МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«МИРЭА – Российский технологический университет»
РТУ МИРЭА

Институт Информационных технологий

Кафедра Математического обеспечения и стандартизации информационных технологий

ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ № 3
по дисциплине
«Структуры и алгоритмы обработки данных»

Тема: «Применение хеш-таблицы для поиска данных в двоичном файле с записями фиксированной длины»

Выполнил студент группы ИКБО-11-22 Гришин А. В.

Принял преподаватель Скворцова Л. А.

Самостоятельная работа выполнена «__»_____202__г. *(подпись студента)*

«Зачтено» «__»_____202__г. *(подпись руководителя)*

Москва 2023

1. Цель работы

Получить навыки по разработке хеш-таблиц и их применении при поиске данных в других структурах данных (файлах).

2. Задание №1

2.1 Постановка задачи

Разработать приложение, которое использует хеш-таблицу для организации прямого доступа к записям двоичного файла, реализованного в практической работе 2.

2.2 Требования:

Создать приложение и включить в него три заголовочных файла: управление хеш-таблицей, управление двоичным файлом (практическая работа 2), управление двоичным файлом посредством хеш-таблицы.

3. Для обеспечения прямого доступа к записи в файле элемент хеш-таблицы должен включать обязательные поля: ключ записи в файле, номер записи с этим ключом в файле. Элемент может содержать другие поля, требующиеся методу (указанному в вашем варианте), разрешающему коллизию.

4. Управление хеш-таблицей

4.1. Определить структуру элемента хеш-таблицы и структуру хеш-таблицы в соответствии с методом разрешения коллизии, указанном в варианте. Определения разместить в соответствующем заголовочном файле. Все операции управления хеш-таблицей размещать в этом заголовочном файле.

4.2. Тестирование операций выполнять в функции main приложения по мере их реализации.

4.3. После тестирования всех операций, создать в заголовочном файле функцию с именем testHeshT переместить в нее содержание функции main, проверить, что приложение выполняется.

4.4. Разработать операции по управлению хеш-таблицей.

4.5. Разработать хеш-функцию (метод определить самостоятельно), выполнить ее тестирование, убедиться, что хеш (индекс элемента таблицы) формируется верно.

4.6. Разработать операции: вставить ключ в таблицу, удалить ключ из таблицы, найти ключ в таблице, рехешировать таблицу. Каждую операцию тестируйте по мере ее реализации.

4.7. Подготовить тесты (последовательность значений ключей), обеспечивающие:

4.7.1. вставку ключа без коллизии

4.7.2. вставку ключа и разрешение коллизии

4.7.3. вставку ключа с последующим рехешированием

4.7.4. удаление ключа из таблицы

4.7.5. поиск ключа в таблице. Для метода с открытым адресом подготовить тест для поиска ключа, который размещен в таблице после удаленного ключа, с одним значением хеша для этих ключей

4.8. Выполнить тестирование операций управления хеш-таблицей. При тестировании операции вставки ключа в таблицу предусмотрите вывод списка индексов, которые формируются при вставке элементов в таблицу.

5. Управление двоичным файлом

5.1. Операции управления двоичным файлом: создание двоичного файла из текстового, добавить запись в двоичный файл, удалить запись с заданным ключом из файла, прочитать запись файла по заданному номеру записи.

5.2. Структура записи двоичного файла и все операции по управлению файлом должны быть размещены в соответствующем заголовочном файле.

5.3. Выполнить тестирование операций в main приложения, и содержание функции main переместить в соответствующую функцию заголовочного файла с именем testBinF.

6. Управление файлом посредством хеш-таблицы

7. В заголовочный файл управления файлом посредством хеш-таблицы подключить заголовочные файлы: управления хеш-таблицей, управления двоичным файлом. Реализовать поочередно все перечисленные ниже операции в этом заголовочном файле, выполняя их тестирование из функции main приложения. После разработки всех операций выполнить их комплексное тестирование.

8. Разработать и реализовать операции

8.1. Прочитать запись из файла и вставить элемент в таблицу (элемент включает:

ключ и номер записи с этим ключом в файле, и для метода с открытой адресацией возможны дополнительные поля).

8.2. Удалить запись из таблицы при заданном значении ключа и соответственно из файла.

8.3. Найти запись в файле по значению ключа (найти ключ в хеш-таблице, получить номер записи с этим ключом в файле, выполнить прямой доступ к записи по ее номеру).

