|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | |  |  |  | | 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-03-21 | Хречко С.В. |
| 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. 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?

Вариант №29. Условие задания:

|  |  |
| --- | --- |
| Открытый адрес (двойное хеширование) | Справочник банков по городам страны. Об отдельном банке хранятся данные: наименование, код банка, адрес (город), форма собственности (коммерческий или государственный). |

# **Решение**

Хеш-функция — функция, осуществляющая преобразование массива входных данных произвольной длины в выходную битовую строку установленной длины, выполняемое определённым алгоритмом. Преобразование, производимое хеш-функцией, называется хешированием. Исходные данные называются входным массивом, «ключом» или «*сообщением*». Результат преобразования называется «*хешем*», «*хеш-кодом*», «*хеш-суммой*», «сводкой сообщения».

По заданию требовалось использовать двойное хеширование. Первая хеш-функция используется для нахождения первой предполагаемой позиции элемента в таблице. В формуле используется размер таблице, чтобы при большом размере таблицы значения первой хеш-функции не группировались в начале.

|  |
| --- |
| int hash1(const int id, const int tableSize){      return (((id % (tableSize + 7) + 1) << 2) - 3); //interesting enough???  } |

По заданию требовалось использовать двойное хеширование. Вторая хеш-функция позволяет пройтись по всей таблице, так как она возвращает взаимно-простые с размером таблицы значения, а для этого размерами таблицы могут быть только простые числа, которыми они и являются.

|  |
| --- |
| int hash2(const int id, const int tableSize){      return (id % (tableSize-1) + 1);  } |

По заданию требовалось добавлять элемент в хеш-таблицу. Функция вычисляет хеш элемента, затем проверяет занято ли вычисленное место в таблице, если нет, то вставляет переданный ключ и значение в таблицу, если же занято, совершает сдвиг на значение второй хеш-функции. Если все позиции в таблице занято, то вызывается функция рехеширования.

|  |
| --- |
| int addHash(vector<vector<int>> &hashTable, int &tableSize, const int id, const int index){      int hash = hash1(id, tableSize) % tableSize;      int step = hash2(id, tableSize);      for(int i = 0; i < tableSize; i++){          if(hashTable[hash][1] < 0){ //-1 means empty, -2 means deleted              hashTable[hash][0] = id;              hashTable[hash][1] = index;              return 0; //0 means success          }          hash = (hash + step) % tableSize;      }      return resizeHash(hashTable, tableSize); //resize if full  } |

По заданию требовалось совершать рехеширование. Функция увеличивает значение размера хеш-таблицы в соответствии с массивом размеров хещ-таблице, в нем присутствуют только простые числа. Затем создается новый массив уже большего размера, и все элементы старого последовательно перехешируются в новый. После чего старый массив будет удален, а переменной, по которой мы работаем с хеш-таблецей будет присвоен новый.

|  |
| --- |
| int resizeHash(vector<vector<int>> &hashTable, int &tableSize){      int oldTableSize = tableSize;      int simpleNums[6] = {13, 19, 53, 601, 12289, 1046527};      for(int i = 0; i < 6; i++){          if(simpleNums[i] > tableSize){              tableSize = simpleNums[i];              break;          }          if(i == 5){              return -1; //error too much data          }      }      vector<vector<int>> newHashTable(tableSize, {-1, -1});      for(int i = 0; i < oldTableSize; i++){          if(hashTable[i][1] >= 0){ // >= 0 means not empty or deleted              addHash(newHashTable, tableSize, hashTable[i][0], hashTable[i][1]);          }      }      hashTable = newHashTable;      return 0;  } |

По заданию требовалось удалять элемент из хеш-таблицы. Функция вычисляет значения двух хеш-функций для нахождения элемента, после чего ищет элемент, если в процессе поиска была найдена пустая, не удаленная ранее позиция, то поиск прекращается, так как мы понимаем, что искомый элемент в таблице отсутствует. При нахождении элемента ему присваевается значение удаленного, чтобы различать удаленные и пустые позиции, чтобы избежать проблем с поиском.

|  |
| --- |
| int deleteHash(vector<vector<int>> &hashTable, const int tableSize, const int id){      int hash = hash1(id, tableSize) % tableSize;      int step = hash2(id, tableSize);      for(int i = 0; i < tableSize; i++){          if(hashTable[hash][1] == -1){ //-1 means empty              return -1; //-1 means not found          }          if(hashTable[hash][0] == id){              hashTable[hash][1] = -2; //-2 means deleted   deleted != empty so search should be fine              return 0; //>= 0 means success;          }          hash = (hash + step) % tableSize;      }      return -1; //-1 means not found  } |

По заданию требовалось найти элемент в хеш-таблице по ключу. Функция вычисляет хеш-функции для нахождения элемента. Затем ищет элемент, если в цикле поиска встречается элемент пустой, не удаленный, а именно пустой, то функция делает вывод, что искомого элемента нет в таблице и возвращает соответствующее значение, если же элемент найден, то возвращается его значение.

|  |
| --- |
| int findByHash(const vector<vector<int>> &hashTable, const int tableSize, const int id){      int hash = hash1(id, tableSize) % tableSize;      int step = hash2(id, tableSize);      for(int i = 0; i < tableSize; i++){          if(hashTable[hash][1] == -1){ //-1 means empty              return -1; //-1 means not found          }          if(hashTable[hash][0] == id){              return hashTable[hash][1]; //return index          }          hash = (hash + step) % tableSize;      }      return -1; //-1 means not found  } |

По условию задания требовалось преобразовать данные из текстового файла в двоичный, функция проверяет открытие текстового файла, а бинарный файл создает, если его нет. Далее построчно читает текстовый файл и разбивает его на слова, разделителем является точка с запятой. Далее считанные слова записываются в структуру, по четыре поля на одну структуру, так как предполагается, что записи в файле соответствует структуре данных. Далее структура записывается в бинарный файл, для этого файл открыт для записи, как бинарный.

|  |
| --- |
| int toBinary(string fileName, string fileNameBin){//read from text file and write to binary file      ifstream fin("../../" + fileName);      fstream fout("../../" + fileNameBin, ios::out|ios::binary|ios::trunc);      if(!fin.is\_open() || !fout.is\_open()){          cout << "file doesn't exist" << endl;          fin.close();          fout.close();          return 1;      }      while(fin.good()){          Bank bank;          vector<string> tokens;          for (string each; getline(fin, each, ';'); tokens.push\_back(each));          for(int i = 0; i < tokens.size(); i++){              //cout << tokens.size() << " " << tokens[i] << endl;              if(i%4 == 0) bank.id = stoi(tokens[i]);              else if(i%4 == 1) strcpy(bank.name, tokens[i].c\_str());              else if(i%4 == 2) strcpy(bank.city, tokens[i].c\_str());              else if(i%4 == 3){                  bank.commercialOrState = (tokens[i] == "true") ? true : false;                  fout.write((char\*)&bank, sizeof(bank));              }          }        }      fin.close();      fout.close();      return 0;  } |

По условию задания требовалось добавлять банк в файле. Функция открывает файл и проверяет его открытие. Добавление происходит по индексу, по умолчанию (установлено в заголовочном файле) установка производится в конец файла.

