

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования

**«Дальневосточный федеральный университет»**

(ДВФУ)

**ИНСТИТУТ МАТЕМАТИКИ И КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ**

**(ШКОЛА)**

**Департамент программной инженерии и искусственного интеллекта**

ПОЖИДАЕВ ДМИТРИЙ АЛЕКСАНДРОВИЧ

РАЗРАБОТКА ИНФОРМАЦИОННОЙ СИСТЕМЫ ДЛЯ РАБОТЫ СО СПРАВОЧНИКАМИ ПРЕДМЕТНОЙ ОБЛАСТИ «ЮРИДИЧЕСКАЯ ФИРМА»

**КУРСОВОЙ ПРОЕКТ**

по дисциплине «Фундаментальные структуры данных и алгоритмы»

по образовательной программе подготовки бакалавров по направлению

09.03.04 «Программная инженерия», профиль «Программная инженерия»

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | | | | | |  |  | Студент группы Б9123-09.03.04 | | | | | | | |
|  |  |  | | | | Пожидаев Д.А. | | | |
|  | | | | | | |  |  | (подпись) | | | |  | | | |
|  | | | | | | | | | | | | | | | | |
|  | | | | | | |  |  | Руководитель | | | | | | | |
|  | | | | | | |  |  | ассистент ДПИиИИ | | | | | | | |
|  | | | | | | |  |  | ученая степень, должность | | | | |  | Решетнев Н.Я. | |
|  | | | | | | |  |  | (подпись) | | | | |  | (ФИО) | |
|  | | | | | | | | | | | | | | | | |
| Регистрационный № | | | | | | |  |  | Защищен с оценкой | | | | | | | |
|  | | | | | | |  |  |  | | | | | | | |
|  | | | |  |  | |  |  | « |  | » |  |  | | | 2025 г. |
| (подпись) | | | |  | (ФИО) | |  |  |  | | | | | | | |
| « |  | » |  | | | 2025 г. |  |  |  | | | | | | | |

г. Владивосток

2025

# Оглавление

[Оглавление 2](#_Toc200743839)

[Введение 3](#_Toc200743840)

[1 Анализ предметной области 4](#_Toc200743841)

[1.1 Объект предметной области 4](#_Toc200743842)

[1.2 Законы ПО 7](#_Toc200743843)

[1.3 Постановка задачи «Формирование отчета «Консультации для клиентов»» 8](#_Toc200743844)

[2 Теоретическая часть 9](#_Toc200743845)

[2.1 Хеш-таблица 9](#_Toc200743846)

[2.1.1 Хеш-функция 10](#_Toc200743847)

[2.1.2 Разрешение коллизий методом открытой адресации 12](#_Toc200743848)

[2.1.3 Пример хеш-таблицы на данных из ПО 14](#_Toc200743849)

[2.2 AVL-дерево 17](#_Toc200743850)

[2.2.1 Динамический односвязный упорядоченный по возрастанию список 19](#_Toc200743851)

[2.2.2 Пример дерева на данных из ПО 19](#_Toc200743852)

[3 Требования к информационной системе 20](#_Toc200743853)

[3.1 Требования к данным 20](#_Toc200743854)

[3.1.1 Требования к входным данным 20](#_Toc200743855)

[3.1.2 Требования к выходным данным 20](#_Toc200743856)

[3.2 Функциональные требования 21](#_Toc200743857)

[3.2.1 Общие требования 21](#_Toc200743858)

[3.2.2 Требования для работы со справочником «Клиенты» 22](#_Toc200743859)

[3.2.3 Требования для работы со справочником «Консультации» 23](#_Toc200743860)

[4 Реализация 24](#_Toc200743861)

[4.1 Спецификация структур данных 24](#_Toc200743862)

[4.2 Описание среды разработки 24](#_Toc200743863)

[4.3 Руководство пользователя 24](#_Toc200743864)

[4.3.1 Работа со справочником «\_\_\_\_\_\_\_» 24](#_Toc200743865)

[4.4 Тестирование 25](#_Toc200743866)

[Заключение 26](#_Toc200743867)

[Список литературы 27](#_Toc200743868)

# Введение

Эффективное управление персоналом и задачами является неотъемлемой частью деятельности современных юридических фирм. В условиях постоянно меняющегося законодательства, возрастающего количества судебных процессов и разнообразия правовых вопросов особенно важно иметь четкую систему учета сотрудников, а также механизм контроля за выполнением порученных дел. Отсутствие такой системы может привести к дезорганизации рабочих процессов, нарушению сроков подачи документов и судебных заседаний, а также снижению качества оказываемых услуг. Поэтому разработка специализированного программного решения для автоматизации данных процессов становится актуальной задачей.

Целью курсового проекта является: разработка информационной системы для работы со справочниками предметной области «Юридическая фирма».

Для достижения поставленной цели необходимо выполнить следующие задачи:

1. Провести анализ предметной области «Юридическая фирма».

2. Изучить теоретические основы методов построения справочников.

3. Определить требования к информационной системе.

4. Реализовать информационную систему и провести тестирование.

# 1 Анализ предметной области

Требуется разработать информационную систему для работы со справочниками предметной области (ПО) «Юридическая фирма».

Система должна решать следующие задачи:

1. хранить, позволять просматривать, добавлять и удалять информацию о клиентах (ФИО, ИНН, телефон) и консультациях (ИНН клиента, тема, юрист, дата); позволять искать информацию о клиентах и консультациях для них;
2. формировать и позволять просматривать список консультаций для клиентов в заданную дату, с заданным юристом и заданным ФИО клиента;
3. проверять целостность информации, представленной в справочниках.

1.1 Объект предметной области

Исходя из задач, которые должна решать информационная система, выделены объекты ПО, представленные в Таблице 1.

Таблица 1 – Объекты ПО

|  |  |
| --- | --- |
| Тип и название объекта | Описание объекта |
| Справочник «Клиенты» | Хранит информацию о клиентах |
| Справочник «Консультации» | Хранит информацию о консультациях для каждого клиента |
| Отчет «Консультации для клиентов» | Хранит информацию о каждой консультации в заданную дату, с заданным юристом и заданным ФИО клиента |

Каждый клиент характеризуется следующими параметрами: ИНН, ФИО, телефон.

**ИНН** – целое 12-значное число.

**ФИО** – аббревиатура, расшифровывается как: фамилия, имя, отчество. Фамилия, имя и отчество являются словами, состоящими из букв русского алфавита, первая буква каждого слова – заглавная, остальные – строчные. Слова разделены одним пробелом.

**Телефон** – строка, состоящая из 11 символов цифр, начинающаяся с комбинации символов "89".

Пример справочника «Клиенты» представлен в таблице 2.

Таблица 2 – Справочник «Клиенты»

|  |  |  |  |
| --- | --- | --- | --- |
| **Индекс** | **ИНН** | **ФИО** | **Телефон** |
| 0 | 770123456789 | Иванов Иван Иванович | 89123456789 |
| 1 | 500234567890 | Петрова Анна Сергеевна | 89234567890 |
| 2 | 230345678901 | Смирнов Алексей Дмитриевич | 89345678901 |
| 3 | 660456789012 | Кузнецова Екатерина Владимировна | 89456789012 |
| 4 | 780567890123 | Васильев Михаил Андреевич | 89567890123 |
| 5 | 160678901234 | Орлова Дарья Павловна | 89678901234 |
| 6 | 050789012345 | Федоров Георгий Николаевич | 89789012345 |
| 7 | 420890123456 | Морозова Ксения Леонидовна | 89890123456 |
| 8 | 990901234567 | Григорьев Артем Юрьевич | 89901234567 |
| 9 | 331012345678 | Ковалева Анастасия Игоревна | 89012345678 |
| 10 | 391238912389 | Лебедев Артемий Викторович | 89112223344 |
| 11 | 821389218391 | Соколов Максим Романович | 89998887766 |

Каждая консультация характеризуется следующими параметрами: ИНН клиента, тема, юрист, дата.