8.4. Подготовить тесты для тестирования приложения:

8.5. Заполните файл небольшим количеством записей.

8.6. Включите в файл записи как не приводящие к коллизиям, так и приводящие.

8.7. Обеспечьте включение в файл такого количества записей, чтобы потребовалось рехеширование.

8.8. Заполните файл большим количеством записей (до 1 000 000).

8.9. Определите время чтения записи с заданным ключом: для первой записи файла, для последней и где-нибудь в середине. Убедитесь (или нет), что время доступа для всех записей одинаково.

9. Составить отчет.

Индивидуальное задание:

№	Тип хеш-таблицы (метод разрешения коллизии)	Структура записи двоичного файла
15	Открытый адрес (смещение на 1)	Дни рождения друзей: дата рождения, имя

2.3 Тестовый пример

Копия содержания текстового файла:

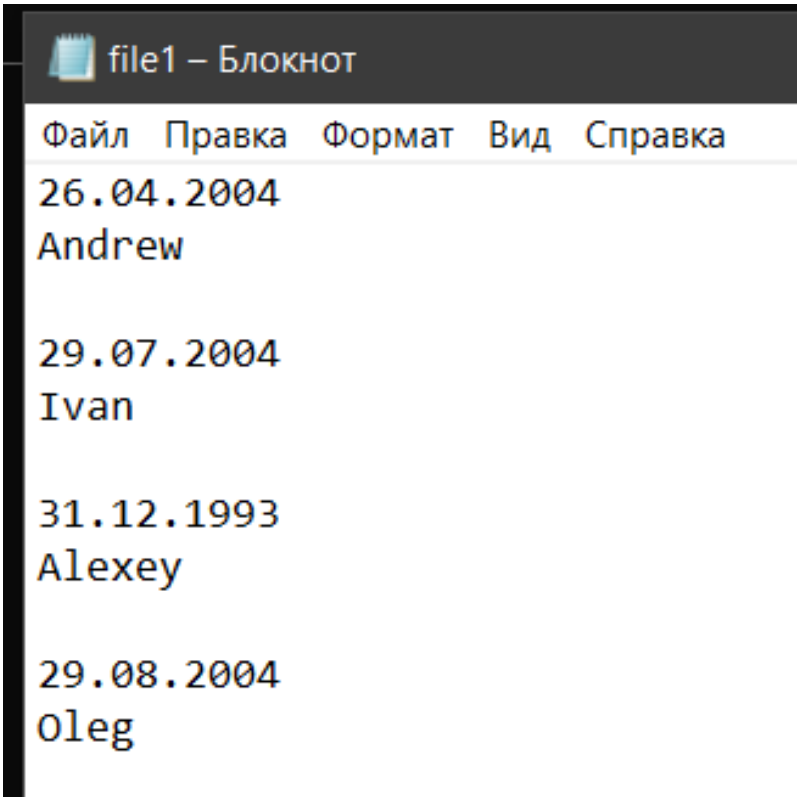

Рисунок 1 – Копия содержания текстового файла

Структура записи двоичного файла:

```
// Определение структуры для хранения данных о друзьях
struct FriendRecord
{
    char dateOfBirth[11];
    char name[50];
};
```

Рисунок 2 – Структура записи двоичного файла

2.4 Реализация функций (управление двоичным файлом)

Приведенные ниже функции относятся к заголовочному файлу управления двоичным файлом.

2.4.1 Функция №1 – txt_to_bin

```
// Функция для конвертации данных из текстового файла в двоичный
void txt_to_bin(ifstream& txt, ofstream& bin)
{
    if (!bin.is_open() || !txt.is_open())
    {
        cout << "Ошибка открытия текстового файла\n";
        return;
    }

    FriendRecord friendRecord;

    // Чтение данных из текстового файла и запись их в двоичный файл
    while (txt.getline(friendRecord.dateOfBirth, sizeof(friendRecord.dateOfBirth)) &&
        txt.getline(friendRecord.name, sizeof(friendRecord.name)))
    {
        bin.write((char*)&friendRecord, sizeof(friendRecord));
        txt.ignore(); // Пропустить символ новой строки
    }
    txt.close();
    bin.close();
    cout << "Перевод прошел успешно\n";
}
```

Рисунок 3 – Функция №1

Функция **txt_to_bin** – функция перевода текстового файла в двоичный. В виде параметров принимает **txt** – объект класса чтения из файла и **bin** – объект класса записи в файл. При неудачном открытии файлов выводит на экран «Ошибка открытия текстового файла», при удачном ничего не выводит.

