



МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования

«Дальневосточный федеральный университет»
(ДВФУ)

ШКОЛА ЕСТЕСТВЕННЫХ НАУК

Кафедра прикладной математики, механики, управления и программного обеспечения

САРАНЦЕВ АЛЕКСАНДР АНДРЕЕВИЧ

**РАЗРАБОТКА ИНФОРМАЦИОННОЙ СИСТЕМЫ ДЛЯ АВТОМАТИЗАЦИИ
РАБОТЫ СО СПРАВОЧНИКАМИ НАЛОГОВОЙ СЛУЖБЫ**

КУРСОВОЙ ПРОЕКТ

по дисциплине «Фундаментальные структуры данных и алгоритмы»
по образовательной программе подготовки бакалавров по направлению
09.03.04 - Программная инженерия

Студент гр. Б8118-09.03.04 прогин

А.А. Саранцев

(подпись)

Защищен с оценкой

Руководитель

Ученая степень

ст. преподаватель

должность

О.А. Крестникова

(подпись)

(И.О. Фамилия)

« ____ » _____ 2020 г.

(подпись)

(И.О. Фамилия)

г. Владивосток
2020

Оглавление

Оглавление.....	2
Введение.....	3
1 Анализ предметной области (ПО).....	4
1.1 Модель ПО.....	4
1.2 Постановки задач обработки.....	7
2 Теоретическая часть.....	9
2.1 Хеш-таблица.....	11
2.1.1 Хеш-функция.....	11
2.1.2 Разрешение коллизий методом открытой адресации.....	14
2.2 Красно-черное дерево.....	17
3 Требования к информационной системе.....	24
3.1 Функциональные требования.....	24
3.2 Требования к данным.....	27
3.2.1 Требования к входным данным при чтении из файла.....	27
3.2.2 Требования к входным данным при вводе через оконный интерфейс.....	34
3.2.3 Требования к выходным данным.....	35
3.3 Требования к интерфейсу.....	36
4 Реализация.....	37
4.1 Диаграмма классов.....	37
4.2 Описание классов.....	39
4.3 Описание интерфейса.....	68
4.4 Тестирование.....	73
Заключение.....	80
Список литературы.....	82

Введение

В настоящее время в мире и в России активно происходит автоматизация самых различных процессов. Это же касается вопросов экономики и налогообложения. У налоговой службы и, возможно, у банков существует необходимость вести учёт различных компаний и фирм, а также их доходов. Поэтому вместо ведения учёта вручную было бы более эффективно делегировать некоторую монотонную работу компьютеру. В данной курсовой работе будет разработано программное обеспечение для работы с некоторыми данными о компаниях и фирмах, которые занимают какую-либо конкретную рыночную нишу. Для примера будет взята сфера ремонтных услуг.

Таким образом требуется разработать информационную систему для автоматизации работы со справочниками информации об организациях, предоставляющих ремонтные услуги.

Целью курсового проекта является: разработка информационной системы для автоматизации работы со справочниками организации, предоставляющей услуги налоговой службы.

Для достижения поставленной цели необходимо выполнить следующие задачи:

1. Провести анализ предметной области и построить ее модель.
2. Изучить теоретические основы методов построения справочников.
3. Определить требования к информационной системе.
4. Реализовать и провести тестирование.

1 Анализ предметной области (ПО)

Система должна решать следующие задачи:

- 1) хранить информацию о компаниях, заказчиках и предоставляемых услугах;
- 2) позволять просматривать всю информацию о компаниях, заказчиках и услугах;
- 3) позволять добавлять информацию о новом заказчике, компании или услуге;
- 4) позволять удалять информацию о заказчике, компании или услуге;
- 5) позволять искать информацию о заказчиках, компаниях и услугах;
- 6) на основании информации в справочниках формировать отчеты отчета о доходах компаний, исходя из текущих заказов. Отчёта о списке заказчиков указанной компании. Отчета о времени выполнения всех заказов каждого заказчика. Отчёта по компаниям, выполняющим данную услугу;
- 7) предусмотреть сортировку данных в отчетах по названию компании, ФИО заказчиков и наименованию услуги;
- 8) предусмотреть проверку целостности информации, представленной в справочниках по компаниям, заказчикам и услугам.

1.1 Модель ПО

Предметная область – рынок ремонтных услуг.

Профессионал предметной области – налоговая служба.

Объекты предметной области:

Объект **Список цен на услуги** - информация о нём содержится в справочнике, который хранит информацию о цене каждой услуги.

Объект **Цена услуги** – информация о объекте содержит наименование услуги, название компании, которая её предоставляет, цене в и единица измерения услуги.

Услуга – набор букв русского и английского алфавита, а также символов пунктуации и пробелов.

Компания – набор букв русского и английского алфавита, а также символов пунктуации и пробелов.

Цена – рациональное число с двумя знаками после запятой (в руб.). Целая часть не более 10 цифр.

Единица измерения – набор букв русского и английского алфавита, а также символов пунктуации и пробелов. (у. е.)

Таблица 1 – Пример справочника:

<i>Услуга</i>	<i>Компания</i>	<i>Цена</i>	<i>У. е.</i>
покраска стен	ООО Ремонт-Строй	200.50	м ²
покраска стен	ОАО Строй-Ремонт	210	м ²
покраска стен	ООО Не Строим	150	м ²
евроремонт	ООО Ремонт-Строй	499.99	м ³
евроремонт	ОАО Строй-Ремонт	550.01	м ³
евроремонт	ООО Не Строим	455.55	м ³
установка натяжных потолков	ООО Ремонт-Строй	800	м ²
установка натяжных потолков	ОАО Строй-Ремонт	799.99	м ²
установка натяжных потолков	ООО Не Строим	685.11	м ²
установка натяжных потолков	ООО РемонтНет	600.25	м ²
услуга1	ИП 'Не ломай'	256	еденица1
услуга1	ИП 'сам себе ремонт'	128.16	еденица1
секретная услуга2	ООО Ремонт-Строй	100	доллары
секретная услуга2	ИП 'Не ломай'	100	доллары
поклейка обоев	ООО Ремонт-Строй	159.99	м ²

Продолжение Таблица 1 – Пример справочника:

<i>Услуга</i>	<i>Компания</i>	<i>Цена</i>	<i>У. е.</i>
поклейка обоев	ОАО Строй-Ремонт	150	м ²
поклейка обоев	ООО Не Строим	148.88	м ²
прикрутить полочку	ИП 'Не ломай'	400	одна полочка
прикрутить полочку	ООО Чинилкин	300	одна полочка
настроить пианино	ИП Лепёха А. А.	3000	одно пианино
сделать	ИП Саранцев А. А.	399.99	одна
лабораторную по			лабораторная
программированию			
посчитать предикат	ИП Саранцев А. А.	63.33	один предикат

Объект Список продолжительностей за единицу измерения – информация о них содержится в справочнике, который хранит информацию о продолжительности каждой услуги за единицу измерения.

Объект Продолжительность услуги за единицу измерения – информация об объекте содержит наименование услуги, её максимальную и минимальную длительность по времени.

Услуга – набор букв русского и английского алфавита, а также символов пунктуации и пробелов.

Минимальная длительность – рациональное число с двумя знаками после запятой. Целая часть не более 2 цифр. (в часах)

Максимальная длительность – рациональное число с двумя знаками после запятой (в руб.). Целая часть не более 3 цифр. (в часах)

Таблица 2 – Пример справочника:

<i>Услуга</i>	<i>Минимальная длительность</i>	<i>Максимальная длительность</i>
покраска стен	0.25	4.5
поклейка обоев	0.5	12

Продолжение Таблица 2 – Пример справочника:

<i>Услуга</i>	<i>Минимальная длительность</i>	<i>Максимальная длительность</i>
установка натяжных потолков	1.1	3
услуга1	0.01	951.67
секретная услуга2	0.05	1
посчитать предикат	0.01	0.02
настроить пианино	16	40
прикрутить полочку	0.2	1
сделать лабораторную по программированию	0.4	3
евроремонт	2.51	5.87

1.2 Постановки задач обработки

Важно заметить, что на основе входных данных необходимо автоматизировать составление некоторой отчётности, которая будет нужна для анализа имеющихся данных. Ниже приведены примеры некоторых вариантов такой отчётности, которые должны автоматически выводиться программным средством по мере надобности.

Формирование отчёта о времени выполнения всех заказов каждого заказчика

Входные данные:

1. Список заказов [1]:

- ФИО
- Услуга
- Компания
- Количество

2. Список продолжительностей за единицу измерения:

- Услуга
- Минимальная длительность
- Максимальная длительность

Выходные данные:

1. Справочник, отсортированный лексикографически по ФИО заказчиков, выведенный в виде текстового файла, который содержит информацию о длительности выполнения всех заказов для каждого заказчика:

- ФИО заказчика
- Минимальная длительность выполнения всех заказов
- Максимальная длительность выполнения всех заказов

Связь:

• Сначала из **Списка заказов [1]** берётся информация обо всех ФИО заказчиков, далее находим все заказы с каждым ФИО и для каждого заказчика составляем перечень услуг которые он заказал и их количество. Далее минимальная длительность выполнения всех заказов для k заказчика вычисляется по формуле:

$$t_k = \sum v_n \cdot m_n ; n \in N,$$

где n – это все заказы с ФИО k -го заказчика, v_n – это количество, m_n – это минимальная длительность услуги из заказа n , а t_k – минимальная длительность выполнения всех заказов заказчика k .

Максимальная длительность выполнения всех заказов для k заказчика вычисляется по формуле:

$$T_k = \sum v_n \cdot M_n ; n \in N,$$

где N – это все заказы с ФИО k -го заказчика, v_n – это количество, M_n – это максимальная длительность услуги заказа n , t_k – максимальная длительность выполнения всех заказов заказчика k .

В итоге будет получен удобный отчёт, содержащий информацию о минимальном и максимальном времени выполнения всех заказов для каждого заказчика.

Формирование отчёта о компаниях, выполняющих данную услугу

Входные данные:

1. Список компаний [1]:

- Компания
- Виды услуг
- Адрес компании

2. Список продолжительностей за единицу измерения:

- Услуга
- Минимальная длительность
- Максимальная длительность

Выходные данные:

Текстовый файл, отсортированный лексикографически по наименованию услуг и перечень, компаний, которые выполняют каждую услугу.

Связь:

Сначала из **Список продолжительностей за единицу измерения** составляем список всех услуг, далее в **Списке компаний** находим компании, которые выполняют какую-либо услугу из нашего списка.

В результате будет получен файл, где для каждой услуги будет иметься перечень компаний, которые её предоставляют.

2 Теоретическая часть

Исходя из требований, приоритетными операциями для работы над справочниками являются операции добавления, удаления и поиска элемента. Такой набор операций предоставляют ассоциативные массивы.

Ассоциативным массивом называется такой абстрактный тип данных, который позволяет хранить пары вида «ключ, значение» и поддерживающий операции добавления пары, а также поиска и удаления пары по ключу [7].

Существует множество способов реализаций ассоциативного массива. Главным критерием обоснования выбора конкретной реализации назовем

асимптотическую сложность выполнения вышеописанных операций. Среди всех реализаций наиболее выделяются следующие две: хеш-таблица (ХТ), сбалансированное бинарное дерево поиска (БДП). Рассмотрим вычислительную сложность операций для этих структур данных.

Таблица 3 – Хеш-таблица

	В среднем случае	В худшем случае
Расход памяти	$O(n)$	$O(n)$
Добавление пары	$O(1)$	$O(n)$
Поиск по ключу	$O(1)$	$O(n)$
Удаление по ключу	$O(1)$	$O(n)$

Хотя и возможная сложность всех операций, присущих ассоциативному массиву, является линейной ($O(n)$), на практике хеш-таблица является очень эффективным решением [2]. Подробнее об этом в пункте 2.1.

Таблица 4 – Сбалансированное бинарное дерево поиска.

	В среднем случае	В худшем случае
Расход памяти	$O(n)$	$O(n)$
Добавление пары	$O(\log n)$	$O(\log n)$
Поиск по ключу	$O(\log n)$	$O(\log n)$
Удаление по ключу	$O(\log n)$	$O(\log n)$

Сбалансированные бинарные деревья поиска очень удобны тем, что гарантируют логарифмическую ($O(\log n)$) сложность выполнения вышеописанных операций [2]. Подробнее о сбалансированных БДП в пункте 2.2.

Как видно, данные структуры данных позволяют выполнять операции добавления пары, поиска по ключу и удаления по ключу эффективно. Следовательно, их выбор обоснован.

2.1 Хеш-таблица

Как было сказано до этого, хеш-таблица (ХТ) представляет собой эффективную структуру данных для реализации ассоциативных массивов. Хеш-таблица обобщает обычный массив. Выполнение любой операции в ХТ начинается с вычисления хеш-функции от ключа), которая отображает произвольный ключ в индекс этого массива. Подробнее хеш-функция будет рассмотрена в пункте 2.1.1. Вычислив индекс при помощи хеш-функции, мы применяем операцию (добавления, удаления или поиска) к соответствующей данному индексу ячейке массива [3].

2.1.1 Хеш-функция

Иногда бывает, что количество возможных ключей настолько мало, что можно реализовать взаимно однозначное отображение ключей в индексы массива. Тогда размер массива будет равен количеству возможных ключей. Если же совокупность ключей велика, то хранение массива такого размера непрактично, а иногда и вовсе невозможно. Кроме того, множество реально сохраненных ключей может быть очень мало по сравнению с совокупностями ключей. Тогда память, выделенная для хеш-таблицы, в основном расходуется напрасно [2].

Как было сказано ранее, хеш-функция отображает произвольные ключи в индексы массива. Если имеется массив, который может вместить M пар «ключ, значение», то хеш-функция должна отображать любой ключ в одно из целых чисел из диапазона $[0, M - 1]$ [3].

Основной идеей хеш-функции является уменьшение диапазона индексов массива, чтобы была возможность обойтись массивом меньшего размера [2].

Построение хеш-функции с помощью метода деления [2] состоит в отображении ключа k в одну из m путем получения остатка от деления k на m , т.е. хеш-функция имеет следующий вид

$$h(k) = k \bmod m$$

Рассмотрим пример хеш-функции с методом деления, использующийся для преобразования строкового типа ключа в индекс массива. Так как строка состоит из последовательности символа, то значение хеш-функции для ключа *str* может вычисляться, к примеру [3], следующим образом. Пусть изначально значение переменной *hash* будет равным нулю. Последовательно, начиная с первого символа строки *str*, будем изменять значение переменной так: $hash = (R * hash + c) \% M$, где *R* – произвольная целочисленная константа, одинаковая для всех ключей, *c* – код символа в таблице *ASCII*, *M* – количество ячеек в массиве. Получившееся значение переменной *hash* будет значением хеш-функции.

