|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | |  |  |  | | 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-33-22 | Шило Ю.С. |
| Принял преподаватель | Филатов А.С. |

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

Москва 2023

# **Цель работы**

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

1. **Постановка задачи**
2. Разработать приложение, которое использует хеш-таблицу для организации прямого доступа к записям двоичного файла, реализованного в практической работе 2.
3. Создать приложение и включить в него три заголовочных файла: управление хеш-таблицей, управление двоичным файлом (практическая работа 2), управление двоичным файлом посредством хеш-таблицы.
4. 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.
5. 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.
6. 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.
7. Управление файлом посредством хеш-таблицы
8. 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.
9. 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 (от 500).
   9. Определите время чтения записи с заданным ключом: для первой записи файла, для средней и последней. Проверьте, что время доступа для всех записей одинаково. Сравнить с временем линейного поиска тех же записей в файле.
10. Составить отчет.
11. Ответы на вопросы
    1. Расскажите о назначении хеш-фунции.
    2. Что такое коллизия?
    3. Что такое «открытый адрес» по отношению к хеш-таблице?
    4. Как в хеш-таблице с открытым адресом реализуется коллизия?
    5. Какая проблема, может возникнуть после удаления элемента из хеш-таблицы с открытым адресом и как ее устранить?
    6. Что определяет коэффициент нагрузки в хеш-таблице?
    7. Что такое «первичный кластер» в таблице с открытым адресом?
    8. Как реализуется двойное хеширование?

Таблица 1. Варианта задания

|  |  |  |
| --- | --- | --- |
| № | Тип хеш-таблицы (метод разрешения коллизии) | Структура записи двоичного файла |
| 29 | 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-таблицы являются одной из наиболее эффективных структур данных для хранения и быстрого поиска информации. Они основаны на использовании хеш-функций, которые преобразуют входные данные в уникальные числовые значения, называемые хешами. Хеш-таблицы используют эти хеши для индексации и организации данных.

Одним из основных преимуществ хеш-таблиц является их высокая скорость поиска. При правильном выборе хеш-функции и эффективной обработке коллизий (ситуаций, когда двум различным ключам соответствует один и тот же хеш), хеш-таблицы позволяют выполнять операции поиска, вставки и удаления данных за постоянное время O(1).

Хеш-таблицы также находят широкое применение при поиске данных в других структурах данных, таких как файлы. Вместо того чтобы хранить все данные в оперативной памяти, которая может быть ограничена по размеру, можно создать хеш-таблицу, которая будет содержать хеши и ссылки на соответствующие данные в файле.

При поиске данных в файле, хеш-таблица позволяет быстро определить местоположение данных, минимизируя количество операций чтения из файла. Вместо того чтобы просматривать файл последовательно, можно использовать хеш-таблицу для поиска хеша ключа и получения соответствующей ссылки на данные в файле. Это существенно сокращает время доступа к данным и повышает производительность.

Разработка хеш-таблиц и их применение при поиске данных в других структурах данных, таких как файлы, являются важными навыками для программистов, особенно при работе с большими объемами данных. Они позволяют эффективно организовывать и обрабатывать информацию, снижая нагрузку на систему и ускоряя выполнение операций поиска данных.

**Задание**

1. Чтобы пользователь мог удобнее взаимодействовать с нашей программы был написан интерфейс. Мы запрашиваем у пользователя цифру, которая соответствует определенной функции после ввода подходящего значения выполняем эту функцию иначе говорим пользователю, что были введены некорректные данные.

|  |
| --- |
| int main() {  Bin bin;  HashBin hashBin;  HashTable hashTable;  int i = 0;  int key = 0;  while (1){  cout << "Enter the number of the operation" << endl  << "1 -> testBinT || 2 -> testHashT" << endl  << "3 -> CreateBin || 4 -> PrintBin" << endl  << "5 -> LinkingHash || 6 -> Deleting" << endl  << "7 -> Search || 8 -> DisplayAll" << endl  << "-1 -> Exit" << endl;  cin >> i;  cout << "\n\n";  if (i == -1) return 0;  else if (i == 1) bin.testBinT();  else if (i == 2) hashTable.testHeshT();  else if (i == 3) bin.TextToBinary("text.txt", "bin.txt");  else if (i == 4) bin.PrintBinary("bin.txt");  else if (i == 5) hashBin.Linking("bin.txt");  else if (i == 6){  cout << "Enter the number of the key to be deleted" << endl;  cin >> key;  hashBin.Deleting("bin.txt", key);  }  else if (i == 7){  cout << "Enter the key number to search for" << endl;  cin >> key;  BankAccount temp = hashBin.Search(key);  cout << "Your key is " << key << endl  << " || AccountNumber || " << temp.accountNumber  << " || Name || " << temp.fullName  << " || Address || " << temp.address << endl;  }  else if (i == 8) hashBin.Display("bin.txt");  else cout << "Wrong number" << endl;  cout << "\n\n";  }  } |

