

## ОГЛАВЛЕНИЕ

ЗАДАНИЕ .....	2
ВВЕДЕНИЕ.....	3
1. Алгоритмы и структуры данных .....	4
2. Описание программы.....	7
3. Тестирование программы.....	9
ЗАКЛЮЧЕНИЕ .....	18
СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ.....	19
ПРИЛОЖЕНИЕ А. ЛИСТИНГ ПРОГРАММНОГО КОДА .....	20

## ЗАДАНИЕ

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

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

Вариант задания на курсовой проект сформирован из следующих составляющих:

- Предметная область - Обслуживание клиентов оператора сотовой связи
- Метод хеширования - Закрытое хеширование с линейным опробованием
- Метод сортировки - Подсчетом
- Вид списка - Линейный однонаправленный
- Метод обхода дерева – Прямой
- Алгоритм поиска слова в тексте - Прямой

## ВВЕДЕНИЕ

Информационная система для предметной области «Обслуживание клиентов оператора сотовой связи» должна осуществлять ввод, хранение, обработку и вывод данных о:

- клиентах;
- SIM-картах, принадлежащих оператору сотовой связи;
- выдаче или возврате SIM-карт клиентами.

Система должна осуществлять следующие операции:

- регистрацию нового клиента;
- снятие с обслуживания клиента;
- просмотр всех зарегистрированных клиентов;
- очистку данных о клиентах;
- поиск клиента по «номер паспорта».
- поиск клиента по фрагментам ФИО или адреса.
- добавление новой SIM-карты;
- удаление сведений о SIM-карте;
- просмотр всех имеющихся SIM-карт;
- очистку данных о SIM-картах;
- поиск SIM-карты по «номеру SIM-карты».
- поиск SIM-карты по тарифу.
- регистрацию выдачи клиенту SIM-карты;
- регистрацию возврата SIM-карты от клиента.

Создание данной информационной системы обусловлено облегчением и автоматизацией рутинных задач, которые выполняет информационная система, в данном случае это обслуживание клиентов сотовой связи. Система так же позволит эффективность хранения данных в принципе, так как для её создания будут использоваться эффективные структуры данных и алгоритмы по их обработке.

## 1. Алгоритмы и структуры данных

Для хранения данных о клиентах используется структура данных АВЛ-дерево. Свойства сбалансированного дерева позволяют быстро находить нужную информацию, которая занесена в дерево. АВЛ-Дерево является двоичным деревом поиска, где каждый узел соответствует правилу: ключ любого узла дерева не меньше любого ключа в левом поддереве данного узла и не больше любого ключа в правом поддереве этого узла. Каждый узел указывает на своего левого и правого потомка.

В программе каждый узел дерева содержит указатель на объект “Клиент”, целочисленное данное, которое хранит высоту поддерева, и указатели на левого и правого потомка.

Для хранения данных о сим-картах оператора сотовой связи используется структура данных Хэш-таблица. Данная структура данных представляет собой таблицу, где каждому значению представлен ключ, получаемый посредством хэш-функции. Таблица позволяет получить значение по ключу в лучшем случае за  $O(1)$ , т.е. за фиксированное время, которое не зависит от размеров таблицы.

В программе таблица имеет 5000 ячеек для хранения указателей на объект “SIM”, для вычисления ключей используется хэш-функция:

$$h(x) = ((int)(x_1 * 1)^2 + (int)(x_2 * 2)^2 + \dots + (int)(x_N * N)^2) \% 5000$$

Каждый код символа строки умножается на его позицию и возводится в квадрат, все вычисления складываются и берётся остаток от деления на 5000. Данная хэш-функция позволяет распределить коллизии (одинаковые ключи разных данных) равномерно по всей структуре данных.

Для хранения данных о получении/возврате сим-карт используется структура данных линейный однонаправленный список. Данная структура данных представляет собой узлы, каждый из которых содержит какое-либо данное и указывает на своего потомка, либо на NULL, если потомка не существует

Преимущество данной структуры заключается в том, что память выделяется только тогда, когда это необходимо.

В программе линейный однонаправленный список организован как узел, состоящий из указателя на объект “Владение SIM”, а также указатель на потомка, либо nullptr, в случае отсутствия потомка.

В работе были реализованы алгоритмы по варианту:

- закрытое хеширование с линейным опробованием;
- сортировка подсчётом;
- прямой обход дерева;
- прямой поиск слова в тексте.

Закрытое хеширование представляет из себя хранение каждого данного в своей определённой хэш-функцией ячейке. В случае возникновения коллизии применяется метод разрешения коллизий под названием “Линейное опробование”, создаётся новый хэш по формуле:

$$h(x) = h(x) + c * i$$

где  $c$  – шаг линейного опробования, а  $i$  – номер итерации повторного создания хэша. В программе  $c$  выбран как 1. В случае если не будет найдено свободной ячейки в хэш-таблице на новом хэше, запускается следующая итерация. Так происходит до тех пор, пока не будет найдена ячейка без данных, либо закончится хэш-таблица, т.е. она будет полностью заполнена.

Сортировка подсчётом представляет из себя сортировку, которая создаёт новую структуру данных, являющейся пустой копией исходной, она и станет результирующей. Для каждого элемента исходной структуры ведётся подсчёт новой позиции в результирующей структуре данных путём счёта элементов, меньше данного, а также равных ему, если они уже были просчитаны алгоритмом.

Прямой обход дерева необходим для просмотра элементов, входящих в АВЛ-Дерево. Из-за нелинейности данной структуры могут использоваться, например, стек и очередь, но в данном случае рассматривается рекурсивный обход дерева. При прохождении дерева в прямом порядке сначала посещается корень, затем левое поддерево и затем правое поддерево.

В алгоритме прямого поиска слова в тексте предполагается что слово и текст будут сравниваться поэлементно. Перед началом работы слово и текст сопоставляется так, что первый символ слова сопоставлен с первым символом текста. Затем начинается сравнение символов, первого символа слова и первого символа текста, второго символа слова и второго символа в тексте и так далее. В случае сопоставления всех символов слова, то слово в тексте найдено. Если в случае сравнения происходит несоответствие символов, то происходит смещение слова так, что его первый символ сопоставляется со вторым символом в тексте. Происходит повторное сравнение символов. Так происходит, пока не будет найдено слово или же не будет достигнут конец текста.

Данный алгоритм достаточно эффективен, если для поиска требуется малое количество смещений. Так же к плюсам алгоритма можно отнести тот факт, что для него не требуется предварительная подготовка текста.

В программе алгоритмы прямого обхода дерева и прямого поиска в тексте используются в связке для поиска определённых данных в АВЛ-дереве.

## 2. Описание программы

Программа, разработанная в процессе курсового проекта, представляет из себя стандартное консольное приложение Visual Studio 2022. У пользователя имеется на выбор 4 кнопки:

- Работа с клиентами;
- Работа с SIM;
- Выдача и возврат;
- Выход.

Каждая из этих групп, не считая пункта “Выход” имеет свои пункты, которые тем или иным образом работают со структурами данных.

Пункт “Работа с клиентами” включает в себя:

1. Зарегистрировать нового клиента;
2. Снять клиента с обслуживания;
3. Список зарегистрированных клиентов;
4. Очистить данные о клиентах;
5. Найти клиента по паспорту;
6. Найти клиентов по ФИО;
7. Найти клиентов по адресу;
8. Назад.

Пункт “Работа с SIM” включает в себя:

1. Добавить новую SIM-карту;
2. Удалить SIM-карту;
3. Список всех SIM-карт;
4. Очистить данные о SIM-картах;
5. Найти SIM-карту по номеру;
6. Найти SIM-карту по тарифу;
7. Назад.

Пункт “Выдача и возврат” включает в себя:

1. Выдать SIM-карту клиенту;
2. Вернуть SIM-карту от клиента;

3. История выдачей и возвратов;

4. Назад.

Для работы с программой необходимо выбирать пункты меню путём ввода номера пункта, который пользователь хочет выбрать. Программа защищена от некорректного выбора и не “падает” при неверных выборах и даёт повторить выбор.

При выборе пункта, связанного с добавлением/удалением/поиском, необходимо вводить клиентские/SIM данные, в этих взаимодействиях производится проверка на заведомо неправильные команды, например, добавление уже существующего элемента, удаление несуществующего и так далее.

Листинг программы приведён в приложении А.



### 3. Тестирование программы

Исходные данные тестирования:

Клиенты представлены в таблице 1.

Таблица 1 – тестовые клиенты

	Номер паспорта	Место и дата выдачи	ФИО	Год рождения	Адрес
1	1419-349234	ГУ МВД России по Красноярскому краю 09.03.2020	Филатов Марк Юрьевич	1980	г.Санкт-Петербург, ул. Макарова, дом 23
2	1419-049234	ГУ МВД России по Краснодарскому краю 03.04.2020	Савельев Богдан Наумович	2000	г. Красноярск, пр. Революции, дом 45
3	4425-433423	ГУ МВД России по г. Санкт-Петербургу и Ленинградской области 09.12.2013	Власов Георгий Пётрович	1973	г. Красноярск, ул. Софийская, дом 3
4	3246-052234	ОП№84 по Приморскому району г. Санкт-Петербурга 04.01.2007	Пономарёв Геннадий Созонович	1987	г. Благовещенск, ул. Советская, дом 6

SIM представлены в таблице 2.