**ИНН клиента** – целое 12-значное число.

**Тема** – строка, содержащая от 10 до 255 букв русского алфавита, знаков препинания и символов пробела.

**Юрист** – ФИО юриста, принимающего клиента. ФИО – аббревиатура, расшифровывается как: фамилия, имя, отчество. Фамилия, имя и отчество являются словами, состоящими из букв русского алфавита, первая буква каждого слова – заглавная, остальные – строчные. Слова разделены одним пробелом.

**Дата** – строка, состоящая из 3 целых чисел, разделенных символом точки. Первое число – день, изменяется от 1 до 31 включительно; второе число – месяц, изменяется от 1 до 12 включительно; Третье число – год, изменяется от 1970 до 2038 включительно. В случае, если второе число равняется «4», 6», 9» или «11», первое число не может превышать «30». В случае, если второе число равняется «2», а третье число делится нацело на 100 и не делится нацело на 400, первое число не может превышать «29». В случае, если второе число равняется «2», а третье число делится нацело на 4 и не делится нацело на 100, первое число не может превышать «28».

Пример справочника «Консультации» представлен в таблице 3.

Таблица 3 – Справочник «Консультации»

|  |  |  |  |
| --- | --- | --- | --- |
| **ИНН клиента** | **Тема** | **Юрист** | **Дата** |
| 770123456789 | Анализ судебной практики по спорам о защите прав потребителей в сфере розничной торговли. | Иванова Анна Петровна | 15.03.2023 |
| 770123456789 | Правовые аспекты заключения и расторжения договоров аренды нежилых помещений: актуальные вопросы. | Иванова Анна Петровна | 22.11.2022 |
| 230345678901 | Особенности рассмотрения корпоративных споров в арбитражных судах Российской Федерации. | Сидорова Елена Игоревна | 01.07.2023 |
| 660456789012 | Проблемы правового регулирования электронной коммерции и их решение в российском законодательстве. | Кузнецов Дмитрий Александрович | 10.04.2024 |
| 780567890123 | Защита интеллектуальных прав на программное обеспечение: современное состояние и перспективы. | Васильева Ольга Константиновна | 28.02.2023 |
| 160678901234 | Комментарий к изменениям в Гражданском кодексе РФ, касающимся обязательственного права. | Смирнов Иван Георгиевич | 05.09.2022 |
| 050789012345 | Анализ судебной практики по делам о банкротстве юридических лиц: ключевые тенденции. | Николаева Мария Валерьевна | 19.01.2024 |
| 420890123456 | Правовые основы международного коммерческого арбитража и его применение в России. | Козлов Андрей Михайлович | 12.06.2023 |
| 990901234567 | Ответственность за нарушение законодательства о персональных данных: актуальные вопросы правоприменения. | Морозова Светлана Викторовна | 07.10.2022 |
| 331012345678 | Исследование практики применения антимонопольного законодательства в сфере информационных технологий. | Волков Максим Евгеньевич | 25.08.2023 |
| 391238912389 | Вопросы наследования бизнеса и активов. | Петровский Игорь Семенович | 11.11.2023 |
| 821389218391 | Консультация по трудовому спору. | Волков Максим Евгеньевич | 30.05.2024 |
| 770123456789 | Составление брачного договора. | Иванова Анна Петровна | 02.06.2024 |

Каждый отчет характеризуется следующими параметрами: дата, юрист, ФИО.

**Дата** – строка, состоящая из 3 целых чисел, разделенных символом точки. Первое число – день, изменяется от 1 до 31 включительно; второе число – месяц, изменяется от 1 до 12 включительно; Третье число – год, изменяется от 1970 до 2038 включительно. В случае, если второе число равняется «4», 6», 9» или «11», первое число не может превышать «30». В случае, если второе число равняется «2», а третье число делится нацело на 100 и не делится нацело на 400, первое число не может превышать «29». В случае, если второе число равняется «2», а третье число делится нацело на 4 и не делится нацело на 100, первое число не может превышать «28».

**Юрист** – ФИО юриста, принимающего клиента. ФИО – аббревиатура, расшифровывается как: фамилия, имя, отчество. Фамилия, имя и отчество являются словами, состоящими из букв русского алфавита, первая буква каждого слова – заглавная, остальные – строчные. Слова разделены одним пробелом.

**ФИО** – ФИО клиента, принимаемого юристом. ФИО – аббревиатура, расшифровывается как: фамилия, имя, отчество. Фамилия, имя и отчество являются словами, состоящими из букв русского алфавита, первая буква каждого слова – заглавная, остальные – строчные. Слова разделены одним пробелом.

Таблица 4 – Отчет «Консультации для клиентов»

|  |  |  |
| --- | --- | --- |
| **Дата** | **Юрист** | **ФИО** |
| 15.03.2023 | Иванова Анна Петровна | Иванов Иван Иванович |
| 22.11.2022 | Иванова Анна Петровна | Иванов Иван Иванович |
| 01.07.2023 | Сидорова Елена Игоревна | Смирнов Алексей Дмитриевич |
| 10.04.2024 | Кузнецов Дмитрий Александрович | Кузнецова Екатерина Владимировна |
| 28.02.2023 | Васильева Ольга Константиновна | Васильев Михаил Андреевич |
| 05.09.2022 | Смирнов Иван Георгиевич | Орлова Дарья Павловна |
| 19.01.2024 | Николаева Мария Валерьевна | Федоров Георгий Николаевич |
| 12.06.2023 | Козлов Андрей Михайлович | Морозова Ксения Леонидовна |
| 07.10.2022 | Морозова Светлана Викторовна | Григорьев Артем Юрьевич |
| 25.08.2023 | Волков Максим Евгеньевич | Ковалева Анастасия Игоревна |

1.2 Законы ПО

1. ИНН в справочнике «Клиенты» уникален.
2. Клиент из справочника «Клиенты» может иметь одну или несколько консультаций, либо не иметь их вовсе;
3. Каждая консультация из справочника «Консультации» должна иметь клиента с соответствующим ИНН в справочнике «Клиенты».
4. Все поля, помеченные как "обязательные", не должны содержать пустые строки или значения NULL

1.3 Постановка задачи «Формирование отчета «Консультации для клиентов»»

Входные данные: Справочник «Клиенты», Справочник «Консультации», ФИО клиента, юрист, дата консультации.

Выходные данные: Отчет «Консультации для клиентов».

Связь:

Отчет «Консультации для клиентов».ИНН = Справочник «Клиенты».ИНН

Отчет «Консультации для клиентов».ФИО = Справочник «Клиенты».ФИО

Отчет «Консультации для клиентов».Юрист = Справочник «Консультации».Юрист

Если существует консультация Справочник «Клиенты».ФИО = ФИО клиента & Справочник «Консультации».Юрист = Юрист & Справочник «Консультации».Дата = Дата консультации.

# 2 Теоретическая часть

В рамках курсового проекта необходимо осуществлять поиск информации о консультациях для клиентов на основании ИНН клиентов.

