "1. Создать пустые структуры\n"

<< "2. Прочитать данные из файлов\n"

<< "3. Выход\n"

<< "> ";

scenario = InputUsersChoice(Initial);

switch (scenario)

{

case 1:

initialised = true;

break;

case 2:

cout << "Введите имя файла с владельцами > ";

string filename;

cin >> filename;

fn.open(filename);

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

cout << "Файл не найден\n";

break;

}

if (fn.eof()) {

cout << "Файл пустой\n";

break;

}

fn >> fileLength;

if (fileLength == 0) {

cout << "В файле нет данных\n";

break;

}

cout << "Количество данных для прочтения: " << fileLength << endl;

hashTable.DeleteArray(true);

hashTable.Rehash(fileLength \* 2);

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

CarOwner\* co = new CarOwner();

fn >> co;

hashTable.InsertToTable(co);

}

fn.close();

cout << "Данные о владельцах прочитаны. Информация о хэш-таблице:\n";

ConsoleInterface::OutputHashTableAnalytics(hashTable.GetAnalytics());

cout << "\n\nВведите имя файла со штрафами > ";

cin >> filename;

fn.open(filename);

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

cout << "Файл не найден\n";

break;

}

if (fn.eof()) {

cout << "Файл пустой\n";

break;

}

fn >> fileLength;

if (fileLength == 0) {

cout << "В файле нет данных\n";

break;

}

cout << "Количество данных для прочтения: " << fileLength << endl;

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

Ticket\* t = new Ticket();

fn >> t;

ticketsTree = TreeRandomisedInsert(ticketsTree, \*t);

delete t;

}

fn.close();

cout << "Данные о штрафах прочитаны\n";

initialised = true;

system("pause");

break;

}

}

} while (scenario != 12 && initialised

|| scenario != 3 && !initialised);

}

static int InputUsersChoice(UserScenarios scenario) {

int variant;

cin >> variant;

if (scenario == Initial) {

while (variant < 1 || variant > 3) {

cout << "Wrong input. Try again > ";

cin >> variant;

}

}

else if (scenario == Menu) {

while (variant < 1 || variant>12) {

cout << "Wrong input. Try again > ";

cin >> variant;

}

}

return variant;

}

};

int main()

{

ConsoleInterfaceMenu::InitialMenu();

}

# Приложение 3

Генератор тестовых наборов данных

from random import randint, choice

from string import ascii\_uppercase as lup

def gen\_lic\_plate():

letters = ["A", "B", "E", "K", "M", "H"]

letters += ["O", "P", "C", "T", "Y", "X"]

plate = choice(letters)

for i in range(3):

plate += str(randint(0,9))

for i in range(2):

plate += choice(letters)

return plate

def gen\_phone():

prefix = ["925", "926", "916", "903", "985", "495"]

num = choice(prefix)+str(randint(1000000, 9999999))

return num

def gen\_ticket(t\_id, plate=""):

ticket = str(t\_id) + ";" # id

ticket += str(randint(0,59))+";" # minutes

ticket += str(randint(0,23))+";" # hours

ticket += str(randint(1,28))+";" # day

ticket += str(randint(1,11))+";" # month

ticket += str(randint(1901,2050))+";"# year

if len(plate)>0:

ticket += plate+";"

else:

ticket += gen\_lic\_plate()+";" # licence plate

ticket += str(randint(0,1)) # isPaid

return ticket

def gen\_owner(name, last\_ticket\_id):

owner = name + ";" # name

owner += str(randint(1,28))+";" # bday day

owner += str(randint(1,11))+";" # bday month

owner += str(randint(1901,2004))+";"# bday year

owner += str(gen\_phone())+";" # phone num

owner += str(randint(0, 10\*\*8))+";" # licence id

owner += str(randint(1,28))+";" # licence day

owner += str(randint(1,11))+";" # licence month

owner += str(randint(2000,2050))+";"# licence year

car\_plate = gen\_lic\_plate()

owner += car\_plate+";" # car plate

owner += choice(lup)+str(randint(0,1000))+";" # tech passport

owner += str(randint(0,100))+";" # gibdd

tickets = []

ticket\_ids = []

for i in range(randint(1,5)):

last\_ticket\_id += 1

tickets += [gen\_ticket(last\_ticket\_id, car\_plate)]

ticket\_ids += [str(last\_ticket\_id)]

owner += " ".join(ticket\_ids)

return (owner, tickets, last\_ticket\_id)

def get\_names():

n = []

with open("c:/users/eldar/desktop/kurs/cleanFirstlast.txt", "r") as file:

n = file.readlines()

return n

def main():

last\_id = 0

owners = []

tickets = []

names = get\_names()

excluded = []

for i in range(200):

name = ""

while len(name)==0 or name in excluded:

name = choice(names)[:-1]

excluded += [name]

generated = gen\_owner(name, last\_id)

owners += [generated[0]]

tickets += generated[1]

last\_id = generated[2]

with open("c:/users/eldar/desktop/ownersFL.txt", "w") as file:

file.write(str(len(owners))+"\n")

for i in range(len(owners)):

file.write(owners[i]+"\n")

with open("c:/users/eldar/desktop/ticketsFL.txt", "w") as file:

file.write(str(len(tickets))+"\n")

for i in range(len(tickets)):

file.write(tickets[i]+"\n")

if \_\_name\_\_ == "\_\_main\_\_":

main()