Данная хеш-функция используется в реализации хеш-таблицы *Alexander::HashTable* с $R = 131071$. Подробнее об этом в пункте 4.2.

Необходимо отметить, что значения хеш-функции для равных ключей должны быть равны. Также качественная хеш-функция должна обладать следующими свойствами: легко вычисляться и равномерно распределять ключи по массиву [3].

Рассмотрим пример работы хеш-таблицы. Последовательно произведем операции добавления объектов *Услуга* из пример в пункте 1.1. Пусть размер нашей таблицы фиксированный и равен 5 (пока не будем учитывать рехеширование).

Тогда:

$$h(\text{«покраска стен»}) = 0$$

$$h(\text{«поклейка обоев»}) = 2$$

$$h(\text{«установка натяжных потолков»}) = 1$$

$$h(\text{«услуга1»}) = 3$$

После выполнения операций добавления вышеперечисленных элементов в хеш-таблицу, она будет иметь следующий вид (Рис. 1):

хеш	ключ объекта	отступ
0	покраска стен	0
1	установка натяжных потолков	0
2	поклейка обоев	0
3	услуга1	0
4		0

Рис. 1

Пока не будем обращать внимание на колонку «отступ». Ее значение объясняется в пункте 2.1.2.

Ситуация, когда в ячейке, соответствующей значению хеш-функции объекта, уже лежит другой объект, называется коллизией. Это одна из главных проблем использования хеш-функций. Так как хеш-функция на практике почти никогда не является инъективной, то вполне возможна такая ситуация, когда значения хеш-функции для разных ключей совпадут.

Остановимся подробнее на добавлении объекта с названием услуги «секретная услуга2» (Рис. 2)



Рис. 2

Но в нашем массиве уже есть какой-то объект, лежащий в этой ячейке. Существует два главных способа решения данной задачи [2]:

1. метод цепочек
2. открытая адресация

Рассмотрим далее метод открытой адресации с некоторыми оптимизациями.

2.1.2 Разрешение коллизий методом открытой адресации

При использовании открытой адресации все элементы хранятся непосредственно в массиве. При поиске элемента мы последовательно проверяем ячейки хеш-таблицы до тех пор, пока не найдем искомый элемент или пока не убедимся в его отсутствии. Здесь, в отличие от метода цепочек, нет ни списков, ни элементов, хранящихся вне массива. Существует несколько способов последовательного размещения элементов, которые называются последовательностями проб. Здесь мы рассмотрим один из способов опробирования, а именно – линейное.

Линейное опробование означает, что, если при попытке вставки в массив/поиска, текущий слот заполнен, то просто осуществляется переход к

следующему слоту. Если и он тоже заполнен, то берётся следующий слот, и так далее [3].

Некоторые оптимизации, которые позволяют увеличить производительность хеш-таблицы:

1. Хеширование Robin Hood при вставке элемента
2. Простые числа в качестве количества слотов
3. Ограничение максимального кол-ва наборов
4. Использование большего кол-ва ячеек массива во избежание

зацикленности хеш-таблицы

Хеширование Robin Hood [8] означает, что алгоритм вставки пытается расположить каждый элемент как можно ближе к его идеальной позиции. Это делается с помощью перемещения окружающих элементов при вставке какого-то элемента. Принцип работы алгоритма в следующем: берём из богатых элементов и передаём в бедные элементы. Богатым называется элемент, получивший слот поблизости от своей идеальной точки вставки (insertion point). Далее вместо идеальной точки вставки будем использовать термин «отступ». Бедный элемент находится далеко от идеальной точки вставки. Вставляя новый элемент, мы считаем, насколько далеко он находится от своей идеальной позиции. Если дальше, чем текущий элемент, то вы ставите новый на место текущего, а затем уже для него пытаетесь найти новое место.

Простые числа в качестве количества слотов [3] были выбраны с целью равнозначности каждого из битов значения хеш-функции. В результате получается результат, очень близкий к равномерному распределению.

Ограничение максимального количества наборов – одна из новых идей, позволяющих улучшить производительность хеш-таблиц. Основная идея данной оптимизации в следующем: если при вставке элемента в хеш-таблицу, его отступ от идеальной позиции больше определенного значения C , то происходит рехеширование. В данной конкретной реализации $C = \lceil \log M \rceil$, где M – количество ячеек в массиве.

Оптимизация, описанная в прошлом абзаце позволяет нам избежать заикленности массива, т. е. такой ситуации, когда значение хеш-функции близко к концу массива, который уже заполнен, и нам приходится продолжать поиск места для элемента с начала таблицы. Во избежание этой ситуации мы используем не M , а $M+C$ ячеек, хотя хеш-функция отображает ключ только в M индексов.

Теперь мы можем подробнее рассмотреть пример, представленный в пункте 2.1.1 (рис. 3).



Рис. 3

В ячейке массива, соответствующей значению хеш-функции объекта с названием услуги «секретная услуга2» уже лежит другой объект. Так как отступ объекта, лежащего в хеш-таблице равен 0, то мы продолжаем поиск места для элемента с ячейки с индексом 1. Так как «секретная услуга2» не может быть вставлена в ячейку с номером 0, ее предполагаемый отступ будет равняться 1. Тогда объект, лежащий в ячейке с номером 1, и имеющий отступ, равный 0 будет «богатым», а объект, который мы пытаемся вставить — «бедным». Поэтому мы вставляем в эту ячейку объект с ключом «секретная услуга2» и производим вставку элемента с ключом «установка натяжных потолков», начиная с позиции 2 и отступа 1.

Продолжая выполнение данного алгоритма, последовательно заменяем объекты, пока не дойдем до пустой ячейки с номером 4, в которую вставляем последний объект. После выполнения вышеперечисленных операций хеш-таблица будет иметь следующий вид:

ячейка	ключ объекта	отступ
0	покраска стен	0
1	секретная услуга2	1
2	установка натяжных потолков	1
3	поклейка обоев	1
4	услуга1	1

Рис. 4

2.2 Красно-черное дерево

Бинарное дерево – это такая структура данных, которая состоит из узлов, хранящих значение, а также ссылку на свою левую и правую ветвь [9]. Каждая ветвь также является деревом. Узел, находящийся в вершине дерева, принято называть корнем. Узлы, которые не имеют потомков, называются листьями. Ветви узла называют потомками, а по отношению к потомкам, узел называется предком. Также, развивая аналогию, имеются братья – узлы с общим родителем, дяди – братья родителя, дедушки – родители родителя.

Бинарное дерево поиска – бинарное дерево, удовлетворяющее следующему свойству. Пусть x представляет собой узел БДП. Если y является узлом в левом поддереве x , то $y.key \leq x.key$. Если y является узлом в правом поддереве x , то $y.key \geq x.key$. Запись $node.key$ обозначает ключ, хранящийся в качестве значения узла $node$ [4].

Бинарное дерево поиска может использоваться в качестве ассоциативного массива, однако сложность выполнения операций, присущих ассоциативному массиву не является удовлетворительной. Например, при последовательном добавлении упорядоченной последовательности в бинарное дерево поиска, сложность выполнения операций будет линейной.

Эта проблема решается использованием сбалансированных БДП. Как было сказано, они гарантируют время выполнения операций добавления, поиска и удаления за логарифмическое время. Одним из наиболее распространенных примеров сбалансированных БДП являются красно-черные деревья.

Красно-черное дерево представляет собой БДП с одним дополнительным битом цвета в каждом узле. Цвет узла может быть либо красным, либо черным. Красно-черные деревья являются приближенно сбалансированными, а именно имеют высоту, которая не превышает значение $2 \cdot \log_2(n+1)$ [2].

Бинарное дерево поиска является красно-черным деревом, если оно удовлетворяет следующим свойствам [2]:

1. Каждый узел является либо красным, либо черным.
2. Корень и конечные узлы (листья) дерева — чёрные
3. Если узел красный, то оба его дочерних узла — черные.
4. Для каждого узла все простые пути от него до листьев, являющихся потомками данного узла, содержат одно и то же количество черных узлов.
5. Чёрный узел может иметь чёрного родителя

Добавление в красно-черное дерево.

Узел, с которым мы работаем, на картинках имеет имя x .

Каждый элемент вставляется вместо листа, поэтому для выбора места вставки идём от корня до тех пор, пока указатель на следующего сына не станет nil (то есть этот сын — лист). Вставляем вместо него новый элемент с нулевыми потомками и красным цветом. Теперь проверяем балансировку. Если отец нового элемента черный, то никакое из свойств дерева не нарушено. Если же он красный, то нарушается свойство 3, для исправления достаточно рассмотреть два случая:

1. "Дядя" этого узла тоже красный. Тогда, чтобы сохранить свойства 3 и 4, просто перекрашиваем "отца" и "дядю" в чёрный цвет, а "деда" — в красный. В таком случае черная высота в этом поддереве одинакова для всех листьев и у всех красных вершин "отцы" черные. Проверяем, не нарушена ли балансировка. Если в результате этих перекрашиваний мы дойдём до корня, то в нём в любом случае ставим чёрный цвет, чтобы дерево удовлетворяло свойству 2.

2. "Дядя" чёрный. Если выполнить только перекрашивание, то может нарушиться постоянство чёрной высоты дерева по всем ветвям. Поэтому выполняем поворот. Если добавляемый узел был правым потомком, то необходимо сначала выполнить левое вращение, которое сделает его левым потомком. Таким образом, свойство 3 и постоянство черной высоты сохраняются.

Удаление из красно-черного дерева

При удалении вершины могут возникнуть три случая в зависимости от количества её детей:

- Если у вершины нет детей, то изменяем указатель на неё у родителя на `nil`.
- Если у неё только один ребёнок, то делаем у родителя ссылку на него вместо этой вершины.
- Если же имеются оба ребёнка, то находим вершину со следующим значением ключа. У такой вершины нет левого ребёнка (так как такая вершина находится в правом поддереве исходной вершины и она самая левая в нём, иначе бы мы взяли ее левого ребенка. Иными словами сначала мы переходим в правое поддерево, а после спускаемся вниз в левое до тех пор, пока у вершины есть левый ребенок). Удаляем уже эту вершину описанным во втором пункте способом, скопировав её ключ в изначальную вершину.

Проверим балансировку дерева. Так как при удалении красной вершины свойства дерева не нарушаются, то восстановление балансировки потребуется только при удалении чёрной. Рассмотрим ребёнка удалённой вершины.

- Если брат этого ребёнка красный, то делаем вращение вокруг ребра между отцом и братом, тогда брат становится родителем отца. Красим его в чёрный, а отца — в красный цвет, сохраняя таким образом черную высоту дерева. Хотя все пути по-прежнему содержат одинаковое количество чёрных узлов, сейчас x имеет чёрного брата и красного отца. Таким образом, мы можем перейти к следующему шагу.

- Если брат текущей вершины был чёрным, то получаем три случая:

- Оба ребёнка у брата чёрные. Красим брата в красный цвет и рассматриваем далее отца вершины. Делаем его черным, это не повлияет на количество чёрных узлов на путях, проходящих через b , но добавит один к числу чёрных узлов на путях, проходящих через x , восстанавливая тем самым влияние удаленного чёрного узла. Таким образом, после удаления вершины черная глубина от отца этой вершины до всех листьев в этом поддереве будет одинаковой.

- Если у брата правый ребёнок чёрный, а левый красный, то перекрашиваем брата и его левого сына и делаем вращение. Все пути по-прежнему содержат одинаковое количество чёрных узлов, но теперь у x есть чёрный брат с красным правым потомком, и мы переходим к следующему случаю. Ни x , ни его отец не влияют на эту трансформацию.

- Если у брата правый ребёнок красный, то перекрашиваем брата в цвет отца, его ребёнка и отца — в чёрный, делаем вращение. Поддерево по-прежнему имеет тот же цвет корня, поэтому свойство 3 и 4 не нарушаются. Но у x теперь появился дополнительный чёрный предок: либо a стал чёрным, или он и был чёрным и b был добавлен в качестве чёрного дедушки. Таким образом, проходящие через x пути проходят через один дополнительный чёрный узел. Выходим из алгоритма.

Продолжаем тот же алгоритм, пока текущая вершина чёрная и мы не дошли до корня дерева. Из рассмотренных случаев ясно, что при удалении выполняется не более трёх вращений.

Поиск в красно-черном дереве.

Следует отметить, что алгоритмы поиска в красно-черном дереве не отличаются от таковых в обычном БДП. Заметим, что вышеприведенные алгоритмы добавления и удаления элементов в красно-черное дерево позволяют хранить объекты с одинаковыми ключами. Значит, нужен такой алгоритм, который позволит по ключу получить список значений, хранящихся в этом дереве, с ключом, равным исходному. Приведем ниже один из способов реализации такого поиска.

Свойство БДП позволяет вывести все ключи, находящиеся в дереве, в отсортированном порядке с помощью простого рекурсивного алгоритма, называемого симметричным обходом дерева [2]. Суть алгоритма состоит в следующем. Пусть мы начинаем с корня дерева.

1. Вывести значения левого поддерева таким же алгоритмом
2. Вывести значение корня текущего поддерева
3. Вывести значения правого поддерева таким же алгоритмом

Пусть L – последовательность, получаем симметричным обходом дерева T . Запишем в конец данной последовательности значение NIL .

Назовем нижней границей ключа key в дереве T такой узел $node$, для которого выполняется:

1. Ключ любого элемента, находящегося до узла $node$ в данной последовательности, меньше ключа key .
2. Ключ любого элемента, находящегося после узла $node$ в данной последовательности, больше либо равен ключу key .

Назовем верхней границей ключа key в дереве T такой узел $node$, для которого выполняется:

1. Ключ любого элемента, находящегося до узла $node$ в данной последовательности, меньше либо равен ключу key .
2. Ключ любого элемента, находящегося после узла $node$ в данной последовательности, больше ключа key .

Алгоритм нахождения нижней границы ключа key [10].

1. Пусть x – корень поддерева, $y = NIL$.
2. Пока x не будет равно NIL
 - Если $key \leq x.key$, то $y = x$, x = левый ребенок x
 - Иначе x = правый ребенок x
3. Результат – y

Алгоритм нахождения верхней границы ключа key [10].

1. Пусть x – корень поддерева, $y = NIL$.
2. Пока x не будет равно NIL
 - Если $key < x.key$, то $y = x$, x = левый ребенок x
 - Иначе x = правый ребенок x
3. Результат – y

Алгоритм нахождения последующего узла (в порядке симметричного обхода) [2].