Для хранения данных о клиентах используется массив, значениями в котором является запись с полями «ИНН», «ФИО», «Телефон». Для поиска данных о клиентах в Справочнике «Клиенты» будет использоваться структура данных динамическая «Хеш-таблица» со статусами 0/1/2. Для реализации хеш-таблицы будут применяться: хеш-функция вида «середина квадрата», квадратичная вторичная хеш-функция для разрешения коллизий и метод открытой адресации.

Для хранения данных о консультациях в Справочнике «Консультации» будем использовать структуру данных «AVL-дерево». Дерево будет содержать в качестве значений вершин линейный односвязный динамический список, построение дерева по ключу вида «ИНН клиента».

Для формирования отчёта «Консультации для клиентов» будет использоваться структура данных «AVL-дерево». Дерево будет содержать в качестве значений вершин линейный односвязный динамический список, построение дерева по ключу вида «Дата».

(тут указываете, какие СД будут использоваться для хранения данных справочника, какие СД будут использоваться для быстрого выполнения поиска и по каким ключам они будут построены. Ниже описываете теорию по этим СД).

2.1 Хеш-таблица

Хеш-таблица представляет собой эффективную структуру данных, реализующую интерфейс ассоциативного массива и подходящую для реализации справочников. По своей сути, это массив, где местоположение каждого элемента определяется значением его ключа с помощью специальной хеш-функции. Возможность прямой индексации элементов массива обеспечивает доступ к произвольному элементу массива за время O(1). Основные операции, которые хеш-таблица позволяет выполнять, включают добавление новой пары "ключ-значение", поиск значения по ключу и удаление пары по ключу [1].

Хеширование — класс методов поиска, идея которого состоит в вычислении хеш-кода, однозначно определяемого элементом с помощью хеш-функции.

2.1.1 Хеш-функция

Хеш-функция — это математический алгоритм, оформленный в виде подпрограммы, который принимает входные данные произвольного размера (ключ) и преобразует их в выходное значение фиксированного размера, называемое хеш-значением. Это хеш-значение затем используется как индекс для размещения соответствующего значения в хеш-таблице.

В данной работе используется метод середины квадрата. Этот метод преобразует значение ключа в число, затем это число возводится в квадрат, из него выбираются несколько средних цифр и интерпретируются, как адрес записи. Он показывает хорошие результаты, когда размер хеш-таблицы является простым числом, а ключи распределены равномерно.

(Теоретический материал со ссылками на литературу, что такое ХФ [номер из списка литературы].

Теоретический материал про метод хеширования, который будете использовать.)

ДОПИСАТЬ

Пусть есть некоторый элемент с ключом key. При хешировании этот элемент будет храниться в ячейке с номером h(k), где h – сама хеш-функция. Функция h отображает совокупность ключей U на ячейки хеш-таблицы T[0..m-1][1].

В рамках курсового проекта ключом хеш-таблицы является строковое поле – название студенческого отряда. Для этого поля значение будет кодом соответствующего символа в UNICODE.

Предположим, что размер хеш-таблицы составляет 10. Вычислим значение хеш-функции для записей из модели предметной области.

Рассмотрим 10 примеров:

h(770123456789) = 7 + 7 + 0 + 1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 = 59. Квадрат суммы: 3481. Середина квадрата с учетом размера таблицы: 4.

h(500234567890) = 5 + 0 + 0 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 0 = 49. Квадрат суммы: 2401. Середина квадрата с учетом размера таблицы: 4.

h(230345678901) = 2 + 3 + 0 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 0 + 1 = 48. Квадрат суммы: 2304. Середина квадрата с учетом размера таблицы: 3.

h(660456789012) = 6 + 6 + 0 + 4 + 5 + 6 + 7 + 8 + 9 + 0 + 1 + 2 = 54. Квадрат суммы: 2916. Середина квадрата с учетом размера таблицы: 9.

h(780567890123) = 7 + 8 + 0 + 5 + 6 + 7 + 8 + 9 + 0 + 1 + 2 + 3 = 56. Квадрат суммы: 3136. Середина квадрата с учетом размера таблицы: 1.

h(160678901234) = 1 + 6 + 0 + 6 + 7 + 8 + 9 + 0 + 1 + 2 + 3 + 4 = 47. Квадрат суммы: 2209. Середина квадрата с учетом размера таблицы: 2.

h(050789012345) = 0 + 5 + 0 + 7 + 8 + 9 + 0 + 1 + 2 + 3 + 4 + 5 = 44. Квадрат суммы: 1936. Середина квадрата с учетом размера таблицы: 9.

h(420890123456) = 4 + 2 + 0 + 8 + 9 + 0 + 1 + 2 + 3 + 4 + 5 + 6 = 44. Квадрат суммы: 1936. Середина квадрата с учетом размера таблицы: 9.

h(990901234567) = 9 + 9 + 0 + 9 + 0 + 1 + 2 + 3 + 4 + 5 + 6 + 7 = 55. Квадрат суммы: 3025. Середина квадрата с учетом размера таблицы: 0.

h(331012345678) = 3 + 3 + 1 + 0 + 1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 = 43. Квадрат суммы: 1849. Середина квадрата с учетом размера таблицы: 8.

Заметим, что в рассмотренном примере, «770123456789» и «500234567890» имеют одинаковое значение хеш-функции равное 4, а «660456789012», «050789012345», «420890123456» равное 9. Возникшая ситуация называется коллизией – ситуацией, когда при разных входных данных хеш-функция возвращает одинаковый хеш-код.

Для разрешения коллизий существует несколько методов: метод цепочек, метод открытой адресации, пакетирование. В рамках курсового проекта будет использоваться метод открытой адресации.

2.1.2 Разрешение коллизий методом открытой адресации

Открытая адресация — это метод разрешения коллизий в хеш-таблицах, при котором все элементы хранятся непосредственно в самой хеш-таблице. Каждая ячейка в этой хеш-таблице может находиться в одном из трех состояний:

Пустая (статус 0)

Занятая (статус 1)

Удаленная (статус 2)

При поиске элемента последовательно проверяются ячейки таблицы до тех пор, пока не будет найден искомый элемент или не будет установлено его отсутствие.

Для вставки при открытой адресации последовательно исследуются (пробируются) ячейки хеш-таблицы до тех пор, пока не будет найдена пустая ячейка для размещения вставляемого ключа. Вместо фиксированного порядка исследования ячеек, последовательность исследуемых ячеек зависит от вставляемого ключа.

Особенность операции удаления в открытой адресации заключается в том, что простое опустошение слота после удаления ключа может нарушить корректность последующих операций поиска. Если ключ, который был удален, находился на пути пробирования для другого ключа (то есть, поиск другого ключа должен был пройти через этот слот), то после его удаления поиск мог бы ошибочно остановиться, посчитав, что искомый ключ отсутствует.

Для предотвращения таких ошибок, удаленные слоты специально помечаются как "удаленные". Это позволяет операции вставки использовать такой слот для размещения нового элемента. Однако, при поиске, алгоритм не останавливается на "удаленном" слоте; он продолжает пробирование так, как если бы слот был занят, пока не найдет искомый ключ или действительно пустой слот.

Метод квадратичного пробирования является одной из стратегий открытой адресации, разработанной для эффективного разрешения коллизий. Формула для квадратичного пробирования определяется следующим образом: , где — это исследуемая позиция в хэш-таблице для ключа k на i-м шаге; — вспомогательная хэш-функция, определяющая начальную позицию; — положительные вспомогательные константы; — номер исследования, принимает значения от 0 до (m – 1); — размер хэш-таблицы. В рамках данной работы используются константы 3 и 4 соответственно.