Таблица 2 – Тестовые SIM

	SIM	Тариф	Дата выпуска
1	434-4561284	Базовый	2024
2	334-4561284	Супер	2025
3	494-4589284	Базовый+	2021
4	434-4599984	Базовый	2024
5	434-4568284	Ультра	2024
6	474-4561284	Базовый	2024

\*Признак наличия автоматически ставится true

Тестирование пункта “Работа с клиентами”

Добавление клиентов с попыткой неправильного ввода представлено на рисунке 1.

```
Введите номер паспорта: Ворошилов Даниил Николаевич
Правильный формат: NNNN-NNNNNN
Повторите ввод: 9003-4444444444
Правильный формат: NNNN-NNNNNN
Повторите ввод: 0419-399932
Введите место и дату выдачи паспорта: ГУ МВД России по Красноярскому краю 03.07.2020
Введите ФИО: Ворошилов Даниил Николаевич
Введите год рождения: 0
Повторите ввод: 2005
Введите адрес проживания: г. Санкт-Петербург, пр. Ветеранов, д.2 , кв. 90
Добавлен!

Для продолжения нажмите любую клавишу . . .
```

Рисунок 1 – Добавление клиента

Удаление клиента по паспорту вместе с неверной попыткой показано на рисунке 2 и 3

```
Выберите пункт меню: 2
Введите номер паспорта: 0423-345543
Клиент не найден!

Для продолжения нажмите любую клавишу . . . |
```

Рисунок 2 – Несуществующий номер паспорта

```
Выберите пункт меню: 2
Введите номер паспорта: 0419-399932
Клиент удалён!

Для продолжения нажмите любую клавишу . . .
```

Рисунок 3 – Удаление клиента

На рисунке 4 представлен вывод списка клиентов

```
Выберите пункт меню: 3

Клиент Филатов Марк Юрьевич
Паспорт 1419-349234
Выдан: ГУ МВД России по Красноярскому краю 09.03.2020
Год рождения: 1980
Адрес проживания: г.Санкт-Петербург, ул. Макарова, дом 23

Клиент Савельев Богдан Наумович
Паспорт 1419-049234
Выдан: ГУ МВД России по Краснодарскому краю 03.04.2020
Год рождения: 2000
Адрес проживания: г. Красноярск, пр. Революции, дом 45

Клиент Власов Георгий Пётрович
Паспорт 4425-433423
Выдан: ГУ МВД России по г. Санкт-Петербургу и Ленинградской области 09.12.2013
Год рождения: 1973
Адрес проживания: г. Красноярск, ул. Софийская, дом 3

Клиент Пономарёв Геннадий Созонович
Паспорт 3246-052234
Выдан: ОП №84 по Приморскому району г. Санкт-Петербурга 04.01.2007
Год рождения: 1987
Адрес проживания: г. Благовещенск, ул. Советская, дом 6

Для продолжения нажмите любую клавишу . . . |
```

Рисунок 4 – Список клиентов

Удаление всех клиентов представлено на рисунке 5

```
Выберите пункт меню: 4
Клиент 1419-049234 удалён!
Клиент 3246-052234 удалён!
Клиент 4425-433423 удалён!
Клиент 1419-349234 удалён!

Для продолжения нажмите любую клавишу . . . |
```

Рисунок 5 – Удаление всех клиентов

Поиск клиентов по фрагменту ФИО представлен на рисунке 6

Выберите пункт меню: 6  
Введите ФИО: Ге

Клиент Власов Георгий Пётрович  
Паспорт 4425-433423  
Выдан: ГУ МВД России по г. Санкт-Петербургу и Ленинградской области 09.12.2013  
Год рождения: 1973  
Адрес проживания: г. Красноярск, ул. Софийская, дом 3

Клиент Пономарёв Геннадий Созонович  
Паспорт 3246-052234  
Выдан: ОП №84 по Приморскому району г. Санкт-Петербурга 04.01.2007  
Год рождения: 1987  
Адрес проживания: г. Благовещенск, ул. Советская, дом 6

Для продолжения нажмите любую клавишу . . .

Рисунок 6 – Поиск по ФИО

Поиск клиентов по адресу представлено на рисунке 7

Выберите пункт меню: 7  
Введите адрес: пр

Клиент Савельев Богдан Наумович  
Паспорт 1419-049234  
Адрес проживания: г. Красноярск, пр. Революции, дом 45

Для продолжения нажмите любую клавишу . . . |

Рисунок 7 – Поиск по адресу

Тестирование пункта “Работа с SIM”

На рисунке 8 представлено добавление новой SIM с проверкой на ввод

```
Выберите пункт меню: 1
Введите номер SIM: 3045-3445543
Правильный формат: NNN-NNNNNNN
Повторите ввод: 234-3454345
Выберите тип тарифа:
1. Базовый
2. Базовый+
3. Супер
4. Ультра
3
Введите год выпуска: 2003
SIM добавлена!

Для продолжения нажмите любую клавишу . . .
```

Рисунок 8 – Добавление SIM

Удаление SIM и проверка на наличие представлено на рисунке 9 и 10

```
Выберите пункт меню: 2
Введите номер SIM: 234-4344323
SIM не найдена!

Для продолжения нажмите любую клавишу . . . |
```

Рисунок 9 – Проверка на наличие при удалении

```
Выберите пункт меню: 2
Введите номер SIM: 434-4561284
SIM удалена!

Для продолжения нажмите любую клавишу . . . |
```