1. Для вывода хэш таблицы в консоль мы вызываем функцию PrintHash.

|  |
| --- |
| void PrintHash() {  cout << "========================[ Print hash ]========================" << endl;  for (int i = 0; i < table.size(); ++i) {  if (table[i].accountNumber != 0) {  std::cout << "Hash " << i << ": (" << table[i].accountNumber << ", " << table[i].fullName << ", " << table[i].address << ")" << std::endl;  }  } } |

1. Для добавления элемента в хэш таблицу была написана функция insert в качестве входных параметров она принимает объект нашей структуры, которую мы должны добавить.

|  |
| --- |
| void Insert(BankAccount account) {  if(size == table.size()-1){  resizeAndRehash();  }   int index = hash1(account.accountNumber);  int step = hash2(account.accountNumber);   while (table[index].accountNumber != 0) {  if (table[index].accountNumber == account.accountNumber) {  std::cout << "An account with this number already exists. Data update." << std::endl;  table[index] = account;  return;  }  index = (index + step) % table.size();  }   table[index] = account;  size++; } |

1. Для удаления элемента из хэш таблицы была написана функция Remove в качестве входных параметров она принимает ключ объекта, который требуется удалить.

|  |
| --- |
| void Remove(int accountNumber) {  int index = hash1(accountNumber);  int step = hash2(accountNumber);   while (table[index].accountNumber != 0) {  if (table[index].accountNumber == accountNumber) {  table[index] = BankAccount(0, "", ""); // Очищаем ячейку  return;  }  index = (index + step) % table.size();  }  size--; } |

1. Что бы проверить работоспособность нашей программы была написана функция testHeshT.

|  |
| --- |
| void testHeshT() {  HashTable table;  BankAccount bankAccount;  bankAccount.accountNumber = 1;  table.Insert(bankAccount);  bankAccount.accountNumber += 1;  table.Insert(bankAccount);  bankAccount.accountNumber += 1;  table.Insert(bankAccount);  bankAccount.accountNumber += 1;  table.Insert(bankAccount);  table.PrintHash();  cout << "Add elem" << endl;  bankAccount.accountNumber += 1;  table.Insert(bankAccount);  table.PrintHash();  cout << "delete elem with key 4" << endl;  table.Remove(4);  table.PrintHash();  bankAccount = table.Search(3);  std::cout << "Hash " << 4 << ": (" << bankAccount.accountNumber << ", " << bankAccount.fullName << ", " << bankAccount.address << ")" << std::endl; } |

1. Для связи бинарного файла с нашей хэш табличкой была написана функция Linking. В качестве входного параметра она принимает строковую переменную, которая будет отвечать за название нашего бинарного файла.

|  |
| --- |
| void Linking(string BinaryFilename) {  ifstream InputFile(BinaryFilename, ios::binary);   if (!InputFile) { cout << "Failed to open files\n"; return; }   BankAccount bank;  while (InputFile.read(reinterpret\_cast<char\*> (&bank), sizeof(BankAccount))){  hashTable.Insert(bank);  }   cout << "The bonding was successful!" << endl;  InputFile.close(); } |

1. Для удаления элемента из бинарного файла и хэш табличкой была написана функция Deleting. В качестве входного параметра она принимает строковую переменную, которая будет отвечать за название нашего бинарного файла, и ключ, отвечающий за объект, который мы должны удалить.

|  |
| --- |
| void Deleting(string BinaryFilename, int key){  bin.Remove(BinaryFilename, key);  hashTable.Remove(key); } |

1. Для поиска объекта по ключу была создана функция Search.

|  |
| --- |
| BankAccount Search(int key){  return hashTable.Search(key); } |

1. Для выводы текста из файла и текста из хэш таблички была создана функция Display.

|  |
| --- |
| void Display(string BinaryFilename){  bin.PrintBinary(BinaryFilename);  hashTable.PrintHash(); } |

Интерфейс программы, которая выполняет задание 1 представлена на рисунке 1.

