|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | |  |  |  | | МИНОБРНАУКИ РОССИИ | | | | Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** | | |   Институт Информационных технологий | |
|  | |
| Кафедра Математического обеспечения и стандартизации информационных технологий | |
|  | |
|  | |

|  |  |
| --- | --- |
| **ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ № 3** | |
| **по дисциплине** | |
| **«**Структуры и алгоритмы обработки данных**»**  **Тема: «Применение хеш-таблицы для поиска данных в двоичном файле с записями фиксированной длины.»** | |
|  | |
| Выполнил студент группы ИКБО-10-21 | Черномуров С.А. |
| 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**

**Задание 1.**

1. Разработать приложение, которое использует хеш-таблицу для организации прямого доступа к записям двоичного файла, реализованного в практической работе 2.
2. Создать приложение и включить в него три заголовочных файла: управление хеш-таблицей, управление двоичным файлом (практическая работа 2), управление двоичным файлом посредством хеш-таблицы.
3. Для обеспечения прямого доступа к записи в файле элемент хеш-таблицы должен включать обязательные поля: ключ записи в файле, номер записи с этим ключом в файле. Элемент может содержать другие поля, требующиеся методу (указанному в вашем варианте), разрешающему коллизию.
4. Управление хеш-таблицей
   1. Определить структуру элемента хеш-таблицы и структуру хеш-таблицы в соответствии с методом разрешения коллизии, указанном в варианте. Определения разместить в соответствующем заголовочном файле. Все операции управления хеш-таблицей размещать в этом заголовочном файле.
   2. Тестирование операций выполнять в функции main приложения по мере их реализации.
   3. После тестирования всех операций, создать в заголовочном файле функцию с именем testHeshT переместить в нее содержание функции main, проверить, что приложение выполняется.
   4. Разработать операции по управлению хеш-таблицей.
   5. Разработать хеш-функцию (метод определить самостоятельно), выполнить ее тестирование, убедиться, что хеш (индекс элемента таблицы) формируется верно.
   6. Разработать операции: вставить ключ в таблицу, удалить ключ из таблицы, найти ключ в таблице, рехешировать таблицу. Каждую операцию тестируйте по мере ее реализации.
   7. Подготовить тесты (последовательность значений ключей), обеспечивающие:
      1. вставку ключа без коллизии
      2. вставку ключа и разрешение коллизии
      3. вставку ключа с последующим рехешированием
      4. удаление ключа из таблицы
      5. поиск ключа в таблице. Для метода с открытым адресом подготовить тест для поиска ключа, который размещен в таблице после удаленного ключа, с одним значением хеша для этих ключей
   8. Выполнить тестирование операций управления хеш-таблицей. При тестировании операции вставки ключа в таблицу предусмотрите вывод списка индексов, которые формируются при вставке элементов в таблицу.
5. Управление двоичным файлом
   1. Операции управления двоичным файлом: создание двоичного файла из текстового, добавить запись в двоичный файл, удалить запись с заданным ключом из файла, прочитать запись файла по заданному номеру записи.
   2. Структура записи двоичного файла и все операции по управлению файлом должны быть размещены в соответствующем заголовочном файле.
   3. Выполнить тестирование операций в main приложения, и содержание функции main переместить в соответствующую функцию заголовочного файла с именем testBinF.
6. Управление файлом посредством хеш-таблицы
7. В заголовочный файл управления файлом посредством хеш-таблицы подключить заголовочные файлы: управления хеш-таблицей, управления двоичным файлом. Реализовать поочередно все перечисленные ниже операции в этом заголовочном файле, выполняя их тестирование из функции main приложения. После разработки всех операций выполнить их комплексное тестирование.
8. Разработать и реализовать операции
   1. Прочитать запись из файла и вставить элемент в таблицу (элемент включает: ключ и номер записи с этим ключом в файле, и для метода с открытой адресацией возможны дополнительные поля).
   2. Удалить запись из таблицы при заданном значении ключа и соответственно из файла.
   3. Найти запись в файле по значению ключа (найти ключ в хеш-таблице, получить номер записи с этим ключом в файле, выполнить прямой доступ к записи по ее номеру).
   4. Подготовить тесты для тестирования приложения:
   5. Заполните файл небольшим количеством записей.
   6. Включите в файл записи как не приводящие к коллизиям, так и приводящие.
   7. Обеспечьте включение в файл такого количества записей, чтобы потребовалось рехеширование.
   8. Заполните файл большим количеством записей (до 1 000 000).
   9. Определите время чтения записи с заданным ключом: для первой записи файла, для последней и где-нибудь в середине. Убедитесь (или нет), что время доступа для всех записей одинаково.
9. Составить отчет.

Вариант №27. Условие задания:

|  |  |
| --- | --- |
| Упражнение 1 | **Метод разрешения коллизий:**  Открытая адресация (смещение на 1)  **Структура записи двоичного файла:**  Учет техосмотра автомобилей. Структура записи об автомобиле: Номер (код региона, цифровой код, буквенный код), Модель, Цвет, Сведения о владельце (Фамилия, Имя, Адрес), дата последнего техосмотра. |

# **Решение**

**Объяснение алгоритмов применяемых функций**

**Задание 1**

Функция записывает в строку str запись, найденную в бинарном файле file\_name по уникальному номеру index, затем формирует из этой строки запись node и вставляет эту запись в хеш-таблицу table.

|  |
| --- |
| //Функция вставки элемента в хеш-таблицу из файла  void pasteEl(string file\_name, HashTable& table, string index) {  string str = fileGet(file\_name, index);  stringstream ss(ios::in | ios::out);  ss << str;  Node node;  ss >> node.number;  ss >> node.model;  ss >> node.color;  ss >> node.surname;  ss >> node.name;  ss >> node.date;  table.pasteKey(node);  } |

Функция удаляет из бинарного файла file\_name строку line, затем формирует из строки line запись node, которая передается в метод deleteKey хеш-таблицы table (deleteKey удаляет элемент из хеш-таблицы).

|  |
| --- |
| //Функция удаления элемента из файла и хеш-таблицы  void deleteEl(string file\_name, HashTable& table, string line) {  fileDelete(file\_name, line);  stringstream ss(ios::in | ios::out);  ss << line;  Node node;  ss >> node.number;  ss >> node.model;  ss >> node.color;  ss >> node.surname;  ss >> node.name;  ss >> node.date;  table.deleteKey(node);  } |

Функция записывает в pos индекс искомого элемента в хеш-таблице, далее берет уникальный номер записи по этому индексу и записывает его в строку index, затем функция возвращает элемент в бинарном файле, найденный по этому уникальному номеру.

