|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | |  |  |  | | 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-04-21 | Исаев В.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.

Таблица 1. Вариант задания

|  |  |  |
| --- | --- | --- |
| № | Тип хеш-таблицы (метод разрешения коллизии) | Структура записи двоичного файла |
| 8 | Открытый адрес(двойное хеширование) | Книга: ISBN – двенадцатизначное число, автор, название, год издания. |

# Решение

Хеш-табли́ца — это структура данных, реализующая интерфейс ассоциативного массива, а именно, она позволяет хранить пары (ключ, значение) и выполнять три операции: операцию добавления новой пары, операцию поиска и операцию удаления пары по ключу.

Существуют два основных варианта хеш-таблиц: с цепочками и открытой адресацией. Хеш-таблица содержит некоторый массив H, элементы которого есть пары (хеш-таблица с открытой адресацией) или списки пар (хеш-таблица с цепочками).

Выполнение операции в хеш-таблице начинается с вычисления хеш-функции от ключа. Получающееся хеш-значение i=hash(key) играет роль индекса в массиве H. Затем выполняемая операция (добавление, удаление или поиск) перенаправляется объекту, который хранится в соответствующей ячейке массива H[i].

Ситуация, когда для различных ключей получается одно и то же хеш-значение, называется коллизией. Такие события не так уж и редки — например, при вставке в хеш-таблицу размером 365 ячеек всего лишь 23 элементов вероятность коллизии уже превысит 50%. Поэтому механизм разрешения коллизий — важная составляющая любой хеш-таблицы.

В некоторых специальных случаях удаётся избежать коллизий вообще. Например, если все ключи элементов известны заранее (или очень редко меняются), то для них можно найти некоторую совершенную хеш-функцию, которая распределит их по ячейкам хеш-таблицы без коллизий. Хеш-таблицы, использующие подобные хеш-функции, не нуждаются в механизме разрешения коллизий, и называются хеш-таблицами с прямой адресацией.

Число хранимых элементов, делённое на размер массива H (число возможных значений хеш-функции), называется коэффициентом заполнения хеш-таблицы (load factor) и является важным параметром, от которого зависит среднее время выполнения операций.

1. Разработать приложение, которое использует хеш-таблицу для организации прямого доступа к записям двоичного файла, реализованного в практической работе 2.

Приложение разработано. Тестирование будет проведено в пункте 4 данного отчета.

1. Создать приложение и включить в него три заголовочных файла: управление хеш-таблицей, управление двоичным файлом (практическая работа 2), управление двоичным файлом посредством хеш-таблицы.

Заголовочные файлы подключены. Управление двоичным файлом включено в управление хеш-таблицей для бинарных файлов.

1. 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 таблице 1), разрешающему коллизию.

Элемент содержит необходимые поля. Листинг кода представлен в пункте 5 даннного отчета.

1. 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.

Элемент хеш-таблицы содержит ключ элемента, значение элемента, информацию о том, заполнена ли клетка и удалена ли из нее информация (необходимо для выполнения задания по варианту). Таблица содержит размер таблицы, Элемент таблицы, количество вставленных элементов, количество удаленных элементов, а также метод для создания таблицы.

|  |
| --- |
| struct TypeItembin  {  long long int key = 0;  int offset = 0;  bool openOrClose = true;  bool deleteOrnot = false;  };  struct HashTablebin  {  int L;  TypeItembin \*T;  int insertedCount;  int deletedCount;  void CreateHashTable(int k)  {  T = new TypeItembin[k];  L = k;  insertedCount = 0;  deletedCount = 0;  }  }; |

* 1. Тестирование операций выполнять в функции main приложения по мере их реализации.

Тестирвоание операций выполнено успешно. Тестирование представлено в пункте 4 данного отчета.

* 1. После тестирования всех операций, создать в заголовочном файле функцию с именем testHeshT переместить в нее содержание функции main, проверить, что приложение выполняется.

В заголовочном файле созданы все необходимые пункты. Полный листинг кода представлен в пункте 5 данного отчета.

* 1. Разработать операции по управлению хеш-таблицей.

Для управления таблицей были разработаны функции «deleteFromHashTable» (отвечает за удаление элемента), «search» (отвечает за поиск элемента), «outTable» (отвечает за вывод таблицы), «InsertInHashTable» (отвечает за вставку в таблицу), «Hash» (отвечает за поиск хеш-значения), «reHash» (отвечает за рехеширвоание таблицы).

* 1. Разработать хеш-функцию (метод определить самостоятельно), выполнить ее тестирование, убедиться, что хеш (индекс элемента таблицы) формируется верно.

Была разработана функция «Hash» (отвечает за поиск хеш-значения). Функция генериует хеш-ключ правильно.

|  |
| --- |
| int Hash(int k, int L, int iter)  {  int h1;  if (k % 2 == 0)  {  h1 = k<<2%9;  }  else  {  h1 = 9%k<<2;  }  h1++;  int h2;  if (h1 % 2 == 0)  {  h2 = h1 % 7;  }  else  {  h2 = 7 % h1;  }  h2++;  return (h1 + iter\*h2)%L;  } |

* 1. Разработать операции: вставить ключ в таблицу, удалить ключ из таблицы, найти ключ в таблице, рехешировать таблицу. Каждую операцию тестируйте по мере ее реализации.

Все операции были разработаны успешно. Тестирвоание представлено в пункте 4 данного отчета.

|  |
| --- |
| int reHash(HashTable& old)  {  int size = old.L;  size = size + 10;  HashTable new\_hash;  new\_hash.CreateHashTable(size);  for (int i = 0; i < size - 10; i++)  {  if (old.T[i].key != 0)  {  InsertInHashTable(old.T[i].key, old.T[i].offset, new\_hash);  int j = search(new\_hash, old.T[i].key);  new\_hash.T[j].deleteOrnot = old.T[i].deleteOrnot;  new\_hash.T[j].openOrClose = old.T[i].openOrClose;  }  }  old = new\_hash;  return 1;  }  int InsertInHashTable(int key, int offset, HashTable& t)  {  if (t.insertedCount >= t.L\*0.75)  {  reHash(t);  }  int iteration = 0;  int i = Hash(key, t.L, iteration);  while (i < t.L&&t.T[i].openOrClose == false)  {  if (iteration > t.L)  {  reHash(t);  }  iteration++;  i = Hash(key, t.L, iteration);  }  if (i < t.L)  {  t.T[i].key = key;  t.T[i].offset = offset;  t.T[i].openOrClose = false;  t.insertedCount++;  return 0;  }  else  {  return 1;  }  }  int search(HashTable& t, int key)  {  int iterator = 0;  int i = Hash(key, t.L, iterator);  while (i < t.L && ((t.T[i].openOrClose == false && t.T[i].deleteOrnot == false) || (t.T[i].openOrClose == true && t.T[i].deleteOrnot == true)) && t.T[i].key != key)  {  iterator++;  i = Hash(key, t.L, iterator);  }  if (t.T[i].openOrClose == true && t.T[i].deleteOrnot == false)  {  return -1;  }  return i;  }  int deleteFromHashTable(HashTable& t, int key)  {  int i = search(t, key);  if (i == -1)  {  return 1;  }  else  {  t.T[i].key = 0;  t.T[i].offset = 0;  t.T[i].deleteOrnot = true;  t.T[i].openOrClose = true;  t.deletedCount++;  return 0;  }  } |

* 1. 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

Все тесты были подготовлены. Тестирование проводится в пункте 4 данного отчета.

* 1. Выполнить тестирование операций управления хеш-таблицей. При тестировании операции вставки ключа в таблицу предусмотрите вывод списка индексов, которые формируются при вставке элементов в таблицу.

Тестирвоание проводится в пункте 4 данного отчета.

1. 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.

Операции были успешно созданы. Тестирование проводится в пункте 4 данного отчета.

* 1. Структура записи двоичного файла и все операции по управлению файлом должны быть размещены в соответствующем заголовочном файле.

Все операции были успешно перенесены в соответствующий заголовочный файл.

* 1. Выполнить тестирование операций в main приложения, и содержание функции main переместить в соответствующую функцию заголовочного файла с именем testBinF.

Тестирвоание операций было провоедено успешно. Результат представлен в пункте 4 данного отчета.

1. Управление файлом посредством хеш-таблицы

Функции были успешно дополнены для управления файлом.

1. 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.

Тестирвоание было успешно проведено. Результаты представлены в пункте 4 данного отчета.

1. 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).

|  |
| --- |
| int ReadBin(HashTablebin& t, ifstream& bin\_file)  {  Book book;  bool isGood = true;  while (true)  {  bin\_file.read((char\*)&book, sizeof(book));  if (!bin\_file.eof())  {  if (InsertInHashTable(book.ISBN, i, t) != 0)  {  isGood = false;  }  i++;  }  else  {  if (isGood)  {  return 0;  }  else  {  return 1;  }  }  }  } |

* 1. Удалить запись из таблицы при заданном значении ключа и соответственно из файла.

|  |
| --- |
| int deleteFromHashTable(HashTablebin& t, long long int key)  {  int i = search(t, key);  if (i == -1)  {  return 1;  }  else  {  t.T[i].key = 0;  t.T[i].offset = 0;  t.T[i].deleteOrnot = true;  t.T[i].openOrClose = true;  t.deletedCount++;  return 0;  }  }  void remove\_struct(string bin\_name, long long key)  {  Book book, replace{};  replace = get\_struct(bin\_name, -1);  ifstream bin\_file(bin\_name, ios::binary | ios::in);  ofstream tmp("tmp\_rem.txt");  while (true)  {  bin\_file.read((char\*)&book, sizeof(book));  if (!bin\_file.eof())  {  if (book.ISBN != key)  {  tmp << book.ISBN << " " << book.fam << " " << book.name << " " << book.year << "\n";  }  else  {  tmp << replace.ISBN << " " << replace.fam << " " << replace.name << " " << replace.year << "\n";  }  }  else  {  break;  }  }  tmp.close();  bin\_file.close();  ifstream txt("tmp\_rem.txt", ios::in);  ofstream bin(bin\_name, ios::binary);  txt\_to\_bin(txt, bin);  txt.close();  bin.close();  } |

* 1. 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).

|  |
| --- |
| int search(HashTablebin& t, long long int key)  {  int iterator = 0;  int i = Hash(key, t.L, iterator);  while (i < t.L && ((t.T[i].openOrClose == false && t.T[i].deleteOrnot == false) || (t.T[i].openOrClose == true && t.T[i].deleteOrnot == true)) && t.T[i].key != key)  {  iterator++;  i = Hash(key, t.L, iterator);  }  if (t.T[i].openOrClose == true && t.T[i].deleteOrnot == false)  {  return -1;  }  return t.T[i].offset;  }  void find(int i, ifstream& file)  {  Book book;  file.seekg(i\*sizeof(Book),ios::beg);  file.read((char\*)&book, sizeof(book));  cout << book.ISBN << ' ' << book.fam << ' ' << book.name << ' ' << book.year << '\n';  } |

* 1. Подготовить тесты для тестирования приложения:
  2. Заполните файл небольшим количеством записей.
  3. Включите в файл записи как не приводящие к коллизиям, так и приводящие.
  4. Обеспечьте включение в файл такого количества записей, чтобы потребовалось рехеширование.
  5. Заполните файл большим количеством записей (до 1 000 000).
  6. 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.

Все тесты были подготовлены. Тестирование проводится в пункте 4 данного отчета.

1. Составить отчет.
2. Ответы на вопросы
   1. Расскажите о назначении хеш-фунции.

Хеш-функции необходимы лоя успешной работы с хеш-таблицами, которые в свою очередь ускоряют время работы с большими объемами данных.

* 1. Что такое коллизия?

Коллизия ー ситуация, при которой два элемента имеют одинаковое хеш-значаение в хеш-таблице.

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

Открытый адрес ーэто метод разрешения коллизий в хеш-таблицах. С помощью этого метода коллизия хэшей разрешается путем зондирования или поиска по альтернативным местоположениям в массиве (последовательности зондирования) до тех пор, пока либо не будет найдена целевая запись, либо не будет найден неиспользуемый слот массива, что указывает на отсутствие такого ключа в таблице.

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

Зависит от метода решения коллизии. Например, двойное хеширвоание высчитывает два значаения хеш-функции, одно из которых умножается на шаг поиска свободной ячейки. Значаением хеш-функции будет являться сумма обычного значения хеш-функции и умноженного на шаг.

* 1. Какая проблема, может возникнуть после удаления элемента из хеш-таблицы с открытым адресом и как ее устранить?

При удалении записи, останется свободная ячейнка, но через нее могла пройти другая запись в поиске свободного места. Програмам при поиске ключа, остановаится на удаленной ячейке, так как после удаления она не вызывает коллизию. Решается с помощью добавления к значаению ячейки логической переменной с информацией о том, была удалена ячейка или нет.

* 1. Что определяет коэффициент нагрузки в хеш-таблице?

Число хранимых элементов, делённое на число возможных значений хеш-функции.

* 1. Что такое «первичный кластер» в таблице с открытым адресом?

Проблема, при которой после добавления большого количества значений, новая запись будет большое количество времени искать свободную ячейку.

* 1. Как реализуется двойное хеширование?

Программа высчитывает два значаения хеш-функции, одно из которых умножается на шаг поиска свободной ячейки. Значаением хеш-функции будет являться сумма обычного значения хеш-функции и умноженного на шаг.

# Тестирование

Тестирование приложения.

Вставка значения представлена на рисунке 1.