Пусть $node$ – узел дерева. Если существует правое поддерево, то результатом будет минимальный элемент правого поддерева (ищется при помощи последовательного перехода к левой ветви узла, пока она не будет равна NIL , тогда, минимумом будет предыдущий элемент). Иначе, пусть y – родитель узла x . Пока $y \neq NIL$ и x – правый ребенок узла y , кладем $x = y$, y = родитель y . Результатом будет y .

Алгоритм поиска [10]

1. Пусть $start$ – нижняя граница ключа key в дереве T
2. Пусть end – верхняя граница ключа key в дереве T
3. При помощи алгоритма нахождения последующего узла последовательно проходим от узла $start$ до узла end .
4. Как только мы достигли узла end , прекращаем поиск.

Совокупность всех пройденных узлов является искомым результатом поиска. Так как алгоритмы поиска нижней и верхней границы имеют логарифмическую сложность, а алгоритм нахождения последующего узла имеет амортизированную константную сложность ($O(1)$), то сложность данного алгоритма поиска будет равна $O(m + \log n)$, где n – количество элементов в дереве, а m – количество одинаковых элементов в данном дереве с ключом key .

Пример дерева

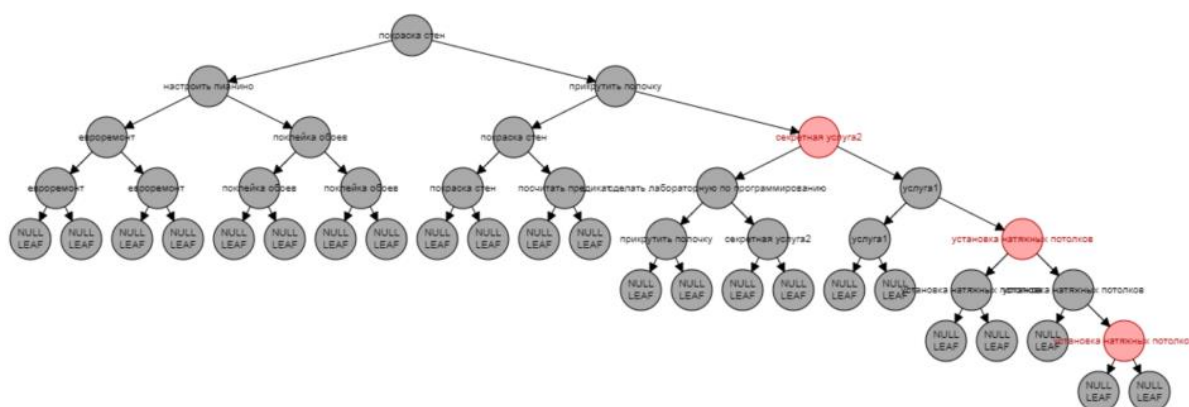


Рис. 5

3 Требования к информационной системе

Для более чёткого понимания задач важно сформулировать более точные требования к программному обеспечению (ПО), которое в дальнейшем будет реализовано. Два новых списка, а также объекты, хранящиеся в них, которые не были описаны в 1 главе этой курсовой работы, описываются в 1 главе [1].

3.1 Функциональные требования

Информационная система, для автоматизации работы со справочниками налоговой службы, должна позволять:

- Хранить информацию о **Списке компаний, Списке цен на услуги, Списке длительности за единицу измерения и Списке заказов**, используя структуры данных, необходимость которых была обоснована во 2 главе, а также во 2 главе курсовой работы [1]. Хранение информации о списках должно производиться непосредственно во время работы ПО.

- Позволять просматривать всю информацию из **Списка компаний, Списка цен на услуги, Списка длительности за единицу измерения и Списка заказов** по отдельности, используя оконный интерфейс ПО. Вывод происходит также в оконный интерфейс. Вывод каждого списка происходит в соответствии с указанными требованиями:

- 1) Таблица **Списка компаний** должна состоять из 4-х столбцов. В первом столбце содержатся индексы соответствующих объектов в хеш-таблице. Во втором столбце перечислены названия компаний, в третьем перечень названий услуг, которые предоставляет компания, а в четвёртом адреса соответствующих компаний.

- 2) Таблица **Списка продолжительностей за единицу измерения** должна содержать 4-х столбца. В первом столбце указывается индекс в хеш-таблице соответствующего объекта. Во втором столбце указывается название услуги, в третьем минимальная длительность, в четвёртом – максимальная.

- 3) Таблица **Списка цен на услуги** должна состоять из 4-х столбцов. В первом столбце должны содержаться названия соответствующих услуг, во

втором – названия компаний, которые их предоставляют. В третьем – указаны цены услуг за у. е. и в четвёртом указаны единицы измерения.

4) Таблица **Списка заказов** также должна состоять из 4-х столбцов. В первом столбце содержится имя заказчика, во втором компания, у которой была заказана услуга, в третьем столбце название услуги и в четвёртом объем в у. е.

- Позволять добавлять информацию об объектах из **Списка компаний, Списка цен на услуги, Списка длительности за единицу измерения и Списка заказов** по отдельности. Добавление должно выполняться через интерфейс оконного приложения, с использованием специальных полей для ввода текстовых и числовых данных или полей для выбора. Также должна выполняться проверка на добавление дублирующийся данных, в частности не могут быть добавлены одинаковые объекты **Компания** и **Продолжительность услуги за единицу измерения**. Также не могут быть добавлены объекты **Цена услуги** с одинаковыми полями **Компания** и **Услуга**.

- Позволять удалять информацию об объектах из **Списка компаний, Списка цен на услуги, Списка длительности за единицу измерения и Списка заказов**. Удаление должно выполняться через интерфейс оконного приложения, с использованием специальных полей для ввода текстовых и числовых данных или полей для выбора. Необходимо проверять целостность данных при удалении какого-либо объекта, т. е. в случае удаления какой-либо **Компании**, автоматически удаляются все **Заказы**, которые содержат соответствующее поле **Компания**. Также автоматически должны удаляться объекты **Цена услуги**, содержащие соответствующее поле **Компания** привязаны. В случае удаления объекта **Продолжительность услуги за единицу измерения** необходимо удалить соответствующую **Услугу** у всех **Компаний**, а также все **Заказы**, которые содержат соответствующее поле **Услуга**. Если же удаляется объект **Цена услуги**, то должны также удаляться **Заказы**, которые имеют соответствующее поле **Компания** и **Услуга**. Если удаление одного объекта влечёт за собой удаление

других объектов справочника, то должно выводиться окно с предупреждением. Пользователь должен иметь возможность отказаться от удаления.

- Позволять выполнять поиск объектов из **Списка компаний, Списка цен на услуги, Списка длительности за единицу измерения и Списка заказов** с дальнейшим выводом информации о найденных объектах в оконное приложение. Также должны быть выведена информация о произведённых сравнениях в структурах данных во время поиска объектов. Поиск должен выполняться через интерфейс оконного приложения, с использованием специальных полей для ввода текстовых и числовых данных или полей для выбора.

- Сохранять всю информацию о списках из текущей сессии в специальный файл разрешения «.sw2020». Формат данных в файле будет подробно описан в пункте **3.2.1**. Сохранение должно происходить при нажатии специальной кнопки в оконном приложении. Также должна быть возможность выбрать директорию компьютера, в которую нужно сохранить файл, а также указать имя файла.

- Читать информацию из предварительно сохранённого файла разрешения «.sw2020». Формат данных в файле будет подробно описан в пункте **3.2.1**. Чтение должно происходить при нажатии специальной кнопки в оконном приложении с дальнейшим выбором файла из любой директории компьютера.

- На основании информации из справочников в рамках текущей сессии формировать:

- 1) Отчёт о доходах компаний, исходя из текущих заказов.
- 2) Отчёт о списке заказчиков каждой компании.
- 3) Отчета о времени выполнения всех заказов каждого заказчика [1].
- 4) Отчёта по компаниям, выполняющим каждую услугу [1].

Подробное описание формирования отчётов приводится в конце 1 главы и 1 главы [1]. Отчёты должны быть выведены в виде файла с расширением «.txt» по нажатию специальной кнопки в оконном приложении. Должна быть реализована возможность сохранять файл в указанную директорию компьютера с указанным именем.

Также необходимо предусмотреть сортировку данных по названию компании в отчётах 1,4 и сортировку по имени заказчиков в отчёте 2.

3.2 Требования к данным

3.2.1 Требования к входным данным при чтении из файла

Основываясь на анализе ПО, входными данными для работы со справочниками является текстовый файл с расширением «.sw2020». Файл не содержит пустых строк и разделён на 4 основных поля, которые содержат информацию обо всех четырёх справочниках:

1. Первое поле содержит информацию о **Списке компаний**, и первая строка поля и файла соответственно хранит неотрицательное целое число n , которое указывает на количество последующих объектов типа **Компания** – каждый объект представлен набором строк, где:

- Первая строка содержит **Название компании**.
- Вторая строка содержит неотрицательное целое число m , которое указывает на количество услуг, которые предоставляет эта компания.
- Последующие m строк содержат **Виды услуг**, которые предоставляет компания – по одной в каждой строке.
- И последняя строка содержит **Адрес** компании.
- Данные об объектах записаны подряд в файле, без пустых строк.

2. Второе поле содержит информацию о **Списке заказов**, и первая строка, сразу после информации о компаниях, содержит неотрицательное целое число n , которое указывает на количество последующих объектов типа **Заказ** – каждый объект представлен набором строк, где:

- Первая строка содержит **ФИО** – имя заказчика.
- Вторая строка содержит **Услугу** – название услуги.
- Третья строка содержит **Компанию** – название компании, предоставляющей эту услугу.

- Последняя строка содержит **Количество** – объем заказа в у. е. в которых измеряется услуга.

3. Третье поле содержит информацию о **Списке цен на услуги**, и первая строка, сразу после информации о заказах, содержит неотрицательное целое число n , которое указывает на количество последующих объектов типа **Цена услуги** – каждый объект представлен набором строк, где:

- Первая строка содержит **Услугу** – название услуги.
- Вторая строка содержит **Компанию** – название компании, которая предоставляет услугу.
- Третья строка содержит **Цену** услуги у этой компании за у. е. – рациональное неотрицательное число.
- Четвёртая строка содержит название **Единицы измерения** (у. е.).

4. Четвёртое поле содержит информацию о **Списке продолжительностей за единицу измерения**, и первая строка, сразу после информации о заказах содержит неотрицательное целое число n , которое указывает на количество следующих объектов типа **Продолжительность услуги за единицу измерения** – каждый объект представлен набором строк, где:

- Первая строка содержит **Услугу** – название услуги.
- Вторая строка содержит **Минимальную длительность** (за у. е. в часах) – рациональное неотрицательное число.
- Вторая строка содержит **Максимальную длительность** (за у. е. в часах) – рациональное неотрицательное число.

Таким образом мы передаём в программу полные данные о каждом справочнике. Необходимым требованием является корректность введённых данных, то есть должен выполняться ряд условий:

- Все **Компании** из **Списка компаний** могут содержать только **Услуги** из **Списка продолжительностей за единицу измерения**.
- Все **Услуги** из **Списка цен на услуги** должны содержать поле **Компания**, которому соответствует какой-либо объект **Компания** из **Списка**

компаний, а также каждой Услуге должна соответствовать услуга из Списка продолжительностей за единицу измерения.

- Каждый **Заказ** из **Списка заказов** должен содержать поле **Компания** и поле **Услуга**, которым соответствуют объекты из **Списка компаний** и **Списка цен на услуги** соответственно.
- Объекты списков не должны повторяться, за исключением объектов **Заказ** из **Списка заказов**.

При невыполнении какого-либо из перечисленных условий программа будет работать некорректно.

Пример содержимого входного файла (через два слеша обозначены комментарии, которых не должно быть в файле):

Листинг 1 – Пример файла

```
10 // количество компаний
ОАО Строй-Ремонт // название компании
4 // кол-во услуг
покраска стен // услуги
евроремонт
установка натяжных потолков
поклейка обоев
Напротив 'ООО Ремонт-Строй' // адрес
ИП Лепёха А. А. // след. компания
1
настроить пианино
Кампус ДВФУ
ООО Руки-Крюки
0
Арбат, 10/6ы
ООО Не Строим
4
покраска стен
евроремонт
установка натяжных потолков
поклейка обоев
Амурская, 25
ООО Чинилкин
1
прикрутить полочку
```

ДВФУ корпус G
 ИП 'сам себе ремонт'
 1
 услуга 1
 Дом у реки
 ООО Ремонт-Строй
 5
 покраска стен
 евроремонт
 установка натяжных потолков
 секретная услуга 2
 поклейка обоев
 Проспект мира, строение 105, корпус 2, офис 5
 ООО РемонтНет
 1
 установка натяжных потолков
 Спортивная, 27
 ИП 'Не ломай'
 3
 услуга 1
 секретная услуга 2
 прикрутить полочку
 Торговый центр Изумруд
 ИП Саранцев А. А.
 2
 сделать лабораторную по программированию
 посчитать предикат
 Рядом с Луговой
 10 // кол-во объектов "заказ"
 Директор Начальников // ФИО
 секретная услуга 2 // услуга
 ООО Ремонт-Строй // компания
 50848 // объем
 Иванов И. И.
 покраска стен
 ООО Ремонт-Строй
 50
 Иванов И. И.
 поклейка обоев
 ООО Ремонт-Строй
 35
 Иванов И. И.

секретная услуга2
 ИП 'Не ломай'
 3
 Лепёха А. А.
 посчитать предикат
 ИП Саранцев А. А.
 10001
 Начальник Директоров
 евроремонт
 ОАО Строй-Ремонт
 1
 Начальник Директоров
 покраска стен
 ООО Ремонт-Строй
 1
 Панфилов А. С.
 настроить пианино
 ИП Лепёха А. А.
 1
 Саранцев А. А.
 услуга 1
 ИП 'сам себе ремонт'
 28
 Саранцев А. А.
 установка натяжных потолков
 ООО РемонтНет
 32
 22 // кол-во объектов "цена услуги"
 евроремонт // услуга
 ООО Ремонт-Строй // компания
 499.99 // цена
 м^3 // у. е.
 евроремонт
 ОАО Строй-Ремонт
 550.01
 м^3
 евроремонт
 ООО Не Строим
 455.55
 м^3
 настроить пианино
 ИП Лепёха А. А.

3000

одно пианино

поклейка обоев

ООО Ремонт-Строй

159.99

м²

поклейка обоев

ОАО Строй-Ремонт

150

м²

поклейка обоев

ООО Не Строим

148.88

м²

покраска стен

ООО Ремонт-Строй

200.5

м²

покраска стен

ОАО Строй-Ремонт

210

м²

покраска стен

ООО Не Строим

150

м²

посчитать предикат

ИП Саранцев А. А.

