|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | |  |  |  | | 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-10-21 | Смольников А.Б. |
| Принял преподаватель | Филатов А.С. |

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

Москва 2022

# **Цель работы**

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

# **Постановка задачи**

* 1. **Задача 1**

1. Разработать приложение, которое использует хеш-таблицу для организации прямого доступа к записям двоичного файла, реализованного в практической работе 2.
2. 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
   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.
   3. После тестирования всех операций, создать в заголовочном файле функцию с именем testHeshT переместить в нее содержание функции main, проверить, что приложение выполняется.
   4. Разработать операции по управлению хеш-таблицей.
   5. Разработать хеш-функцию (метод определить самостоятельно), выполнить ее тестирование, убедиться, что хеш (индекс элемента таблицы) формируется верно.
   6. Разработать операции: вставить ключ в таблицу, удалить ключ из таблицы, найти ключ в таблице, рехешировать таблицу. Каждую операцию тестируйте по мере ее реализации.
   7. Подготовить тесты (последовательность значений ключей), обеспечивающие:
      1. вставку ключа без коллизии
      2. вставку ключа и разрешение коллизии
      3. вставку ключа с последующим рехешированием
      4. удаление ключа из таблицы
      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
   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
   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.
   2. Структура записи двоичного файла и все операции по управлению файлом должны быть размещены в соответствующем заголовочном файле.
   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
   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:
   5. Заполните файл небольшим количеством записей.
   6. Включите в файл записи как не приводящие к коллизиям, так и приводящие.
   7. Обеспечьте включение в файл такого количества записей, чтобы потребовалось рехеширование.
   8. Заполните файл большим количеством записей (до 1 000 000).
   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.
10. Ответы на вопросы
    1. Расскажите о назначении хеш-фунции.
    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?

Таблица 1. Задания варианта №22

|  |  |
| --- | --- |
| Задание 1 | Цепное хеширование  Расписание занятий группы: номер группы, название дисциплины, номер пары, номер недели, номер дня недели, вид занятия, номер аудитории. |

# **Решение**

* 1. **Теоретическое введение**

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.

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, метод открытой адресации: линейное и квадратичное зондирование.

Суть метода цепочек проста: 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.

Стоит отметить и 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.

Ответы на вопросы:

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, В 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.

Что определяет коэффициент нагрузки в хеш-таблице? Коэффициентом нагрузки (load factor) называется такое число, показывающее число хранимых элементов в массиве, разделенное на размер самого массива. Используется для отслеживания необходимости расширения и рехеширования.

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.

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

* 1. **Функции задания №1**

Функции управления бинарным файлом заимствованы из практической работы №2 без изменений.

Для управления таблицей создадим структуру Node:

* Поле key – строка, являющаяся 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.

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, описанные ниже.

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::add() добавляет запись в направленный список хеш-таблицы, перезаписывая первый элемент в хеш-таблице с необходимым индексом. Операция выполняется за постоянное время, так как не содержит циклов:

|  |
| --- |
| void HashTable::add(std::string 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(std::string 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(std::string 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() {  // std::cout<<"Rehashing..."<<std::endl;  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) {  std::cout<< i<< " | " << node->key << " " << node->value << std::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:

|  |
| --- |
| int testHeshT(){  HashTable hashTable;  hashTable.add("keyNoCol", 1);  hashTable.add("key1", 1);  hashTable.add("key1", 2);  hashTable.add("key1", 3);  hashTable.add("key2", 4);  hashTable.add("key3", 5);  //remove test  hashTable.print();  hashTable.remove("key1");  std::cout<<std::endl;  hashTable.print();  //hash test  std::cout<<hashTable.hash("key1")<<std::endl;  std::cout<<hashTable.hash("key2")<<std::endl;  //rehash test  std::cout<<"rehash\nsize before"<<std::endl;  std::cout<<hashTable.size<<std::endl;  hashTable.rehash();  std::cout<<"size after"<<std::endl;  std::cout<<hashTable.size<<std::endl;  //add with rehash test  hashTable.add("keyRehash", 1, true);  hashTable.print();  std::cout<<"after rehash add "<<hashTable.size<<std::endl;  return 0;  } |

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

|  |
| --- |
| int testBinF(){  std::string bitFileName = "schedule.bin";  convertToBitFile("schedule.txt", bitFileName);  printBitFile(bitFileName);  addNoteToFile(bitFileName, "\n005 ИКБО-04-21 пр4 2 1 4 сем 210");  printBitFile(bitFileName);  deleteByKey(bitFileName, "005");  printBitFile(bitFileName);  directAccess(bitFileName, 2);  return 0;  } |

Функция migrateBinHash() 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 migrateBinHash(HashTable &hashTable, std::string bitFileName,int number) {  std::fstream bitFile(bitFileName, std::ios::binary| std::ios::in | std::ios::app);  if(!bitFile.is\_open()){  std::cout<<"Error opening file";  return;  }  int cnt=0;  if(number==-1){  while(!bitFile.eof()){  std::string temp;  std::getline(bitFile,temp);  hashTable.add(temp.substr(0,temp.find(' ')), cnt);  cnt++;  }  }else{  std::string temp;  temp = directAccess(bitFileName, number);  if(temp.length()==0){  std::cout<<"Error";  return;  }  hashTable.add(temp.substr(0,temp.find(' ')), number);  }  } |

Функция deleteByKeyHash() принимет 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, std::string bitFileName,std::string key) {  int number = hashTable.get(key);  if(number==-1){  std::cout<<"Error";  return;  }  hashTable.remove(key);  deleteByKey(bitFileName, 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, std::string bitFileName,std::string key) {  int number = hashTable.get(key);  if(number==-1){  std::cout<<"Error";  return;  }  directAccess(bitFileName, number);  } |

* 1. **Интерфейс**

При запуске программы пользователю предоставляется выбор одного Предложенных заданий, указанных в постановке задачи. Любой другой введенный символ спровоцирует завершение программы.