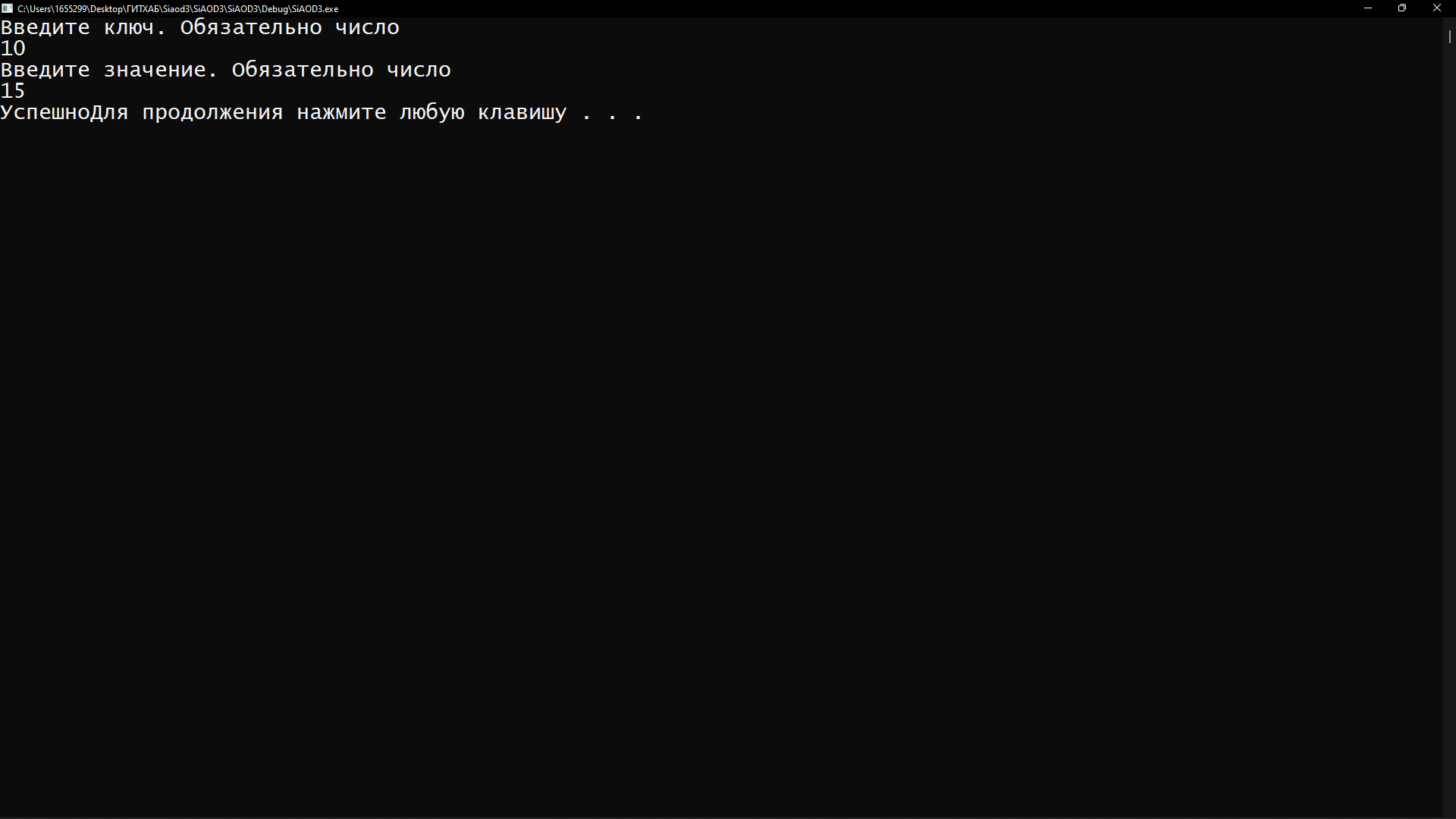


Рисунок 1. Вставка значения

Вставка значения с коллизией представлена на рисунке 2.

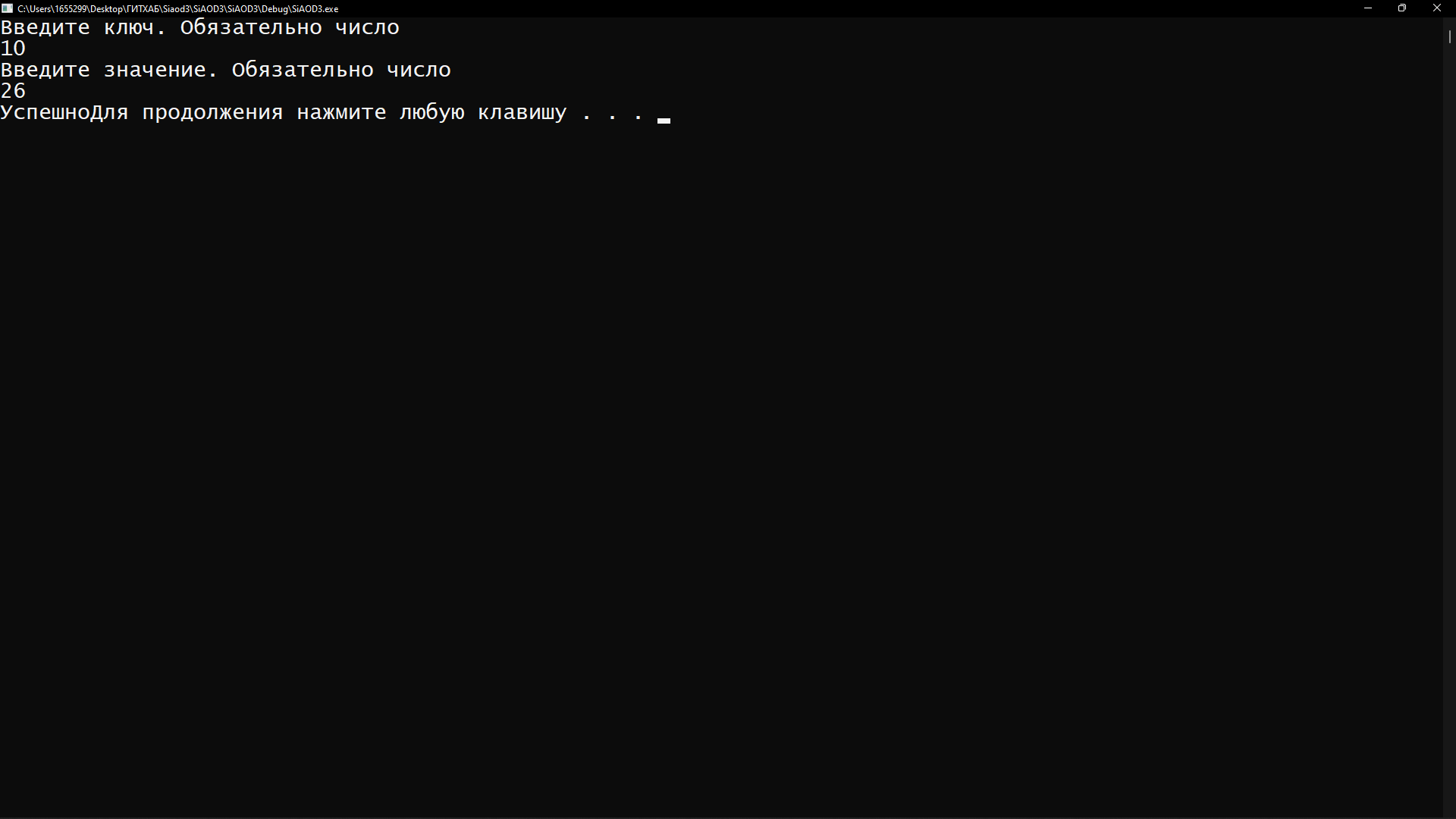


Рисунок 2. Вставка значения с коллизией

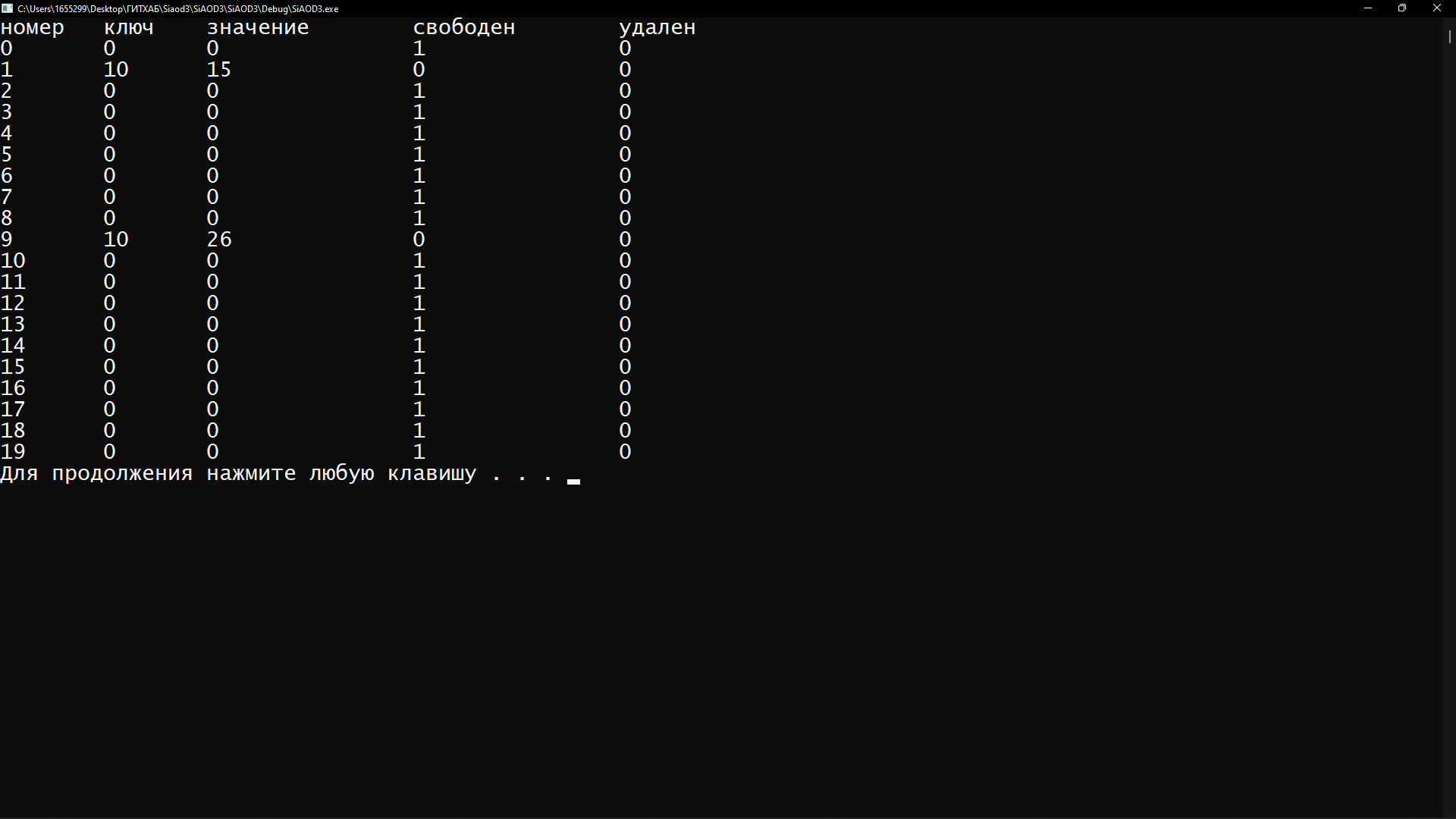
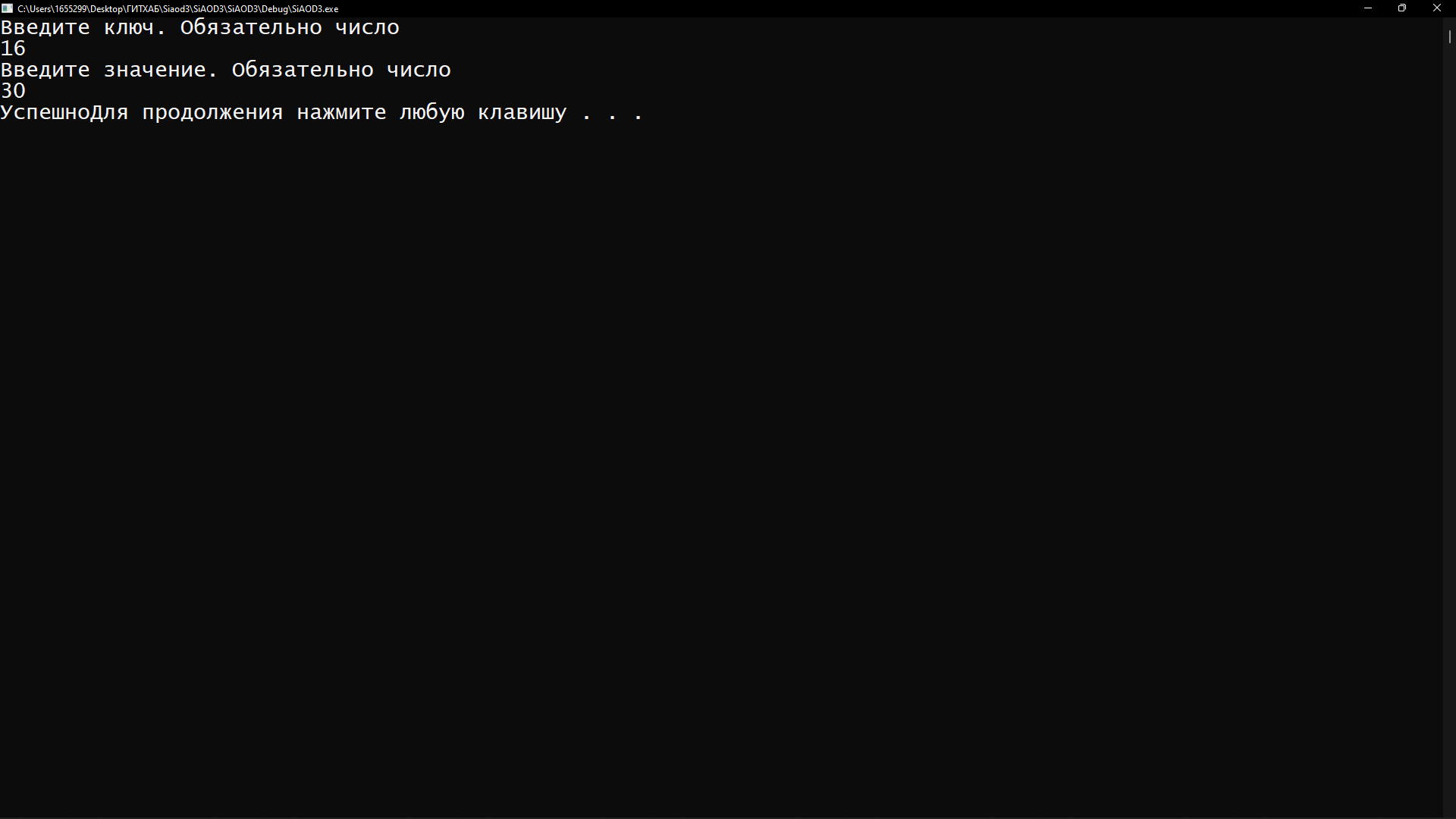
Вывод хеш-таблицы представлен на рисунке 3.

Рисунок 3. Вывод хеш-таблицы

Вставка значаения без коллизии представлен на рисунке 4.Рисунок 4. Вставка значаения без коллизии

Поиск ключа в хеш-таблице представлен на рисунке 5.