63.33

один предикат

прикрутить полочку

ИП 'Не ломай'

400

одна полочка

прикрутить полочку

ООО Чинилкин

300

одна полочка

сделать лабораторную по программированию

ИП Саранцев А. А.

399.99

одна лабораторная

секретная услуга2
 ООО Ремонт-Строй
 100
 доллары
 секретная услуга2
 ИП 'Не ломай'
 100
 доллары
 услуга 1
 ИП 'Не ломай'
 256
 еденица 1
 услуга 1
 ИП 'сам себе ремонт'
 128.16
 еденица 1
 установка натяжных потолков
 ООО Ремонт-Строй
 800
 м^2
 установка натяжных потолков
 ОАО Строй-Ремонт
 799.99
 м^2
 установка натяжных потолков
 ООО Не Строим
 685.11
 м^2
 установка натяжных потолков
 ООО РемонтНет
 600.25
 м^2
 10 // кол-во
 прикрутить полочку // объект "длительность услуги"
 0.2 // мин. Длительность
 1 // макс. Длительность
 поклейка обоев
 0.5
 12
 секретная услуга2
 0.05
 1

настроить пианино	
16	
40	
покраска стен	
0.25	
4.5	
услуга 1	
0.01	
951.67	
евроремонт	
2.51	
5.87	
посчитать предикат	
0.01	
0.02	
сделать лабораторную по программированию	
0.4	
3	
установка натяжных потолков	
1.1	
3	

3.2.2 Требования к входным данным при вводе через оконный интерфейс

Также входными данными для работы со справочником могут быть поля для какого-либо объекта, введённые по отдельности:

1. Для добавления объекта типа **Компания** пользователь должен ввести её название и адрес в два разных поля. Должны выполняться следующие условия:
 - Поля не могут быть пустыми.
 - Название компании не должно совпадать с названием компании из какого-либо другого объекта из **Списка компаний**.
 - Поля не могут быть пустыми.
2. Для добавления объекта типа **Длительность услуги** пользователь должен ввести название услуги, её **Минимальную длительность** и **Максимальную длительность**. Должны выполняться следующие условия:

- Длительности должны быть рациональными неотрицательными числами, причём максимальная не меньше минимальной.
- Название услуги не должно совпадать с названием какой-либо другой услуги из **Длительность услуги**.
- Поля не могут быть пустыми.

3. Для добавления объекта типа **Цена услуги** пользователь должен выбрать из предложенного списка существующих услуг нужную услугу, а также выбрать из другого списка **Компанию**. Затем необходимо заполнить поле **Цена** и поле **Единица измерения**. Должны выполняться следующие условия:

- Цена должна быть представлена неотрицательным рациональным числом.
- Выбранная **Компания** не должна содержать данную **Услугу**.
- Поля не могут быть пустыми.

4. Для добавления объекта типа **Заказ** пользователь должен выбрать из предложенного списка сначала **Компанию**, а затем из следующего списка **Услугу**. Затем необходимо заполнить поля **ФИО** заказчика и **Количество** – объем заказываемой услуги в у. е. При этом должны выполняться следующие условия:

- **Количество** должно введено в виде натурального числа.
- Поля не могут быть пустыми.

3.2.3 Требования к выходным данным

Выходными данными для работы со справочником являются:

1. Отчёты, требования к содержанию которых описаны в пункте **1.2.** и **3.1.**
2. Текстовый файл расширения «.sw2020», подробное описание формата которого представлено в пункте **3.2.1** и **3.1.**
3. Окна сообщаемые пользователю об ошибках либо предупреждающие о совершающихся операциях.

3.3 Требования к интерфейсу

Интерфейс должен быть реализован в виде оконного приложения, которое соответствует ряду требований:

- Добавление, удаление, поиск и демонстрация должны производиться путём нажатия специально отведенных для этого кнопок с дальнейшим вызовом соответствующих окон для ввода данных.
- При демонстрации списки должны отображаться в основной части окна в виде таблицы, требования к которой указаны в **3.2.3**.
- Интерфейс должен быть реализован на русском языке.
- Сохранение и чтение из файла должно быть реализовано в виде кнопок на панели меню.
- Вывод сообщений об ошибках производится в отдельных окнах.
- Окно приложения должно работать в полноэкранном режиме, а иметь изменяемый размер.
- Сохранение отчётов должно быть реализовано по нажатию специальных кнопок на панели меню.
- Должна быть реализована кнопка сворачивания окна в панель задач, а также кнопка выхода из приложения.

4 Реализация

4.1 Диаграмма классов

Основываясь на анализе ПО и на функциональных требованиях к информационной системе, определены типы классов и связи между ними, которые представлены в виде UML-диаграммы классов на Рисунках 6-8.

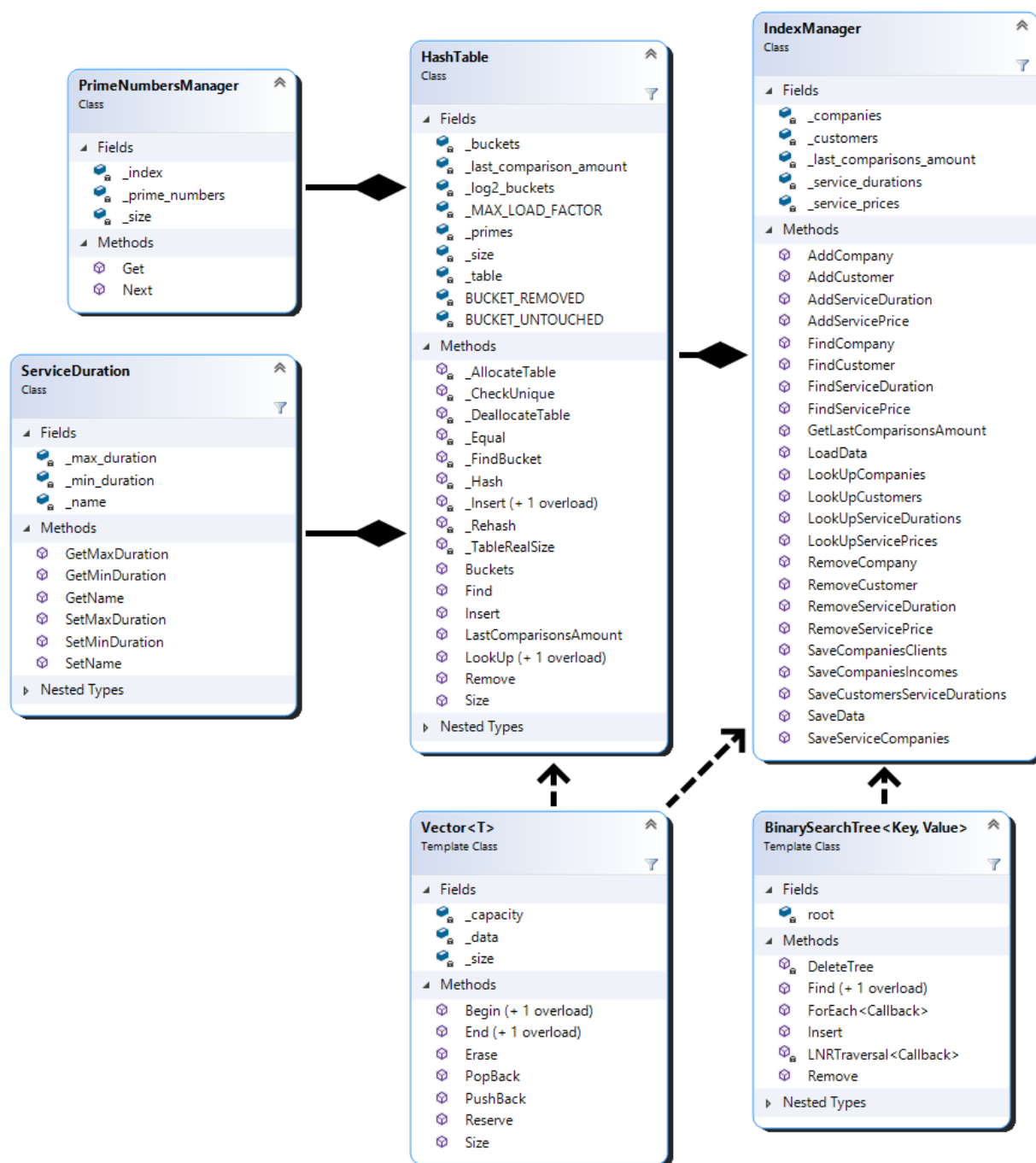


Рис. 6

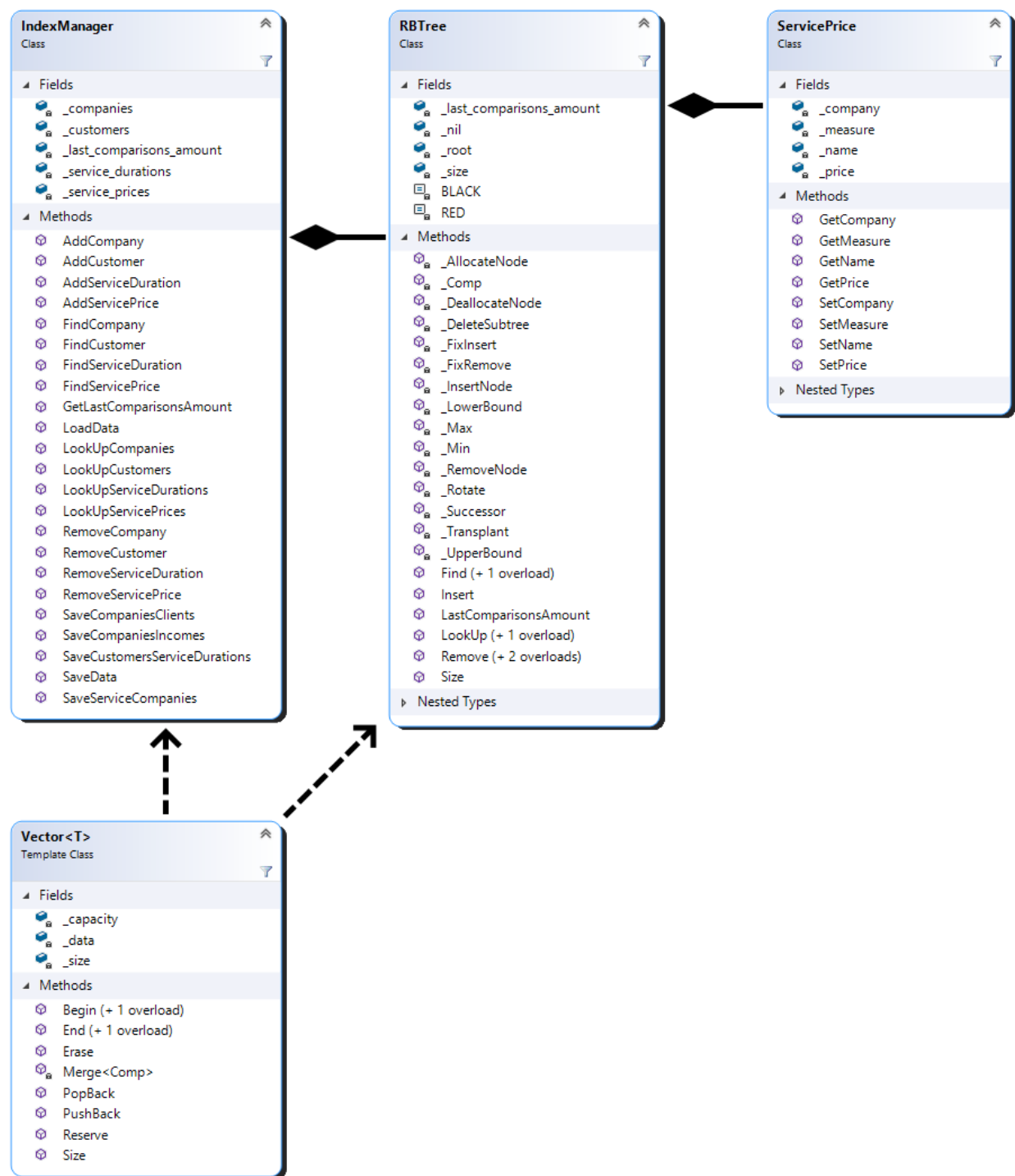


Рис. 7

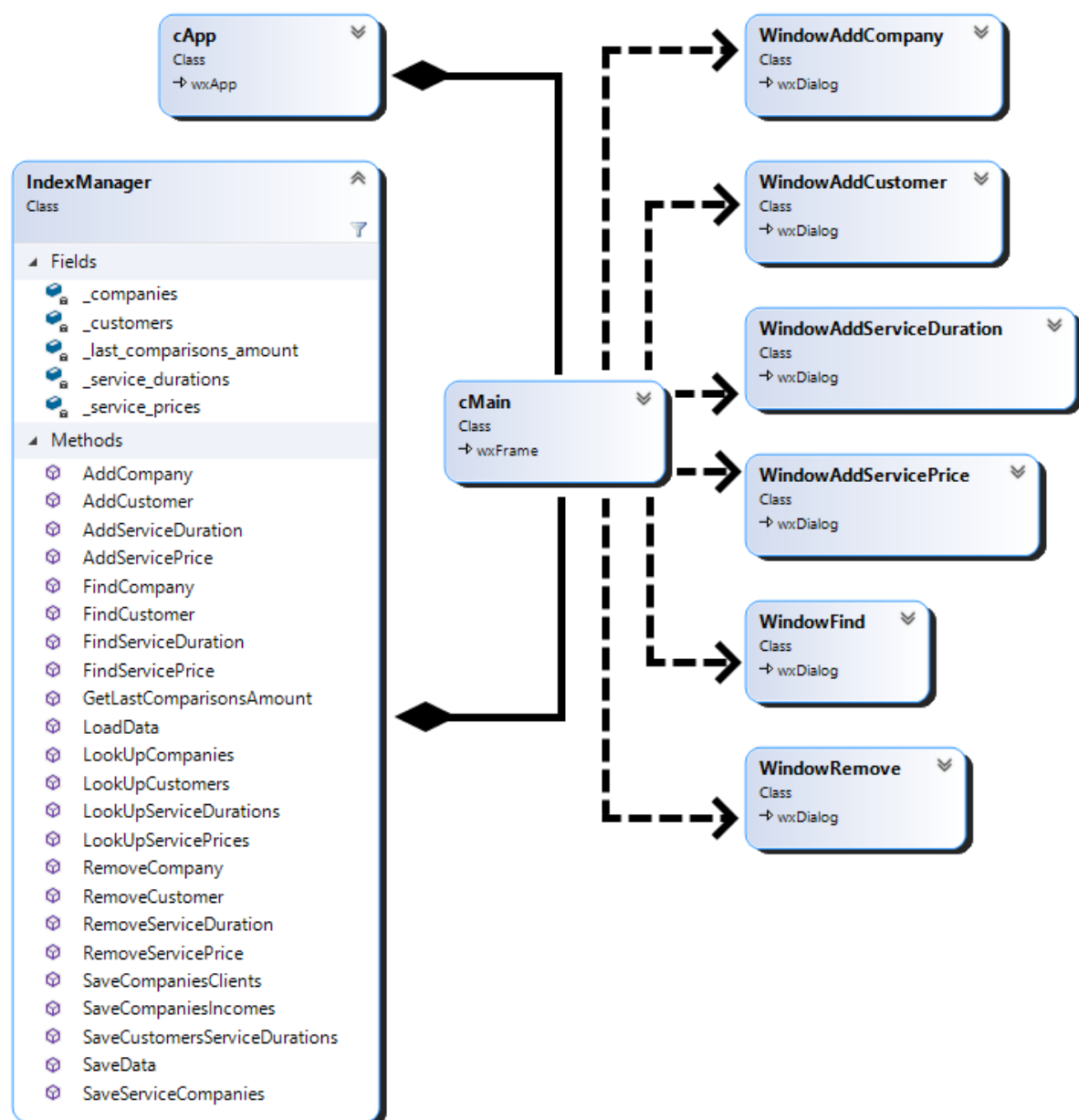


Рис. 8

4.2 Описание классов

Класс Alexander::HashTable – класс, описывающий хеш-таблицу.

