МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное учреждение высшего образования
"МИРЭА - Российский технологический университет"

РТУ МИРЭА

Институт информационных технологий (ИТ)
Кафедра математического обеспечения и стандартизации информационных технологий (МОСИТ)

ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ №3

по дисциплине

«Структуры и алгоритмы обработки данных»

Тема. Разработка и программная реализация задач поиска данных в таблицах с применением механизма хеширования.

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

Принял преподаватель Скворцова Л.А.

Москва 2023

СОДЕРЖАНИЕ

Москва 2023

1 Задание 1

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

Требования к выполнению

1. Создать приложение и включить в него три заголовочных файла: управление хеш-таблицей, управление двоичным файлом (практическая работа 2), управление двоичным файлом посредством хеш-таблицы. Имена заголовочным файлам определите сами. Подключите заголовочные файлы к приложению.

2. 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.

3. Управление хеш-таблицей.

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.

2) Тестирование операций выполнять в функции main приложения по мере их реализации. После тестирования всех операций, создать в заголовочном файле функцию с именем testHeshT переместить в нее содержание функции main, проверить, что приложение выполняется. Разработать операции по управлению хеш-таблицей.

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

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

5) Подготовить тесты (последовательность значений ключей), обеспечивающие:

– вставку ключа без коллизии
– вставку ключа и разрешение коллизии
– вставку ключа с последующим рехешированием
– удаление ключа из таблицы
– поиск ключа в таблице

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

6) Выполнить тестирование операций управления хеш-таблицей. При тестировании операции вставки ключа в таблицу предусмотрите вывод списка индексов, которые формируются привставке элементов в таблицу.

4. Управление двоичным файлом. 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.

Примечание. Эти операции должны быть отлажены в практической работе 2, или уже в этой работе, если их пока нету. Структура записи двоичного файла и все операции, по управлению файлом, должны быть размещены в соответствующем заголовочном файле. Выполнить тестирование операций в main приложения, и содержание функции main переместить в соответствующую функцию заголовочного файла с именем testBinF.

5. Управление файлом посредством хеш-таблицы. 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. Разработать и реализовать операции.

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).

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

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)..

4) Подготовить тесты для тестирования приложения:

Заполните файл небольшим количеством записей.

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

– Обеспечьте включение в файл такого количества записей, чтобы потребовалось рехеширование. *Заполните файл большим количеством записей (до 1 000 000).*

1.1 Код программы

Данная программа на языке C++ определяет структуру FriendBirthday, которая представляет собой объект, содержащий информацию о дне рождения друга. Структура содержит четыре поля: dateOfBirth (строковое поле для хранения даты рождения), name (строковое поле для хранения имени друга), openKey (логическое поле, которое по умолчанию установлено в true) и isDeleted (логическое поле, которое по умолчанию установлено в false).

В структуре определены два конструктора. Первый - это конструктор по умолчанию, который инициализирует поля openKey и isDeleted значениями true и false соответственно. Второй конструктор принимает два параметра: dateOfBirth и

name, и инициализирует соответствующие поля этими значениями. Поля openKey и isDeleted инициализируются также, как и в конструкторе по умолчанию.

```
// Определение структуры FriendBirthday
struct FriendBirthday {
    string dateOfBirth;
    string name;
    bool openKey;
    bool isDeleted;

    FriendBirthday()
    {
        openKey = true;
        isDeleted = false;
    }

    FriendBirthday(string dateOfBirth, string name)
    {
        this->dateOfBirth = dateOfBirth;
        this->name = name;
        openKey = true;
        isDeleted = false;
    }
};
```

Рисунок 1 – Структура FriendBirthday

Данная программа на языке C++ определяет структуру HashTable, которая представляет собой хеш-таблицу. Структура содержит два целочисленных поля: M и N, которые инициализируются в конструкторе.

Внутри структуры HashTable определена структура KeyValuePair, которая содержит строковое поле key, объект value типа FriendBirthday и целочисленное поле numberRecord.