|  |
| --- |
| //Функция поиска элемента в хеш-таблице и бинарном файле  string findEl(string file\_name, HashTable& table, Node node) {  int pos=table.findKey(node);  string index = table.table[pos].number;  return fileGet(file\_name, index); |

Метод определяет длину хеш-таблицы length, обнуляет число хранящихся в ней записей totalNodes, создает массив table длины length, на основе которого сделана хеш-таблица, а также заполняет хеш-таблицу пустыми записями.

|  |
| --- |
| //Метод конструктора хеш-таблицы  HashTable::HashTable(int length) {  this->length = length;  this->totalNodes = 0;  this->table = new Node[length];  Node clearNode;  clearNode.color = "";  clearNode.date = "";  clearNode.model = "";  clearNode.name = "";  clearNode.number = "";  clearNode.surname = "";  for (int i = 1; i <= length; i++) {  table[i-1] = clearNode;  } |

Метод получает на вход запись node, далее считывает ее посимвольно, применяет к каждому символу операции сложения, умножения, побитовых сдвигов, исключающего ИЛИ, полученный результат добавляется в переменную-аккумулятор hashCode. Далее метод обрабатывает hashCode посредством логических и арифметических операций и возвращает остаток от деления значения hashCode на длину хеш-таблицы length.

|  |
| --- |
| //Метод хеширования ключа  long long HashTable::hashCode(Node node) {  long long hashCode=0;  for (int i = 0; i < node.number.length(); i++) {  hashCode += (long long)node.number[i]<<5;  hashCode += ((node.number[i] << 1) \* 133 / 53)<<4;  }    for (int i = 0; i < node.surname.length(); i++) {  hashCode += (long long)node.surname[i]<<3;  hashCode += ((node.surname[i] << 1) \* 57 / 7)<<3 ;  }  for (int i = 0; i < node.date.length(); i++) {  hashCode += (long long)node.date[i]<<8;  hashCode += (long long)(node.date[i] \* 3 +997)>>13 ;  }  for (int i = 0; i < node.model.length(); i++) {  hashCode += (long long)node.model [i] << 8;  hashCode += (long long)(node.model[i] \* 997) << 1;  }  for (int i = 0; i < node.name.length(); i++) {  hashCode += (long long)node.name[i] << 8;  }  for (int i = 0; i < node.color.length(); i++) {  hashCode += (long long)node.color[i] << 8;  }  hashCode += hashCode<<3;  hashCode ^= hashCode >> 2;  hashCode += hashCode << 5;  hashCode = hashCode % length;  return hashCode;  } |

Метод хеширует запись node, вставляет ее в хеш-таблицу по индексу index (либо на другую позицию, в зависимости от появления коллизии (коллизии разрешаются методом открытой адресации со смещением 1)), далее к значению totalNodes добавляется единица и вызывается метод рехеширования хеш-таблицы.

|  |
| --- |
| //Метод вставки ключа в хеш-таблицу  bool HashTable::pasteKey(Node node) {  int index = hashCode(node);  for (int i = index+1; i <= length; i++) {  string check = table[i-1].color + table[i-1].date + table[i-1].model + table[i-1].name + table[i-1].number + table[i-1].surname;  if (check.length() == 0) {  table[i-1] = node;  break;  }  if (i == length) i = 0;  if (i == index) return false;  }  totalNodes += 1;  rehashTable();  return true;  } |

Метод хеширует запись node, находит ее в хеш-таблице по индексу index (либо на другой позиции, в зависимости от появления коллизии (коллизии разрешаются методом открытой адресации со смещением 1)), удаляет её и уменьшает значение totalNodes на единицу.

|  |
| --- |
| //Метод удаления ключа из хеш-таблиц  bool HashTable::deleteKey(Node node) {  int index = hashCode(node);  for (int i = index+1; i <= length; i++) {  if (table[i-1].color == node.color && table[i-1].date == node.date && table[i-1].model == node.model && table[i-1].name == node.name && table[i-1].number == node.number && table[i-1].surname == node.surname) {    table[i-1].color = "";  table[i-1].date = "";  table[i-1].model = "";  table[i-1].name = "";  table[i-1].number = "";  table[i-1].surname = "";  break;  }  if (i == length) i = 0;  if (i == index) return false;  }  totalNodes -= 1;  return true;  } |

Метод хеширует запись node, находит ее в хеш-таблице и возвращает индекс найденной записи. Если запись не найдена, метод возвращает -1.

|  |
| --- |
| //Метод поиска ключа в хеш-таблице  int HashTable::findKey(Node node) {  int index = hashCode(node);  for (int i = index+1; i <= length; i++) {  if (table[i-1].color == node.color &&  table[i-1].date == node.date &&  table[i-1].model == node.model &&  table[i-1].name == node.name &&  table[i-1].number == node.number &&  table[i-1].surname == node.surname) {  return i-1;  }  if (i == length) i = 0;  if (i == index) return -1;  }  } |

Метод рехеширует таблицу, если она заполнена более, чем на половину, а именно: создается массив для копирования данных copy, в него копируется текущая хеш-таблица, далее на таблицу выделяется в 5 раз больше памяти, чем до этого, и записи из массива copy перевставляются в хеш-таблицу.

|  |
| --- |
| //Метод рехеширования хеш-таблицы.  void HashTable::rehashTable() {  if (totalNodes > length / 2) {  Node\* copy = new Node[length];  for (int i = 0; i < length; i++) {  copy[i] = table[i];  }  table = new Node[length \* 5];  int oldIndex = length;  length \*= 5;  totalNodes = 0;  for (int i = 0; i < oldIndex; i++) {  pasteKey(copy[i]);  }  }  } |

Метод проходит всю хеш-таблицу и выводит все индексы и все записи таблицы на экран.

|  |
| --- |
| //Метод вывода хеш-таблицы на экран  void HashTable::outputTable() {  for (int i = 0; i < length; i++) {  cout <<setw(5)<< i << " | ";  cout << table[i].number << " ";  cout << table[i].model << " ";  cout << table[i].color << " ";  cout << table[i].surname << " ";  cout << table[i].name << " ";  cout << table[i].date << "\n";  }  } |

Функция создает хеш-таблицу на 10 элементов, создает 2 записи, не создающих коллизию, и 4 записи, создающих коллизию. Далее функция вставляет «неколизионные» записи в таблицу и выводит ее на экран. Затем функция вставляет 3 «колизионные» записи в таблицу и выводит ее на экран. Затем функция вставляет оставшуюся «колизионную» запись в таблицу, что провоцирует ее рехеширование. Далее из хеш-таблицы удаляется элемент «V716HW EHT Z ASMNEK UT 77.41.5535» и таблица выводится на экран. Затем происходит поиск и вывод на экран индекса элемента «J932PU WXI V MYGTCT FO 28.05.2617».