Поля:

`size_t _buckets` – количество ячеек в хеш-таблице.

`size_t _log2_buckets` – количество дополнительных ячеек в конце хеш-таблицы.

`size_t _size` – количество элементов в хеш-таблице.

size_t _last_comparison_amount – количество сравнений, выполненных за последнюю примененную операцию добавления, удаления или поиска.

_Bucket* _table – указатель на таблицу.

PrimesNumberManager _primes – генератор простых чисел для хеширования.

double _MAX_LOAD_FACTOR – максимально возможный коэффициент заполненности хеш-таблицы.

int BUCKET_REMOVED – константа, используемая для обозначения того, что элемент, лежащий в ячейке хеш-таблицы, был удален.

int BUCKET_UNTOUCHED – константа, используемая для обозначения того, что в ячейке не лежало никаких элементов.

Методы:

_Hash(key) – вычисление значения хеш-функции ключа key

1. Входные данные

- key – строка, обозначающая название услуги
- _buckets – количество элементов в хеш-таблице

2. Выходные данные

- значение хеш-функции строки key для таблицы размера _buckets

Пример работы хеш-функции:

Пусть ключом (key) будет строка «услуга1», количеством ячеек в хеш-таблице (_buckets) будет равным 5. В качестве константы R возьмем число 131071. Определим операцию взятия остатка для отрицательного числа n как операцию взятия остатка от $(2^{32} + n)$.

hash = 0

hash = (R * hash + код(«у»)) mod _buckets = (131071 * 0 + (-13)) mod 5 = -13
mod 5 = $(2^{32} - 13) = 3$

hash = (R * hash + код(«с»)) mod _buckets = (131071 * 3 + (-15)) mod 5 = 393198
mod 5 = 3

$\text{hash} = (\text{R} * \text{hash} + \text{код}(\langle\langle\text{л}\rangle\rangle)) \bmod _buckets = (131071 * 3 + (-21)) \bmod 5 = 393192 \bmod 5 = 2$

$\text{hash} = (\text{R} * \text{hash} + \text{код}(\langle\langle\text{у}\rangle\rangle)) \bmod _buckets = (131071 * 2 + (-13)) \bmod 5 = 262129 \bmod 5 = 4$

$\text{hash} = (\text{R} * \text{hash} + \text{код}(\langle\langle\text{г}\rangle\rangle)) \bmod _buckets = (131071 * 4 + (-29)) \bmod 5 = 524255 \bmod 5 = 0$

$\text{hash} = (\text{R} * \text{hash} + \text{код}(\langle\langle\text{а}\rangle\rangle)) \bmod _buckets = (131071 * 0 + (-32)) \bmod 5 = -32 \bmod 5 = (2^{32} - 32) \bmod 5 = 4$

$\text{hash} = (\text{R} * \text{hash} + \text{код}(\langle\langle\text{л}\rangle\rangle)) \bmod _buckets = (131071 * 4 + 49) \bmod 5 = 524333 \bmod 5 = 3$

Значение хеш-функции для строки «услуга1» и размера массива 5 равно 3.

`_TableRealSize()` – подсчет реального количества выделенной памяти для хранения элементов хеш-таблицы

1. Входные данные

- `*this` – хеш-таблица

2. Выходные данные

- Реальное количество памяти, выделенного для хранения элементов хеш-таблицы

`_CheckUnique(value, hash)` – проверка ключа на уникальность

1. Входные данные

- `value` – объект, который надо проверить на уникальность
- `hash` – значение хеш-функции для данного объекта
- `*this` – хеш-таблица

2. Выходные данные

- `true`, если объекта с названием услуги `value.GetName()` нет в хеш-таблице

`_Insert(value, hash)` – вставка объекта `value` в ячейку с индексом `hash`

1. Входные данные

- `value` – объект, который надо вставить
- `hash` – значение хеш-функции для данного объекта
- `*this` – хеш-таблица

2. Выходные данные

- `*this` – хеш-таблица, в которой лежит элемент с ключом `value.GetName()` и все элементы, которые лежали до этого

`_Insert(value)` – вставка объекта `value` в хеш-таблицу

1. Входные данные

- `value` – объект, который надо вставить
- `*this` – хеш-таблица

2. Выходные данные

- `*this` – хеш-таблица, в которой лежит элемент с ключом `value.GetName()` и все элементы, которые лежали до этого

`_Equal(lhs, rhs)` – сравнение строк `lhs` и `rhs`

1. Входные данные

- `lhs` – первая строка
- `rhs` – вторая строка

2. Выходные данные

- `true`, если первая строка равна второй строке. Иначе – `false`.

`_FindBucket(key)` – поиск ячейки, которая содержит объект с названием услуги `key`

1. Входные данные

- `key` – название услуги
- `*this` – хеш-таблица

2. Выходные данные

- Указатель на ячейку, содержащую `key` в качестве названия услуги, либо нулевой указатель, если такого объекта не существует в хеш-таблице

`_Rehash()` – рехеширование хеш-таблицы

1. Входные данные

- `*this` – хеш-таблица

2. Выходные данные

- хеш-таблица большего размера, содержащая те же элементы

`_AllocateTable(size)` – аллокация массива размера `size`

1. Входные данные

- `size` – размер массива

2. Выходные данные

- указатель на массив размера `size`

`_DeallocateTable()` – деаллокация массива

1. Входные данные

- `start` – указатель на массив

2. Выходные данные

- Освобожденная память, ранее занимаемая массивом `start`

`Insert(value)` – вставка объекта `value` в хеш-таблицу

1. Входные данные

- `value` – объект, который надо вставить в хеш-таблицу
- `*this` – хеш-таблица

2. Выходные данные

- хеш-таблица, в которой лежит элемент с ключом `value.GetName()` и все элементы, которые лежали до этого

- `_last_comparison_amount` – количество сравнений, совершенных за операцию

- `true`, если объект был вставлен в хеш-таблицу. Иначе – `false`.

`Remove(key)` – удаление объекта с названием услуги `key`

1. Входные данные

- `key` – название услуги, объект с которой необходимо удалить из хеш-таблицы

- `*this` – хеш-таблица

2. Выходные данные

- хеш-таблица, не содержащая объектов с ключом `key`
- `_last_comparison_amount` – количество сравнений, совершенных за операцию

- `true`, если объект был в хеш-таблице. Иначе – `false`.

`Find(key)` – поиск объекта с заданным ключом

1. Входные данные

- `key` – название услуги

2. Выходные данные

- указатель на объект в хеш-таблице, название услуги которого совпадает с `key`, либо нулевой указатель, если объекта с таким ключом в хеш-таблице нет.

- `_last_comparison_amount` – количество сравнений, совершенных за операцию

`LookUp()` – формирование списка объектов из заданной хеш-таблицы

1. Входные данные

- *this – хеш-таблица

2. Выходные данные

- динамический массив, содержащий пары типа «объект, номер ячейки», составленный на основе хеш-таблицы

LookUp(pred) – формирование списка объектов из заданной хеш-таблицы

1. Входные данные

- pred – функция, принимающая один аргумент и возвращающая логическое значение

- *this – хеш-таблица

2. Выходные данные

- динамический массив, содержащий только те пары типа «объект, номер», которые принадлежат предикату на основе функции pred. Составлен на основе хеш-таблицы

Size() – количество элементов в хеш-таблице

1. Входные данные

- *this – хеш-таблица

2. Выходные данные

- количество элементов в хеш-таблице

Buckets() – количество ячеек в хеш-таблице

1. Входные данные

- *this – хеш-таблица

2. Выходные данные

- количество ячеек в этой хеш-таблице

LastComparisonsAmount() – количество сравнений, произведенный за последнюю выполненную операцию добавления, удаления или вставки

1. Входные данные
 - *this – хеш-таблица
2. Выходные данные
 - количество сравнений

Класс Alexander::RBTree – класс, описывающий красно-черное дерево.

Поля:

_Node* _root – указатель на корень дерева.

_Node* _nil – указатель на лист дерева.

size_t _last_comparison_amount – количество сравнений, произведенных за последнюю операцию добавления, удаления или поиска элемента.

size_t _size – количество элементов в дереве

Методы:

_Comp(lhs, rhs) – сравнение двух строк

1. Входные данные
 - lhs – первая строка
 - rhs – вторая строка
 2. Выходные данные
 - true, если первая строка лексикографически меньше второй.
- Иначе – false.

_DeleteSubtree(st_root) – рекурсивное удаление поддерева

1. Входные данные
 - st_root – корень поддерева
2. Выходные данные
 - освобожденная память, ранее занимаемая узлами поддерева

_Rotate(node, side) – поворот поддерева с корнем node в сторону side

1. Входные данные

- node – корень поддерева
 - side – сторона, в которую надо повернуть поддерево
2. Выходные данные
 - повернутое поддерево

`_Transplant(_Node* x, _Node* y)` – замена узла

1. Входные данные
 - x – указатель на первый узел
 - y – указатель на второй узел
2. Выходные данные
 - сохранение структуры, где вместо узла x располагается узел y

`_InsertNode(node)` – вставка узла node в дерево

1. Входные данные
 - node – узел, который надо вставить
 - *this – красно-черное дерево
2. Выходные данные
 - красно-черное дерево с узлом node

`_FixInsert(node)` – балансировка дерева после вставки объекта

1. Входные данные
 - node – узел, который был вставлен
 - *this – красно-черное дерево
2. Выходные данные
 - сбалансированное красно-черное дерево

`_RemoveNode(node)` – удаление из дерева узла node

1. Входные данные
 - node – узел, который надо удалить

- *this – красно-черное дерево
2. Выходные данные
 - красно-черное дерево, не содержащее узла node

`_FixRemove(node)` – балансировка дерева после удаления узла node

1. Входные данные
 - node – узел, который был удален
 - *this – красно-черное дерево
2. Выходные данные
 - сбалансированное красно-черное дерево

`_LowerBound(key)` – получение первого элемента (в порядке симметричного обхода), название услуги которого больше либо равно key

1. Входные данные
 - key – строка, с которой производится сравнение
 - *this – красно-черное дерево
2. Выходные данные
 - указатель на узел, содержащий объект, подходящий под упомянутое в назначении условие

`_UpperBound(key)` – получение первого элемента (в порядке симметричного обхода), название услуги которого больше key

1. Входные данные
 - key – строка, с которой производится сравнение
 - *this – красно-черное дерево
2. Выходные данные
 - указатель на узел, содержащий объект, подходящий под упомянутое в назначении условие

`_Successor(node)` – получение следующего в порядке симметричного обхода узла дерева

1. Входные данные

- `node` – указатель на узел дерева
- `*this` – красно-черное дерево

2. Выходные данные

- указатель на следующий узел дерева, или `_nil`, если `node` – последний элемент дерева (в порядке симметричного обхода)

`_Min(node)` – получение указателя на минимальный элемент поддерева с корнем `node`

1. Входные данные

- `node` – корень поддерева

2. Выходные данные

- указатель на минимальный элемент поддерева с корнем `node`, либо `nil`, если `node = nil`

`_Max(node)` – получение указателя на максимальный элемент поддерева с корнем `node`

1. Входные данные

- `node` – корень поддерева

2. Выходные данные

- указатель на максимальный элемент поддерева с корнем `node`, либо `nil`, если `node = nil`

`_AllocateNode(sp)` – аллокация узла, содержащего объект `sp`

1. Входные данные

- `sp` – объект, представляющий цену услуги

2. Выходные данные

- указатель на узел, содержащий объект `sp`

`_DeallocateNode(node)` – деаллокация узла `node`

1. Входные данные

- `node` – узел дерева

2. Выходные данные

- освобожденная память, ранее занимаемая узлом `node`

`Insert(sp)` – вставка объекта в дерево

1. Входные данные

- `sp` – объект, который надо вставить в дерево
- `*this` – красно-черное дерево

2. Выходные данные

- `*this` – красно-черное дерево с узлом, содержащим объект `sp`
- `_last_comparisons_amount` – количество сравнений объектов, совершенных за эту операцию

`Remove(key)` – удаление из дерева всех узлов, содержащих в качестве названия услуги `key`

1. Входные данные

- `key` – строка
- `*this` – красно-черное дерево

2. Выходные данные

- `*this` – красно-черное дерево, в котором ни один из узлов не содержит в качестве названия услуги значение `key`
- `_last_comparisons_amount` – количество сравнений объектов, совершенных за эту операцию

`Remove(key, pred)` – удаление из дерева всех узлов, содержащих в качестве названия услуги `key` и для которых функция `pred` возвращает `true`

1. Входные данные

- `key` – строка
- `pred` – функция, которая принимает объект, содержащийся в узле и возвращающая логическое значение
- `*this` – красно-черное дерево

2. Выходные данные

- `*this` – красно-черное дерево, в котором удалены объекты, попадающие под вышеупомянутые условия
- `_last_comparisons_amount` – количество сравнений объектов, совершенных за эту операцию

`Remove(pred)` – удаление из дерева всех узлов, для которых функция `pred` возвращает `true`

1. Входные данные

- `pred` – функция, которая принимает объект, содержащийся в узле и возвращающая логическое значение
- `*this` – красно-черное дерево

2. Выходные данные

- красно-черное дерево, в котором удалены объекты, попадающие под вышеупомянутые условия
- `_last_comparisons_amount` – количество сравнений объектов, совершенных за эту операцию

`Find(key)` – поиск узлов в красно-черном дереве

1. Входные данные

- `key` – строка
- `*this` – красно-черное дерево

2. Выходные данные

- динамический массив указателей на объекты с ключом `key`, содержащиеся в данном дереве
- `_last_comparisons_amount` – количество сравнений объектов, совершенных за эту операцию

`Find(key, pred)` – поиск узлов в красно-черном дереве с предикатом

1. Входные данные

- `key` – строка
- `pred` – функция, которая принимает объект, содержащийся в узле и возвращающая логическое значение

- `*this` – красно-черное дерево

2. Выходные данные

- динамический массив указателей на объекты с ключом `key`, содержащиеся в данном дереве, для которых функция `pred` возвращает `true`
- `_last_comparisons_amount` – количество сравнений объектов, совершенных за эту операцию

`LookUp()` – список всех элементов дерева в динамическом массиве в порядке симметричного обхода

1. Входные данные

- `*this` – красно-черное дерево

2. Выходные данные

- динамический массив, содержащий указатели на все элементы дерева в порядке прямого обхода

`LookUp(pred)` – список всех элементов дерева в динамическом массиве в порядке симметричного обхода, подходящих под заданное условие

1. Входные данные

- *this – красно-черное дерево
- pred – функция, которая принимает объект, содержащийся в узле и возвращающая логическое значение

2. Выходные данные

- динамический массив, содержащий указатели на все элементы дерева в порядке прямого обхода, для которых pred возвращает true

Size() – количество элементов в красно-черном дереве

1. Входные данные

- *this – красно-черное дерево

2. Выходные данные

- количество элементов в красно-черном дереве

LastComparisonsAmount() – количество сравнений, произведенное за последнюю выполненную операцию добавления, удаления или вставки

1. Входные данные

- *this – красно-черное дерево

2. Выходные данные

- количество сравнений

Класс BinarySearchTree<T> – класс, описывающий бинарное дерево поиска.

