|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | |  |  |  | | 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-07-21 | Дамарад Д.В. |
| Принял преподаватель | Скворцова Л.А. |

|  |  |  |
| --- | --- | --- |
| Лабораторная работа выполнена | «\_\_»\_\_\_\_\_\_\_202\_\_ г. | *(подпись студента)* |
|  |  |  |
| «Зачтено» | «\_\_»\_\_\_\_\_\_\_202\_\_ г. | *(подпись руководителя)* |

**СОДЕРЖАНИЕ**

[**ЦЕЛЬ РАБОТЫ** 3](#_Toc118843025)

[**ПОСТАНОВКА ЗАДАЧИ** 3](#_Toc118843026)

[**РЕШЕНИЕ** 6](#_Toc118843027)

[**Ответы на вопросы:** 6](#_Toc118843028)

[**Функции задания** 8](#_Toc118843029)

[**Интерфейс** 17](#_Toc118843030)

[**Тестирование** 18](#_Toc118843031)

[**ВЫВОД** 22](#_Toc118843032)

[**ПОЛНЫЙ КОД ПРОГРАММЫ** 22](#_Toc118843033)

# **ЦЕЛЬ РАБОТЫ**

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

# **ПОСТАНОВКА ЗАДАЧИ**

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.
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.

1. 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.

1. 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.

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.

− Включить в файл записи как не приводящие к коллизиям, так и приводящие.

− Обеспечить включение в файл такого количества записей, чтобы потребовалось рехеширование.

Заполнить файл большим количеством записей (до 1 000 000).

− Определить время чтения записи с заданным ключом: для первой записи файла, для последней и где-то в середине. Убедиться (или нет), что время доступа для всех записей одинаково.

1. Составить отчет.
2. Ответы на вопросы:

1. Расскажите о назначении хеш-функции.

2. Что такое коллизия?

3. Что такое «открытый адрес» по отношению к хеш-таблице?

4. Как в хеш-таблице с открытым адресом реализуется коллизия?

5. Какая проблема, может возникнуть после удаления элемента из хеш-таблицы с открытым адресом и как ее устранить?

6. Что определяет коэффициент нагрузки в хеш-таблице?

7. Что такое «первичный кластер» в таблице с открытым адресом?

8. Как реализуется двойное хеширование?

Вариант №6:

|  |  |
| --- | --- |
| Цепное хеширование | Товар: название, артикул – шестиразрядное число, завод изготовитель, цена, страна (название). |

# **РЕШЕНИЕ**

# **Ответы на вопросы:**

1. Расскажите о назначении хеш-функции.

Хеш-таблица — это структура данных, в которой все элементы хранятся в виде пары ключ-значение, где:

* ключ — уникальное число, которое используется для индексации значений и которое генерируется специализированной хеш-функция;
* значение — данные, которые с этим ключом связаны.

Хеш-функция - функция, которая на основании некоторых входных данных генерирует уникальный (не всегда) ключ. Идеальная хеш-функция не вызывает коллизий в хеш-таблице, не требует много вычислительной мощности и не имеет привязи к данным на входе.

1. Что такое коллизия?

Коллизия – явление при заполнении хеш-таблицы, при котором хеш-функция возвращает два одинаковых ключа на разные записи

Есть несколько методов борьбы с коллизиями:

Метод цепочек: если хеш-функция выделяет один индекс сразу двум элементам, то храниться они будут в одном и том же индексе, но уже с помощью двусвязного списка. Если j — ячейка для нескольких элементов, то она содержит указатель на первый элемент списка. Если же j пуста, то она содержит NULL. Каждая ячейка хэш-таблицы является однонаправленным списком.

В отличие от метода цепочек, в открытой адресации несколько элементов в одной ячейке храниться не могут. Суть этого метода заключается в том, что каждая ячейка либо содержит одну запись, либо ничего, добивается это методом сдвига. Например, открытый адрес со смещением предусматривает сдвиг индекса, пока не найдется свободная ячейка.

Двойное хеширование при использовании метода открытой адресации: если возникает коллизия, то она разрешается путем расчета сдвига при помощи вторичной хеш-функции.

В универсальном хешировании хеш-функция выбирается случайным образом и не зависит от ключей.

1. Что такое «открытый адрес» по отношению к хеш-таблице?

Открытый адрес – это свободный элемент таблицы, а закрытый адрес – это занятый элемент.

1. Как в хеш-таблице с открытым адресом реализуется коллизия?

В открытой адресации несколько элементов в одной ячейке храниться не могут. Суть этого метода заключается в том, что каждая ячейка либо содержит одну запись, либо ничего, добивается это методом сдвига. Например, открытый адрес со смещением предусматривает сдвиг индекса, пока не найдется свободная ячейка.

1. Какая проблема, может возникнуть после удаления элемента из хеш-таблицы с открытым адресом и как ее устранить?

При такой операции может возникнуть неправильное поведение при дальнейшем обращении к таблице. Например, если были вставлены элементы А, Б с одним хешем, то после удаления элемента А при поиске элемента Б будет возвращен NULL, что неверно. Чтобы этого избежать нужно либо заполнить ячейку таблицы служебной комбинацией, которая несет смысл «ищи дальше», или перезаписать все элементы с данным хешем, сдвинув их.

1. Что определяет коэффициент нагрузки в хеш-таблице?

Коэффициентом нагрузки называется такое число, показывающее число хранимых элементов в массиве, разделенное на размер самого массива. Используется для отслеживания необходимости расширения и рехеширования.

1. Что такое «первичный кластер» в таблице с открытым адресом?

Первичным кластером называется некоторое количество ячеек, заполненных и расположенных рядом друг с другом. В таком случае время поиска места для вставки сильно увеличивается.

1. Как реализуется двойное хеширование?

Двойное хеширование является подвидом хеширования с использованием открытой адресации, но вместо, например, единичного сдвига, используется величина, сгенерированная вспомогательной хеш-функцией.