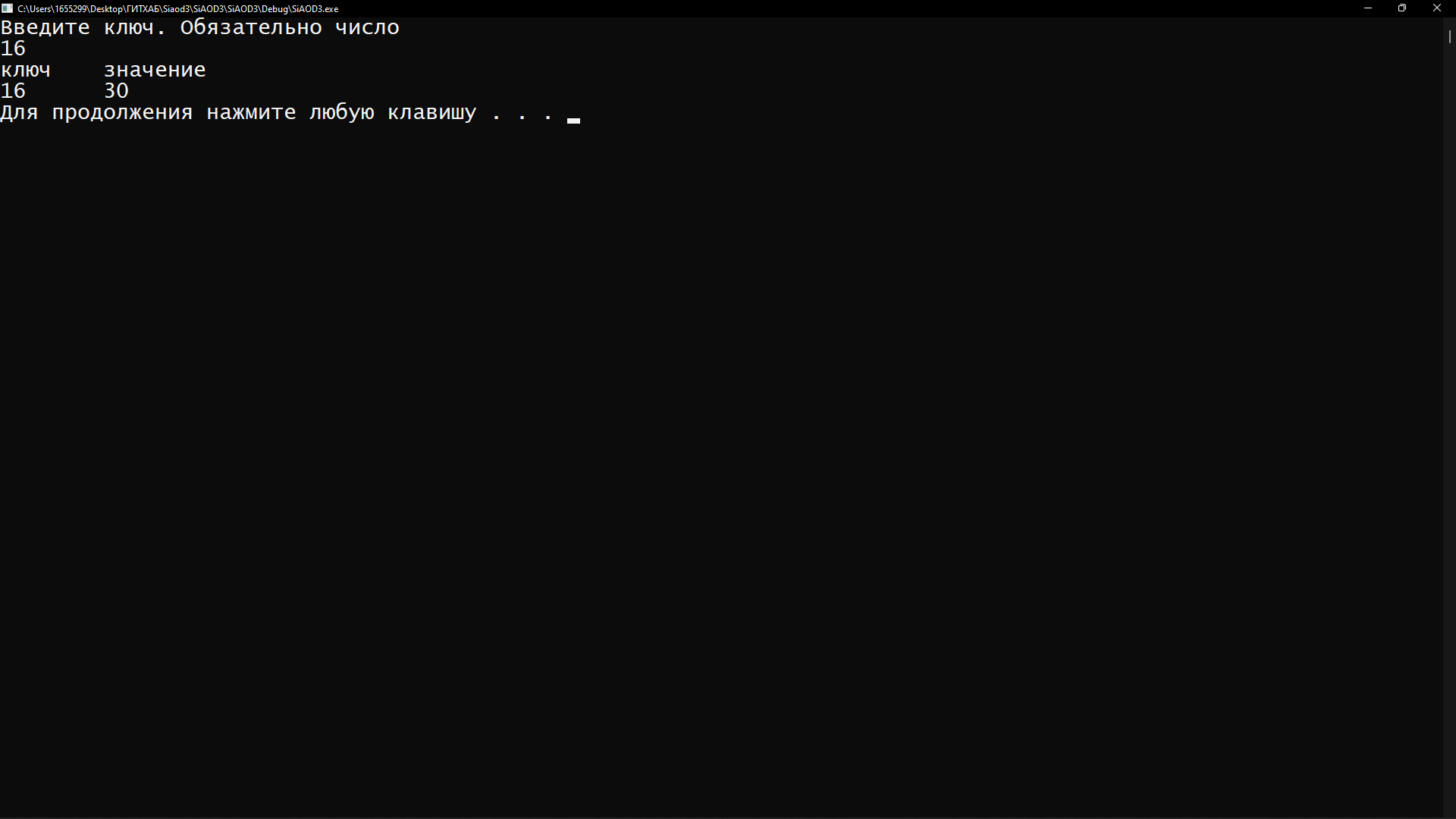


Рисунок 5. Поиск ключа в хеш-таблице

Удаление ключа представлено на рисунке 6.

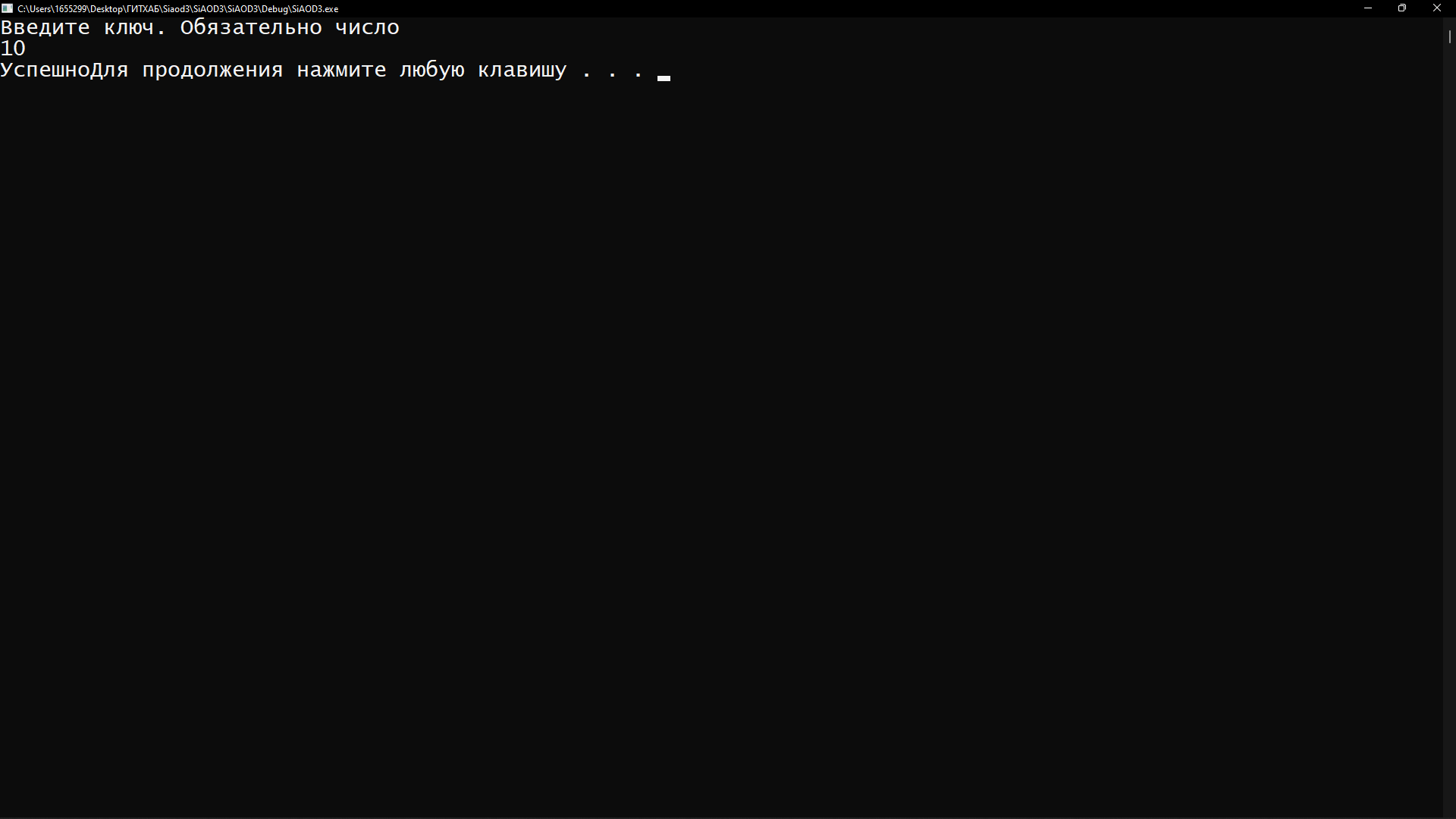


Рисунок 6. Удаление ключа

Вывод таблицы с удвленным ключом представлен на рисунке 7.

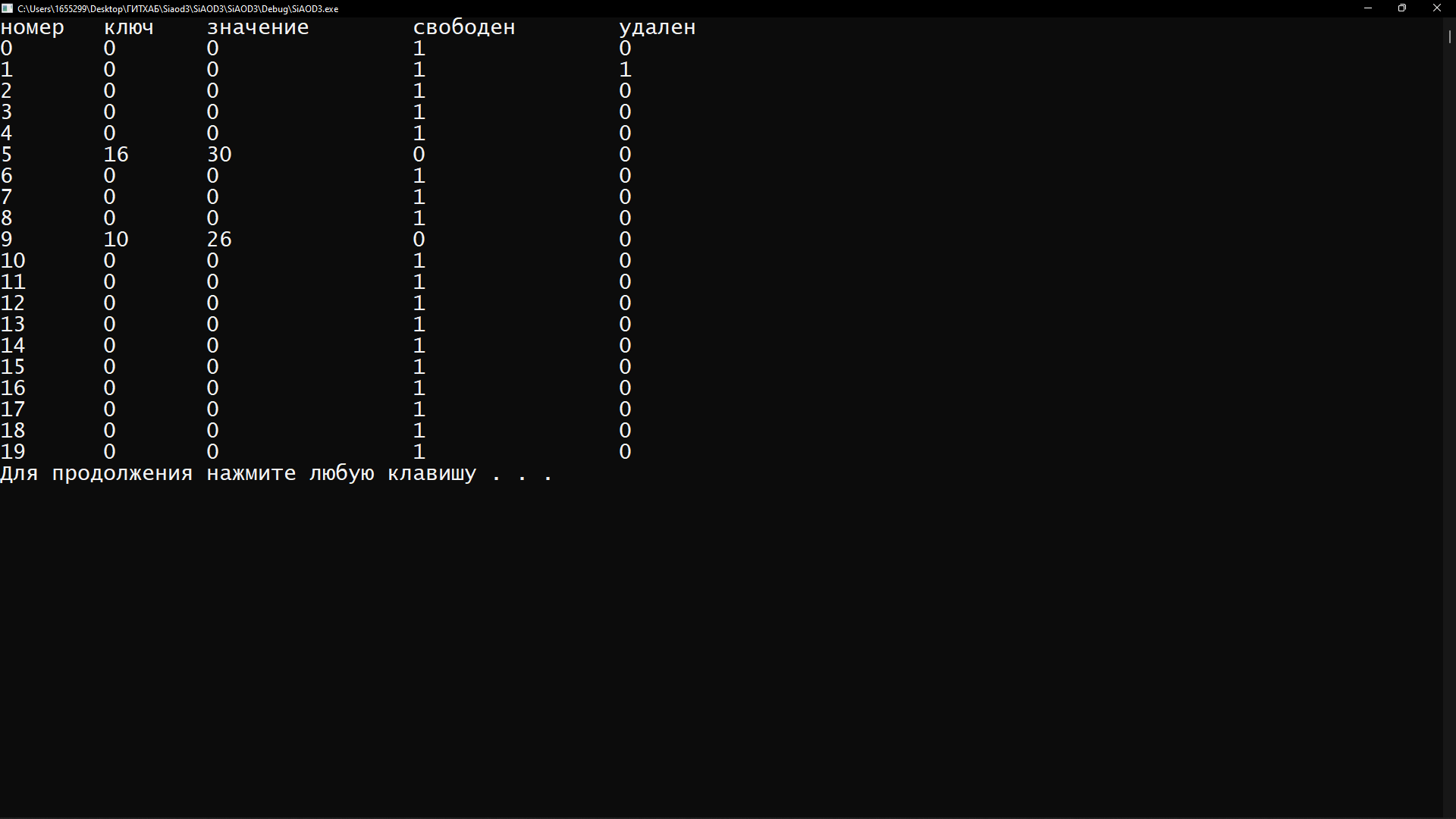


Рисунок 7. Вывод таблицы с удвленным ключом

Вывод бинарного файла представлен на рисунке 8.