Поля:

Node* root – корень дерева.

Методы:

DeleteTree(tree_root) – рекурсивное удаление поддерева

1. Входные данные

- tree_root – корень поддерева

2. Выходные данные

- освобожденная память, ранее занимаемая узлами поддерева

`LNRTraversal(tree_root, callback)` – симметричный обход поддерева с корнем `tree_root`

1. Входные данные

- `tree_root` – корень поддерева
- `callback` – функция, принимающая один аргумент типа `T`

2. Выходные данные

- вызовы функции `callback` на объектах, хранящихся в узлах поддерева с корнем `tree_root`

`Insert(key, value)` – вставка пары «`key, value`» в дерево

1. Входные данные

- `key` – ключ, по которому будет происходить поиск в дереве
- `value` – объект, к которому можно будет получить доступ по

ключу

- `*this` – БДП

2. Выходные данные

- БДП с узлом, содержащим пару «`key, value`»
- `true`, если объект был вставлен в дерево. Иначе – `false`.

`Remove(key)` – удаление элемента из дерева по ключу

1. Входные данные

- `key` – ключ, по которому будет происходить поиска объекта для удаления

- `*this` – БДП

2. Выходные данные

- БДП, не содержащее пары «ключ, значение» с ключом, равным `key`

- true, если объект был вставлен в дерево. Иначе – false.

Find(key) – поиск значения, лежащего по указанному ключу

1. Входные данные

- key – ключ, по которому будет происходить поиск значения
- *this – БДП

2. Выходные данные

- указатель на значение объект, лежащего в этом дереве по данному ключу, или, если такого ключа нет – нулевой указатель.

ForEach(callback) – вызов функции callback от всех пар «ключ, значение» этого дерева при симметричный обходе

1. Входные данные

- callback – функция, принимающая два аргумента типов ключ и значение
- *this – БДП

2. Выходные данные

- вызовы функции callback на объектах, хранящихся в узлах поддерева с корнем tree_root

Класс IndexManager – класс, управляющий работой справочника.

Поля:

size_t _last_comparisons_amount – количество сравнений, совершенных за последний вызванный метод поиска;

Aleksei::HashTable _companies – справочник (хеш-таблица), хранящий информацию о компаниях;

Aleksei::RBTree _customers – справочник (красно-черное дерево), хранящий информацию о заказах;

Alexander::RBTree _service_prices – справочник (красно-черное дерево), хранящий информацию о ценах услуг;

Alexander::HashTable _service_durations – справочник (хеш-таблица), хранящий информацию о продолжительностях услуг за единицу измерения.

Методы:

AddCompany(company_name, address) – добавление в справочник компании с названием company_name и адресом address

1. Входные данные

- company_name – название компании
- address – адрес компании
- _companies – справочник компаний

2. Выходные данные

- обновленный справочник компаний, хранящий информацию о компании с названием company_name и адресом address
- если такая компания уже присутствует в справочнике – исключение

AddCustomer(name, service, company, volume) – добавление в справочник заказа с ФИО заказчика name, услугой service, компанией company, количеством volume

1. Входные данные

- name – ФИО заказчика
- service – название услуги
- company – название компании
- volume – количество (объем) услуги
- _customers – справочник заказов

2. Выходные данные

- обновленный справочник заказов, хранящий вышеупомянутую информацию о заказе.

- если компании с таким названием нет в справочнике, то выбрасывается исключение
- если компания с таким названием не предоставляет такой услуги, то выбрасывается исключение

AddServicePrice(name, company, price, measure) – добавление в справочник цены услуги с названием name, компанией company, ценой price и единицей измерения measure

1. Входные данные

- name – название услуги
- company – название компании
- price – цена услуги
- measure – единица измерения услуги
- _service_prices – справочник цен услуг

2. Выходные данные

- обновленный справочник цен услуг, содержащий информацию о вышеупомянутой цене услуги
- если услуги с таким названием нет в справочнике, то выбрасывается исключение
- если услуга уже предоставляется данной компанией, то выбрасывается исключение
- если компании с таким названием нету в базе данных, то выбрасывается исключение

AddServiceDuration(name, min_duration, max_duration) – добавление продолжительности услуги за единицу измерения с названием name, минимальной длительностью min_duration и максимальной длительностью max_duration

1. Входные данные

- name – название услуги

- `min_duration` – минимальная длительность
- `max_duration` – максимальная длительность
- `_service_durations` – справочник продолжительностей услуг за единицу измерения

2. Выходные данные

- обновленный справочник продолжительностей услуг за единицу измерения, содержащий информацию о вышеупомянутом объекте

`RemoveCompany(name)` – удаление компании с названием `name`

1. Входные данные

- `name` – название компании
- `_companies` – справочник компаний

2. Выходные данные

- обновленные справочники `_companies`, `_customers`, `_service_prices`, не содержащие информацию о компании с названием `name`
- если компании с таким названием нет в справочнике, то выбрасывается исключение

`RemoveCustomer(name)` – удаление заказа с именем заказчика `name`

1. Входные данные

- `name` – имя заказчика
- `_customers` – справочник заказов

2. Выходные данные

- обновленный справочник `_customers`, не содержащий информацию о заказчике с именем `name`
- если заказчика с таким именем нет в справочнике, то выбрасывается исключение

`RemoveServicePrice(name, company)` – удаление цены услуги с названием услуги `name` и названием компании `company`

1. Входные данные

- `name` – название услуги
- `company` – название компании
- `_service_prices` – справочник цен услуг
- `_companies` – справочник компаний

2. Выходные данные

- обновленные справочники `_companies`, `_customers`, `_service_prices`, не содержащие информацию о том, что компания `company` выполняет услугу `name`
- если компании с таким названием нет в справочнике, то выбрасывается исключение

`RemoveServiceDuration(name)` – удаление компании с названием `name`

1. Входные данные

- `name` – название компании
- `_service_durations` – справочник компаний

2. Выходные данные

- обновленные справочники `_customers`, `_service_prices`, `_service_durations` не содержащие информацию об услуге с названием `name`
- если услуги с таким названием нет в справочнике, то выбрасывается исключение

`FindCompany(name)` – поиск компании в справочнике по ее названию

1. Входные данные

- `name` – название компании
- `_companies` – справочник компаний

2. Выходные данные

- объект Компания с названием name, находящийся в справочнике
- если компании с таким названием нет, то выбрасывается

исключение

FindCustomer(name) – поиск заказов в справочнике по ФИО заказчика

1. Входные данные

- name – ФИО заказчика
- _companies – справочник заказчиков

2. Выходные данные

- Динамический массив объектов Заказ с ФИО заказчика name, находящихся в справочнике

- если заказов с таким ФИО заказчика нет, то выбрасывается исключение

FindServicePrice(name) – поиск цен услуг в справочнике по названию услуги

1. Входные данные

- name – название услуги
- _service_prices – справочник цен услуг

2. Выходные данные

- Динамический массив объектов Цена услуги с названием name, находящихся в справочнике

- если цен услуг с таким названием нет, то выбрасывается исключение

FindServiceDuration(name) – поиск продолжительности услуги за единицу измерения в справочнике по названию услуги

1. Входные данные

- name – название услуги

- `_service_durations` – справочник продолжительностей услуг за единицу измерения

2. Выходные данные

- объект Продолжительность услуги за единицу измерения с названием `name`, находящийся в справочнике
- если продолжительности услуги с таким названием нет, то выбрасывается исключение

`LookUpCompanies()` – составление списка компаний на основе справочника `_companies`

1. Входные данные

- `_companies` – справочник компаний

2. Выходные данные

- динамический массив, содержащий все компании, лежащие в справочнике `_companies`

`LookUpCustomers()` – составление списка заказчиков на основе справочника `_customers`

1. Входные данные

- `_customers` – справочник заказчиков

2. Выходные данные

- динамический массив, содержащий всех заказчиков, лежащих в справочнике `_customers`

`LookUpServicePrices()` – составление списка цен услуг на основе справочника `_service_prices`

1. Входные данные

- `_service_prices` – справочник цен услуг

2. Выходные данные

- динамический массив, содержащий все цены услуг, лежащие в справочнике `_service_prices`

`LookUpServiceDurations()` – составление списка продолжительностей услуг за единицу измерения на основе справочника `_service_durations`

1. Входные данные

- `_service_durations` – справочник продолжительностей услуг за единицу измерения

2. Выходные данные

- динамический массив, содержащий все продолжительности услуг за единицу измерения, лежащие в справочнике `_service_durations`

`SaveData(file_name)` – сохранение информации в файле, находящегося по пути `file_name`, способом, который был показан в пункте 3.2.1

1. Входные данные

- `_companies` – справочник компаний
- `_customers` – справочник заказов
- `_service_prices` – справочник цен услуг
- `_service_durations` – справочник продолжительностей услуг за единицу измерения

2. Выходные данные

- файл, находящийся по пути `file_name`, с сохраненной информацией из справочников

`LoadData(file_name)` – загрузка информации из файла, находящегося по пути `file_name`, способом, который был показан в пунктах 3.2.1 и 3.2.3

1. Входные данные

- `file_name` – имя файла

2. Выходные данные

- `_companies` – справочник компаний
- `_customers` – справочник заказов
- `_service_prices` – справочник цен услуг
- `_service_durations` – справочник продолжительностей услуг за единицу измерения

`SaveCompaniesIncomes(file_name)` – формирование отчета о доходах компаний, исходя из текущих заказов

1. Входные данные

- `file_name` – имя файла
- `_customers` – справочник заказчиков
- `_service_prices` – справочник цен услуг

2. Выходные данные

- файл с вышеупомянутым отчетом в формате, описанном в пункте

3.2.3

`SaveCompaniesClients(file_name)` – формирование отчета о списке клиентов каждой компании

1. Входные данные

- `file_name` – имя файла
- `_customers` – справочник заказчиков

2. Выходные данные

- файл с вышеупомянутым отчетом в формате, описанном в пункте

3.2.3

`SaveCustomersServiceDurations(file_name)` – формирование отчета о времени выполнения всех заказов каждого заказчика

1. Входные данные

- `file_name` – имя файла

- `_customers` – справочник заказчиков
- `_service_durations` – справочник продолжительностей услуг

2. Выходные данные

- файл с вышеупомянутым отчетом в формате, описанном в пункте

3.2.3

`SaveServiceCompanies(file_name)` – формирование отчета о компаниях, выполняющих услугу

1. Входные данные

- `file_name` – имя файла
- `_companies` – справочник компаний

2. Выходные данные

- файл с вышеупомянутым отчетом в формате, описанном в пункте

3.2.3

`LastComparisonsAmount()` – получение количества сравнений, совершенных за последнюю операцию поиска

1. Входные данные

- `*this` – управляющий справочниками

2. Выходные данные

- количество сравнений, совершенных за последнюю операцию поиска

Класс `PrimeNumbersManager` – класс, описывающий генератор простых чисел.

Поля:

`_index` – текущий номер простого числа в массиве

`_size` – размер массива простых чисел

`_prime_numbers[_size]` – массив простых чисел

Методы:

Get() – получение простого числа по текущему индексу

1. Входные данные
 - `_index` – текущий индекс
2. Выходные данные
 - простое число по индексу `_index`

Next() – увеличение индекса

1. Входные данные
 - `_index` – текущий индекс
2. Выходные данные
 - ссылка на объект типа `PrimeNumbersManager` с увеличенным на единицу индексом

Класс `ServicePrice` – класс, описывающий объект Цена услуги.

Поля:

`std::string _name` – название услуги

`std::string _company` – название компании

`double _price` – цена

`std::string _measure` – единица измерения

Методы:

`GetName()`, `GetCompany()`, `GetPrice()`, `GetMeasure()` возвращают значения, хранящиеся в полях класса соответственно

`SetName(name)`, `SetCompany(company)`, `SetPrice(price)`, `SetMeasure(measure)` устанавливают аргументы в поля класса, соответственно

Класс `ServiceDuration` – класс, описывающий объект Продолжительность услуги за единицу измерения.

Поля:

`std::string _name` – название услуги

double _min_duration – минимальная длительность

double _max_duration – максимальная длительность

Методы:

GetName(), GetMinDuration(), GetMaxDuration() возвращают значения, хранящиеся в полях класса соответственно

SetName(name), SetMinDuration(duration), SetMaxDuration(duration), устанавливают аргументы в поля класса, соответственно

Класс Vector<T> – класс, описывающий динамический массив (вектор).