|  |
| --- |
| int addBank(string fileNameBin, const Bank &bank, int ind){      fstream fout("../../" + fileNameBin, ios::in|ios::out|ios::binary);      if(!fout.is\_open()){          cout << "file doesn't exist" << endl;          fout.close();          return 1;      }      if(ind == -1){          fout.seekp(0, ios::end);          fout.write((char\*)&bank, sizeof(bank));          fout.close();          return 0;      }      fout.seekp((ind)\*sizeof(bank), ios::beg);      // if(fout.write((char\*)&bank, sizeof(bank))){      //  fout.close();      //  return 0;      // }      fout.write((char\*)&bank, sizeof(bank));      fout.close();      return 0;  } |

По условию задания требовалось считать банк по номеру в файле. Функция открывает файл и проверяет открытие. Функция смещает указатель чтения файла на нужное количество записей, после чего читает запись, если чтение было не удачным, то возвращается соответственный код, если успешно, то мы присваиваем переданной переменной для банка считанную запись.

|  |
| --- |
| int getBank(string fileNameBin, Bank &bank, int ind){//get bank by index from binary file starting from 1      //returns 0 if bank was found, 1 if file doesn't exist, 2 if bank wasn't found      fstream fin("../../" + fileNameBin, ios::in|ios::binary);      if(!fin.is\_open()){          cout << "file doesn't exist" << endl;          fin.close();          return 1;      }        fin.seekg(sizeof(Bank)\*(ind), ios::beg);      Bank readBank;      fin.read((char\*)&readBank, sizeof(Bank));      if(!fin.good()){          return 2;      }      bank = readBank;      fin.close();      return 0;  } |

По условию задания требовалось удалить запись по ключу, для этого также будет функция использующая хеш-таблице, но по условию задания, нужно было выполнить и без нее. Функция открывает файл и проверяет открытие, затем в цикле считывает подряд записи, если находит нужную запись, то присваивает ей значение удалено. Если запись найдена не была, то возвращается соответствующий код.

|  |
| --- |
| int deleteBankById(string fileNameBin, int id){//delete bank by id from binary file      //returns 0 if bank was deleted, 1 if file doesn't exist, 2 if bank wasn't found      fstream fin("../../" + fileNameBin, ios::in|ios::out|ios::binary);      if(!fin.is\_open()){          cout << "file doesn't exist" << endl;          fin.close();          return 1;      }      Bank bank;      fin.seekg(0, ios::beg);      while(fin.read((char\*)&bank, sizeof(bank))){          if(bank.id == id){              bank.id = -1; //means that bank was deleted              fin.seekp(-sizeof(bank), ios::cur);              fin.write((char\*)&bank, sizeof(bank));              fin.close();              return 0;          }      }      fin.close();      return 2;  } |

Для последующего использования совместно с хеш-таблице была написана функция удаляющая запись по порядковому номеру в файле. Функция открывает файл и проверяет открытие, затем перемещает курсор считывания на нужное количество записей, считывает запись, помечает ее удаленной и записывает на то же место в файл, таким образом удаляя запись.

|  |
| --- |
| int deleteBankByIndex(string fileNameBin, int ind){      //returns 0 if bank was deleted, 1 if file doesn't exist, 2 if bank wasn't found      fstream fin("../../" + fileNameBin, ios::in|ios::out|ios::binary);      if(!fin.is\_open()){          cout << "file doesn't exist" << endl;          fin.close();          return 1;      }      fin.seekg(sizeof(Bank)\*(ind), ios::beg);      Bank readBank;      fin.read((char\*)&readBank, sizeof(Bank));      if(!fin.good()){          return 2;      }      readBank.id = -1; //means that bank was deleted      fin.seekp(-sizeof(readBank), ios::cur);      fin.write((char\*)&readBank, sizeof(readBank));      fin.close();      return 0;  } |

По условию задания требовалось прочитать одну запись из файла и захешировать ее. Функция открывает файл и проверяет его открытие. Затем по переданному порядковому номеру находит и считывает запись, если считывание было успешным, то функция вызывают функцию хеш-таблицы по добавлению элемента, в противном случае возвращает код ошибки. Также функция увеличивает переданную переменную количества занятых позиций в таблице, а также отслеживает заполнение таблицы, если таблица окажется более чем на 75% заполненной, то перед добавлением, вызывается функция рехеширования для увеличения таблицы.

|  |
| --- |
| int hashRecord(string fileName, int ind, vector<vector<int>> &hashTable, int &tableSize, int &hashElements, int &deletedElements){      fstream fin("../../" + fileName, ios::in|ios::binary);      if(!fin.is\_open()){          cout << "file doesn't exist" << endl;          fin.close();          return 1;      }      fin.seekg(sizeof(Bank)\*(ind), ios::beg);      Bank readBank;      fin.read((char\*)&readBank, sizeof(Bank));      if(!fin.good()){          return 2; //index out of bounds      }      int id = readBank.id;      if(hashElements > (int)(0.75 \* tableSize)){          resizeHash(hashTable, tableSize);          hashElements -= deletedElements;          deletedElements = 0;      }      hashElements++;      return addHash(hashTable, tableSize, id, ind);  } |

Для удобства была добавлена функция хеширующая целый файл. Функция открывает файл и проверяет открытие. Затем в цикле читает записи в файле и для каждой запси вызывает функцию хеш-таблицы по добавлению записи в таблицу. Также функция ведет учет того, насколько хеш-таблица заполнена и если заполнение превышает 75% то вызывает функцию рехеширования.

|  |
| --- |
| int hashFile(string fileName, vector<vector<int>> &hashTable, int &tableSize, int &hashElements, int &deletedElements){      //returns 0 if file was hashed, 1 if file doesn't exist      fstream fin("../../" + fileName, ios::in|ios::binary);      if(!fin.is\_open()){          cout << "file doesn't exist" << endl;          fin.close();          return 1;      }      Bank bank;      fin.seekg(0, ios::beg);      int ind = 0;      while(fin.read((char\*)&bank, sizeof(bank))){          int id = bank.id;          if(hashElements > (int)(0.75 \* tableSize)){              resizeHash(hashTable, tableSize);              hashElements -= deletedElements;              deletedElements = 0;          }          hashElements++;          addHash(hashTable, tableSize, id, ind);          ind++;      }      fin.close();      return 0;  } |

По условию задания требовалось удалить запись в файле по ключу, используя хеш-таблицу. Функция открывает файл и проверяет открытие. Затем находит искомый порядковый номер записи, вызывая функцию поиска в хеш-таблице. Также функция увеличивает переменную, хранящую количество удаленных записей.

|  |
| --- |
| int deleteRecordById(string fileName, int id, vector<vector<int>> &hashTable, int &tableSize, int &deletedElements){      //returns 0 if bank was deleted, 1 if file doesn't exist, 2 if bank wasn't found      fstream fin("../../" + fileName, ios::in|ios::out|ios::binary);      if(!fin.is\_open()){          cout << "file doesn't exist" << endl;          fin.close();          return 1;      }      int ind = findByHash(hashTable, tableSize, id);      if(ind == -1){          return 2;      }        deleteHash(hashTable, tableSize, id);      deletedElements++;      deleteBankByIndex(fileName, ind);      fin.close();      return 0;  } |

По условию задания требовалось найти запись в файле по ключу. Функция открывает файл и проверяет открытие. Затем находит порядковый номер записи используя функцию поиска хеш-таблицы, после чего считывает из файла нужную запись и возвращает считанное.

|  |
| --- |
| Bank getRecordById(string fileName, int id, vector<vector<int>> &hashTable, int &tableSize){      fstream fin("../../" + fileName, ios::in|ios::binary);      if(!fin.is\_open()){          cout << "file doesn't exist" << endl;          fin.close();          Bank b; b.id = -2; return b; // id == -2 is file not found      }      int ind = findByHash(hashTable, tableSize, id);      if(ind == -1){          Bank b; b.id = -1; return b;      }      Bank bank;      fin.seekg(sizeof(Bank)\*(ind), ios::beg);      fin.read((char\*)&bank, sizeof(Bank));        fin.close();      return bank;  } |

Пользовательский интерфейс программы предлагает пользователю выбрать режим работы, проверять время работы или работать с небольшими файлами для отладки. После чего просит ввести названия файла и предлагает найти банк по ID или удалить его.