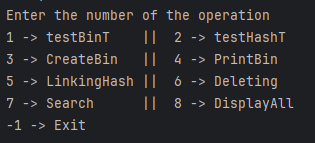


Рисунок 1. Интерфейс программы, выполняющей задание

**Тестирование**

Протестируем программу, для проверки методов созданных для хэш таблички введем 1. В консоли мы должны увидеть то, что хэш таблица выводиться, затем в нее добавляться запись и после этого удаляется. Текст выведенный в консоль предоставлен на рисунке 3.

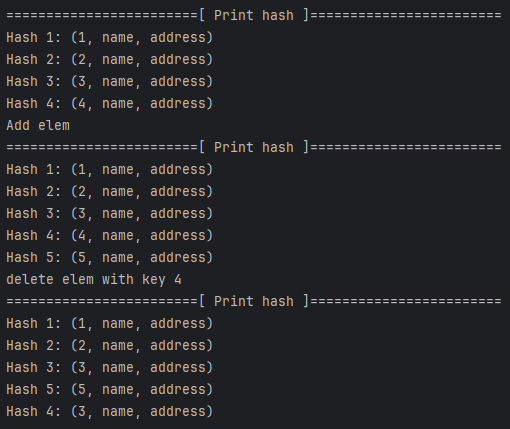


Рисунок 3. Результат тестирования и вывода программы

Теперь проверим время, которое потребуется для поиска элемента, находящегося в начале в середине и в конце для линейного поиска и поиска через хэш таблицу. Временные замеры предоставлены в таблице 1.

Таблица 1. Тестирование поиска

|  |  |  |
| --- | --- | --- |
| Позиция | Линейный поиск (ns) | Поиск через хэш таблицу (ns) |
| Начало | 113200 | 100 |
| Середина | 249800 | 100 |
| Конец | 256700 | 100 |

**Вывод**

В процессе выполнения данной практической работы я успешно овладел навыками работы с файловыми потоками в языке программирования C++ для управления текстовыми и двоичными файлами. Эти навыки позволят мне эффективно читать, записывать и обрабатывать данные из файлов, что является важной частью разработки программ, работающих с файловой системой.

# **Исходный код программы**

Исходный код файла для выполнения задания - main.cpp

|  |
| --- |
| #include "Bin.h" #include "Hash.h" #include "BinHash.h"  int main() {  Bin bin;  HashBin hashBin;  HashTable hashTable;   bin.Search(1);  bin.Search(250);  bin.Search(500);   hashBin.Linking("bin.txt");  hashBin.TimeS(1);  hashBin.TimeS(250);  hashBin.TimeS(500);    int i = 0;  int key = 0;   while (1){  cout << "Enter the number of the operation" << endl  << "1 -> testBinT || 2 -> testHashT" << endl  << "3 -> CreateBin || 4 -> PrintBin" << endl  << "5 -> LinkingHash || 6 -> Deleting" << endl  << "7 -> Search || 8 -> DisplayAll" << endl  << "-1 -> Exit" << endl;  cin >> i;   cout << "\n\n";  if (i == -1) return 0;  else if (i == 1) bin.testBinT();  else if (i == 2) hashTable.testHeshT();  else if (i == 3) bin.TextToBinary("text.txt", "bin.txt");  else if (i == 4) bin.PrintBinary("bin.txt");  else if (i == 5) hashBin.Linking("bin.txt");  else if (i == 6){  cout << "Enter the number of the key to be deleted" << endl;  cin >> key;  hashBin.Deleting("bin.txt", key);  }  else if (i == 7){  cout << "Enter the key number to search for" << endl;  cin >> key;  BankAccount temp = hashBin.Search(key);  cout << "Your key is " << key << endl  << " || AccountNumber || " << temp.accountNumber  << " || Name || " << temp.fullName  << " || Address || " << temp.address << endl;  }  else if (i == 8) hashBin.Display("bin.txt");  else cout << "Wrong number" << endl;   cout << "\n\n";  } } |

