|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | |  |  |  | | 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 № 4** | |
| **по дисциплине** | |
| **«**Структуры и алгоритмы обработки данных**»**  **Тема: «Нелинейные структуры данных. Бинарное дерево.»** | |
|  | |
| 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**

Получение умений и навыков разработки и реализаций операций над структурой данных бинарное дерево.

# **Постановка задачи**

* 1. **Задача 1**

Вариант 2 (22). Вид дерева: идеально сбалансированное из n узлов (не AVL).

1. Реализовать операции общие для вариантов c 1 по 7
   1. Создать идеально сбалансированное бинарное дерево из n узлов. Структура узла дерева включает: информационная часть узла, указатель на левое и указатель на правое поддерево. Информационная часть узла определена вариантом.
   2. Отобразить дерево на экране, повернув его против часовой стрелки.
2. Реализовать операции варианта.
3. Разработать программу, демонстрирующую выполнение всех операций.
4. Составить отчет, отобразив в нем описание выполнения всех этапов разработки, тестирования и код всей программы со скриншотами результатов тестирования.

Таблица 1. Задания варианта №22

|  |  |
| --- | --- |
| Задание 1 | Значение информационной части - Целое число  Определить количество листьев с положительными значениями  Определить, сколько узлов дерева содержат заданное число.  Увеличить значения узлов вдвое, обходя дерево алгоритмом в ширину. |

# **Решение**

* 1. **Теоретическое введение**

Нелинейные структуры данных – такие структуры данных, отображающие сложные отношения между элементами структуры. Пример нелинейной структуры данных граф, и дерево, в частности.

Граф - топологическая модель, состоящая из множества вершин и множества ребер, соединяющих эти вершины. Дерево – связный граф, в котором нет циклов. Это означает, что из любой вершины можно попасть в любую, притом единственным путем. Деревья часто используются для организации иерархической структуры данных, например, при создании двоичных деревьев поиска или кучи, в этом случае одну вершину дерева называют корнем. Лист – вершина, не имеющая потомков. Обход дерева – систематический просмотр всех вершин, при котором каждая вершина встречается один раз.

Бинарное дерево – то дерево, в котором каждая вершина имеет два потомка (правые и левые поддеревья – left и right).

Идеально сбалансированное дерево – то дерево, в котором высоты поддеревьев отличаются не более, чем на 1.

Дерево выражений - такое дерево, которое в своем корне хранит число (если операторов нет), или оператор, связные вершины на расстоянии 1 хранят операторы (оператором может быть выражение). При этом каждое поддерево данного дерева является деревом выражений.

Бинарное дерево поиска – такое бинарное дерево, которое в левом узле хранит значение, меньшее значения потомка, а в правом – значение, большее значения потомка. Данные в бинарном дереве поиска хранятся в отсортированном виде. При каждой операции вставки нового или удаления существующего узла отсортированный порядок дерева сохраняется. При поиске элемента сравнивается искомое значение с корнем. Если искомое больше корня, то поиск продолжается в правом потомке корня, если меньше, то в левом, если равно, то значение найдено и поиск прекращается.

Обход дерева – алгоритм, в котором рассматриваются все вершины дерева.

Обход в глубину – рекурсивный метод обхода, в котором посещается корень дерева, левое поддерево и правое поддерево (в нужном порядке).

Обход в ширину – метод обхода, в котором используется очередь. Начиная с корня добавляем текущий узел в конец стека, далее пока очередь не кончится, итеративно посещаем каждую вершину из начала очереди, добавляя в ее конец дочерние вершины.

**Ответы на четные вопросы:**

*Какова степень сильноветвящегося дерева?* Если сильноветвящееся дерево содержит n-1 ключей и n указателей, то степень узлов n>2.

*Как рассчитать длину пути в дереве?* Длина пути дерева определяется как сумма длин путей ко всем его вершинам. Рассчитывается по рекурсивной формуле: Длина пути = длина пути в левом поддереве + длина пути в правом поддереве + количество узлов в дереве – 1.

*Может ли дерево быть пустым?* Да, ровно, как и любое множество. Такое дерево является nullptr.

*Дайте определение алгоритму обхода.* Алгоритм обхода – такой алгоритм, который осуществляет посещение каждой вершины дерева один раз.

Изобразите бинарное дерево, корень которого имеет индекс 6, и которое представлено в памяти таблицей вида

|  |  |  |  |
| --- | --- | --- | --- |
| *Индекс* | *key* | *left* | *right* |
| 1 | 12 | 7 | 3 |
| 2 | 15 | 8 | NULL |
| 3 | 4 | 10 | NULL |
| 4 | 10 | 5 | 9 |
| 5 | 2 | NULL | NULL |
| 6 | 18 | 1 | 4 |
| 7 | 7 | NULL | NULL |
| 8 | 14 | 6 | 2 |
| 9 | 21 | NULL | NULL |
| 10 | 5 | NULL | NULL |

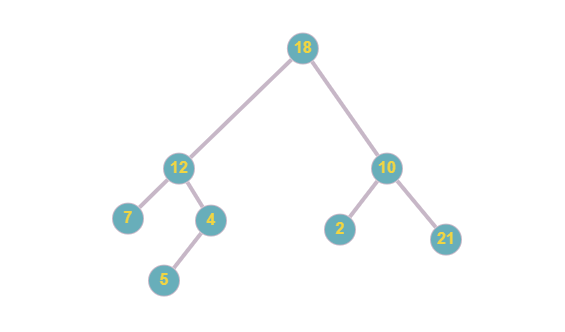


Рисунок 1.