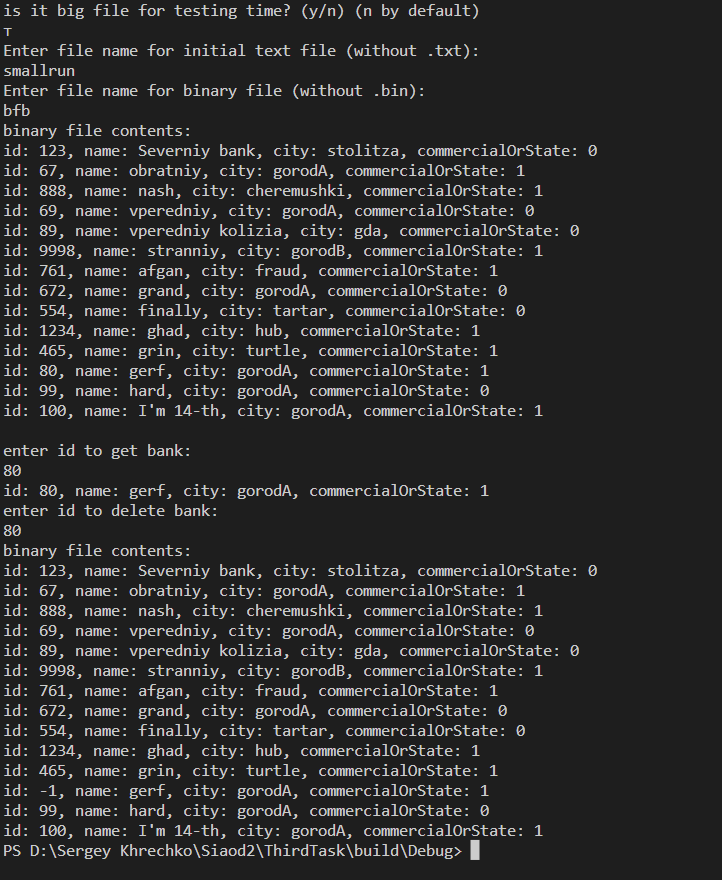


Рисунок 1. Интерфейс программы

Ответы на вопросы 10-ого пункта задания.

1. Хеш-функции применяются в очень большом спектре задач, в данной задаче и наиболее распространенная – это построение ассоциативных массивов.
2. Коллизия – это, когда два различных хешируемых объекта после применения к ним хеш-функции дают одинаковый результат.
3. При открытой адресации все элементы таблицы хранятся непосредственно в ней, а не в связанных списках или чем-то подобном.
4. В Хеш-таблице с открытым адресом коллизия реализуется так же как и в любом другом, а разрешается путем помещения элемента в другую ячейку, номер который вычисляется в зависимости от метода.
5. При удалении элемента из таблицы с открытым адресом может быть проблема при последующем поиске, так как если удаленные и не заполненные элементы будут восприниматься алгоритмом одинаково, то при встрече удаленного элемента поиск будет прекращен, хотя за удаленным элементом могут еще оставаться нужные нам, тогда как после не заполненного не могут. Разрешается эта проблемой путем пометки удаленных элементов, чтобы различать удаленные и пустые элемента.
6. Коэффициент нагрузки в хеш-таблице определяет то, какой процент от размера таблицы может быть заполнен до того, как потребуется рехеширование.
7. Первичный кластер в таблице с открытым адресом это ситуация при, которой уже установленные элементы сильно сгруппированы, и потому, когда нужно вставить элемент даже не относящийся к этому кластеру (хеш-код другой), перебору элементов прийдется обойти уже существующий кластер элементов не относящихся к его хеш-коду.
8. Двойное хеширование реализуется созданием второй хеш-функции, которая используется для рассчитывания сдвига элемента. Первая хеш-функция находит первое предполагаемое положение элемента в таблице, если возникает коллизия, то используется уже вторая хеш-функция.

# **Тестирование**

Тестирование хеш-функций. Для тестирования используется минимальный размер таблицы, а именно 13. Для такого размера первая хеш-функция будет брать остаток от деления на 20 (13 + 7), а потому каждые двадцать чисел значения будут повторяться (для этого размера). На (рис. 2, 3) представлены результаты для чисел от 0 до 29 со значениями хеш-функции и остатками этих значений от деления на размер таблицы, то есть 13.

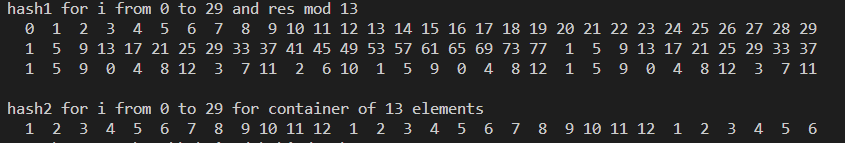


Рисунок 2. Тестирование Хэш-функций

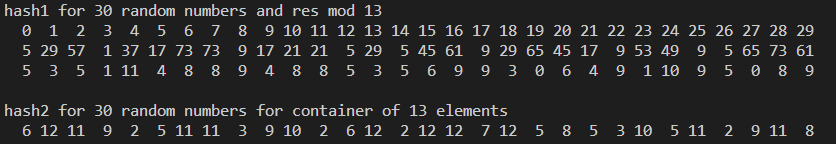


Рисунок 3. Тестирование Хэш-функций

Тестирование функции добавления без рехеширования. На (рис. 4, 5) представлены результаты тестирования. В первой строке приведены числа, которые мы будем хешировать (числа будут являться ключами). Во второй строке позиции в хеш-таблице. Так как используется таблица размера 13, то если мы попробуем захешировать 14-ое число, то мы не сможем, что видно на (рис. 5). Для данного тестирования автоматической вызов рехеширования был отключен.

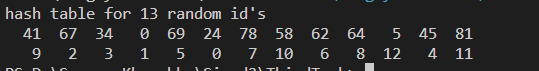


Рисунок 4. Тестирование функции добавления

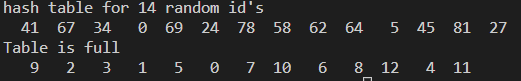


Рисунок 5. Тестирование функции добавления

Тестирование функции удаления из хеш-таблицы. На (рис. 6, 7) приведены результаты. Использовались случайные числа от 0 до 100. Из тестирования видно, что при удалении, удаленным элементам присваивается значение -2, что показывает, что данный элемент был именно удален. Пустые же изначально заполняются значениями -1.