Рисунок 10 - Удаление SIM

Вывод всех SIM представлено на рисунке 11

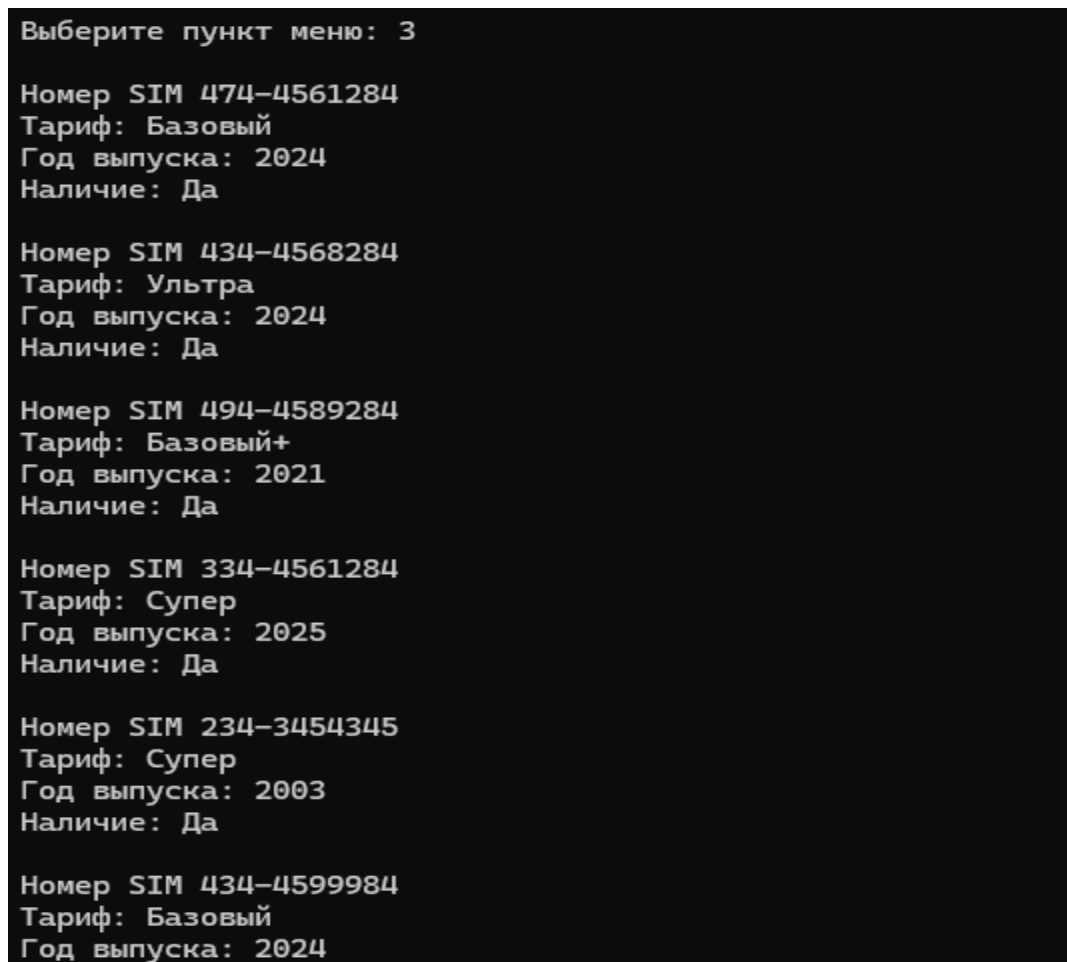


Рисунок 11 – Список всех SIM

Очистка данных SIM представлена на рисунке 12

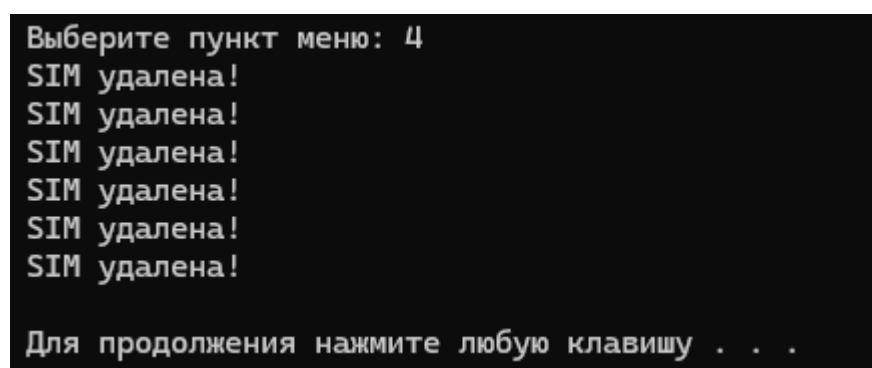


Рисунок 12 – Очистка данных SIM

Поиск SIM по номеру представлено на рисунке 13

```
Выберите пункт меню: 5
Введите номер SIM: 494-4589284
Найденная SIM:

Номер SIM 494-4589284
Тариф: Базовый+
Год выпуска: 2021
Наличие: Да

Для продолжения нажмите любую клавишу . . .
```

Рисунок 13 – Поиск SIM по номеру

Поиск SIM по тарифу, а также проверка на ввод представлены на рисунке

14

```
Выберите пункт меню: 6
Выберите тип тарифа:
1. Базовый
2. Базовый+
3. Супер
4. Ультра
6
Повторите ввод: 3

Номер SIM 334-4561284
Тариф: Супер
Год выпуска: 2025
```

Рисунок 14 – Поиск SIM по тарифу

Тестирование выдачи и возвратов

Выдача клиенту SIM представлена на рисунке 15, а также проверка наличия SIM на рисунке 16, и выводы поиска клиента и SIM на рисунках 17 и 18 соответственно.

4. назад  
Выберите пункт меню: 1  
Введите номер паспорта: 1419-049234  
Введите номер SIM: 494-4589284  
SIM выдана!  
  
Для продолжения нажмите любую клавишу . . . |

Рисунок 15 – Выдача SIM

Номер SIM 494-4589284  
Тариф: Базовый+  
Год выпуска: 2021  
Наличие: Нет

Рисунок 16 – Проверка наличия SIM

Клиент Савельев Богдан Наумович  
Паспорт 1419-049234  
Выдан: ГУ МВД России по Краснодарскому краю 03.04.2020  
Год рождения: 2000  
Адрес проживания: г. Красноярск, пр. Революции, дом 45  
SIM клиента:  
494-4589284

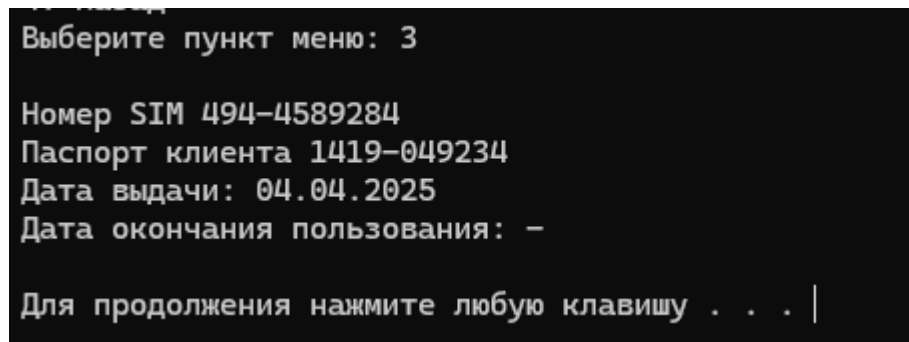
Рисунок 17 – Информация при поиске клиента

Номер SIM 494-4589284  
Тариф: Базовый+  
Год выпуска: 2021  
Наличие: Нет  
Владелец Савельев Богдан Наумович 1419-049234

Рисунок 18 – Информация при поиске SIM

История выдачи до возврата представлена на рисунке 19, после возврата на рисунке 20.



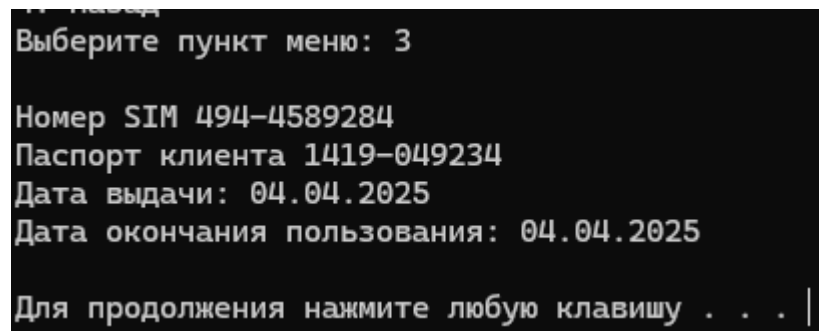


```
Выберите пункт меню: 3

Номер SIM 494-4589284
Паспорт клиента 1419-049234
Дата выдачи: 04.04.2025
Дата окончания пользования: -

Для продолжения нажмите любую клавишу . . . |
```

Рисунок 19 – История получений и возвратов до возврата



```
Выберите пункт меню: 3

Номер SIM 494-4589284
Паспорт клиента 1419-049234
Дата выдачи: 04.04.2025
Дата окончания пользования: 04.04.2025

Для продолжения нажмите любую клавишу . . . |
```

Рисунок 20 – История получений и возвратов после возврата

После тестирования можно сделать вывод, что программа работает исправно и выполняет свои заданные функции хранения и обработки.

## ЗАКЛЮЧЕНИЕ

В процессе выполнения курсового проекта была создана и протестирована информационная система “Обслуживание клиентов оператора сотовой связи” с использованием алгоритмов и структур данных по варианту. При написании программы были закреплены практические навыки создания структур данных и реализации алгоритмов разного рода, которые изучались на дисциплине “Алгоритмы и структуры данных”.

## СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

- 1) Ключарев А.А., Матьяш В.А., Щекин С.В. Структуры и алгоритмы обработки данных: Учебное пособие / ГУАП. СПб., 2004.
- 2) В. А. Матьяш, С. А. Рогачев Алгоритмы и структуры данных. Учебное пособие/ ГУАП. СПб., 2021

## ПРИЛОЖЕНИЕ А. ЛИСТИНГ ПРОГРАММНОГО КОДА

Kursac.cpp

// 665

// Обслуживание клиентов оператора сотовой связи

// Закрытое хеширование с линейным опробованием

// Сортировка подсчётом

// Список Линейный однонаправленный

// Метод обхода прямой

// Алгоритм поиска слова в тексте прямой

#include "AVLTree.h"

#include "Classes.h"

#include "SimOwningList.h"

#include "SimTable.h"

#include "SubFunctions.h"

#include <iostream>

#include <windows.h>

using namespace std;

int main()

{

    setlocale(LC\_ALL, "RUS");

    SetConsoleCP(1251);

    SetConsoleOutputCP(1251);

    Tree\* Root = nullptr;

    SimTable SimInfos;

    SimOwningList\* OwningList = nullptr;

```

        cout << "Добро пожаловать в систему 'Обслуживание клиентов оператора сотовой
связи'" << endl;

        cout << endl;

        int x;

        string Passport;
        string PassportInfo;
        string FIO;
        int YearOfBirth;
        string Adress;

        string SIM;
        int TarrifName;
        int YearOfRelease;

        int count;

        Client* ClientInfo;
        Sim* SimInfo;
        vector<string> ClientSims;

        // Изначальная информация

        // Клиенты

        AddElem(&Root, new Client("1419-349234", "ГУ МВД России по Красноярскому краю
09.03.2020", "Филатов Марк Юрьевич",
1980, "г.Санкт-Петербург, ул. Макарова,
дом 23"));