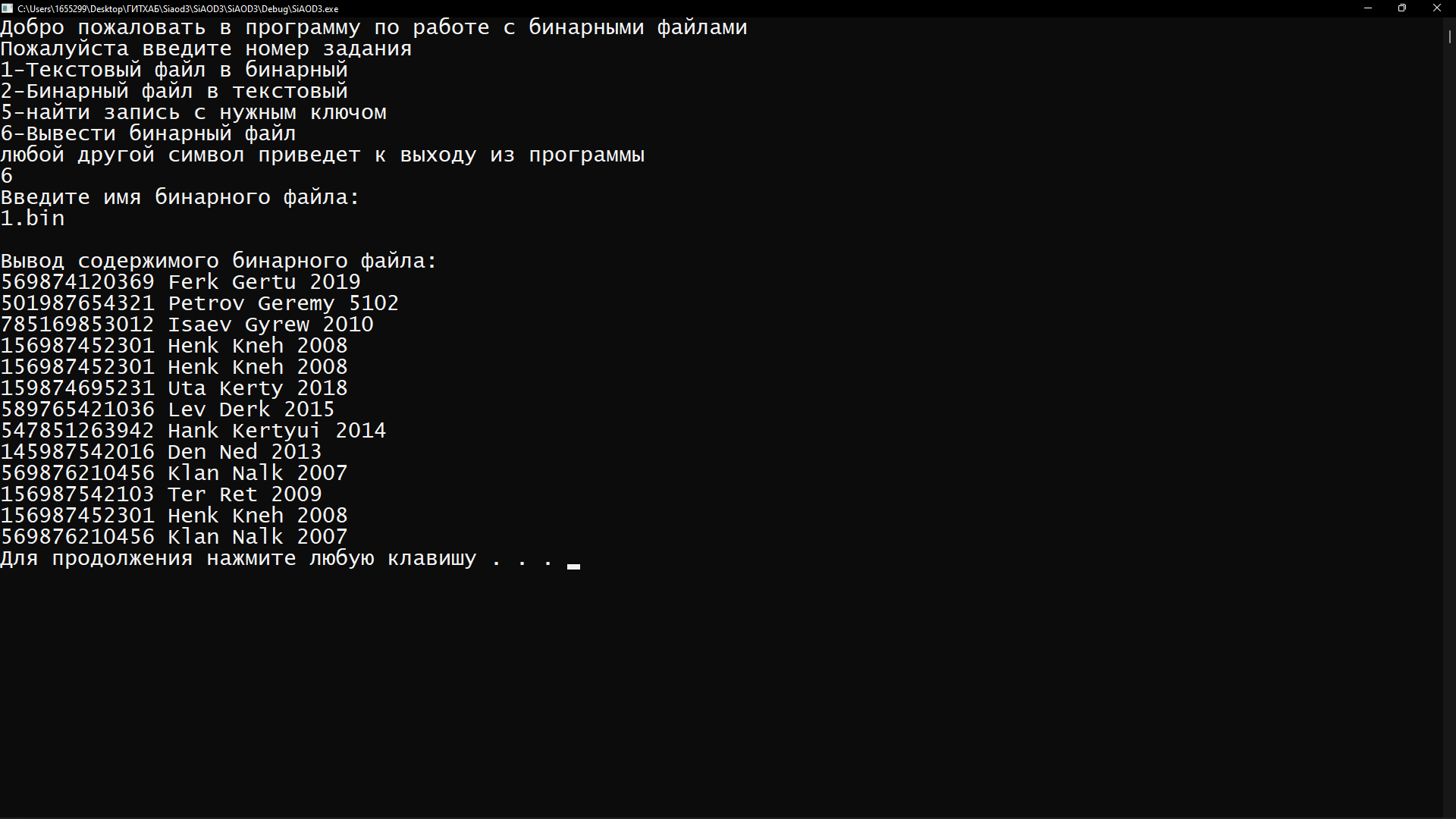


Рисунок 8. Вывод бинарного файла

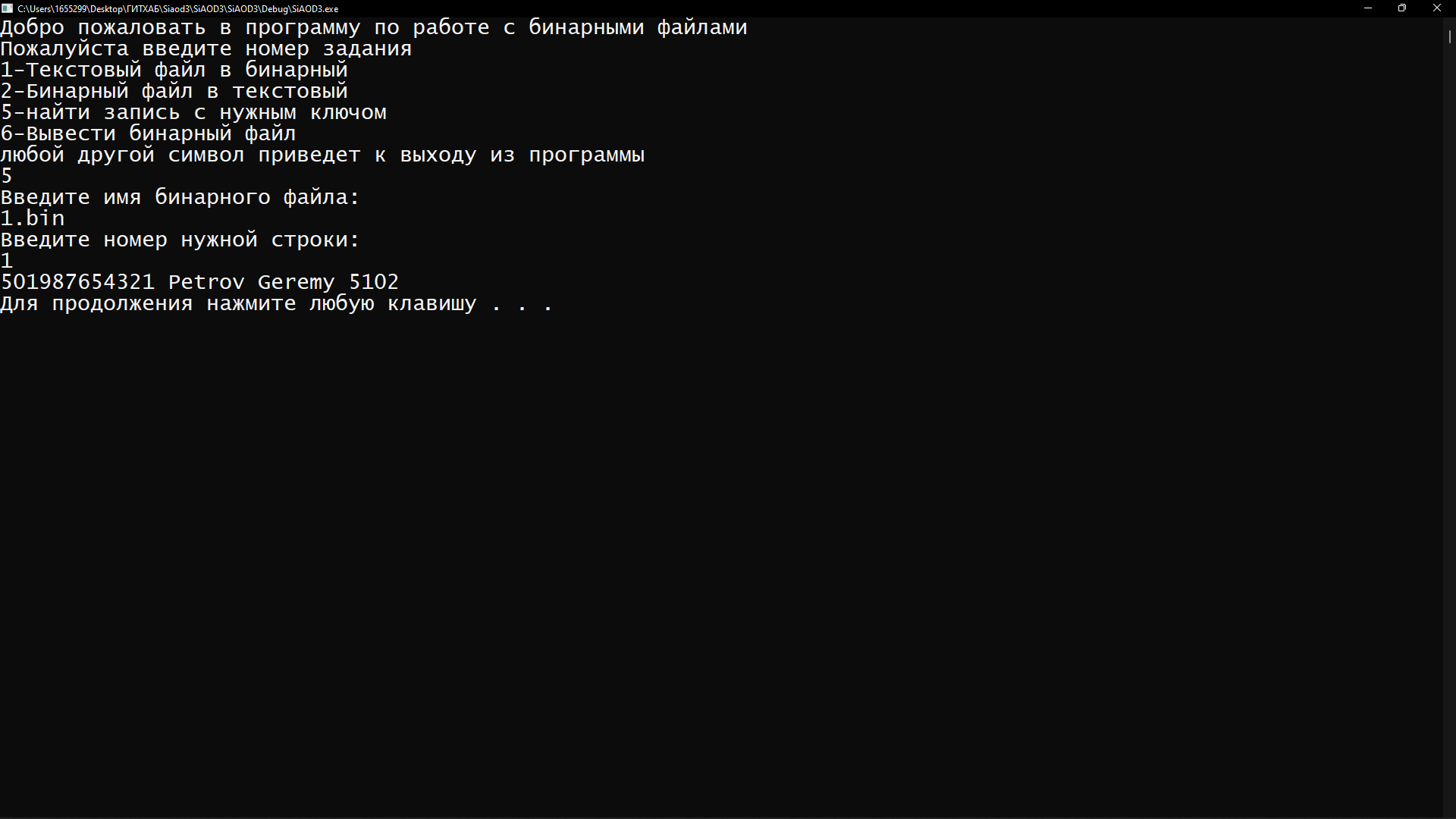
Поиск записи в бинарном файле представлен на рисунке 9.

Рисунок 9. Поиск записи в бинарном файле

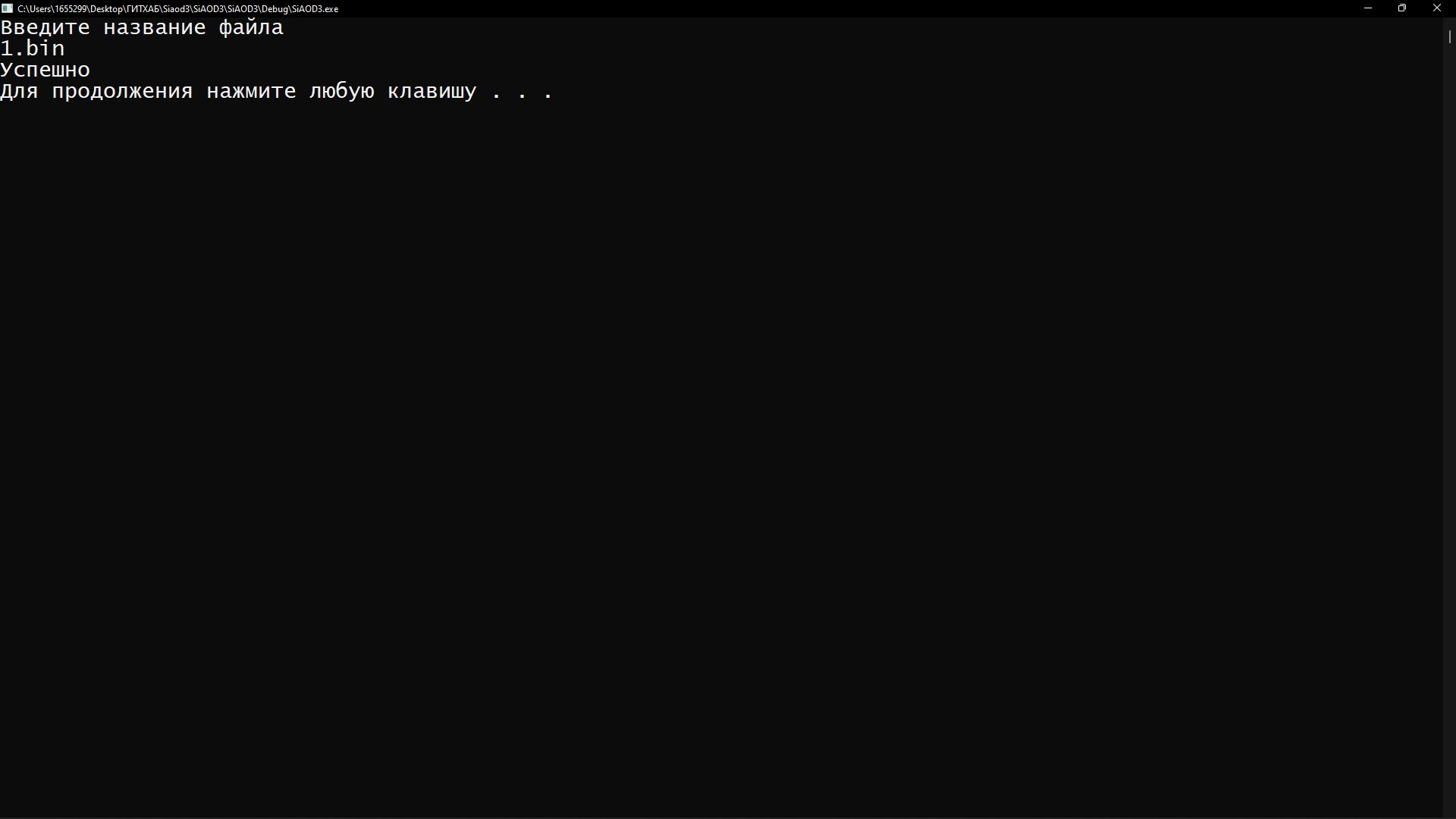
Преобразование бинарного файла в хеш-таблицу представлено на рисунке 10.

Рисунок 10. Преобразование бинарного файла в хеш-таблицу

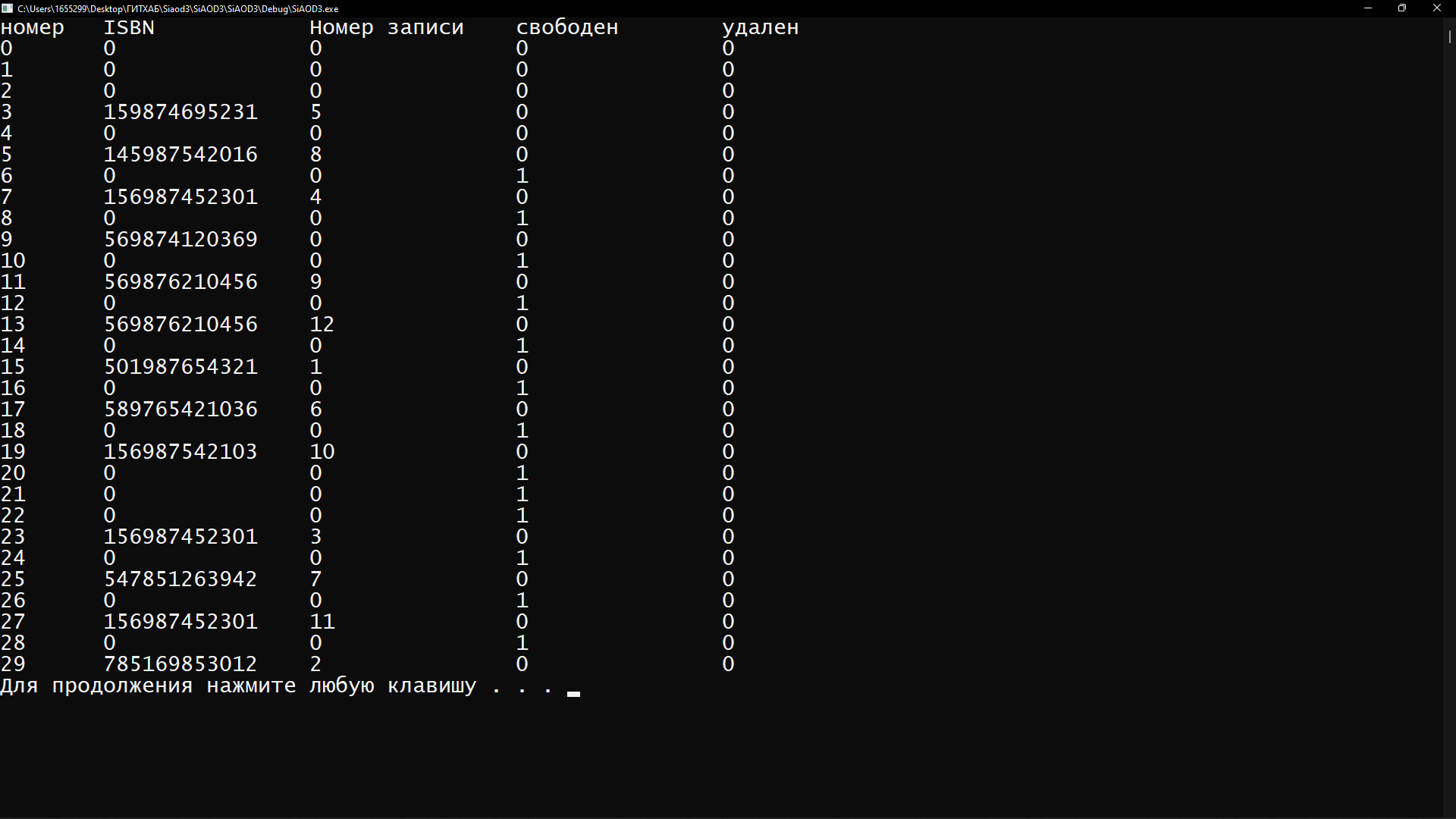
Вывод хеш-таблицы файла представлен на рисунке 11.

Рисунок 11. Вывод хеш-таблицы файла

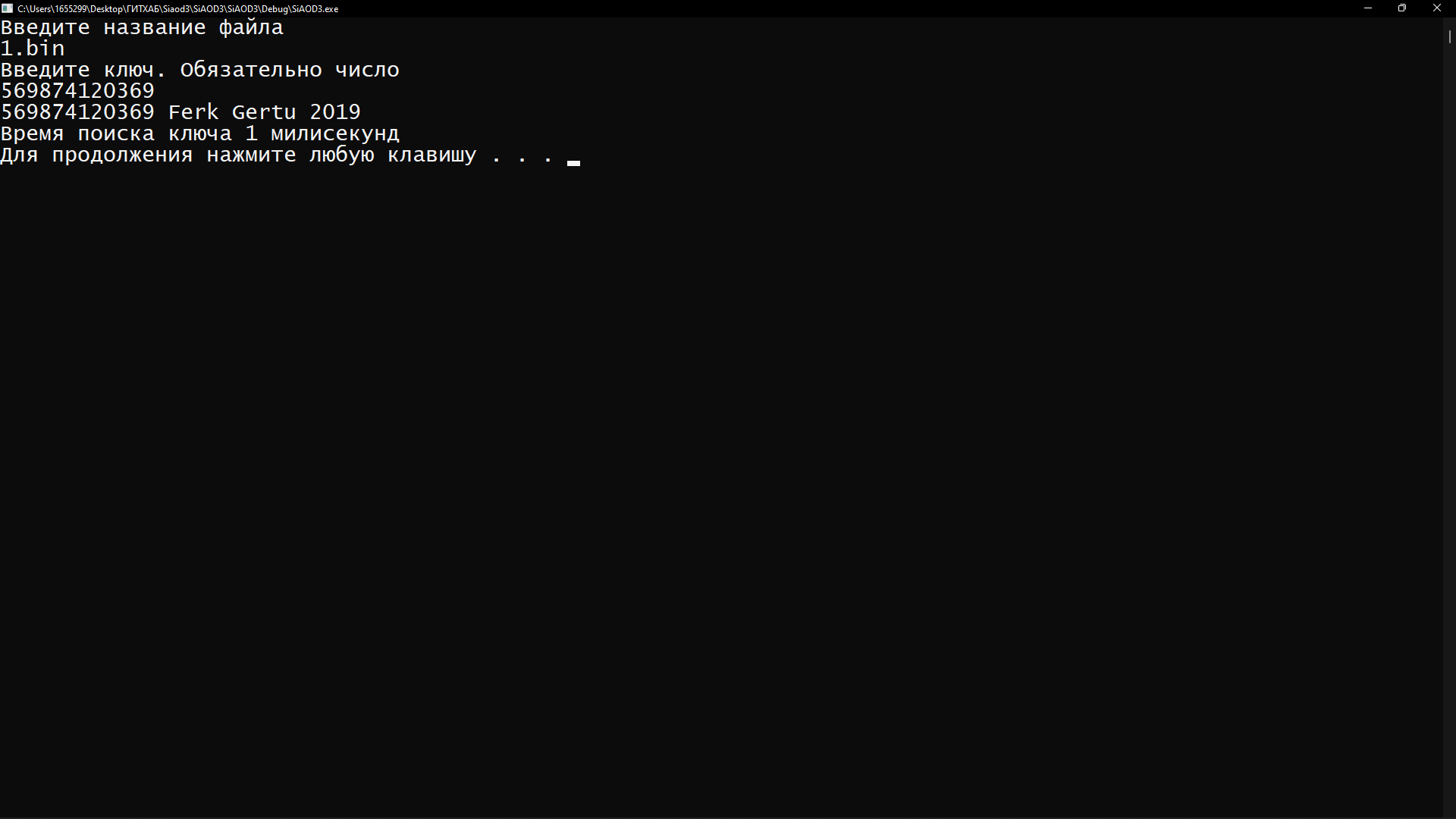
Поиск ключа в хеш-таблице представлен на рисунке 12.