Поля:

size_t _size – количество элементов в векторе

size_t _capacity – максимальное количество элементов, которые могут храниться в векторе

T* data – указатель на массив

Методы:

PushBack(t) – вставка объекта в конец динамического массива

1. Входные данные

- t – объект, который надо вставить
- *this – динамический массив

2. Выходные данные

- динамический массив большего размера с объектом t в конце

PopBack() – удаление объекта из конца динамического массива

1. Входные данные

- *this – динамический массив

2. Выходные данные

- динамический массив меньшего размера с удаленным объектом, ранее находящимся в конце массива

Erase(index) – удаление объекта из динамического массива, лежащего в ячейке с номером index

1. Входные данные

- index – индекс массива
- *this – динамический массив

2. Выходные данные

- динамический массив меньшего размера с удаленным объектом, ранее находящимся по индексу index

Reserve(capacity) – резервирование места под элементы массива

1. Входные данные

- capacity – целое число, обозначающее количество элементов, которые смогут храниться в динамическом массиве без необходимости в реаллокации массива

- *this – динамический массив

2. Выходные данные

- динамический массив с размером внутреннего массива capacity

Begin() – получение указателя на начало динамического массива

1. Входные данные

- *this – динамический массив

2. Выходные данные

- указатель на начало динамического массива

End() – получение указателя на следующий элемент после конца динамического массива

1. Входные данные

- *this – динамический массив

2. Выходные данные

- указатель на следующий элемент после конца динамического массива

Size() – получение количества элементов в массиве

1. Входные данные

- *this – динамический массив

2. Выходные данные

- количество элементов в массиве

4.3 Описание интерфейса

Интерфейс программного средства представляет собой оконное приложение, которое содержит панель меню, кнопки для выхода, раскрытия в полноэкранном режиме и сворачивания окна в панель задач. В основной части окна расположены четыре кнопки, которые ответственны за выполнения добавления, удаления, поиска и демонстрации каждого списка по отдельности в виде таблицы.

Ниже приведён скриншот окна приложения.

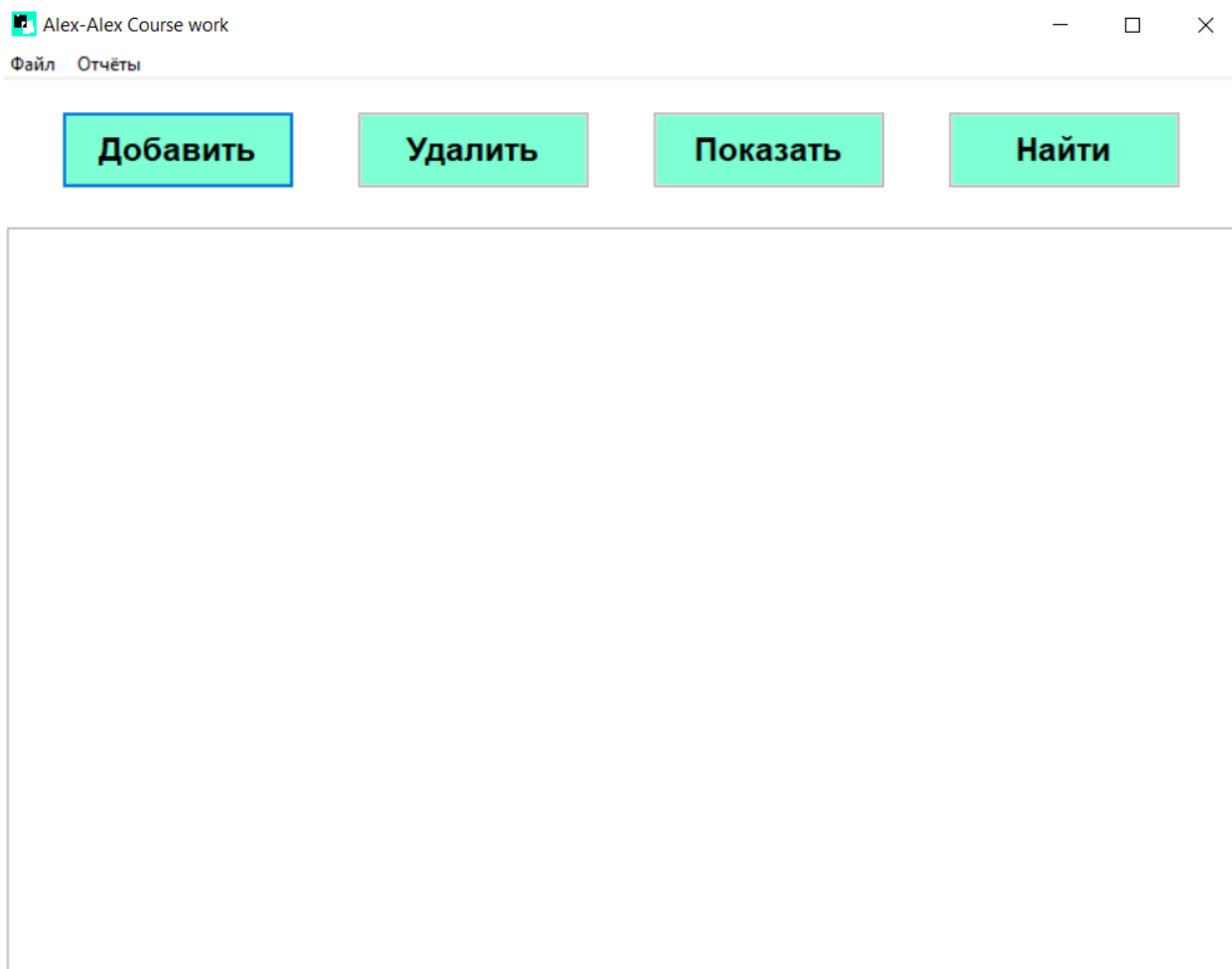


Рис. 9

В панели меню есть кнопка «Файл» и кнопка «Отчеты», по нажатию кнопки файл будет открыта небольшая панель с набором кнопок для сохранения текущей сессии в файл, чтения из файла, создания новой сессии и выхода из приложения.

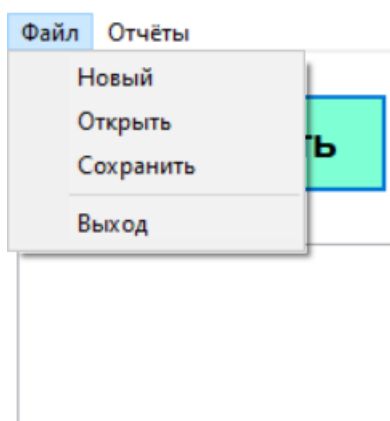


Рис. 10

В данных кнопках реализованы соответствующие функциональные требования из 3 главы.

По нажатию кнопки «Отчёты» можно сохранить любой из вариантов отчетов, указанных в функциональных требованиях 3 главы и 1 главе.

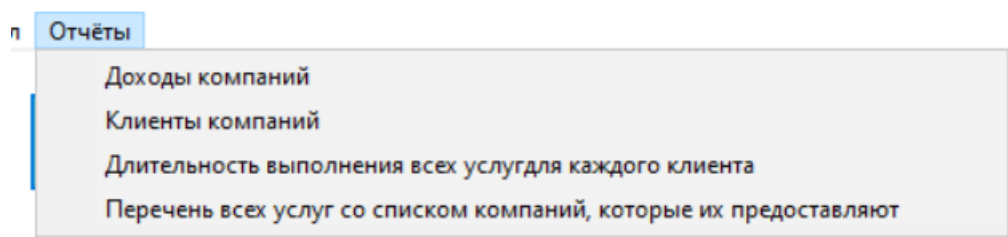


рис. 11

По нажатию на одну из четырёх основных кнопок с названием «Показать» пользователю будет представлен выбор из четырёх списков, которые можно отобразить в окне приложения.



рис. 12

Далее каждый список при выборе будет выведен в специальное поле в окне в соответствии с требованиями к выходным данным из 3 главы.

По нажатию на кнопку «Добавление» пользователю будет предложен выбор из четырех объектов, которые можно добавлять: **Компания, Заказ, Цена услуги и Длительность услуги за единицу измерения.** Окно не будет закрываться до тех пор, пока пользователь не закроет его нажатием кнопки «Ок» или просто крестика в правом верхнем углу.



Рис. 13

Далее при выборе объекта, который нужно добавить будет открыто дополнительное окно для ввода и выбора параметров добавления нового объекта. Например, так выглядит окно добавления **Цены услуги**, которое открывается по нажатию кнопки «Добавить услугу к компании»:

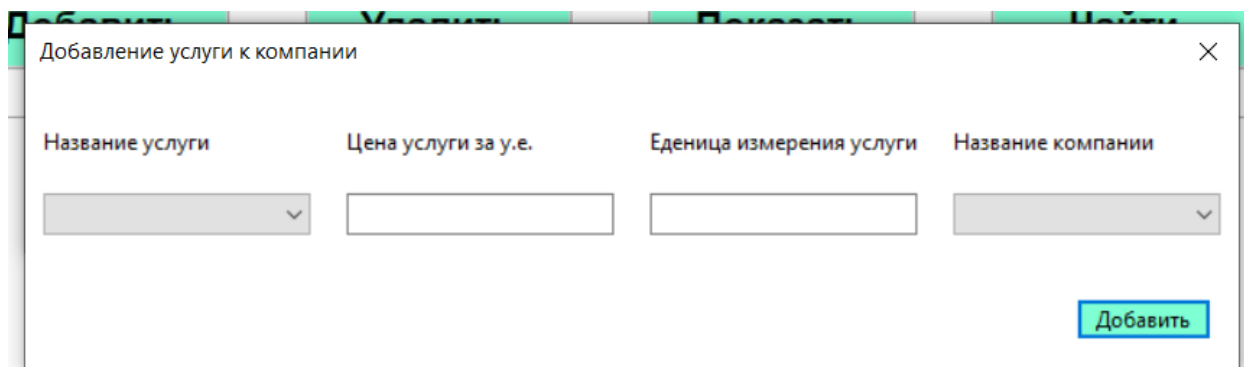
A screenshot of a software dialog box titled "Добавление услуги к компании" (Adding service to company). The dialog has a close button (X) in the top right corner. It contains four input fields: "Название услуги" (Service name) with a dropdown arrow, "Цена услуги за у.е." (Service price per unit) as a text box, "Единица измерения услуги" (Service unit of measurement) as a text box, and "Название компании" (Company name) with a dropdown arrow. A blue "Добавить" (Add) button is located at the bottom right.

Рис.14

Остальные окна добавления будут выглядеть аналогично в соответствии с требованиями к входным данным [пункт 3.2.2].

Однако важно заметить, что есть **Список компаний** и **Список длительностей за единицу измерения пуст**, то при попытке добавления объекта **Цена услуги** или **Заказ** будет выводиться окно с сообщением об ошибке. Это сделано в соответствии с требованиями к целостности данных [пункт 3.1]. Все сообщения об ошибках в приложении выглядят следующим образом: ошибке. Это сделано в соответствии с требованиями к целостности данных [пункт 3.1]. Все сообщения об ошибках в приложении выглядят следующим образом:

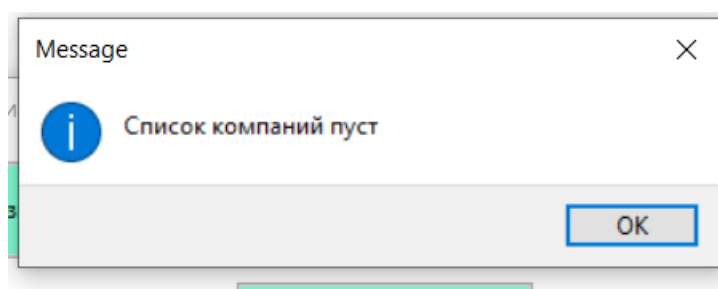


Рис. 15

При нажатии на кнопку «Удаление» будет открыто дополнительное окно, для выбора типа объекта, который нужно удалить и затем, при выборе, будут доступен выбор ключа этого объекта, по которому будет происходить удаление. Окно выглядит следующим образом:

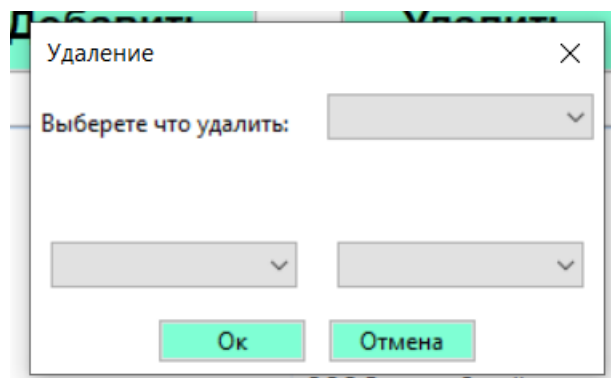


Рис. 16

При выборе полей и нажатии кнопки «Ок» будет происходить удаление указанного объекта из соответствующего списка. При этом пользователю не нужно беспокоиться о целостности и связности других списков, в которых есть объекты, связанные так или иначе с удалённым. Программа будет сохранять целостность автоматически в соответствии с требованиями [3 глава]. При нажатии кнопки отмена, будет произведён выход из этого окна.

Далее при нажатии на главном окне кнопки «Найти» будет открыто вспомогательное окно для поиска объектов в списках.

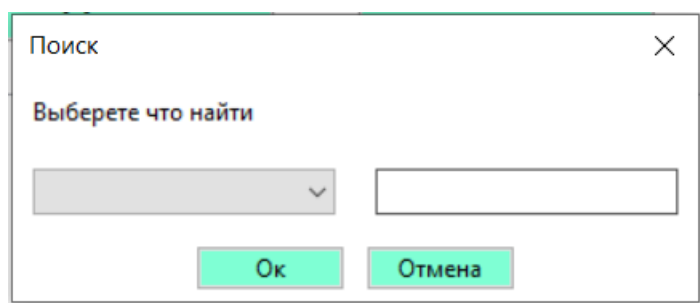


Рис. 17

Пользователю необходимо ввести какой тип объекта искать, а также ввести ключ к этому объекту и нажать кнопку «Ок». В случае успешного нахождения объект или список объектов с одинаковыми ключами будет выведен в главном окне приложения, также будет отображено маленькое окно с данными о количестве произведённых сравнений в структурах данных.