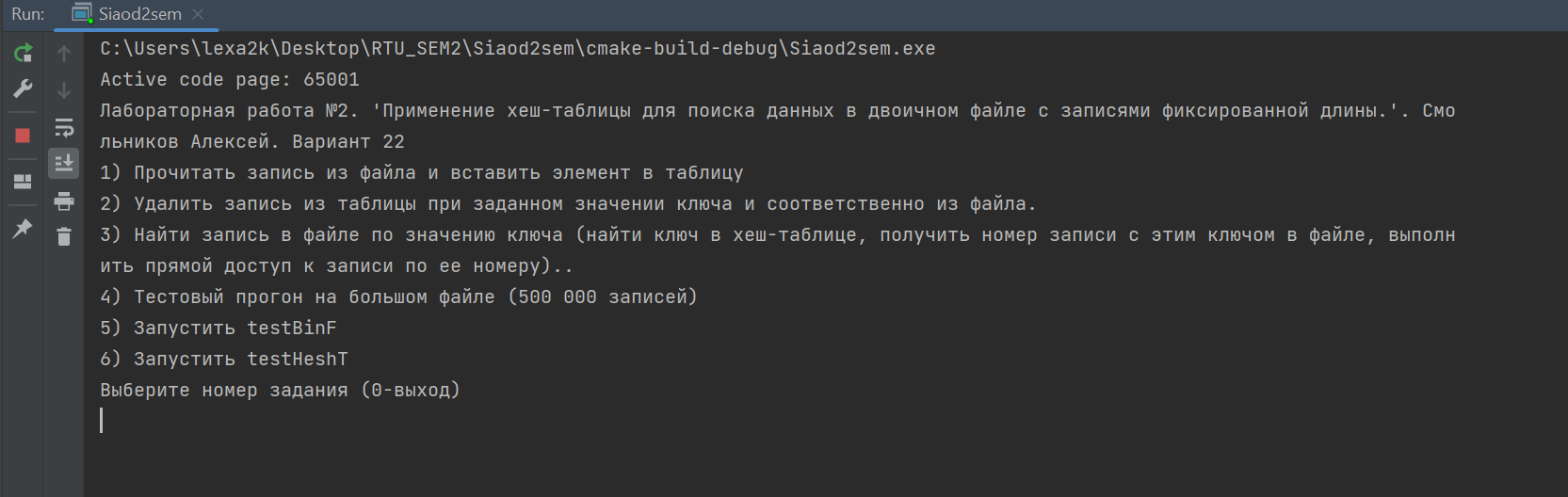


Рисунок 1. Интерфейс программы

# **Тестирование**

В силу простоты вычислений совместим тестовый прогон с рабочим, проверив каждое задание по одному разу.

Рассмотрим структуру бинарного файла, используемого для тестирования программы (длина – 38 символов):

|  |
| --- |
| 00591551 ИКБО-10-21 Пр1 2 1 7 сем 201  00662943 ИКБО-01-21 Пр2 2 1 7 сем 201  00814653 ИКБО-02-21 Пр3 2 1 7 сем 203  00183875 ИКБО-03-21 пр4 2 1 4 сем 203 |
|  |

Функция migrateBinHash() заносит в хеш-таблицу необходимую запись из переданного файла, из рисунка 1 видно, что функция корректно получает запись по номеру в файле:

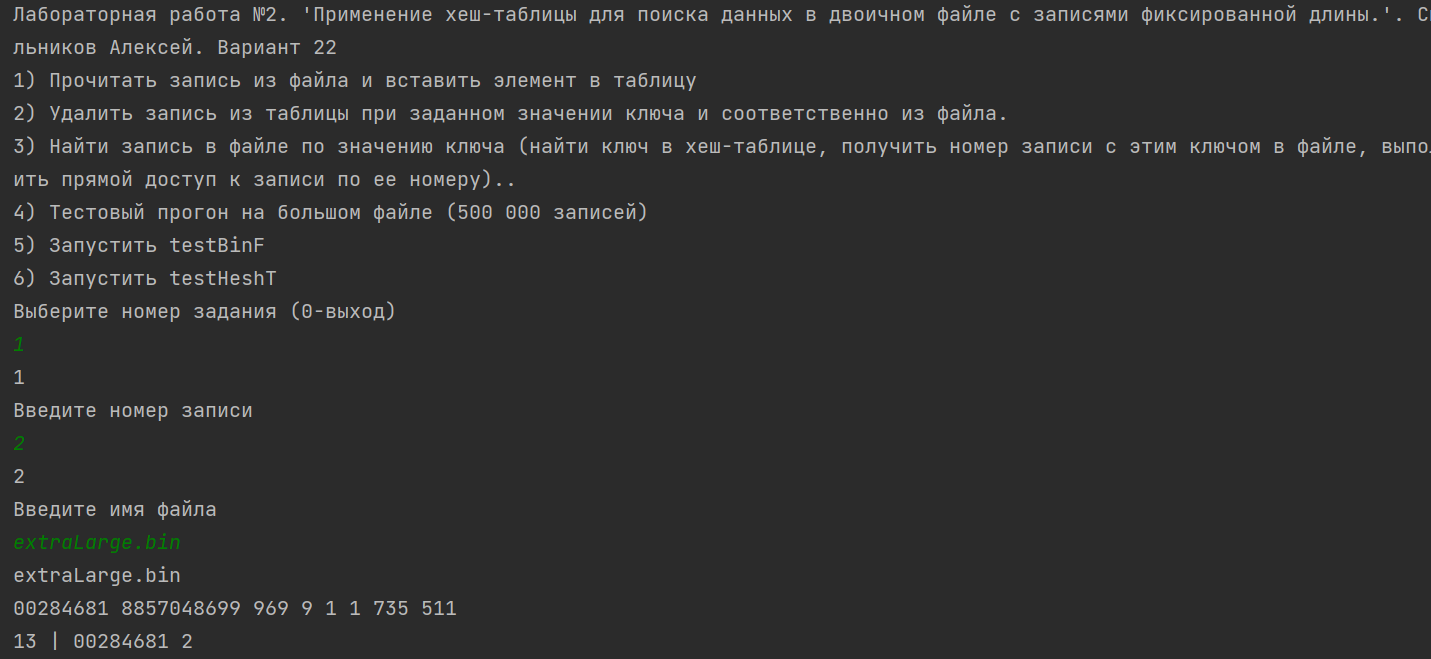


Рисунок 1. Случайно сгенерированный файл

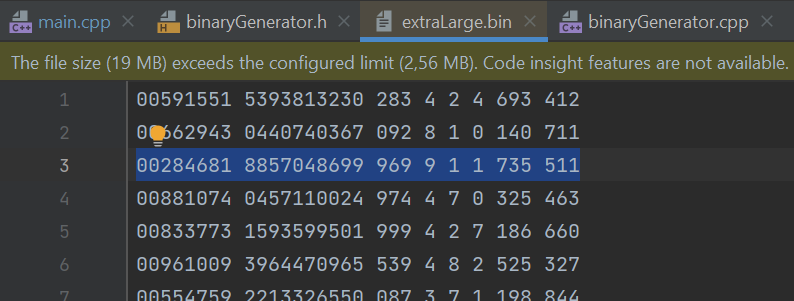


Рисунок 2. Третья строка в используемом файле

В задании 2 выполняется удаление записи по ключу. На вход передадим ключ записи с индексом 2 (строка 3). После удаления, сравнивая данные на рисунках 2 и 4, можно удостовериться, что удаление выполняется корректно:

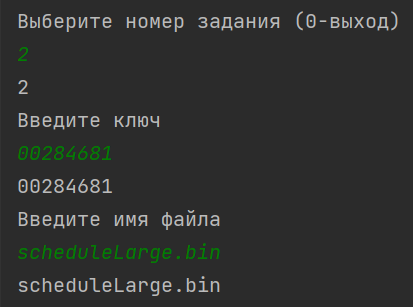


Рисунок 3. Вывод текстового файла

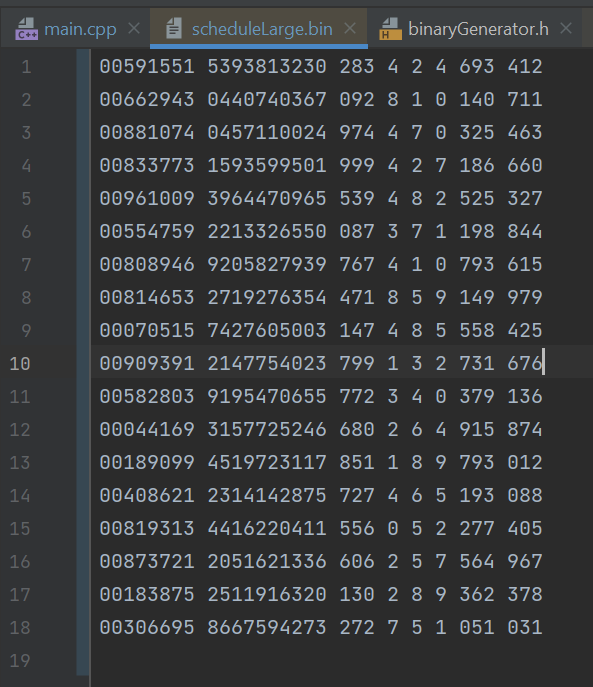


Рисунок 4. Файл после удаления строки

Функция appendLine() добавляет строчку в файл, на рисунке 4 видно, что добавление новой строки происходит корректно:

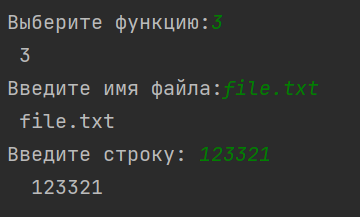


Рисунок 3. Ввод строки для добавления

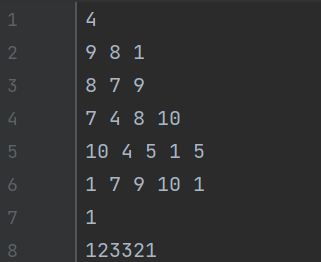


Рисунок 4. Файл после добавления строки

Функция getByKeyHash() считывает из хеш-таблицы запись по введенному ключу и выводит ее в консоль. Из рисунков 5, 6 видно, что найденная запись соответствует файлу

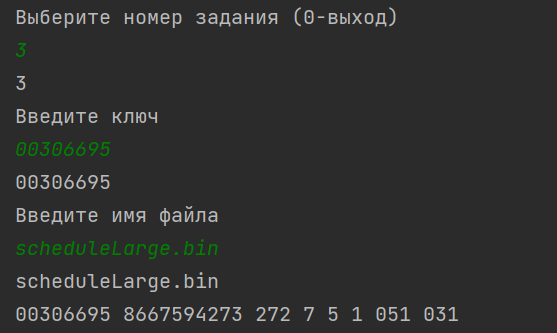


Рисунок 5. Результат работы функции

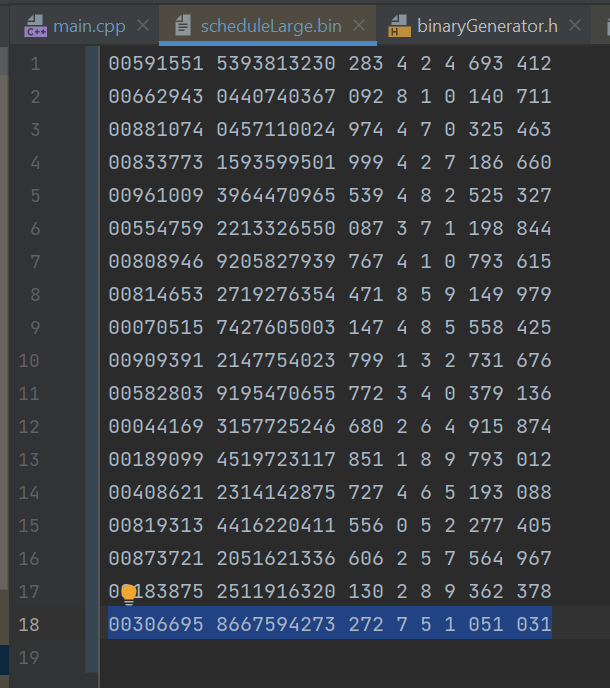


Рисунок 6. Файл, в котором производился поиск

В задании 4 проведем замер времени выполнения функций поиска в хеш-таблице на 500000 записей использую встроенную библиотеку chrono. Выполним поиск первой записи, последней записи и записи, расположенной в середине файла. Из рисунка 7 видно, что время поиска пренебрежительно мало отличается, от 220 микросекунд для последней записи до 1538 микросекунд для первой записи. Заметно, что получение порядкового номера строки из хэш-таблицы значительно увеличивает скорость работы программы, поиск выполняется за постоянное время:

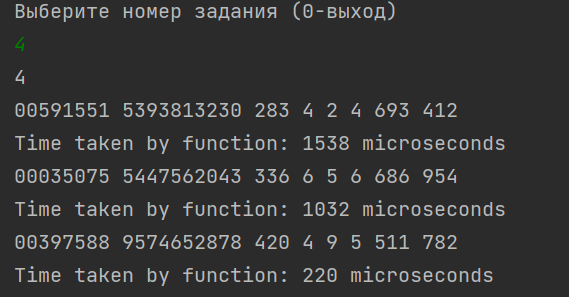


Рисунок 7. Время выполнения функций

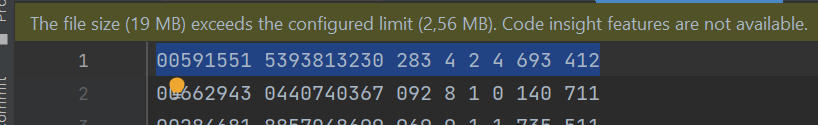


Рисунок 8. Первая запись

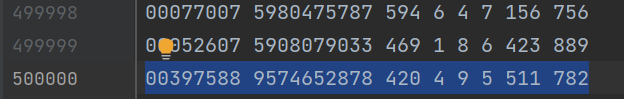


Рисунок 9. Последняя запись

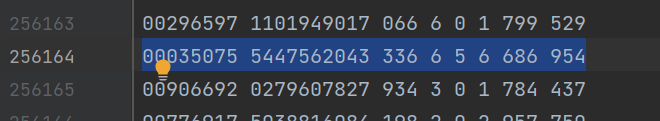


Рисунок 10. Запись около середины

Функция testiBinF() производит автоматизированное тестирование функций бинарного файла, добавляя записи в файл, а затем удаляя их, сопровождая выводом файла после каждой операции. Из рисунков 8 и 9 видно, что все операции выполняются корректно:

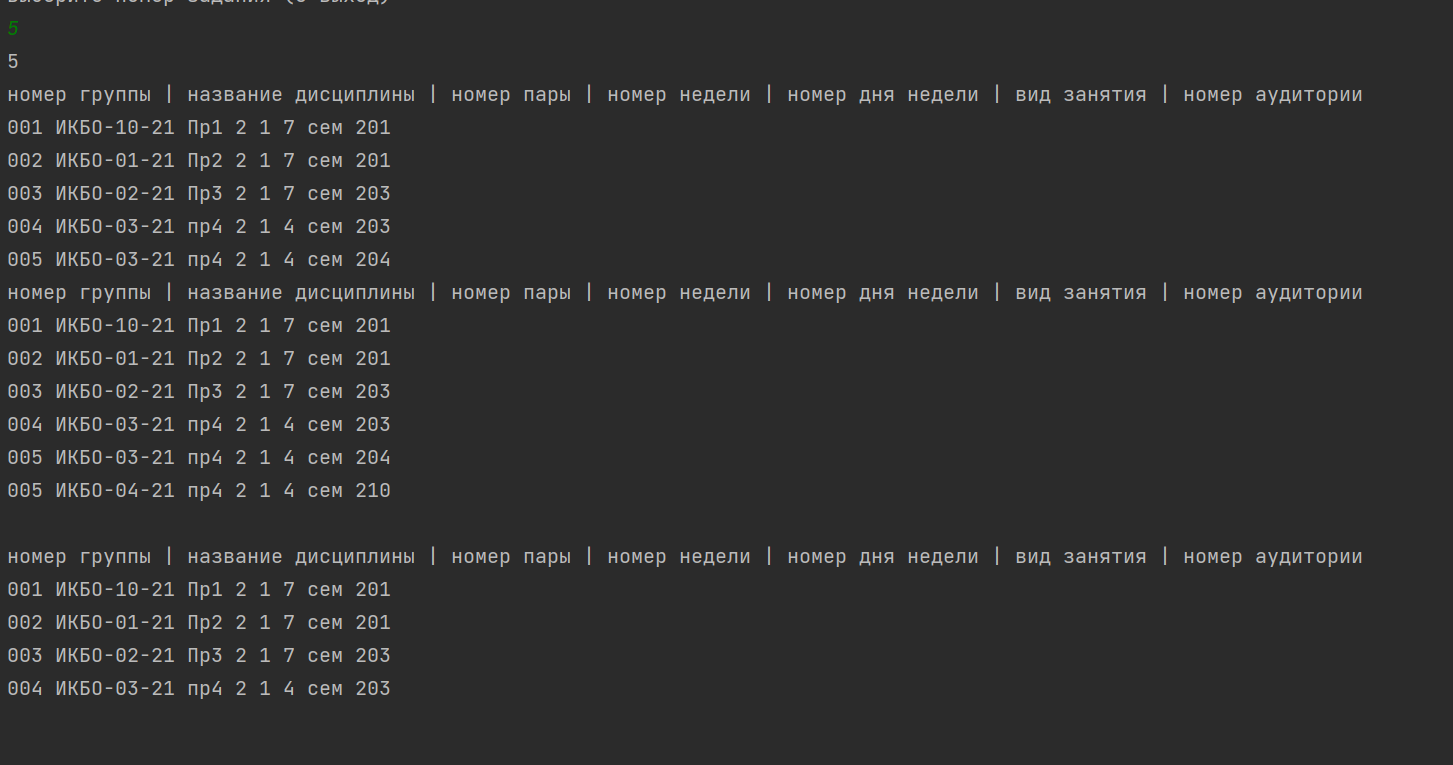


Рисунок 8. Результат работы функции

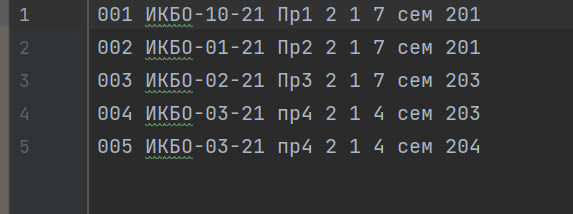


Рисунок 9. Текстовый файл

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 (рис.12), а также вывод всех элементов таблицы. Из рисунка 11 видно, что все функции работают корректно, функционал хеш-таблицы полный:

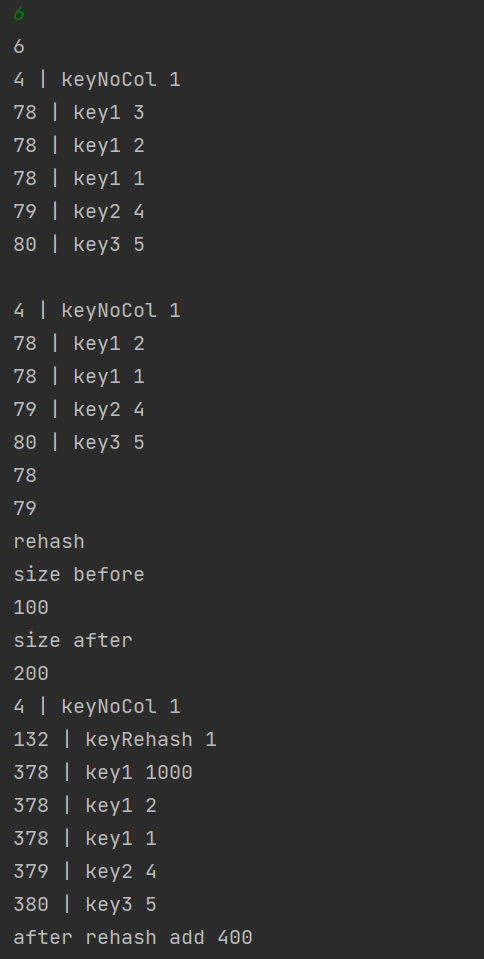


Рисунок 11. Интерфейс функции

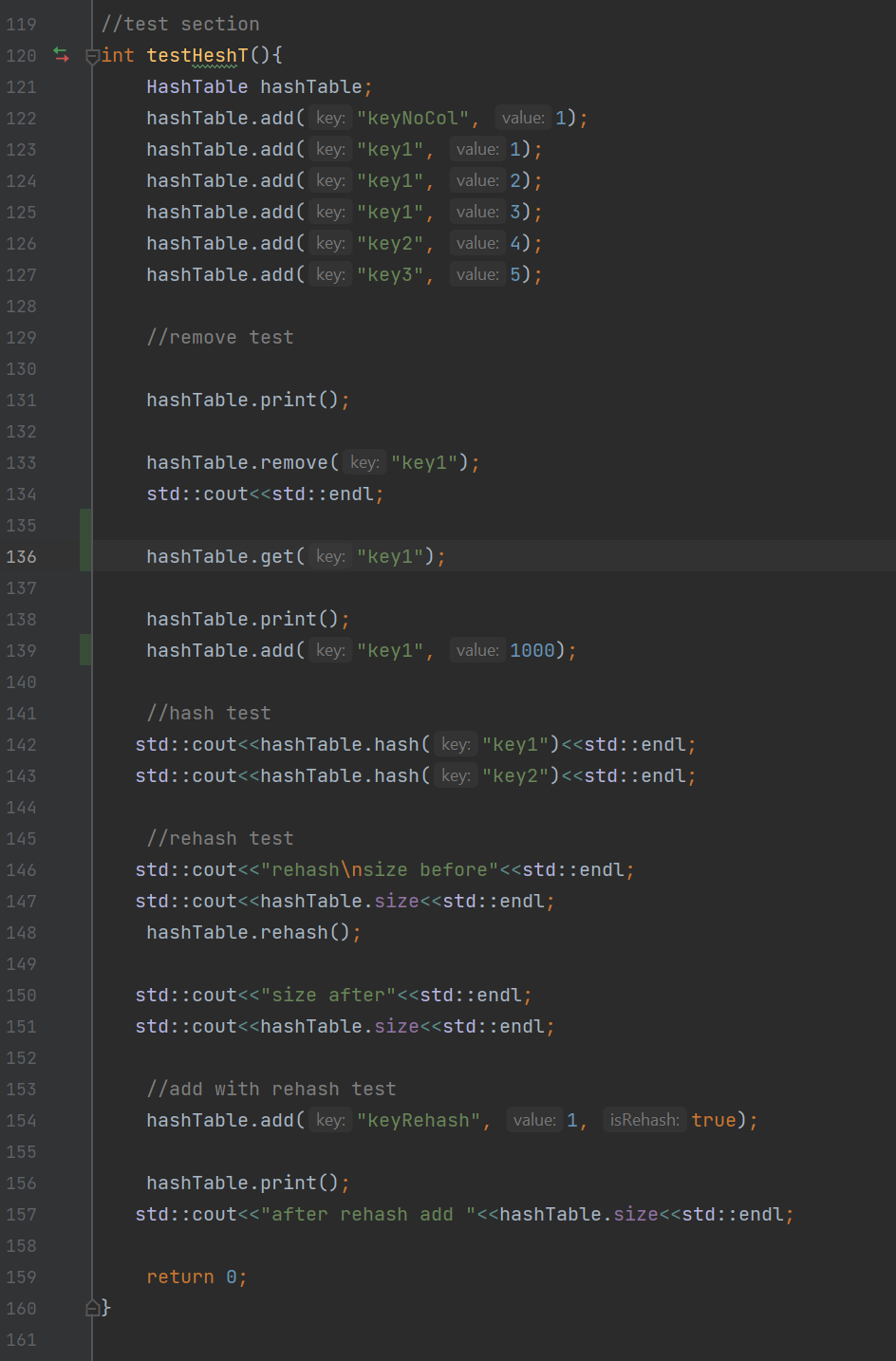


Рисунок 12. Код функции тестирования

# **Выводы**

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

# **Исходный код программы**

main.cpp

|  |
| --- |
|  |
| //  // Created by lexa2k on 26.09.2022.  //  #include <iostream>  #include "manipulateHash.h"  #include "bitFile.h"  #include "manipulateBinHash.h"  #include "binaryGenerator.h"  #include <chrono>  using namespace std;  int main() {  system("chcp 65001");  //create switch-case menu for 5 elements where 0 calls exit  int choice=1;  unsigned int inp;  HashTable hashTable;  while(choice!=0){  cout<<"Лабораторная работа №2. 'Применение хеш-таблицы для поиска данных в двоичном файле с записями фиксированной длины.'. Смольников Алексей. Вариант 22"<<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 (500 000 записей)"<<endl;  cout<<"5) Запустить testBinF"<<endl;  cout<<"6) Запустить testHeshT"<<endl;  cout<<"Выберите номер задания (0-выход)"<<endl;  cin>>choice;  switch (choice) {  case 1: {  cout << "Введите номер записи" << endl;  cin >> inp;  cout << "Введите имя файла" << endl;  string file;  cin >> file;  migrateBinHash(hashTable, file, inp);  hashTable.print();  }  break;  case 2:{  string key;  cout << "Введите ключ" << endl;  cin >> key;  cout << "Введите имя файла" << endl;  string file;  cin >> file;  deleteByKeyHash(hashTable, file,key);  break;  }  case 3:{  string sinp;  cout<<"Введите ключ"<<endl;  cin>>sinp;  cout<<"Введите имя файла"<<endl;  string file;  cin>>file;  getByKeyHash(hashTable, file,sinp);  break;  }  case 4:{  HashTable hashTable1;  string bitFileName = "extraLarge.bin";  migrateBinHash(hashTable1, bitFileName);  //measure time  auto start = chrono::high\_resolution\_clock::now();  getByKeyHash(hashTable1, bitFileName, "00591551");  auto end = chrono::high\_resolution\_clock::now();  auto duration = chrono::duration\_cast<chrono::microseconds>(end - start);  cout << "Time taken by function: "  << duration.count() << " microseconds" << endl;  start = chrono::high\_resolution\_clock::now();  getByKeyHash(hashTable1, bitFileName, "00035075");  end = chrono::high\_resolution\_clock::now();  duration = chrono::duration\_cast<chrono::microseconds>(end - start);  cout << "Time taken by function: "  << duration.count() << " microseconds" << endl;  start = chrono::high\_resolution\_clock::now();  getByKeyHash(hashTable1, bitFileName, "00397588");  end = chrono::high\_resolution\_clock::now();  duration = chrono::duration\_cast<chrono::microseconds>(end - start);  cout << "Time taken by function: "  << duration.count() << " microseconds" << endl;  break;  }  case 5:  testBinF();  break;  case 6: {  testHeshT();  break;  }  case 7:  hashTable.print();  break;  default:  break;  }  }  return 0;  }  int main1() {  system("chcp 65001");  // generateBinaryFile("extraLarge.bin", 500000);  HashTable hashTable;  string bitFileName = "extraLarge.bin";  migrateBinHash(hashTable, bitFileName);  getByKeyHash(hashTable, bitFileName, "00284681");  //measure time  auto start = chrono::high\_resolution\_clock::now();  getByKeyHash(hashTable, bitFileName, "00591551");  auto end = chrono::high\_resolution\_clock::now();  auto duration = chrono::duration\_cast<chrono::microseconds>(end - start);  cout << "Time taken by function: "  << duration.count() << " microseconds" << endl;  start = chrono::high\_resolution\_clock::now();  getByKeyHash(hashTable, bitFileName, "00035075");  end = chrono::high\_resolution\_clock::now();  duration = chrono::duration\_cast<chrono::microseconds>(end - start);  cout << "Time taken by function: "  << duration.count() << " microseconds" << endl;  start = chrono::high\_resolution\_clock::now();  getByKeyHash(hashTable, bitFileName, "00397588");  end = chrono::high\_resolution\_clock::now();  duration = chrono::duration\_cast<chrono::microseconds>(end - start);  cout << "Time taken by function: "  << duration.count() << " microseconds" << endl;  return 0;  } }  case 2: {  cout<<"Расписание занятий группы: номер группы, название дисциплины, номер пары, номер недели, номер дня недели, вид занятия, номер аудитории."<<endl;  cout<<"1 преобразование тестовых данных из текстового файла в двоичный файл;"<<endl;  cout<<"2 преобразование данных из двоичного файла в текстовый;"<<endl;  cout<<"3 вывод всех записей двоичного файла;"<<endl;  cout<<"4 доступ к записи по ее порядковому номеру в файле, используя механизм прямого доступа к записи в двоичном файле;"<<endl;  cout<<"5 удаление записи с заданным значением ключа, выполнить путем замены на последнюю запись."<<endl;  cout<<"6 Сформировать в двоичном файле расписание заданной группы на заданный день недели."<<endl;  cout<<"7 Обновить расписание, найти дисциплины, которые стоят на одинаковых парах в одной аудитории, и определить для них новые аудитории.."<<endl;  cout<<"Выберите функцию: ";  int func;  cin>>func;  switch (func) {  case 1:{  cout<<"Введите имя файла: ";  string fileName;  cin>>fileName;  cout<<"Введите имя нового файла: ";  string newFileName;  cin>>newFileName;  convertToBitFile(fileName, newFileName);  break;  }  case 2:{  cout<<"Введите имя файла: ";  string fileName;  cin>>fileName;  cout<<"Введите имя нового файла: ";  string newFileName;  cin>>newFileName;  convertToTextFile(fileName, newFileName);  break;  }  case 3:{  cout<<"Введите имя файла: ";  string fileName;  cin>>fileName;  printBitFile(fileName);  break;  }  case 4:{  cout<<"Введите имя файла: ";  string fileName;  cin>>fileName;  cout<<"Введите номер записи: ";  int num;  cin>>num;  directAccess(fileName, num);  break;  }  case 5:{  cout<<"Введите имя файла: ";  string fileName;  cin>>fileName;  cout<<"Введите ключ: ";  string key;  cin>>key;  deleteByKey(fileName, key);  break;  }  case 6:{  cout<<"Введите имя файла: ";  string fileName;  cin>>fileName;  cout<<"Введите номер группы: ";  string group;  cin>>group;  cout<<"Введите номер дня недели: ";  string day;  cin>>day;  generateByWeekDay(fileName, group, day);  break;  }  case 7:{  cout<<"Введите имя файла: ";  string fileName;  cin>>fileName;  cout<<"Введите имя нового файла: ";  string newFileName;  cin>>newFileName;  updateSchedule(fileName, newFileName);  break;  }  default:{  cout<<"Неверный ввод"<<endl;  break;  }  }  }  }  }  return 0;  } |