Также в структуре HashTable есть поле KeyValues, которое является указателем на массив объектов типа KeyValuePair. В конструкторе HashTable этот массив инициализируется новым массивом KeyValuePair размером M, и все ключи в этом массиве устанавливаются в пустую строку.

```
// Определение структуры HashTable
struct HashTable {
    int M;
    int N;
    struct KeyValuePair {
        string key;
        FriendBirthday value;
        int numberRecord;
    };
    KeyValuePair* KeyValues;

    HashTable(int M, int N)
    {
        this->M = M;
        this->N = N;
        this->KeyValues = new KeyValuePair[M];
        for (int i = 0; i < M; i++) {
            this->KeyValues[i].key = "";
        }
    }
};
```

Рисунок 2 – Структура HashTable

Эта функция insertInHashTable предназначена для вставки элемента в хеш-таблицу. Вот что делает каждый шаг:

1. Функция принимает дату рождения и имя в качестве ключа и значения, а также индекс и ссылку на хеш-таблицу.
2. Ключ хешируется с помощью функции hashFunc, которая возвращает индекс в массиве KeyValues хеш-таблицы.
3. Затем функция в цикле проверяет, свободна ли ячейка с этим индексом. Если ячейка свободна (то есть ключ пуст), то в неё записываются ключ, значение и номер записи.
4. Если ячейка занята, то индекс увеличивается на 1 (с переходом в начало массива при достижении его конца), и процесс повторяется.
5. Если в процессе обхода функция вернулась к исходному индексу, это означает, что таблица полностью заполнена, и выводится соответствующее сообщение.
6. После каждой успешной вставки функция проверяет, не превышает ли текущий коэффициент заполнения (число элементов, делённое на размер таблицы) значение 0.7. Если превышает, то происходит перехеширование с помощью функции rehashTable.

Таким образом, эта функция реализует метод открытой адресации (линейное пробирование) для разрешения коллизий в хеш-таблице. При этом используется динамическое изменение размера таблицы для поддержания эффективности операций.

```
void insertInHashTable(const string& dateOfBirth, const string& name, int i, HashTable& hashTable)
{
    string key = dateOfBirth;
    int id = hashFunc(key, hashTable.M);
    int originalId = id;

    do {
        if (hashTable.KeyValues[id].key == "") {
            hashTable.KeyValues[id].key = key;
            hashTable.KeyValues[id].numberRecord = i;
            FriendBirthday friendInfo(dateOfBirth, name);
            hashTable.KeyValues[id].value = friendInfo;
            hashTable.N++;
            if (static_cast<double>(hashTable.N) / hashTable.M > 0.7) {
                rehashTable(hashTable);
            }
            return;
        }
        id = (id + 1) % hashTable.M;
    } while (id != originalId);
    cout << "Хеш-таблица заполнена, вставка невозможна." << endl;
}
```

Рисунок 3 – Функция для вставки в хэш-таблицу

Функция deleteInHashTable предназначена для удаления элемента из хеш-таблицы. Вот что делает каждый шаг:

1. Функция принимает дату рождения в качестве ключа и ссылку на хеш-таблицу.
2. Ключ хешируется с помощью функции hashFunc, которая возвращает индекс в массиве KeyValues хеш-таблицы.
3. Затем функция в цикле ищет ячейку с этим ключом, увеличивая индекс на 1 (с переходом в начало массива при достижении его конца), пока не найдет нужный ключ.
4. Когда нужный ключ найден, он удаляется (то есть в ячейке устанавливается пустой ключ), уменьшается число элементов в таблице, и в значение записывается информация о том, что элемент удален.

Таким образом, эта функция реализует удаление элемента из хеш-таблицы с открытой адресацией. При этом удаленные элементы помечаются специальным образом, чтобы при поиске и вставке учитывать, что эти ячейки теперь свободны. Это позволяет избежать проблем, связанных с удалением элементов в таких таблицах.