|  |
| --- |
| //Функция тестирования хеш-таблицы  void testHeshT() {  int len = 10;  HashTable table(len);    Node noCols1;  Node noCols2;  Node cols1;  Node cols2;  Node cols3;  Node cols4;  //U156ZJ CSO R OVFNQS CN 13.80.5713  noCols1.color="R";  noCols1.date="13.80.5713";  noCols1.model="CSO";  noCols1.name="CN";  noCols1.number="U156ZJ";  noCols1.surname="OVFNQS";  cout << "Хеш 1: " << table.hashCode(noCols1)<<endl;  //X617JG ZCQ Q KTKOVR LB 62.35.8929  noCols2.color="Q";  noCols2.date="62.35.8929";  noCols2.model="ZCQ";  noCols2.name="LB";  noCols2.number="X617JG";  noCols2.surname="KTKOVR";  cout << "Хеш 2: " << table.hashCode(noCols2) << endl;  table.pasteKey(noCols1);  table.pasteKey(noCols2);  table.outputTable();  cout << endl;  //P371DD IYV C NZKTMN NW 73.49.5453  cols1.color = "C";  cols1.date = "73.49.5453";  cols1.model = "IYV";  cols1.name = "NW";  cols1.number = "P371DD";  cols1.surname = "NZKTMN";  cout << "Хеш 1 (cols): " << table.hashCode(cols1) << endl;  //V716HW EHT Z ASMNEK UT 77.41.5535  cols2.color = "Z";  cols2.date = "77.41.5535";  cols2.model = "EHT";  cols2.name = "UT";  cols2.number = "V716HW";  cols2.surname = "ASMNEK";  cout << "Хеш 2 (cols): " << table.hashCode(cols2) << endl;  //J932PU WXI V MYGTCT FO 28.05.2617  cols3.color = "V";  cols3.date = "28.05.2617";  cols3.model = "WXI";  cols3.name = "FO";  cols3.number = "J932PU";  cols3.surname = "MYGTCT";  cout << "Хеш 3 (cols): " << table.hashCode(cols3) << endl;  //X015TI LCM T XSPUAH MC 22.58.9276  cols4.color = "T";  cols4.date = "22.58.9276";  cols4.model = "LCM";  cols4.name = "MC";  cols4.number = "X015TI";  cols4.surname = "XSPUAH";  cout << "Хеш 4 (cols): " << table.hashCode(cols3) << endl;  table.pasteKey(cols1);  table.pasteKey(cols2);  table.pasteKey(cols3);  table.outputTable();  cout << endl;  table.pasteKey(cols4);  table.outputTable();  cout << endl;  cout << "Удаляемый ключ: V716HW EHT Z ASMNEK UT 77.41.5535\n";  table.deleteKey(cols2);  table.outputTable();  cout << endl;  cout << "Искомый ключ: J932PU WXI V MYGTCT FO 28.05.2617" << endl;  cout << "Индекс искомого ключа: " << table.findKey(cols3);  } |

Функция конвертирует файл data.txt в файл databin.bin, выводит databin.bin на экран. Далее функция ищет и выводит запись с уникальным номером A131BC из файла databin.bin. Затем функция удаляет запись «A131BC TOY R DURKIN VS 28.08.2003» из файла databin.bin, дозаписывает в его конец запись «U156ZJ CSO R OVFNQS CN 13.80.5713» и выводит содержимое файла databin.bin на экран.

|  |
| --- |
| //Функция тестирования операций с бинарным файлом  void testBinF() {  textToBin("data.txt", "databin.bin");  fileOutputBin("databin.bin");  cout << "\n\n\n";  cout << fileGet("databin.bin", "A131BC");  cout << "\n\n\n";  fileDelete("databin.bin", "A131BC TOY R DURKIN VS 28.08.2003\n");  fileAppend("databin.bin", "U156ZJ CSO R OVFNQS CN 13.80.5713");  fileOutputBin("databin.bin");  } |

**Описание работы пользовательского интерфейса**

Сначала выводится выбор задания, пользователь вводом с клавиатуры выбирает задание. После чего консоль очищается и отображается ввод в зависимости от выбранного задания. Введенные данные обрабатываются, результат обработки выводится на экран, программа автоматически перезапускается.

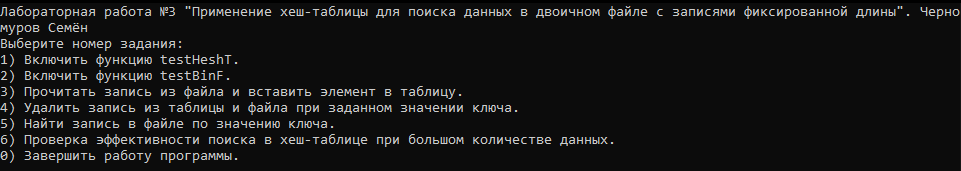


Рисунок 1. Интерфейс программы

# **Тестирование**

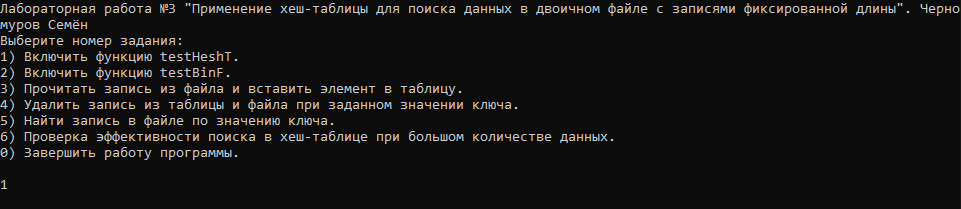


Рисунок 2. Выбор задания 1

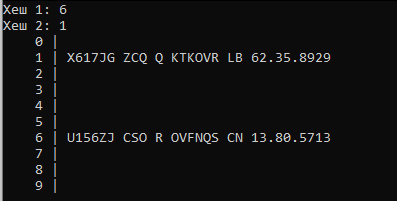


Рисунок 3. Хеши «неколизионных» ключей и их вставка в таблицу

Функция создает хеш-таблицу на 10 элементов, хеширует ключи «U156ZJ CSO R OVFNQS CN 13.80.5713» и «X617JG ZCQ Q KTKOVR LB 62.35.8929», а затем вставляет их в хеш-таблицу в соответствии с полученными значениями хешей.