2.4.2 Функция №2 – find_by_key

```
// Функция для поиска записи по указанному ключу
FriendRecord find_by_key(const char* bin_name, int n)
{
    ifstream bin(bin_name, ios::binary);
    FriendRecord friendRecord;

    // Проверка на успешное открытие файла
    if (!bin)
    {
        cerr << "Не удалось открыть файл для записи";
        return FriendRecord();
    }

    // Поиск и вывод на экран записи по указанному ключу
    size_t friendSize = sizeof(FriendRecord);
    bin.seekg((n - 1) * friendSize, ios::beg);
    bin.read((char*)&friendRecord, sizeof(FriendRecord));
    cout << friendRecord.dateOfBirth << endl << friendRecord.name << endl;

    // Проверка на ошибки вывода
    if (bin.good())
    {
        cout << "Ошибок вывода не обнаружено\n";
    }
    else
    {
        cout << "Обнаружена ошибка вывода\n";
    }
    bin.close();
    return friendRecord;
}
```

Рисунок 4 – Функция №2

Функция **find_by_key** – функция поиска записи по ее порядковому номеру в файле, используя механизм прямого доступа к записи в двоичном файле. В виде параметров принимает **const_char* bin_name** – название файла, **int n** – порядковый номер в файле искомой строки. Выводит на экран данные записи по ключу, при удачном выводе «Ошибок вывода не обнаружено», при неудачном выводе «Обнаружена ошибка вывода». Возвращает найденную по порядковому номеру структуру.

2.4.3 Функция №3 – delete_by_key

```
// Функция для удаления записи по указанному ключу
void delete_by_key(const char* bin_name, int n) {
    fstream bin(bin_name, ios::binary | ios::in);
    // Проверка на успешное открытие файла
    if (!bin.is_open())
    {
        cerr << "Ошибка открытия файла\n";
        return;
    }
    fstream temp("temp_file.dat", ios::binary | ios::out);
    // Проверка на успешное открытие временного файла
    if (!temp.is_open())
    {
        cerr << "Ошибка открытия временного файла\n";
        bin.close();
        return;
    }
    FriendRecord friendRecord;
    int key = 1;
    // Копирование данных из исходного файла во временный, исключая запись с указанным ключом
    while (bin.read((char*)&friendRecord, sizeof(FriendRecord)))
    {
        if (key != n)
        {
            temp.write((char*)&friendRecord, sizeof(FriendRecord));
        }
        key++;
    }
    bin.close();
    temp.close();
    // Удаление исходного файла и переименование временного файла
    remove(bin_name);
    rename("temp_file.dat", bin_name);
}
```

Рисунок 5 – Функция №3

Функция **delete_by_key** – функция для удаления записи с заданным значением ключа. В виде параметров получает **const_char* bin_name** – название файла, **int n** – строку для удаления. При неудачном открытии файла выводит на экран «Ошибка открытия текстового файла» или при «Ошибка открытия временного файла», при удачном ничего не выводит.

2.5 Реализация функций (управление хеш-таблицей)

Приведенные ниже функции относятся к заголовочному файлу управления хеш-таблицей.

```
// Определение структуры HashTableBucket для хранения элемента хеш-таблицы
struct HashTableBucket {
    Tkey key; // Ключ элемента
    int numberRecord; // Номер записи
    bool Popen = false; // Флаг доступности элемента
};
```

Рисунок 6 – Структура элемента хеш-таблицы

Структура **HashTableBucket** является структурой элемента хеш-таблицы.

```
// Определение структуры HashTable для реализации хеш-таблицы
struct HashTable {
private:
    HashTableBucket* table; // Указатель на массив элементов хеш-таблицы
    int size; // Размер таблицы
```

Рисунок 7 – структура хеш-таблицы

Структура **HashTable** является структурой хеш-таблицы.