```

```
AddElem(&Root, new Client("1419-049234", "ГУ МВД России по Краснодарскому краю 03.04.2020",
```

```
    "Савельев Богдан Наумович", 2000, "г. Красноярск, пр. Революции, дом 45"));
```

```
AddElem(&Root, new Client("4425-433423", "ГУ МВД России по г. Санкт-Петербургу и Ленинградской области 09.12.2013",
```

```
    "Власов Георгий Пётрович", 1973, "г. Красноярск, ул. Софийская, дом 3"));
```

```
AddElem(&Root, new Client("3246-052234", "ОП№84 по Приморскому району г. Санкт-Петербурга 04.01.2007",
```

```
    "Пономарёв Геннадий Созонович", 1987, "г. Благовещенск, ул. Советская, дом 6"));
```

```
// SIM
```

```
AddSim(SimInfos, new Sim("434-4561284", Sim::Tarrif::Base, 2024));
```

```
AddSim(SimInfos, new Sim("334-4561284", Sim::Tarrif::Super, 2025));
```

```
AddSim(SimInfos, new Sim("494-4589284", Sim::Tarrif::Medium, 2021));
```

```
AddSim(SimInfos, new Sim("434-4599984", Sim::Tarrif::Base, 2024));
```

```
AddSim(SimInfos, new Sim("434-4568284", Sim::Tarrif::Ultra, 2024));
```

```
AddSim(SimInfos, new Sim("474-4561284", Sim::Tarrif::Base, 2024));
```

```
bool bRunMenu = true;
```

```
bool bRunClient;
```

```
bool bRunSim;
```

```
bool bRunGiveReturn;
```

```
while (bRunMenu)
```

```
{
```

```
    cout << "1. Работа с клиентами" << endl;
```

```
    cout << "2. Работа с SIM" << endl;
```

```
    cout << "3. Выдача и возврат" << endl;
```

```

cout << "4. Выход" << endl;

cout << "Выберите пункт меню: ";

x = GetInt();

switch (x)
{
case 1:
    bRunClient = true;

    while (bRunClient)
    {
        system("cls");

        cout << "1. Зарегистрировать нового клиента" << endl;
        cout << "2. Снять клиента с обслуживания" << endl;
        cout << "3. Список зарегистрированных клиентов" << endl;
        cout << "4. Очистить данные о клиентах" << endl;
        cout << "5. Найти клиента по паспорту" << endl;
        cout << "6. Найти клиентов по ФИО" << endl;
        cout << "7. Найти клиентов по адресу" << endl;
        cout << "8. Назад" << endl;

        cout << "Выберите пункт меню: ";

        x = GetInt();

        switch (x)
        {
case 1:
            // Добавление клиента

```

```

        cout << "Введите номер паспорта: ";

        Passport = GetPassportNo();

        // Проверка на существование
        if (FindClientByPassport(&Root, Passport))
        {
            cout << "Данный клиент уже существует!" << endl;
            break;
        }

        cout << "Введите место и дату выдачи паспорта: ";
        getline(cin, PassportInfo);

        cout << "Введите ФИО: ";
        getline(cin, FIO);

        cout << "Введите год рождения: ";

        // Ввод корректного года
        YearOfBirth = GetValidYear();

        cout << "Введите адрес проживания: ";
        getline(cin, Address);

        // Добавление
        AddElem(&Root, new Client(Passport, PassportInfo, FIO,
YearOfBirth, Address));

        cout << "Добавлен!" << endl;

        break;
    case 2:

```



```

// Удаление по паспорту

cout << "Введите номер паспорта: ";

Passport = GetPassportNo();

ClientInfo = FindClientByPassport(&Root, Passport);

// Проверка на существование

if (ClientInfo)

{

    ClientSims = GetAllSimsOfClient(OwningList,

Passport);

    // Проверка на существование привязанных симок

    if (not(ClientSims.empty()))

    {

        auto first = ClientSims.begin();

        auto last = ClientSims.end();

        for (; first != last; ++first)

        {

            // Нахождение симок по коду и

удаление

            SimInfo = FindSim(SimInfos, *first);

            ReturnSim(OwningList, ClientInfo-

>GetPassport(), SimInfo->GetID());

            SimInfo->SetAvailable(true);

            cout << "SIM " << SimInfo->GetID()

<< " освобождена" << endl;

        }

    }

    // Удаление клиента

    DeleteElem(&Root, Passport);

```

```

        BalanceTree(&Root);

        cout << "Клиент удалён!" << endl;

    }

    else

    {

        cout << "Клиент не найден!" << endl;

    }


    break;

case 3:

    // Список клиентов

    if (Root)

    {

        ShowClients(Root);

    }

    else

    {

        cout << "Информация по клиентам пустая" << endl;

    }


    break;

case 4:

    // Удалить всех клиентов

    ClearAllOfClient(&Root, SimInfos, OwningList);

    break;

case 5:

    // Найти клиента по паспорту

```

```

cout << "Введите номер паспорта: ";

Passport = GetPassportNo();

ClientInfo = FindClientByPassport(&Root, Passport);

if (ClientInfo)
{
    cout << "Найденный клиент: " << endl << endl;
    cout << "Клиент " << ClientInfo->GetFIO() << endl;
    cout << "Паспорт " << ClientInfo->GetPassport() <<
endl;

    cout << "Выдан: " << ClientInfo->GetPassportInfo()
<< endl;

    cout << "Год рождения: " << ClientInfo-
>GetYearBirth() << endl;

    cout << "Адрес проживания: " << ClientInfo-
>GetAdress() << endl;

    // В случае существования привязанных симок -
ВЫВЕСТИ

    ClientSims = GetAllSimsOfClient(OwningList,
Passport);

    if (not(ClientSims.empty()))
    {
        cout << "SIM клиента:" << endl;
        auto first = ClientSims.begin();
        auto last = ClientSims.end();
        for (; first != last; ++first)
        {
            cout << *first << endl;
        }
    }
}

```

```

        }

    }

    else

    {

        cout << "Клиент не найден!" << endl;

    }

    break;

case 6:

    // Поиск клиента по ФИО

    count = 0;

    cout << "Введите ФИО: ";

    getline(cin, FIO);

    PrintAllClientsByFIO(Root, FIO, count);

    if (count == 0)

    {

        cout << "Информации по данному атрибуту не

найдено." << endl;

    }

    break;

case 7:

    // Поиск клиента по адресу

    count = 0;

    cout << "Введите адрес: ";

```

```

        getline(cin, Adress);

        PrintAllClientsByAdress(Root, Adress, count);

        if (count == 0)
        {
            cout << "Информации по данному атрибуту не
найдено." << endl;
        }

        break;

    case 8:
        bRunClient = false;
        break;

    default:
        break;
    }

    cout << endl;
    system("pause");
}

break;

case 2:
    bRunSim = true;
    while (bRunSim)
    {
        system("cls");
        cout << "1. Добавить новую SIM-карту" << endl;

```

```

cout << "2. Удалить SIM-карту" << endl;

cout << "3. Список всех SIM-карт" << endl;

cout << "4. Очистить данные о SIM-картах" << endl;

cout << "5. Найти SIM-карту по номеру" << endl;

cout << "6. Найти SIM-карты по тарифу" << endl;

cout << "7. Назад" << endl;


cout << "Выберите пункт меню: ";

x = GetInt();


switch (x)
{
case 1:

    // Добавление симки

    cout << "Введите номер SIM: ";

    // Валидация на существование

    SIM = GetSimNo();

    if (FindSim(SimInfos, SIM))

    {

        cout << "Данная SIM уже существует!" << endl;

        break;

    }

    cout << "Выберите тип тарифа: " << endl;

    cout << "1. Базовый" << endl;

    cout << "2. Базовый+" << endl;

    cout << "3. Супер" << endl;

    cout << "4. Ультра" << endl;

    TariffName = GetInt(1, 4);

```

```

        cout << "Введите год выпуска: ";

        // Валидация года
        YearOfRelease = GetValidYear();

        // Добавление
        AddSim(SimInfos, new Sim(SIM, Sim::Tarrif(TarrifName),
YearOfRelease));

        cout << "SIM добавлена!" << endl;

        break;

    case 2:

        // Удаление по коду
        cout << "Введите номер SIM: ";

        SIM = GetSimNo();

        SimInfo = FindSim(SimInfos, SIM);

        if (SimInfo)
        {

            // Проверка на существование владельцев

            SimOwning* OwningInfo =
GetSimOwningInfo(OwningList, SimInfo->GetID());

            if (OwningInfo)
            {

                // Если есть владелец, записать возврат сим
карты

                ReturnSim(OwningList, OwningInfo-
>GetPassport(), OwningInfo->GetSimID());

                cout << "Клиент " << OwningInfo-
>GetPassport() << " освобождён от SIM!" << endl;

            }

            // Удалить сим карту

            DeleteSim(SimInfos, SIM);

```

```

        cout << "SIM удалена!" << endl;
    }
    else
    {
        cout << "SIM не найдена!" << endl;
    }
    break;
case 3:
    // Список симок
    if (not(IsEmpty(SimInfos)))
    {
        PrintAllOfSims(SimInfos);
    }
    else
    {
        cout << "Информация по SIM картам пустая" <<
endl;
    }
    break;
case 4:
    // Очистить все симкарты
    ClearAllSims(SimInfos, OwningList);
    break;
case 5:
    // Найти симкарту по коду
    cout << "Введите номер SIM: ";
    SIM = GetSimNo();
    SimInfo = FindSim(SimInfos, SIM);