Какая структура используется в алгоритме обхода дерева методом в «ширину»? Используется структура «Очередь». Она же – std::list, она же – двунаправленный список

Какая структура используется в не рекурсивном обходе дерева методом в «глубину»? Используется структура «Очередь». Она же – std::list, она же – двунаправленный список

Для каждого заданного арифметического выражения постройте бинарное дерево выражений:

1. a+b-c\*d+e

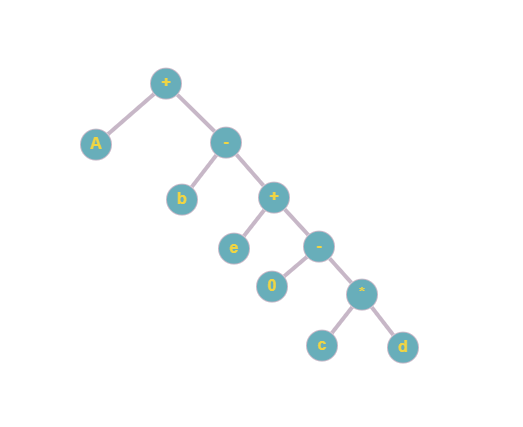


Рисунок 2.

1. /a-b\*c d

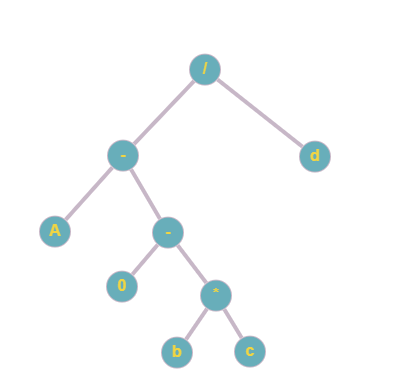


Рисунок 3.

1. a b c d / - \*

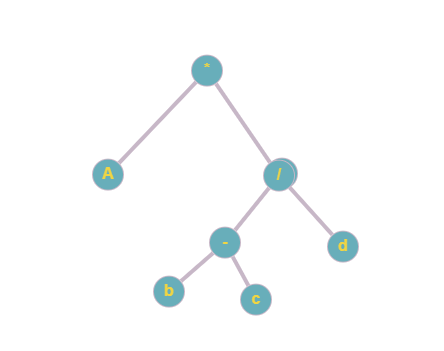


Рисунок 4.

1. - / + a b c d e

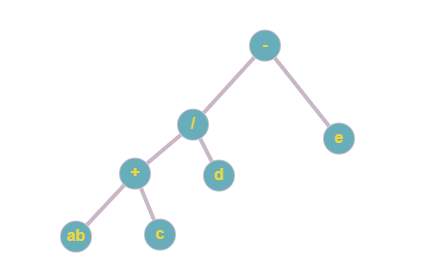


Рисунок 5.

Постройте бинарное дерево поиска, которое в результате симметричного обхода дало бы следующую последовательность узлов 40 45 46 50 65 70 75.

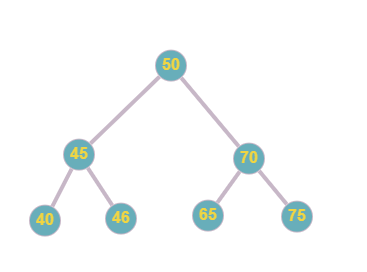


Рисунок 6.

* 1. **Функции задания №1**

Функции управления бинарным файлом заимствованы из практической работы №2 c некоторыми изменениями.

Для организации записей используется структура Record, состоящая из массивов знаковых переменных фиксированной длины:

|  |
| --- |
| struct Record{  char key[9];  char group[11];  char disciplineName[4];  char week[2];  char number[2];  char day[2];  char type[4];  char aud[4];  }; |

Функция printBitFile() осуществляет вывод содержимого бинарного файла в консоль, итеративно читая записи при помощи функции file.read():

|  |
| --- |
| struct Record{  char key[9];  char group[11];  char disciplineName[4];  char week[2];  char number[2];  char day[2];  char type[4];  char aud[4];  }; |

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

|  |
| --- |
| struct Record{  char key[9];  char group[11];  char disciplineName[4];  char week[2];  char number[2];  char day[2];  char type[4];  char aud[4];  }; |

Функция directAccess() принимает на вход имя файла, номер строки и длину 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 стркутуру):

|  |
| --- |
| Record 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);  if ((length\*sizeof(char))\*number > size){  bitFile.close();  return {};  }  bitFile.seekg((length)\*number);  Record record;  bitFile.read((char\*)&record, sizeof(record));  bitFile.close();  return record;  } |

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(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);  std::string s;  Record record;  while ( bitFile.read((char\*)&record, sizeof(record))){  if (record.key == key){  continue;  }  tempFile.write((char\*)&record, sizeof(record));  }  bitFile.close();  tempFile.close();  std::remove(bitFileName.c\_str());  if(std::rename("temp.bin", bitFileName.c\_str())==0){  return true;  } else{  return false;  }  } |

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

|  |
| --- |
| bool addNoteToFile(std::string bitFileName, Record note, int length){  std::fstream bitFile(bitFileName, std::ios::binary| std::ios::out | std::ios::app);  if (!bitFile.is\_open()){  return false;  }  bitFile.write((char\*)&note, length);  bitFile.close();  return true;  } |

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::hash() является хеш-функцией разработанной таблицы. На вход принимает строку-ключ, складывает все числовые значения char и возвращает остаток деления этого числа на размер таблицы:

|  |
| --- |
| 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;  } |

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: вывод в консоль, добавление записи в файл, удаление по ключу, прямой доступ к файлу:

|  |
| --- |
| int testBinF(){  std::string bitFileName = "trueBinShort.bin";  printBitFile(bitFileName);  deleteByKey(bitFileName, "00163287");  printBitFile(bitFileName);  std::cout<<std::endl;  directAccess(bitFileName, 1);  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.

