

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**

ПОЛЯКОВ АРТЁМ ВИКТОРОВИЧ

РАЗРАБОТКА ИНФОРМАЦИОННОЙ СИСТЕМЫ ДЛЯ РАБОТЫ СО СПРАВОЧНИКОМ ПРЕДМЕТНОЙ ОБЛАСТИ «ПУНКТ ВЫДАЧИ ТОВАРОВ ОНЛАЙН МАГАЗИНА»

**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 Б9122-09.03.04 | | | | | | | |
|  |  |  | | | | Поляков. А.В. | | | |
|  | | | | | | |  |  | (подпись) | | | |  | | | |
|  | | | | | | | | | | | | | | | | |
|  | | | | | | |  |  | Руководитель | | | | | | | |
|  | | | | | | |  |  | ст. преподаватель ДПИиИИ | | | | | | | |
|  | | | | | | |  |  | ученая степень, должность | | | | |  | Крестникова О.А. | |
|  | | | | | | |  |  | (подпись) | | | | |  | (ФИО) | |
|  | | | | | | | | | | | | | | | | |
| Регистрационный № | | | | | | |  |  | Защищен с оценкой | | | | | | | |
|  | | | | | | |  |  |  | | | | | | | |
|  | | | |  |  | |  |  | « |  | » |  |  | | | 2024 г. |
| (подпись) | | | |  | (ФИО) | |  |  |  | | | | | | | |
| « |  | » |  | | | 2024 г. |  |  |  | | | | | | | |

г. Владивосток

2024

# Оглавление

[Оглавление 2](#_Toc171804387)

[Введение 3](#_Toc171804388)

[1 Анализ предметной области 4](#_Toc171804389)

[1.1 Объект предметной области 4](#_Toc171804390)

[1.2 Законы ПО 5](#_Toc171804391)

[2 Теоретическая часть 6](#_Toc171804392)

[2.1 Хеш-таблица 6](#_Toc171804393)

[2.1.1 Хеш-функция 7](#_Toc171804394)

[2.1.2 Разрешение коллизий методом цепочек. 9](#_Toc171804395)

[2.2 Бинарное дерево поиска 10](#_Toc171804396)

[3 Требования к информационной системе 12](#_Toc171804397)

[3.1 Требования к данным 12](#_Toc171804398)

[3.1.1 Требования к входным данным 12](#_Toc171804399)

[3.1.2 Требования к выходным данным 13](#_Toc171804400)

[3.2 Функциональные требования 14](#_Toc171804401)

[3.2.1 Общие требования 14](#_Toc171804402)

[4 Реализация 16](#_Toc171804403)

[4.1 Спецификация структур данных 16](#_Toc171804404)

[4.2 Описание среды разработки 34](#_Toc171804405)

[4.3 Руководство пользователя 34](#_Toc171804406)

[4.3.1 Работа со справочником «Пункт выдачи товара онлайн магазина» 35](#_Toc171804407)

[4.4 Тестирование 40](#_Toc171804408)

[Заключение 44](#_Toc171804409)

[Список литературы 45](#_Toc171804410)

# Введение

Пункт выдачи товара онлайн магазина представляет собой место, где покупатель может забрать различный заказанный им ранее товар.

Целью курсового проекта является: разработка информационной системы для работы со справочником предметной области «Пункт выдачи товара онлайн магазина».

Для достижения поставленной цели необходимо выполнить следующие задачи:

1. Провести анализ предметной области «Пункт выдачи товара онлайн магазина»;

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 справочником предметной области (ПО) «Пункт выдачи товара онлайн магазина».

Система должна решать следующие задачи:

1. хранить, позволять просматривать, добавлять и удалять информацию о выдаче товара: артикул, наименование поставщика, наименование товара, дата;
2. позволять искать информацию о выдаче товара по артикулу, наименованию поставщика и дате.

1.1 Объект предметной области

Исходя из задач, которые должна решать информационная система, выделен объект ПО, представленный в Таблице 1.

Таблица 1 – Объект ПО

|  |  |
| --- | --- |
| Тип и название объекта | Описание объекта |
| Справочник «Выдачи» | Хранит информацию по каждой выполненной выдаче товара на склад |

Каждая выдача товара характеризуется следующими параметрами: артикул, наименование поставщика, наименование товара, дата.

**Артикул** – слово без пробелов. Состоит из строчных букв латинского алфавита, арабских цифр и дефиса. Не может быть составлен из букв другого алфавита.

**Наименование поставщика** – название компании, состоящее из нескольких слов. Первое слово состоит только из заглавных букв. Слова разделены одним пробелом. Слова состоят только из букв латинского алфавита и арабских цифр. Наименование не может быть пустым. Не может быть составлено из букв другого алфавита, специальных символов и каких-то других цифр кроме арабских.

**Наименование товара** – слово или словосочетание, составленное из строчных букв латинского алфавита и арабских цифр. Слова в словосочетании разделены одним пробелом. Наименование не может быть пустым. Не может содержать заглавные буквы, буквы других алфавитов, какие-то иные цифры, а также специальные символы.

**Дата** – структура, состоящая из трёх числовых полей. Первое поле означает день месяца – двузначное число, второе поле означает номер месяц – двузначное число, третье поле означает год 21-го века – четырехзначное число. Максимальное число, допустимое для записи дня месяца, зависит от номера месяца, а также от года, в частном случае. Вводимый пользователем формат даты: 01.01.2024 - точно копируют григорианский календарь в пределах от 01.01.2000 до 31.12.2099

Пример справочника «Поставки» представлен в таблице 2.

Таблица 2 – Справочник «Поставки»

|  |  |  |  |
| --- | --- | --- | --- |
| **Артикул** | **Наименование поставщика** | **Наименование товара** | **Дата** |
| j4g5ka8qvj | PAO Gazprom | smartphone viva | 01.01.2024 |
| 5mz0zo6ugw | OOO Rosneft | smartphone | 01.01.2023 |
| wc3jv0d3dd | PAO Gazprom | gaming laptop | 01.01.2022 |
| gfnd7h06vj | OOO Lukoil | washing machine | 04.01.2021 |
| vxxm-zh8uq | PAO Gazprom | fridge | 05.02.2020 |
| tuoznyyw71 | OAO Sberbank | kettle | 06.03.2023 |
| l4v66hc6d5 | PAO Sberbank | chocolate | 12.05.2024 |
| dlqo9j4jsm | PAO Gazprom | banana | 31.06.2025 |
| jqhgbhdlat | OPP Rosneft | shrimp | 15.07.2026 |
| bq12bhdlat | OOO Rosneft | lobster | 30.01.2027 |

1.2 Законы ПО

1. в один день не может быть более одной выдачи товара одного артикула.

# 2 Теоретическая часть

Для хранения данных будет использоваться хэш-таблица (далее ХТ) и несбалансированное бинарное дерево поиска (далее БДП), которое хранится в каждой ячейке хеш-таблицы. Дата и артикул уникальны. ХТ строится по ключу (дата и артикул), а БДП, которое хранится в ячейке таблице, по наименованию поставщика. БДП используется для удобного хранения данных. Его структура поддерживает операции вставки, удаления (удаляемый элемент заменяется на минимальный справа) и поиска за O(n) времени в худшем случае и за O(log(n)) времени в лучшем случае, что упрощает доступ к данным. Данная структура позволяет динамически добавлять и удалять данные без перестройки всей структуры и использует память эффективно. В хэш-таблице для разрешения коллизии используется метод цепочек, цепочка – это и есть само БДП, хэш-функция – середина квадрата. Удаление из хэш-таблицы производится при полном совпадении всех полей, а именно даты, наименованию поставщика, наименованию товара и артикула.

2.1 Хеш-таблица

Хеш-таблица (hash table) представляет собой массив 𝑇[0..𝑚−1], в котором номер ячейки для ключа key ∈ U вычисляется посредством хеш-функции (функции хеширования). Размер m хеш-таблицы значительно меньше размера исходной совокупности ключей U. [1]

Существуют два основных типа хеш-таблиц: с цепочками и с открытой адресацией. Хеш-таблица содержит некоторый массив H, элементы которого могут быть парами (в случае хеш-таблицы с открытой адресацией) или списками пар (в случае хеш-таблицы с цепочками). [1]

Операция в хеш-таблице начинается с вычисления хеш-функции от ключа. Полученное хеш-значение i=hash(key) служит индексом в массиве H. Затем операция (добавление, удаление или поиск) перенаправляется объекту, который находится в соответствующей ячейке массива H[i]. [1]

Ситуация, когда разные ключи приводят к одинаковому хеш-значению, называется коллизией. Такие события достаточно распространены — например, при вставке в хеш-таблицу размером 365 ячеек всего лишь 23 элементов вероятность коллизии уже превышает 50% (если каждый элемент равновероятно может попасть в любую ячейку). Поэтому механизм разрешения коллизий является важной частью любой хеш-таблицы. [1]

В некоторых особых случаях удаётся полностью избежать коллизий. Например, если все ключи элементов известны заранее (или очень редко изменяются), то можно найти совершенную хеш-функцию, которая распределит их по ячейкам хеш-таблицы без коллизий. Хеш-таблицы, использующие такие хеш-функции, не требуют механизма разрешения коллизий и называются хеш-таблицами с прямой адресацией. [1]

2.1.1 Хеш-функция

Хеш-функция принимает ключ из множества U и возвращает индекс ячейки в массиве T[0..m−1]. Эта функция отображает множество ключей U на множество ячеек хеш-таблицы T[0..m−1].

hash(key):U→{0,1,...,m−1}

Значения, которые возвращает хеш-функция, называются хеш-значениями, хеш-кодами, дайджестами или просто хешами. Они используются для индексирования таблицы фиксированного размера, называемой хеш-таблицей. Применение хеш-функции для индексирования хеш-таблицы называется хешированием или рассеянной адресацией. [2]

Хеш-функции и связанные с ними хеш-таблицы применяются в программных приложениях для хранения и извлечения данных, обеспечивая доступ к данным за небольшое и почти постоянное время. Они требуют объем памяти, который лишь немного превышает общий объем, необходимый для самих данных или записей. Хеширование является эффективным способом доступа к данным с точки зрения вычислительных затрат и использования памяти, избегая непостоянного времени доступа, характерного для упорядоченных и неупорядоченных списков и структурированных деревьев, а также часто экспоненциальных требований к памяти при прямом доступе к пространствам состояний ключей большой или переменной длины. [2]

Применение хеш-функций зависит от статистических свойств взаимодействия ключа и функции: вероятность наихудшего случая крайне мала, тогда как поведение в среднем случае может быть почти оптимальным (минимальное количество коллизий).