2.5.1 Функция №4 – findIndex

Хеш-функция **findIndex** используется для поиска индекса в хеш-таблице.

```
// Функция для поиска индекса элемента по ключу
int findIndex(int key) {
    return key % size;
}
```

Рисунок 8 – Функция 4

2.5.2 Функция №5 – insertInHashTable

```
// Функция для вставки элемента в хеш-таблицу
void insertInHashTable(const int& key, int numberRecord) {
    if (loadFactor() > 0.7) {
        rehashTable();
    }

    int index = findIndex(key);
    while (table[index].Popen) {
        index = (index + 1) / size;
    }
    table[index].key = key;
    table[index].numberRecord = numberRecord;
    table[index].Popen = true;
}
```

Рисунок 9 – Функция №5

Функция **insertInHashTable** – функция для вставки элемента в хеш-таблицу. В виде параметров получает **const int& key** – ключ элемента, **int numberRecord** –порядковый номер элемента в двоичном файле. Ничего не возвращает. При заполнении таблицы более чем на 70% вызывает функцию рехеширования.

2.5.3 Функция №6 - deleteKey

```
// Функция для удаления элемента по ключу из таблицы
void deleteKey(const int& key) {
    int index = findIndex(key);
    while (table[index].Popen) {
        if (table[index].key == key) {
            table[index].Popen = false;
            return;
        }
        index = (index + 1) / size;
    }
    cout << "Key not found" << endl;
}
```

Рисунок 10 – Функция №6

Функция **deleteKey** удаляет объект их хеш-таблицы. В виде параметров получает **const int& key** – ключ элемента. При удачном удалении ничего не возвращает, при неудачном – «Key not found».

2.5.4 Функции №7 – findKey

```
// Функция для поиска элемента по ключу в таблице
HashTableBucket findKey(const int& key) {
    int index = findIndex(key);
    while (table[index].Popen) {
        if (table[index].key == key) {
            return table[index];
        }
        index = (index + 1) / size;
    }
    cout << "Key not found" << endl;
    return HashTableBucket();
}
```

Рисунок 11 – Функция №7

Функция **findKey** – функция поиска элемента в хеш-таблице по ключу. В виде параметров принимает **const int& key** – ключ элемента. При удачном нахождении возвращает этот элемент, при неудачном – «Key not found».

```
Ваш выбор: 1
Введите номер записи
1
Содержание записи:
26.04.2004
Andrew
Ошибок вывода не обнаружено
Ваш выбор: 1
Введите номер записи
2
Содержание записи:
29.07.2004
Ivan
Ошибок вывода не обнаружено
Ваш выбор: 1
Введите номер записи
3
Содержание записи:
31.12.1993
Alexey
Ошибок вывода не обнаружено
Ваш выбор: 4
Bucket 0 : (Key: 0 , recordNumber: 2)
----------------пусто----------------
----------------пусто----------------
----------------пусто----------------
Bucket 4 : (Key: 4 , recordNumber: 1)
Bucket 5 : (Key: 5 , recordNumber: 3)
----------------пусто----------------
----------------пусто----------------
----------------пусто----------------
----------------пусто----------------
Ваш выбор: 2
Введите значение ключа
29.07.2004
----------------пусто----------------
----------------пусто----------------
----------------пусто----------------
----------------пусто----------------
Bucket 4 : (Key: 4 , recordNumber: 1)
Bucket 5 : (Key: 5 , recordNumber: 3)
----------------пусто----------------
----------------пусто----------------
----------------пусто----------------
----------------пусто----------------
Ваш выбор: 3
Введите значение ключа
31.12.1993
31.12.1993
Alexey
Ошибок вывода не обнаружено
```

2.5.5 Функции №8 – rehashTable

```
// Функция для перехеширования таблицы
void rehashTable() {
    int new_size = size * 2;
    HashTableBucket* newTable = new HashTableBucket[new_size];
    for (int i = 0; i < size; i++) {
        if (table[i].Popen) {
            int key = table[i].key;
            int recordNumber = table[i].numberRecord;
            int newIndex = findIndex(key) % new_size;
            while (newTable[newIndex].Popen) {
                newIndex = (newIndex + 1) % new_size;
            }
            newTable[newIndex].key = key;
            newTable[newIndex].numberRecord = recordNumber;
            newTable[newIndex].Popen = true;
        }
    }
    delete[] table;
    table = newTable;
    size = new_size;
}
```

Рисунок 12 – Функция №8

Функция **rehashTable** – функция рехеширования таблицы, т.е. увеличения её размера в два раза. Ничего не возвращает.

2.6 Реализация функций (Управление файлом посредством хеш-таблицы)

Приведенные ниже функции относятся к заголовочному файлу управления двоичным файлом посредством хеш-таблицы.