# **Функции задания**

Для организации записей используется структура Product:

struct Product {

char name[20];

char manufacturer[20];

char country[20];

int price = 0;

int code = 0;

};

Функция printRecord() выводит содержимое переданной структуры Record в консоль:

void printRecord(Product product) {

cout << product.name << " " << product.code << " " << product.manufacturer << " " << product.price << " " << product.country << endl;

}

Некоторые функции управления бинарным файлом заимствованы из практической работы №2.

Функция AddRecordToBin() принимает на вход имя файла и запись, при помощи флага ios::app и функции write() добавляет запись в конец файла.

void AddRecordToBin(string bin\_file\_name, Product product) {

fstream fout;

fout.open(bin\_file\_name, ios::binary | ios::in | ios::out);

if (fout.is\_open()) {

fout.seekg(0, ios::end);

fout.write((char\*)&product, sizeof(Product));

}

else {

cout << "Файла с таким именем не существует" << endl;

}

}

Функция DirectAccess() принимает на вход имя файла и номер записи, осуществляет прямой доступ к записи файла по номеру при помощи функций seekg (используется для установки указателя на необходимую строку), read (используется для чтения данных и записи их в структуру):

Product DirectAccess(string bin\_file\_name, int number) {

ifstream binFile(bin\_file\_name, ios::binary | std::ios::in);

if (!binFile.is\_open()) {

cout << "Файла с таким именем не существует" << endl;

}

binFile.seekg(0, ios::end);

long long size = binFile.tellg();

binFile.seekg(0, std::ios::beg);

if ((PRODUCT\_SIZE \* sizeof(char)) \* number > size) {

binFile.close();

return {};

}

binFile.seekg((PRODUCT\_SIZE)\*number);

Product product;

binFile.read((char\*)&product, sizeof(product));

binFile.close();

return product;

}

Функция DeleteByKey() принимает на вход имя файла, ключ и длину записи, осуществляет удаление необходимой записи путем выборочного перезаписывания в новый файл и дальнейшего его переименования.

bool DeleteByKey(string bin\_file\_name, int key) {

ifstream bitFile(bin\_file\_name, ios::binary | ios::in);

if (!bitFile.is\_open()) {

cout << "Файла с таким именем не существует" << endl;

}

ofstream tempFile("temp.bin", ios::out | ios::binary);

if (!tempFile.is\_open()) {

bitFile.close();

cout << "Файла с таким именем не существует" << endl;

}

bitFile.seekg(0, ios::end);

int size = bitFile.tellg();

bitFile.seekg(0, ios::beg);

string s;

Product product;

while (bitFile.read((char\*)&product, sizeof(product))) {

if (product.code == key) {

continue;

}

tempFile.write((char\*)&product, sizeof(product));

}

bitFile.close();

tempFile.close();

remove(bin\_file\_name.c\_str());

if (rename("temp.bin", bin\_file\_name.c\_str()) == 0) {

return true;

}

else {

return false;

}

}

Функция testBinF() выполняет автоматизированное тестирование функций, управляющих бинарными файлами: вывод в консоль, добавление записи в файл, удаление по ключу, прямой доступ к файлу:

void testBinF() {

cout << "Создание файла..." << endl;

string bin\_file\_name = "test.bin";

ConvertTextToBin(bin\_file\_name, "file.txt");

OutputBinFile(bin\_file\_name);

cout << "-----------------------" << endl;

cout << "Добавление записи..." << endl;

Product product = { "orange", "orange\_factory", "egypt", 1, 343434 };

AddRecordToBin(bin\_file\_name, product);

OutputBinFile(bin\_file\_name);

cout << "-----------------------" << endl;

cout << "Удаление записи..." << endl;

DeleteByKey(bin\_file\_name, 343434);

OutputBinFile(bin\_file\_name);

cout << "-----------------------" << endl;

cout << "Прямой доступ к 0 записи..." << endl;

printRecord(DirectAccess(bin\_file\_name, 0));

cout << "-----------------------" << endl;

}

Для управления таблицей создадим структуру Node:

* Поле key – целое число, являющееся ключом записи;
* Поле value – целое число, номер строки в бинарном файле;
* Поле next – указатель типа Node, обязательный элемент однонаправленного списка, указывающий на следующий элемент структуры.

struct Node {

int key;

int value;

Node\* next;

};

Для хеш-таблицы создадим отдельный класс HashTable:

* Защищенное поле table типа Node\*\*, хранящее саму хеш-таблицу;
* Защищенное поле currentSize, хранящее число записей в таблице;
* Открытое поле size, хранящее максимальный размер хеш-таблицы;
* Конструктор hashTable();
* Деструктор ~HashTable();
* Хеш-функция hash целочисленного типа;
* Методы get, add, remove, rehash, print.

class HashTable {

protected:

Node\*\* table;

int currentSize = 0;

public:

HashTable();

~HashTable();

int hash(int key);

void add(int key, int value, bool isRehash = false);

void remove(int key);

int get(int key);

void rehash();

void print();

int size = 100;

};

Конструктор hashTable() выделяет память под хеш-таблицу и при помощи цикла заполняет массив указателей значениями nullptr:

HashTable::HashTable() {

this->size = size;

table = new Node \* [size];

for (int i = 0; i < size; i++) {

table[i] = nullptr;

}

}

Деструктор ~HashTable() по окончании работы программы при помощи цикла удаляет все данные из массива, попутно освобождая память:

HashTable::~HashTable() {

for (int i = 0; i < size; i++) {

Node\* node = table[i];

while (node != nullptr) {

Node\* temp = node;

node = node->next;

delete temp;

}

}

delete[] table;

}

Метод HashTable::hash() является хеш-функцией разработанной таблицы. На вход принимает целое число-ключ, делит его на 17 и возвращает остаток от деления:

int HashTable::hash(int key) {

return key % 17;

}

Метод HashTable::add() добавляет запись в направленный список хеш-таблицы, перезаписывая первый элемент в хеш-таблице с необходимым индексом:

void HashTable::add(int key, int value, bool isRehash) {

currentSize++;

if (currentSize / float(size) > 0.9 or isRehash) {

rehash();

}

int index = hash(key);

Node\* node = table[index];

table[index] = new Node;

table[index]->key = key;

table[index]->value = value;

table[index]->next = node;