Многочисленные тесты показали эффективность двух основных типов хеш-функций: основанных на делении и на умножении.

Однако в нашем же случае будет использоваться на много более усовершенствованный метод генерации случайных чисел: метод "середина квадрата".

Алгоритм хеш-функции метода "середина квадрата" был впервые предложен Джоном фон Нейманом в 1949 году. Он заключается в следующем: берется число, возводится в квадрат, и из середины полученного числа извлекается новое число, которое становится следующим в последовательности. Например, если исходное число 11, его квадрат 121, из которого средние цифры будут получаться в зависимости от размера хеш-таблицы. Например: размер таблицы 10 – средина будет 2, размер таблицы 100 – середина будет 12. Этот процесс повторяется, генерируя псевдослучайные числа. [2]

Пример:

Размер ХТ: M = 10.

1. Hash(j4g5ka8qvj, 21.08.2090) = 1;
2. Hash(5mz0zo6ugw, 21.12.2090) = 3;
3. Hash(wc3jv0d3dd, 05.07.2019) = 5;
4. Hash(gfnd7h06vj, 10.05.2021) = 6;
5. Hash(vxxm-zh8uq, 13.02.2064) = 7;
6. Hash(tuoznyyw71, 13.08.2016) = 7;
7. Hash(l4v66hc6d5, 12.02.2008) = 7;
8. Hash(dlqo9j4jsm, 21.08.2090) = 8;
9. Hash(jqhgbhdlat, 09.12.2036) = 10;
10. Hash(bq12bhdlat, 03.06.2017) = 5.

Как можно заметить, в примерах (3, 10) и (5, 6, 7) хэши получились одинаковыми. Есть два метода разрешения коллизии: открытая адресация и метод цепочек. В данном проекте будет использоваться метод цепочек, так как он имеет ряд преимуществ, например: простота реализации, гибкость по отношению к размеру таблицы, отсутствие необходимости перераспределения.

2.1.2 Разрешение коллизий методом цепочек.

Технология объединения элементов заключается в том, что элементы множества с одинаковым хеш-значением организуются в связный список.

В ячейке под номером i хранится указатель на корень БДП, значение хеша элементов которого равны i. Если таких элементов нет, в ячейке i содержится NULL.

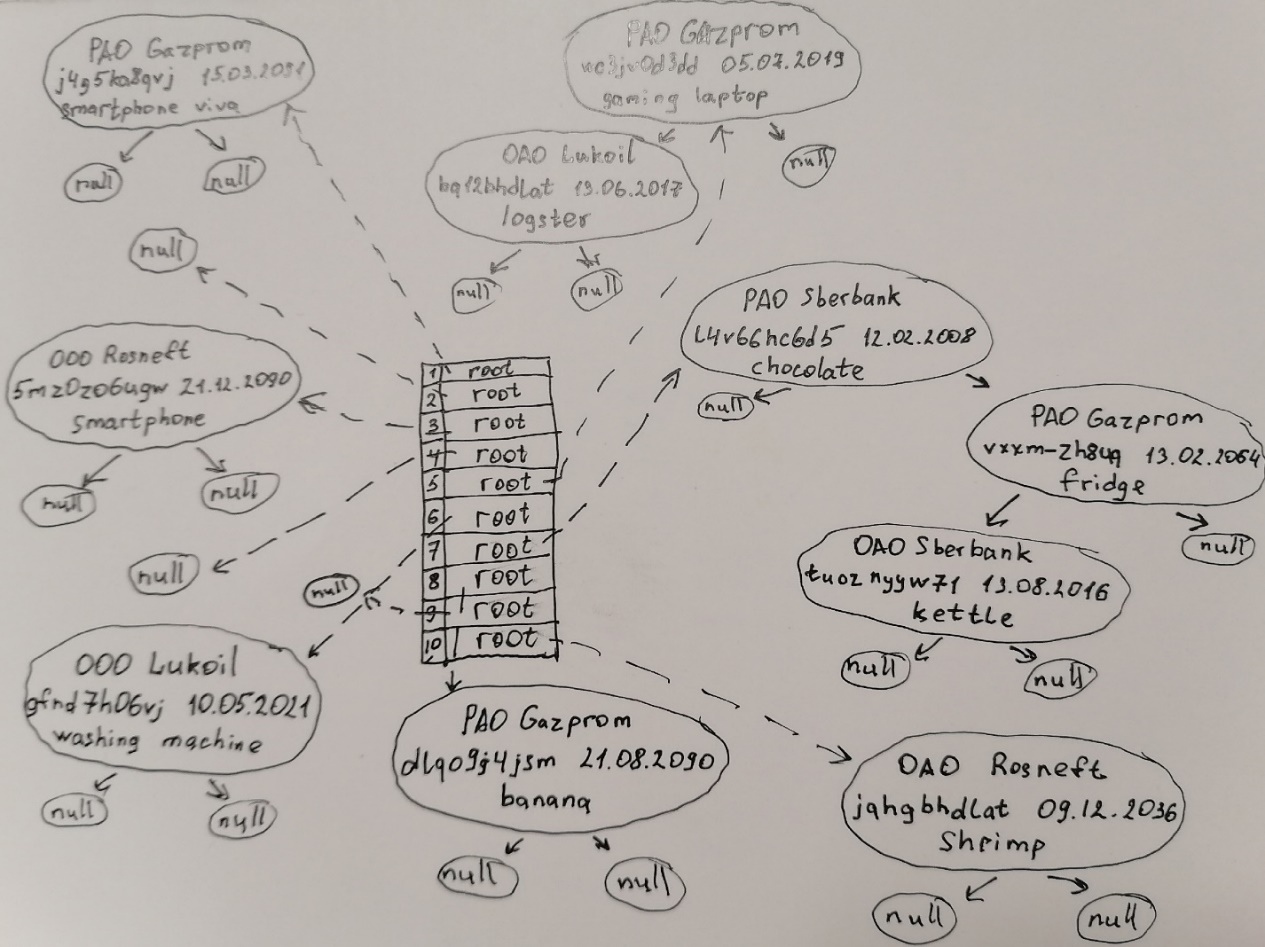


Рисунок 1 – пример хеш-таблицы с деревом в каждой ячейке для справочника п. 1.1

2.2 Бинарное дерево поиска

Несбалансированное бинарное дерево поиска (БДП) — это структура данных, в которой отсутствует гарантированная сбалансированность высоты поддеревьев. Такое дерево имеет следующие особенности:

1. Высота поддеревьев: в несбалансированном бинарном дереве высота левого и правого поддеревьев может значительно различаться;
2. Зависимость времени выполнения операций от количества узлов: время выполнения операций (поиск, вставка, удаление) в несбалансированном БДП в худшем случае пропорционально числу узлов, т.е. O(n). В лучшем случае – O(log(n));
3. Высокая вероятность значительного дисбаланса.

Операции:

1. Поиск. Поиск узла в несбалансированном БДП начинается с корня и идет вниз по дереву, сравнивая ключи в каждом узле. Если дерево несбалансировано, время поиска может быть очень большим, в худшем случае O(n), в лучшем – O(log(n));
2. Вставка. Чтобы вставить новый узел, алгоритм проходит от корня дерева вниз, сравнивая ключи, пока не найдет подходящее место для нового узла. Вставка поддерживает порядок элементов в дереве. Если дерево становится сильно несбалансированным, вставка также может занять как O(n) времени, так и O(log(n));
3. Удаление. Удаление узла в несбалансированном БДП также начинается с поиска узла для удаления. Если узел найден, возможны три случая:
   1. У узла нет потомков (он является листом): узел просто удаляется;
   2. У узла один потомок: узел удаляется, и его потомок заменяет его;
   3. У узла два потомка: необходимо найти наименьший узел в правом поддереве, заменить удаляемый узел на найденный, а затем удалить найденный узел. [3]

Несбалансированное БДП имеет высокую вероятность вырождения в линейный список, что значительно снижает эффективность операций. Тем не менее имеет очень простую реализацию и удобство в использовании, в связи с чем будет применено в проекте.