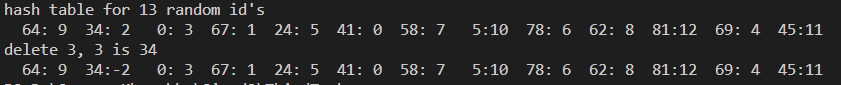


Рисунок 6. Тестирование функции удаления

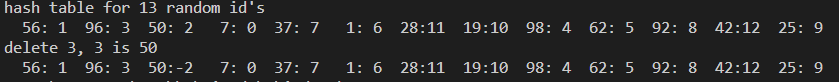


Рисунок 7. Тестирование функции удаления

Тестирование рехеширования. На (рис. 8) приведена ситуация, в которой потребовалось рехеширование при добавлении. Также можно убедиться, что пустые и удаленные элементы отмечены по-разному.

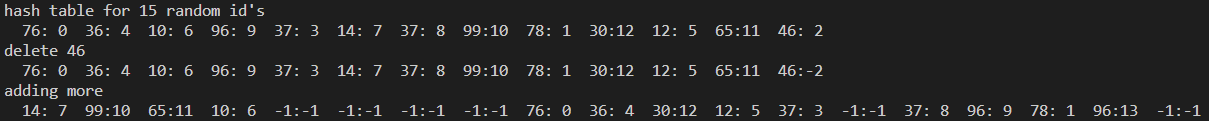


Рисунок 8. Тестирование добавления с рехэшированием

Тестирование поиска в хеш-таблице. На (рис. 9, 10) показаны результаты тестирования, как нахождения так и не нахождения (после удаления 45, 45 найти не удалось, что ожидаемо, это так же подтверждает работоспособность удаления). На (рис. 9) видно, что после удаления 40-ка, на позиции 1 в таблице (удаление проверено, присваивает -2), поиск пошел искать дальше, и добавив 3 (hash2 для 20), нашел искомое.

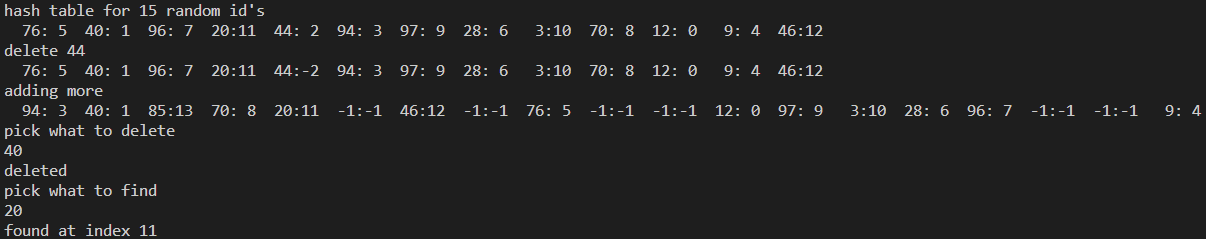


Рисунок 9. Тестирование поиска

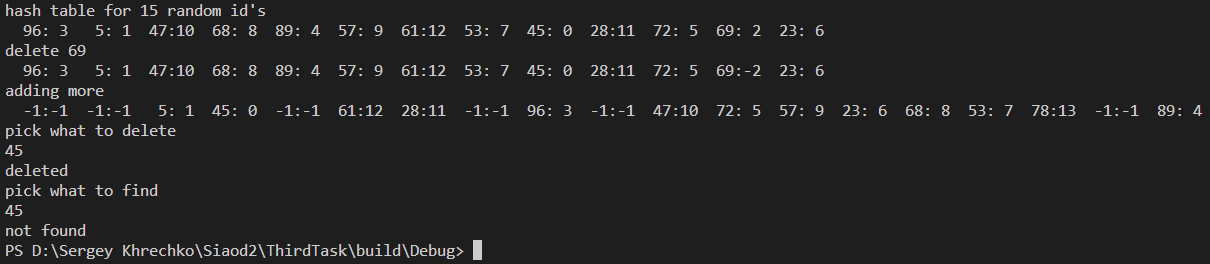


Рисунок 10. Тестирование поиска

Тестирование перевода из текстового файла в бинарный. На (рис. 11) приведен изначальный текстовый файл. На (рис. 12) представлен вывод тестовой программы, после работы функции перевода в бинарный файл (читали для вывода естественно бинарный файл). Можно наблюдать, что информация была переведена верно.

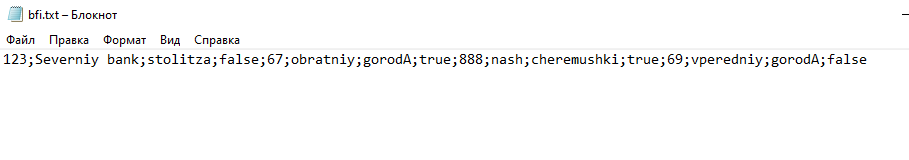


Рисунок 11. Тестирование перевода в бинарный файл

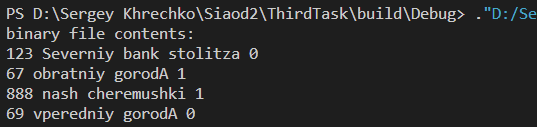


Рисунок 12. Тестирование перевода в бинарный файл

На (рис. 13) приведено тестирование поиска записи в файле. Функция идентична функции из лабораторной работы 2, поэтому тестирование было проведено в меньшем объеме. По-прежнему можно наблюдать работоспособность.

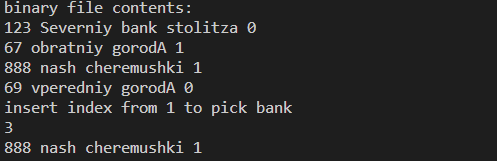


Рисунок 13. Тестирование получения записи файла

Тестирование функции удаления по ключу представлено на (рис. 14). После удаления ID записи стал равняться -1, что означает, что запись была удалена.

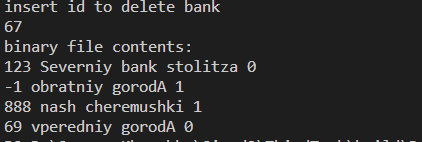


Рисунок 14. Тестирование удаления записи по id

Тестирование функции удаления записи из файла по порядковому номеру приведена на (рис. 15). На данном рисунке приведен вывод программы от начала и до конца, таким образом можно наблюдать изначальные данные, после чего удаление, как по ID, так и по индексу, оба успешны.

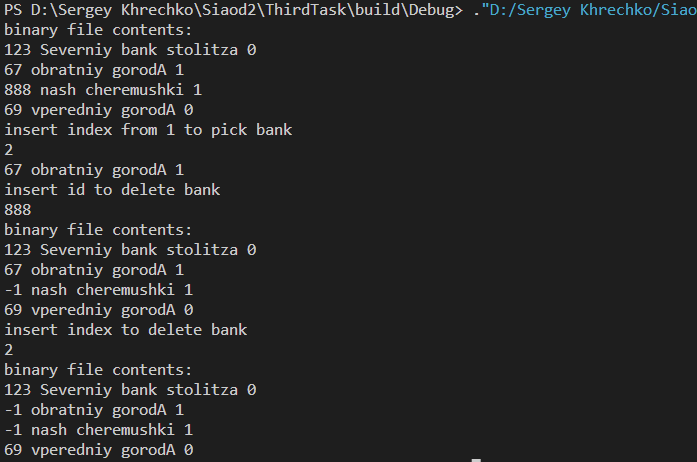


Рисунок 15. Тестирование удаления записи по индексу

Тестирование функции добавления записи в файл. На (рис. 16) показаны изначальные данные с одним удаленным банком, далее выведено, что будет добавлено. Добавляем на позицию два, туда где была удаленная запись, а также в конец файла. Как мы можем видеть, оба добавления работают согласно с изначальной задумкой.