```



```

        if (SimInfo)
        {
            cout << "Найденная SIM:" << endl << endl;
            cout << "Номер SIM " << SimInfo->GetID() << endl;
            cout << "Тариф: " << SimInfo->TariffToString() <<
endl;

            cout << "Год выпуска: " << SimInfo-
>GetYearRelease() << endl;

            cout << "Наличие: " << SimInfo->IsAvailable() <<
endl;

            // Проверка на владельца
            SimOwning* OwningInfo =
GetSimOwningInfo(OwningList, SimInfo->GetID());
            if (OwningInfo)
            {
                Client* ClientInfo =
FindClientByPassport(&Root, OwningInfo->GetPassport());
                // Вывод владельца при его существовании
                if (ClientInfo)
                {
                    cout << "Владелец " << ClientInfo-
>GetFIO() << " " << ClientInfo->GetPassport() << endl;
                }
            }
        }
    else
    {
        cout << "SIM не найдена!" << endl;
    }
}

```

```

        break;

case 6:

    // Найти симкарты по тарифу

    count = 0;

    cout << "Выберите тип тарифа: " << endl;

    cout << "1. Базовый" << endl;

    cout << "2. Базовый+" << endl;

    cout << "3. Супер" << endl;

    cout << "4. Ультра" << endl;

    TarrifName = GetInt(1, 4);

    PrintAllOfSimsByTarrif(SimInfos, Sim::Tarrif(TarrifName),
count);

    if (count == 0)

    {

        cout << "Информации по данному атрибуту не
найдено." << endl;

    }

    break;

case 7:

    bRunSim = false;

    break;

default:

    break;

}

```

```

        cout << endl;

        system("pause");

    }

    break;

case 3:

    bRunGiveReturn = true;

    while (bRunGiveReturn)
    {

        system("cls");

        cout << "1. Выдать SIM-карту клиенту" << endl;
        cout << "2. Вернуть SIM-карту от клиента" << endl;
        cout << "3. История выдачей и возвратов" << endl;
        cout << "4. Назад" << endl;


        cout << "Выберите пункт меню: ";

        x = GetInt();

        switch (x)
        {

        case 1:

            // Выдача симкарты

            cout << "Введите номер паспорта: ";

            Passport = GetPassportNo();

            ClientInfo = FindClientByPassport(&Root, Passport);

            if (not(ClientInfo))
            {

                cout << "Данного клиента не существует!" << endl;

                break;

```

```

    }

    cout << "Введите номер SIM: ";

    SIM = GetSimNo();

    SimInfo = FindSim(SimInfos, SIM);

    if (not(SimInfo))

    {

        cout << "Данной SIM не существует!" << endl;

        break;

    }

    else if (not(SimInfo->GetAvailable()))

    {

        cout << "Данная SIM уже выдана!" << endl;

        break;

    }

    GiveSim(&OwningList, new SimOwning(ClientInfo-
>GetPassport(), SimInfo->GetID()));

    SimInfo->SetAvailable(false);

    SortList(&OwningList);

    cout << "SIM выдана!" << endl;

    break;

case 2:

    // Возврат симкарты

    cout << "Введите номер паспорта: ";

    Passport = GetPassportNo();

    ClientInfo = FindClientByPassport(&Root, Passport);

    if (not(ClientInfo))

    {

        cout << "Данного клиента не существует!" << endl;

```

```

        break;
    }
    cout << "Введите номер SIM: ";
    SIM = GetSimNo();
    SimInfo = FindSim(SimInfos, SIM);
    if (not(SimInfo))
    {
        cout << "Данной SIM не существует!" << endl;
        break;
    }
    else if (SimInfo->GetAvailable())
    {
        cout << "Данная SIM никому не выдана!" << endl;
        break;
    }
    else if (not(IsClientOwningSim(OwningList, ClientInfo-
>GetPassport(), SimInfo->GetID()))
    {
        cout << "Данная SIM не выдана данному клиенту!"
<< endl;

        break;
    }

    ReturnSim(OwningList, ClientInfo->GetPassport(), SimInfo-
>GetID());

    SimInfo->SetAvailable(true);
    cout << "SIM возвращена!" << endl;
    break;
case 3:

```

```

        // Список выдач и возвратов
        PrintList(OwnningList);

        break;

    case 4:

        bRunGiveReturn = false;

        break;

    default:

        break;

    }

    cout << endl;

    system("pause");

}

break;

case 4:

    bRunMenu = false;

    goto exitMenu;

default:

    break;

}

system("cls");

}

exitMenu:

    // Очистка динамической памяти

    DeleteTree(&Root);

    ClearAllSim(SimInfos);

    ClearList(&OwnningList);

}

```

## AVLTree.cpp

```
#include "AVLTree.h"
#include "SimOwningList.h"
#include "SimTable.h"
#include "math.h"
#include <algorithm>

// Вывод дерева(номеров паспорта)
void ShowTree(Tree* Root, int level)
{
    if (Root != nullptr)
    {
        ShowTree(Root->right, level + 1);
        for (int i = 1; i <= level; i++)
            cout << "          ";
        cout << Root->ClientInfo->GetPassport() << " " << Root->h << endl;
        ShowTree(Root->left, level + 1);
    }
}

// Вывод клиентов(прямой обход)
void ShowClients(Tree* Root)
{
    if (Root == nullptr)
    {
        return;
    }

    cout << endl;
    cout << "Клиент " << Root->ClientInfo->GetFIO() << endl;
    cout << "Паспорт " << Root->ClientInfo->GetPassport() << endl;
    cout << "Выдан: " << Root->ClientInfo->GetPassportInfo() << endl;
    cout << "Год рождения: " << Root->ClientInfo->GetYearBirth() << endl;
    cout << "Адрес проживания: " << Root->ClientInfo->GetAddress() << endl;

    ShowClients(Root->left);
    ShowClients(Root->right);
}

// Очистка памяти от дерева
void DeleteTree(Tree** Root)
{
    if (*Root == nullptr)
    {
        return;
    }

    DeleteTree(&(*Root)->left);
    DeleteTree(&(*Root)->right);
    delete (*Root)->ClientInfo;
    delete (*Root);
    *Root = nullptr;
}

// Получение высоты поддерева
int GetHeight(Tree* Root)
{
    if (Root == nullptr)
    {
        return 0;
    }
    else
    {
        return Root->h;
    }
}
```

```

    }
}

// Исправление высот поддеревьев после балланса
void FixHeight(Tree** Root)
{
    (*Root)->h = max(GetHeight((*Root)->left), GetHeight((*Root)->right)) + 1;
}

// Разница высот поддеревьев
int GetHeightDiff(Tree* Root)
{
    return GetHeight(Root->right) - GetHeight(Root->left);
}

// Малый правый поворот
void RotateRight(Tree** Root)
{
    Tree* left = (*Root)->left;
    (*Root)->left = left->right;
    left->right = (*Root);
    FixHeight(Root);
    FixHeight(&left);
    *Root = left;
}

// Малый левый поворот
void RotateLeft(Tree** Root)
{
    Tree* right = (*Root)->right;
    (*Root)->right = right->left;
    right->left = (*Root);
    FixHeight(Root);
    FixHeight(&right);
    *Root = right;
}

// Балансировка поворотами
void BalanceTree(Tree** Root)
{
    FixHeight(Root);

    if (GetHeightDiff(*Root) == 2)
    {
        if (GetHeightDiff((*Root)->right) < 0)
        {
            RotateRight(&(*Root)->right);
        }
        RotateLeft(Root);
    }
    else if (GetHeightDiff(*Root) == -2)
    {
        if (GetHeightDiff((*Root)->left) > 0)
        {
            RotateLeft(&(*Root)->left);
        }
        RotateRight(Root);
    }
}

// Добавление элемента в дерево
void AddElem(Tree** Root, Client* ClientToAdd)
{
    if (*Root == nullptr)
    {

```



```

        Tree* R = new Tree;
        R->ClientInfo = ClientToAdd;
        R->left = nullptr;
        R->right = nullptr;
        *Root = R;
        return;
    }
    if (ClientToAdd->GetPassport() < (*Root)->ClientInfo->GetPassport())
    {
        AddElem(&((*Root)->left), ClientToAdd);
    }
    else if (ClientToAdd->GetPassport() > (*Root)->ClientInfo->GetPassport())
    {
        AddElem(&((*Root)->right), ClientToAdd);
    }

    BalanceTree(Root);
}

// Поиск минимального элемента в дереве
Tree* FindMin(Tree* Root)
{
    if (Root->left)
    {
        FindMin(Root->left);
    }
    else
    {
        return Root;
    }
}

// Удаление минимального в дереве
void RemoveMin(Tree** Root)
{
    if ((*Root)->left == 0)
    {
        (*Root) = (*Root)->right;
        return;
    }
    RemoveMin(&((*Root)->left));
    BalanceTree(Root);
}

// Удаление элемента в дереве
bool DeleteElem(Tree** Root, string Passport)
{
    if (*Root == nullptr)
    {
        return false;
    }

    if (Passport < (*Root)->ClientInfo->GetPassport())
    {
        return DeleteElem(&((*Root)->left), Passport);
    }
    else if (Passport > (*Root)->ClientInfo->GetPassport())
    {
        return DeleteElem(&((*Root)->right), Passport);
    }
    else
    {
        Tree* Left = (*Root)->left;
        Tree* Right = (*Root)->right;
        delete (*Root)->ClientInfo;
    }
}