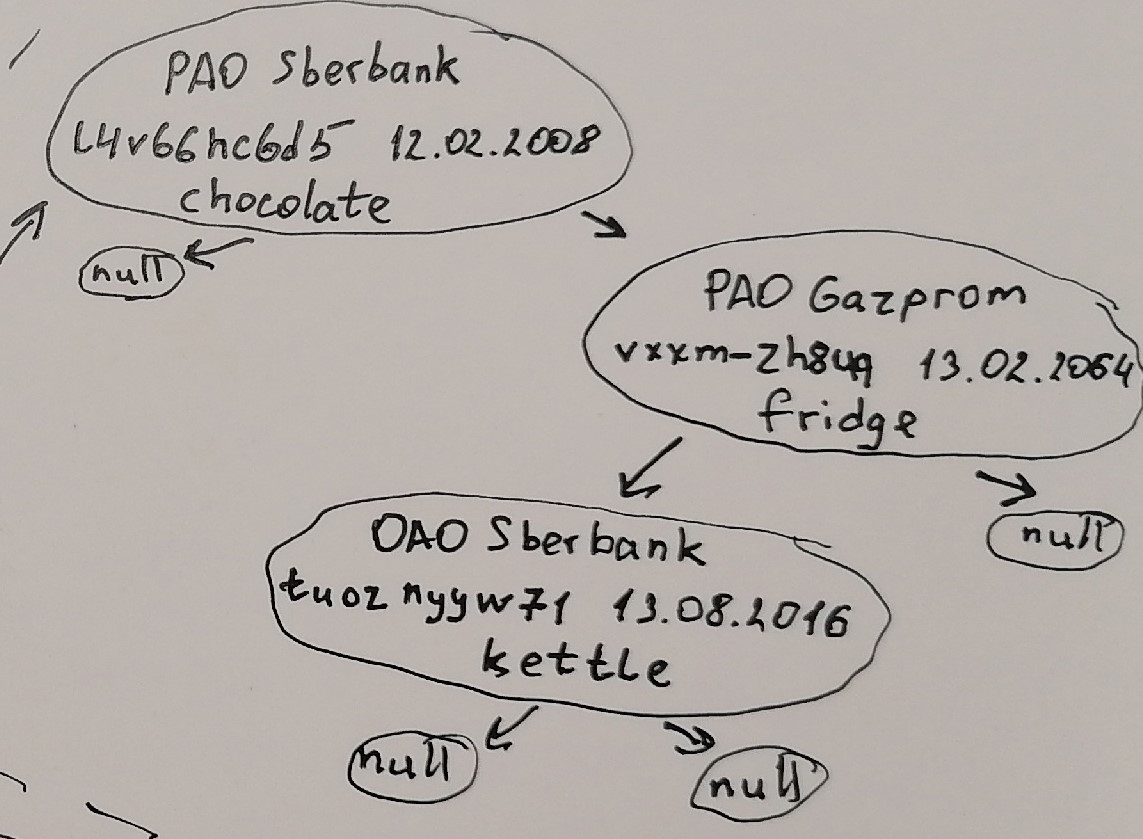


Рисунок 2 – бинарное дерево поиска из ячейки хеш-таблицы для справочника п. 1.1

# 3 Требования к информационной системе

В данной главе описываются требования к информационной системе для работы со справочником, 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 input.txt, каждая строка файла содержит информацию об объекте выдаваемый товар, а именно артикула (слово без пробелов, состоящее из строчных букв латинского алфавита, арабских цифр и дефиса); наименования поставщика (название компании, состоящее из нескольких слов, разделенных символом « », где первое слово состоит только из заглавных букв латинского алфавита и арабских цифр); наименования товара (слово или словосочетание, составленное из строчных букв латинского алфавита и арабских цифр, разделенных символом « »); даты (структура, состоящая из трёх числовых полей, разделенных символом «.», где первое поле означает день месяца – число от 1 до 31, второе поле означает номер месяц – число от 1 до 12, третье поле означает год 21-го века – четырехзначное число от 2000 до 2099); разделенные символом « ».

Пример текстового файла:

PAO Sberbank l4v66hc6d5 12.02.2008 chocolate

OAO Rosneft jqhgbhdlat 09.12.2036 shrimp

PAO Gazprom j4g5ka8qvj 15.03.2091 smartphone viva

PAO Gazprom vxxm-zh8uq 13.02.2064 fridge

PAO Gazprom wc3jv0d3dd 05.07.2019 gaming laptop

OAO Sberbank tuoznyyw71 13.08.2016 kettle

PAO Gazprom dlqo9j4jsm 21.08.2090 banana

OOO Lukoil gfnd7h06vj 10.05.2021 washing machine

OOO Rosneft 5mz0zo6ugw 21.12.2090 smartphone

OAO Lukoil bq12bhdlat 03.06.2017 lobster

* Наименование поставщика (см. п. 1.1)
* Артикул (см. п. 1.1)
* Дата (см. п. 1.1)
* Наименование товара (см. п. 1.1)

3.1.2 Требования к выходным данным

Выходными данными являются:

* текстовый файл output.txt, каждая строка файла содержит информацию об объекте выдаваемый товар, а именно артикула (слово без пробелов, состоящее из строчных букв латинского алфавита, арабских цифр и дефиса); наименования поставщика (название компании, состоящее из нескольких слов, разделенных символом « », где первое слово состоит только из заглавных букв латинского алфавита и арабских цифр); наименования товара (слово или словосочетание, составленное из строчных букв латинского алфавита и арабских цифр, разделенных символом « »); даты (структура, состоящая из трёх числовых полей, разделенных символом «.», где первое поле означает день месяца – число от 1 до 31, второе поле означает номер месяц – число от 1 до 12, третье поле означает год 21-го века – четырехзначное число от 2000 до 2099); разделенные символом « ».

Сообщения:

* File loaded successfully!;
* Elements were inputted from + «filePath» +.;
* Table was printed.;
* Please fill in the field.;
* Please enter a valid natural index from table.;
* Tree in + «data» + cell of table was printed.;
* Please enter all required data in the correct format.;
* Please enter a date in format: dd.mm.yyyy.;
* Invalid input data.;
* Element added successfully!;
* Element was inserted in the table by hand.;
* Element already exists.;
* Element removed successfully!;
* Element was removed from the table by hand.;
* Unable to open file for writing.;
* Table exported to + «filePath» + successfully!;
* Table was exported to the file + «filePath»;
* Element not found in the table.;
* Element found in + «found» + cell of the table.;
* IP Polyakov l4v66hc6d5 10.07.2024 gaming laptop;
* Example was printed.;
* Basic list was updated.;
* Hash table size set to + «number»;
* Hash table cleared.;
* Enter a valid index in range of table size..

3.2 Функциональные требования

3.2.1 Общие требования

Информационная система должна позволять:

1. Считывать данные для справочника из текстового файла (см. п. 3.1.1). Для считывания проверяется следующее условие: полнота, корректность и безопасность данных;
2. Сохранять данные справочника в текстовый файл (см. п. 3.1.2);
3. Добавлять информацию в справочник по значениям: артикул, наименование поставщика, дата. Для функции добавления проверяются следующие условия: проверка правильности формата артикула, проверка правильности формата наименования поставщика, проверка правильности формата даты, проверка на дублирование данных;
4. Проверять добавляемые значения в справочник на корректность (см. п. 1.1);
5. Удалять информацию из справочника по значения: артикул, наименование поставщика, дата, наименование товара. Для функции удаления проверяются следующие условия: проверка правильности формата наименования поставщика, проверка правильности формата артикула, проверка правильности формата даты, проверка существования комбинации для удаления;
6. Находить информацию по значениям: дата, наименование поставщика, артикул. Для функции поиска проверяются следующие условия: проверка правильности формата наименования поставщика, проверка правильности формата артикула, проверка правильности формата даты;
7. В окне отладки выводятся сообщения об ошибках и сообщения о совершенных действиях.

# 4 Реализация

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

4.1 Спецификация структур данных

* Класс **Element** - класс, описывающий структуру элемента, хранящегося в узлах БДП.

**Поля:**

std::string \_articul – артикул товара.

std::string \_nameSupplier – название поставщика.

std::string \_nameProduct – название товара.

Date \_date – дата выдачи.

**Методы:**

Element() – конструктор по умолчанию.

Входные данные: отсутствуют.

Выходные данные: объект класса Element с полями по умолчанию.

Element(string articul, string nameSupplier, string nameProduct, const Date &date) – конструктор с параметрами.

Входные данные: артикул товара, название поставщика, название товара, дата.

Формальные параметры:

string articul - артикул товара;

string nameSupplier - название поставщика;

string nameProduct - название товара;

const Date &date – дата выдачи.

Выходные данные: объект класса Element с инициализированными полями.

std::string getArticul() const – метод для получения артикула товара.

Входные данные: отсутствуют.

Формальные параметры: отсутствуют.

Выходные данные: строка с артикулом товара.

std::string getNameSupplier() const – метод для получения названия поставщика.

Входные данные: отсутствуют.

Формальные параметры: отсутствуют.

Выходные данные: строка с названием поставщика.

std::string getNameProduct() const – метод для получения названия товара.

Входные данные: отсутствуют.