bitFile.h

|  |
| --- |
| //  // Created by lexa2k on 17.09.2022.  //  #ifndef SIAOD2SEM\_BITFILE\_H  #define SIAOD2SEM\_BITFILE\_H  #include <string>  //  //struct Record {  // std::string group;  // std::string disciplineName;  // std::string pairNumber;  // std::string weekNumber;  // std::string dayNumber;  // std::string lessonType;  // std::string roomNumber;  //  // void cutRecord();  // void printRecord();  // std::string createNote();  //};  struct customVector{  int size;  std::string\* data;  int capacity;  customVector();  void push\_back(std::string s);  void print();  ~customVector();  };  bool convertToBitFile(std::string fileName, std::string bitFileName);  bool convertToBitFile(std::string fileName, std::string bitFileName);  bool convertToTextFile(std::string bitFileName, std::string fileName);  bool printBitFile(std::string bitFileName);  std::string directAccess(std::string bitFileName, int number, int length=38);  bool deleteByKey(std::string bitFileName, std::string key, int length=32);  void generateByWeekDay(std::string bitFileName, std::string group, std::string day);  std::string\* splitter (std::string s, std::string delim, int size);  //Обновить расписание, найти дисциплины, которые стоят на одинаковых парах в одной аудитории, и определить для них новые аудитории.  void updateSchedule(std::string bitFileName, std::string newFileName);  bool addNoteToFile(std::string bitFileName, std::string note, int length=32);  int testBinF();  #endif SIAOD2SEM\_BITFILE\_H |

bitFile.h

|  |
| --- |
| //  // Created by lexa2k on 17.09.2022.  //  #ifndef SIAOD2SEM\_BITFILE\_H  #define SIAOD2SEM\_BITFILE\_H  #include <string>  //  //struct Record {  // std::string group;  // std::string disciplineName;  // std::string pairNumber;  // std::string weekNumber;  // std::string dayNumber;  // std::string lessonType;  // std::string roomNumber;  //  // void cutRecord();  // void printRecord();  // std::string createNote();  //};  struct customVector{  int size;  std::string\* data;  int capacity;  customVector();  void push\_back(std::string s);  void print();  ~customVector();  };  bool convertToBitFile(std::string fileName, std::string bitFileName);  bool convertToBitFile(std::string fileName, std::string bitFileName);  bool convertToTextFile(std::string bitFileName, std::string fileName);  bool printBitFile(std::string bitFileName);  std::string directAccess(std::string bitFileName, int number, int length=38);  bool deleteByKey(std::string bitFileName, std::string key, int length=32);  void generateByWeekDay(std::string bitFileName, std::string group, std::string day);  std::string\* splitter (std::string s, std::string delim, int size);  //Обновить расписание, найти дисциплины, которые стоят на одинаковых парах в одной аудитории, и определить для них новые аудитории.  void updateSchedule(std::string bitFileName, std::string newFileName);  bool addNoteToFile(std::string bitFileName, std::string note, int length=32);  int testBinF();  #endif SIAOD2SEM\_BITFILE\_H |