```

```

        delete (*Root);
        *Root = nullptr;
        if (!Right)
        {
            (*Root) = Left;
        }
        else
        {
            Tree* Min = FindMin(Right);
            RemoveMin(&Right);
            Min->right = Right;
            Min->left = Left;
            (*Root) = Min;
            BalanceTree(Root);
        }
        return true;
    }
}

// Поиск клиента по паспорту в дереве
Client* FindClientByPassport(Tree** Root, string Passport)
{
    if (*Root == nullptr)
    {
        return nullptr;
    }
    if (Passport < (*Root)->ClientInfo->GetPassport())
    {
        return FindClientByPassport(&((*Root)->left), Passport);
    }
    else if (Passport > (*Root)->ClientInfo->GetPassport())
    {
        return FindClientByPassport(&((*Root)->right), Passport);
    }
    else
    {
        return (*Root)->ClientInfo;
    }

    return nullptr;
}

// Поиск текста прямой
bool FindSubTextInText(string NodeInfo, string ClientInfo)
{
    int all;
    std::transform(NodeInfo.begin(), NodeInfo.end(), NodeInfo.begin(), [](unsigned
char c) { return std::tolower(c); });
    std::transform(ClientInfo.begin(), ClientInfo.end(), ClientInfo.begin(),
                    [](unsigned char c) { return std::tolower(c); });

    for (int i{0}; i < NodeInfo.length() - ClientInfo.length() + 1; i++)
    {
        all = 0;
        for (int j{0}; j < ClientInfo.length(); j++)
        {
            if (NodeInfo[i + j] == ClientInfo[j])
            {
                all += 1;
            }
            else
            {
                break;
            }
        }
    }
}

```

```

        if (all == ClientInfo.length())
        {
            return true;
        }
    }
    return false;
}

// Поиск клиентов по ФИО(прямой обход)
void PrintAllClientsByFIO(Tree* Root, string FIO, int& count)
{
    if (Root == nullptr)
    {
        return;
    }

    if (FindSubTextInText(Root->ClientInfo->GetFIO(), FIO))
    {
        count++;
        cout << endl;
        cout << "Клиент " << Root->ClientInfo->GetFIO() << endl;
        cout << "Паспорт " << Root->ClientInfo->GetPassport() << endl;
        cout << "Выдан: " << Root->ClientInfo->GetPassportInfo() << endl;
        cout << "Год рождения: " << Root->ClientInfo->GetYearBirth() << endl;
        cout << "Адрес проживания: " << Root->ClientInfo->GetAddress() << endl;
    }

    PrintAllClientsByFIO(Root->left, FIO, count);
    PrintAllClientsByFIO(Root->right, FIO, count);
}

// Поиск клиентов по адресу(прямой обход)
void PrintAllClientsByAdress(Tree* Root, string Adress, int& count)
{
    if (Root == nullptr)
    {
        return;
    }

    if (FindSubTextInText(Root->ClientInfo->GetAddress(), Adress))
    {
        count++;
        cout << endl;
        cout << "Клиент " << Root->ClientInfo->GetFIO() << endl;
        cout << "Паспорт " << Root->ClientInfo->GetPassport() << endl;
        cout << "Адрес проживания: " << Root->ClientInfo->GetAddress() << endl;
    }

    PrintAllClientsByAdress(Root->left, Adress, count);
    PrintAllClientsByAdress(Root->right, Adress, count);
}

// Очистка всех клиентов и корректировка информации в остальных структурах(Обратных обход)
void ClearAllofClient(Tree** Root, SimTable& SimInfos, SimOwningList* OwningRoot)
{
    if (*Root == nullptr)
    {
        return;
    }

    ClearAllofClient(&(*Root)->left, SimInfos, OwningRoot);
    ClearAllofClient(&(*Root)->right, SimInfos, OwningRoot);

    Client* ClientInfo = (*Root)->ClientInfo;

```

```

        vector<string> ClientSims = GetAllSimsOfClient(OwnningRoot, ClientInfo-
>GetPassport());
        if (not(ClientSims.empty()))
        {
            auto first = ClientSims.begin();
            auto last = ClientSims.end();
            for (; first != last; ++first)
            {
                Sim* SimInfo = FindSim(SimInfos, *first);
                ReturnSim(OwnningRoot, ClientInfo->GetPassport(), SimInfo->GetID());
                SimInfo->SetAvailable(true);
                cout << "SIM " << SimInfo->GetID() << " освобождена" << endl;
            }
        }

        cout << "Клиент " << ClientInfo->GetPassport() << " удалён!" << endl;

        delete (*Root)->ClientInfo;
        delete (*Root);
        *Root = nullptr;

```

```

}

```

Client.cpp

```

#include "Classes.h"
using namespace std;

// Функции класса клиент

Client::Client(string passport, string passport_info, string FIO, int year_bith,
string adress)
{
    this->Passport = passport;
    this->PassportInfo = passport_info;
    this->FIO = FIO;
    this->YearOfBirth = year_bith;
    this->Adress = adress;
}

string Client::GetPassport() const
{
    return this->Passport;
}

string Client::GetPassportInfo() const
{
    return this->PassportInfo;
}

string Client::GetFIO() const
{
    return this->FIO;
}

int Client::GetYearBirth() const
{
    return this->YearOfBirth;
}

string Client::GetAdress() const
{
    return this->Adress;
}

```

```
}
```

Sim.cpp

```
#include "Classes.h"  
using namespace std;
```

```
Sim::Sim()
```

```
{  
    this->SIM_ID = "-----";  
    this->TarriInfo = Tarri::Base;  
    this->YearRelease = 0;  
    this->Available = false;  
}
```

```
Sim::Sim(string SIM_ID, Sim::Tarri Tarri, int YearRelease)
```

```
{  
    this->SIM_ID = SIM_ID;  
    this->TarriInfo = Tarri;  
    this->YearRelease = YearRelease;  
    this->Available = true;  
}
```

```
string Sim::GetID() const
```

```
{  
    return this->SIM_ID;  
}
```

```
Sim::Tarri Sim::GetTariff() const
```

```
{  
    return this->TarriInfo;  
}
```

```
int Sim::GetYearRelease() const
```

```
{  
    return this->YearRelease;  
}
```

```
bool Sim::GetAvailable() const
```

```
{  
    return Available;  
}
```

```
void Sim::SetAvailable(bool Available)
```

```
{  
    this->Available = Available;  
}
```

```
string Sim::IsAvailable() const
```

```
{  
    switch (Available)  
    {  
        case (true):  
            return "Да";  
        case (false):  
            return "Нет";  
    }  
    return "-";  
}
```

```
string Sim::TariffToString() const
```

```
{  
    switch (TarriInfo)  
    {  
        case (Tarri::Base):
```

```

        return "Базовый";
    case (Tarriif::Medium):
        return "Базовый+";
    case (Tarriif::Super):
        return "Супер";
    case (Tarriif::Ultra):
        return "Ультра";
    }
    return "";
}

```

SimOwning.cpp

```

#include "Classes.h"
#include "SubFunctions.h"
using namespace std;

SimOwning::SimOwning(string Passport, string SimID)
{
    this->Passport = Passport;
    this->SimID = SimID;
    this->DateOfIssue = GetDateByTime(time(0));
    this->DateOfExpiration = "-";
}

SimOwning::SimOwning(string Passport, string SimID, string DateOfIssue, string
DateOfExpiration)
{
    this->Passport = Passport;
    this->SimID = SimID;
    this->DateOfIssue = DateOfIssue;
    this->DateOfExpiration = DateOfExpiration;
}

string SimOwning::GetPassport() const
{
    return this->Passport;
}

string SimOwning::GetSimID() const
{
    return this->SimID;
}

string SimOwning::GetDateOfIssue() const
{
    return this->DateOfIssue;
}

string SimOwning::GetDateOfExpiration() const
{
    return this->DateOfExpiration;
}

void SimOwning::SetDateOfExpiration(string DateOfExpiration)
{
    this->DateOfExpiration = DateOfExpiration;
}

```

SimOwningList.cpp

```

#include "SimOwningList.h"
#include "SubFunctions.h"
#include "AVLTree.h"

```

```

// Очистка памяти
void ClearList(SimOwningList** Root)
{
    if (not(*Root))
    {
        return;
    }

    SimOwningList* cur = nullptr;

    while ((*Root)->next)
    {
        cur = (*Root);
        *Root = (*Root)->next;
        delete cur->OwningInfo;
        delete cur;
    }
    delete *Root;
    *Root = nullptr;
}

// Создание пустого списка размером N
SimOwningList* MakeListOfN(int n)
{
    SimOwningList* OwningList = nullptr;

    for (int i{0}; i < n; i++)
    {
        GiveSim(&OwningList, nullptr);
    }

    return OwningList;
}

// Вывод списка
void PrintList(SimOwningList* Root)
{
    if (not(Root))
    {
        cout << "Информация по выдачам и возвратам пустая" << endl;
        return;
    }

    while (Root)
    {
        if (Root->OwningInfo)
        {
            cout << endl;
            cout << "Номер SIM " << Root->OwningInfo->GetSimID() << endl;
            cout << "Паспорт клиента " << Root->OwningInfo->GetPassport() <<
endl;
            cout << "Дата выдачи: " << Root->OwningInfo->GetDateOfIssue() <<
endl;
            cout << "Дата окончания пользования: " << Root->OwningInfo->
GetDateOfExpiration() << endl;
        }
        Root = Root->next;
    }
}

// Получение длины списка
int GetSize(SimOwningList* Root)
{
    int size = 0;
    while (Root)