}

Метод HashTable::remove() удаляет запись из хеш-таблицы. После получения необходимого индекса изменяется однонаправленный список в соответствии с позицией записи: в хеш таблицу записывается либо нулевой указатель, либо указатель на существующую запись:

void HashTable::remove(int key) {

int index = hash(key);

Node\* node = table[index];

Node\* prev = nullptr;

while (node != nullptr) {

if (node->key == key) {

if (prev == nullptr) {

table[index] = node->next;

}

else {

prev->next = node->next;

}

delete node;

return;

}

prev = node;

node = node->next;

}

}

Метод HashTable::get() принимает на вход ключ-строку и в соответствии с хэшем данного ключа проверяется весь направленный список до соответствия, возвращается значение – номер строки в файле. Если совпадений нет, возвращается -1.

int HashTable::get(int key) {

int index = hash(key);

Node\* node = table[index];

while (node != nullptr) {

if (node->key == key) {

return node->value;

}

node = node->next;

}

return -1;

}

Метод HashTable::rehash() не принимает ничего на вход и выполняет функцию расширения размера хеш-таблицы. Метод выделяет память, в раза большую, чем до этого, далее копирует все данные из предыдущего массива и освобождает память:

void HashTable::rehash() {

int oldSize = size;

size \*= 2;

Node\*\* oldTable = table;

table = new Node \* [size];

for (int i = 0; i < size; i++) {

table[i] = nullptr;

}

for (int i = 0; i < oldSize; i++) {

Node\* node = oldTable[i];

while (node != nullptr) {

add(node->key, node->value);

node = node->next;

}

}

for (int i = 0; i < oldSize; i++) {

Node\* node = oldTable[i];

while (node != nullptr) {

Node\* temp = node;

node = node->next;

delete temp;

}

}

delete[] oldTable;

}

Метод HashTable::print() ничего не принимает на вход и при помощи двумерного цикла выводит в консоль все данные, которые хранит хэш таблица в формате «Индекс ячейки | номер строки»:

void HashTable::print() {

for (int i = 0; i < size; i++) {

Node\* node = table[i];

while (node != nullptr) {

cout << "[" << i << "]" << " | " << node->key << " " << node->value << endl;

node = node->next;

}

}

}

Функция testHeshT() выполняет автоматизированное тестирование всех методов класса: добавление ключа с коллизией, без коллизии, вывод в консоль, удаление ключа из таблицы, расширение таблицы:

void testHeshT() {

cout << "Создание хэш-таблицы..." << endl;

HashTable hashTable;

hashTable.add(0, 1);

hashTable.add(1, 1);

hashTable.add(1, 2);

hashTable.add(1, 3);

hashTable.add(2, 4);

hashTable.add(3, 5);

hashTable.print();

cout << "-----------------------" << endl;

cout << "Удаление элемента..." << endl;

hashTable.remove(1);

hashTable.print();

cout << "-----------------------" << endl;

cout << "Получение элемента с ключом 1..." << endl;

cout<<hashTable.get(1)<<endl;

cout << "-----------------------" << endl;

cout << "Добавление элемента..." << endl;

hashTable.add(1, 1000);

hashTable.print();

cout << "-----------------------" << endl;

cout <<"Тест хэш-функции..." << endl;

cout << hashTable.hash(1) << endl;

cout << hashTable.hash(2) << endl;

cout << "-----------------------" << endl;

cout <<"Рехэширование...." << endl;

cout << "rehash...\nsize before = "<< hashTable.size << endl;

hashTable.rehash();

cout << "size after = " << hashTable.size << endl;

cout << "-----------------------" << endl;

cout <<"Добавление и рехэшивание..." << endl;

hashTable.add(77, 1, true);

hashTable.print();

cout << "after rehash add size = " << hashTable.size << endl;

cout << "-----------------------" << endl;

}

Функция InsertBinHash() заполняет хеш-таблицу на основании данных из переданного бинарного файла. Также принимает на вход номер строки. Если номер равен -1, то переносится весь файл, иначе вносится номер указанной записи:

void InsertBinHash(HashTable& hashTable, string bin\_file\_name, int number) {

fstream bitFile(bin\_file\_name, ios::binary | ios::in | ios::app);

if (!bitFile.is\_open()) {

cout << "Файл не найден";

return;

}

int cnt = 0;

if (number == -1) {

while (!bitFile.eof()) {

Product product;

bitFile.read((char\*)&product, sizeof(Product));

hashTable.add(product.code, cnt);

cnt++;

}

}

else {

int temp;

temp = DirectAccess(bin\_file\_name, number).code;

hashTable.add(temp, number);

}

}

Функция DeleteByKeyHash() принимает на вход таблицу, имя бинарного файла и строку-ключ. После обращения к хеш-таблице в переменную записывается номер записи, запись удаляется из таблицы и файла:

void DeleteByKeyHash(HashTable& hashTable, string bin\_file\_name, int key) {

int number = hashTable.get(key);

if (number == -1) {

cout << "Ошибка" << endl;

return;

}

hashTable.remove(key);

DeleteByKey(bin\_file\_name, key);

}

Функция GetByKeyHash() принимет на вход таблицу, имя бинарного файла и строку-ключ. После обращения к хеш-таблице в переменную записывается номер записи, при помощи функции осуществляется прямой доступ:

void GetByKeyHash(HashTable& hashTable, string bin\_file\_name, int key) {

int number = hashTable.get(key);

if (number == -1) {

cout << "Ошибка" << endl;

return;

}

printRecord(DirectAccess(bin\_file\_name, number));

}

# **Интерфейс**

При запуске программы пользователю предоставляется выбор одного из предложенных заданий (Рисунок 1), указанных в постановке задачи. Любой другой введенный символ или число не входящее в диапазон [0;7] спровоцирует повторный ввод.