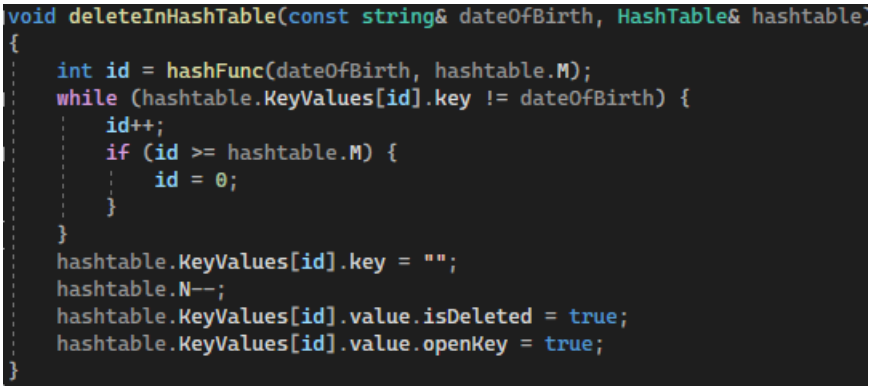


Рисунок 4 – Функция для удаления элемента из хеш-таблицы

Функция findInHashTable предназначена для поиска элемента в хеш-таблице. Вот что делает каждый шаг:

1. Функция принимает дату рождения в качестве ключа и ссылку на хеш-таблицу.
2. Ключ хешируется с помощью функции hashFunc, которая возвращает индекс в массиве KeyValues хеш-таблицы.
3. Затем функция в цикле ищет ячейку с этим ключом, увеличивая индекс на 1 (с переходом в начало массива при достижении его конца), пока не найдет нужный ключ или не вернется к исходному индексу.
4. Если нужный ключ найден, функция возвращает его индекс в таблице.
5. Если функция вернулась к исходному индексу, это означает, что ключ не найден, и функция возвращает -1.

Таким образом, эта функция реализует поиск элемента в хеш-таблице с открытой адресацией. При этом учитывается возможность коллизий и удаления

элементов. Это позволяет эффективно находить элементы даже в условиях частых вставок и удалений.

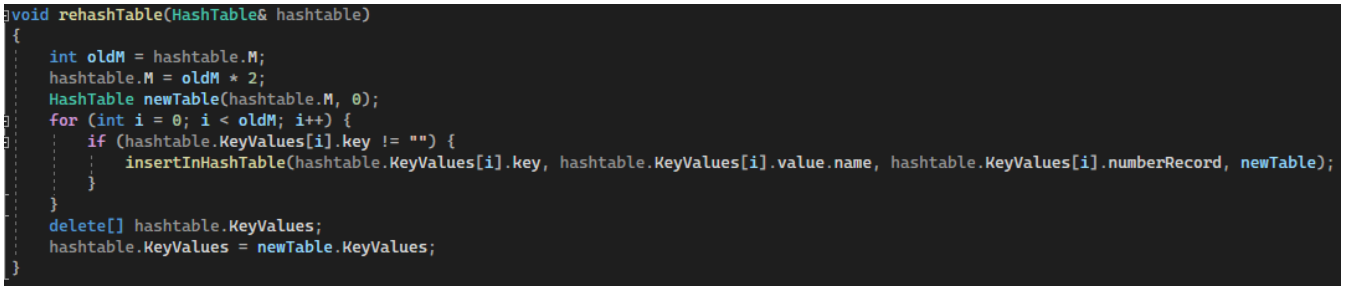
```
int findInHashTable(HashTable& hashtable, const string& dateOfBirth)
{
    int id = hashFunc(dateOfBirth, hashtable.M);
    int originalId = id;

    do {
        if (hashtable.KeyValues[id].key == dateOfBirth) {
            return id;
        }
        id = (id + 1) % hashtable.M;
    } while (id != originalId);
    return -1;
}
```

Рисунок 5 – Функция поиска по ключу

Функция rehashTable предназначена для перехеширования хеш-таблицы. Вот что делает каждый шаг:

1. Функция принимает ссылку на хеш-таблицу.
2. Размер таблицы удваивается, и создается новая таблица этого размера.
3. Затем функция в цикле переносит все элементы из старой таблицы в новую с помощью функции insertInHashTable. При этом учитывается, что в старой таблице могут быть удаленные элементы (с пустым ключом), которые не нужно переносить.
4. После переноса всех элементов старая таблица удаляется, и вместо нее используется новая.