2.6.1 Функция №9 - fromFileToHashTab

```
// Функция для переноса данных из файла в хеш-таблицу
void fromFileToHashTab(HashTable& hashtab, int n, const char* bin_name)
{
    cout << "Содержание записи: \n";

    // Вычисление хеша по дате рождения и добавление записи в хеш-таблицу
    int key = hashtab.hashFunc(find_by_key(bin_name, n).dateOfBirth);
    hashtab.insertInHashTable(key, n);
}
```

Рисунок 13 – Функция №9

Функция **fromFileToHashTab** – функция прочтения записи из файла и вставки его в хеш-таблицу. В качестве параметров принимает **HashTable& hashtab** – ссылка на хеш-таблицу, **int n** – порядковый номер элемента в файле, **const char* bin_name** – название двоичного файла. Ничего не возвращает.

```
Перевод прошел успешно
1. Прочитать запись из файла и вставить элемент в таблицу
2. Удалить запись из таблицы при заданном значении ключа и соответственно из файла
3. Найти запись в файле по значению ключа
4. Вывести хеш-таблицу на экран
0. Выход
Ваш выбор: 1
Введите номер записи
1
Содержание записи:
26.04.2004
Andrew
Ошибок вывода не обнаружено
Ваш выбор: 1
Введите номер записи
2
Содержание записи:
29.07.2004
Ivan
Ошибок вывода не обнаружено
Ваш выбор: 1
Введите номер записи
3
Содержание записи:
31.12.1993
Alexey
Ошибок вывода не обнаружено
Ваш выбор: 4
Bucket 0 : (Key: 0 , recordNumber: 2)
----------------пусто----------------
----------------пусто----------------
----------------пусто----------------
Bucket 4 : (Key: 4 , recordNumber: 1)
Bucket 5 : (Key: 5 , recordNumber: 3)
----------------пусто----------------
----------------пусто----------------
----------------пусто----------------
----------------пусто----------------
```

Рисунок 14 – Тест функции №9

2.6.2 Функция №10 – deleteFromTabnFile

```
// Функция для удаления данных из хеш-таблицы и файла по ключу
void deleteFromTabnFile(HashTable hashtab, int key, const char* bin_name)
{
    // Поиск номера записи по ключу в хеш-таблице
    int n = hashtab.findKey(key).key;

    // Удаление ключа из хеш-таблицы и соответствующей записи из файла
    hashtab.deleteKey(key);
    delete_by_key(bin_name, n);
    hashtab.printHashTable();
}
```

Рисунок 15 – Функция №10

Функция **deleteFromTabnFile** – функция удаления записи из таблицы при заданном значении ключа и соответственно из файла. В качестве параметров принимает **HashTable& hashtab** – ссылка на хеш-таблицу, **int key** – ключ элемента, **const char* bin_name** – название двоичного файла. Ничего не возвращает.

```
Перевод прошел успешно
1. Прочитать запись из файла и вставить элемент в таблицу
2. Удалить запись из таблицы при заданном значении ключа и соответственно из файла
3. Найти запись в файле по значению ключа
4. Вывести хеш-таблицу на экран
0. Выход
Ваш выбор: 1
Введите номер записи
1
Содержание записи:
26.04.2004
Andrew
Ошибок вывода не обнаружено
Ваш выбор: 1
Введите номер записи
2
Содержание записи:
29.07.2004
Ivan
Ошибок вывода не обнаружено
Ваш выбор: 2
Введите значение ключа
29.07.2004
----------------пусто----------------
----------------пусто----------------
----------------пусто----------------
----------------пусто----------------
Bucket 4 : (Key: 4 , recordNumber: 1)
----------------пусто----------------
----------------пусто----------------
----------------пусто----------------
----------------пусто----------------
----------------пусто----------------
```

Рисунок 16 – Тест функции №10

2.6.3 Функция №11 – findRecordInFile

```
// Функция для поиска записи в файле по ключу из хеш-таблицы
void findRecordInFile(HashTable hashtab, int key, const char* bin_name)
{
    ifstream file(bin_name, ios::binary);

    // Поиск номера записи в файле по ключу из хеш-таблицы
    int n = hashtab.findKey(key).numberRecord;
    if (n < 1)
        return; // Если запись не найдена, выход из функции

    // Поиск и вывод записи из файла по номеру записи
    find_by_key(bin_name, n);
}
```