Рисунок 12. Поиск ключа в хеш-таблице

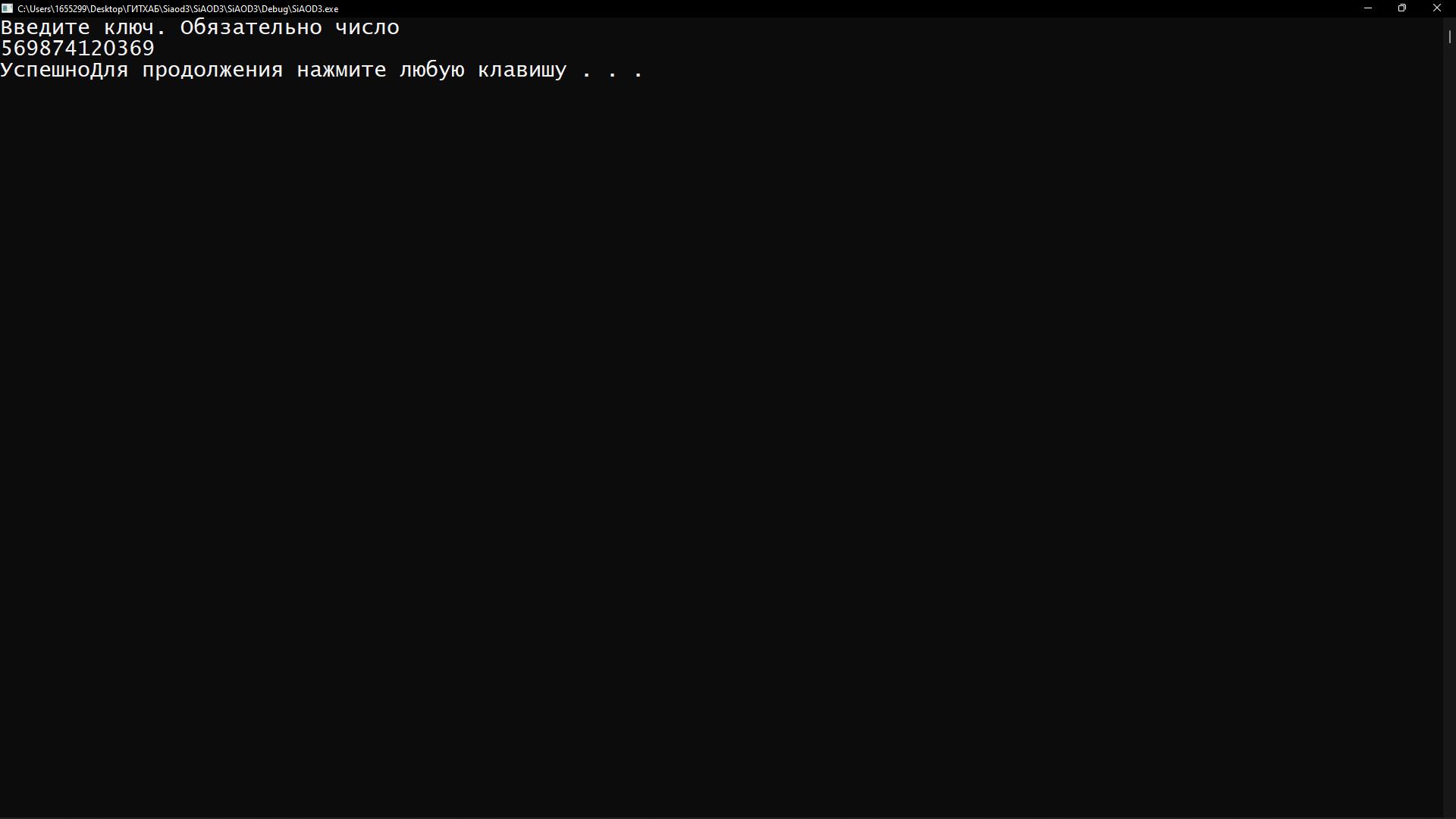
Удаление ключа представлено на рисунке 13.

Рисунок 13. Удаление ключа

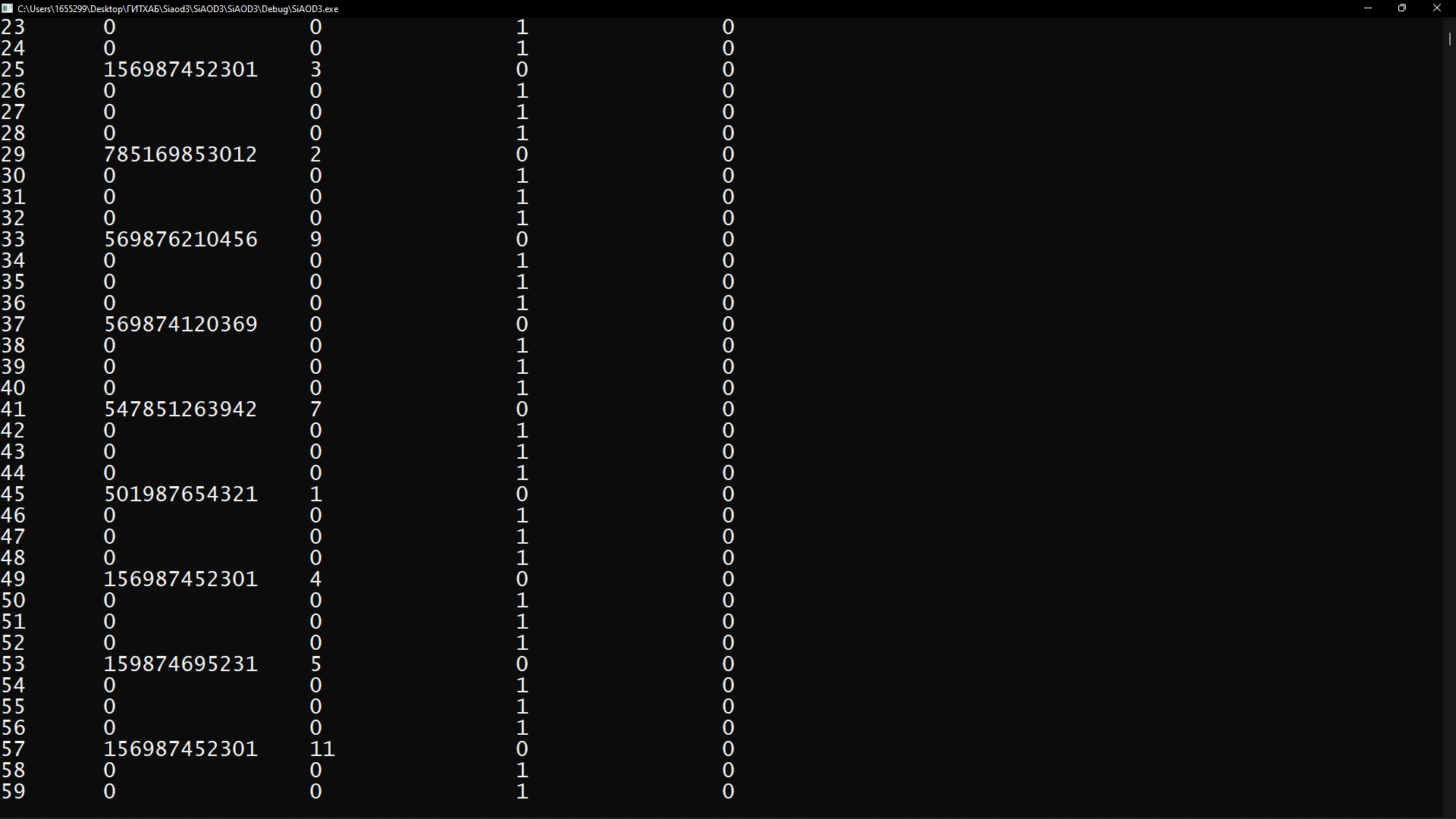
Вывод хеш-таблиы без удаленного кллюча представлен ан рисунке 14.

Рисунок 14. Вывод хеш-таблиы без удаленного кллюча

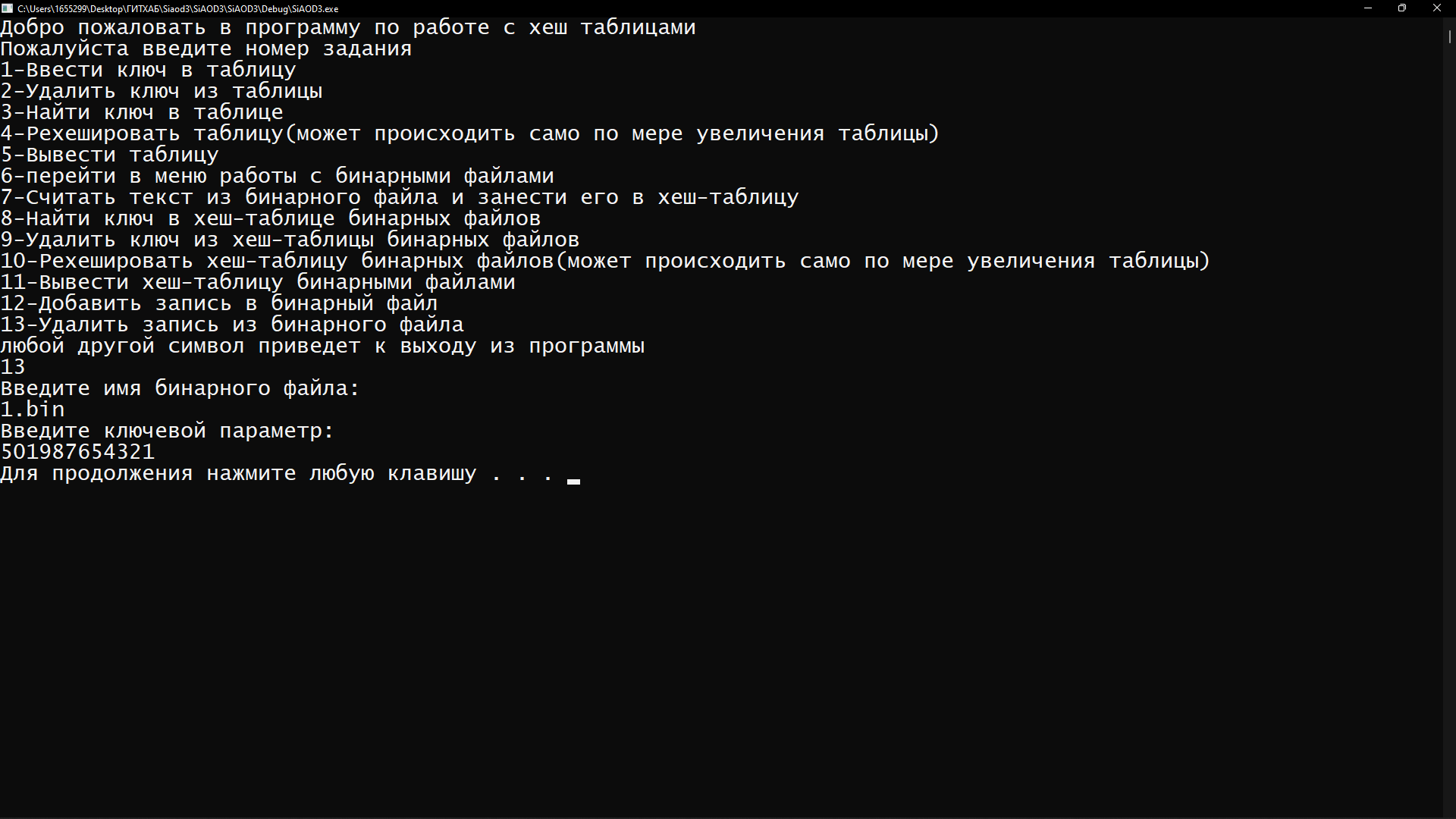
Удаление записи в хеш-таблице и в файле представлено на рисунке 15.

Рисунок 15. Удаление записи в хеш-таблице и в файле

Преобразование большого фапйла в хеш-таблицу представлено на рисункке 16.