Исходный код файла для выполнения задания Bin.h

|  |
| --- |
| #ifndef ABOBA\_BIN\_H  #define ABOBA\_BIN\_H  #include <iostream>  #include <string>  #include <fstream>  #include <chrono>  using namespace std;  struct BankAccount {  int accountNumber;  char fullName[256] = {'n', 'a', 'm', 'e'};  char address[256] = {'a', 'd', 'd', 'r', 'e', 's', 's'};  };  class Bin {  public:  // Done  void TextToBinary(string TextFilename, string BinaryFilename) {  ifstream InputFile(TextFilename);  ofstream OutFile(BinaryFilename, ios::binary);  if (!InputFile || !OutFile) { cout << "Failed to open files\n"; return; }  BankAccount bank;  while (InputFile >> bank.accountNumber >> bank.fullName >> bank.address)  OutFile.write(reinterpret\_cast<const char \*>(&bank), sizeof(BankAccount));  cout << "Text data converted to binary successfully!" << endl;  InputFile.close();  OutFile.close();  }  // Done  void PrintBinary(string BinaryFilename) {  ifstream InputFile(BinaryFilename, ios::binary);  if (!InputFile) { cout << "Failed to open files\n"; return; }  BankAccount bank;  cout << "========================[ Print binary ]========================" << endl;  while (InputFile.read(reinterpret\_cast<char\*> (&bank), sizeof(BankAccount)))  cout << " Code:: " << bank.accountNumber << " Name:: " << bank.fullName << " Address:: " << bank.address << endl;  InputFile.close();  }  // Done  void Remove (string BinaryFilename, int Code){  fstream BinaryFile(BinaryFilename, ios::in | ios::binary);  ofstream temp("temp.txt", ios::out | ios::binary);  BankAccount bank;  while (BinaryFile.read(reinterpret\_cast<char\*>(&bank), sizeof(bank)))  if (bank.accountNumber != Code)  temp.write(reinterpret\_cast<char\*>(&bank), sizeof(bank));  BinaryFile.close();  remove(BinaryFilename.c\_str());  temp.close();  rename("temp.txt", BinaryFilename.c\_str());  }  void Search (int Code){  fstream BinaryFile("text.txt", ios::binary);  BankAccount bank;  auto begin = std::chrono::steady\_clock::now();  cout << Code << "\t| ";  while (BinaryFile.read(reinterpret\_cast<char\*> (&bank), sizeof(BankAccount)))  if (bank.accountNumber == Code) {  cout << " Code:: " << bank.accountNumber << " Name:: " << bank.fullName << " Address:: " << bank.address << endl;  break;  }  auto end = std::chrono::steady\_clock::now();  auto elapsed\_ms = std::chrono::duration\_cast<std::chrono::nanoseconds>(end - begin);  std::cout << "The time: " << elapsed\_ms.count() << " ns\n";  BinaryFile.close();  }  // Сделать BinTest  void testBinT() {  Bin bin;  bin.TextToBinary("text-test.txt", "bin-test.txt");  bin.PrintBinary("bin=test.txt");  bin.Remove("bin-test.txt", 5);  bin.PrintBinary("bin-test.txt");  }  };  #endif //ABOBA\_BIN\_H |
|  |

Исходный код файла для выполнения задания BinHash.h

|  |
| --- |
| #ifndef ABOBA\_BINHASH\_H  #define ABOBA\_BINHASH\_H  #include "Bin.h"  #include "Hash.h"  class HashBin{  Bin bin;  HashTable hashTable;  public:  void Linking(string BinaryFilename) {  ifstream InputFile(BinaryFilename, ios::binary);  if (!InputFile) { cout << "Failed to open files\n"; return; }  BankAccount bank;  while (InputFile.read(reinterpret\_cast<char\*> (&bank), sizeof(BankAccount))){  hashTable.Insert(bank);  }  cout << "The bonding was successful!" << endl;  InputFile.close();  }  void Deleting(string BinaryFilename, int key){  bin.Remove(BinaryFilename, key);  hashTable.Remove(key);  }  BankAccount Search(int key){  return hashTable.Search(key);  }  void TimeS(int code){  hashTable.TimeS(code);  }  void Display(string BinaryFilename){  bin.PrintBinary(BinaryFilename);  hashTable.PrintHash();  }  };  #endif //ABOBA\_BINHASH\_H |