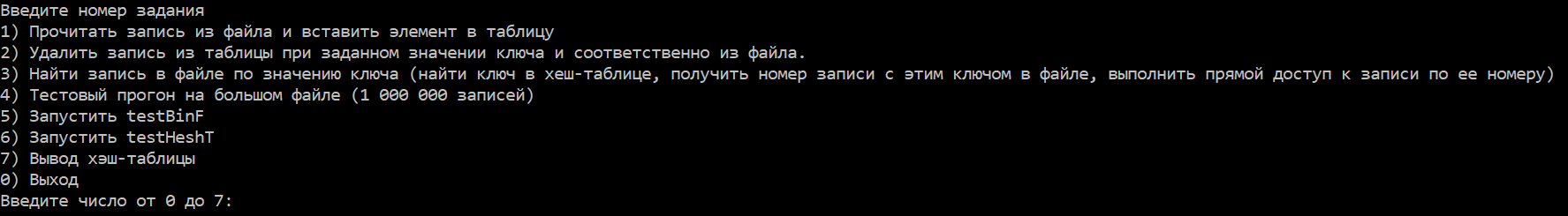


Рисунок 1 - Интерфейс программы

# **Тестирование**

Функция testiBinF() производит автоматизированное тестирование функций бинарного файла, добавляя записи в файл, а затем удаляя их, сопровождая выводом файла после каждой операции. Видно, что все операции выполняются корректно (Рисунок 2):

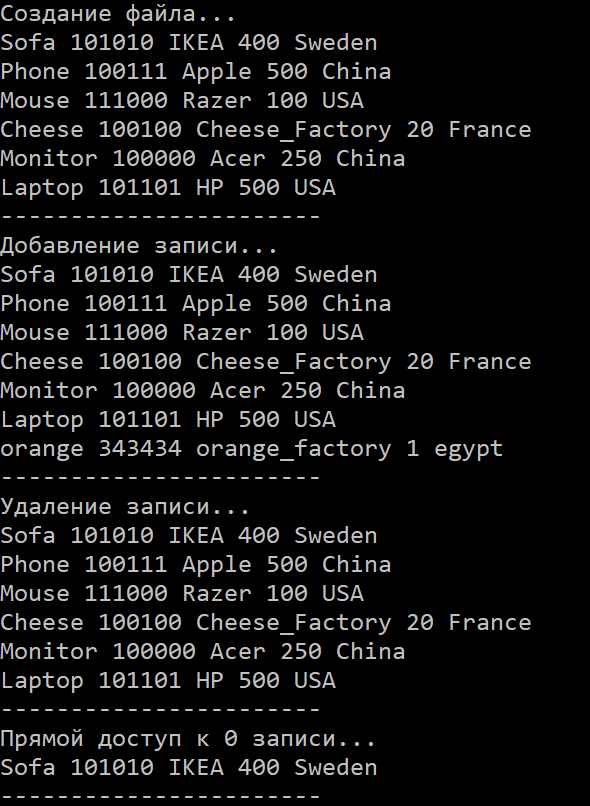


Рисунок 2 - Тестирование функции testiBinF()

Функция testHashT() выполняет автоматизированное тестирование функционала хеш-таблицы. Происходит добавление ключей с коллизией, без коллизии, удаление по ключу, поиск после удаления, добавление коллизии после удаления, результаты работы хеш-функции, рехеширование таблицы с наглядной демонстрацией ее размеров до и после, а также вывод всех элементов таблицы. Видно, что все функции работают корректно (Рисунок 3):

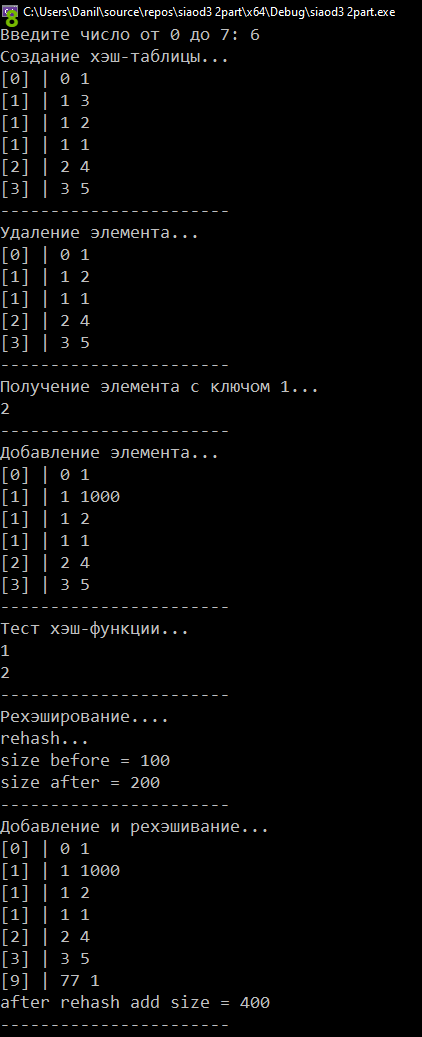


Рисунок 3 - Тестирование функции testHashT()

Функция InsertBinHash() заносит в хеш-таблицу необходимую запись из переданного файла. Зададим сначала ключ -1, чтобы внести все записи из файла, а затем ключ 2, чтобы внести запись с индексом 2 (Рисунок 4).

