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

"МИРЭА - Российский технологический университет"

РТУ МИРЭА

Институт информационных технологий (ИИТ)
Кафедра МОиСИТ

ОТЧЕТ
ПО ПРАКТИЧЕСКОЙ РАБОТЕ №6.1

«Быстрый доступ к данным с помощью хеш-таблиц»
по дисциплине
«Структуры и алгоритмы обработки данных»

Выполнил студент группы ИКБО-10-24 Таганов А.А.

Практическую работу выполнил «_»_____2025 г.

«Зачтено» «__»_____2025 г.

Москва 2025

Цель работы: освоить приёмы хеширования и эффективного поиска элементов множества.

Задание: разработать приложение, которое использует хеш-таблицу

Описание алгоритма: Хеш-таблица представляет собой ассоциативный массив пар ключ-значение. Расположение элемента в массиве определяется специально хеш-функцией. Когда хеш-функция выдает одинаковый результат для двух разных записей вызывает вторая хеш-функция, которая определяет собой шаг, с которым после будет происходить перебор по массиву пока не найдется свободное место.

Реализация: Одна запись представляет собой экземпляр структуры UniversitySpecialization (листинг 1). В качестве ключа будет использовать поле code. Две записи считаются одинаковыми, если совпадают их ключи (коды).

Листинг 1 – Структура UniversitySpecialization

```
struct UniversitySpecialization {
    string code;
    string name;

    UniversitySpecialization(string c = "", string n = "") : code(c), name(n)
{}
    bool isEmpty() const { return code.empty(); }
    bool isDeleted() const { return code == "DELETED"; }
};
```

Хеш-таблица реализована через класс HashTable. Функция getCodeValue преобразует текстовое представление ключа в ключ типа unsigned short. Первая хеш-функция принимает ключ и возвращает остаток от деления на размер массива. Вторая хеш-функция делает то же самое, но возвращает остаток от деления на размер массива – 1 и к получившемуся значению прибавляет 1. Функция rehash вызывается при превышении LOAD_FACTOR_THRESHOLD, равного 0.7.

Листинг 2 – private поля класса HashTable

```
class HashTable {
private:
    vector<UniversitySpecialization> table;
    int size;
```

```
    int capacity;

    unsigned short getCodeValue(const string& code) {
        unsigned short value = 0;
        for (char c : code) {
            if (c >= '0' && c <= '9') {
                value = value * 10 + (c - '0');
            }
        }
        return value;
    }

    int hash1(unsigned short code) { return code % capacity; }
    int hash2(unsigned short code) { return code % (capacity - 1) + 1; }

    void rehash() {
        int oldCapacity = capacity;
        capacity *= 2;
        vector<UniversitySpecialization> newTable(capacity);

        for (const auto& item : table) {
            if (!item.isEmpty() && !item.isDeleted()) {
                unsigned short codeValue = getCodeValue(item.code);
                int index = hash1(codeValue);
                int step = hash2(codeValue);

                while (!newTable[index].isEmpty()) {
                    index = (index + step) % capacity;
                }
                newTable[index] = item;
            }
        }
        table = newTable;
        cout << "Table expanded. New size: " << capacity << endl;
    }
```

Конструктор HashTable инициализирует размер таблицы и вызывет функцию autofill для заполнения таблицы начальными записями.

Листинг 3 – Конструктор и функция autofill

```
public:
    HashTable(int cap = INITIAL_SIZE) : size(0), capacity(cap) {
        table.resize(capacity);
        autoFill();
    }

    void autoFill() {
        insert("09.03.01", "MIREA");
        insert("09.03.02", "SPbSU");
        insert("09.03.03", "NSU");
        insert("09.03.04", "MIPT");
        insert("01.03.02", "MIFI");
        insert("38.03.05", "HSE");
        insert("27.03.04", "MSU");
    }
```

Реализован метод insert для вставки в таблицу. Алгоритм его работы следующий. Сначала проверяется не превышает ли вместимость таблицы LOAD_FACTOR_THRESHOLD. При превышении вызывается rehash. Затем

рассчитывается начальный индекс и шаг. Также определяется стартовый индекс для предотвращения циклического обхода. Затем в цикле выполняется обход таблицы и поиск пустой ячейки путем высчитывания нового индекса. Если индекс равен стартовому, то был произведен полный обход, а таблица оказалась полной. В конце мы добавляем ключ и значение и увеличиваем счетчик.