```

```

        {
            size++;
            Root = Root->next;
        }
        return size;
    }

// Установить N по счёту элемент на другой
void SetElementInListOnIndex(SimOwningList* Root, int index, SimOwning* OwningInfo)
{
    int ListIndex = 0;
    while (ListIndex != index)
    {
        ListIndex++;
        Root = Root->next;
    }

    Root->OwningInfo = new SimOwning(OwningInfo->GetPassport(), OwningInfo->GetSimID(), OwningInfo->GetDateOfIssue(),
                                     OwningInfo->GetDateOfExpiration());
}

// Получить элемент по порядку(индексу)
SimOwning* GetElementByIndex(SimOwningList* Root, int index)
{
    int ListIndex = 0;
    while (ListIndex != index)
    {
        ListIndex++;
        Root = Root->next;
    }

    return Root->OwningInfo;
}

// Получение смещения для сортировки подсчётом
int GetKForSort(SimOwningList* Root, string SimID, int ListSize, int SourceIndex)
{
    int k = 0;
    for (int i{0}; i < ListSize; i++)
    {
        if (GetElementByIndex(Root, i)->GetSimID() < SimID)
        {
            k++;
        }
        else if (i < SourceIndex and GetElementByIndex(Root, i)->GetSimID() ==
SimID)
        {
            k++;
        }
        else if (i == SourceIndex)
        {
            continue;
        }
    }
    return k;
}

// Сортировка подсчётом
void SortList(SimOwningList** Root)
{
    SimOwningList* NewOwningList = nullptr;
    int ListSize = GetSize(*Root);

```



```

NewOwningList = MakeListOfN(ListSize);

for (int i{0}; i < ListSize; i++)
{
    SimOwning* OwningInfo = GetElementByIndex(*Root, i);

    int k = GetKForSort(*Root, OwningInfo->GetSimID(), ListSize, i);
    SetElementInListOnIndex(NewOwningList, k, OwningInfo);
}

ClearList(Root);
*Root = NewOwningList;
}

// Выдача симкарты клиенту
void GiveSim(SimOwningList** Root, SimOwning* OwningInfo)
{
    SimOwningList* cur = *Root;
    SimOwningList* p = new SimOwningList;

    if (cur == nullptr)
    {
        p->next = nullptr;
        p->OwningInfo = OwningInfo;
        *Root = p;
    }
    else
    {
        while (cur->next)
        {
            cur = cur->next;
        }
        p->next = nullptr;
        p->OwningInfo = OwningInfo;
        cur->next = p;
    }
}

// Возврат симкарты от клиента
bool ReturnSim(SimOwningList* Root, string ClientPassport, string ClientSim)
{
    while (Root and (Root->OwningInfo->GetSimID() != ClientSim or Root->OwningInfo->GetPassport() != ClientPassport))
    {
        Root = Root->next;
    }

    if (Root)
    {
        Root->OwningInfo->SetDateOfExpiration(GetDateByTime(time(0)));
        return true;
    }

    return false;
}

// Владеет ли клиент симкартой
bool IsClientOwningSim(SimOwningList* Root, string ClientPassport, string ClientSim)
{
    while (Root and (Root->OwningInfo->GetSimID() != ClientSim or Root->OwningInfo->GetPassport() != ClientPassport))
    {
        Root = Root->next;
    }
}

```

```

        if (Root)
        {
            return true;
        }

        return false;
    }

    // Получить информацию о владении симкартой
    SimOwning* GetSimOwningInfo(SimOwningList* Root, string ClientSim)
    {
        while (Root and (Root->OwningInfo->GetSimID() != ClientSim or Root->OwningInfo->GetDateOfExpiration() != "-"))
        {
            Root = Root->next;
        }

        if (Root)
        {
            return Root->OwningInfo;
        }
        return nullptr;
    }

    // Получение номеров симкарт клиента
    vector<string> GetAllSimsOfClient(SimOwningList* Root, string ClientPassport)
    {
        vector<string> ClientSims{};
        while (Root)
        {
            if (Root->OwningInfo->GetPassport() == ClientPassport and Root->OwningInfo->GetDateOfExpiration() == "-")
            {
                ClientSims.push_back((Root->OwningInfo->GetSimID()));
            }

            Root = Root->next;
        }

        return ClientSims;
    }
}

```

Simtable.cpp

```

#include "SimTable.h"
#include "SimOwningList.h"

Sim SimTable::DeleteSign = Sim();

// Конструктор хэш-таблицы
SimTable::SimTable()
{
    SimInfo = new Sim*[TableSize];
    for (int i{0}; i < TableSize; i++)
    {
        SimInfo[i] = nullptr;
    }
}

// Деструктор таблицы
SimTable::~~SimTable()
{
    for (int i{0}; i < TableSize; i++)
    {
        if (SimInfo[i] == &SimTable::DeleteSign)

```

```

        {
            continue;
        }
        delete SimInfo[i];
    }
    delete SimInfo;
    SimInfo = nullptr;
}

// Хэш-функция
int GetHashCode(string SIMID)
{
    int hash = 0;
    for (int i{0}; i < SIMID.length(); i++)
    {
        int CharOfID = SIMID[i];
        hash += (int)pow((CharOfID * (i + 1)), 2);
    }
    hash = hash % SimTable::TableSize;

    return hash;
}

// Добавление элемента в таблицу
bool AddSim(SimTable& SimInfos, Sim* SimData)
{
    int SimKey = GetHashCode(SimData->GetID());

    int i = 0;
    while (SimInfos.SimInfo[SimKey] and SimInfos.SimInfo[SimKey] !=
&SimTable::DeleteSign)
    {
        i++;
        if (i >= SimTable::TableSize)
        {
            return false;
        }
        SimKey = (SimKey + 1) % SimTable::TableSize;
    }

    SimInfos.SimInfo[SimKey] = SimData;
    return true;
}

// Удаление элемента
bool DeleteSim(SimTable& SimInfos, string SimID)
{
    int SimKey = GetHashCode(SimID);

    int i = 0;
    while (SimInfos.SimInfo[SimKey])
    {
        if (SimInfos.SimInfo[SimKey]->GetID() == SimID)
        {
            delete SimInfos.SimInfo[SimKey];
            SimInfos.SimInfo[SimKey] = nullptr;
            SimInfos.SimInfo[SimKey] = &SimTable::DeleteSign;
            return true;
        }
        else
        {
            i++;
            if (i >= SimTable::TableSize)
            {
                return false;
            }
        }
    }
}

```

```

        }
        SimKey = (SimKey + 1) % SimTable::TableSize;
    }
}

return false;
}

// Поиск элемента
Sim* FindSim(SimTable& SimInfos, string SimID)
{
    int SimKey = GetHash(SimID);

    int i = 0;
    while (SimInfos.SimInfo[SimKey])
    {
        if (SimInfos.SimInfo[SimKey]->GetID() == SimID)
        {
            return SimInfos.SimInfo[SimKey];
        }
        else
        {
            i++;
            if (i >= SimTable::TableSize)
            {
                return nullptr;
            }
            SimKey = (SimKey + 1) % SimTable::TableSize;
        }
    }

    return nullptr;
}

// Очистка памяти
void ClearAllSim(SimTable& SimInfos)
{
    for (int i{0}; i < SimTable::TableSize; i++)
    {
        if (SimInfos.SimInfo[i] == &SimTable::DeleteSign)
        {
            continue;
        }
        delete SimInfos.SimInfo[i];
        SimInfos.SimInfo[i] = nullptr;
    }
}

// Получение случайного номера симкарты
string GetRandomSimID()
{
    string SIMID = "";

    for (int i{0}; i < 11; i++)
    {
        if (i == 3)
        {
            SIMID += "-";
        }
        else
        {
            SIMID += '0' + rand() % (('9' - '0'));
        }
    }
}