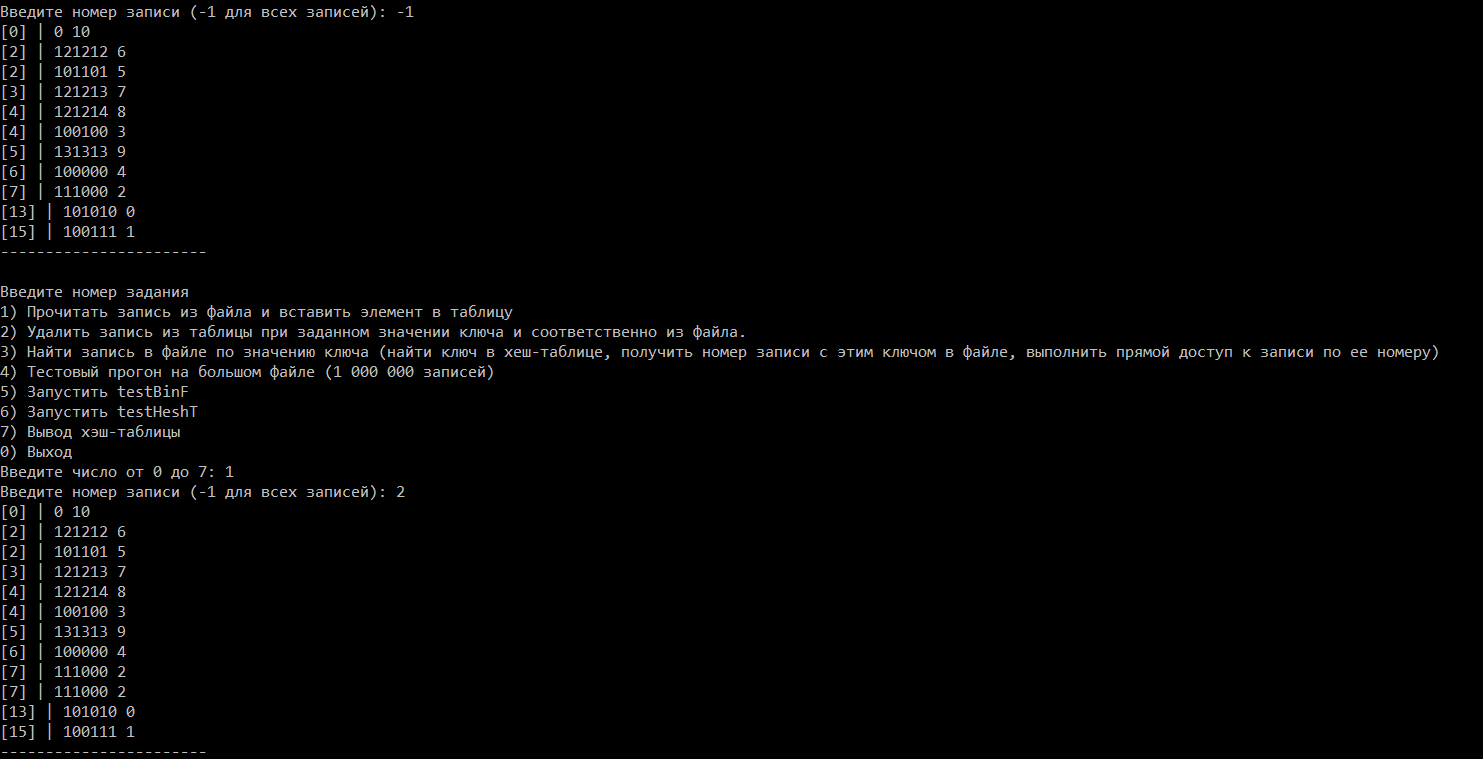


Рисунок 4 - Тестирование функции InsertBinHash()

Функция DeleteByKeyHash() удаляет элемент из хэш-таблицы и файла по заданному ключу. Удалим записи с ключом 0 и 100100 (Рисунок 5).



Рисунок 5 - Тестирование функции DeleteByKeyHash()

Функция GetByKeyHash() считывает из хеш-таблицы запись по введенному ключу и выводит ее в консоль (Рисунок 6). Получим записи с ключами 101010 и 111000. Функция работает корректно.

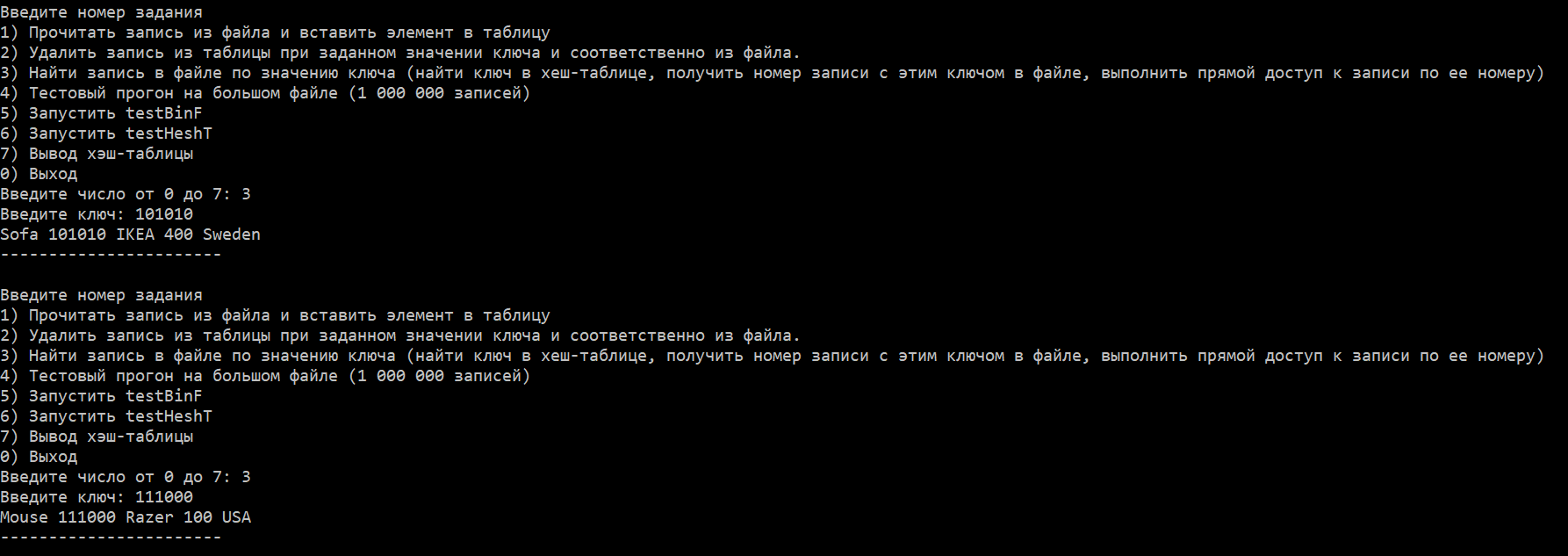


Рисунок 6 - Тестирование функции GetByKeyHash()

В задании 4 проведем замер времени выполнения функций поиска в хеш-таблице на 500000 записей использую встроенную библиотеку ctime. Выполним поиск первой записи, последней записи и записи, расположенной в середине файла. Видно, что время поиска у всех трех вариантов практически мгновенное (Рисунок 7).