Рисунок 17 – Функция №11

Функция **findRecordInFile** – функция нахождения записи в файле по значению ключа из хеш-таблицы. В качестве параметров принимает **HashTable& hashtab** – ссылка на хеш-таблицу, **int key** – ключ элемента, **const char* bin_name** – название двоичного файла. Ничего не возвращает.

```
Перевод прошел успешно
1. Прочитать запись из файла и вставить элемент в таблицу
2. Удалить запись из таблицы при заданном значении ключа и соответственно из файла
3. Найти запись в файле по значению ключа
4. Вывести хеш-таблицу на экран
0. Выход
Ваш выбор: 1
Введите номер записи
1
Содержание записи:
26.04.2004
Andrew
Ошибок вывода не обнаружено
Ваш выбор: 4
----------------пусто----------------
----------------пусто----------------
----------------пусто----------------
----------------пусто----------------
Bucket 4 : (Key: 4 , recordNumber: 1)
----------------пусто----------------
----------------пусто----------------
----------------пусто----------------
----------------пусто----------------
----------------пусто----------------
Ваш выбор: 3
Введите значение ключа
26.04.2004
26.04.2004
Andrew
Ошибок вывода не обнаружено
```

Рисунок 18 – Тест функции №11

2.6.4 Функция Main

```
int main()
{
    setlocale(LC_ALL, "RUS");
    locale::global(locale("en_US.UTF-8"));
    const char* txt_name = "file1.txt";
    const char* bin_name = "bin.dat";
    ifstream txt(txt_name, ios::out);
    ofstream bin(bin_name, ios::binary | ios::out | ios::trunc);
    HashTable hashtab(10);
    if (!txt)
    {
        cout << "Ошибка при открытии текстового файла\n";
        return -1;
    }
    else if (!bin)
    {
        cout << "Ошибка при открытии бинарного файла\n";
        return -1;
    }
    txt_to_bin(txt, bin);
    int choose = 1;
    cout << "1. Прочитать запись из файла и вставить элемент в таблицу\n"
         << "2. Удалить запись из таблицы при заданном значении ключа и соответственно из файла\n"
         << "3. Найти запись в файле по значению ключа\n"
         << "4. Вывести хеш-таблицу на экран\n"
         << "0. Выход\n";
    string key;
    while (choose != 0)
    {
        cout << "Ваш выбор: ";
        cin >> choose;
        switch (choose)
        {
            case 1:
                cout << "Введите номер записи\n";
                int n;
                cin >> n;
                fromFileToHashTab(hashtab, n, bin_name);
                break;
            case 2:
                cout << "Введите значение ключа\n";
                cin >> key;
                deleteFromTabnFile(hashtab, hashtab.hashFunc(key), bin_name);
                break;
            case 3:
            {
                cout << "Введите значение ключа\n";
                cin >> key;
                findRecordInFile(hashtab, hashtab.hashFunc(key), bin_name);
                break;
            }
            case 4:
                hashtab.printHashTable();
                break;
            case 0:
                return 0;
        }
    }
}
```