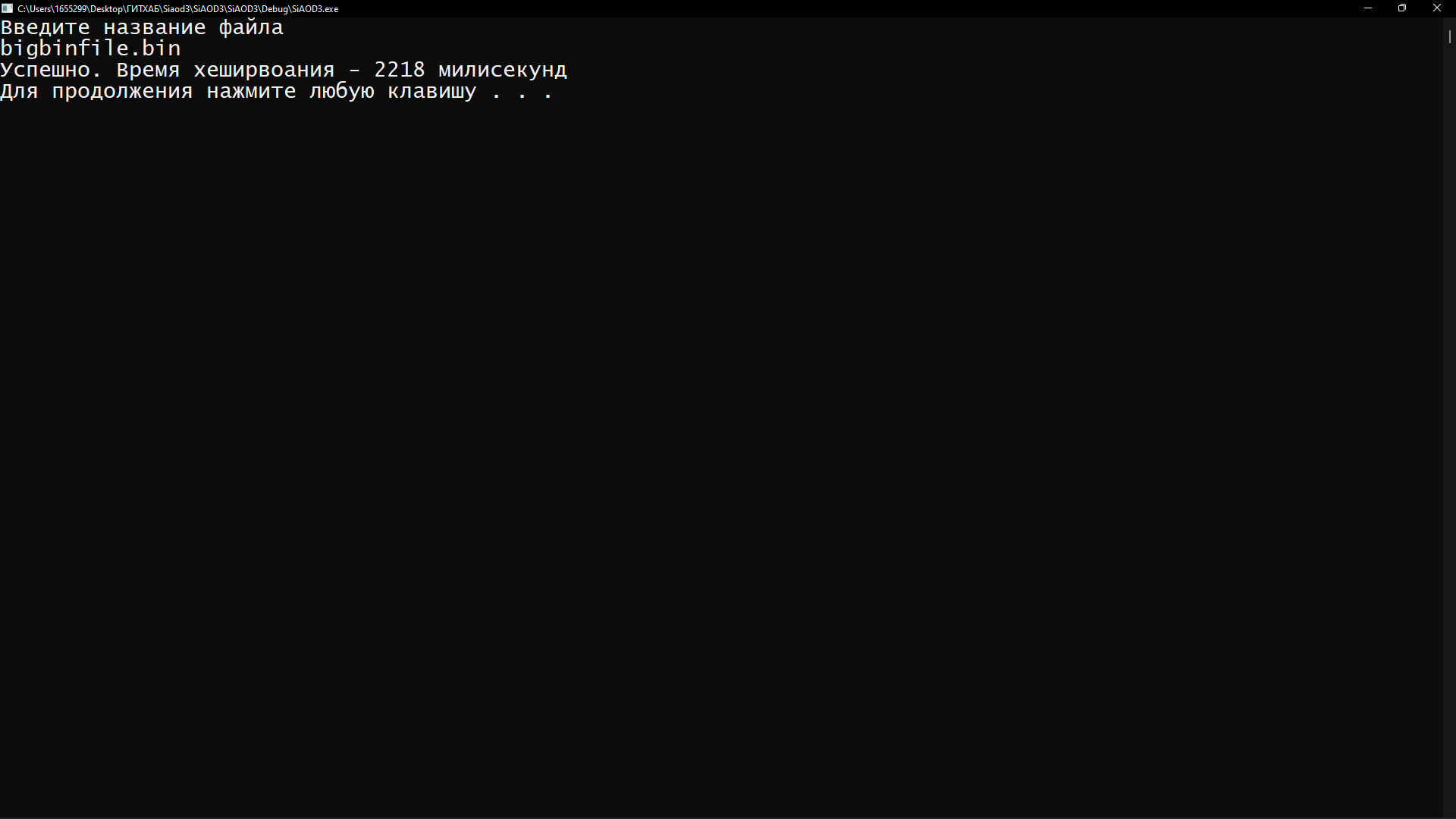


Рисунок 16. Преобразование большого фапйла в хеш-таблицу

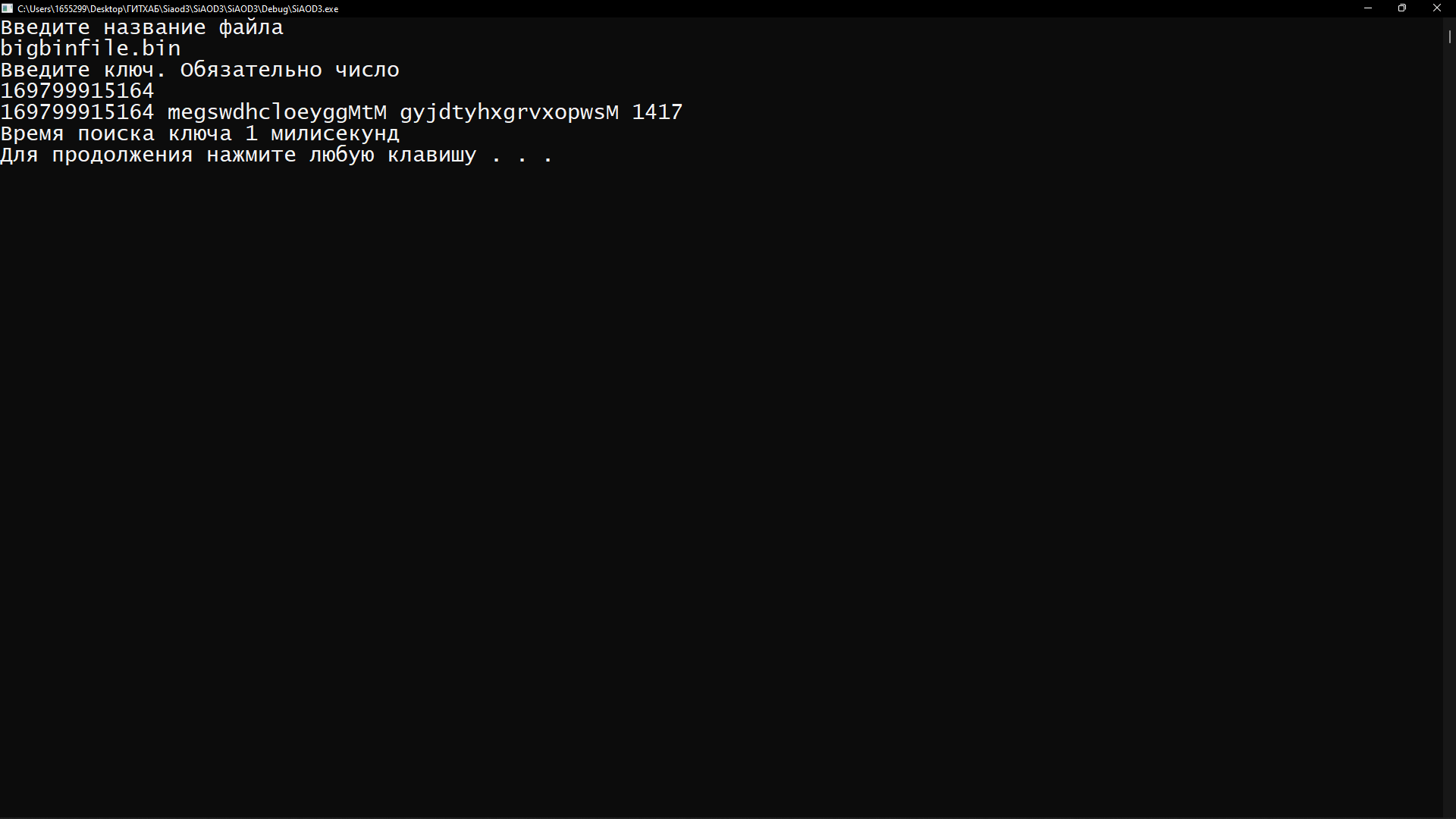
Поиск ключа в начале файла представлен на рисунке 17. 

Рисунок 17. Поиск ключа в начале файла

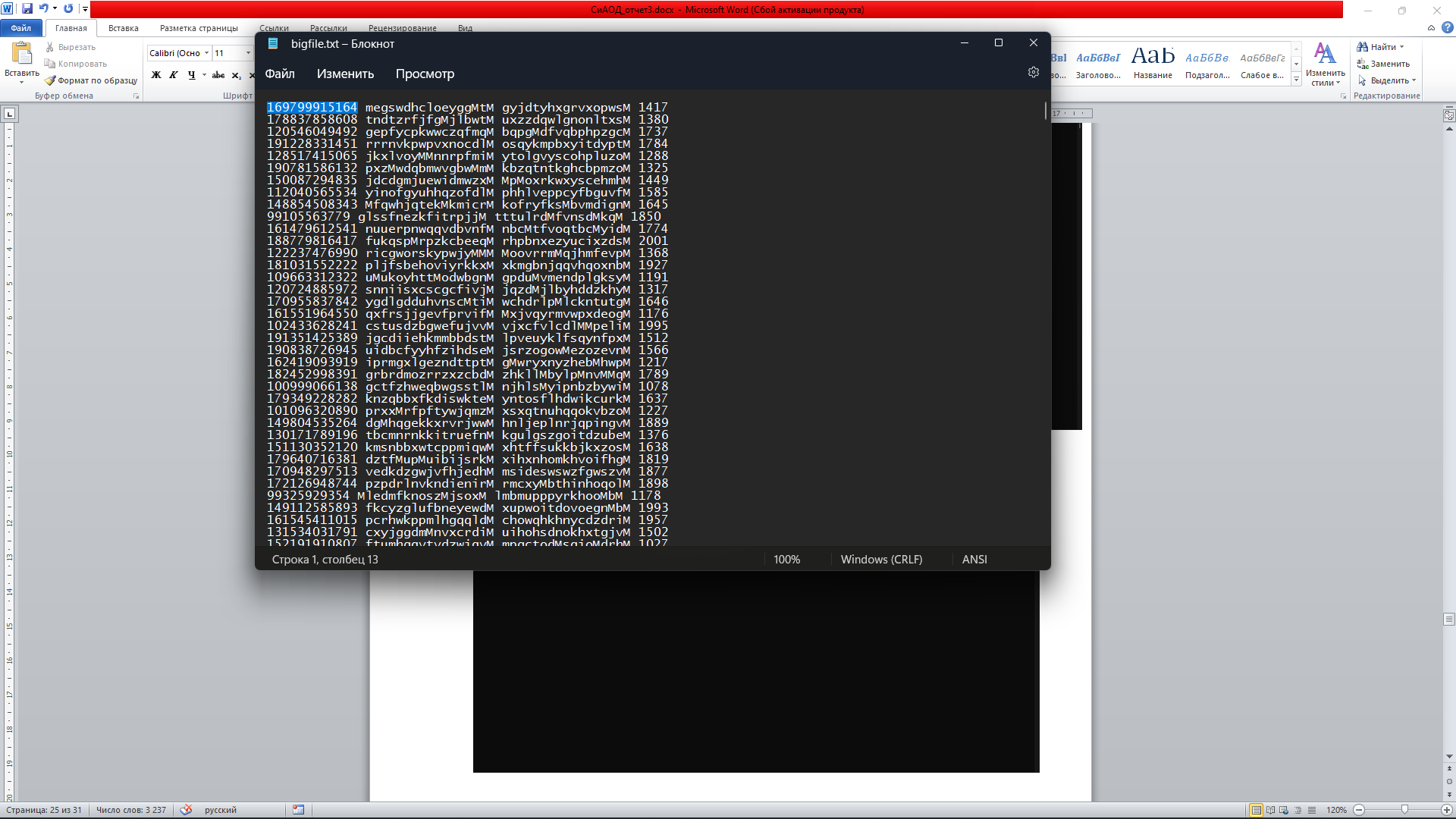
Подтверждение правильности поиска ключа в начале файла представлено на рисунке 18.

Рисунок 18. Подтверждение правильности поиска ключа в начале файла

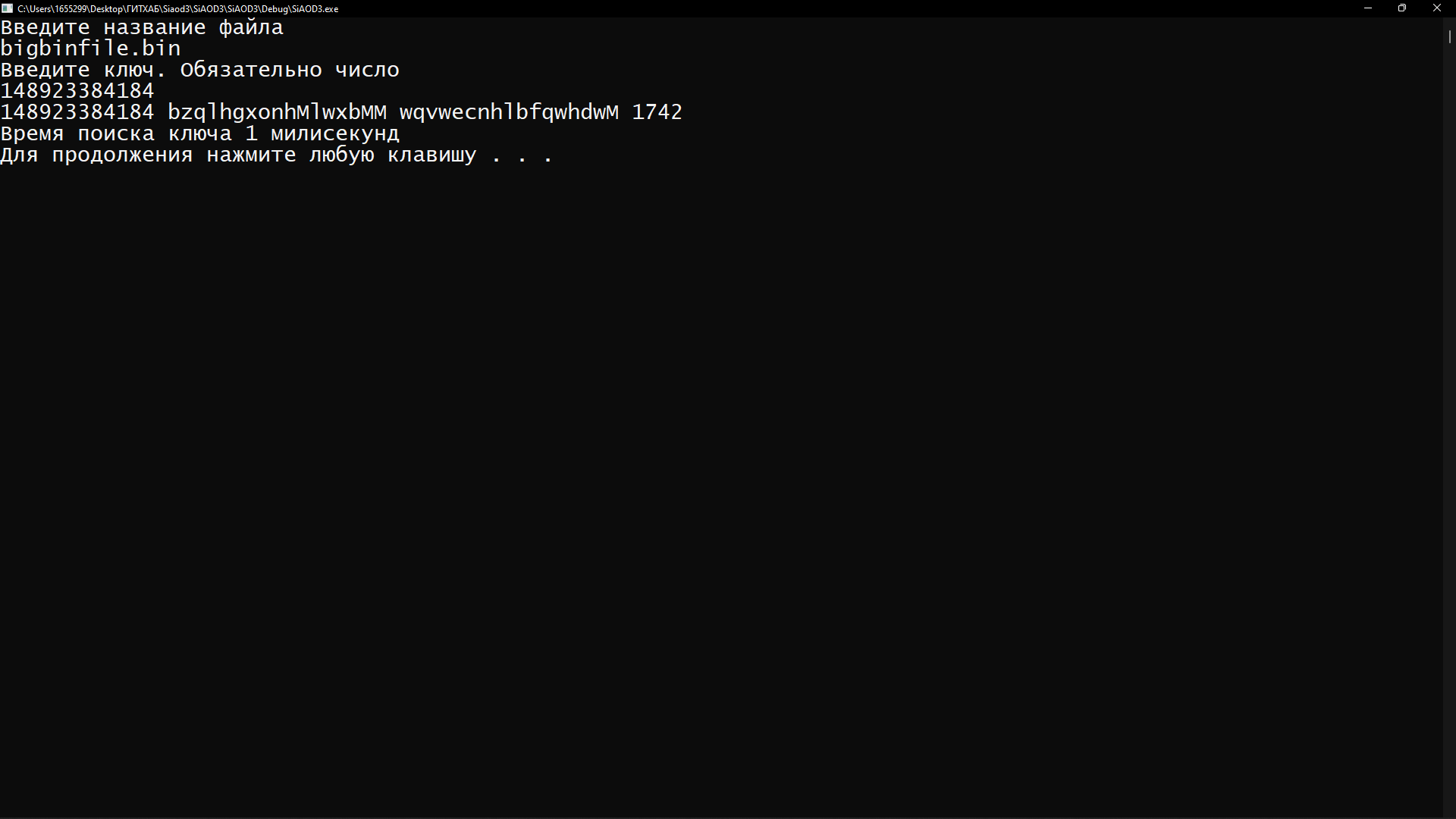
Поиск ключа в середине файла представлен на рисунке 19.

Рисунок 19. Поиск ключа в середине файла

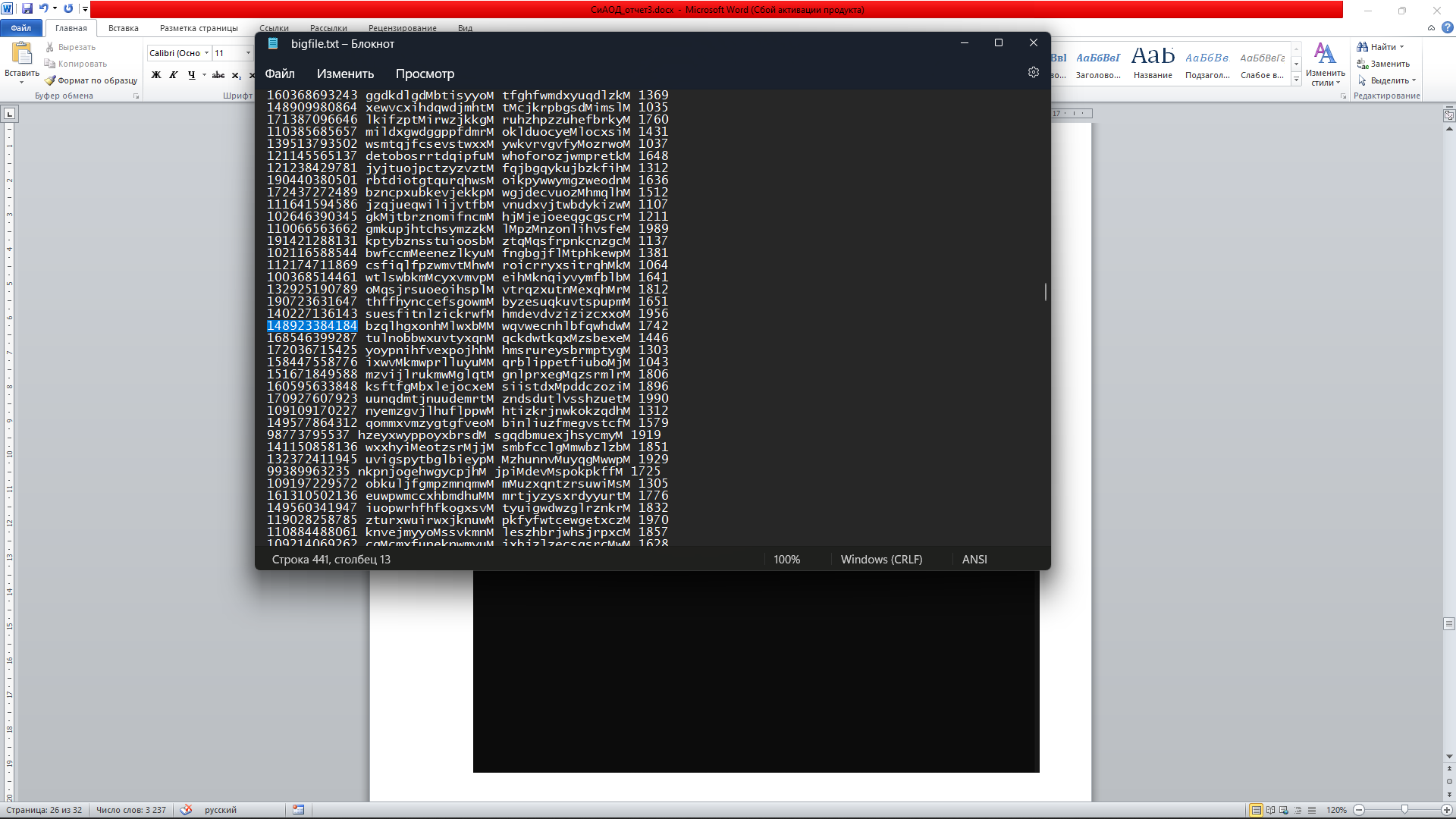
Подтверждение правильности поиска ключа в середине файла представлено на рисунке 20.