Листинг 4 – Функция search

```
void search(string code) {
        unsigned short codeValue = getCodeValue(code);
        int index = hash1(codeValue);
        int step = hash2(codeValue);
        int startIndex = index;

        while (!table[index].isEmpty()) {
            if (!table[index].isDeleted() && table[index].code == code) {
                cout << "Found: " << code << " - " << table[index].name <<
endl;
                return;
            }
            index = (index + step) % capacity;
            if (index == startIndex) break;
        }
        cout << "Record with code " << code << " not found" << endl;
    }
```

Функция remove работает по аналогичному алгоритму, но если мы нашли пустую ячейку, то прерываем программу, потому что такой записи не может существовать.

Листинг 5 – Функция remove

```
void remove(string code) {
        unsigned short codeValue = getCodeValue(code);
        int index = hash1(codeValue);
        int step = hash2(codeValue);
        int startIndex = index;

        while (!table[index].isEmpty()) {
            if (!table[index].isDeleted() && table[index].code == code) {
                table[index].code = "DELETED";
                table[index].name = "";
                size--;
                cout << "Record with code " << code << " deleted" << endl;
                return;
            }
            index = (index + step) % capacity;
            if (index == startIndex) break;
        }
        cout << "Record with code " << code << " not found" << endl;
    }
```

Также реализована функция display для вывода всей таблицы.

Листинг 6 – Функция display

```
void display() {
        cout << "Hash table (" << size << "/" << capacity << "):" << endl;
        for (int i = 0; i < capacity; i++) {
            cout << "[" << i << "]: ";
            if (table[i].isEmpty()) cout << "Empty";
            else if (table[i].isDeleted()) cout << "Deleted";
            else cout << table[i].code << " - " << table[i].name;
            cout << endl;
        }
    }
```

Реализован текстовый интерфейс. В основной функции создается объект класса HshTable и производится взаимодействие с ним при помощи реализованного интерфейса.

Листинг 7 – Основная функция программы

```
void ui(HashTable& ht) {
    int choice;
    cout << endl;
    cout << "Commands:" << endl;
    cout << "1. Show table" << endl;
    cout << "2. Find record" << endl;
    cout << "3. Add record" << endl;
    cout << "4. Delete record" << endl;
    cout << "5. Exit" << endl;
    cout << "Enter command: ";
    cin >> choice;
    cin.ignore();
    cout << endl;

    switch (choice) {
        case 1:
            ht.display();
            break;
        case 2: {
            string code;
            cout << "Enter specialty code: ";
            getline(cin, code);
            ht.search(code);
            break;
        }
        case 3: {
            string code, name;
            cout << "Enter specialty code: ";
            getline(cin, code);
            cout << "Enter university name: ";
            getline(cin, name);
            ht.insert(code, name);
            break;
        }
        case 4: {
            string code;
            cout << "Enter code to delete: ";
            getline(cin, code);
```

```
            ht.remove(code);
            break;
        }
        case 5:
            exit(0);
        default:
            cout << "Invalid command!" << endl;
    }
}

int main() {
    HashTable ht;
    while (true) {
        ui(ht);
    }
    return 0;
}
```

На следующих рисунках представлен интерфейс для работы с таблицей и тестирование команд.

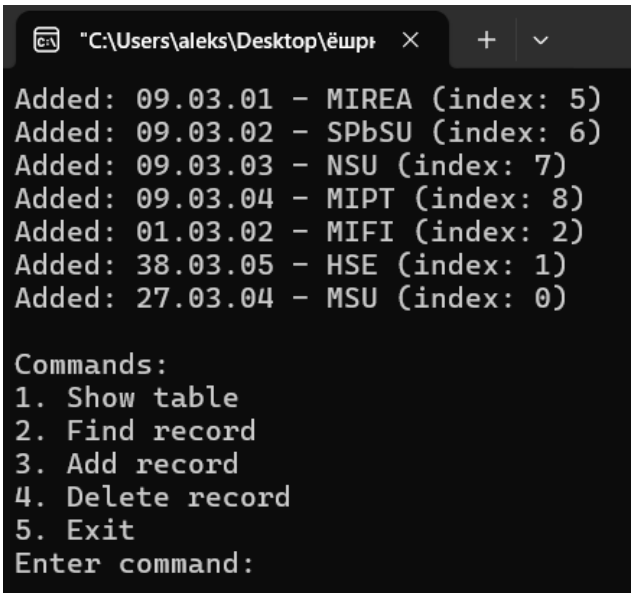

Рисунок 1 – Начальный экран

Команда Find Record просит ввести ключ и выводит найденное поле.