Таким образом, эта функция реализует динамическое изменение размера хеш-таблицы при ее заполнении. Это позволяет поддерживать эффективность операций вставки, удаления и поиска даже при большом числе элементов. При этом используется метод открытой адресации для разрешения коллизий.



```
void rehashTable(HashTable& hashtable)
{
    int oldM = hashtable.M;
    hashtable.M = oldM * 2;
    HashTable newTable(hashtable.M, 0);
    for (int i = 0; i < oldM; i++) {
        if (hashtable.KeyValues[i].key != "") {
            insertInHashTable(hashtable.KeyValues[i].key, hashtable.KeyValues[i].value.name, hashtable.KeyValues[i].numberRecord, newTable);
        }
    }
    delete[] hashtable.KeyValues;
    hashtable.KeyValues = newTable.KeyValues;
}
```

Рисунок 6 – Функция рехеширования

Функция printHashTable предназначена для вывода содержимого хеш-таблицы. Вот что делает каждый шаг:

1. Функция принимает ссылку на хеш-таблицу.
2. Затем функция в цикле проходит по всем ячейкам таблицы.
3. Если ячейка не пуста (то есть ключ не пустой), то выводится ее индекс, ключ (дата рождения) и значение (имя).
4. Если ячейка была удалена (то есть в значении установлен флаг isDeleted), то выводится ее индекс и слово “УДАЛЕНО”.
5. Если ячейка пуста и не была удалена, то выводится ее индекс и слово “ПУСТО”.

Таким образом, эта функция позволяет увидеть структуру хеш-таблицы, включая распределение элементов по ячейкам и наличие удаленных элементов. Это может быть полезно для отладки и анализа эффективности работы хеш-таблицы.

```
void printHashTable(const HashTable& hashtable)
{
    cout << "Содержимое хеш-таблицы: " << endl;
    for (int i = 0; i < hashtable.M; i++) {
        if (hashtable.KeyValues[i].key != "") {
            cout << "Индекс " << i << ": Дата рождения=" << hashtable.KeyValues[i].key << ", Имя=" << hashtable.KeyValues[i].value.name << endl;
        } else if (hashtable.KeyValues[i].value.isDeleted) {
            cout << "Индекс " << i << ": УДАЛЕНО" << endl;
        } else {
            cout << "Индекс " << i << ": ПУСТО" << endl;
        }
    }
}
```

Рисунок 7 – Функция вывода хэш-таблицы

Функция writeToBinaryFile предназначена для записи содержимого хэш-таблицы в бинарный файл. Вот что делает каждый шаг:

1. Функция принимает ссылку на хэш-таблицу и ссылку на бинарный файл.
2. Производится попытку открытия файла.
3. Если файл не существует, выводится сообщение об ошибке.
4. В этот файл записываются размер таблицы и количество записей.
5. Затем функция в цикле проходит по всем ячейкам таблицы.
6. Для каждый отдельной ячейки в файл записываются такие данные, как ключ, имя и дата рождения.
7. После записи файл закрывается.

Таким образом, эта функция реализует запись хэш-таблицы в бинарный файл. Это позволяет сохранять данные хэш-таблицы для дальнейшей работы с ними, даже после перезапуска программы.

```
// Функция записи данных в бинарный файл
void writeToBinaryFile(const HashTable& hashtable, const string& filename)
{
    ofstream outFile(filename, ios::binary | ios::out);
    if (!outFile) {
        cout << "Ошибка открытия файла." << endl;
        return;
    }

    // Записываем размер таблицы и количество записей
    outFile.write(reinterpret_cast<const char*>(&hashtable.M), sizeof(hashtable.M));
    outFile.write(reinterpret_cast<const char*>(&hashtable.N), sizeof(hashtable.N));

    // Записываем каждую запись
    for (int i = 0; i < hashtable.M; ++i) {
        int keySize = hashtable.KeyValues[i].key.size();
        outFile.write(reinterpret_cast<const char*>(&keySize), sizeof(keySize));
        outFile.write(hashtable.KeyValues[i].key.c_str(), keySize);

        outFile.write(reinterpret_cast<const char*>(&hashtable.KeyValues[i].value), sizeof(FriendBirthday));
        outFile.write(reinterpret_cast<const char*>(&hashtable.KeyValues[i].numberRecord), sizeof(hashtable.KeyValues[i].numberRecord));
    }

    outFile.close();
}
```

Рисунок 8 – Функция записи данных в бинарный файл

Функция readFromBinaryFile предназначена для чтения содержимого хеш-таблицы из бинарного файла. Вот что делает каждый шаг:

1. Функция принимает ссылку на хэш-таблицу и ссылку на бинарный файл.
2. Производится попыткя открытия файла.
3. Если файл не существует, выводится сообщение об ошибке.