Формальные параметры: отсутствуют.

Выходные данные: строка с названием товара.

Date getDate() const – метод для получения даты, связанной с товаром.

Входные данные: отсутствуют.

Формальные параметры: отсутствуют.

Выходные данные: объект класса Date.

void setArticul(const std::string &articul) – метод для установки артикула товара.

Входные данные: строка с артикулом товара.

Формальные параметры: const std::string &articul - строка с артикулом товара.

Выходные данные: отсутствуют.

void setNameSupplier(const std::string &nameSupplier) – метод для установки названия поставщика.

Входные данные: строка с названием поставщика.

Формальные параметры: const std::string &nameSupplier - строка с названием поставщика.

Выходные данные: отсутствуют.

void setNameProduct(const std::string &nameProduct) – метод для установки названия товара.

Входные данные: строка с названием товара.

Формальные параметры: const std::string &nameProduct - строка с названием товара.

Выходные данные: отсутствуют.

void setDate(const Date &date) – метод для установки даты, связанной с товаром.

Входные данные: объект класса Date.

Формальные параметры: const Date &date - объект класса Date.

Выходные данные: отсутствуют.

bool lessKeyHT(const Element &element) – метод для сравнения ключей элементов в хеш-таблице.

Входные данные: элемент для сравнения.

Формальные параметры: const Element &element - элемент для сравнения.

Выходные данные: логическое значение, указывающее, меньше ли текущий элемент по ключу.

bool operator < (const Element &element) const – оператор для сравнения элементов по ключу (меньше).

Входные данные: элемент для сравнения.

Формальные параметры: const Element &element - элемент для сравнения.

Выходные данные: логическое значение, указывающее, меньше ли текущий элемент по ключу.

bool operator > (const Element &element) const – оператор для сравнения элементов по ключу (больше).

Входные данные: элемент для сравнения.

Формальные параметры: const Element &element - элемент для сравнения.

Выходные данные: логическое значение, указывающее, больше ли текущий элемент по ключу.

bool operator == (const Element &element) const – оператор для сравнения элементов на равенство.

Входные данные: элемент для сравнения.

Формальные параметры: const Element &element - элемент для сравнения.

Выходные данные: логическое значение, указывающее, равны ли текущий элемент и элемент для сравнения.

bool operator != (const Element &element) const – оператор для сравнения элементов на неравенство.

Входные данные: элемент для сравнения.

Формальные параметры: const Element &element - элемент для сравнения.

Выходные данные: логическое значение, указывающее, не равны ли текущий элемент и элемент для сравнения.

friend ostream& operator << (ostream& out, const Element &element) – оператор для вывода элемента в поток.

Входные данные: поток вывода и элемент для вывода.

Формальные параметры: ostream& out - поток вывода; const Element &element - элемент для вывода.

Выходные данные: поток вывода с информацией об элементе.

* Класс **Date** - класс, описывающий дату.

**Поля:**

short m\_year – год.

short m\_month – месяц.

short m\_day – день.

**Методы:**

Date() – конструктор по умолчанию.

Входные данные: отсутствуют.

Выходные данные: объект класса Date с полями по умолчанию (0).

Date(short \_year, short \_month, short \_day) – конструктор с параметрами.

Входные данные: год, месяц, день.

Формальные параметры:

short \_year - год;

short \_month - месяц;

short \_day - день.

Выходные данные: объект класса Date с инициализированными полями.

explicit Date(const string &date) – конструктор с параметром строки.

Входные данные: строка с датой в формате "dd.mm.yyyy".

Формальные параметры: const string &date - строка с датой.

Выходные данные: объект класса Date с инициализированными полями.

~Date() – деструктор.

Входные данные: отсутствуют.

Выходные данные: отсутствуют.

bool operator==(const Date &date) const – оператор для сравнения дат на равенство.

Входные данные: дата для сравнения.

Формальные параметры: const Date &date - дата для сравнения.

Выходные данные: логическое значение, указывающее, равны ли даты.

bool operator!=(const Date &date) const – оператор для сравнения дат на неравенство.

Входные данные: дата для сравнения.

Формальные параметры: const Date &date - дата для сравнения.

Выходные данные: логическое значение, указывающее, не равны ли даты.

bool operator>(const Date &date) const – оператор для сравнения дат (больше).

Входные данные: дата для сравнения.

Формальные параметры: const Date &date - дата для сравнения.

Выходные данные: логическое значение, указывающее, больше ли текущая дата.

bool operator<(const Date &date) const – оператор для сравнения дат (меньше).

Входные данные: дата для сравнения.

Формальные параметры: const Date &date - дата для сравнения.

Выходные данные: логическое значение, указывающее, меньше ли текущая дата.

void operator=(const Date &date) – оператор присваивания.

Входные данные: дата для присваивания.

Формальные параметры: const Date &date - дата для присваивания.

Выходные данные: отсутствуют.

friend ostream &operator<<(ostream &out, const Date &date) – оператор для вывода даты в поток.

Входные данные: поток вывода и дата для вывода.

Формальные параметры:

ostream &out - поток вывода;

const Date &date - дата для вывода.

Выходные данные: поток вывода с информацией о дате.

string toString() const – метод для получения строки, представляющей дату.

Входные данные: отсутствуют.

Формальные параметры: отсутствуют.

Выходные данные: строка в формате "dd.mm.yyyy".

void fromString(const std::string& dateStr) – метод для инициализации даты из строки.

Входные данные: строка с датой в формате "dd.mm.yyyy".

Формальные параметры: const std::string& dateStr - строка с датой.

Выходные данные: отсутствуют.

short getYear() const – метод для получения года.

Входные данные: отсутствуют.

Формальные параметры: отсутствуют.

Выходные данные: значение года.

short getMonth() const – метод для получения месяца.

Входные данные: отсутствуют.

Формальные параметры: отсутствуют.

Выходные данные: значение месяца.

short getDay() const – метод для получения дня.

Входные данные: отсутствуют.

Формальные параметры: отсутствуют.

Выходные данные: значение дня.

bool correctDate() const – метод для проверки корректности даты.

Входные данные: отсутствуют.

Формальные параметры: отсутствуют.

Выходные данные: логическое значение, указывающее, корректна ли дата.

* Класс **Node** - класс, описывающий узел дерева.

**Поля:**

Node \*\_left – указатель на левого потомка узла дерева.

Node \*\_right – указатель на правого потомка узла дерева.

Element \_element – объект класса Element, хранящий данные узла.

**Методы:**

Node() – конструктор по умолчанию.

Входные данные: отсутствуют.

Выходные данные: объект класса Node с полями по умолчанию (указатели \_left и \_right инициализируются как nullptr).

explicit Node(Element element, Node \*left=nullptr, Node \*right=nullptr) – конструктор с параметрами.

Входные данные: элемент, указатель на левого потомка, указатель на правого потомка.

Формальные параметры:

Element element - элемент узла дерева;

Node \*left=nullptr - указатель на левого потомка (по умолчанию nullptr);

Node \*right=nullptr - указатель на правого потомка (по умолчанию nullptr).

Выходные данные: объект класса Node с инициализированными полями.

Element getElement() const – метод для получения элемента узла.

Входные данные: отсутствуют.

Формальные параметры: отсутствуют.

Выходные данные: объект класса Element, хранящийся в узле.

void setElement(const Element &element) – метод для установки элемента узла.

Входные данные: элемент для установки.

Формальные параметры: const Element &element - элемент для установки.

Выходные данные: отсутствуют.

* Класс **Tree** - класс, несбалансированное описывающий бинарное дерево.

**Поля:**

Node\* root – указатель на корневой узел дерева.

**Публичные методы:**

Tree() – конструктор по умолчанию.

Входные данные: отсутствуют.

Выходные данные: объект класса Tree с полем root, инициализированным как nullptr.

explicit Tree(const Element& element) – конструктор с параметром элемента.

Входные данные: элемент для корневого узла дерева.

Формальные параметры:

const Element& element - элемент для корневого узла.

Выходные данные: объект класса Tree с инициализированным корневым узлом.

~Tree() – деструктор.

Входные данные: отсутствуют.

Выходные данные: отсутствуют. Освобождает память, занятую узлами дерева.

bool insert(const Element &element) – метод для вставки элемента в дерево.

Входные данные: элемент для вставки.

Формальные параметры: const Element &element - элемент для вставки.

Выходные данные: логическое значение, указывающее, был ли элемент успешно вставлен.

bool remove(const Element &element) – метод для удаления элемента из дерева.

Входные данные: элемент для удаления.

Формальные параметры: const Element &element - элемент для удаления.

Выходные данные: логическое значение, указывающее, был ли элемент успешно удален.

bool find(const Element &element) const – метод для поиска элемента в дереве.

Входные данные: элемент для поиска.

Формальные параметры: const Element &element - элемент для поиска.

Выходные данные: логическое значение, указывающее, найден ли элемент.

bool isEmpty() const – метод для проверки, пусто ли дерево.

Входные данные: отсутствуют.

Формальные параметры: отсутствуют.

Выходные данные: логическое значение, указывающее, пусто ли дерево.

void print(std::ostream& os) const – метод для печати дерева в поток (для приложения).

Входные данные: поток вывода.

Формальные параметры: std::ostream& os - поток вывода.

Выходные данные: отсутствуют.

void print() const – метод для печати дерева в консоль.