Рисунок 20. Подтверждение правильности поиска ключа в середине файла

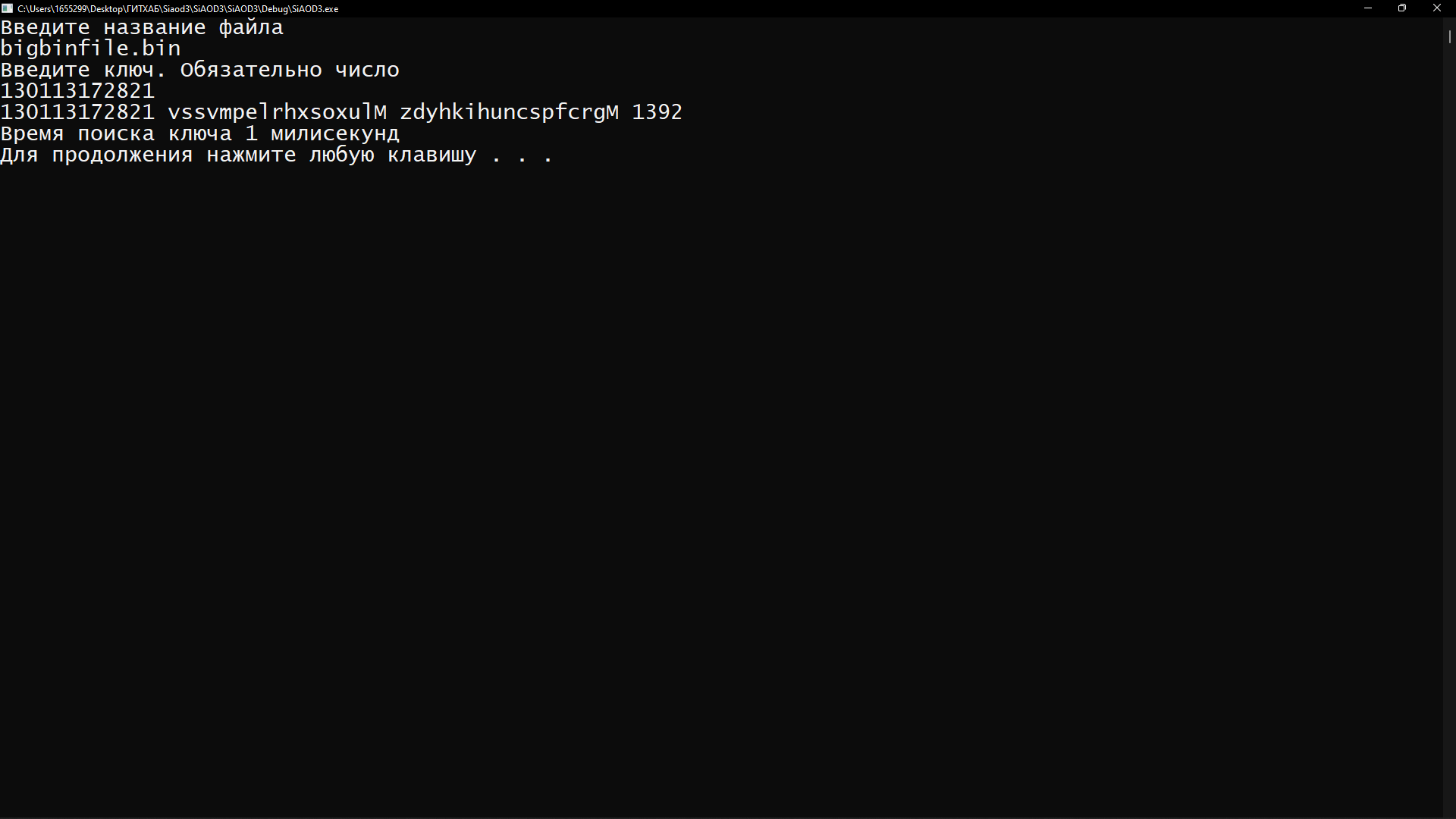
Поиск ключа в конце файла представлен на рисунке 21.

Рисунок 21. Поиск ключа в конце файла

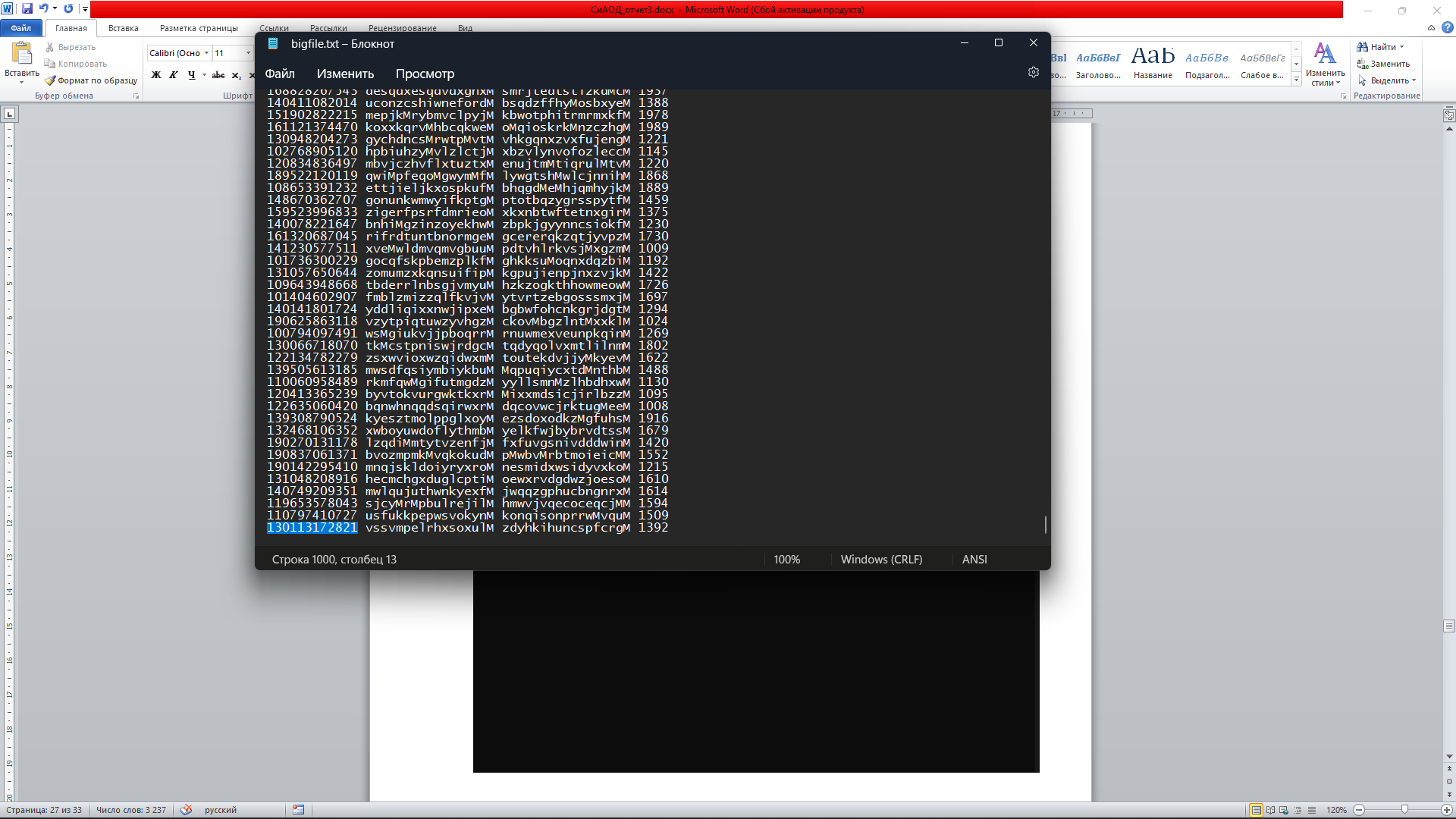
Подтверждение правильности поиска ключа в конце файла представлено на рисунке 22.

Рисунок 22. Подтверждение правильности поиска ключа в конце файла

Таким образом, программа работает правильно. Поиск ключей в конце, начале и середине файла занимает одинакоове время.

# Вывод

В ходе выполнения практической работы были выполнены все задания, а также:

1. Изучен принцип работы хеш-таблиц;
2. Улучшены знания о работе с бинарными файлами;
3. Улучшены знания о библиотеке «ctime»

# Исходный код программы

Source.cpp

|  |
| --- |
| #include "HashBin.h"  #include "HashTable.h"  #include <ctime>  void menubin()  {  ifstream inp;  ofstream outp;  string inp\_file, outp\_file;  system("cls");  cout << "Добро пожаловать в программу по работе с бинарными файлами\nПожалуйста введите номер задания\n1-Текстовый файл в бинарный\n2-Бинарный файл в текстовый\n5-найти запись с нужным ключом\n6-Вывести бинарный файл\nлюбой другой символ приведет к выходу из программы" << endl;  int k;  cin >> k;  switch (k)  {  case 1:  inp\_file = "";  outp\_file = "";  cout << "Введите имя текстового файла: \n";  cin >> inp\_file;  inp.open(inp\_file, ios::in);  if (inp.good())  {  cout << "Введите имя бинарного файла: \n";  cin >> outp\_file;  outp.open(outp\_file, ios::binary);  if (outp.good())  {  txt\_to\_bin(inp, outp);  }  else  {  cout << "Файл не найден или не создан.";  }  outp.close();  inp.close();  system("pause");  menubin();  }  else  {  cout << "Файла с таким именем не существует.";  system("pause");  menubin();  }  break;  case 2:  inp\_file = "";  outp\_file = "";  cout << "Введите имя бинарного файла:\n";  cin >> inp\_file;  inp.open(inp\_file, ios::binary | ios::in);  if (inp.good())  {  cout << "Введите имя текстового файла:\n";  cin >> outp\_file;  outp.open(outp\_file, ios::out);  if (outp.good())  {  bin\_to\_txt(inp, outp);  }  else  {  cout << "Файл не найден или не создан.";  }  outp.close();  inp.close();  system("pause");  menubin();  }  else  {  cout << " Файла с таким именем не существует.";  system("pause");  menubin();  }  break;  case 4:  {  inp\_file = "";  long long int key\_val;  cout << "Введите имя бинарного файла: \n";  cin >> inp\_file;  ifstream check(inp\_file);  if (check.is\_open())  {  cout << "Введите ключевой параметр:\n";  cin >> key\_val;  remove\_struct(inp\_file, key\_val);  system("pause");  menubin();  }  else  {  cout << "Файла с таким именем не существует. ";  system("pause");  menubin();  }  }  break;  case 5:  {  inp\_file = "";  int pos\_num;  cout << "Введите имя бинарного файла: \n";  cin >> inp\_file;  ifstream check(inp\_file);  if (check.is\_open())  {  cout << "Введите номер нужной строки: \n";  cin >> pos\_num;  ifstream tmp(inp\_file, ios::binary | ios::in);  int c = line\_counter(tmp) + 1;  if (pos\_num < c)  {  Book tt = get\_struct(inp\_file, pos\_num);  cout << tt.ISBN << ' ' << tt.fam << ' ' << tt.name << ' ' << tt.year << '\n';  system("pause");  menubin();  }  else  {  cout << "Введено слишком большое число.\n";  system("pause");  menubin();  }  }  else  {  cout << "Файла с таким именем не существует. ";  system("pause");  menubin();  }  }  break;  case 6:  {  inp\_file = "";  cout << "Введите имя бинарного файла:\n";  cin >> inp\_file;  inp.open(inp\_file, ios::in | ios::binary);  if (inp.good())  {  cout << "\nВывод содержимого бинарного файла:\n";  print\_bin(inp);  inp.close();  system("pause");  menubin();  }  else  {  cout << "Файла с таким именем не существует.";  system("pause");  menubin();  }  }  break;  default:  cout << "Вы уверены, что хотите выйти? Д/Н" << endl;  char u;  cin >> u;  if (u == 'Д')  {  exit(0);  }  else  {  menubin();  }  break;  }  }  char getchar(int k)  {  char let []= {'a','b','c','d','e','f','g','h','i','j','k','l','m','n','o','p','q','r','s','t','u','v','w','x','y','z'};  return let[k];  }  char word()  {  char word;  int k=rand()%26+1;  word = getchar(k);  return word;  }  void generate(int size)  {  ofstream l;  l.open("bigfile.txt", ios::app);  Book k;  char z[15];  for (long long int i = 1; i <= size; i++)  {  k.fam[17] = '\0';  k.name[17] = '\0';  k.ISBN = (rand() % 9 + 1 \* 100000000000) + (rand() % 10 \* 10000000000) + (rand() % 10 \* 1000000000) + (rand() % 10 \* 100000000) + (rand() % 10 \* 10000000)+(rand() % 10 \* 1000000) + (rand() % 10 \* 100000) + (rand() % 10 \* 10000) + (rand() % 10 \* 1000) + (rand() % 10 \* 100) + (rand() % 10 \* 10) + (rand() % 10);  for (int i = 0; i < 16; i++)  {  k.fam[i] = word();  k.name[i] = word();  }  k.year = (rand()%9+1\*1000)+(rand()%10\*100)+(rand()%10\*10)+(rand()%10);  l << k.ISBN << " " << k.fam << " " << k.name << " " << k.year << endl;  //cout << k.ISBN << " " << k.fam << " " << k.name << " " << k.year << endl;  }  l.close();  }  int main()  {  HashTable T;  T.CreateHashTable(20);  TypeItem item;  HashTablebin Tb;  Tb.CreateHashTable(20);  TypeItembin itembin;  ifstream inp;  ofstream outp;  string inp\_file, outp\_file;  int h;  long long int h1;  int i;  int j;  system("chcp 1251");  while (true)  {  system("cls");  cout << "Добро пожаловать в программу по работе с хеш таблицами\nПожалуйста введите номер задания\n1-Ввести ключ в таблицу\n2-Удалить ключ из таблицы\n3-Найти ключ в таблице\n4-Рехешировать таблицу(может происходить само по мере увеличения таблицы)\n5-Вывести таблицу\n6-перейти в меню работы с бинарными файлами\n7-Считать текст из бинарного файла и занести его в хеш-таблицу\n8-Найти ключ в хеш-таблице бинарных файлов\n9-Удалить ключ из хеш-таблицы бинарных файлов\n10-Рехешировать хеш-таблицу бинарных файлов(может происходить само по мере увеличения таблицы)\n11-Вывести хеш-таблицу бинарными файлами\n12-Добавить запись в бинарный файл\n13-Удалить запись из бинарного файла\nлюбой другой символ приведет к выходу из программы" << endl;  int k;  cin >> k;  switch (k)  {  case 1:  system("cls");  cout << "Введите ключ. Обязательно число"<<endl;  cin >> h;  cout << "Введите значение. Обязательно число"<<endl;  cin >> j;  if (InsertInHashTable(h, j, T) == 0)  {  cout << "Успешно";  system("pause");  break;  }  else  {  cout << "Возникли ошибки";  system("pause");  break;  }  break;  case 2:  system("cls");  cout << "Введите ключ. Обязательно число" << endl;  cin >> h;  if (deleteFromHashTable(T, h) == 0)  {  cout << "Успешно";  system("pause");  break;  }  else  {  cout << "Значение не найдено";  system("pause");  break;  }  break;  case 3:  system("cls");  cout << "Введите ключ. Обязательно число" << endl;  cin >> h;  i = search(T, h);  if (i != -1)  {  item = T.T[i];  cout << "ключ\tзначение" << endl;  cout << item.key << '\t' << item.offset << endl;  system("pause");  break;  }  else  {  cout << "Запись не найдена" << endl;  system("pause");  break;  }  break;  case 4:  system("cls");  if (reHash(T) == 1)  {  cout << "Успешно рехешировано" << endl;  system("pause");  break;  }  else  {  cout << "Возникли ошибки" << endl;  system("pause");  break;  }  break;  case 5:  system("cls");  outTable(T);  system("pause");  break;  break;  case 6:  menubin();  break;  case 7:  system("cls");  cout << "Введите название файла" << endl;  cin >> inp\_file;  inp.open(inp\_file, ios::binary);  if (inp.good())  {  unsigned int start\_time = clock();  if (ReadBin(Tb, inp) == 0)  {  unsigned int finish\_time = clock();  unsigned int rez\_time = finish\_time - start\_time;  cout << "Успешно. Время хеширвоания - "<<rez\_time<<" милисекунд" << endl;  system("pause");  inp.close();  break;  }  else  {  cout << "Возникли ошибки" << endl;  system("pause");  inp.close();  break;  }  }  else  {  cout << "Файл не найден" << endl;  system("pause");  inp.close();  break;  }  break;  case 8:  system("cls");  cout << "Введите название файла" << endl;  cin >> inp\_file;  inp.open(inp\_file, ios::binary);  if (inp.good())  {  cout << "Введите ключ. Обязательно число" << endl;  cin >> h1;  unsigned int start\_time = clock();  i = search(Tb, h1);  if (i != -1)  {  find(i, inp);  unsigned int finish\_time = clock();  unsigned int search\_time = finish\_time - start\_time;  cout << "Время поиска ключа " << search\_time<<" милисекунд"<< endl;  system("pause");  break;  }  else  {  cout << "Запись не найдена" << endl;  system("pause");  break;  }  }  else  {  cout << "Файл не найден" << endl;  system("pause");  break;  }  break;  case 9:  system("cls");  cout << "Введите ключ. Обязательно число" << endl;  cin >> h1;  if (deleteFromHashTable(Tb, h1) == 0)  {  cout << "Успешно";  system("pause");  break;  }  else  {  cout << "Значение не найдено";  system("pause");  break;  }  break;  case 10:  system("cls");  if (reHash(Tb) == 1)  {  cout << "Успешно рехешировано" << endl;  system("pause");  break;  }  else  {  cout << "Возникли ошибки" << endl;  system("pause");  break;  }  break;  case 11:  system("cls");  outTable(Tb);  system("pause");  break;  case 12:  {  Book temp;  string temp1;  outp\_file = "";  cout << "Введите имя бинарного файла:\n";  cin >> outp\_file;  outp.open(outp\_file, ios::binary | ios::in);  if (outp.good())  {  cout << "Введите ISBN" << endl;  cin >> temp.ISBN;  cout << "Введите Имя автора(английский язык)" << endl;  cin >> temp1;  strcpy(temp.fam, temp1.c\_str());  cout << "Введите название книги(английский язык)" << endl;  cin >> temp1;  strcpy(temp.name, temp1.c\_str());  cout << "Введите год издания" << endl;  cin >> temp.year;  AddToBin(outp, temp, Tb);  outp.close();  inp.open(outp\_file, ios::binary);  Tb.CreateHashTable(20);  ReadBin(Tb, inp);  inp.close();  system("pause");  }  else  {  cout << " Файла с таким именем не существует.";  system("pause");  }  }  break;  case 13:  {  inp\_file = "";  long long int key\_val;  cout << "Введите имя бинарного файла: \n";  cin >> inp\_file;  ifstream check(inp\_file);  if (check.is\_open())  {  cout << "Введите ключевой параметр:\n";  cin >> key\_val;  remove\_struct(inp\_file, key\_val);  Tb.CreateHashTable(20);  ReadBin(Tb, check);  check.close();  system("pause");  }  else  {  cout << "Файла с таким именем не существует. ";  system("pause");  }  }  break;  case 14:  {  ofstream l;  l.open("bigfile.txt");  l << "";  l.close();  generate(1000);  system("pause");  }  break;  default:  cout << "Вы уверены, что хотите выйти? Д/Н" << endl;  char u;  cin >> u;  if (u == 'Д')  {  exit(0);  }  else  {  break;  }  break;  }  }  system("pause");  } |