bitFile.cpp

|  |
| --- |
| //  // Created by lexa2k on 17.09.2022.  //Структура записи  //Расписание занятий группы: номер группы, название дисциплины, номер пары, номер недели, номер дня недели, вид занятия, номер аудитории.  //Доп. операция  //Сформировать в двоичном файле расписание заданной группы на заданный день недели.  //Обновить расписание, найти дисциплины, которые стоят на одинаковых парах в одной аудитории, и определить для них новые аудитории.  //001 ИКБО-10-21 Пр1 1 1 7 сем 223  //002 ИКБО-01-21 Пр2 2 1 7 сем 223  //003 ИКБО-02-21 Пр3 2 1 7 сем 220  //004 ИКБО-03-21 пр4 2 1 4 сем 220  //005 ИКБО-03-21 пр4 2 1 4 сем 220  #include "bitFile.h"  #include <fstream>  #include <iostream>  #include <string>  #include <iomanip>  #include <map>  //struct Record {  // std::string hash;  // std::string group;  // std::string disciplineName;  // std::string pairNumber;  // std::string weekNumber;  // std::string dayNumber;  // std::string lessonType;  // std::string roomNumber;  //  // void cutRecord(){  // this->hash = this->hash.substr(0);  // this->group = this->group.substr(0, this->group.find(','));  // this->disciplineName = this->disciplineName.substr(0, this->disciplineName.find(','));  // this->pairNumber = this->pairNumber.substr(0, this->pairNumber.find(','));  // this->weekNumber = this->weekNumber.substr(0, this->weekNumber.find(','));  // this->dayNumber = this->dayNumber.substr(0, this->dayNumber.find(','));  // this->lessonType = this->lessonType.substr(0, this->lessonType.find(','));  // this->roomNumber = this->roomNumber.substr(0, this->roomNumber.find(','));  // }  //  // void printRecord(){  // std::cout << this->group << " " << this->disciplineName << " " << this->pairNumber << " " << this->weekNumber << " " << this->dayNumber << " " << this->lessonType << " " << this->roomNumber << std::endl;  // }  //  // std::string createNote(){  // std::string note = this->group + " " + this->disciplineName + " " + this->pairNumber + " " + this->weekNumber + " " + this->dayNumber + " " + this->lessonType + " " + this->roomNumber + "\n";  // return note;  // }  //};  bool convertToBitFile(std::string fileName, std::string bitFileName){  std::ifstream file(fileName, std::ios::in);  if (!file.is\_open()){  throw std::runtime\_error("File not found");  }  std::ofstream bitFile(bitFileName, std::ios::binary | std::ios::out);  if (!bitFile.is\_open()){  throw std::runtime\_error("File not found");  }  bitFile << file.rdbuf();  bitFile.close();  file.close();  return true;  }  bool convertToTextFile(std::string bitFileName, std::string fileName){  std::ifstream bitFile(bitFileName, std::ios::binary | std::ios::in);  if (!bitFile.is\_open()){  throw std::runtime\_error("File not found");  }  std::ofstream file(fileName, std::ios::out);  if (!file.is\_open()){  throw std::runtime\_error("File not found");  }  file<<bitFile.rdbuf();  bitFile.close();  file.close();  return true;  }  bool printBitFile(std::string bitFileName){  std::ifstream bitFile(bitFileName, std::ios::binary | std::ios::in);  if (!bitFile.is\_open()){  throw std::runtime\_error("File not found");  }  std::cout<<"номер группы | название дисциплины | номер пары | номер недели | номер дня недели | вид занятия | номер аудитории"<<std::endl;  std::cout<<bitFile.rdbuf();  std::cout<<std::endl;  bitFile.close();  return true;  }  std::string directAccess(std::string bitFileName, int number, int length){  std::ifstream bitFile(bitFileName, std::ios::binary | std::ios::in);  if (!bitFile.is\_open()){  throw std::runtime\_error("File not found");  }  bitFile.seekg(0, std::ios::end);  long long size = bitFile.tellg();  bitFile.seekg(0, std::ios::beg);  //for 32 length  // if ((length\*sizeof(char)+10)\*number > size){  // bitFile.close();  // return "";  // }  if ((length\*sizeof(char))\*number > size){  bitFile.close();  return "";  }  bitFile.seekg((length)\*number);  std::string s;  std::getline(bitFile, s);  std::cout<<s<<std::endl;  bitFile.close();  return s;  }  bool deleteByKey(std::string bitFileName, std::string key, int length){  std::ifstream bitFile(bitFileName, std::ios::binary | std::ios::in);  if (!bitFile.is\_open()){  throw std::runtime\_error("File not found");  }  std::ofstream tempFile("temp.bin", std::ios::out | std::ios::binary);  if (!tempFile.is\_open()){  bitFile.close();  throw std::runtime\_error("File not found");  }  bitFile.seekg(0, std::ios::end);  int size = bitFile.tellg();  bitFile.seekg(0, std::ios::beg);  // bitFile.seekg(size - length-9);  //  // std::string temp;  //  // std::getline(bitFile, temp);  //  // std::string tempNum = temp.substr(0, temp.find(' '));  // temp = key+ temp.substr(key.size(), temp.size());  //  // bitFile.seekg(0, std::ios::beg);  std::string s;  while (!bitFile.eof()){  std::getline(bitFile, s);  if(s.substr(0, key.size()) != key){  tempFile<<s<<std::endl;  }  }  bitFile.close();  tempFile.close();  std::remove(bitFileName.c\_str());  if(std::rename("temp.bin", bitFileName.c\_str())==0){  return true;  } else{  return false;  }  }  std::string\* splitter (std::string s, std::string delim = " ", int size =8){  std::string\* result = new std::string[size];  int i = 0;  while (s.find(delim) != std::string::npos){  result[i] = s.substr(0, s.find(delim));  s = s.substr(s.find(delim)+1, s.size());  i++;  }  result[i] = s;  return result;  }  void generateByWeekDay(std::string bitFileName, std::string group, std::string day){  std::ifstream bitFile(bitFileName, std::ios::binary | std::ios::in);  if (!bitFile.is\_open()){  throw std::runtime\_error("File not found");  }  std::ofstream file("generated.bin", std::ios::out | std::ios::binary);  std::string s;  while (!bitFile.eof()){  std::getline(bitFile, s);  std::string\* temp = splitter(s);  if (temp[1] == group && temp[5] == day){  file<<s<<std::endl;  }  }  bitFile.close();  file.close();  }  customVector::customVector() {  data = new std::string[1];  size = 0;  capacity = 1;  }  void customVector::push\_back(std::string s){  if (size == capacity){  std::string\* temp = new std::string[capacity\*2];  for (int i = 0; i < size; i++){  temp[i] = data[i];  }  delete[] data;  data = temp;  capacity \*= 2;  }  data[size] = s;  size++;  }  void customVector::print(){  for (int i = 0; i < size; i++){  std::cout<<data[i]<<std::endl;  }  }  customVector::~customVector(){  delete[] data;  }  void updateSchedule(std::string bitFileName, std::string newFileName){  std::ifstream bitFile(bitFileName, std::ios::binary | std::ios::in);  if (!bitFile.is\_open()){  throw std::runtime\_error("File not found");  }  std::ofstream tempFile(newFileName, std::ios::out | std::ios::binary);  if (!tempFile.is\_open()){  bitFile.close();  throw std::runtime\_error("File not found");  }  customVector scheudele;  std::string s;  while (!bitFile.eof()){  std::getline(bitFile, s);  std::string\* temp = splitter(s);  // s = temp[0] + " " + temp[3] + " " + temp[4] + " " + temp[5] + " " + temp[7];  scheudele.push\_back(s);  }  scheudele.print();  for(int i=0;i<scheudele.size;i++){  std::string\* temp = splitter(scheudele.data[i]);  // for(int z=0;z<8;z++){  // tempFile<<"!"<<temp[z];  // }  // tempFile<<std::endl;  for(int j=i+1;j<scheudele.size;j++){  std::string\* temp2 = splitter(scheudele.data[j]);  if(temp[7]==temp2[7] && temp[3]==temp2[3] && temp[4]==temp2[4] && temp[5]==temp2[5]){  scheudele.data[i] = temp[0]+ " " +temp[1] + " " + temp[2] + " " + temp[3] + " " + temp[4] + " " + temp[5] + " " + temp[6] + " " + std::to\_string(200+ rand()%100);  }  }  }  for(int i=0;i<scheudele.size;i++){  tempFile<<scheudele.data[i]<<std::endl;  }  bitFile.close();  tempFile.close();  }  bool addNoteToFile(std::string bitFileName, std::string note, int length){  std::fstream bitFile(bitFileName, std::ios::binary| std::ios::out | std::ios::app);  if (!bitFile.is\_open()){  return false;  }  bitFile<<note<<std::endl;  bitFile.close();  return true;  }  int testBinF(){  std::string bitFileName = "schedule.bin";  convertToBitFile("schedule.txt", bitFileName);  printBitFile(bitFileName);  addNoteToFile(bitFileName, "\n005 ИКБО-04-21 пр4 2 1 4 сем 210");  printBitFile(bitFileName);  deleteByKey(bitFileName, "005");  printBitFile(bitFileName);  directAccess(bitFileName, 2);  return 0;  } |