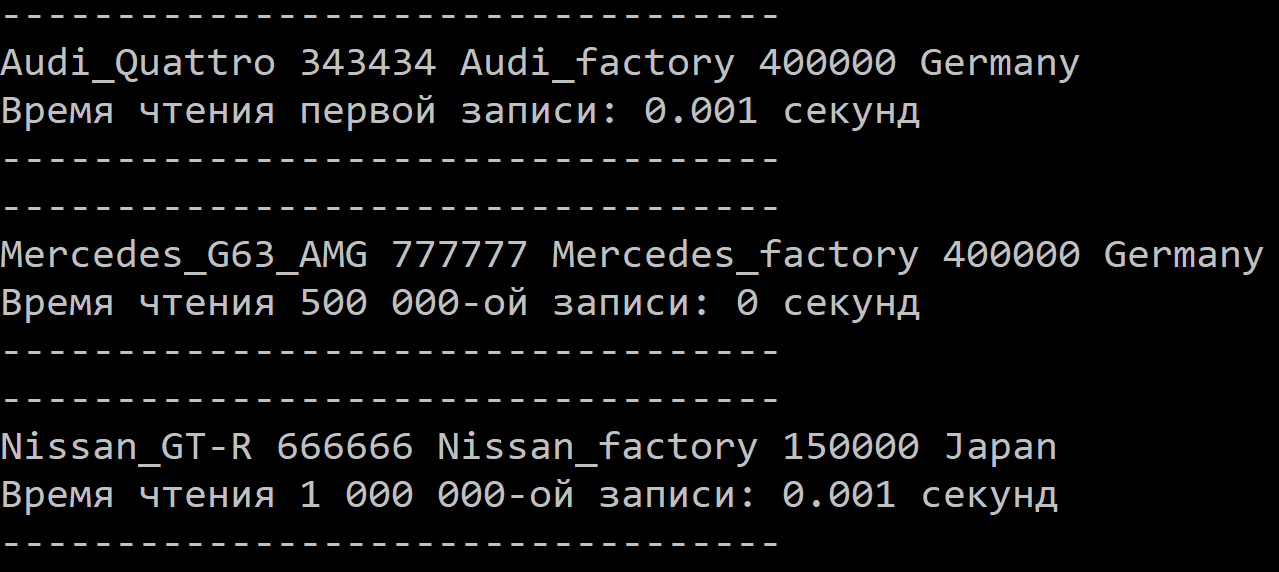


Рисунок 7 - Тестирование поиска записи в большой таблице

# **ВЫВОД**

В результате выполнения работы были:

1. Освоены принципы использования хеширования на примере хеш-таблицы с цепным хешированием
2. Связаны хеш-таблица и файл
3. Протестированы все спроектированные функции

# **ПОЛНЫЙ КОД ПРОГРАММЫ**

Файл main.cpp

#include <iostream>

#include <stdlib.h>

#include <ctime>

#include <iomanip>

#include <algorithm>

#include "Product.h"

#include "binfile.h"

#include "hashtable.h"

#include "hashfile.h"

using namespace std;

int inputMenu(int& number) {

cout << "Введите номер задания" << endl;

cout << "1) Прочитать запись из файла и вставить элемент в таблицу" << endl;

cout << "2) Удалить запись из таблицы при заданном значении ключа и соответственно из файла." << endl;

cout << "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)" << endl;

cout << "4) Тестовый прогон на большом файле (1 000 000 записей)" << endl;

cout << "5) Запустить testBinF" << endl;

cout << "6) Запустить testHeshT" << endl;

cout << "7) Вывод хэш-таблицы" << endl;

cout << "0) Выход" << endl;

while (1) {

cout << "Введите число от 0 до 7: ";

if ((cin >> number).good() && (number >= 0) && (number <= 7)) {

return number;

}

if (cin.fail()) {

cin.clear();

cout << endl << "Неверный ввод, повторите." << endl << endl;

}

else {

cout << endl << "Число вне допустимого диапазона значений. Повторите ввод." << endl << endl;

}

cin.ignore(100, '\n');

}

}

int main() {

system("chcp 1251 > null");

HashTable hashtable;

int menu;

string bin\_file\_name = "file.bin";

while (true) {

inputMenu(menu);

if (menu == 0) {

cout << "Задание не выбрано\nВыход из программы..." << endl;

break;

}

switch (menu) {

case 1: {

int index;

cout << "Введите номер записи (-1 для всех записей): ";

cin >> index;

InsertBinHash(hashtable, bin\_file\_name, index);

hashtable.print();

cout << "-----------------------" << endl << endl;

break;

}

case 2: {

int key;

cout << "Введите ключ: ";

cin >> key;

DeleteByKeyHash(hashtable, bin\_file\_name, key);

hashtable.print();

cout << "-----------------------" << endl << endl;

break;

}

case 3: {

int key;

cout << "Введите ключ: ";

cin >> key;

GetByKeyHash(hashtable, bin\_file\_name, key);

cout << "-----------------------" << endl << endl;

break;

}

case 4: {

//ConvertTextToBin("bigfile.bin", "bigfile.txt");

//OutputBinFile("bigfile.bin");

double search\_time;

clock\_t start\_time, end\_time;

HashTable big\_hashtable;

InsertBinHash(big\_hashtable, "bigfile.bin");

cout << "----------------------------------" << endl;

start\_time = clock();

GetByKeyHash(big\_hashtable, "bigfile.bin", 343434);

end\_time = clock();

search\_time = double(end\_time - start\_time) / CLOCKS\_PER\_SEC;

cout << "Время чтения первой записи: " << search\_time << " секунд" << endl;

cout << "----------------------------------" << endl;

cout << "----------------------------------" << endl;

start\_time = clock();

GetByKeyHash(big\_hashtable, "bigfile.bin", 777777);

end\_time = clock();

search\_time = double(end\_time - start\_time) / CLOCKS\_PER\_SEC;

cout << "Время чтения 500 000-ой записи: " << search\_time << " секунд" << endl;

cout << "----------------------------------" << endl;

cout << "----------------------------------" << endl;

start\_time = clock();