Входные данные: отсутствуют.

Формальные параметры: отсутствуют.

Выходные данные: отсутствуют.

void printTree(std::ostream& os) const – метод для красивой печати дерева в поток (для приложения).

Входные данные: поток вывода.

Формальные параметры: std::ostream& os - поток вывода.

Выходные данные: отсутствуют.

void printTree() const – метод для красивой печати дерева в консоль.

Входные данные: отсутствуют.

Формальные параметры: отсутствуют.

Выходные данные: отсутствуют.

Node\* getRoot() const – метод для получения корневого узла дерева.

Входные данные: отсутствуют.

Формальные параметры: отсутствуют.

Выходные данные: указатель на корневой узел дерева.

**Приватные методы:**

static Node\* insertRec(Node \*&node, const Element &element, bool& inserted) – рекурсивный метод для вставки элемента в дерево.

Входные данные: указатель на узел дерева, элемент для вставки, логическое значение для указания успешности вставки.

Формальные параметры:

Node \*&node - указатель на узел дерева;

const Element &element - элемент для вставки;

bool& inserted - логическое значение для указания успешности вставки.

Выходные данные: указатель на узел дерева.

static bool removeRec(Node \*&node, const Element &element) – рекурсивный метод для удаления элемента из дерева.

Входные данные: указатель на узел дерева, элемент для удаления.

Формальные параметры:

Node \*&node - указатель на узел дерева;

const Element &element - элемент для удаления.

Выходные данные: логическое значение, указывающее, был ли элемент успешно удален.

static void removeMinRight(Node \*&root, Node \*&q) – метод для удаления узла с минимальным значением в правом поддереве.

Входные данные: указатель на корень дерева, указатель на узел для замены.

Формальные параметры:

Node \*&root - указатель на корень дерева;

Node \*&q - указатель на узел для замены.

Выходные данные: отсутствуют.

static bool findRec(const Node \*node, const Element &element) – рекурсивный метод для поиска элемента в дереве.

Входные данные: указатель на узел дерева, элемент для поиска.

Формальные параметры:

const Node \*node - указатель на узел дерева;

const Element &element - элемент для поиска.

Выходные данные: логическое значение, указывающее, найден ли элемент.

static void deleteTree(Node \*&node) – рекурсивный метод для удаления всех узлов дерева.

Входные данные: указатель на узел дерева.

Формальные параметры: Node \*&node - указатель на узел дерева.

Выходные данные: отсутствуют.

static void printRec(std::ostream& os, const Node\* node) – рекурсивный метод для печати дерева в поток (для приложения).

Входные данные: поток вывода, указатель на узел дерева.

Формальные параметры:

std::ostream& os - поток вывода;

const Node\* node - указатель на узел дерева.

Выходные данные: отсутствуют.

static void printRec(const Node \*node) – рекурсивный метод для печати дерева в консоль.

Входные данные: указатель на узел дерева.

Формальные параметры: const Node \*node - указатель на узел дерева.

Выходные данные: отсутствуют.

static void drawTree(std::ostream& os, const Node\* node, short space) – рекурсивный метод для красивой печати дерева в поток (для приложения).

Входные данные: поток вывода, указатель на узел дерева, отступ.

Формальные параметры:

std::ostream& os - поток вывода;

const Node\* node - указатель на узел дерева;

short space - отступ.

Выходные данные: отсутствуют.

static void drawTree(const Node \*node, short space) – рекурсивный метод для красивой печати дерева в консоль.

Входные данные: указатель на узел дерева, отступ.

Формальные параметры:

const Node \*node - указатель на узел дерева;

short space - отступ.

Выходные данные: отсутствуют.

* Класс **Key** - класс, описывающий ключ для хеш-таблицы, содержащий дерево и статус.

**Поля:**

short \_status – статус ключа.

Tree \*\_tree – указатель на дерево.

**Методы:**

Key() – конструктор по умолчанию.

Входные данные: отсутствуют.

Выходные данные: объект класса Key с полями \_status, инициализированными как 0, и \_tree, инициализированными как nullptr.

Key(Tree \*tree) – конструктор с параметром.

Входные данные: указатель на дерево.

Формальные параметры:

Tree \*tree - указатель на дерево.

Выходные данные: объект класса Key с инициализированными полями \_status и \_tree.

Tree \*getTree() const – метод для получения дерева, связанного с ключом.

Входные данные: отсутствуют.

Формальные параметры: отсутствуют.

Выходные данные: указатель на дерево (объект класса Tree).

short getStatus() const – метод для получения статуса ключа.

Входные данные: отсутствуют.

Формальные параметры: отсутствуют.

Выходные данные: значение статуса ключа.

void setStatus(short status) – метод для установки статуса ключа.

Входные данные: статус ключа.

Формальные параметры: short status - значение статуса ключа.

Выходные данные: отсутствуют.

* Класс **HashTable** - класс, описывающий хеш-таблицу, использующую деревья для хранения элементов.

**Поля:**

vector<Key\*> table – вектор указателей на объекты класса Key, представляющих ячейки хеш-таблицы.

unsigned countElements – количество элементов в хеш-таблице.

**Публичные методы:**

HashTable(unsigned size) – конструктор с параметром размера таблицы.

Входные данные: размер таблицы.

Формальные параметры: unsigned size - размер хеш-таблицы.

Выходные данные: объект класса HashTable с инициализированной таблицей заданного размера.

bool insert(const Element &element) – метод для вставки элемента в хеш-таблицу.

Входные данные: элемент для вставки.

Формальные параметры: const Element &element - элемент для вставки.

Выходные данные: логическое значение, указывающее, был ли элемент успешно вставлен.

void remove(const Element &element) – метод для удаления элемента из хеш-таблицы.

Входные данные: элемент для удаления.

Формальные параметры: const Element &element - элемент для удаления.

Выходные данные: отсутствуют.

unsigned find(const Element &element) const – метод для поиска элемента в хеш-таблице.

Входные данные: элемент для поиска.

Формальные параметры: const Element &element - элемент для поиска.

Выходные данные: индекс ячейки хеш-таблицы, содержащей элемент, или -1, если элемент не найден.

void print(std::ostream& os) const – метод для печати хеш-таблицы в поток.

Входные данные: поток вывода.

Формальные параметры: std::ostream& os - поток вывода.

Выходные данные: отсутствуют.

void print() const – метод для печати хеш-таблицы в консоль.

Входные данные: отсутствуют.

Формальные параметры: отсутствуют.

Выходные данные: отсутствуют.

void printTree(std::ostream& os, unsigned short index) const – метод для красивой печати дерева из заданной ячейки хеш-таблицы в поток.

Входные данные: поток вывода, индекс ячейки.

Формальные параметры:

std::ostream& os - поток вывода;

unsigned short index - индекс ячейки.

Выходные данные: отсутствуют.

void printTree(unsigned short index) const – метод для красивой печати дерева из заданной ячейки хеш-таблицы в консоль.

Входные данные: индекс ячейки.

Формальные параметры: unsigned short index - индекс ячейки.

Выходные данные: отсутствуют.

void inputFile(const string &path) – метод для ввода данных в хеш-таблицу из файла.

Входные данные: путь к файлу.

Формальные параметры: const string &path - путь к файлу.

Выходные данные: отсутствуют.

Element getElement(unsigned index) const – метод для получения элемента из заданной ячейки хеш-таблицы.

Входные данные: индекс ячейки.

Формальные параметры: unsigned index - индекс ячейки.

Выходные данные: объект класса Element.

const std::vector<Key\*>& getKeys() const – метод для получения вектора ключей хеш-таблицы.

Входные данные: отсутствуют.

Формальные параметры: отсутствуют.

Выходные данные: вектор указателей на объекты класса Key.

bool validateData(const std::string &nameSupplier, const std::string &articul, const std::string &nameProduct) – метод для проверки корректности данных.

Входные данные: название поставщика, артикул, название продукта.

Формальные параметры:

const std::string &nameSupplier - название поставщика;

const std::string &articul - артикул;

const std::string &nameProduct - название продукта.

Выходные данные: логическое значение, указывающее, корректны ли данные.

**Приватные методы:**

unsigned hashFunction(const Element &element) const – метод для вычисления хеш-функции.

Входные данные: элемент.

Формальные параметры: const Element &element - элемент.

Выходные данные: значение хеша для элемента.

4.2 Описание среды разработки

Среда разработки, выбранная для реализации информационной системы, — Qt Creator. Это интегрированная среда разработки (IDE), предназначенная специально для работы с фреймворком Qt, который используется для создания кроссплатформенных приложений на C++.

4.3 Руководство пользователя

При запуске приложения CourseProject, открывается окно MainWindow.