HashBin.h

|  |
| --- |
| #pragma once  #include <iostream>  #include <fstream>  #include <string>  using namespace std;  int i = 0;  struct Book  {  long long int ISBN;  char fam[50];  char name[50];  int year;  };  struct TypeItembin  {  long long int key = 0;  int offset = 0;  bool openOrClose = true;  bool deleteOrnot = false;  };  struct HashTablebin  {  int L;  TypeItembin \*T;  int insertedCount;  int deletedCount;  void CreateHashTable(int k)  {  T = new TypeItembin[k];  L = k;  insertedCount = 0;  deletedCount = 0;  }  };  int InsertInHashTable(long long int key, int offset, HashTablebin& t);  int search(HashTablebin& t, long long int key);  int reHash(HashTablebin& old)  {  int size = old.L;  size = size + 10;  HashTablebin new\_hash;  new\_hash.CreateHashTable(size);  for (int i = 0; i < size - 10; i++)  {  if (old.T[i].key != 0)  {  InsertInHashTable(old.T[i].key, old.T[i].offset, new\_hash);  int t = search(new\_hash,old.T[i].key);  new\_hash.T[t].deleteOrnot = old.T[i].deleteOrnot;  new\_hash.T[t].openOrClose = old.T[i].openOrClose;  }  }  old = new\_hash;  return 1;  }  int Hash(long long int k, int L, int iter)  {  long long int h1;  if (k % 2 == 0)  {  h1 = k << 2 % 9;  }  else  {  h1 = 9 % k << 2;  }  h1++;  long long int h2;  if (h1 % 2 == 0)  {  h2 = h1 % 7;  }  else  {  h2 = 7 % h1;  }  h2++;  return (h1 + iter\*h2) % L;  }  int InsertInHashTable(long long int key, int offset, HashTablebin& t)  {  if (t.insertedCount >= t.L\*0.75)  {  reHash(t);  }  int iteration = 0;  int i = Hash(key, t.L, iteration);  while (i < t.L&&t.T[i].openOrClose == false)  {  if (iteration > t.L)  {  reHash(t);  }  iteration++;  i = Hash(key, t.L, iteration);  }  while (i > t.L)  {  return -1;  }  if (i < t.L)  {  t.T[i].key = key;  t.T[i].offset = offset;  t.T[i].openOrClose = false;  t.insertedCount++;  return 0;  }  else  {  return 1;  }  }  void outTable(HashTablebin& t)  {  cout << "номер" << '\t' << "ISBN" << '\t' << '\t' << "Номер записи" << '\t' << "свободен" << '\t' << "удален" << '\n';  for (int i = 0; i < t.L; i++)  {  if (t.T[i].key == 0)  {  cout << i << '\t' << t.T[i].key << '\t' << '\t' << t.T[i].offset << '\t'<<'\t' << t.T[i].openOrClose << '\t' << '\t' << t.T[i].deleteOrnot << '\n';  }  else  {  cout << i << '\t' << t.T[i].key << '\t' << t.T[i].offset << '\t' << '\t' << t.T[i].openOrClose << '\t' << '\t' << t.T[i].deleteOrnot << '\n';  }  }  }  int search(HashTablebin& t, long long int key)  {  int iterator = 0;  int i = Hash(key, t.L, iterator);  while (i < t.L && ((t.T[i].openOrClose == false && t.T[i].deleteOrnot == false) || (t.T[i].openOrClose == true && t.T[i].deleteOrnot == true)) && t.T[i].key != key)  {  iterator++;  i = Hash(key, t.L, iterator);  }  if (t.T[i].openOrClose == true && t.T[i].deleteOrnot == false)  {  return -1;  }  return t.T[i].offset;  }  void find(int i, ifstream& file)  {  Book book;  file.seekg(i\*sizeof(Book),ios::beg);  file.read((char\*)&book, sizeof(book));  cout << book.ISBN << ' ' << book.fam << ' ' << book.name << ' ' << book.year << '\n';  }  int deleteFromHashTable(HashTablebin& t, long long int key)  {  int i = search(t, key);  if (i == -1)  {  return 1;  }  else  {  t.T[i].key = 0;  t.T[i].offset = 0;  t.T[i].deleteOrnot = true;  t.T[i].openOrClose = true;  t.deletedCount++;  return 0;  }  }  int ReadBin(HashTablebin& t, ifstream& bin\_file)  {  Book book;  bool isGood = true;  while (true)  {  bin\_file.read((char\*)&book, sizeof(book));  if (!bin\_file.eof())  {  if (InsertInHashTable(book.ISBN, i, t) != 0)  {  isGood = false;  }  i++;  }  else  {  if (isGood)  {  return 0;  }  else  {  return 1;  }  }  }  }  //----------------------------------------  int line\_counter(ifstream& file)  {  string num\_string = "";  int counter = 1;  while (!file.eof())  {  while (getline(file, num\_string))  {  counter++;  }  }  return counter;  }  void txt\_to\_bin(ifstream& txt\_file, ofstream& bin\_file)  {  Book book;  while (true)  {  txt\_file >> book.ISBN;  if (!txt\_file.eof())  {  txt\_file >> book.fam >> book.name >> book.year;  bin\_file.write((char\*)&book, sizeof(Book));  }  else break;  }  }  void AddToBin(ofstream& bin\_file, Book book, HashTablebin& t)  {  bin\_file.seekp(0,ios::end);  bin\_file.write((char\*)&book, sizeof(Book));  }  void bin\_to\_txt(ifstream& bin\_file, ofstream& txt\_file)  {  Book book;  while (true)  {  bin\_file.read((char\*)&book, sizeof(book));  if (!bin\_file.eof())  {  txt\_file << book.ISBN << " " << book.fam << " " << book.name << " " << book.year << " ";  }  else  {  break;  }  }  }  void print\_bin(ifstream& bin\_file)  {  Book book;  while (true)  {  bin\_file.read((char\*)&book, sizeof(book));  if (!bin\_file.eof())  {  cout << book.ISBN << ' ' << book.fam << ' ' << book.name << ' ' << book.year << '\n';  }  else break;  }  }  Book get\_struct(string filename, int position)  {  Book book;  ifstream bin\_file(filename, ios::binary | ios::in);  if (position == -1)  {  bin\_file.seekg(-1 \* (int)sizeof(book), ios::end);  }  else  {  bin\_file.seekg((position) \* sizeof(book));  }  bin\_file.read((char\*)&book, sizeof(Book));  bin\_file.close();  return book;  }  void remove\_struct(string bin\_name, long long key)  {  Book book, replace{};  replace = get\_struct(bin\_name, -1);  ifstream bin\_file(bin\_name, ios::binary | ios::in);  ofstream tmp("tmp\_rem.txt");  while (true)  {  bin\_file.read((char\*)&book, sizeof(book));  if (!bin\_file.eof())  {  if (book.ISBN != key)  {  tmp << book.ISBN << " " << book.fam << " " << book.name << " " << book.year << "\n";  }  else  {  tmp << replace.ISBN << " " << replace.fam << " " << replace.name << " " << replace.year << "\n";  }  }  else  {  break;  }  }  tmp.close();  bin\_file.close();  ifstream txt("tmp\_rem.txt", ios::in);  ofstream bin(bin\_name, ios::binary);  txt\_to\_bin(txt, bin);  txt.close();  bin.close();  } |

HashTable.h

|  |
| --- |
| #pragma once  struct TypeItem  {  int key = 0;  int offset = 0;  bool openOrClose = true;  bool deleteOrnot = false;  };  struct HashTable  {  int L;  TypeItem \*T;  int insertedCount;  int deletedCount;  void CreateHashTable(int k)  {  T = new TypeItem[k];  L = k;  insertedCount = 0;  deletedCount = 0;  }  };  int InsertInHashTable(int key, int offset, HashTable& t);  int search(HashTable& t, int key);  int reHash(HashTable& old)  {  int size = old.L;  size = size + 10;  HashTable new\_hash;  new\_hash.CreateHashTable(size);  for (int i = 0; i < size - 10; i++)  {  if (old.T[i].key != 0)  {  InsertInHashTable(old.T[i].key, old.T[i].offset, new\_hash);  int j = search(new\_hash, old.T[i].key);  new\_hash.T[j].deleteOrnot = old.T[i].deleteOrnot;  new\_hash.T[j].openOrClose = old.T[i].openOrClose;  }  }  old = new\_hash;  return 1;  }  int Hash(int k, int L, int iter)  {  int h1;  if (k % 2 == 0)  {  h1 = k<<2%9;  }  else  {  h1 = 9%k<<2;  }  h1++;  int h2;  if (h1 % 2 == 0)  {  h2 = h1 % 7;  }  else  {  h2 = 7 % h1;  }  h2++;  return (h1 + iter\*h2)%L;  }  int InsertInHashTable(int key, int offset, HashTable& t)  {  if (t.insertedCount >= t.L\*0.75)  {  reHash(t);  }  int iteration = 0;  int i = Hash(key, t.L, iteration);  while (i < t.L&&t.T[i].openOrClose == false)  {  if (iteration > t.L)  {  reHash(t);  }  iteration++;  i = Hash(key, t.L, iteration);  }  if (i < t.L)  {  t.T[i].key = key;  t.T[i].offset = offset;  t.T[i].openOrClose = false;  t.insertedCount++;  return 0;  }  else  {  return 1;  }  }  void outTable(HashTable& t)  {  cout << "номер" << '\t' << "ключ" << '\t' << "значение" << '\t' << "свободен" << '\t' << "удален" << '\n';  for (int i = 0; i < t.L; i++)  {  cout << i << '\t' << t.T[i].key << '\t' << t.T[i].offset << '\t' << '\t' << t.T[i].openOrClose << '\t' << '\t' << t.T[i].deleteOrnot << '\n';  }  }  int search(HashTable& t, int key)  {  int iterator = 0;  int i = Hash(key, t.L, iterator);  while (i < t.L && ((t.T[i].openOrClose == false && t.T[i].deleteOrnot == false) || (t.T[i].openOrClose == true && t.T[i].deleteOrnot == true)) && t.T[i].key != key)  {  iterator++;  i = Hash(key, t.L, iterator);  }  if (t.T[i].openOrClose == true && t.T[i].deleteOrnot == false)  {  return -1;  }  return i;  }  int deleteFromHashTable(HashTable& t, int key)  {  int i = search(t, key);  if (i == -1)  {  return 1;  }  else  {  t.T[i].key = 0;  t.T[i].offset = 0;  t.T[i].deleteOrnot = true;  t.T[i].openOrClose = true;  t.deletedCount++;  return 0;  }  } |