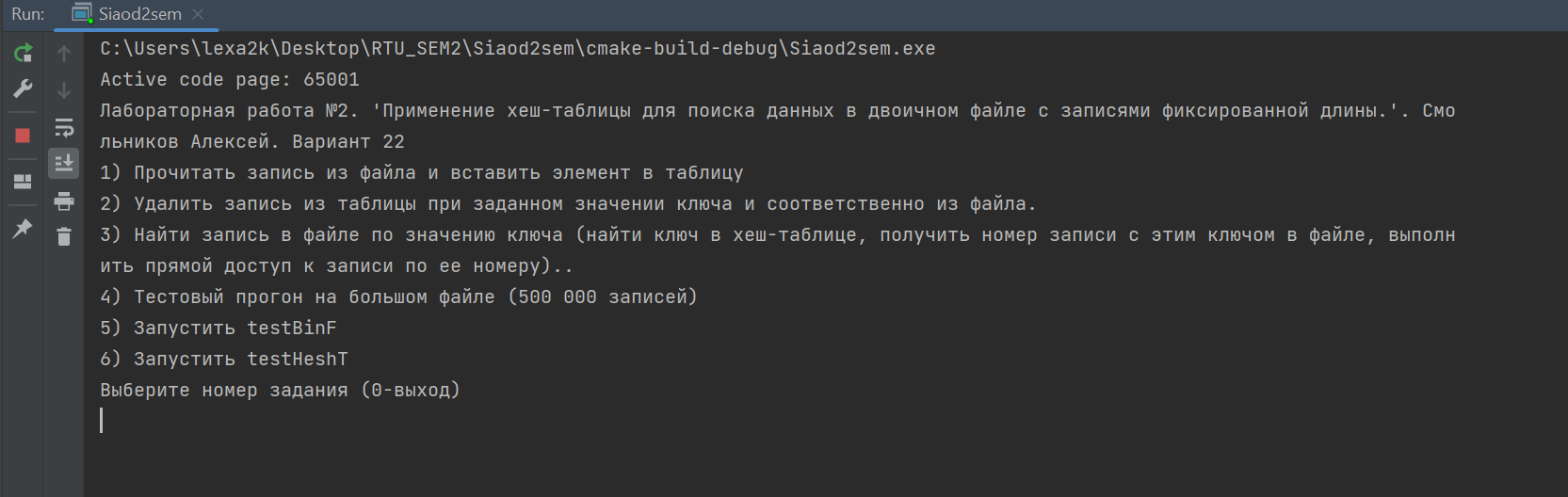


Рисунок 1. Интерфейс программы

# **Тестирование**

Рассмотрим структуру бинарного файла, в котором хранятся записи в виде структур:

|  |
| --- |
| 30 30 34 36 31 30 37 32 00 38 32 39 32 39 39 37 31 34 34 00 36 30 35 00 31 00 36 00 31 00 39 32 31 00 30 31 34 00 30 30 34 36 35 32 34 30 00 35 33 32 32 37 30 33 39 30 35 00 36 36 34 00 30 00 38 00 39 00 30 33 34 00 38 31 37 00 |

Функция migrateBinHash() заносит в хеш-таблицу необходимую запись из переданного файла:

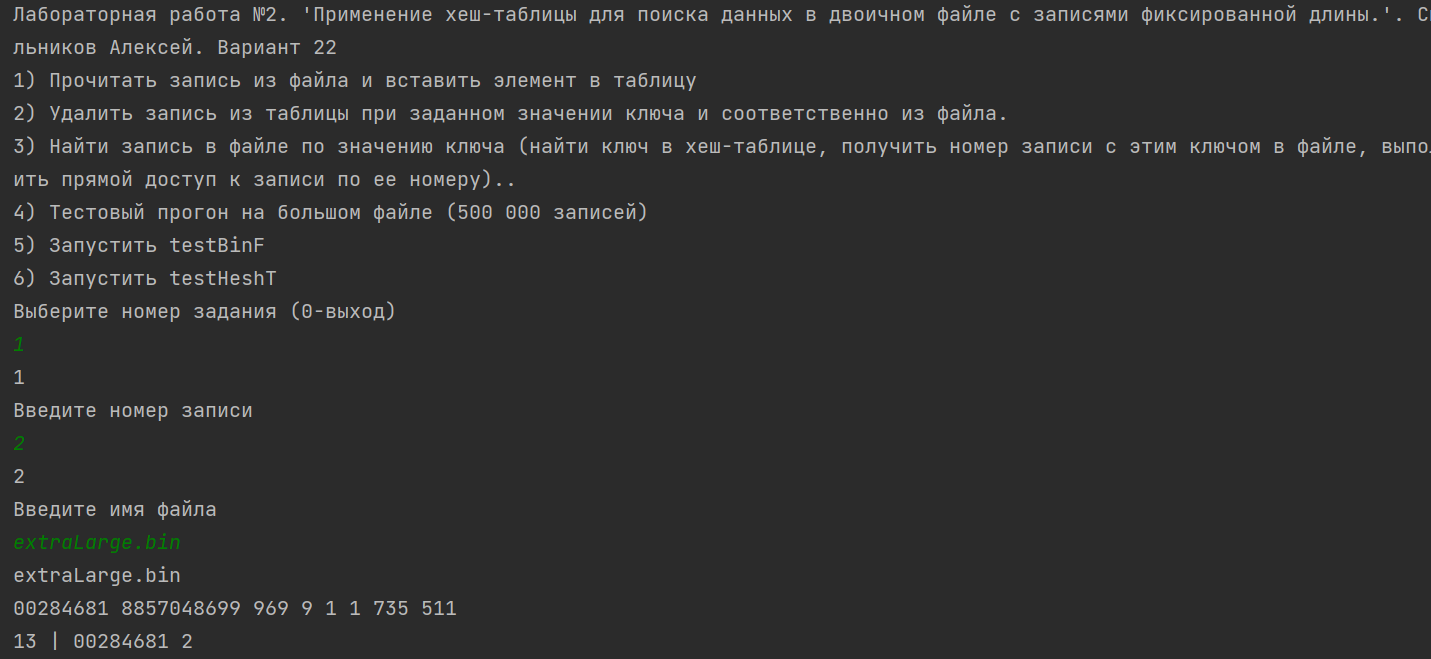


Рисунок 1. Случайно сгенерированный файл

В задании 2 выполняется удаление записи по ключу. На вход передадим ключ записи. После удаления можно удостовериться, что операция выполняется корректно:

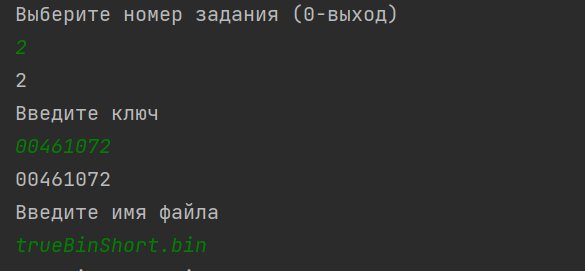


Рисунок 2. Вывод текстового файла

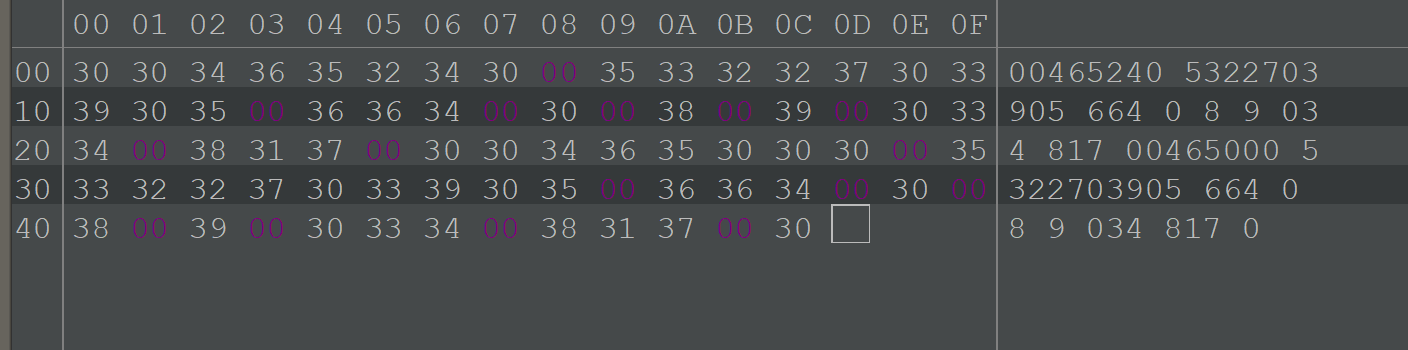


Рисунок 3. Файл после удаления строки

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

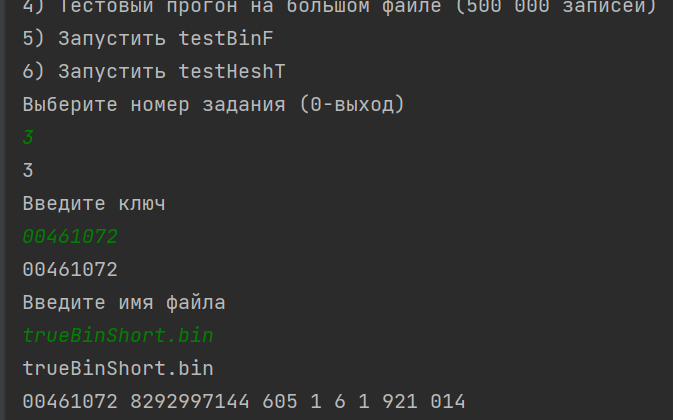


Рисунок 4. Результат работы функции

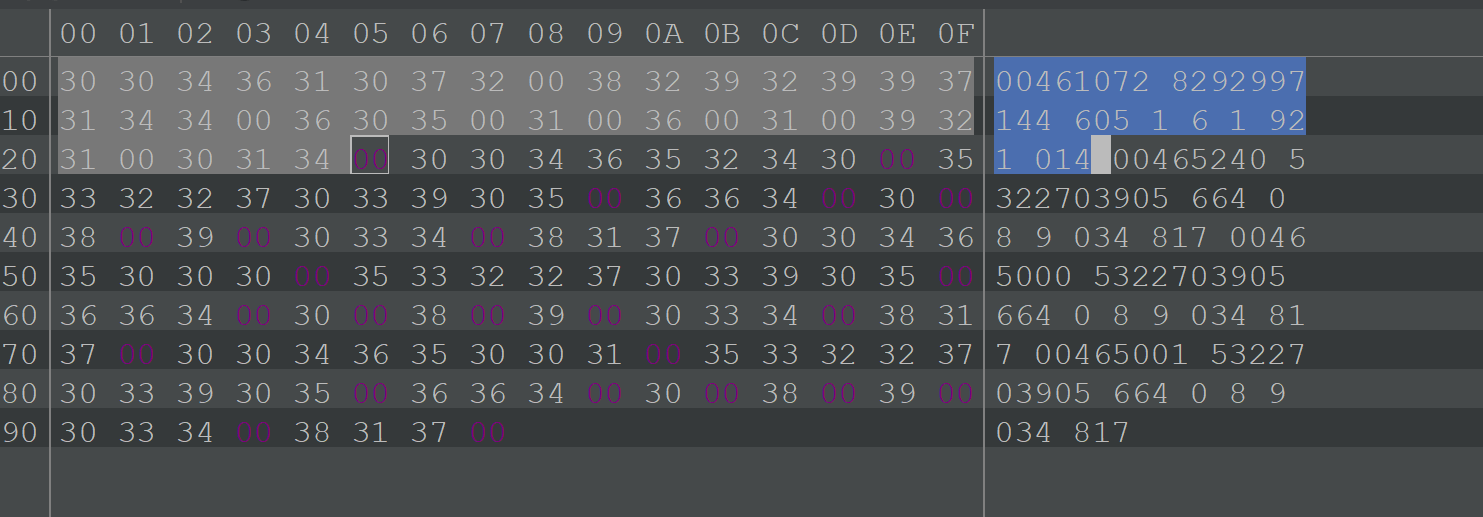


Рисунок 5. Файл, в котором производился поиск

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 816 микросекунд для последней записи до 4852 микросекунд для первой записи. Заметно, что получение порядкового номера строки из хэш-таблицы значительно увеличивает скорость работы программы, поиск выполняется за постоянное время:

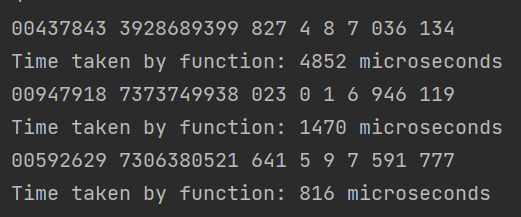


Рисунок 6. Время выполнения функций

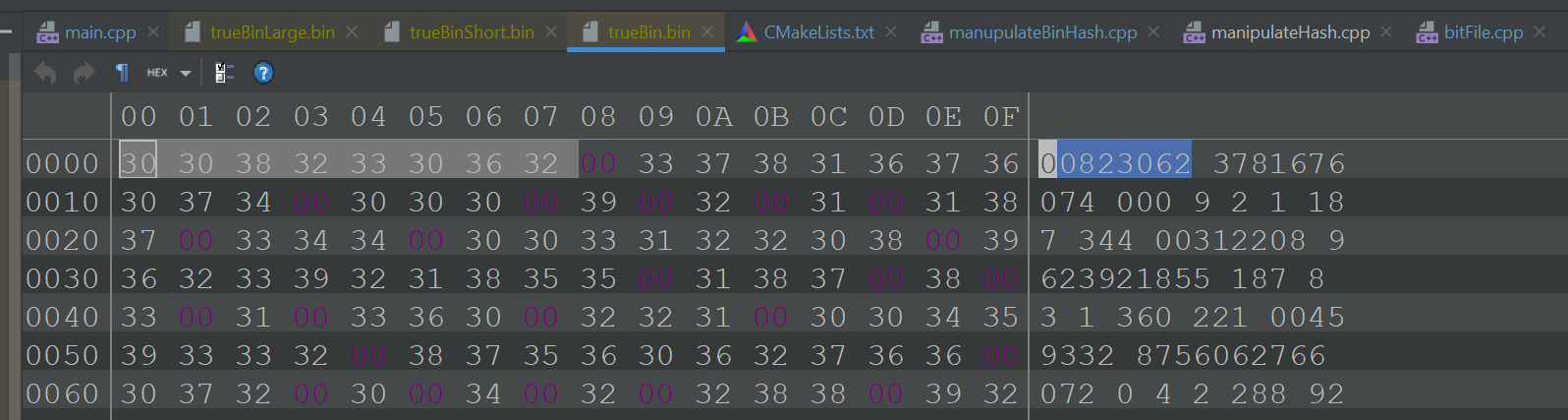


Рисунок 7. Первая запись

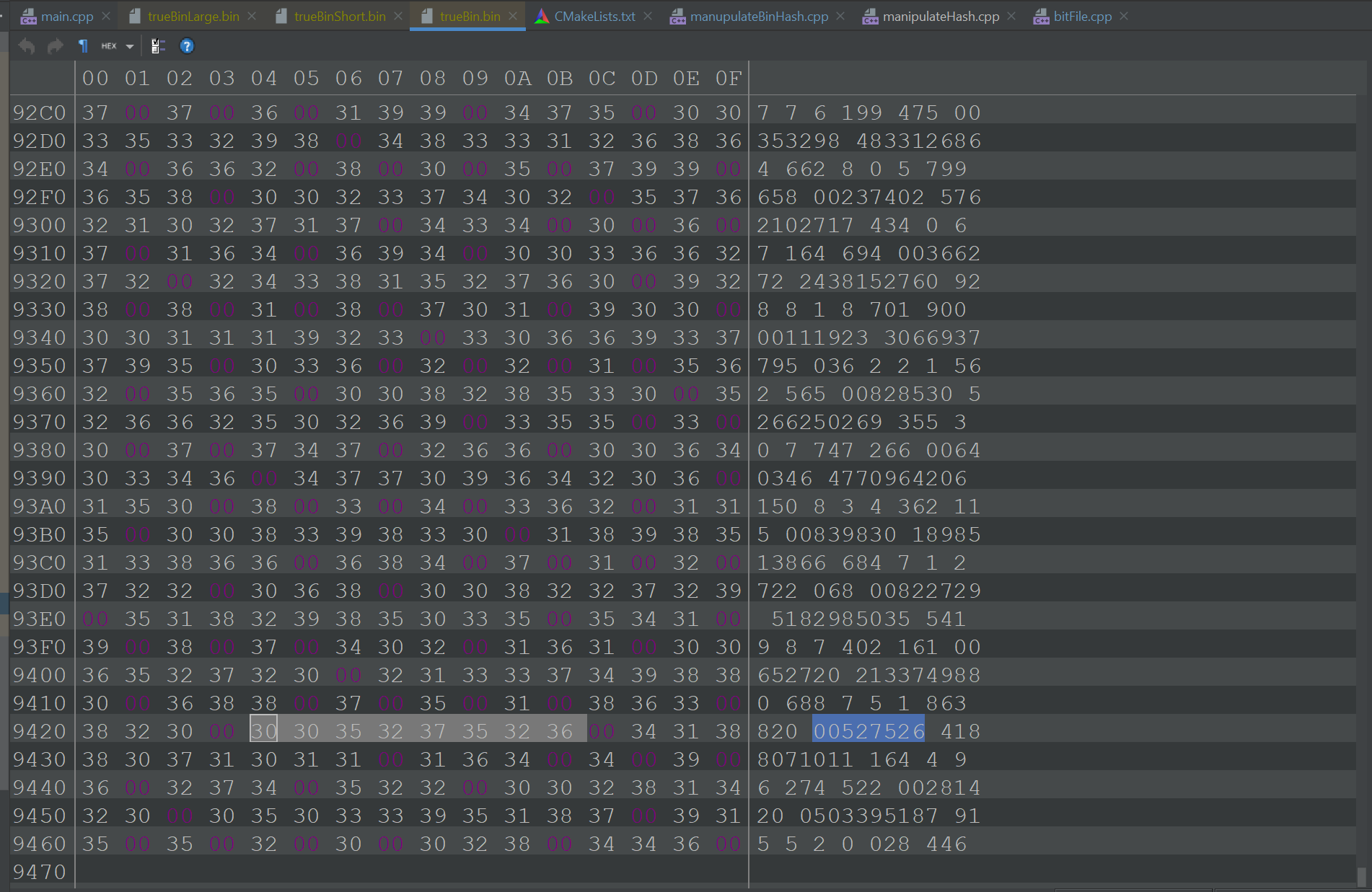


Рисунок 8. Последняя запись

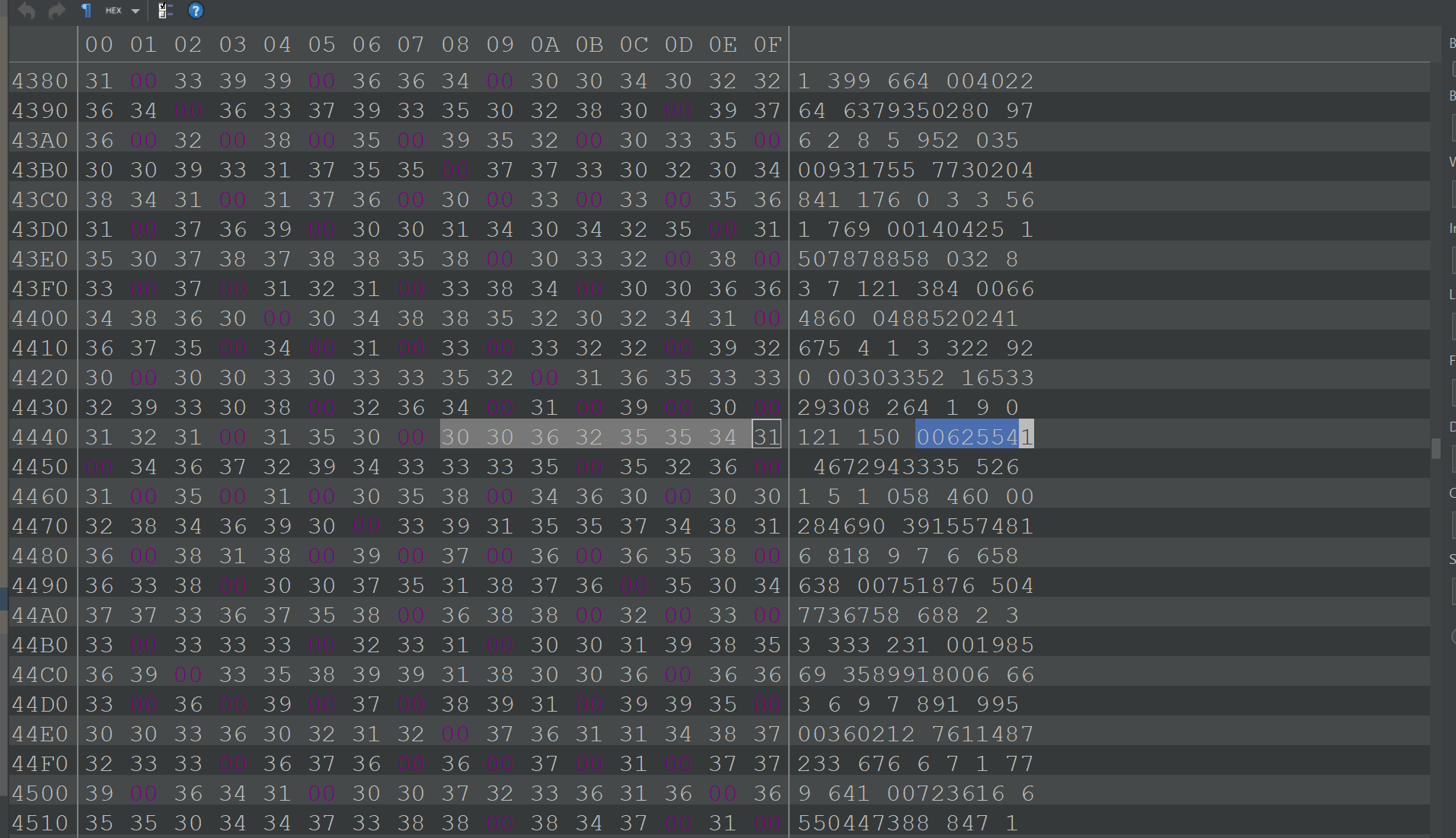