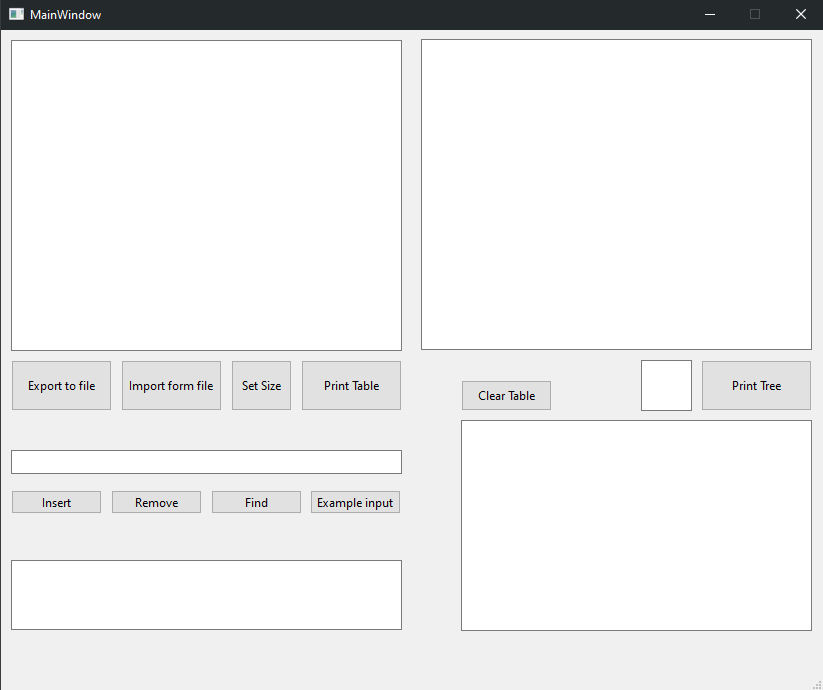


Рисунок 3 - «Окно «Главное окно»»

4.3.1 Работа со справочником «Пункт выдачи товара онлайн магазина»

На рисунке 3 мы видим кнопки. Нажав на кнопку «Set Size», мы можем задать начальный размер таблицы. Также, данная кнопка удаляет старую таблицу, если таковая была. По умолчанию размер таблицы – 10.

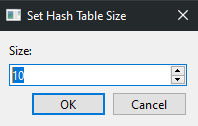


Рисунок 4 – «Окно выбора начального размера таблицы»

Будучи в главном меню, рисунок 3, нажав на кнопку «Import from file», мы можем выбрать файл, из которого в справочник будут загружены все данные.

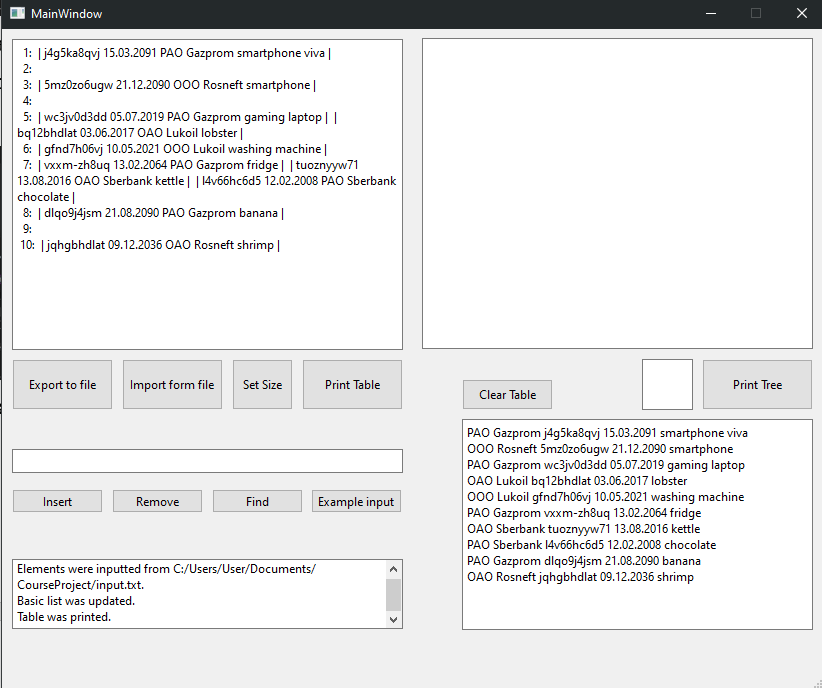


Рисунок 5 – «Загрузка данных из файла»

На рисунке 5 можно заметить в правом нижнем окне, что все данные загрузились в справочник. Однако, помимо этого у нас также вывелась хеш-таблица данного справочника в левом верхнем окне. Левое нижнее окно отвечает за логирование действий, которые были произведены – там можно ознакомиться более детально с тем, что программа делает, когда нажимается одна из кнопок.

Если мы вернемся к главному окну, рисунок 3, мы можем нажать на кнопку «Export to file». Данная кнопка полностью сохранит текущий справочник в файле output.txt на рабочий стол.

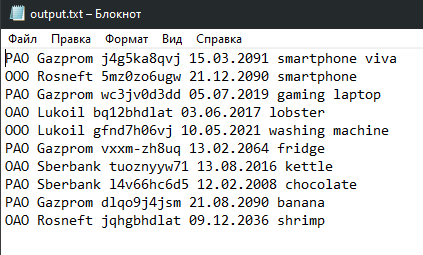


Рисунок 6 – «Файл output.txt после сохранения»

Также, на главном экране, рисунок 3, можно нажать на кнопку «Example input», результатом работы которой будет отображение примера входной информации, которую мы можем ввести в пустую строку над этой кнопкой.

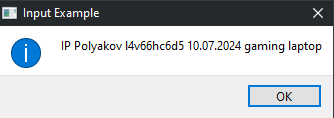


Рисунок 7 – «Пример ввода»

Введя пример в строку, нам открывается функционал трех кнопок: «Insert», «Remove», «Find».

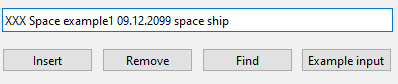


Рисунок 8 – «Кнопки Insert, Remove, Find»

Нажав на кнопку «Insert», рисунок 8, мы вставим элемент в справочник, а как следствие и в таблицу.

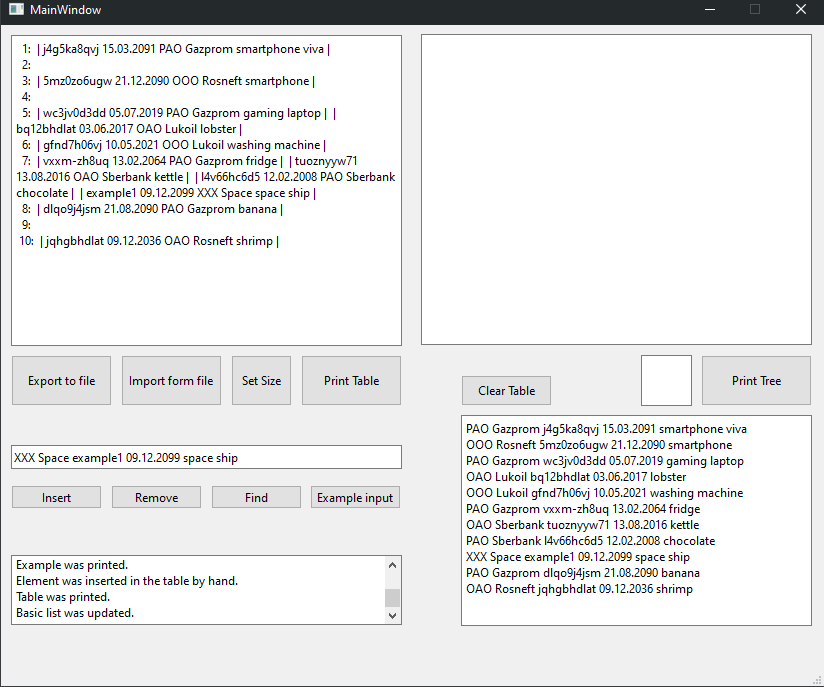


Рисунок 9 – «Результат работы кнопки Insert»

Нажав на кнопку «Remove», рисунок 8, мы удалим элемент из справочника, а как следствие и из таблицы.

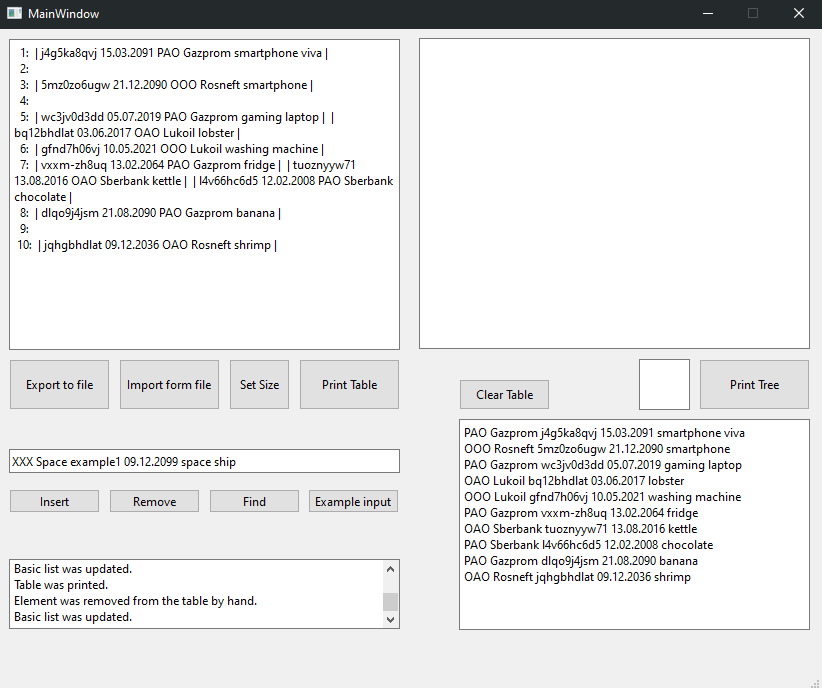


Рисунок 10 – «Результат работы кнопки Remove»

Нажав на кнопку «Find», рисунок 8, вернувшись к таблице, которая была на рисунке 9, мы найдем элемент из справочника, и выведется его положение в хеш-таблице.