Рисунок 19 – Функция Main

3. Код программы

3.1 Код модуля HashTable.h

```
#ifndef _3_HASHTABLE_H
#define _3_HASHTABLE_H

#include <iostream>
#include <utility>
#include <fstream>
#include <string>
#include <vector>

typedef int Tkey;
using namespace std;

// Определение структуры HashTableBucket для хранения элемента хеш-таблицы
struct HashTableBucket {
    Tkey key; // Ключ элемента
    int numberRecord; // Номер записи
    bool Popen = false; // Флаг доступности элемента
};

// Определение структуры HashTable для реализации хеш-таблицы
struct HashTable {
private:
    HashTableBucket* table; // Указатель на массив элементов хеш-таблицы
    int size; // Размер таблицы
public:
    // Конструктор для инициализации таблицы заданного размера
    HashTable(int size) {
        this->size = size;
        table = new HashTableBucket[size];
    }

    // Функция для вычисления хеша по строковому ключу
    int hashFunc(const string& key) {
        int sum = 0;
        for (char c : key) { // Перебор символов ключа
            sum += c; // Подсчёт суммы ASCII кодов символов
        }
        return sum % size; // Возвращение остатка от деления суммы на размер таблицы
    }

    // Функция для поиска индекса элемента по ключу
    int findIndex(int key) {
        return key % size;
    }

    // Функция для вычисления коэффициента заполнения таблицы
    float loadFactor() const {
        int count = 0;
        for (int i = 0; i < size; i++) {
            if (table[i].Popen) {
                count++;
            }
        }
        return static_cast<float>(count) / size;
    }

    // Функция для вставки элемента в хеш-таблицу
    void insertInHashTable(const int& key, int numberRecord) {
        if (loadFactor() > 0.7) {
            rehashTable();
        }

        int index = findIndex(key);
```

```
        while (table[index].Popen) {
            index = (index + 1) / size;
        }
        table[index].key = key;
        table[index].numberRecord = numberRecord;
        table[index].Popen = true;
    }

    // Функция для поиска элемента по ключу в таблице
    HashTableBucket findKey(const int& key) {
        int index = findIndex(key);
        while (table[index].Popen) {
            if (table[index].key == key) {
                return table[index];
            }
            index = (index + 1) / size;
        }
        cout << "Key not found" << endl;
        return HashTableBucket();
    }

    // Функция для удаления элемента по ключу из таблицы
    void deleteKey(const int& key) {
        int index = findIndex(key);
        while (table[index].Popen) {
            if (table[index].key == key) {
                table[index].Popen = false;
                return;
            }
            index = (index + 1) / size;
        }
        cout << "Key not found" << endl;
    }

    // Функция для вывода содержимого хеш-таблицы
    void printHashTable() {
        for (int i = 0; i < size; i++) {
            if (table[i].Popen) {
                cout << "Bucket " << i << " : (Key: " << table[i].key << " ,
recordNumber: " << table[i].numberRecord << ")" << endl;
            }
            else {
                cout << "----------------пусто----------------" << endl;
            }
        }
    }

    // Функция для перехеширования таблицы
    void rehashTable() {
        int new_size = size * 2;
        HashTableBucket* newTable = new HashTableBucket[new_size];
        for (int i = 0; i < size; i++) {
            if (table[i].Popen) {
                int key = table[i].key;
                int recordNumber = table[i].numberRecord;
                int newIndex = findIndex(key) % new_size;
                while (newTable[newIndex].Popen) {
                    newIndex = (newIndex + 1) % new_size;
                }
                newTable[newIndex].key = key;
                newTable[newIndex].numberRecord = recordNumber;
                newTable[newIndex].Popen = true;
            }
        }
        delete[] table;
        table = newTable;
        size = new_size;
    }
};
```

```
#endif //_3_HASHTABLE_H
```