4. Из файла считываются размер таблицы и количество записей.

5. Создается новая таблица с полученными размером и количеством записей.

6. В цикле читается информация о каждой ячейке таблицы из файла.

7. Для каждой ячейки считываются данные: ключ, имя и дата рождения.

8. После завершения чтения данных из файла, файл закрывается.

Таким образом, эта функция позволяет загружать сохраненные данные хэш-таблицы из бинарного файла, восстанавливая состояние таблицы для дальнейшего использования в программе.

```
// Функция чтения данных из бинарного файла
void readFromBinaryFile(HashTable& hashtable, const string& filename)
{
    ifstream inFile(filename, ios::binary | ios::in);
    if (!inFile) {
        cout << "Ошибка открытия файла." << endl;
        return;
    }

    int M, N;
    inFile.read(reinterpret_cast<char*>(&M), sizeof(M));
    inFile.read(reinterpret_cast<char*>(&N), sizeof(N));

    HashTable newTable(M, N);

    for (int i = 0; i < M; ++i) {
        int keySize;
        inFile.read(reinterpret_cast<char*>(&keySize), sizeof(keySize));

        char* keyBuffer = new char[keySize + 1];
        inFile.read(keyBuffer, keySize);
        keyBuffer[keySize] = '\0';
        newTable.KeyValues[i].key = keyBuffer;
        delete[] keyBuffer;

        inFile.read(reinterpret_cast<char*>(&newTable.KeyValues[i].value), sizeof(FriendBirthday));
        inFile.read(reinterpret_cast<char*>(&newTable.KeyValues[i].numberRecord), sizeof(newTable.KeyValues[i].numberRecord));
    }

    inFile.close();
    delete[] hashtable.KeyValues;
    hashtable = newTable;
}
```

Рисунок 9 – Функция чтения данных из бинарного файла

2 Задание 2

Тип хеш-таблицы (метод разрешения коллизии):

– Открытый адрес (смещение на 1)

Структура записи двоичного файла:

– Дни рождения друзей: дата рождения, имя.

2.1 Код программы

В данной программе используется метод открытой адресации с линейным смещением на 1 для разрешения коллизий. Этот метод подразумевает следующее:

Вставка данных: когда происходит попытка вставить данные в хеш-таблицу, и возникает коллизия (когда два ключа сопоставляются одной ячейке хеш-таблицы), программа ищет следующую доступную пустую ячейку для вставки. Это достигается смещением на одну ячейку вперед от текущей позиции до тех пор, пока не будет найдена свободная ячейка для вставки данных.

Удаление данных: при удалении элемента из хеш-таблицы программа помечает соответствующую ячейку как пустую. Однако сохраняет информацию о том, что ячейка была ранее занята, чтобы избежать потери информации о друге.

Поиск данных: при поиске элемента программа использует алгоритм хеширования для определения начальной позиции и затем последовательно проверяет ячейки, смещаясь на одну позицию вперед до тех пор, пока не найдет элемент или не обнаружит пустую ячейку, указывающую на отсутствие данных.

Это реализация открытой адресации с использованием линейного метода разрешения коллизий, который использует линейное смещение для поиска следующей свободной ячейки при возникновении коллизии.

3 Листинг программы

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

using namespace std;

// Определение структуры FriendBirthday
struct FriendBirthday {
    string dateOfBirth;
    string name;
    bool openKey;
    bool isDeleted;

    FriendBirthday()
    {
        openKey = true;
        isDeleted = false;
    }

    FriendBirthday(string dateOfBirth, string name)
    {
        this->dateOfBirth = std::move(dateOfBirth);
        this->name = name;
        openKey = true;
        isDeleted = false;
    }
};

// Определение структуры HashTable
struct HashTable {
    int M;
    int N;
    struct KeyValuePair {
        string key;
        FriendBirthday value;
        int numberRecord;
    };
    KeyValuePair* KeyValues;