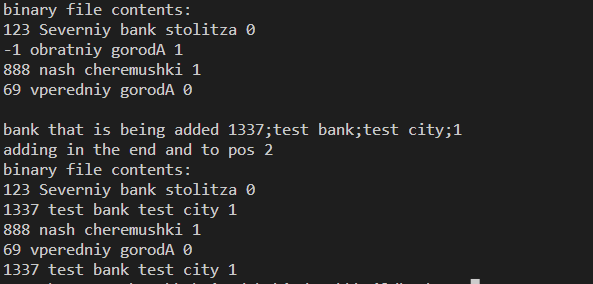


Рисунок 16. Тестирование добавления записи

Тестирование функции хеширования записей в файле. Для тестирования используется файл с небольшим записей, специально подобранных так, чтобы присутствовали записи приводящие к коллизии, не преводящие к ней, а также записей в файле больше 75% от 13, следовательно понадобится рехеширование. Результат тестирования приведен на (рис. 17). Можно наблюдать, что все записи были захешированы и хранящиеся в таблице значение соответствует порядковому номеру записи в файле, что подтверждает работоспособность алгоритма.

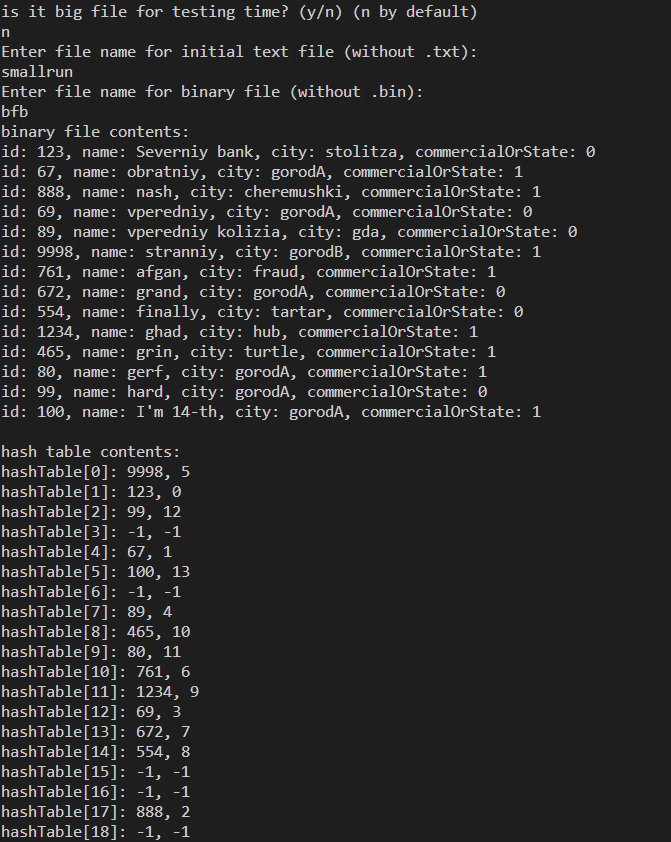


Рисунок 17. Тестирование хеширования файла

Тестирование функции поиска записи в файле с помощью хеш-таблицы приведено на (рис. 18, 19). Используется тот же файл, что и в предыдущем тестировании. На первом рисунке показан успешный поиск, на втором, попытка найти отсутствующую запись.

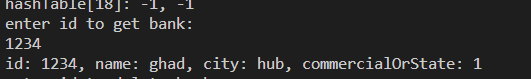


Рисунок 18. Тестирование поиска



Рисунок 19. Тестирование поиска

Тестирование функции записи по ключу приведено на (рис. 20). Используется тот же файл, что и в предыдущем тестировании. Из записей в таблице видно, что выбранный ключ 554 соответствует записи на позиции 8 (считая от 0-я). Из результатов видно, что на 8-мой позиции теперь стоит ключ -1, а значит, запись была удалена. Значит тестирование успешно.

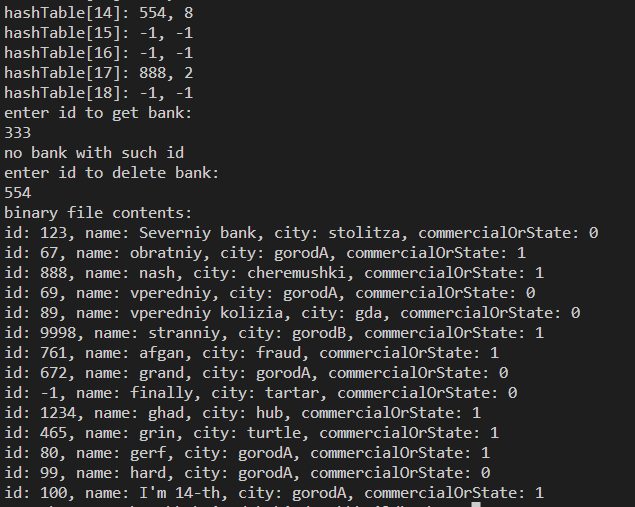


Рисунок 20. Тестирование поиска

Тестирование на большом количестве записей приведено на (рис. 21). Для тестирования был создан файл, содержащий 200000 записей. Для тестирования был проведен поиск записи в начале файла (в именах банков указаны их порядковые номера), середине файла и конце. Во всех трех случаях получено близкое время выполнения примерно равное 250 микросекундам. Мы также можем видеть, что при довольно большом количестве записей время поиска остается довольно маленьким.

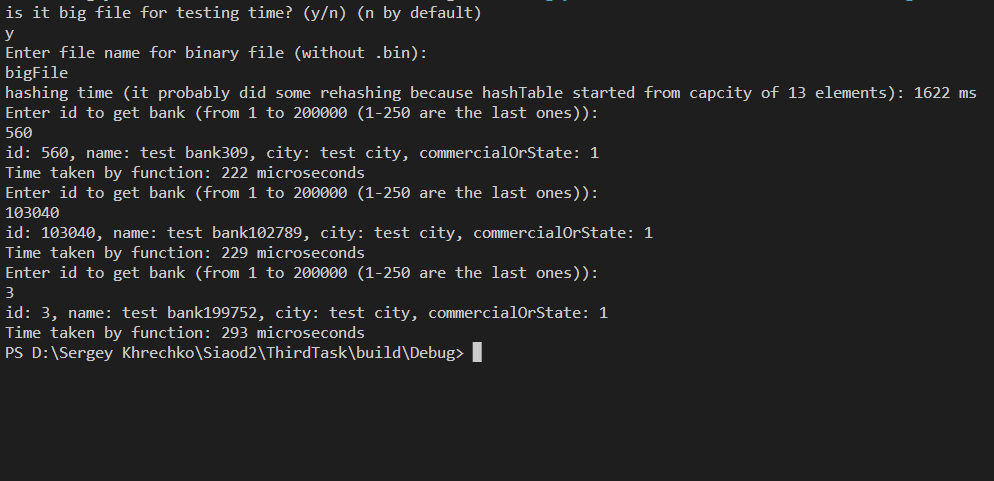


Рисунок 21. Тестирование по времени

Из результатов выполнения программы видно:

1. Программа работает корректно, все функции работоспособны.