3.2 Код модуля BinaryFile.h

```
#ifndef _3_BINARYFILE_H
#define _3_BINARYFILE_H

#include <iostream>
#include <fstream>
#include <string>
using namespace std;

// Определение структуры для хранения данных о друзьях
struct FriendRecord
{
    char dateOfBirth[11];
    char name[50];
};

// Функция для конвертации данных из текстового файла в двоичный
void txt_to_bin(ifstream& txt, ofstream& bin)
{
    if (!bin.is_open() || !txt.is_open())
    {
        cout << "Ошибка открытия текстового файла\n";
        return;
    }

    FriendRecord friendRecord;

    // Чтение данных из текстового файла и запись их в двоичный файл
    while (txt.getline(friendRecord.dateOfBirth, sizeof(friendRecord.dateOfBirth)) &&
        txt.getline(friendRecord.name, sizeof(friendRecord.name)))
    {
        bin.write((char*)&friendRecord, sizeof(friendRecord));
        txt.ignore(); // Пропустить символ новой строки
    }
    txt.close();
    bin.close();
    cout << "Перевод прошел успешно\n";
}

// Функция для конвертации данных из двоичного файла в текстовый
void bin_to_txt(ifstream& binn, ofstream& txtt)
{
    FriendRecord friendRecord;

    if (!txtt.is_open())
    {
        cout << "Ошибка открытия текстового файла\n";
        return;
    }

    // Чтение данных из двоичного файла и запись их в текстовый файл
    while (binn.read((char*)&friendRecord, sizeof(FriendRecord)))
    {
        txtt << friendRecord.dateOfBirth << endl << friendRecord.name << endl;
    }

    // Проверка на ошибки чтения из двоичного файла
    if (!binn.eof() && binn.fail())
    {
        cout << "Ошибка чтения из двоичного файла" << endl;
    }
    cout << "Перевод прошел успешно\n";
}

// Функция для вывода данных из двоичного файла на экран
```

```
void print_from_bin(ifstream& bin)
{
    FriendRecord friendRecord;
    bin.read((char*)&friendRecord, sizeof(FriendRecord));

    // Вывод данных из двоичного файла на экран
    while (!bin.eof())
    {
        cout << friendRecord.dateOfBirth << endl << friendRecord.name << endl;
        bin.read((char*)&friendRecord, sizeof(FriendRecord));
    }
    bin.close();
}

// Функция для удаления записи по указанному ключу
void delete_by_key(const char* bin_name, int n)
{
    fstream bin(bin_name, ios::binary | ios::in);

    // Проверка на успешное открытие файла
    if (!bin.is_open())
    {
        cerr << "Ошибка открытия файла\n";
        return;
    }

    fstream temp("temp_file.dat", ios::binary | ios::out);

    // Проверка на успешное открытие временного файла
    if (!temp.is_open())
    {
        cerr << "Ошибка открытия временного файла\n";
        bin.close();
        return;
    }

    FriendRecord friendRecord;
    int key = 1;

    // Копирование данных из исходного файла во временный, исключая запись с указанным
ключом
    while (bin.read((char*)&friendRecord, sizeof(FriendRecord)))
    {
        if (key != n)
        {
            temp.write((char*)&friendRecord, sizeof(FriendRecord));
        }
        key++;
    }
    bin.close();
    temp.close();

    // Удаление исходного файла и переименование временного файла
    remove(bin_name);
    rename("temp_file.dat", bin_name);
}

// Функция для поиска записи по указанному ключу
FriendRecord find_by_key(const char* bin_name, int n)
{
    ifstream bin(bin_name, ios::binary);
    FriendRecord friendRecord;

    // Проверка на успешное открытие файла
    if (!bin)
    {
        cerr << "Не удалось открыть файл для записи";
        return FriendRecord();
    }
```

```
    // Поиск и вывод на экран записи по указанному ключу
    size_t friendSize = sizeof(FriendRecord);
    bin.seekg((n - 1) * friendSize, ios::beg);
    bin.read((char*)&friendRecord, sizeof(FriendRecord));
    cout << friendRecord.dateOfBirth << endl << friendRecord.name << endl;

    // Проверка на ошибки вывода
    if (bin.good())
    {
        cout << "Ошибок вывода не обнаружено\n";
    }
    else
    {
        cout << "Обнаружена ошибка вывода\n";
    }
    bin.close();
    return friendRecord;
}

#endif //_3_BINARYFILE_H
```

3.3 Код модуля FileByHash.h

```
#ifndef _3_FILEBYHASH_H
#define _3_FILEBYHASH_H

#include "BinaryFile.h"
#include "HashTable.h"

// Функция для переноса данных из файла в хеш-таблицу
void fromFileToHashTab(HashTable& hashtab, int n, const char* bin_name)
{
    cout << "Содержание записи: \n";

    // Вычисление хеша по дате рождения и добавление записи в хеш-таблицу
    int key = hashtab.hashFunc(find_by_key(bin_name, n).dateOfBirth);
    hashtab.insertInHashTable(key, n);
}

// Функция для удаления данных из хеш-таблицы и файла по ключу
void deleteFromTabnFile(HashTable hashtab, int key, const char* bin_name)
{
    // Поиск номера записи по ключу в хеш-таблице
    int n = hashtab.findKey(key).key;

    // Удаление ключа из хеш-таблицы и соответствующей записи из файла
    hashtab.deleteKey(key);
    delete_by_key(bin_name, n);
    hashtab.printHashTable();
}

// Функция для поиска записи в файле по ключу из хеш-таблицы
void findRecordInFile(HashTable hashtab, int key, const char* bin_name)
{
    ifstream file(bin_name, ios::binary);

    // Поиск номера записи в файле по ключу из хеш-таблицы
    int n = hashtab.findKey(key).numberRecord;
    if (n < 1)
        return; // Если запись не найдена, выход из функции

    // Поиск и вывод записи из файла по номеру записи
    find_by_key(bin_name, n);
}

#endif //_3_FILEBYHASH_H
```

4. Вывод

Получили навыки по разработке хеш-таблиц и их применении при поиске данных в других структурах данных (файлах).