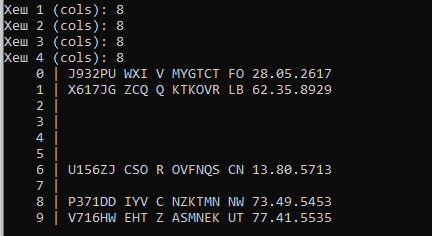


Рисунок 4. Хеши «колизионных» ключей и их вставка в таблицу (3 из 4)

Функция хеширует ключи «P371DD IYV C NZKTMN NW 73.49.5453», «V716HW EHT Z ASMNEK UT 77.41.5535», «J932PU WXI V MYGTCT FO 28.05.2617» и «X015TI LCM T XSPUAH MC 22.58.9276» и вставляет первые три из них в таблицу.

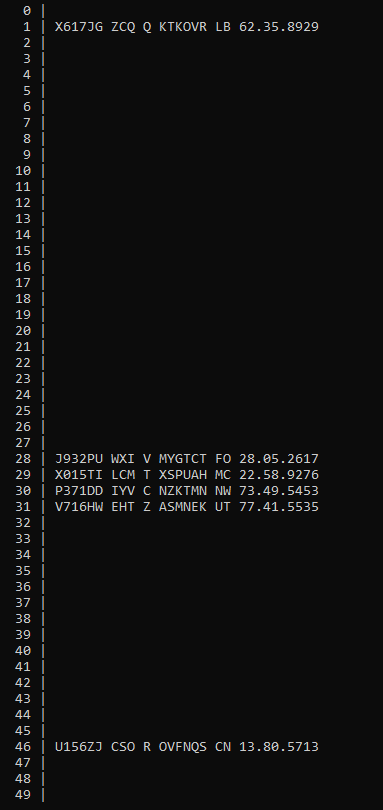


Рисунок 5. Вставка четвертого ключа с последующим рехеширование

Функция вставляет четвертый «колизионный» ключ в хеш-таблицу, что провоцирует ее рехеширование с увеличением размера в 5 раз.

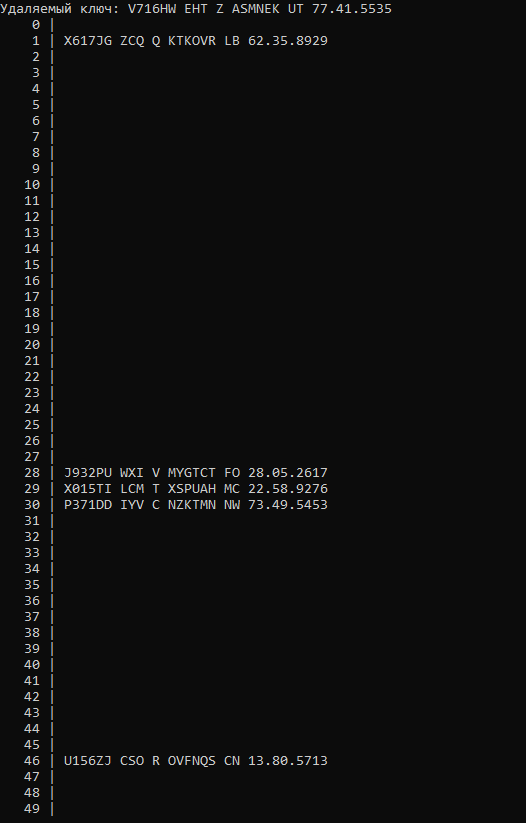


Рисунок 6. Удаление ключа из хеш-таблицы

Функция находит удаляемый ключ «V716HW EHT Z ASMNEK UT 77.41.5535» с учетом хеширования и открытой адресации со смещением на 1 в хеш-таблице и удаляет его, после чего выводит хеш-таблицу на экран.



Рисунок 7. Поиск ключа в хеш-таблице

Функция находит ключ «J932PU WXI V MYGTCT FO 28.05.2617» с учетом хеширования и открытой адресации со смещением на 1 в хеш-таблице и выводит на экран индекс этого ключа в хеш-таблице.

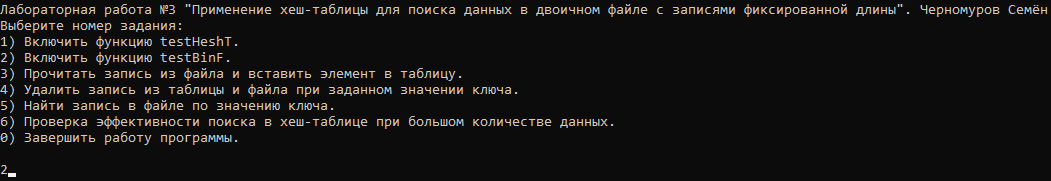


Рисунок 8. Выбор задания 2

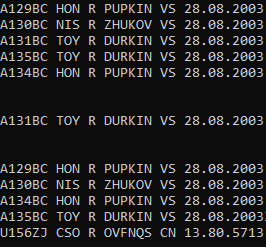


Рисунок 9. Результат работы программы (задание 2)

Функция конвертирует текстовый файл в бинарный, выводит полученный бинарный файл на экран, далее выводит запись с уникальным индексом «A131BC» на экран, затем удаляет её, дописывает в конец файла новую запись «U156ZJ CSO R OVFNQS CN 13.80.5713», и снова выводит бинарный файл на экран.

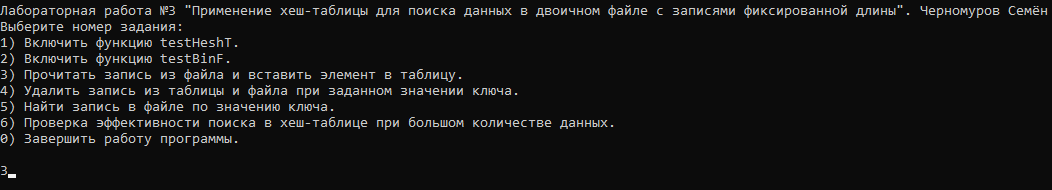


Рисунок 10. Выбор задания 3

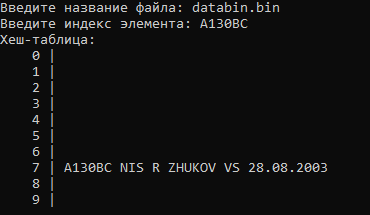


Рисунок 11. Результат работы программы (задание 3)

Функция находит запись с уникальным номером «A130BC» в файле databin.bin, после чего вставляет эту запись в хеш-таблицу с учетом хеширования, и выводит хеш-таблицу на экран.