GetByKeyHash(big\_hashtable, "bigfile.bin", 666666);

end\_time = clock();

search\_time = double(end\_time - start\_time) / CLOCKS\_PER\_SEC;

cout << "Время чтения 1 000 000-ой записи: " << search\_time <<" секунд" << endl;

cout << "----------------------------------" << endl;

break;

}

case 5: {

testBinF();

break;

}

case 6: {

testHeshT();

break;

}

case 7: {

hashtable.print();

}

default: {

cout << "Задание не выбрано\nВыход из программы..." << endl;

break;

}

}

}

return 0;

}

Файл Product.h

#ifndef PRODUCT\_H

#define PRODUCT\_H

struct Product {

char name[20];

char manufacturer[20];

char country[20];

int price = 0;

int code = 0;

};

static int PRODUCT\_SIZE = sizeof(Product);

void printRecord(Product product);

#endif // !1

Файл Product.cpp

#include "Product.h"

#include "Product.h"

#include <iostream>

using namespace std;

void printRecord(Product product) {

cout << product.name << " " << product.code << " " << product.manufacturer << " " << product.price << " " << product.country << endl;

}

Файл binfile.h

#ifndef BINFILE\_H

#define BINFILE\_H

#include "Product.h"

#include <fstream>

#include <iostream>

using namespace std;

void ConvertTextToBin(string bin\_file\_name, string txt\_file\_name);

bool CheckIndex(string bin\_file\_name, int index);

Product GetRecordFromBin(string path, int index);

void AddRecordToBin(string path, Product product);

void OutputBinFile(string bin\_file\_name);

bool DeleteByKey(string bin\_file\_name, int key);

Product DirectAccess(string bin\_file\_name, int number);

void testBinF();

#endif

Файл binfile.cpp

#include "binfile.h"

void ConvertTextToBin(string bin\_file\_name, string txt\_file\_name) {

ifstream fin(txt\_file\_name, ios::in);

if (fin.is\_open()) {

ofstream fout(bin\_file\_name, ios::out | ios::binary);

Product product;

while (!fin.eof()) {

fin >> product.name >> product.code >> product.manufacturer >> product.price >> product.country;

fout.write((char\*)&product, sizeof(Product));

}

fout.close();

}

else {

cout << "Файла с таким именем не существует" << endl;

}

fin.close();

}

bool CheckIndex(string bin\_file\_name, int index) {

ifstream fin;

fin.open(bin\_file\_name, ios::binary | ios::in);

Product temp;

int count = 0;

while (fin.read((char\*)&temp, sizeof(Product))) {

count++;

}

fin.close();

if (count > index) {

return true;

}

else {

return false;

}

}

Product GetRecordFromBin(string bin\_file\_name, int index) {

ifstream fin;

fin.open(bin\_file\_name, ios::binary | ios::in);

Product result{};

if (fin.is\_open()) {

if (!CheckIndex(bin\_file\_name, index)) return result;

Product dict;

fin.seekg(index \* sizeof(Product), ios::beg);

fin.read((char\*)&dict, sizeof(Product));

result = dict;

}

return result;

fin.close();

}

void AddRecordToBin(string bin\_file\_name, Product product) {

fstream fout;

fout.open(bin\_file\_name, ios::binary | ios::in | ios::out);

if (fout.is\_open()) {

fout.seekg(0, ios::end);

fout.write((char\*)&product, sizeof(Product));

}

else {

cout << "Файла с таким именем не существует" << endl;

}

}

void OutputBinFile(string bin\_file\_name) {

ifstream fin(bin\_file\_name, ios::in | ios::binary);

Product product;

if (fin.is\_open()) {

while (fin.read((char\*)&product, sizeof(Product))) {

cout << product.name << " " << product.code << " " << product.manufacturer << " " << product.price << " " << product.country << "\n";

}

}

else {

cout << "Файла с таким именем не существует" << endl;

}

fin.close();

}

Product DirectAccess(string bin\_file\_name, int number) {

ifstream binFile(bin\_file\_name, ios::binary | std::ios::in);

if (!binFile.is\_open()) {

cout << "Файла с таким именем не существует" << endl;

}

binFile.seekg(0, ios::end);

long long size = binFile.tellg();

binFile.seekg(0, std::ios::beg);

if ((PRODUCT\_SIZE \* sizeof(char)) \* number > size) {

binFile.close();

return {};

}

binFile.seekg((PRODUCT\_SIZE)\*number);

Product product;

binFile.read((char\*)&product, sizeof(product));

binFile.close();

return product;

}

bool DeleteByKey(string bin\_file\_name, int key) {

ifstream bitFile(bin\_file\_name, ios::binary | ios::in);

if (!bitFile.is\_open()) {

cout << "Файла с таким именем не существует" << endl;

}

ofstream tempFile("temp.bin", ios::out | ios::binary);

if (!tempFile.is\_open()) {

bitFile.close();

cout << "Файла с таким именем не существует" << endl;

}

bitFile.seekg(0, ios::end);

int size = bitFile.tellg();

bitFile.seekg(0, ios::beg);

string s;

Product product;

while (bitFile.read((char\*)&product, sizeof(product))) {

if (product.code == key) {

continue;

}

tempFile.write((char\*)&product, sizeof(product));

}

bitFile.close();

tempFile.close();

remove(bin\_file\_name.c\_str());

if (rename("temp.bin", bin\_file\_name.c\_str()) == 0) {

return true;

}

else {

return false;

}

}

void testBinF() {

cout << "Создание файла..." << endl;

string bin\_file\_name = "test.bin";

ConvertTextToBin(bin\_file\_name, "file.txt");

OutputBinFile(bin\_file\_name);

cout << "-----------------------" << endl;

cout << "Добавление записи..." << endl;

Product product = { "orange", "orange\_factory", "egypt", 1, 343434 };

AddRecordToBin(bin\_file\_name, product);