Рисунок 9. Запись около середины

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:

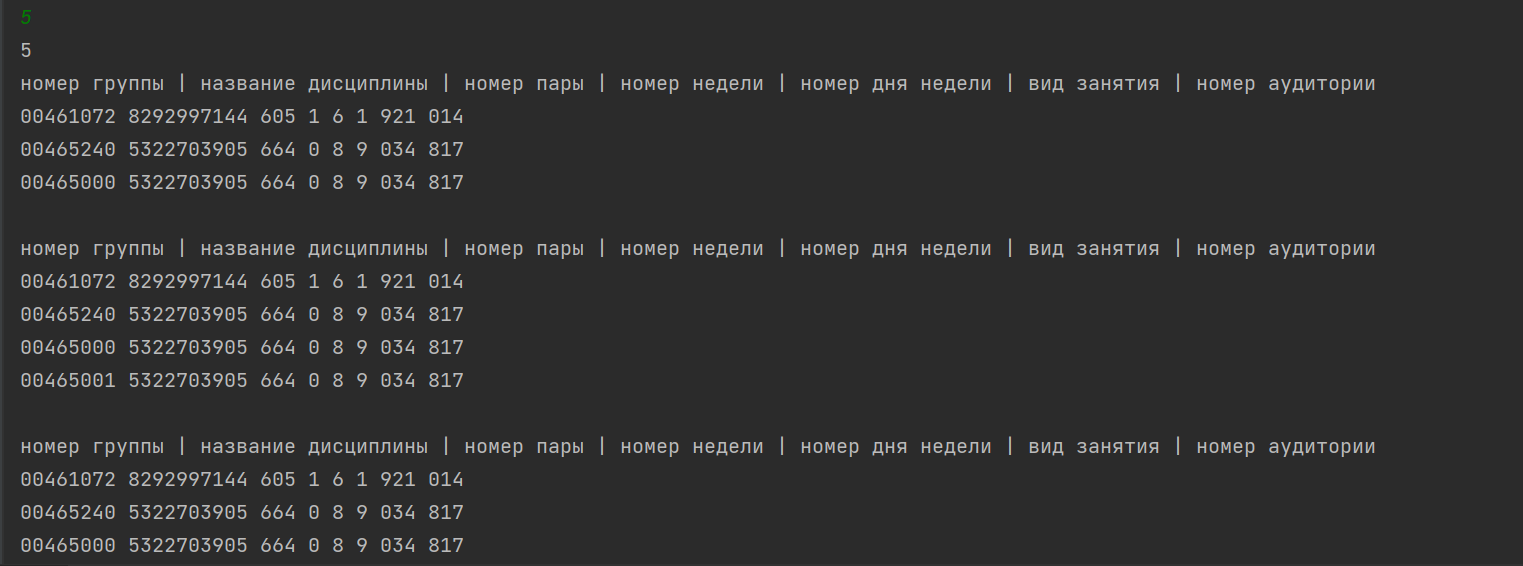


Рисунок 10. Результат работы функции

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:

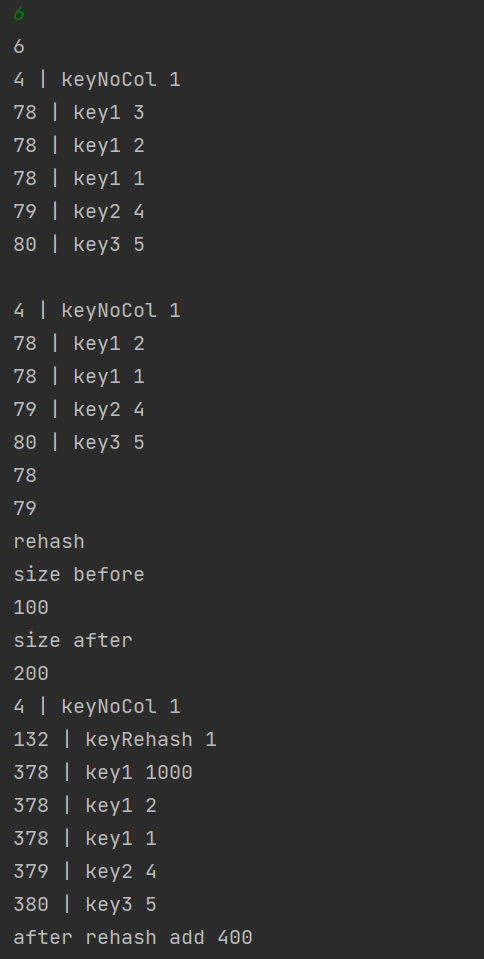


Рисунок 11. Интерфейс функции

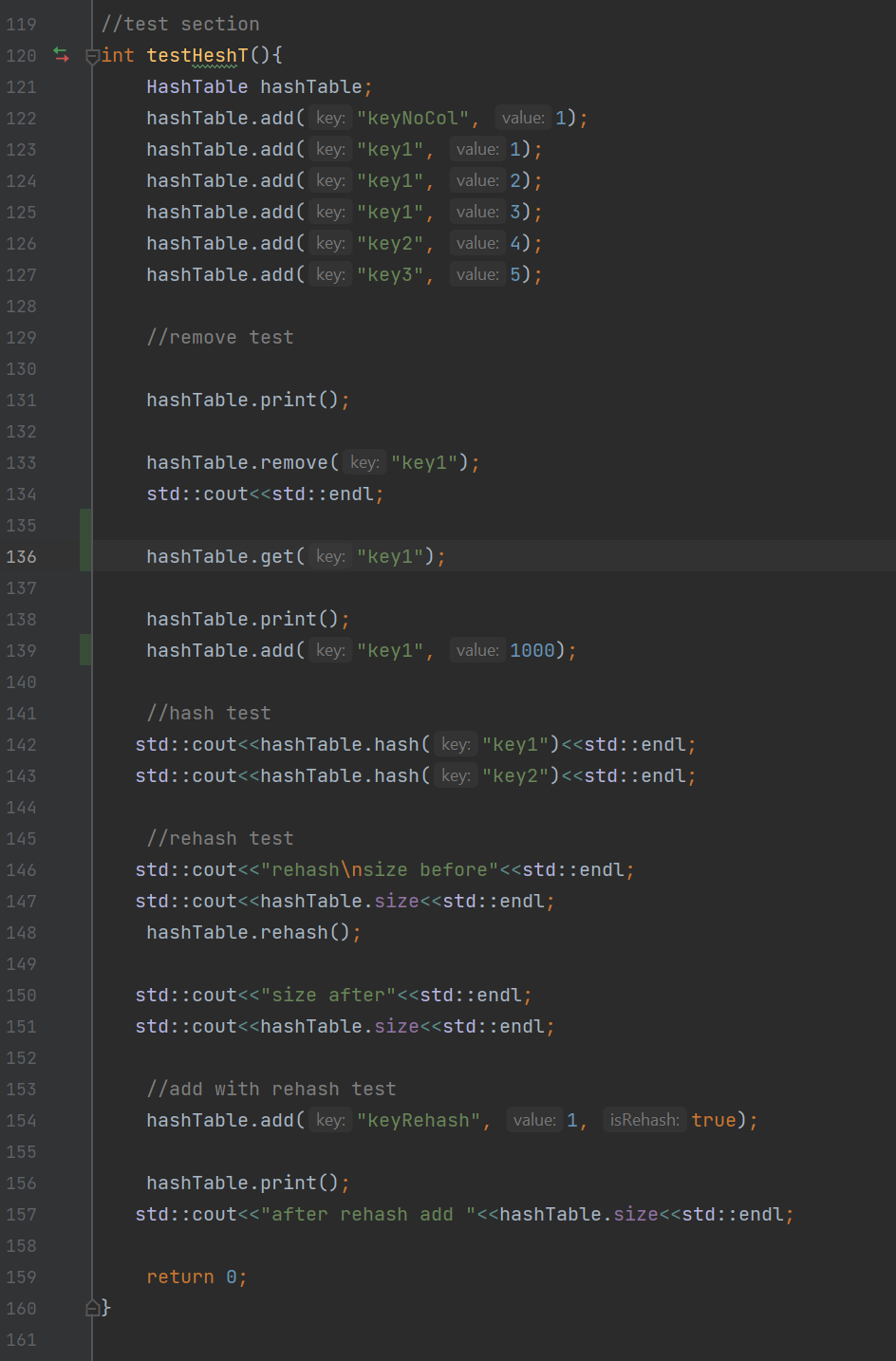


Рисунок 12. Код функции тестирования

# **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);  } |