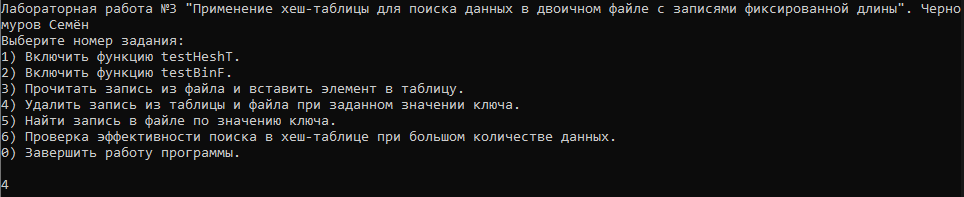


Рисунок 12. Выбор задания 4

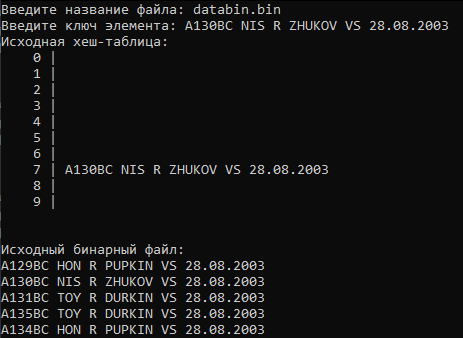


Рисунок 13. Исходные данные в хеш-таблице и бинарном файле

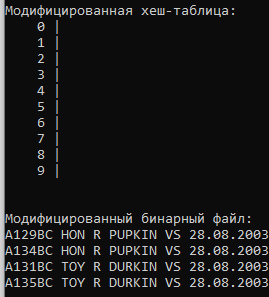


Рисунок 14. Данные в хеш-таблице и бинарном файле после выполнения программы

Функция находит в бинарном файле и хеш-таблице запись «A130BC NIS R ZHUKOV VS 28.08.2003» и удаляет их, после чего измененные хеш-таблица и бинарный файл выводятся на экран.

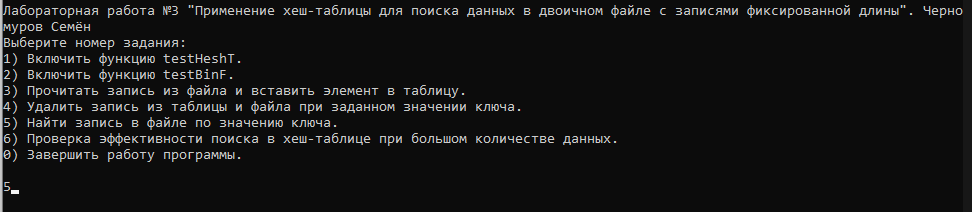


Рисунок 15. Выбор задания 5



Рисунок 16. Указание файла и результат работы программы (задание 5)

Функция находит запись «A130BC NIS R ZHUKOV VS 28.08.2003» в хеш-таблице, берет из этой записи уникальный номер «A130BC», находит в бинарном файле databin.bin эту запись и выводит ее на экран.

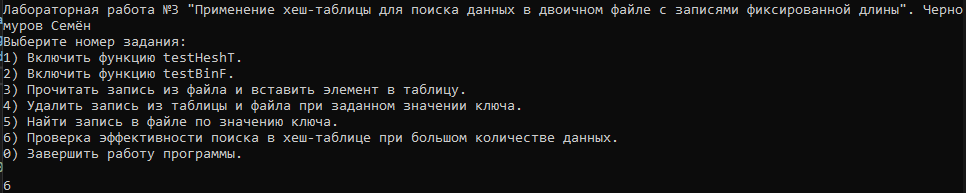


Рисунок 17. Выбор задания 6

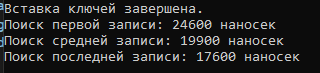


Рисунок 18. Результат работы программы (задание 6)

Создается хеш-таблица на 5000 элементов, в нее из бинарного файла databigbin.bin вставляется 181617 записей (это возможно, так как хеш-таблица автоматически рехешируется с увеличением своего размера), после чего подсчитывается время поиска первой, последней, и серединной записей в этой таблице, результат выводится на экран (тестирование показало, что поиск элемента в хеш-таблице происходит за фиксированное время, и время до получения доступа ко всем ячейкам примерно одинаковое).

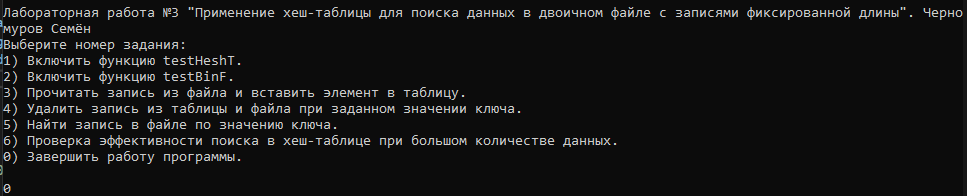


Рисунок 19. Выбор завершения программы

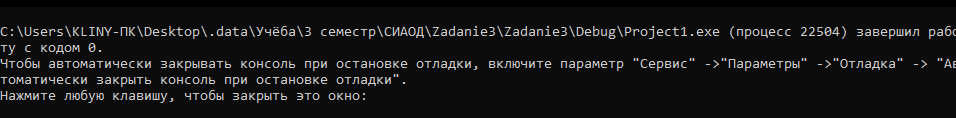


Рисунок 20. Результат выбора завершения программы

При выборе функции завершения работы программы программа завершает свою работу.

Из результатов выполнения программы видно, что программа работает корректно.

# **Вывод**

В результате выполнения работы я:

1. Получил навыки по разработке хеш-таблиц и их применении при поиске данных в других структурах данных (файлах).
2. Закрепил знания по тестированию корректности работы программы.
3. Закрепил навыки создания пользовательских интерфейсов.