    HashTable(int M, int N)
    {
        this->M = M;
        this->N = N;
        this->KeyValues = new KeyValuePair[M];
        for (int i = 0; i < M; i++) {
            this->KeyValues[i].key = "";
        }
    }
};

int hashFunc(const string& key, int M)
{
    int sum = 0;
    for (char c : key) {
        sum += c;
    }
    return sum % M;
}
void rehashTable(HashTable& hashtable); // Прототип функции rehashTable
```

```
void insertInHashTable(const string& dateOfBirth, const string& name, int i, HashTable&
hashTable)
{
    string key = dateOfBirth;
    int id = hashFunc(key, hashTable.M);
    int originalId = id;

    do {
        if (hashTable.KeyValues[id].key == "") {
            hashTable.KeyValues[id].key = key;
            hashTable.KeyValues[id].numberRecord = i;
            FriendBirthday friendInfo(dateOfBirth, name);
            hashTable.KeyValues[id].value = friendInfo;
            hashTable.N++;
            if (static_cast<double>(hashTable.N) / hashTable.M > 0.7) {
                rehashTable(hashTable);
            }
            return;
        }
        id = (id + 1) % hashTable.M;
    } while (id != originalId);
    cout << "Хеш-таблица заполнена, вставка невозможна." << endl;
}

void deleteInHashTable(const string& dateOfBirth, HashTable& hashtable)
{
    int id = hashFunc(dateOfBirth, hashtable.M);
    while (hashtable.KeyValues[id].key != dateOfBirth) {
        id++;
        if (id >= hashtable.M) {
            id = 0;
        }
    }
    hashtable.KeyValues[id].key = "";
    hashtable.N--;
    hashtable.KeyValues[id].value.isDeleted = true;
    hashtable.KeyValues[id].value.openKey = true;
}

int findInHashTable(HashTable& hashtable, const string& dateOfBirth)
{
    int id = hashFunc(dateOfBirth, hashtable.M);
    int originalId = id;

    do {
        if (hashtable.KeyValues[id].key == dateOfBirth) {
            return id;
        }
        id = (id + 1) % hashtable.M;
    } while (id != originalId);
    return -1;
}

void rehashTable(HashTable& hashtable)
{
    int oldM = hashtable.M;
    hashtable.M = oldM * 2;
    HashTable newTable(hashtable.M, 0);
    for (int i = 0; i < oldM; i++) {
        if (hashtable.KeyValues[i].key != "") {
            insertInHashTable(hashtable.KeyValues[i].key,
hashtable.KeyValues[i].value.name, hashtable.KeyValues[i].numberRecord, newTable);
        }
    }
    delete[] hashtable.KeyValues;
    hashtable.KeyValues = newTable.KeyValues;
```

```
}

void printHashTable(const HashTable& hashtable)
{
    cout << "Содержимое хеш-таблицы: " << endl;
    for (int i = 0; i < hashtable.M; i++) {
        if (hashtable.KeyValues[i].key != "") {
            cout << "Индекс " << i << ": Дата рождения=" << hashtable.KeyValues[i].key <<
", Имя=" << hashtable.KeyValues[i].value.name << endl;
        }
        else if (hashtable.KeyValues[i].value.isDeleted) {
            cout << "Индекс " << i << ": УДАЛЕНО" << endl;
        }
        else {
            cout << "Индекс " << i << ": ПУСТО" << endl;
        }
    }
}

// Функция записи данных в бинарный файл
void writeToBinaryFile(const HashTable& hashtable, const string& filename)
{
    ofstream outFile(filename, ios::binary | ios::out);
    if (!outFile) {
        cout << "Ошибка открытия файла." << endl;
        return;
    }

    // Записываем размер таблицы и количество записей
    outFile.write(reinterpret_cast<const char*>(&hashtable.M), sizeof(hashtable.M));
    outFile.write(reinterpret_cast<const char*>(&hashtable.N), sizeof(hashtable.N));