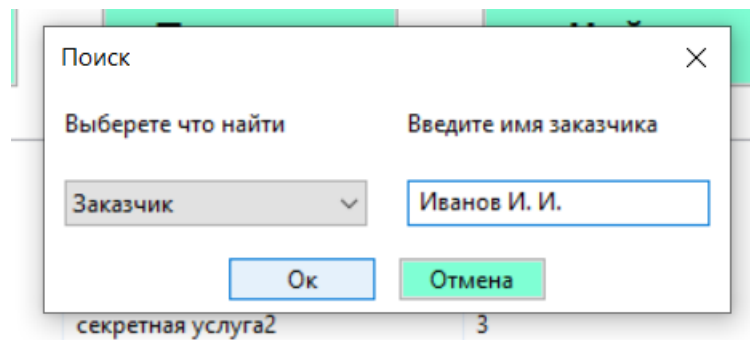


Рис. 18

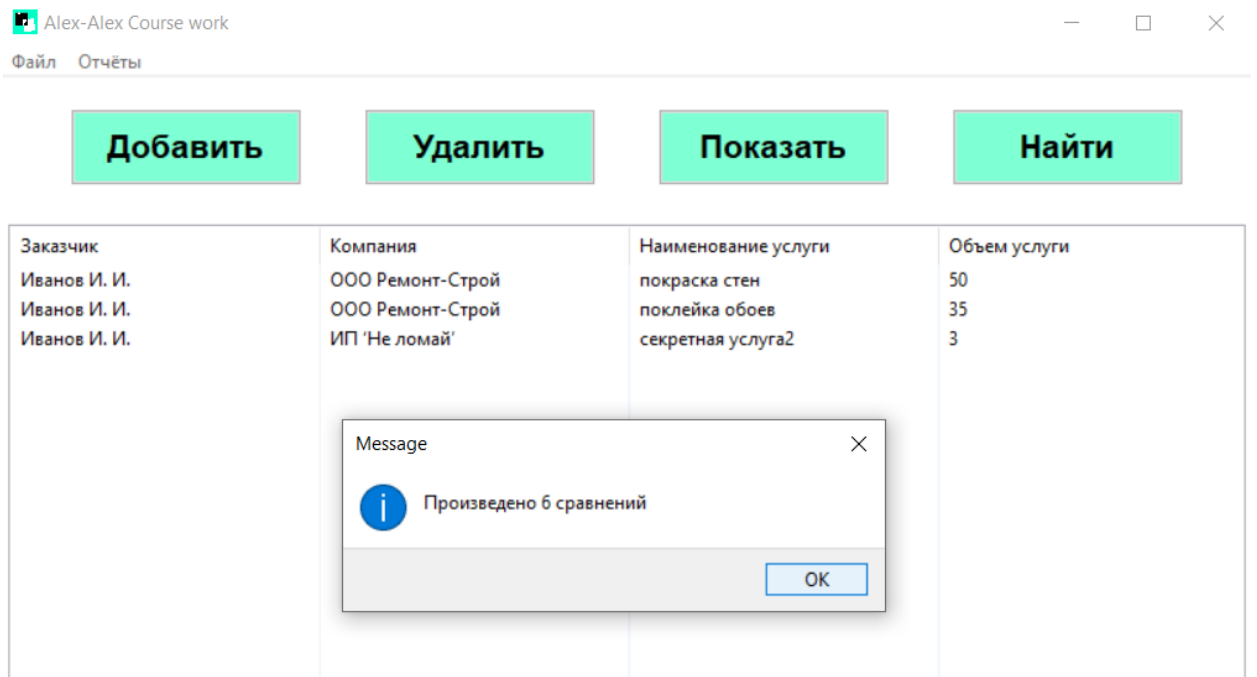


Рис. 19

В случае если объект не был найден будет выведено сообщение об ошибке.

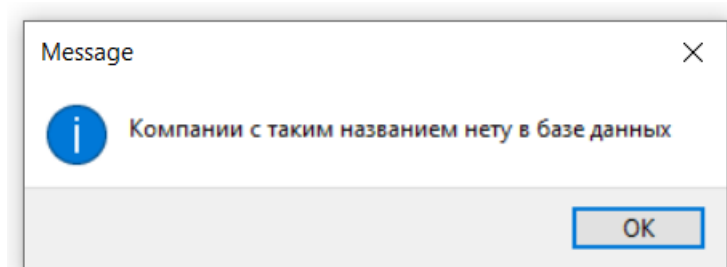


Рис. 20

4.4 Тестирование

Справочник цен услуг

В качестве начального КЧ дерева возьмем пример из пункта 1.1. Также примем некоторые вполне обоснованные допущения, касательно наличия

информации о компании с названием «ООО «Ремонтники»» и услуги с названием «отремонтировать ремонт» в остальных наших справочниках.

Таблица 5 – Тестирование

Описание тестовой ситуации		Входные данные		Выходные данные	
		КЧ дерево	{ название услуги, название компании, цена услуги, единица измерения }	КЧ дерево	{ название услуги, название компании, цена услуги, единица измерения }
Добавление					
1	Добавление некорректных данных	КЧ дерево	{ отремонтировать ремонт, ООО «Ремонтники», число, м ² }	Сообщение об ошибке: «В поле для цены должно быть число», неизменившееся КЧ дерево	
2	Добавление корректных данных	КЧ дерево	{ отремонтировать ремонт, ООО «Ремонтники», 10, м ² }	КЧ дерево с успешно вставленным элементом	
3	Добавление объекта с неуникальным ключом	КЧ дерево, содержащее также следующие элементы: { покраска стен, ООО Ремонт-Строй, 200.500000, м ² }; { покраска стен, ОАО Строй-Ремонт, 210.000000, м ² }; { покраска стен, ООО Не Строим, 148.880000, м ² }	{ покраска стен, ООО Чинилкин, 180, м ² }	КЧ дерево с успешно вставленным элементом	
Поиск					

Продолжение Таблица 5 – Тестирование

Описание тестовой ситуации	Входные данные	Выходные данные		
	КЧ дерево	{ название услуги, название компании, цена услуги, единица измерения }	КЧ дерево	{ название услуги, название компании, цена услуги, единица измерения }
Запись не существует	КЧ дерево, не содержащее записей с ключом «настроить рояль»	настроить рояль	Сообщение об ошибке «Компаний, выполняющих данную услугу нету в базе данных»	
Запись существует	КЧ дерево, содержащее записей с ключом «настроить рояль»	настроить пианино	Сообщение «Произведено 8 сравнений»	{ настроить пианино, ИП Лепёха А. А., 3000.000000, одно пианино }
Поиск объектов с одинаковым ключом	КЧ дерево, содержащее несколько записей с одинаковым ключом «прикрутить полочку»	прикрутить полочку	«Произведено 9 сравнений»	{ прикрутить полочку, ИП 'Не ломай', 400.000000, одна полочка }; { прикрутить полочку, ООО Чинилкин, 300.000000, одна полочка }
Удаление				

Продолжение Таблица 5 – Тестирование

Описание тестовой ситуации	Входные данные	Выходные данные		
	КЧ дерево	{ название услуги, название компании, цена услуги, единица измерения }	КЧ дерево	{ название услуги, название компании, цена услуги, единица измерения }
Запись существует	КЧ дерево, содержащее объект с названием услуги «посчитать предикат» и названием компании «ИП Саранцев А. А.»	ИП Саранцев А. А., посчитать предикат	КЧ дерево с удаленным объектом { посчитать предикат, ИП Саранцев А. А., 63.330000, один предикат }	
Удаление по неуникальному ключу	КЧ дерево, содержащее несколько объектов с названием услуги «посчитать предикат»	ИП 'Не ломай', услуга 1	КЧ дерево с удаленным объектом { услуга 1, ИП 'Не ломай', 256.000000, еденица 1 }	

Справочник продолжительностей услуг за единицу измерения

В качестве начальной хеш-таблицы возьмем пример из пункта 1.1.

Таблица 6 – Тестирование

Описание тестовой ситуации		Входные данные		Выходные данные	
		Хеш-таблица	{ название услуги, минимальная длительность, максимальная длительность }	Хеш-таблица	{ название услуги, минимальная длительность, максимальная длительность }
Добавление					
1	Добавление некорректных данных	Хеш-таблица	{ Временной парадокс, 4, 3 }	Сообщение об ошибке «Некорректный диапазон длительностей»	
2	Добавление корректных данных	Хеш-таблица	{ Не временной парадокс, 3, 4 }	Хеш-таблица с успешно вставленным элементом	
3	Проверка уникальности ключа	Хеш-таблица, содержащая объект с ключом «поклейка обоев»	{ поклейка обоев, 1, 2 }	Сообщение об ошибке «Данная услуга уже существует»	
4	Добавление при коллизии	Хеш-таблица, содержащая объект с ключом «услуга 1»	{ нарисовать картину, 1000, 2000 }	Хеш-таблица следующего вида: 0: объект с ключом «услуга 1» 1: объект с ключом «нарисовать картину» 2:...	
Поиск					
5	Запись не существует	Хеш-таблица, не содержащая объект с ключом «отремонтировать машину»	отремонтировать машину	Сообщение об ошибке «Услуги с таким названием нету в базе данных»	

Продолжение Таблица 6 – Тестирование

6	Запись существует	Хеш-таблица, содержащая объект с ключом «посчитать предикат»	посчитать предикат	Сообщение «Произведено 3 сравнений»	{ посчитать предикат, 0.010000, 0.020000 }
7	Поиск при коллизии	Хеш-таблица, содержащая объекты с ключами «услуга 1» и «нарисовать картину», хеш-значения которых совпадают	нарисовать картину	Сообщение «Произведено 2 сравнения»	{ нарисовать картину, 1000, 2000 }
Удаление					
8	Запись не существует	Хеш-таблица, не содержащая объект с ключом «ключ»	ключ	невозможно, согласно требованиям к интерфейсу, описанных в главе 3, пункте 3.2.3	
9	Запись существует	Хеш-таблица, содержащая объект с ключом «поклейка обоев»	поклейка обоев	Хеш-таблица с удаленным объектом { поклейка обоев, 0.500000, 12.000000 }	
10	Удаление при коллизии	Хеш-таблица, содержащая объект с ключом «посчитать предикат»	посчитать предикат	Хеш-таблица, не содержащая объект с ключом «посчитать предикат»	

Таблица 7 – Тестирование

Описание тестовой ситуации		Входные данные		Выходные данные
		Ваша структура данных	Ваши поля	Текстовый файл
Формирование отчетов				
1	Формирование отчета о времени выполнения всех заказов каждого заказчика			1. Директор Начальников : 2542.400000,50848.000000 2. Иванов И. И. : 30.150000, 648.000000 3. Лепёха А. А. : 100.010000, 200.020000 4. Начальник Директоров : 2.760000, 10.370000 5. Панфилов А. С. : 16.000000, 40.000000 6. Саранцев А. А. : 35.480000, 26742.760000
2	Формирование отчета о компаниях, выполняющих данную услугу			1. "евроремонт" : {"ОАО Строй-Ремонт", "ООО Не Строим", "ООО Ремонт-Строй"} 2. "настроить пианино" : {"ИП Лепёха А. А."} 3. "поклейка обоев" : {"ОАО Строй-Ремонт", "ООО Не Строим", "ООО Ремонт-Строй"} 4. "покраска стен" : {"ОАО Строй-Ремонт", "ООО Не Строим", "ООО Ремонт-Строй"} 5. "посчитать предикат" : {"ИП Саранцев А. А."} 6. "прикрутить полочку" : {"ООО Чинилкин", "ИП 'Не ломай'"} 7. "сделать лабораторную по программированию" : {"ИП Саранцев А. А."} 8. "секретная услуга2" : {"ООО Ремонт-Строй", "ИП 'Не ломай'"} 9. "услуга1" : {"ИП 'сам себе ремонт'", "ИП 'Не ломай'"}

Заключение

Целью курсового проекта было: разработать информационную систему для автоматизации работы со справочниками налоговой службы, которая должна обрабатывать информацию о компаниях, занимающихся деятельностью в сфере ремонта.

В итоге цель была достигнута. Для её достижения были выполнены следующие задачи:

- Был проведён анализ предметной области и построена её модель
- Были изучены теоретические основы построения структур данных и построения, на их основе, требуемых справочников.
- Были обозначены функциональные требования к информационной системе.
- Информационная система была реализована и протестирована.

Для предметной области была грамотно составлена её модель и описаны все объекты. Кроме того, были приведены примеры объектов предметной области, которые были в дальнейшем использованы. Была поставлена и выполнена задача формирования отчётов на основе объектов из предметной области.

Для построения справочников были избраны две структуры данных «Хеш-таблица» с использованием метода цепочек и «Красно-Чёрное дерево». Они были тщательно изучены и выбор был обоснован, был подробно описан алгоритм работы с этими структурами данных и приведены примеры на основе объектов из анализа предметной области.

Были подробно описаны функциональные требования, требования к входным данным для информационной системы, как с файла, так и с оконного интерфейса. Также были заданы требования к выходным данным, отчётам и оконному интерфейсу. В дальнейшем все это было реализовано на практике.

Для разработки оконного приложения была изучена и использована библиотека инструментов C/C++ wxWidgets версии 3.1.3. Также все необходимые объекты и структуры данных, а также методы и вспомогательные классы для

работы с ними, были реализованы на языке C++. Информационное средство было проверено на соответствие функциональным требованиям и протестировано.

Вся практическая часть курсовой выполнялась в средах разработки Microsoft Visual Studio 2019. Также был использован сервис для контроля версий GitHub.

Таким образом все поставленные задачи были выполнены и была достигнута цель курсовой работы – разработка информационной системы для налоговой службы.

Список литературы

1. ЛЕПЕХА А. А., РАЗРАБОТКА ИНФОРМАЦИОННОЙ СИСТЕМЫ ДЛЯ АВТОМАТИЗАЦИИ РАБОТЫ СО СПРАВОЧНИКАМИ НАЛОГОВОЙ СЛУЖБЫ, Владивосток: ДВФУ, 2020
2. Кормен, Т., Лейзерсон, Ч., Ривест, Р., Штайн, К. // Алгоритмы: построение и анализ / Под ред. И. В. Красикова. — 2-е изд. — М.: Вильямс, 2005. — 1296 с.
3. Седжвик Р. Фундаментальные алгоритмы на C++. Анализ/Структура данных/Сортировка/Поиск — К.: Диасофт, 2001. — 688 с.
4. Кнут Д., Искусство программирования, т.3. М.: Вильямс, 2000
5. Официальная документация wxWidgets 3.1.3
6. Вирт Н., Алгоритмы + структуры данных = программы, М.: Мир, 1985.
7. Paul E. Black, "associative array", Dictionary of Algorithms and Data Structures, изм. 2004.
8. Pedro Celis, Robin Hood Hashing, Canada, Waterloo: University of Waterloo, 1986, 74с.
9. Майкл Ласло, Вычислительная геометрия и компьютерная графика на C++, М.: БИНОМ, 1997, 304с.
10. The GNU C++ Library, 2020