В хеш-таблицах, использующих открытую адресацию, производительность операций (поиск, вставка, удаление) критически зависит от коэффициента заполнения α = n/m, где n — число ключей, m — размер таблицы. При высоких значениях α (стремящихся к 1) ожидаемое время выполнения операций возрастает до O(1/(1-α)), что приводит к существенному уменьшению производительности.

Для поддержания эффективности при росте числа элементов, когда α превышает заранее заданный коэффициент максимальной заполненности (например, 0.75), выполняется процесс, который включает в себя создание новой хеш-таблицы большего размера (обычно вдвое большего или равного следующему простому числу), а также рехеширование и перемещение всех существующих элементов из старой таблицы в новую с учетом нового размера [2].

Аналогичные операции происходят при удалении: так же вводится коэффициент заранее заданный коэффициент минимальной заполненности (например, 0.25), выполняется процесс, который включает в себя создание новой хеш-таблицы меньшего размера (обычно вдвое меньше или равного предыдущему простому числу) а также рехеширование и перемещение всех существующих элементов из старой таблицы в новую с учетом нового размера.

Теоретический материал про данный метод разрешения коллизий.

Если метод открытой адресации, указываете, какой статус ячейки хранится, что происходит при удалении. Если таблица динамическая, что происходит при заполненности и т.д.

Рисуете пример построенной ХТ для справочников п. 1.1.

2.1.3 Пример хеш-таблицы на данных из ПО

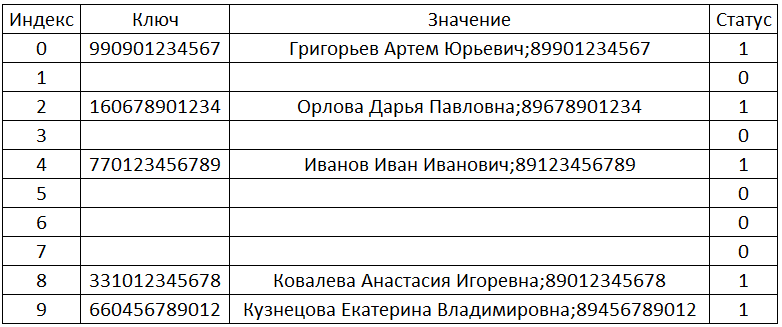
В рамках курсового проекта в хеш-таблице хранится ключ «ИНН» и значение «ФИО;Телефон». На рисунке 1 представлен пример для справочника «Клиенты».

Рисунок 1 - пример хеш-таблицы для справочника «Врачи»

Добавим запись с ключом ИНН «780567890123» и значением «Васильев Михаил Андреевич;89567890123». Значение первичной хеш-функции для ИНН 780567890123 равно 1 (см. пункт 2.1.1). Ячейка с индексом 1 в хеш-таблице свободна. Добавляем запись (см. рисунок 2).

Рисунок 2 – хеш-таблица после вставки без коллизий

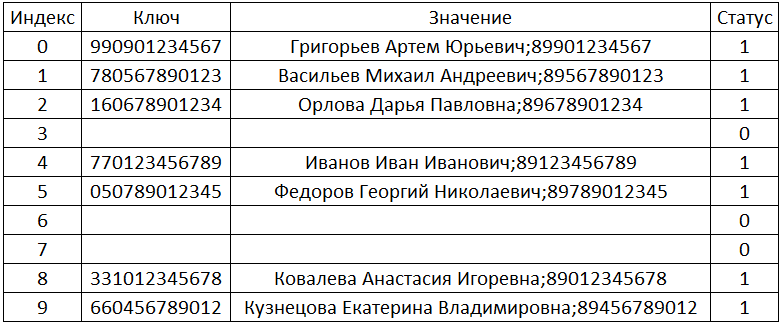
Для добавления записи с ключом «500234567890» и значением «Петрова Анна Сергеевна;89234567890» необходимо разрешить коллизию с элементом, имеющим ключ «770123456789». Используем вторичную хеш-функцию. При первой попытке разрешить коллизию j = 1. . Ячейка занята, повторяем вызов хеш-функции с j = 2. . Ячейка занята, повторяем вызов хеш-функции с j = 3. . Ячейка занята, повторяем вызов хеш-функции с j = 4. . Ячейка свободна. Коллизия устранена, добавляем новую запись под индексом 5 (см. рисунок 3).

Рисунок 3 – хеш-таблица для справочника «Клиенты» после вставки с решением коллизий

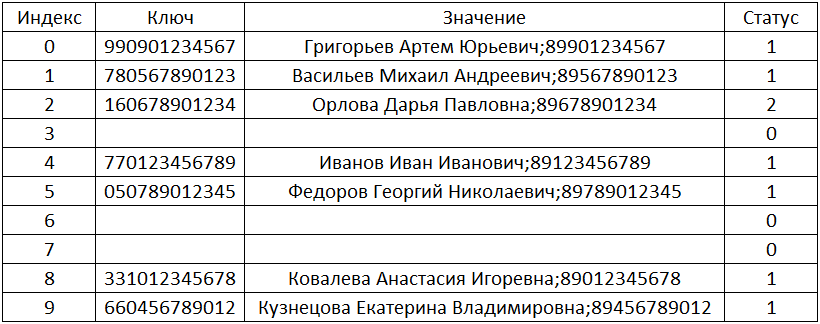
Удалим запись, у которой отсутствуют коллизии при вставке и последующие элементы не зависят от нее при поиске. Например, рассмотрим элемент с ключом «160678901234». Первичная хеш-функция для этого ИНН равна 2. Меняем статус ячейки на «2» (см. рисунок 4). 

Рисунок 4 – удаление из хеш-таблицы без коллизий

Удалим запись с ключом «770123456789». Первичная хеш-функция для этого ИНН равна 4. Эта запись была добавлена по своему первичному хешу, но после нее были вставлены элементы, которые столкнулись с коллизией с индексом 4, а затем искали свободное место, используя квадратичное пробирование. Например, «500234567890» имеет первичный хеш 4, но в итоге был размещен по индексу 6. Если мы просто удалим элемент «770123456789», то элемент «500234567890» может стать "недостижимым" при поиске.

Следовательно, при удалении элемента с ключом «770123456789», который является частью цепочки коллизий (или предшествует ей), необходимо пометить удаляемую ячейку статусом "2". Это позволяет алгоритму поиска проходить через эту ячейку, не останавливаясь и при этом она считается свободной для вставки новых элементов.

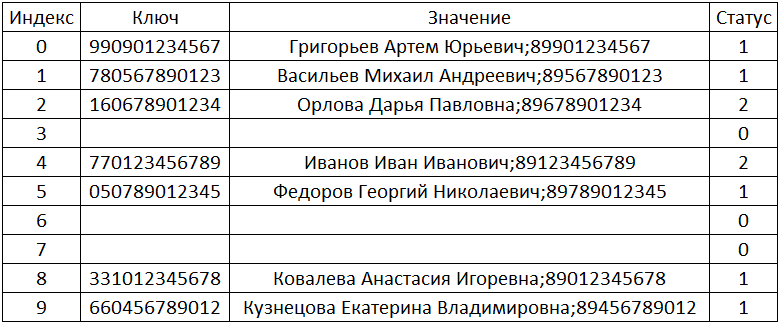
Запись с ключом «770123456789» имеет коллизию, следовательно, при удалении необходимо заменить удаляемый элемент на последнюю коллизию. 