# **Вывод**

В результате выполнения работы я:

1. Освоил алгоритмы работы с Хеш-таблицами.
2. Научился программировать поиск записей одинаковой длины в файле с помощью хеш-таблиц.

# **Исходный код программы**

|  |
| --- |
| #ifndef \_\_HASHFUNCTIONS\_H\_\_  #define \_\_HASHFUNCTIONS\_H\_\_  #include <iostream>  #include <iomanip>  #include <fstream>  #include <time.h>  #include <string>  #include <vector>  using namespace std;  int hash1(const int id, const int tableSize);  int hash2(const int id, const int tableSize);  int addHash(vector<vector<int>> &hashTable, int &tableSize, const int id, const int index);  int deleteHash(vector<vector<int>> &hashTable, const int tableSize, const int id);  int findByHash(const vector<vector<int>> &hashTable, const int tableSize, const int id);  int resizeHash(vector<vector<int>> &hashTable, int &tableSize);  void testHashT();  #endif  #include "hashFunctions.h"  int hash1(const int id, const int tableSize){      return (((id % (tableSize + 7) + 1) << 2) - 3); //interesting enough???  }  int hash2(const int id, const int tableSize){      return (id % (tableSize-1) + 1);  }  int addHash(vector<vector<int>> &hashTable, int &tableSize, const int id, const int index){      int hash = hash1(id, tableSize) % tableSize;      int step = hash2(id, tableSize);      for(int i = 0; i < tableSize; i++){          if(hashTable[hash][1] < 0){ //-1 means empty, -2 means deleted              hashTable[hash][0] = id;              hashTable[hash][1] = index;              return 0; //0 means success          }          hash = (hash + step) % tableSize;      }      return resizeHash(hashTable, tableSize); //resize if full  }  int deleteHash(vector<vector<int>> &hashTable, const int tableSize, const int id){      int hash = hash1(id, tableSize) % tableSize;      int step = hash2(id, tableSize);      for(int i = 0; i < tableSize; i++){          if(hashTable[hash][1] == -1){ //-1 means empty              return -1; //-1 means not found          }          if(hashTable[hash][0] == id){              hashTable[hash][1] = -2; //-2 means deleted   deleted != empty so search should be fine              return 0; //>= 0 means success;          }          hash = (hash + step) % tableSize;      }      return -1; //-1 means not found  }  int findByHash(const vector<vector<int>> &hashTable, const int tableSize, const int id){      int hash = hash1(id, tableSize) % tableSize;      int step = hash2(id, tableSize);      for(int i = 0; i < tableSize; i++){          if(hashTable[hash][1] == -1){ //-1 means empty              return -1; //-1 means not found          }          if(hashTable[hash][0] == id){              return hashTable[hash][1]; //return index          }          hash = (hash + step) % tableSize;      }      return -1; //-1 means not found  }  int resizeHash(vector<vector<int>> &hashTable, int &tableSize){      int oldTableSize = tableSize;      int simpleNums[6] = {13, 19, 53, 601, 12289, 1046527};      for(int i = 0; i < 6; i++){          if(simpleNums[i] > tableSize){              tableSize = simpleNums[i];              break;          }          if(i == 5){              return -1; //error too much data          }      }      vector<vector<int>> newHashTable(tableSize, {-1, -1});      for(int i = 0; i < oldTableSize; i++){          if(hashTable[i][1] >= 0){ // >= 0 means not empty or deleted              addHash(newHashTable, tableSize, hashTable[i][0], hashTable[i][1]);          }      }      hashTable = newHashTable;      return 0;  }  void testHashT(){      srand(time(NULL));      int simpleNums[6] = {13, 19, 53, 601, 12289, 1046527}; //simple numbers for hash table size the last one is 1 046 527 for 1 million test      int hashSize = simpleNums[0];      // hash table: (id, index)      vector<vector<int>> hashTable(hashSize, {-1, -1});      int a[15];      for(int i = 0; i < 15; i++){          a[i] = rand() % 100;      }      cout << "hash table for 15 random id's" << endl;      for(int i = 0; i < hashSize; i++){          if(addHash(hashTable, hashSize, a[i], i) == -1) cout << "Table is full" << endl;      }      for(int i = 0; i < hashSize; i++){          cout << setw(4) << hashTable[i][0] << ":" << setw(2) << hashTable[i][1];      }      cout << endl;      cout << "delete " << a[2] << endl;      deleteHash(hashTable, hashSize, a[2]);      for(int i = 0; i < hashSize; i++){          cout << setw(4) << hashTable[i][0] << ":" << setw(2) << hashTable[i][1];      }      cout << endl;      cout << "adding more" << endl;      for(int i = hashSize; i < 15; i++){          if(addHash(hashTable, hashSize, a[i], i) == -1) cout << "Table is full" << endl;      }      for(int i = 0; i < hashSize; i++){          cout << setw(4) << hashTable[i][0] << ":" << setw(2) << hashTable[i][1];      }      cout << endl;      cout << "pick what to delete" << endl;      int delId;      cin >> delId;      if(deleteHash(hashTable, hashSize, delId) >= 0) cout << "deleted" << endl;      else cout << "not found" << endl;      cout << "pick what to find" << endl;      int findId;      cin >> findId;      int findIndex = findByHash(hashTable, hashSize, findId);      if(findIndex >= 0) cout << "found at index " << findIndex << endl;      else cout << "not found" << endl;      return;  }  #ifndef \_\_BINFILE\_H\_\_  #define \_\_BINFILE\_H\_\_  #include <iostream>  #include <fstream>  #include <time.h>  #include <string>  #include <vector>  #include <cstring>  using namespace std;  struct Bank{      int id;      char name[61];      char city[61];      bool commercialOrState; //true - commercial, false - state  };  int toBinary(string fileName, string fileNameBin);//read from text file and write to binary file  int getBank(string fileNameBin, Bank &bank, int ind);//get bank by id from binary file  int deleteBankById(string fileNameBin, int id);  int deleteBankByIndex(string fileNameBin, int ind);  int addBank(string fileNameBin, const Bank &bank, int ind = -1);  void testBinF();  #endif  #include "binFile.h"  int toBinary(string fileName, string fileNameBin){//read from text file and write to binary file      ifstream fin("../../" + fileName);      fstream fout("../../" + fileNameBin, ios::out|ios::binary|ios::trunc);      if(!fin.is\_open() || !fout.is\_open()){          cout << "file doesn't exist" << endl;          fin.close();          fout.close();          return 1;      }      while(fin.good()){          Bank bank;          vector<string> tokens;          for (string each; getline(fin, each, ';'); tokens.push\_back(each));          for(int i = 0; i < tokens.size(); i++){              //cout << tokens.size() << " " << tokens[i] << endl;              if(i%4 == 0) bank.id = stoi(tokens[i]);              else if(i%4 == 1) strcpy(bank.name, tokens[i].c\_str());              else if(i%4 == 2) strcpy(bank.city, tokens[i].c\_str());              else if(i%4 == 3){                  bank.commercialOrState = (tokens[i] == "true") ? true : false;                  fout.write((char\*)&bank, sizeof(bank));              }          }        }      fin.close();      fout.close();      return 0;  }  int addBank(string fileNameBin, const Bank &bank, int ind){      fstream