```

```

        return SIMID;
    }

    // Проверка на пустоту таблицы
    bool IsEmpty(SimTable& SimInfos)
    {
        for (int i{0}; i < SimTable::TableSize; i++)
        {
            Sim* Sim = SimInfos.SimInfo[i];
            if ((Sim) and (Sim != &SimTable::DeleteSign))
            {
                return false;
            }
        }
        return true;
    }

    // Вывод всех симкарт
    void PrintAllOfSims(SimTable& SimInfos)
    {
        for (int i{0}; i < SimTable::TableSize; i++)
        {
            Sim* Sim = SimInfos.SimInfo[i];
            if ((Sim) and (Sim != &SimTable::DeleteSign))
            {
                cout << endl;
                cout << "Номер SIM " << Sim->GetID() << endl;
                cout << "Тариф: " << Sim->TariffToString() << endl;
                cout << "Год выпуска: " << Sim->GetYearRelease() << endl;
                cout << "Наличие: " << Sim->IsAvailable() << endl;
            }
        }
    }

    // Вывод всех симкарт по тарифу
    void PrintAllOfSimsByTarrif(SimTable& SimInfos, Sim::Tarrif TarrifInfo, int& count)
    {
        for (int i{0}; i < SimTable::TableSize; i++)
        {
            Sim* Sim = SimInfos.SimInfo[i];
            if ((Sim) and (Sim != &SimTable::DeleteSign) and (Sim->GetTariff() ==
TarrifInfo))
            {
                count++;
                cout << endl;
                cout << "Номер SIM " << Sim->GetID() << endl;
                cout << "Тариф: " << Sim->TariffToString() << endl;
                cout << "Год выпуска: " << Sim->GetYearRelease() << endl;
            }
        }
    }

    // Удаление всех симкарт и корректировка других структур
    void ClearAllSims(SimTable& SimInfos, SimOwningList* OwningList)
    {
        for (int i{0}; i < SimTable::TableSize; i++)
        {
            Sim* SimInfo = SimInfos.SimInfo[i];
            if (not(SimInfo) or (SimInfo == &SimTable::DeleteSign))
            {
                continue;
            }

            SimOwning* OwningInfo = GetSimOwningInfo(OwningList, SimInfo->GetID());

```

```

        if (OwningInfo)
        {
            ReturnSim(OwningList, OwningInfo->GetPassport(), OwningInfo->GetSimID());
        }

        delete SimInfos.SimInfo[i];
        SimInfos.SimInfo[i] = &SimTable::DeleteSign;
        cout << "SIM удалена!" << endl;
    }
}

```

Subfunctions.cpp

```

#include "ctime"
#include <iostream>
#include <string>

using namespace std;

// Перевод времени в удобный формат
string GetDateByTime(time_t arg1)
{
    tm date_struct;
    char s[40];
    localtime_s(&date_struct, &arg1);
    strftime(s, 40, "%d.%m.%Y", &date_struct);

    return s;
}

// Функции валидации данных
bool IsPassportValid(string ID)
{
    if (ID.length() != 11)
    {
        return false;
    }

    for (int i{0}; i < ID.length(); i++)
    {
        if (i == 4)
        {
            if (ID[4] == '-')
                continue;
            else
                return false;
        }
        else if (isdigit(ID[i]))
        {
            continue;
        }
        else
        {
            return false;
        }
    }

    return true;
}

bool IsSimValid(string ID)
{
    if (ID.length() != 11)
    {

```

```

        return false;
    }

    for (int i{0}; i < ID.length(); i++)
    {
        if (i == 3)
        {
            if (ID[3] == '-')
                continue;
            else
                return false;
        }
        else if (isdigit(ID[i]))
        {
            continue;
        }
        else
        {
            return false;
        }
    }

    return true;
}

bool IsYearValid(int year)
{
    if (to_string(year).length() != 4)
    {
        return false;
    }
    return true;
}

// Функции запроса правильных данных

string GetPassportNo()
{
    string x;

    getline(cin, x);

    while (not(IsPassportValid(x)))
    {
        cout << "Правильный формат: NNNN-NNNNNN" << endl;
        cout << "Повторите ввод: ";
        getline(cin, x);
    }

    return x;
}

string GetSimNo()
{
    string x;

    getline(cin, x);

    while (not(IsSimValid(x)))
    {
        cout << "Правильный формат: NNN-NNNNNNNN" << endl;
        cout << "Повторите ввод: ";
        getline(cin, x);
    }
}

```

```

        return x;
    }

    int GetInt(int a = 0, int b = 0)
    {
        bool bInRange = true;
        if (a == b)
        {
            bInRange = false;
        }

        int x;

        cin >> x;

        while (cin.fail() or (cin.peek() != '\n') or (bInRange and not((x >= a and x <=
b))))
        {
            cin.clear();
            cin.ignore(numeric_limits<streamsize>::max(), '\n');
            cout << "Повторите ввод: ";
            cin >> x;
        }

        cin.ignore(numeric_limits<streamsize>::max(), '\n');
        return x;
    }

    int GetValidYear()
    {
        int x;

        cin >> x;

        while (cin.fail() or (cin.peek() != '\n') or not(IsYearValid(x)))
        {
            cin.clear();
            cin.ignore(numeric_limits<streamsize>::max(), '\n');
            cout << "Повторите ввод: ";
            cin >> x;
        }

        cin.ignore(numeric_limits<streamsize>::max(), '\n');
        return x;
    }
}

```

AVLTree.h

```

#pragma once
#include "Classes.h"
#include "SimTable.h"
#include "SimOwningList.h"

// AVL-дерево узел
struct Tree
{
    Client* ClientInfo = nullptr;
    int h = 1;
    Tree* left = nullptr;
    Tree* right = nullptr;
};

void ShowTree(Tree*, int);

void ShowClients(Tree*);

```



```

void DeleteTree(Tree**);

void AddElem(Tree**, Client*);

bool DeleteElem(Tree**, string);

Client* FindClientByPassport(Tree**, string);

void PrintAllClientsByFIO(Tree*, string, int&);

void PrintAllClientsByAdress(Tree*, string, int&);

void ClearAllOfClient(Tree**, SimTable&, SimOwningList*);

```

Classes.h

```

#pragma once
#include <iostream>
using namespace std;

// Класс Клиент
class Client
{
private:
    string Passport;      // Номер паспорта
    string PassportInfo;  // Кто выдал паспорт
    string FIO;           // ФИО
    int YearOfBirth;      // год рождения
    string Adress;        // Адрес проживания

public:
    Client(string, string, string, int, string);
    string GetPassport() const;
    string GetPassportInfo() const;
    string GetFIO() const;
    int GetYearBirth() const;
    string GetAdress() const;
};

// Класс симкарты
class Sim
{
public:
    enum class Tarriif
    {
        Base = 1,
        Medium = 2,
        Super = 3,
        Ultra = 4
    };

private:
    string SIM_ID;        // Номер Симкарты
    Tarriif TarriifInfo;  // Тарриф симкарты
    int YearRelease;      // Год выпуска
    bool Available;       // Доступность

public:
    Sim();
    Sim(string, Tarriif, int);
    string GetID() const;
    Tarriif GetTariff() const;
    int GetYearRelease() const;
    bool GetAvailable() const;
};

```

```

        string IsAvailable() const;
        string TariffToString() const;
        void SetAvailable(bool);
};

// Класс о владении симкарты клиентом
class SimOwning
{
private:
    string Passport;           // Номер паспорта
    string SimID;              // Номер симкарты
    string DateOfIssue;        // Дата выдачи
    string DateOfExpiration;   // Дата возврата

public:
    SimOwning(string, string);
    SimOwning(string, string, string, string);
    string GetPassport() const;
    string GetSimID() const;
    string GetDateOfIssue() const;
    string GetDateOfExpiration() const;
    void SetDateOfExpiration(string);
};

SimOwningList.h

#pragma once
#include "Classes.h"
#include <vector>

// Узел списка
struct SimOwningList
{
    SimOwning* OwningInfo;
    SimOwningList* next;
};

void ClearList(SimOwningList**);

SimOwningList* MakeListOfN(int);

void PrintList(SimOwningList*);

void GiveSim(SimOwningList**, SimOwning*);

bool ReturnSim(SimOwningList*, string, string);

void SortList(SimOwningList**);

bool IsClientOwningSim(SimOwningList*, string, string);

SimOwning* GetSimOwningInfo(SimOwningList*, string);

vector<string> GetAllSimsOfClient(SimOwningList*, string);

SimTable.h

#pragma once
#include "Classes.h"
#include "SimOwningList.h"

// Хэш-таблица
struct SimTable
{
    const static int TableSize = 5000;
};

```

```

        static Sim DeleteSign;
        Sim** SimInfo;
        SimTable();
        ~SimTable();
};

int GetHash(string);

bool AddSim(SimTable&, Sim*);

bool DeleteSim(SimTable&, string);

Sim* FindSim(SimTable&, string);

void ClearAllSim(SimTable&);

string GetRandomSimID();

bool IsEmpty(SimTable&);

void PrintAllOfSims(SimTable&);

void PrintAllOfSimsByTarriif(SimTable&, Sim::Tarriif, int&);

void ClearAllSims(SimTable&, SimOwningList*);

Subfunctions.h

#pragma once
#include "ctime"
#include <string>
using namespace std;

// Перевод времени в удобный формат
string GetDateByTime(time_t arg1);

// Функции валидации данных
bool IsPassportValid(string ID);

bool IsSimValid(string ID);

bool IsYearValid(int year);

// Функции запроса правильных данных

int GetInt(int a=0, int b=0);

int GetValidYear();

string GetPassportNo();

string GetSimNo();

```