Рисунок 5 – удаление из хеш-таблицы с коллизиями

Поиск записи в хеш-таблице осуществляется ключу. Найдём элемент с ключом «331012345678». Вызываем первичную хеш-функцию, она будет равна 8. Обращаемся к строке 8 хеш-таблицы, найденный в ней элемент соответствует заданному. В результате поиска получаем индекс 8.

При попытке найти элемент с ключом «770123456789» обратимся к строке 4, она имеет статус «2». По методу разрешения коллизий дальше сравниваем последовательно с каждым элементом. Пройдя всю таблицу, совпадение найдено не будет. Получим индекс -1.

Рисуете пример построенной ХТ для справочников п. 1.1, как выполняется поиск, что происходит после добавления и удаления.

2.2 AVL-дерево

АВЛ-дерево, названное в честь изобретателей Адельсона-Вельского и Ландиса, представляет собой самобалансирующееся бинарное дерево поиска (БДП), характеризующееся строгим свойством баланса: для любого узла разность высот его левого и правого поддеревьев (фактор баланса) ограничена интервалом [-1, 1]. Отклонение фактора баланса за пределы данного интервала вызывает автоматическое выполнение операций ротации для восстановления сбалансированного состояния.

В отличие от стандартных БДП, подверженных деградации производительности до O(n) в худшем случае при последовательной вставке отсортированных элементов, АВЛ-дерево поддерживает логарифмическую высоту O(log n) за счет механизма балансировки. Это гарантирует временную сложность O(log n) для всех основных операций, включая поиск, вставку и удаление, независимо от порядка поступления данных. АВЛ-дерево сохраняет при этом инварианты БДП: ключи левого поддерева меньше ключа узла, правого — больше.

Основные операции на АВЛ-дереве:

1. **Поиск:** Итеративное или рекурсивное сравнение искомого ключа с ключом текущего узла с последующим переходом в соответствующее поддерево. Временная сложность O(log n) благодаря сбалансированности.
2. **Вставка:** Новый узел добавляется как лист, аналогично стандартной вставке в БДП. После вставки осуществляется восходящий обход от вставленного узла к корню (retracing). Для несбалансированных узлов (фактор баланса > 1 или < -1) выполняются соответствующие операции ротации (левая, правая, лево-правая, право-левая) для восстановления AVL-свойства.
3. **Удаление:** Реализуется как стандартное удаление в БДП. Узел с двумя потомками заменяется своим непосредственным предшественником или преемником, который затем удаляется. После удаления производится проверка и, при необходимости, балансировка восходящим обходом от поддерева изменения к корню. В отличие от вставки, удаление может требовать каскадных ротаций на предках.

В этом параграфе пишете про БДП, АВЛ или КЧ. Если БДП с ХТ, значит переносите в 2.1.

Теоретический материал про дерево + операции работы (для удаления указать на что замена).

Рисуете пример построенного дерева для справочника п. 1.1.

2.2.1 Динамический односвязный упорядоченный по возрастанию список

Для списка определены операции добавления, удаления и поиска. Добавление записи осуществляется при помощи поиска места вставки с сохранением порядка возрастания. Список может содержать повторяющиеся элементы. Операции удаления и поиска выполняются путем последовательной проверки всех элементов списка для нахождения необходимого элемента по заданному ключу. Каждый узел списка, помимо данных, содержит указатель на следующий узел. Указатель на следующий узел хвостового элемента имеет значение NULL.

Если в элементе дерева хранится цепочка, то добавится этот пункт. Теоретический материал что из себя представляет цепочка в дереве.

2.2.2 Пример дерева на данных из ПО

В рамках курсового проекта в дереве хранится указываете какой ключ и значение. На рисунке номер представлен пример для справочника «Название».

Рисуете пример построенной дерева для справочников п. 1.1, как выполняется поиск, что происходит после добавления и удаления.

И аналогично для дерева, которое используется при фильтрации для формирования отчета.

# 3 Требования к информационной системе

В данной главе описываются требования к информационной системе «Юридическая фирма», а именно: требования к входным данным системы, требования к ее выходным данным и требования к ее функционалу.

3.1 Требования к данным

3.1.1 Требования к входным данным

Основываясь на анализе ПО, входными данными является:

* текстовый файл «clients.txt», каждая строка которого содержит информацию о клиенте из Справочника «Клиенты»: ИНН, ФИО, телефон, разделенные символом «;»;

Пример текстового файла:

770123456789;Иванов Иван Иванович;89123456789

500234567890;Петрова Анна Сергеевна;89234567890

230345678901;Смирнов Алексей Дмитриевич;89345678901

660456789012;Кузнецова Екатерина Владимировна;89456789012

780567890123;Васильев Михаил Андреевич;89567890123

160678901234;Орлова Дарья Павловна;89678901234

050789012345;Федоров Георгий Николаевич;89789012345

420890123456;Морозова Ксения Леонидовна;89890123456

990901234567;Григорьев Артем Юрьевич;89901234567

331012345678;Ковалева Анастасия Игоревна;89012345678

391238912389;Лебедев Артемий Викторович;89112223344

821389218391;Соколов Максим Романович;89998887766

* текстовый файл «consultations.txt», каждая строка которого содержит информацию о консультации: ИНН клиента, тема, юрист, дата, разделенные символом «;»;

Пример текстового файла:

770123456789;Анализ судебной практики по спорам о защите прав потребителей.;Иванова Анна Петровна;15.03.2023

770123456789;Правовые аспекты расторжения договоров аренды нежилых помещений.;Иванова Анна Петровна;22.11.2022

230345678901;Особенности рассмотрения корпоративных споров в арбитражных судах.;Сидорова Елена Игоревна;01.07.2023

660456789012;Проблемы правового регулирования электронной коммерции в РФ.;Кузнецов Дмитрий Александрович;10.04.2024

780567890123;Защита интеллектуальных прав на программное обеспечение.;Васильева Ольга Константиновна;28.02.2023

160678901234;Комментарий к изменениям в Гражданском кодексе РФ.;Смирнов Иван Георгиевич;05.09.2022

050789012345;Анализ судебной практики по делам о банкротстве юридических лиц.;Николаева Мария Валерьевна;19.01.2024

420890123456;Правовые основы международного коммерческого арбитража в России.;Козлов Андрей Михайлович;12.06.2023

990901234567;Ответственность за нарушение законодательства о персональных данных.;Морозова Светлана Викторовна;07.10.2022

331012345678;Исследование практики применения антимонопольного законодательства.;Волков Максим Евгеньевич;25.08.2023

391238912389;Вопросы наследования бизнеса и активов.;Петровский Игорь Семенович;11.11.2023

821389218391;Консультация по трудовому спору.;Волков Максим Евгеньевич;30.05.2024

770123456789;Составление брачного договора.;Иванова Анна Петровна;02.06.2024

3.1.2 Требования к выходным данным

Выходными данными являются:

* текстовый файл «clients\_output.txt», каждая строка файла содержит информацию об объекте «клиент» из Справочника «Клиент», а именно ИНН, ФИО, Телефон, разделенные символом «;». Разделителем внутри поля ФИО служит символ пробела;
* текстовый файл «consultations\_output.txt», каждая строка файла содержит информацию об объекте «консультация» из Справочника «Консультации», а именно ИНН клиента, Тема, Юрист, Дата разделенные символом «;». Разделителем внутри поля Юрист служит символ пробела;
* окно графического интерфейса, содержащее информацию из отчёта «Консультации для клиентов»;
* сообщения об ошибках должны выводиться в отдельной форме и содержать информацию об ошибке. Всевозможные сообщения об ошибках приведены ниже:

1. "Неверное поле: Фамилия";
2. "Неверное поле: Имя";
3. "Неверное поле: Отчество";
4. "Неверное поле: Номер телефона";
5. "Неверное поле: Наименование отряда";
6. "Неверное поле: Список профессий";
7. "Неверное поле: Категория";
8. "Неверное поле: Профессия";
9. "Запись с таким ключом уже есть";
10. "Не удалось удалить";
11. "Неудачно"

3.2 Функциональные требования

3.2.1 Общие требования

Информационная система «Юридическая фирма» должна позволять:

1. считывать данные для справочника «Клиенты» из текстового файла «clients.txt» в массив, который выводиться в виде таблицы в графическом интерфейсе;
2. считывать данные для справочника «Консультации» из текстового файла «consultations.txt» в массив, который выводиться в виде таблицы в графическом интерфейсе
3. сохранять данные справочника «Клиенты» в текстовый файл «clients\_output.txt»;
4. сохранять данные справочника «Консультации» в текстовый файл «consultations\_output.txt»
5. работать со справочниками «Клиенты» и «Консультации» при помощи графического интерфейса;
6. проверять корректность вводимых пользователем данных для справочников «Клиенты» и «Консультации» (см. п. 1.1);
7. формировать список для отчёта «Консультации для клиентов» и выводить в окно графического интерфейса;
8. проверять целостность данных, хранимых в справочниках.

3.2.2 Требования для работы со справочником «Клиенты»

Информационная система «Юридическая фирма» должна позволять:

* 1. Хранить информацию о клиентах: ИНН, ФИО, Телефон;
  2. Просматривать информацию о клиентах, используя графический интерфейс информационной системы;
  3. Осуществлять поиск информации о клиенте на основе его ИНН. Поиск должен осуществляться через графический интерфейс с помощью полей для ввода данных. Если в справочнике отсутствует искомая запись, программа возвращает соответствующее сообщение. Иначе выводится сообщение о нахождении нужного клиента;
  4. Добавлять информацию о клиенте в справочник «Клиенты». Добавление должно осуществляться через графический интерфейс с помощью полей для ввода данных. Каждая запись уникальна, при наличии добавляемой записи в справочнике выводится сообщение об ошибке;
  5. Удалять информацию о клиенте из справочника «Клиенты». Удаление должно осуществляться через графический интерфейс с помощью полей для ввода данных. После удаления должен быть проверен справочник «Консультации» на наличие записей с удалённым клиентом, в случае нахождения записи должны быть удалены. Если в справочнике «Клиенты» изначально не был найден удаляемый клиент, система возвращает соответствующее сообщение;

3.2.3 Требования для работы со справочником «Консультации»

Информационная система «Юридическая фирма» должна позволять:

1. Хранить информацию о консультациях: ИНН клиента, Тема, ФИО юриста, Дата;
2. Просматривать информацию о консультациях, используя графический интерфейс информационной системы;
3. Осуществлять поиск информации о консультациях на основе ИНН клиента. Поиск должен осуществляться через графический интерфейс с помощью полей для ввода данных. Если в справочнике отсутствуют искомые записи, программа возвращает соответствующее сообщение. Иначе выводится сообщение о нахождении нужных консультаций;
4. Добавлять информацию о консультации в справочник «Консультации». Добавление должно осуществляться через графический интерфейс с помощью полей для ввода данных. Каждая запись уникальна, при наличии добавляемой записи в справочнике выводится сообщение об ошибке. Если в справочнике «Клиенты» не существует клиента, указанного в добавляемой консультации, возвращается сообщение об ошибке;
5. Удалять информацию о консультации из справочника «Консультации». Удаление должно осуществляться через графический интерфейс с помощью полей для ввода данных. Если в справочнике не была найдена удаляемая консультация, система возвращает соответствующее сообщение;

# 4 Реализация

В данной главе содержится спецификация реализованных структур данных, описание графического интерфейса, а также приведены результаты тестирования структур данных.

4.1 Спецификация структур данных

Тут описание СД (можно в табличном виде)

**LinkedList** - класс, описывающий односвязный упорядоченный по возрастанию динамический список.

Поля:

* int data: Целочисленное значение, представляющее индекс соответствующей записи в CustomVector<Consultation>;
* ListNode\* next: Указатель на следующий узел в списке.

Методы:

* ListNode\* init(), инициализирует пустой список, нет параметров, нет входных данных, возвращает nullptr, представляющий пустой список;
* void dispose(ListNode\*& head), освобождает всю память, занятую элементами списка, ListNode\*& head - ссылка на указатель на голову списка, нет входных данных, список полностью очищен, память освобождена, head устанавливается в nullptr;
* void addSorted(ListNode\*& head, int value), добавляет новый элемент в список с сохранением упорядоченности по возрастанию, ListNode\*& head - ссылка на указатель на голову списка; int value - целочисленное значение (индекс записи) для добавления, value для добавления, элемент value добавлен в список в соответствующую позицию;
* void removeAll(ListNode\*& head, int value), удаляет все вхождения заданного значения из списка, ListNode\*& head - ссылка на указатель на голову списка; int value - значение для удаления, value для удаления, все узлы, содержащие value, удалены из списка;
* bool find(ListNode\* head, int value), ищет заданный элемент по значению в списке, ListNode\* head - указатель на голову списка; int value - искомое значение, value для поиска, true, если элемент найден; false в противном случае;
* string toString(ListNode\* head) const, преобразует содержимое списка в строковое представление, ListNode\* head - указатель на голову списка, нет входных данных, строка, содержащая значения всех элементов списка, разделенные запятыми.

**Item** - структура, описывающая запись хеш-таблицы.

Поля:

* unsigned long long key: Ключ элемента (ИНН клиента);
* int index: Индекс связанной записи в исходном массиве CustomVector<Client>;
* int status: Статус ячейки (0 - свободна, 1 - занята, 2 - удалена).

**HashTable** - класс, динамическую хеш-таблицу с открытой адресацией и квадратичным пробированием.

Поля:

* Item\* table: Указатель на динамический массив узлов хеш-таблицы.
* int capacity: Текущая емкость (размер) хеш-таблицы.
* double size: Текущее количество занятых элементов в таблице.
* const double MAX\_FILL\_FACTOR: Максимальный коэффициент заполнения, при котором происходит расширение таблицы (0.75).
* const double MIN\_FILL\_FACTOR: Минимальный коэффициент заполнения, при котором происходит сжатие таблицы (0.25).
* const int INIT\_CAPACITY: Начальная емкость таблицы.
* const int k1, k2: Константы для квадратичного пробирования (равны 3 и 4 соответственно).