Исходный код файла для выполнения задания Hash.h

|  |
| --- |
| #ifndef ABOBA\_HASH\_H  #define ABOBA\_HASH\_H  #include <iostream>  #include <vector>  #include "Bin.h"  class HashTable {  private:  static const int BASE\_TABLE = 4;  std::vector<BankAccount> table;  int size = 0;  int hash1(int accountNumber) { return accountNumber % table.size(); }  int hash2(int accountNumber) { return (accountNumber % (table.size() - 1)) + 1; }  void resizeAndRehash() {  int newTableSize = table.size() \* 2;  std::vector<BankAccount> newTable(newTableSize);  for (const BankAccount& account : table) {  if (account.accountNumber != 0) {  int index = hash1(account.accountNumber);  int step = hash2(account.accountNumber);  while (newTable[index].accountNumber != 0) {  index = (index + step) % newTableSize;  }  newTable[index] = account;  }  }  table = newTable;  }  public:  // Done  HashTable(){  table = std::vector<BankAccount>(BASE\_TABLE);  }  // Done  void Insert(BankAccount account) {  if(size == table.size()-1){  resizeAndRehash();  }  int index = hash1(account.accountNumber);  int step = hash2(account.accountNumber);  while (table[index].accountNumber != 0) {  if (table[index].accountNumber == account.accountNumber) {  std::cout << "An account with this number already exists. Data update." << std::endl;  table[index] = account;  return;  }  index = (index + step) % table.size();  }  table[index] = account;  size++;  }  // Done  BankAccount Search(int accountNumber) {  int index = hash1(accountNumber);  int step = hash2(accountNumber);  while (table[index].accountNumber != 0) {  if (table[index].accountNumber == accountNumber) {  return table[index];  }  index = (index + step) % table.size();  }  return BankAccount(0, "", ""); // Счет не найден  }  void TimeS(int accountNumber){  int index = hash1(accountNumber);  int step = hash2(accountNumber);  auto begin = std::chrono::steady\_clock::now();  while (table[index].accountNumber != 0) {  if (table[index].accountNumber == accountNumber) {  auto end = std::chrono::steady\_clock::now();  auto elapsed\_ms = std::chrono::duration\_cast<std::chrono::nanoseconds>(end - begin);  std::cout << "The time: " << elapsed\_ms.count() << " ns\n";  }  index = (index + step) % table.size();  }  }  // Done  void Remove(int accountNumber) {  int index = hash1(accountNumber);  int step = hash2(accountNumber);  while (table[index].accountNumber != 0) {  if (table[index].accountNumber == accountNumber) {  table[index] = BankAccount(0, "", ""); // Очищаем ячейку  return;  }  index = (index + step) % table.size();  }  size--;  }  // Done  void PrintHash() {  cout << "========================[ Print hash ]========================" << endl;  for (int i = 0; i < table.size(); ++i) {  if (table[i].accountNumber != 0) {  std::cout << "Hash " << i << ": (" << table[i].accountNumber << ", " << table[i].fullName << ", " << table[i].address << ")" << std::endl;  }  }  }  // Сделать HashTest  void testHeshT() {  HashTable table;  BankAccount bankAccount;  bankAccount.accountNumber = 1;  table.Insert(bankAccount);  bankAccount.accountNumber += 1;  table.Insert(bankAccount);  bankAccount.accountNumber += 1;  table.Insert(bankAccount);  bankAccount.accountNumber += 1;  table.Insert(bankAccount);  table.PrintHash();  cout << "Add elem" << endl;  bankAccount.accountNumber += 1;  table.Insert(bankAccount);  table.PrintHash();  cout << "delete elem with key 4" << endl;  table.Remove(4);  table.PrintHash();  bankAccount = table.Search(3);  std::cout << "Hash " << 4 << ": (" << bankAccount.accountNumber << ", " << bankAccount.fullName << ", " << bankAccount.address << ")" << std::endl;  }  };  #endif //ABOBA\_HASH\_H |