fout("../../" + fileNameBin, ios::in|ios::out|ios::binary);      if(!fout.is\_open()){          cout << "file doesn't exist" << endl;          fout.close();          return 1;      }      if(ind == -1){          fout.seekp(0, ios::end);          fout.write((char\*)&bank, sizeof(bank));          fout.close();          return 0;      }      fout.seekp((ind)\*sizeof(bank), ios::beg);      // if(fout.write((char\*)&bank, sizeof(bank))){      //  fout.close();      //  return 0;      // }      fout.write((char\*)&bank, sizeof(bank));      fout.close();      return 0;  }  int getBank(string fileNameBin, Bank &bank, int ind){//get bank by index from binary file starting from 1      //returns 0 if bank was found, 1 if file doesn't exist, 2 if bank wasn't found      fstream fin("../../" + fileNameBin, ios::in|ios::binary);      if(!fin.is\_open()){          cout << "file doesn't exist" << endl;          fin.close();          return 1;      }        fin.seekg(sizeof(Bank)\*(ind), ios::beg);      Bank readBank;      fin.read((char\*)&readBank, sizeof(Bank));      if(!fin.good()){          return 2;      }      bank = readBank;      fin.close();      return 0;  }  int deleteBankById(string fileNameBin, int id){//delete bank by id from binary file      //returns 0 if bank was deleted, 1 if file doesn't exist, 2 if bank wasn't found      fstream fin("../../" + fileNameBin, ios::in|ios::out|ios::binary);      if(!fin.is\_open()){          cout << "file doesn't exist" << endl;          fin.close();          return 1;      }      Bank bank;      fin.seekg(0, ios::beg);      while(fin.read((char\*)&bank, sizeof(bank))){          if(bank.id == id){              bank.id = -1; //means that bank was deleted              fin.seekp(-sizeof(bank), ios::cur);              fin.write((char\*)&bank, sizeof(bank));              fin.close();              return 0;          }      }      fin.close();      return 2;  }  int deleteBankByIndex(string fileNameBin, int ind){      //returns 0 if bank was deleted, 1 if file doesn't exist, 2 if bank wasn't found      fstream fin("../../" + fileNameBin, ios::in|ios::out|ios::binary);      if(!fin.is\_open()){          cout << "file doesn't exist" << endl;          fin.close();          return 1;      }      fin.seekg(sizeof(Bank)\*(ind), ios::beg);      Bank readBank;      fin.read((char\*)&readBank, sizeof(Bank));      if(!fin.good()){          return 2;      }      readBank.id = -1; //means that bank was deleted      fin.seekp(-sizeof(readBank), ios::cur);      fin.write((char\*)&readBank, sizeof(readBank));      fin.close();      return 0;  }  void testBinF(){      //test toBinary      toBinary("bfi.txt", "bfb.txt");      cout << "binary file contents:" << endl;      for(int i = 1; i <= 4; i++){          Bank bank;          getBank("bfb.txt", bank, i-1);          cout << bank.id << " " << bank.name << " " << bank.city << " " << bank.commercialOrState << endl;      }      //test getBank      cout << "insert index from 1 to pick bank" << endl;      int ind;      cin >> ind;      Bank bank;      getBank("bfb.txt", bank, ind-1);      cout << bank.id << " " << bank.name << " " << bank.city << " " << bank.commercialOrState << endl;      //test deleteBank by id      cout << "insert id to delete bank" << endl;      int id;      cin >> id;      deleteBankById("bfb.txt", id);      cout << "binary file contents:" << endl;      for(int i = 1; i <= 4; i++){          Bank bank;          getBank("bfb.txt", bank, i-1);          cout << bank.id << " " << bank.name << " " << bank.city << " " << bank.commercialOrState << endl;      }      //test deleteBank by index      cout << "insert index to delete bank" << endl;      cin >> ind;      deleteBankByIndex("bfb.txt", ind-1);      cout << "binary file contents:" << endl;      for(int i = 1; i <= 4; i++){          Bank bank;          getBank("bfb.txt", bank, i-1);          cout << bank.id << " " << bank.name << " " << bank.city << " " << bank.commercialOrState << endl;      }      //test adding      bank.id = 1337;      string tmpName = "test bank";      string tmpCity = "test city";      strcpy(bank.name, tmpName.c\_str());      strcpy(bank.city, tmpCity.c\_str());      bank.commercialOrState = true;      cout << endl << "bank that is being added " << bank.id << ";" << bank.name << ";" << bank.city << ";" << bank.commercialOrState << endl;      cout << "adding in the end and to pos 2" << endl;      addBank("bfb.txt", bank, 2-1);      addBank("bfb.txt", bank);      cout << "binary file contents:" << endl;      for(int i = 1; i <= 5; i++){          Bank bank;          getBank("bfb.txt", bank, i-1);          cout << bank.id << " " << bank.name << " " << bank.city << " " << bank.commercialOrState << endl;      }  }  #ifndef \_\_HASH\_BIN\_FILE\_H\_\_  #define \_\_HASH\_BIN\_FILE\_H\_\_  #include "hashFunctions.h"  #include "binFile.h"  using namespace std;  int hashRecord(string fileName, int ind, vector<vector<int>> &hashTable, int &tableSize, int &hashElements, int &deletedElements);  int hashFile(string fileName, vector<vector<int>> &hashTable, int &tableSize, int &hashElements, int &deletedElements);  int deleteRecordById(string fileName, int id, vector<vector<int>> &hashTable, int &tableSize, int &deletedElements);  Bank getRecordById(string fileName, int id, vector<vector<int>> &hashTable, int &tableSize);  #endif  #include "hashBinFile.h"  int hashRecord(string fileName, int ind, vector<vector<int>> &hashTable, int &tableSize, int &hashElements, int &deletedElements){      fstream fin("../../" + fileName, ios::in|ios::binary);      if(!fin.is\_open()){          cout << "file doesn't exist" << endl;          fin.close();          return 1;      }      fin.seekg(sizeof(Bank)\*(ind), ios::beg);      Bank readBank;      fin.read((char\*)&readBank, sizeof(Bank));      if(!fin.good()){          return 2; //index out of bounds      }      int id = readBank.id;      if(hashElements > (int)(0.75 \* tableSize)){          resizeHash(hashTable, tableSize);          hashElements -= deletedElements;          deletedElements = 0;      }      hashElements++;      return addHash(hashTable, tableSize, id, ind);  }  int hashFile(string fileName, vector<vector<int>> &hashTable, int &tableSize, int &hashElements, int &deletedElements){      //returns 0 if file was hashed, 1 if file doesn't exist      fstream fin("../../" + fileName, ios::in|ios::binary);      if(!fin.is\_open()){          cout << "file doesn't exist" << endl;          fin.close();          return 1;      }      Bank bank;      fin.seekg(0, ios::beg);      int ind = 0;      while(fin.read((char\*)&bank, sizeof(bank))){          int id = bank.id;          if(hashElements > (int)(0.75 \* tableSize)){              resizeHash(hashTable, tableSize);              hashElements -= deletedElements;              deletedElements = 0;          }          hashElements++;          addHash(hashTable, tableSize, id, ind);          ind++;      }      fin.close();      return 0;  }  int deleteRecordById(string fileName, int id, vector<vector<int>> &hashTable, int &tableSize, int &deletedElements){      //returns 0 if bank was deleted, 1 if file doesn't exist, 2 if bank wasn't found      fstream