    // Записываем каждую запись
    for (int i = 0; i < hashtable.M; ++i) {
        int keySize = hashtable.KeyValues[i].key.size();
        outFile.write(reinterpret_cast<const char*>(&keySize), sizeof(keySize));
        outFile.write(hashtable.KeyValues[i].key.c_str(), keySize);

        outFile.write(reinterpret_cast<const char*>(&hashtable.KeyValues[i].value),
sizeof(FriendBirthday));
        outFile.write(reinterpret_cast<const char*>(&hashtable.KeyValues[i].numberRecord),
sizeof(hashtable.KeyValues[i].numberRecord));
    }

    outFile.close();
}

// Функция чтения данных из бинарного файла
void readFromBinaryFile(HashTable& hashtable, const string& filename)
{
    ifstream inFile(filename, ios::binary | ios::in);
    if (!inFile) {
        cout << "Ошибка открытия файла." << endl;
        return;
    }

    int M, N;
    inFile.read(reinterpret_cast<char*>(&M), sizeof(M));
    inFile.read(reinterpret_cast<char*>(&N), sizeof(N));

    HashTable newTable(M, N);

    for (int i = 0; i < M; ++i) {
        int keySize;
        inFile.read(reinterpret_cast<char*>(&keySize), sizeof(keySize));
```

```
        char* keyBuffer = new char[keySize + 1];
        inFile.read(keyBuffer, keySize);
        keyBuffer[keySize] = '\0';
        newTable.KeyValues[i].key = keyBuffer;
        delete[] keyBuffer;

        inFile.read(reinterpret_cast<char*>(&newTable.KeyValues[i].value),
sizeof(FriendBirthday));
        inFile.read(reinterpret_cast<char*>(&newTable.KeyValues[i].numberRecord),
sizeof(newTable.KeyValues[i].numberRecord));
    }

    inFile.close();
    delete[] hashtable.KeyValues;
    hashtable = newTable;
}

int main()
{
    setlocale(0, "");
    int M = 5;
    int N = 0;
    vector<string> datesOfBirth = { "01-01-1990", "02-02-1991", "03-03-1992", "04-04-
1993", "05-05-1994" };
    vector<string> names = { "Alice", "Bob", "Charlie", "David", "Eva" };

    HashTable hashTable(M, N);
    int choice;

    while (true) {
        cout << "Меню:" << endl;
        cout << "1. Добавить друга" << endl;
        cout << "2. Удалить друга" << endl;
        cout << "3. Найти друга" << endl;
        cout << "4. Вывести хеш-таблицу" << endl;
        cout << "5. Записать данные в бинарный файл" << endl;
        cout << "6. Записать данные из бинарного файла" << endl;
        cout << "7. Выход" << endl;
        cout << "Введите ваш выбор: ";
        cin >> choice;

        switch (choice) {
        case 1: {
            for (int i = 0; i < 5; i++) {
                insertInHashTable(datesOfBirth[i], names[i], i, hashTable);
            }
            break;
        }
        case 2: {
            string dateOfBirth;
            cout << "Введите дату рождения друга для удаления: ";
            cin >> dateOfBirth;
            int index = findInHashTable(hashTable, dateOfBirth);
            if (index != -1) {
                deleteInHashTable(dateOfBirth, hashTable);
                cout << "Друг удален." << endl;
            }
            else {
                cout << "Друг не найден." << endl;
            }
            break;
        }
        case 3: {
            string dateOfBirth;
            cout << "Введите дату рождения друга для поиска: ";
            cin >> dateOfBirth;
```

```
            int index = findInHashTable(hashTable, dateOfBirth);
            if (index != -1) {
                cout << "Найдено: Дата рождения=" << hashTable.KeyValues[index].key << ",
Имя=" << hashTable.KeyValues[index].value.name << endl;
            }
            else {
                cout << "Друг не найден." << endl;
            }
            break;
        }
        case 4:
            printHashTable(hashTable);
            break;
        case 5:
            writeToBinaryFile(hashTable, "friends.bin");
            cout << "Данные сохранены в файл." << endl;
            break;
        case 6:
            readFromBinaryFile(hashTable, "friends.bin");
            cout << "Данные загружены из файла." << endl;
            break;
        case 7:
            return 0;
        default:
            cout << "Неверный выбор. Пожалуйста, попробуйте еще раз." << endl;
        }
    }