Методы:

* unsigned long long keyToNum(const unsigned long long& key) const, вспомогательная функция для преобразования 12-значного ИНН в промежуточное числовое значение (сумму цифр), используемое в первичной хеш-функции, const unsigned long long& key - ИНН клиента, 770123456789, 59.
* unsigned long long primaryHash(const unsigned long long& key) const, вычисляет первичный хеш-адрес для ключа методом "середина квадрата", const unsigned long long& key - ИНН клиента, key = 770123456789, key = 500234567890, key = 660456789012, 14 (для 770123456789), 6 (для 500234567890), 6 (для 660456789012).
* int secondaryHash(int initHash, int attempt) const, вычисляет следующий хеш-адрес при коллизии с использованием квадратичного пробирования, int initHash - результат первичной хеш-функции; int attempt - номер попытки пробирования, initHash = 6, attempt = 1 или initHash = 6, attempt = 2, 13 (для attempt=1), 11 (для attempt=2).
* bool add(const unsigned long long& key, int index), добавляет новую пару ключ-значение (ИНН-индекс записи) в хеш-таблицу. Включает логику разрешения коллизий и динамического рехеширования, const unsigned long long& key - ИНН клиента; int index - индекс клиента в CustomVector, key и index для добавления, true при успешном добавлении; false, если ключ уже существует или не удалось найти свободную ячейку после всех попыток.
* bool remove(const unsigned long long& key), удаляет элемент из хеш-таблицы по его ключу. Ячейке присваивается статус "удалена" (2). Включает логику динамического рехеширования при снижении заполненности, const unsigned long long& key - ИНН клиента, key для удаления, true при успешном удалении; false, если элемент не найден.
* bool updateIndex(const unsigned long long& key, int new\_index), обновляет индекс, связанный с существующим ключом в хеш-таблице, const unsigned long long& key - ИНН клиента; int new\_index - новый индекс, key и new\_index для обновления, true при успешном обновлении; false, если ключ не найден.
* void resize(bool isExpands), изменяет размер хеш-таблицы (рехеширование) путем создания новой таблицы и перемещения в нее всех существующих элементов, bool isExpands - флаг, указывающий, нужно ли расширять (true) или сжимать (false) таблицу, нет входных данных, таблица изменена в размере и перестроена.
* const Item\* search(const unsigned long long& key) const, ищет элемент в хеш-таблице по его ключу, const unsigned long long& key - ИНН клиента, key для поиска, указатель на найденную структуру Item или nullptr, если элемент не найден.
* string toString() const, возвращает строковое представление текущего состояния хеш-таблицы, включая статус каждой ячейки и содержимое занятых/удаленных ячеек, нет параметров, нет входных данных, форматированная строка с деталями хеш-таблицы.

**TreeNode** - структура, описывающая узел самобалансирующегося бинарного дерева поиска (AVL-дерева).

Поля:

* TKey key: Ключ узла дерева (может быть quint64 для ИНН клиента или Date для фильтрации по дате).
* LinkedList indexList: Объект класса LinkedList, который хранит индексы записей в CustomVector<Consultation>, связанных с данным ключом.
* ListNode\* head: Указатель на голову LinkedList (для удобства использования LinkedList).
* int balanceFactor: Фактор баланса узла (-1, 0, 1), отражающий разницу высот левого и правого поддеревьев.
* TreeNode<TKey>\* left: Указатель на левого потомка.
* TreeNode<TKey>\* right: Указатель на правого потомка.

**AVLTree** - класс, самобалансирующееся бинарное дерево поиска (AVL-дерево).

Поля:

* TreeNode<TKey>\* balanceL(TreeNode<TKey>\* p, bool& h), выполняет балансировку (вращения) левого поддерева для восстановления AVL-свойства, когда правое поддерево стало слишком тяжелым, TreeNode<TKey>\* p - текущий узел; bool& h - флаг, указывающий, изменилась ли высота дерева, нет входных данных, указатель на корень сбалансированного поддерева.
* TreeNode<TKey>\* balanceR(TreeNode<TKey>\* p, bool& h), выполняет балансировку (вращения) правого поддерева для восстановления AVL-свойства, когда левое поддерево стало слишком тяжелым, TreeNode<TKey>\* p - текущий узел; bool& h - флаг, указывающий, изменилась ли высота дерева, нет входных данных, указатель на корень сбалансированного поддерева.
* TreeNode<TKey>\* del(TreeNode<TKey>\* r, TreeNode<TKey>\* q, bool& h), вспомогательный метод для поиска и удаления максимального элемента в левом поддереве (или минимального в правом) при удалении узла с двумя потомками. Копирует ключ и список индексов в узел q, TreeNode<TKey>\* r - текущий узел в поддереве поиска; TreeNode<TKey>\* q - узел, в который копируются данные удаляемого; bool& h - флаг изменения высоты, нет входных данных, указатель на корень поддерева после операции.
* TreeNode<TKey>\* deleteNode(TreeNode<TKey>\* p, const TKey& key, int indexList, bool& h), рекурсивное удаление элемента из дерева. Если ключ совпадает, удаляет indexList из внутреннего списка. Если список становится пустым, удаляет узел из дерева и выполняет балансировку, TreeNode<TKey>\* p - текущий узел; const TKey& key - ключ для поиска; int indexList - индекс записи для удаления из связанного списка; bool& h - флаг изменения высоты, key и indexList для удаления, указатель на корень поддерева после удаления и балансировки.
* void remove(const TKey& key, int index), удаляет запись из AVL-дерева по ключу и индексу, const TKey& key - ключ узла; int index - индекс записи в списке узла, key и index для удаления, запись удалена из дерева/списка.
* TreeNode<TKey>\* insert(TreeNode<TKey>\* p, const TKey& key, int index, bool& h), рекурсивная вставка нового элемента в дерево. Если ключ уже существует, элемент добавляется в связанный список. Выполняет балансировку после вставки, TreeNode<TKey>\* p - текущий узел; const TKey& key - ключ для вставки; int index - индекс записи для добавления в связанный список; bool& h - флаг изменения высоты, key и index для вставки, указатель на корень поддерева после вставки и балансировки.
* void add(const TKey& key, int index), добавляет новый элемент в AVL-дерево, const TKey& key - ключ узла; int index - индекс записи в списке узла, key и index для добавления, элемент добавлен в дерево/список.
* TreeNode<TKey>\* search(TreeNode<TKey>\* node, const TKey& key), рекурсивный поиск узла в дереве по заданному ключу, TreeNode<TKey>\* node - текущий узел для поиска; const TKey& key - искомый ключ, key для поиска, указатель на найденный TreeNode или nullptr, если узел не найден.
* TreeNode<TKey>\* find(const TKey& key), ищет узел по ключу, const TKey& key - искомый ключ, key для поиска, указатель на найденный TreeNode или nullptr.
* void freeTree(TreeNode<TKey>\* node), освобождает всю память, занятую деревом (удаление всех узлов), TreeNode<TKey>\* node - текущий узел, нет входных данных, вся память дерева освобождена.
* void initializeTree(AVLTree<TKey>& tree), инициализирует дерево (делает его пустым), AVLTree<TKey>& tree - ссылка на дерево, которое нужно инициализировать, нет входных данных, дерево становится пустым.
* string toString(TreeNode<TKey>\* node, int indent = 0), генерирует строковое представление структуры дерева для визуального отображения, TreeNode<TKey>\* node - текущий узел; int indent - уровень отступа для форматирования, нет входных данных, строка, представляющая дерево в удобочитаемом виде.

**CustomVector** - класс, описывающий динамический массив.

Поля:

* T\* data: Указатель на внутренний массив, в котором хранятся элементы.
* size\_t currentSize: Текущее количество элементов, фактически хранящихся в векторе.
* size\_t currentCapacity: Текущая общая емкость внутреннего массива (сколько элементов он может вместить без перевыделения памяти).