manipulateHash.h

|  |
| --- |
| //  // Created by lexa2k on 26.09.2022.  //  #ifndef SIAOD2SEM\_MANIPULATEHASH\_H  #define SIAOD2SEM\_MANIPULATEHASH\_H  #include <iostream>  #include <string>  //Цепное хеширование  //Расписание занятий группы: номер группы,  //название дисциплины, номер пары, номер недели, номер дня недели, вид занятия, номер аудитории.  //структура - ключ-номер строки  //способ разрешения коллизий - цепное хеширование  struct Node {  std::string key;  int value;  Node \*next;  };  class HashTable {  protected:  Node \*\*table;  int currentSize=0;  public:  HashTable();  ~HashTable();  int hash(std::string key);  void add(std::string key, int value, bool isRehash = false);  void remove(std::string key);  int get(std::string key);  void rehash();  void print();  int size=100;  };  //test section  int testHeshT();  #endif //SIAOD2SEM\_MANIPULATEHASH\_H |

manipulateHash.cpp

|  |
| --- |
| //  // Created by lexa2k on 26.09.2022.  //  #include "manipulateHash.h"  #include <iostream>  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(std::string key) {  int hash = 0;  for (int i = 0; i < key.length(); i++) {  hash += static\_cast<int>(key[i]);  }  return hash % size;  }  void HashTable::add(std::string 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(std::string 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(std::string 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() {  // std::cout<<"Rehashing..."<<std::endl;  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) {  std::cout<< i<< " | " << node->key << " " << node->value << std::endl;  node = node->next;  }  }  }  //test section  int testHeshT(){  HashTable hashTable;  hashTable.add("keyNoCol", 1);  hashTable.add("key1", 1);  hashTable.add("key1", 2);  hashTable.add("key1", 3);  hashTable.add("key2", 4);  hashTable.add("key3", 5);  //remove test  hashTable.print();  hashTable.remove("key1");  std::cout<<std::endl;  hashTable.get("key1");  hashTable.print();  hashTable.add("key1", 1000);  //hash test  std::cout<<hashTable.hash("key1")<<std::endl;  std::cout<<hashTable.hash("key2")<<std::endl;  //rehash test  std::cout<<"rehash\nsize before"<<std::endl;  std::cout<<hashTable.size<<std::endl;  hashTable.rehash();  std::cout<<"size after"<<std::endl;  std::cout<<hashTable.size<<std::endl;  //add with rehash test  hashTable.add("keyRehash", 1, true);  hashTable.print();  std::cout<<"after rehash add "<<hashTable.size<<std::endl;  return 0;  } |

manipulateBinHash.h

|  |
| --- |
| //  // Created by lexa2k on 26.09.2022.  //  #ifndef SIAOD2SEM\_MANIPULATEBINHASH\_H  #define SIAOD2SEM\_MANIPULATEBINHASH\_H  #include "manipulateHash.h"  #include "bitFile.h"  void migrateBinHash(HashTable &hashTable, std::string bitFileName, int number=-1);  void deleteByKeyHash(HashTable &hashTable, std::string bitFileName, std::string key);  void getByKeyHash(HashTable &hashTable, std::string bitFileName, std::string key);  void insertNoteToHash(std::string file, int note, HashTable &hashTable);  #endif //SIAOD2SEM\_MANIPULATEBINHASH\_H |

manipulateBinHash.cpp

|  |
| --- |
| //  // Created by lexa2k on 26.09.2022.  //  #include "manipulateBinHash.h"  #include "bitFile.h"  #include "manipulateHash.h"  #include <fstream>  void migrateBinHash(HashTable &hashTable, std::string bitFileName,int number) {  std::fstream bitFile(bitFileName, std::ios::binary| std::ios::in | std::ios::app);  if(!bitFile.is\_open()){  std::cout<<"Error opening file";  return;  }  int cnt=0;  if(number==-1){  while(!bitFile.eof()){  std::string temp;  std::getline(bitFile,temp);  hashTable.add(temp.substr(0,temp.find(' ')), cnt);  cnt++;  }  }else{  std::string temp;  temp = directAccess(bitFileName, number);  if(temp.length()==0){  std::cout<<"Error";  return;  }  hashTable.add(temp.substr(0,temp.find(' ')), number);  }  }  void insertNoteToHash(std::string file, int note, HashTable &hashTable){  std::string temp = directAccess(file, note);  if(temp.length()==0){  std::cout<<"Error";  return;  }  hashTable.add(temp.substr(0,temp.find(' ')), note);  }  void deleteByKeyHash(HashTable &hashTable, std::string bitFileName,std::string key) {  int number = hashTable.get(key);  if(number==-1){  std::cout<<"Error";  return;  }  hashTable.remove(key);  deleteByKey(bitFileName, key);  }  void getByKeyHash(HashTable &hashTable, std::string bitFileName,std::string key) {  int number = hashTable.get(key);  if(number==-1){  std::cout<<"Error";  return;  }  directAccess(bitFileName, number);  } |