    return 0;
}
```

4 Тестирование программы

```
Меню:
1. Добавить друга
2. Удалить друга
3. Найти друга
4. Вывести хеш-таблицу
5. Записать данные в бинарный файл
6. Записать данные из бинарного файла
7. Выход
Введите ваш выбор: 1
Меню:
1. Добавить друга
2. Удалить друга
3. Найти друга
4. Вывести хеш-таблицу
5. Записать данные в бинарный файл
6. Записать данные из бинарного файла
7. Выход
Введите ваш выбор: 4
Содержимое хеш-таблицы:
Индекс 0: ПУСТО
Индекс 1: Дата рождения=03-03-1992, Имя=Charlie
Индекс 2: ПУСТО
Индекс 3: ПУСТО
Индекс 4: Дата рождения=04-04-1993, Имя=David
Индекс 5: Дата рождения=01-01-1990, Имя=Alice
Индекс 6: ПУСТО
Индекс 7: Дата рождения=05-05-1994, Имя=Eva
Индекс 8: Дата рождения=02-02-1991, Имя=Bob
Индекс 9: ПУСТО
Меню:
1. Добавить друга
2. Удалить друга
3. Найти друга
4. Вывести хеш-таблицу
5. Записать данные в бинарный файл
6. Записать данные из бинарного файла
7. Выход
Введите ваш выбор: 5
Данные сохранены в файл.
Меню:
1. Добавить друга
2. Удалить друга
3. Найти друга
4. Вывести хеш-таблицу
5. Записать данные в бинарный файл
6. Записать данные из бинарного файла
7. Выход
Введите ваш выбор: 6
Данные загружены из файла.
Меню:
1. Добавить друга
2. Удалить друга
3. Найти друга
4. Вывести хеш-таблицу
5. Записать данные в бинарный файл
6. Записать данные из бинарного файла
7. Выход
Введите ваш выбор: 2
Введите дату рождения друга для удаления: 03-03-1992
Друг удален.
```

Рисунок 10 – Вывод программы

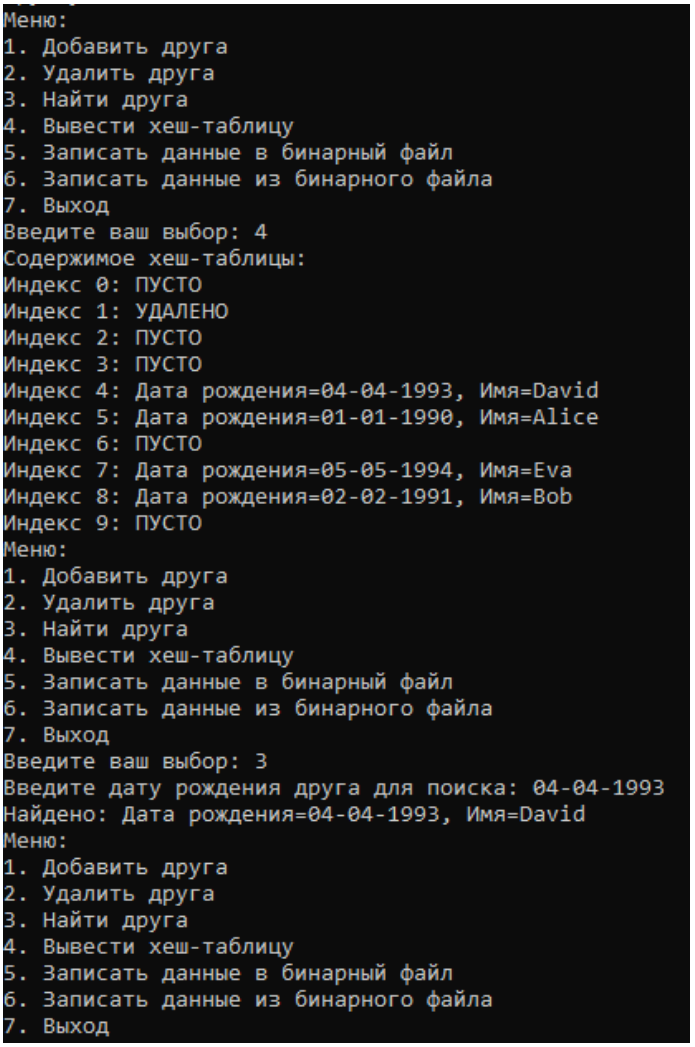


Рисунок 11 – Вывод программы