# **Исходный код программы**

**Файл main.cpp (основной алгоритм программы)**

|  |
| --- |
| #include "functions\_bin.h"  #include "functions\_main.h"  #include "hash\_table.h"  #include <chrono>  #include <sstream>  #include <fstream>  HashTable table(10);  int main() {  setlocale(LC\_ALL, "");    cout << "Лабораторная работа №3 \"Применение хеш-таблицы для поиска данных в двоичном файле с записями фиксированной длины\". Черномуров Семён\n";  cout << "Выберите номер задания:\n1) Включить функцию testHeshT.\n" <<  "2) Включить функцию testBinF.\n" <<  "3) Прочитать запись из файла и вставить элемент в таблицу.\n" <<  "4) Удалить запись из таблицы и файла при заданном значении ключа.\n" <<  "5) Найти запись в файле по значению ключа.\n" <<  "6) Проверка эффективности поиска в хеш-таблице при большом количестве данных.\n" <<  "0) Завершить работу программы.\n\n";    int choice1;  do {  cin >> choice1;  if (choice1 != 1 && choice1 != 2 && choice1 != 3 && choice1 != 4 && choice1 != 5 && choice1 != 6 && choice1 != 0) cout << "Введено неверное значение, попробуйте снова.\n";  } while (choice1 != 1 && choice1 != 2 && choice1 != 3 && choice1 != 4 && choice1 != 5 && choice1 != 6 && choice1 != 0);  system("cls");  switch (choice1)  {  case 1: {  testHeshT();  break;  }  case 2: {  testBinF();  break;  }  case 3: {  cout << "Введите название файла: ";  string file\_name;  cin >> file\_name;  cout << "Введите индекс элемента: ";  string index;  cin >> index;  pasteEl(file\_name, table, index);  cout << "Хеш-таблица:\n";  table.outputTable();  break;  }  case 4: {  cout << "Введите название файла: ";  string file\_name;  cin >> file\_name;  cout << "Введите ключ элемента: ";  string line;  cin.ignore(32767, '\n');  getline(cin, line);  cout << "Исходная хеш-таблица:\n";  table.outputTable();  cout << "\n\n";  cout << "Исходный бинарный файл:\n";  fileOutputBin(file\_name);  cout << "\n\n";  deleteEl(file\_name, table, line);  cout << "Модифицированная хеш-таблица:\n";  table.outputTable();  cout << "\n\n";  cout << "Модифицированный бинарный файл:\n";  fileOutputBin(file\_name);  break;  }  case 5: {  cout << "Введите название файла: ";  string file\_name;  cin >> file\_name;  cout << "Введите ключ элемента: ";  string line;  cin.ignore(32767, '\n');  getline(cin, line);  stringstream ss(ios::in | ios::out);  ss << line;  Node node;  ss >> node.number;  ss >> node.model;  ss >> node.color;  ss >> node.surname;  ss >> node.name;  ss >> node.date;  cout<<"Найденная запись в файле: " << findEl(file\_name, table, node);  break;  }  case 6: {  string file\_name = "databigbin.bin";  textToBin("databig.txt", "databigbin.bin");  HashTable tablebig(5000);  cout << "f";  string line;  fstream fin(file\_name, ios::in | ios::binary);  int i = 0;  while (getline(fin, line)) {  stringstream ss(ios::in | ios::out);  ss << line;  Node node;  ss >> node.number;  ss >> node.model;  ss >> node.color;  ss >> node.surname;  ss >> node.name;  ss >> node.date;  tablebig.pasteKey(node);  i++;  if (i % 500 == 0) cout << i << endl;  }  cout << "Вставка ключей завершена.\n";  fin.close();  stringstream ss(ios::in | ios::out);  ss << "V873AM ZPD U BRWEUH DH 03.46.0459";  Node node1;  ss >> node1.number;  ss >> node1.model;  ss >> node1.color;  ss >> node1.surname;  ss >> node1.name;  ss >> node1.date;  stringstream ss2(ios::in | ios::out);  ss2 << "I844WM QWH V LKNFDJ MZ 06.87.3722";  Node node2;  ss2 >> node2.number;  ss2 >> node2.model;  ss2 >> node2.color;  ss2 >> node2.surname;  ss2 >> node2.name;  ss2 >> node2.date;  stringstream ss3(ios::in | ios::out);  ss3 << "V438UU ACE M SBIBCS JS 99.01.8490";  Node node3;  ss3 >> node3.number;  ss3 >> node3.model;  ss3 >> node3.color;  ss3 >> node3.surname;  ss3 >> node3.name;  ss3 >> node3.date;  auto start = chrono::steady\_clock::now();  tablebig.findKey(node1);  auto end = chrono::steady\_clock::now();  cout << "Поиск первой записи: " << chrono::duration\_cast<chrono::nanoseconds>(end - start).count() << " наносек" << endl;  auto start2 = chrono::steady\_clock::now();  tablebig.findKey(node2);  auto end2 = chrono::steady\_clock::now();  cout << "Поиск средней записи: " << chrono::duration\_cast<chrono::nanoseconds>(end2 - start2).count()<< " наносек" <<endl;  auto start3 = chrono::steady\_clock::now();  tablebig.findKey(node3);  auto end3 = chrono::steady\_clock::now();  cout << "Поиск последней записи: " << chrono::duration\_cast<chrono::nanoseconds>(end3 - start3).count()<< " наносек";  break;  }  case 0:  return 0;  }  cout << "\n\n";  //textToBin("data.txt", "databin.bin");  main();  } |

**Файл functions\_bin.h (прототипы разработанных функций для работы с бинарным файлом)**

|  |
| --- |
| #pragma once  #include <string>  #include <iostream>  using namespace std;  void fileInput(string file\_name, string data);  string fileOutput(string file\_name);  void textToBin(string file\_name\_txt, string file\_name\_bin);  void fileOutputBin(string file\_name);  string fileGet(string file\_name, string index);  void fileDelete(string file\_name, string line);  void testBinF(); |

**Файл functions\_bin.cpp (тела разработанных функций для работы с бинарным файлом)**

|  |
| --- |
| #include "functions\_bin.h"  #include <iostream>  #include <fstream>  #include <string>  using namespace std;  //Функция записи в файл  void fileInput(string file\_name, string data) {  ofstream fout;  fout.open(file\_name);  if (fout.is\_open()) {  if (fout.good()) fout << data;  }  if (fout.good()) fout.close();  }  //Функция чтения файла  string fileOutput(string file\_name) {  ifstream fin;  fin.open(file\_name);  char chr;  string res = "";  if (fin.is\_open()) {  while (fin.get(chr) && fin.good()) {  res += chr;  }  }  if (fin.eof())  fin.close();  return res;  }  //Функция дозаписи в конец файла  void fileAppend(string file\_name, string data) {  ofstream fout;  fout.open(file\_name, ios::app | ios::binary);  if (fout.is\_open()) {  if (fout.good())  fout << "\n" << data;  }  if (fout.good()) fout.close();  }  //Функция конвертации текстового файла в бинарный  void textToBin(string file\_name\_txt, string file\_name\_bin) {  fstream fin = fstream(file\_name\_txt, ios::in);  fstream fout = fstream(file\_name\_bin, ios::binary | ios::out);  if (fin.is\_open() && fout.is\_open()) {  fout << fin.rdbuf();  if (fin.eof())  fin.close();  if (fout.fail() != ios::failbit)  fout.close();  }  }  //Функция вывода содержимого бинарного файла  void fileOutputBin(string file\_name) {  fstream fin(file\_name, ios::binary | ios::in);  if (fin.is\_open()) cout << fin.rdbuf();  if (fin.eof())  fin.close();  }  //Функция получения записи по ее индексу  string fileGet(string file\_name, string index) {  ifstream fin(file\_name, ios::in | ios::binary);  string line = "";  string str = "";  if (fin.is\_open()) {  ifstream fin2(file\_name, ios::in | ios::binary);  while (fin2) {  str += fin2.get();  }  fin2.close();  if (str.find(index)!=-1) {  fin.seekg(str.find(index)\*sizeof(char));  getline(fin, line);  }  }  if (fin.fail() != ios::failbit)  fin.close();  return line;  }  //Функция удаления записи по ее индексу  void fileDelete(string file\_name, string line) {  string str = fileOutput(file\_name);  string strToCopy = "";  for (int i = str.rfind("\n") + 1; i < str.length(); i++) {  strToCopy += str[i];  }  int pos = str.find(line);  for (int i = pos; i < pos+strToCopy.length(); i++) {  str[i] = strToCopy[i-pos];  }  str = str.erase(str.rfind("\n"), strToCopy.length());  fileInput(file\_name, str);  }  //Функция тестирования операций с бинарным файлом  void testBinF() {  textToBin("data.txt", "databin.bin");  fileOutputBin("databin.bin");  cout << "\n\n\n";  cout << fileGet("databin.bin", "A131BC");  cout << "\n\n\n";  fileDelete("databin.bin", "A131BC TOY R DURKIN VS 28.08.2003\n");  fileAppend("databin.bin", "U156ZJ CSO R OVFNQS CN 13.80.5713");  fileOutputBin("databin.bin");  } |