OutputBinFile(bin\_file\_name);

cout << "-----------------------" << endl;

cout << "Удаление записи..." << endl;

DeleteByKey(bin\_file\_name, 343434);

OutputBinFile(bin\_file\_name);

cout << "-----------------------" << endl;

cout << "Прямой доступ к 0 записи..." << endl;

printRecord(DirectAccess(bin\_file\_name, 0));

cout << "-----------------------" << endl;

}

Файл hashtable.h

#ifndef HASHTABLE\_H

#define HASHTABLE\_H

#include "Product.h"

#include <iostream>

using namespace std;

struct Node {

int key;

int value;

Node\* next;

};

class HashTable {

protected:

Node\*\* table;

int currentSize = 0;

public:

HashTable();

~HashTable();

int hash(int key);

void add(int key, int value, bool isRehash = false);

void remove(int key);

int get(int key);

void rehash();

void print();

int size = 100;

};

void testHeshT();

#endif

Файл hashtable.cpp

#include "hashtable.h"

HashTable::HashTable() {

this->size = size;

table = new Node \* [size];

for (int i = 0; i < size; i++) {

table[i] = nullptr;

}

}

HashTable::~HashTable() {

for (int i = 0; i < size; i++) {

Node\* node = table[i];

while (node != nullptr) {

Node\* temp = node;

node = node->next;

delete temp;

}

}

delete[] table;

}

int HashTable::hash(int key) {

return key % 17;

}

void HashTable::add(int key, int value, bool isRehash) {

currentSize++;

if (currentSize / float(size) > 0.9 or isRehash) {

rehash();

}

int index = hash(key);

Node\* node = table[index];

table[index] = new Node;

table[index]->key = key;

table[index]->value = value;

table[index]->next = node;

}

void HashTable::remove(int key) {

int index = hash(key);

Node\* node = table[index];

Node\* prev = nullptr;

while (node != nullptr) {

if (node->key == key) {

if (prev == nullptr) {

table[index] = node->next;

}

else {

prev->next = node->next;

}

delete node;

return;

}

prev = node;

node = node->next;

}

}

int HashTable::get(int key) {

int index = hash(key);

Node\* node = table[index];

while (node != nullptr) {

if (node->key == key) {

return node->value;

}

node = node->next;

}

return -1;

}

void HashTable::rehash() {

int oldSize = size;

size \*= 2;

Node\*\* oldTable = table;

table = new Node \* [size];

for (int i = 0; i < size; i++) {

table[i] = nullptr;

}

for (int i = 0; i < oldSize; i++) {

Node\* node = oldTable[i];

while (node != nullptr) {

add(node->key, node->value);

node = node->next;

}

}

for (int i = 0; i < oldSize; i++) {

Node\* node = oldTable[i];

while (node != nullptr) {

Node\* temp = node;

node = node->next;

delete temp;

}

}

delete[] oldTable;

}

void HashTable::print() {

for (int i = 0; i < size; i++) {

Node\* node = table[i];

while (node != nullptr) {

cout << "[" << i << "]" << " | " << node->key << " " << node->value << endl;

node = node->next;

}

}

}

void testHeshT() {

cout << "Создание хэш-таблицы..." << endl;

HashTable hashTable;

hashTable.add(0, 1);

hashTable.add(1, 1);

hashTable.add(1, 2);

hashTable.add(1, 3);

hashTable.add(2, 4);

hashTable.add(3, 5);

hashTable.print();

cout << "-----------------------" << endl;

cout << "Удаление элемента..." << endl;

hashTable.remove(1);

hashTable.print();

cout << "-----------------------" << endl;

cout << "Получение элемента с ключом 1..." << endl;

cout<<hashTable.get(1)<<endl;

cout << "-----------------------" << endl;

cout << "Добавление элемента..." << endl;

hashTable.add(1, 1000);

hashTable.print();

cout << "-----------------------" << endl;

cout <<"Тест хэш-функции..." << endl;

cout << hashTable.hash(1) << endl;

cout << hashTable.hash(2) << endl;

cout << "-----------------------" << endl;

cout <<"Рехэширование...." << endl;

cout << "rehash...\nsize before = "<< hashTable.size << endl;

hashTable.rehash();

cout << "size after = " << hashTable.size << endl;

cout << "-----------------------" << endl;

cout <<"Добавление и рехэшивание..." << endl;

hashTable.add(77, 1, true);

hashTable.print();

cout << "after rehash add size = " << hashTable.size << endl;

cout << "-----------------------" << endl;

}

Файл hashfile.h

#pragma once

#ifndef HASHFILE\_H

#define HASHFILE\_H

#include "Product.h"

#include "hashtable.h"

#include "binfile.h"

#include <string>

using namespace std;

void InsertBinHash(HashTable& hashTable, string bin\_file\_name, int number = -1);

void DeleteByKeyHash(HashTable& hashTable, string bin\_file\_name, int key);

void GetByKeyHash(HashTable& hashTable, string bin\_file\_name, int key);

#endif // !1

Файл hashfile.cpp

#include "hashfile.h"

void InsertBinHash(HashTable& hashTable, string bin\_file\_name, int number) {

fstream bitFile(bin\_file\_name, ios::binary | ios::in | ios::app);

if (!bitFile.is\_open()) {

cout << "Error opening file";

return;

}

int cnt = 0;

if (number == -1) {

while (!bitFile.eof()) {

Product product;

bitFile.read((char\*)&product, sizeof(Product));

hashTable.add(product.code, cnt);

cnt++;

}

}

else {

int temp;

temp = DirectAccess(bin\_file\_name, number).code;

hashTable.add(temp, number);

}

}

void DeleteByKeyHash(HashTable& hashTable, string bin\_file\_name, int key) {

int number = hashTable.get(key);

if (number == -1) {

cout << "Ошибка" << endl;

return;

}

hashTable.remove(key);

DeleteByKey(bin\_file\_name, key);

}

void GetByKeyHash(HashTable& hashTable, string bin\_file\_name, int key) {

int number = hashTable.get(key);

if (number == -1) {

cout << "Ошибка" << endl;

return;

}

printRecord(DirectAccess(bin\_file\_name, number));

}