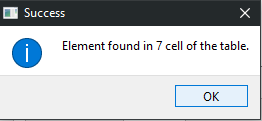


Рисунок 11 – «Результат работы кнопки Find»

Вернемся к главному экрану, рисунок 3. Здесь мы можем увидеть кнопку «Print Tree» и пустое окошко слева от нее. Кнопка выводит бинарное дерево поиска из определенной ячейки в окно над самой кнопкой. В окошко, находящееся слева от кнопки, следует ввести номер ячейки хеш-таблицы, дерево которой мы хотим вывести.

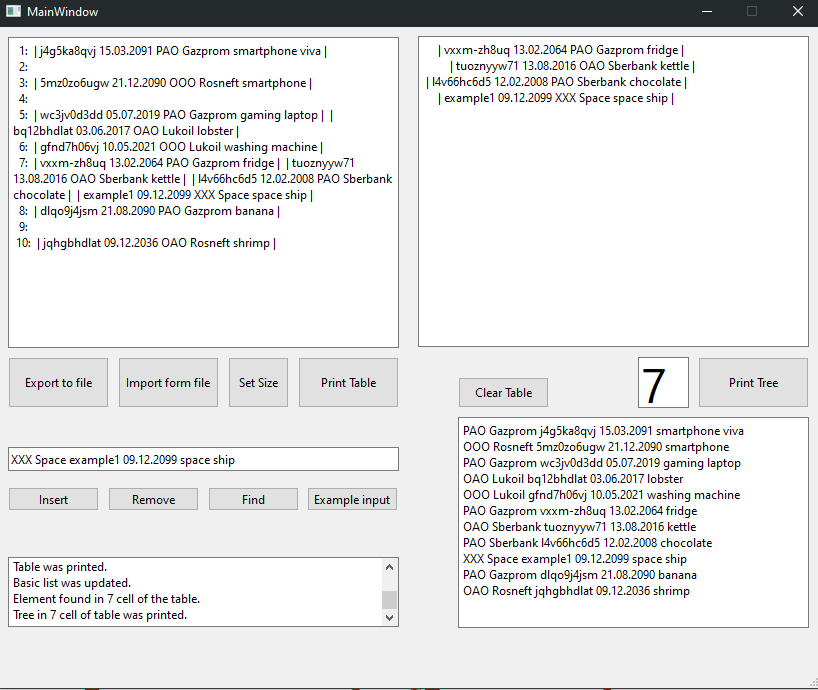


Рисунок 12 – «Результат работы кнопки Print Tree»

Помимо этого, если мы хотим заново начать работу с таблицами – создать новую таблицу, мы можем нажать на кнопку «Clear Table» на главном экране, рисунок 3, что полностью удалит справочник и задаст размер по умолчанию: 10.

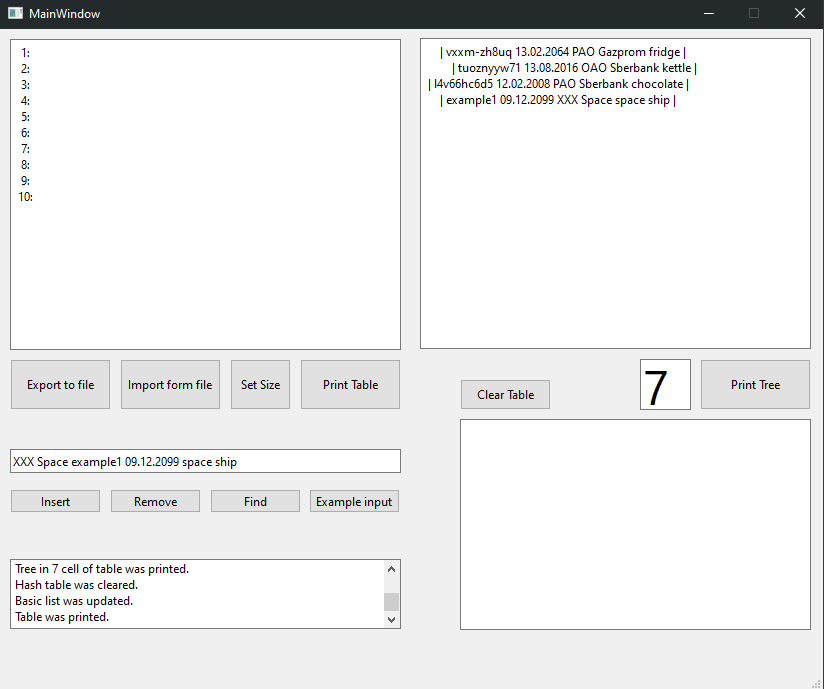


Рисунок 13 – «Пример работы кнопки Clear Table»

На главном экране, рисунок 3, можно также воспользоваться кнопкой «Print Table», которая может понадобиться в случае, когда мы решили задать начальный размер таблицы, чтобы отобразить на экране таблицу с заданным начальным размером.

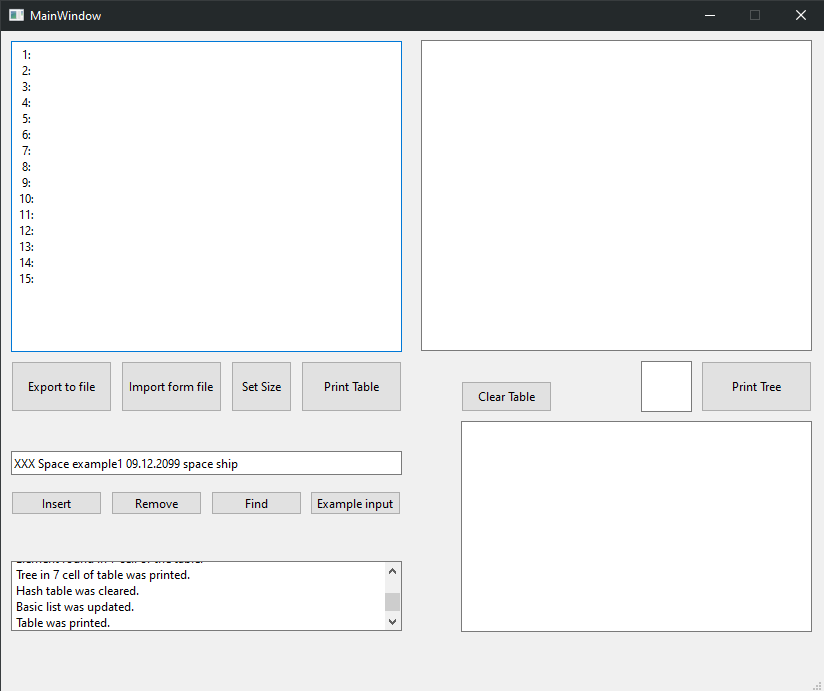


Рисунок 14 – «Пример работы кнопки Print Table»

4.4 Тестирование

Тестирование проводилось методом черного ящика, результаты которого представлены в Таблицах 3, 4, 5.

Таблица 3 – Тестирование работы со справочником «Пункт выдачи товара онлайн магазина»

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Описание тестовой ситуации | | Входные данные | | Выходные данные | |
| Ваш справочник | Ваши поля | Ваш справочник | Сообщение |
| Добавление | | | | | |
| 1 | Добавление некорректных данных (наименование поставщика/артикул/дата) | - | pao Sberbank ds!2DFs 02.03.2100 knife | - | Invalid input data. |
| 2 | Добавление корректных данных в пустой справочник | - | OAO Lukoil bq12bhdlat 03.06.2017 lobster | OAO Lukoil bq12bhdlat 03.06.2017 lobster | Element added successfully! |
| 3 | Добавление корректных данных при коллизии | OOO Rosneft 5mz0zo6ugw 21.12.2090 smartphone | OOO Gurman sa2b63hd1lat 13.26.2017 omar | OOO Rosneft 5mz0zo6ugw 21.12.2090 smartphone and OOO Gurman sa2b63hd1lat 13.26.2017 omar | Element added successfully! |
| 4 | Добавление повторяющегося | OOO Rosneft 5mz0zo6ugw 21.12.2090 smartphone | OOO Rosneft 5mz0zo6ugw 21.12.2090 smartphone | OOO Rosneft 5mz0zo6ugw 21.12.2090 smartphone | Element already exists! |
| Удаление | | | | | |
| 5 | Удаление некорректных данных (наименование поставщика/артикул/дата) | - | pao Sberbank ds!2DFs 02.03.2100 knife | - | Invalid input data. |
| 6 | Удаление корректных данных, когда в справочнике 1 элемент | OAO Lukoil bq12bhdlat 03.06.2017 lobster | OAO Lukoil bq12bhdlat 03.06.2017 lobster | - | Element removed successfully! |
| 7 | Удаление корректных данных при коллизии | OOO Rosneft 5mz0zo6ugw 21.12.2090 smartphone and OOO Gurman sa2b63hd1lat 13.26.2017 omar | OOO Gurman sa2b63hd1lat 13.26.2017 omar | OOO Rosneft 5mz0zo6ugw 21.12.2090 smartphone | Element removed successfully! |
| 8 | Удаление несуществующего | OAO Lukoil bq12bhdlat 03.06.2017 lobster | OOO Rosneft 5mz0zo6ugw 21.12.2090 smartphone | OAO Lukoil bq12bhdlat 03.06.2017 lobster | Element nor removed. |
| Поиск | | | | | |
| 9 | Поиск некорректных данных (наименование поставщика/артикул/дата) | - | pao Sberbank ds!2DFs 02.03.2100 knife | - | Invalid input data. |
| 10 | Поиск корректных данных, когда в справочнике 1 элемент | OAO Lukoil bq12bhdlat 03.06.2017 lobster | OAO Lukoil bq12bhdlat 03.06.2017 lobster | OAO Lukoil bq12bhdlat 03.06.2017 lobster | Element found in 5 cell of the table. |
| 11 | Поиск корректных данных при коллизии | OOO Rosneft 5mz0zo6ugw 21.12.2090 smartphone and OOO Gurman sa2b63hd1lat 13.26.2017 omar | OOO Gurman sa2b63hd1lat 13.26.2017 omar | OOO Rosneft 5mz0zo6ugw 21.12.2090 smartphone and OOO Gurman sa2b63hd1lat 13.26.2017 omar | Element found in 3 cell of the table. |
| 12 | Поиск несуществующего | OAO Lukoil bq12bhdlat 03.06.2017 lobster | OOO Rosneft 5mz0zo6ugw 21.12.2090 smartphone | OAO Lukoil bq12bhdlat 03.06.2017 lobster | Element not found in the table. |