Методы:

* void append(const T& value), добавляет новый элемент в конец вектора. При необходимости (если currentSize достигает currentCapacity) увеличивает емкость внутреннего массива, const T& value - добавляемый элемент, value для добавления, элемент добавлен в вектор.
* T& operator[](size\_t index) / const T& operator[](size\_t index) const, предоставляет прямой доступ к элементу по указанному индексу без проверки границ, size\_t index - индекс элемента, нет входных данных, ссылка на элемент по указанному индексу.
* const T& at(size\_t index) const, предоставляет безопасный доступ к элементу по индексу с проверкой границ. В случае выхода за пределы выбрасывает исключение std::out\_of\_range, size\_t index - индекс элемента, нет входных данных, константная ссылка на элемент по указанному индексу.
* size\_t size() const, возвращает текущее количество элементов, хранящихся в векторе, нет параметров, нет входных данных, size\_t - количество элементов.
* void clear(), очищает вектор, устанавливая currentSize в 0, но не освобождая выделенную память, нет параметров, нет входных данных, вектор становится пустым (логически).
* T& last() / const T& last() const, возвращает ссылку на последний элемент в векторе, нет параметров, нет входных данных, ссылка на последний элемент.
* void removeLast(), удаляет последний элемент из вектора путем уменьшения currentSize. Память фактически не освобождается, нет параметров, нет входных данных, последний элемент удален (логически).
* bool isEmpty() const, проверяет, пуст ли вектор, нет параметров, нет входных данных, true, если currentSize равен 0, иначе false.
* iterator begin(), iterator end(), reverse\_iterator rbegin(), reverse\_iterator rend() и их const версии, предоставляют итераторы для обхода вектора, что позволяет использовать его с алгоритмами стандартной библиотеки C++, нет параметров, нет входных данных, соответствующие итераторы.
* перечисляете методы – заголовок, назначение, список формальных параметров, входные данные, выходные данные

Для класса, реализующего хеш-таблицу, в методе для хеш-функции приводите пример значений для ключей из анализа.

4.2 Описание среды разработки

Для реализации информационной системы «Юридическая фирма» была выбрана среда разработки Microsoft Visual Studio 2022. Язык реализации C++.

4.3 Руководство пользователя

При запуске исполняемого файла приложения KursProject.exe открывается главное окно с заголовком «KursProject», представляющее собой основной интерфейс информационной системы. Стартовое окно содержит две вкладки: «Справочники» (активная по умолчанию) и «Отладка». Вкладка «Справочники» содержит две таблицы: «Клиенты» (слева) и «Консультации» (справа), предназначенные для просмотра и управления данными (см. Рисунок F).

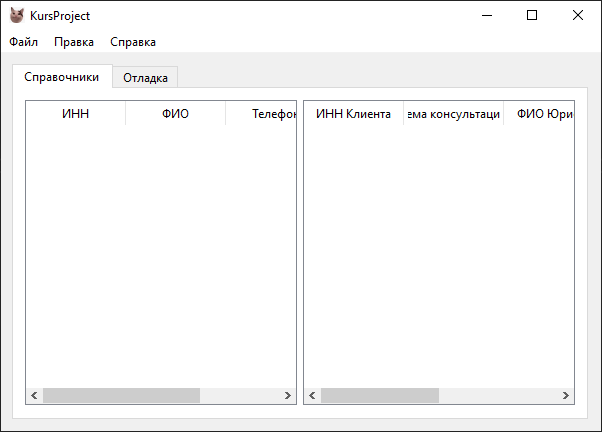


Рисунок F – Главное окно информационной системы

4.3.1 Работа со справочником «Клиенты»

После запуска приложения и загрузки данных, информация о клиентах автоматически отображается в таблице «Клиенты» на вкладке «Справочники». Таблица содержит колонки: «ИНН», «ФИО», «Телефон».

Чтобы загрузить данные необходимо выполнить следующие действия:

1. Выберите пункт меню «Файл» → «Загрузить справочник Клиенты».
2. В открывшемся диалоговом окне выбора файла укажите путь к текстовому файлу clients.txt (см. п. 3.1.1).
3. Нажмите кнопку «Открыть». Данные из файла будут загружены в систему и отображены в таблице «Клиенты». Появится сообщение об успешной загрузке.

Чтобы добавить информацию о клиенте, необходимо выполнить следующее:

1. Выберите пункт меню «Правка» → «Добавить клиента».
2. Откроется диалоговое окно «Добавить клиента» (см. Рисунок F).
3. Заполните поля «ИНН», «ФИО» и «Телефон», соблюдая требования к формату данных (см. п. 1.1).
4. Нажмите кнопку «Добавить». Если данные корректны и ИНН уникален, запись будет добавлена в таблицу и появится сообщение «Запись успешно добавлена!». В противном случае появится сообщение об ошибке.

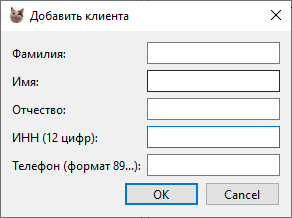


Рисунок F – Диалоговое окно «Добавить клиента»

Чтобы удалить информацию о клиенте, необходимо выполнить следующее:

4.4 Тестирование

Тестируете отдельно работу СД и отдельно работу со справочником.

Тестирование проводилось методом черного ящика, результаты которого представлены в Таблицах \_\_\_\_\_.

Таблица \_\_\_\_ – Тестирование работы со справочником «Название»

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Описание тестовой ситуации | | Входные данные | | Выходные данные | |
| Ваш справочник | Ваши поля | Ваш справочник | Сообщение |
| Добавление | | | | | |
| 1 | Название тестовой ситуации |  |  |  |  |
| 2 | и т.д. |  |  |  |  |
| Удаление | | | | | |
| 3 | Название тестовой ситуации |  |  |  |  |
| 4 | и т.д. |  |  |  |  |
| Поиск | | | | | |
| 5 | Название тестовой ситуации |  |  |  |  |
| 6 | и т.д. |  |  |  |  |

Таблица \_\_\_\_ – Тестирование структуры данных \_\_\_\_\_\_

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Описание тестовой ситуации | | Входные данные | | Выходные данные | |
| Ваша структура | Ваши поля | Ваша структура | Ваши поля |
| Добавление | | | | | |
| 1 | Название тестовой ситуации |  |  |  |  |
| 2 | и т.д. |  |  |  |  |
| Удаление | | | | | |
| 3 | Название тестовой ситуации |  |  |  |  |
| 4 | и т.д. |  |  |  |  |
| Поиск | | | | | |
| 5 | Название тестовой ситуации |  |  |  |  |
| 6 | и т.д. |  |  |  |  |

Аналогично для второго справочника и для отчета.

# Заключение

Целью курсового проекта было: повторение из введения.

Цель достигнута. Для достижения поставленной цели были выполнены следующие задачи:

Перечисление задач + что было дополнительно изучено для решения каждой (например, среда разработки и т. д.).

# Список литературы

должны быть ссылки на учебники (в том числе интернет-источники), из которых брали информацию теоретической части

1. Вики-конспекты Университет ИТМО: Хеш-таблица : сайт – URL: https://neerc.ifmo.ru/wiki/index.php?title=Хеш-таблица (дата обращения 01.06.2025).
2. T\_Kormen\_Ch\_Leyzerson\_R\_Rivest\_K\_Shtayn\_-\_Algo
3. <https://www.geeksforgeeks.org/open-addressing-collision-handling-technique-in-hashing/>
4. Niklaus\_Virt\_Algoritmy\_i\_struktury\_dannykh\_DMK\_Press\_2010