**Файл functions\_main.h (прототипы разработанных функций для взаимодействия между хеш-таблицей и бинарным файлом)**

|  |
| --- |
| #pragma once  #include "hash\_table.h"  void pasteEl(string file\_name, HashTable& table, string index);  void deleteEl(string file\_name, HashTable& table, string line);  string findEl(string file\_name, HashTable& table, Node node); |

**Файл functions\_main.cpp (тела разработанных функций для взаимодействия между хеш-таблицей и бинарным файлом)**

|  |
| --- |
| #include "functions\_bin.h"  #include "hash\_table.h"  #include "functions\_main.h"  #include <sstream>  //Функция вставки элемента в хеш-таблицу из файла  void pasteEl(string file\_name, HashTable& table, string index) {  string str = fileGet(file\_name, index);  stringstream ss(ios::in | ios::out);  ss << str;  Node node;  ss >> node.number;  ss >> node.model;  ss >> node.color;  ss >> node.surname;  ss >> node.name;  ss >> node.date;  table.pasteKey(node);  }  //Функция удаления элемента из файла и хеш-таблицы  void deleteEl(string file\_name, HashTable& table, string line) {  fileDelete(file\_name, line);  stringstream ss(ios::in | ios::out);  ss << line;  Node node;  ss >> node.number;  ss >> node.model;  ss >> node.color;  ss >> node.surname;  ss >> node.name;  ss >> node.date;  table.deleteKey(node);  }  //Функция поиска элемента в хеш-таблице и бинарном файле  string findEl(string file\_name, HashTable& table, Node node) {  int pos=table.findKey(node);  string index = table.table[pos].number;  return fileGet(file\_name, index);  } |

**Файл hash\_table.h (класс хеш-таблицы)**

|  |
| --- |
| #pragma once  #include <string>  using namespace std;  struct Node {  string number;  string model;  string color;  string surname;  string name;  string date;  };  class HashTable {  private:  int length=0;  int totalNodes;    public:  Node\* table;  HashTable(int length);  long long hashCode(Node node);  bool pasteKey(Node node);  bool deleteKey(Node node);  int findKey(Node node);  void rehashTable();  void outputTable();  };  void testHeshT(); |