Таблица 4 – Тестирование структуры данных хеш-таблица с методом разрешения коллизии: цепочки, представляющие собой бинарное несбалансированное дерево.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Описание тестовой ситуации | | Входные данные | | Выходные данные | |
| Ваша структура | Ваши поля | Ваша структура | Сообщение |
| Добавление | | | | | |
| 1 | Добавление некорректных данных (наименование поставщика/артикул/дата) | - | pao Sberbank ds!2DFs 02.03.2100 knife | - | Invalid input data. |
| 2 | Добавление корректных данных в пустой справочник | - | OAO Lukoil bq12bhdlat 03.06.2017 lobster | OAO Lukoil bq12bhdlat 03.06.2017 lobster | Element added successfully! |
| 3 | Добавление корректных данных при коллизии | OOO Rosneft 5mz0zo6ugw 21.12.2090 smartphone | OOO Gurman sa2b63hd1lat 13.26.2017 omar | OOO Rosneft 5mz0zo6ugw 21.12.2090 smartphone and OOO Gurman sa2b63hd1lat 13.26.2017 omar | Element added successfully! |
| 4 | Добавление повторяющегося | OOO Rosneft 5mz0zo6ugw 21.12.2090 smartphone | OOO Rosneft 5mz0zo6ugw 21.12.2090 smartphone | OOO Rosneft 5mz0zo6ugw 21.12.2090 smartphone | Element already exists! |
| Удаление | | | | | |
| 5 | Удаление некорректных данных (наименование поставщика/артикул/дата) | - | pao Sberbank ds!2DFs 02.03.2100 knife | - | Invalid input data. |
| 6 | Удаление корректных данных, когда в справочнике 1 элемент | OAO Lukoil bq12bhdlat 03.06.2017 lobster | OAO Lukoil bq12bhdlat 03.06.2017 lobster | - | Element removed successfully! |
| 7 | Удаление корректных данных при коллизии | OOO Rosneft 5mz0zo6ugw 21.12.2090 smartphone and OOO Gurman sa2b63hd1lat 13.26.2017 omar | OOO Gurman sa2b63hd1lat 13.26.2017 omar | OOO Rosneft 5mz0zo6ugw 21.12.2090 smartphone | Element removed successfully! |
| 8 | Удаление несуществующего | OAO Lukoil bq12bhdlat 03.06.2017 lobster | OOO Rosneft 5mz0zo6ugw 21.12.2090 smartphone | OAO Lukoil bq12bhdlat 03.06.2017 lobster | Element nor removed. |
| 5 | Удаление некорректных данных (наименование поставщика/артикул/дата) | - | pao Sberbank ds!2DFs 02.03.2100 knife | - | Invalid input data. |
| Поиск | | | | | |
| 9 | Поиск некорректных данных (наименование поставщика/артикул/дата) | - | pao Sberbank ds!2DFs 02.03.2100 knife | - | Invalid input data. |
| 10 | Поиск корректных данных, когда в справочнике 1 элемент | OAO Lukoil bq12bhdlat 03.06.2017 lobster | OAO Lukoil bq12bhdlat 03.06.2017 lobster | OAO Lukoil bq12bhdlat 03.06.2017 lobster | Element found in 5 cell of the table. |
| 11 | Поиск корректных данных при коллизии | OOO Rosneft 5mz0zo6ugw 21.12.2090 smartphone and OOO Gurman sa2b63hd1lat 13.26.2017 omar | OOO Gurman sa2b63hd1lat 13.26.2017 omar | OOO Rosneft 5mz0zo6ugw 21.12.2090 smartphone and OOO Gurman sa2b63hd1lat 13.26.2017 omar | Element found in 3 cell of the table. |
| 12 | Поиск несуществующего | OAO Lukoil bq12bhdlat 03.06.2017 lobster | OOO Rosneft 5mz0zo6ugw 21.12.2090 smartphone | OAO Lukoil bq12bhdlat 03.06.2017 lobster | Element not found in the table. |

Таблица 5 – Тестирование структуры данных несбалансированное бинарное дерево, которое лежит в ячейке хеш-таблицы. (Используется ХТ с размером 1)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Описание тестовой ситуации | | Входные данные | | Выходные данные | |
| Ваша структура | Ваши поля | Ваша структура | Сообщение |
| Добавление | | | | | |
| 1 | Добавление в пустое дерево | - | P Js art1 11.11.2011 s |  | Element added successfully! |
| 2 | Добавление большего элемента |  | P Js art2 11.11.2011 s |  | Element added successfully! |
| 3 | Добавление меньшего элемента |  | P Js art0 11.11.2011 s |  | Element added successfully! |
| 4 | Добавление внука |  | P Js art4 11.11.2011 s |  | Element added successfully! |
| Удаление | | | | | |
| 5 | Удаление листа |  | P Js art4 11.11.2011 s |  | Element removed successfully! |
| 6 | Удаление, когда 1 потомок |  | P Js art2 11.11.2011 s |  | Element removed successfully! |
| 7 | Удаление, когда 2 потомка |  | P Js art1 11.11.2011 s |  | Element removed successfully! |
| Поиск | | | | | |
| 9 | Поиск элемента |  | P Js art4 11.11.2011 s |  | Element found in 1 cell of the table. |

# Заключение

Целью курсового проекта было: разработка информационной системы для работы со справочником предметной области «Пункт выдачи товара онлайн магазина».

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

1. Проведён анализ предметной области «Пункт выдачи товара онлайн магазина»;
2. Изучены теоретические основы методов построения справочников;
3. Определены требования к информационной системе;
4. Информационная система была реализована и спроектирована;
5. Были получены новые навыки разработки объемных проектов: формирование красивой структуры – менеджмент заголовочных и исходных файлов;
6. Были получены новые навыки в работе со средой разработки Qt Creator.

# Список литературы

1. Кормен Т. Х. Алгоритмы: построение и анализ. / Т. Х. Кормен, Ч. И. Лейзерсон, Р. Л. Ривест, Клиффорд Штайн – 2-е изд. – Москва : Издательский дом «Вильямс», 2005. – 1296 с.
2. Кнут Д. Э. Искусство программирования, Том 3. Сортировка и поиск – 2-е изд. – Москва : Издательский дом «Вильямс», 2007. – 832 с.
3. Вирт Н. Алгоритмы и структуры данных. / Никлаус Вирт – Москва : Издательство «Мир», 1989. – 360 с.
4. Седжвик Р. Фундаментальные алгоритмы на С++. Анализ/Структуры данных/Сортировка/Поиск. / Роберт Седжвик. – Киев : Издательство «ДиаСофт», 2001. – 688 с.
5. Лекции дисциплины «Фундаментальные структуры данных и алгоритмы».

 **Вычисление ASCII-суммы артикула:**

* Берем артикул элемента.
* Преобразуем каждый символ артикула в ASCII.
* Складываем все значения.

 **Определение количества цифр в размере таблицы:**

* Берем размер таблицы.
* Определяем, сколько цифр в числе, обозначающем размер таблицы. Например, для числа 100 это будет 3 цифры.

 **Добавление даты:**

* Записываем день, месяц и год даты элемента.
* Складываем эти значения вместе и прибавляем сумме артикула

 **Преобразование суммы в строку и деление на группы:**

* Делим строку на группы, где каждая группа содержит количество цифр, определенное на шаге 2.
* Преобразуем каждую группу обратно в число и складываем их все вместе.

 **Возведение в квадрат:**

* Возводим полученную сумму в квадрат.

 **Извлечение средней части результата:**

* Находим среднюю часть строки, длина которой равна количеству цифр, определенному на шаге 2.