fin("../../" + fileName, ios::in|ios::out|ios::binary);      if(!fin.is\_open()){          cout << "file doesn't exist" << endl;          fin.close();          return 1;      }      int ind = findByHash(hashTable, tableSize, id);      if(ind == -1){          return 2;      }        deleteHash(hashTable, tableSize, id);      deletedElements++;      deleteBankByIndex(fileName, ind);      fin.close();      return 0;  }  Bank getRecordById(string fileName, int id, vector<vector<int>> &hashTable, int &tableSize){      fstream fin("../../" + fileName, ios::in|ios::binary);      if(!fin.is\_open()){          cout << "file doesn't exist" << endl;          fin.close();          Bank b; b.id = -2; return b; // id == -2 is file not found      }      int ind = findByHash(hashTable, tableSize, id);      if(ind == -1){          Bank b; b.id = -1; return b;      }      Bank bank;      fin.seekg(sizeof(Bank)\*(ind), ios::beg);      fin.read((char\*)&bank, sizeof(Bank));        fin.close();      return bank;  }  #include <iostream>  #include <iomanip>  #include <fstream>  #include <time.h>  #include <string>  #include <vector>  #include <chrono>  #include "libraries/hashBinFile.h"  using namespace std;  int main(){      //testHashT();      //testBinF();      // fstream fout("../../bigFile.txt", ios::out|ios::binary);      // Bank bank;      // for(int i = 0; i < 200000; i++){      //  bank.id = (i + 250) % 200000 + 1;      //  string tmpName = "test bank" + to\_string(i);      //  string tmpCity = "test city";      //  strcpy(bank.name, tmpName.c\_str());      //  strcpy(bank.city, tmpCity.c\_str());      //  bank.commercialOrState = true;      //  if((i / 100) % 2 == 0){      //      bank.commercialOrState = false;      //  }      //  fout.write((char\*)&bank, sizeof(Bank));      // }      // fout.close();      cout << "is it big file for testing time? (y/n) (n by default)" << endl;      char bigFile;      cin >> bigFile;      cin.ignore(INT\_MAX, '\n');      if(bigFile != 'y' && bigFile != 'n') bigFile = 'n';      if(bigFile == 'n'){          cout << "Enter file name for initial text file (without .txt): " << endl;          string fileName;          getline(cin, fileName);          fileName += ".txt";          ifstream fin("../../" + fileName);          if(!fin.is\_open()){              cout << "Error opening file" << endl;              return 1;          }          fin.close();          cout << "Enter file name for binary file (without .bin): " << endl;          string fileNameBin;          getline(cin, fileNameBin);          fileNameBin += ".txt";          ifstream finBin("../../" + fileNameBin);          if(!finBin.is\_open()){              cout << "Error opening file" << endl;              return 1;          }          finBin.close();          toBinary(fileName, fileNameBin);          cout << "binary file contents: " << endl;          Bank bank;          int iter = 0;          while(getBank(fileNameBin, bank, iter) == 0){              //print bank              cout << "id: " << bank.id;              cout << ", name: " << bank.name;              cout << ", city: " << bank.city;              cout << ", commercialOrState: " << bank.commercialOrState << endl;              iter++;          }          cout << endl;          int simpleNums[6] = {13, 19, 53, 601, 12289, 1046527}; //simple numbers for hash table size the last one is 1 046 527 for 1 million test          int hashElements = 0; //this is the way to store how much elements we have stored and deleted (kind of a hack, but it's fiiiiiine)          int deletedElements = 0;          int hashSize = simpleNums[0];          // hash table: (id, index)          vector<vector<int>> hashTable(hashSize, {-1, -1});          hashFile(fileNameBin, hashTable, hashSize, hashElements, deletedElements);          cout << "hash table contents: " << endl;          for(int i = 0; i < hashSize; i++){              cout << "hashTable[" << i << "]: " << hashTable[i][0] << ", " << hashTable[i][1] << endl;          }          cout << "enter id to get bank: " << endl;          int id;          cin >> id;          bank = getRecordById(fileNameBin, id, hashTable, hashSize);          if(bank.id == -1){              cout << "no bank with such id" << endl;          }          else{              cout << "id: " << bank.id;              cout << ", name: " << bank.name;              cout << ", city: " << bank.city;              cout << ", commercialOrState: " << bank.commercialOrState << endl;          }          cout << "enter id to delete bank: " << endl;          cin >> id;          deleteRecordById(fileNameBin, id, hashTable, hashSize, deletedElements);            cout << "binary file contents: " << endl;          iter = 0;          while(getBank(fileNameBin, bank, iter) == 0){              //print bank              cout << "id: " << bank.id;              cout << ", name: " << bank.name;              cout << ", city: " << bank.city;              cout << ", commercialOrState: " << bank.commercialOrState << endl;              iter++;          }      }      else{          cout << "Enter file name for binary file (without .bin): " << endl;          string fileNameBin;          getline(cin, fileNameBin);          fileNameBin += ".txt";          ifstream finBin("../../" + fileNameBin);          if(!finBin.is\_open()){              cout << "Error opening file" << endl;              return 1;          }          finBin.close();          int simpleNums[6] = {13, 19, 53, 601, 12289, 1046527}; //simple numbers for hash table size the last one is 1 046 527 for 1 million test          int hashElements = 0; //this is the way to store how much elements we have stored and deleted (kind of a hack, but it's fiiiiiine)          int deletedElements = 0;          int hashSize = simpleNums[0];          // hash table: (id, index)          vector<vector<int>> hashTable(hashSize, {-1, -1});          auto hashingStart = chrono::high\_resolution\_clock::now();          hashFile(fileNameBin, hashTable, hashSize, hashElements, deletedElements);          auto hashingEnd = chrono::high\_resolution\_clock::now();          auto hashingTime = chrono::duration\_cast<chrono::milliseconds>(hashingEnd - hashingStart);          cout << "hashing time (it probably did some rehashing because hashTable started from capcity of 13 elements): " << hashingTime.count() << " ms" << endl;          for(int i = 0; i < 3; i++){              cout << "Enter id to get bank (from 1 to 200000 (1-250 are the last ones)): " << endl;              int id;              cin >> id;              Bank bank;              auto start = chrono::high\_resolution\_clock::now();              bank = getRecordById(fileNameBin, id, hashTable, hashSize);              auto end = chrono::high\_resolution\_clock::now();              auto duration = chrono::duration\_cast<chrono::microseconds>(end - start);              cout << "id: " << bank.id;              cout << ", name: " << bank.name;              cout << ", city: " << bank.city;              cout << ", commercialOrState: " << bank.commercialOrState << endl;              cout << "Time taken by function: " << duration.count() << " microseconds" << endl;          }      }      return 0;  } |