**Файл hash\_table.cpp (тела методов класса хеш-таблицы)**

|  |
| --- |
| #include "hash\_table.h"  #include <iostream>  #include <iomanip>  #include <cmath>  #include <fstream>  #include <sstream>  #include "random"  //Метод конструктора хеш-таблицы  HashTable::HashTable(int length) {  this->length = length;  this->totalNodes = 0;  this->table = new Node[length];  Node clearNode;  clearNode.color = "";  clearNode.date = "";  clearNode.model = "";  clearNode.name = "";  clearNode.number = "";  clearNode.surname = "";  for (int i = 1; i <= length; i++) {  table[i-1] = clearNode;  }  }  //Метод хеширования ключа  long long HashTable::hashCode(Node node) {  long long hashCode=0;  for (int i = 0; i < node.number.length(); i++) {  hashCode += (long long)node.number[i]<<5;  hashCode += ((node.number[i] << 1) \* 133 / 53)<<4;  }    for (int i = 0; i < node.surname.length(); i++) {  hashCode += (long long)node.surname[i]<<3;  hashCode += ((node.surname[i] << 1) \* 57 / 7)<<3 ;  }  for (int i = 0; i < node.date.length(); i++) {  hashCode += (long long)node.date[i]<<8;  hashCode += (long long)(node.date[i] \* 3 +997)>>13 ;  }  for (int i = 0; i < node.model.length(); i++) {  hashCode += (long long)node.model [i] << 8;  hashCode += (long long)(node.model[i] \* 997) << 1;  }  for (int i = 0; i < node.name.length(); i++) {  hashCode += (long long)node.name[i] << 8;  }  for (int i = 0; i < node.color.length(); i++) {  hashCode += (long long)node.color[i] << 8;  }  hashCode += hashCode<<3;  hashCode ^= hashCode >> 2;  hashCode += hashCode << 5;  hashCode = hashCode % length;  return hashCode;  }  //Метод вставки ключа в хеш-таблицу  bool HashTable::pasteKey(Node node) {  int index = hashCode(node);  for (int i = index+1; i <= length; i++) {  string check = table[i-1].color + table[i-1].date + table[i-1].model + table[i-1].name + table[i-1].number + table[i-1].surname;  if (check.length() == 0) {  table[i-1] = node;  break;  }  if (i == length) i = 0;  if (i == index) return false;  }  totalNodes += 1;  rehashTable();  return true;  }  //Метод удаления ключа из хеш-таблицы  bool HashTable::deleteKey(Node node) {  int index = hashCode(node);  for (int i = index+1; i <= length; i++) {  if (table[i-1].color == node.color && table[i-1].date == node.date && table[i-1].model == node.model && table[i-1].name == node.name && table[i-1].number == node.number && table[i-1].surname == node.surname) {    table[i-1].color = "";  table[i-1].date = "";  table[i-1].model = "";  table[i-1].name = "";  table[i-1].number = "";  table[i-1].surname = "";  break;  }  if (i == length) i = 0;  if (i == index) return false;  }  totalNodes -= 1;  return true;  }  //Метод поиска ключа в хеш-таблице  int HashTable::findKey(Node node) {  int index = hashCode(node);  for (int i = index+1; i <= length; i++) {  if (table[i-1].color == node.color &&  table[i-1].date == node.date &&  table[i-1].model == node.model &&  table[i-1].name == node.name &&  table[i-1].number == node.number &&  table[i-1].surname == node.surname) {  return i-1;  }  if (i == length) i = 0;  if (i == index) return -1;  }  }  //Метод рехеширования хеш-таблицы.  void HashTable::rehashTable() {  if (totalNodes > length / 2) {  Node\* copy = new Node[length];  for (int i = 0; i < length; i++) {  copy[i] = table[i];  }  table = new Node[length \* 5];  int oldIndex = length;  length \*= 5;  totalNodes = 0;  for (int i = 0; i < oldIndex; i++) {  pasteKey(copy[i]);  }  }  }  //Метод вывода хеш-таблицы на экран  void HashTable::outputTable() {  for (int i = 0; i < length; i++) {  cout <<setw(5)<< i << " | ";  cout << table[i].number << " ";  cout << table[i].model << " ";  cout << table[i].color << " ";  cout << table[i].surname << " ";  cout << table[i].name << " ";  cout << table[i].date << "\n";  }  }  //Функция тестирования хеш-таблицы  void testHeshT() {  int len = 10;  HashTable table(len);    Node noCols1;  Node noCols2;  Node cols1;  Node cols2;  Node cols3;  Node cols4;  //U156ZJ CSO R OVFNQS CN 13.80.5713  noCols1.color="R";  noCols1.date="13.80.5713";  noCols1.model="CSO";  noCols1.name="CN";  noCols1.number="U156ZJ";  noCols1.surname="OVFNQS";  cout << "Хеш 1: " << table.hashCode(noCols1)<<endl;  //X617JG ZCQ Q KTKOVR LB 62.35.8929  noCols2.color="Q";  noCols2.date="62.35.8929";  noCols2.model="ZCQ";  noCols2.name="LB";  noCols2.number="X617JG";  noCols2.surname="KTKOVR";  cout << "Хеш 2: " << table.hashCode(noCols2) << endl;  table.pasteKey(noCols1);  table.pasteKey(noCols2);  table.outputTable();  cout << endl;  //P371DD IYV C NZKTMN NW 73.49.5453  cols1.color = "C";  cols1.date = "73.49.5453";  cols1.model = "IYV";  cols1.name = "NW";  cols1.number = "P371DD";  cols1.surname = "NZKTMN";  cout << "Хеш 1 (cols): " << table.hashCode(cols1) << endl;  //V716HW EHT Z ASMNEK UT 77.41.5535  cols2.color = "Z";  cols2.date = "77.41.5535";  cols2.model = "EHT";  cols2.name = "UT";  cols2.number = "V716HW";  cols2.surname = "ASMNEK";  cout << "Хеш 2 (cols): " << table.hashCode(cols2) << endl;  //J932PU WXI V MYGTCT FO 28.05.2617  cols3.color = "V";  cols3.date = "28.05.2617";  cols3.model = "WXI";  cols3.name = "FO";  cols3.number = "J932PU";  cols3.surname = "MYGTCT";  cout << "Хеш 3 (cols): " << table.hashCode(cols3) << endl;  //X015TI LCM T XSPUAH MC 22.58.9276  cols4.color = "T";  cols4.date = "22.58.9276";  cols4.model = "LCM";  cols4.name = "MC";  cols4.number = "X015TI";  cols4.surname = "XSPUAH";  cout << "Хеш 4 (cols): " << table.hashCode(cols3) << endl;  table.pasteKey(cols1);  table.pasteKey(cols2);  table.pasteKey(cols3);  table.outputTable();  cout << endl;  table.pasteKey(cols4);  table.outputTable();  cout << endl;  cout << "Удаляемый ключ: V716HW EHT Z ASMNEK UT 77.41.5535\n";  table.deleteKey(cols2);  table.outputTable();  cout << endl;  cout << "Искомый ключ: J932PU WXI V MYGTCT FO 28.05.2617" << endl;  cout << "Индекс искомого ключа: " << table.findKey(cols3);  }  //int main() {  // Node node;  // node.number = "P371DD";  // node.model = "IYV";  // node.color = "C";  // node.surname = "NZKTMN";  // node.name = "NW";  // node.date = "73.49.5453";  //  // HashTable table(5000);  //  // //cout << table.hashCode(node); // 4428  //  // fstream fin("databig.txt", ios::in);  // string line;  // while (getline(fin,line)){  // stringstream ss(ios::in | ios::out);  //  // ss << line;  //  // Node node2;  // ss >> node2.number;  // ss >> node2.model;  // ss >> node2.color;  // ss >> node2.surname;  // ss >> node2.name;  // ss >> node2.date;  //  // if (table.hashCode(node2) == table.hashCode(node)) cout << line<<endl;  // }  //  //  //} |

1. **Ответы на вопросы (1-4, вариант №27)**
2. **Расскажите о назначении хеш-функции.**

Хеш-функция нужна для того, чтобы преобразовать массив входных данных в строку (или число) фиксированной длины посредством прогона входных данных через определенный алгоритм, причем должно выполняться условие: на одних и тех же входных данных хеш-функция будет возвращать одинаковое значение. Хеш-функции используются при создании таких структур данных, как хеш-таблица, дерево хешей и т.д., также хеш-функции используются в криптографии (алгоритмы шифрования SHA-1, SHA-2, MD5 и т.д.)

1. **Что такое коллизия?**

Коллизия – это ситуация, при которой разные элементы хеш-таблицы имеют одно и то же хеш-значение. Коллизии могут возникать из-за несовершенства применяемой хеш-функции и из-за недостаточности длины хеш-значения для присвоения всем входным данным уникальных ключей (тип int содержит 32 бита для хранения около 4-х миллиардов чисел, если подать на вход 8 миллиардов входных данных, то коллизий не получится избежать при условии, что хеш-функция не может работать за пределами диапазона int)

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

«Открытый адрес» по отношению к хеш-таблице означает, что местоположение (адрес) элемента не всегда определяется именно его хеш-значением, так как ячейка, соответствующая полученному хеш-значению, может быть занята.

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

В хеш-таблице с открытым адресом коллизия разрешается путем поиска новой ячейки, в которой еще не содержится информация. Для этого поиска существует множество различных методов, например:

1. Линейное смещение
2. Квадратичное смещение
3. Двойное хеширование