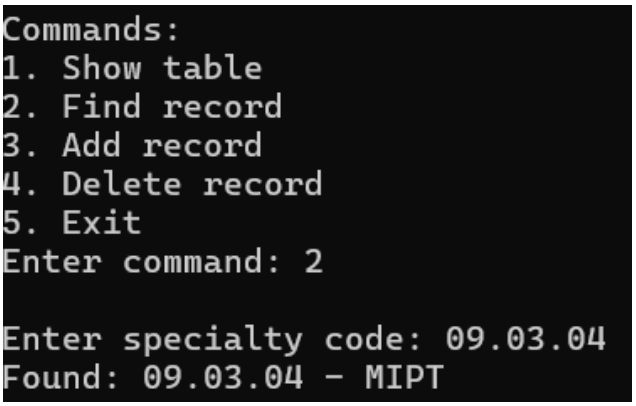

Рисунок 2 – Команда Find Record

Команда Add record добавляет запись в таблицу и выводит ее индекс.

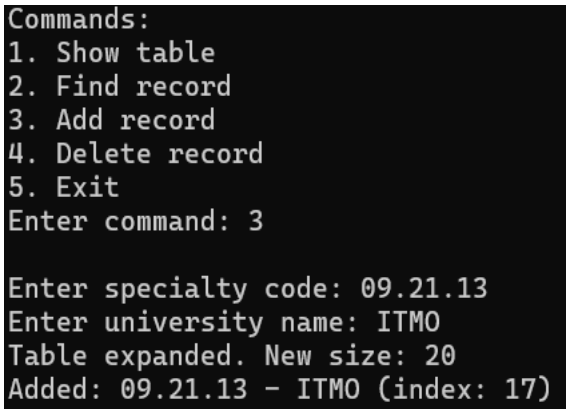

Рисунок 3 – Команда Add record

Команда Delete record удаляет поле таблицы.

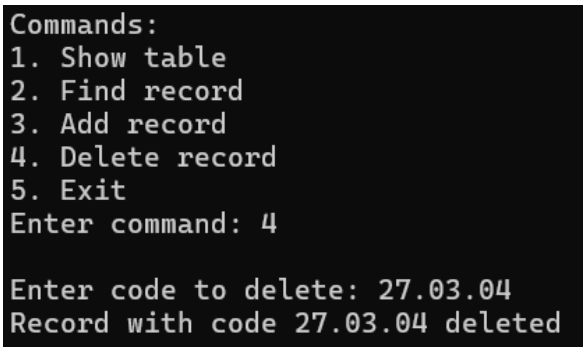

Рисунок 4 – Команда Delete record

Команда Show table выводит таблицу на экран.

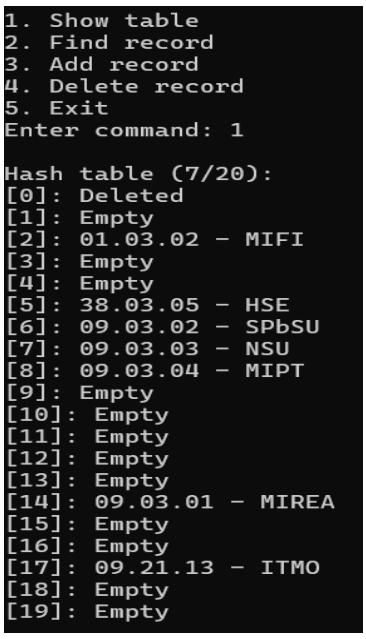

Рисунок 5 – Команда Show table

Вывод

Поставленные задачи выполнены: реализованы функционал хеш-таблицы, методы для взаимодействия с ней, разработана программа для внесения элементов в хеш-таблицу, удаления и их поиска.

Литература.

1. Страуструп Б. Программирование. Принципы и практика с использованием C++. 2-е изд., 2016.

2. Документация по языку C++ [Электронный ресурс]. URL: https://docs.microsoft.com/ruru/cpp/cpp/ (дата обращения 31.09.2021).

3. Курс: Структуры и алгоритмы обработки данных. Часть 2 [Электронный ресурс]. URL:

https://online-edu.mirea.ru/course/view.php?id=4020 (дата обращения 31.09.2021).