100

Test✓

mem✓

```
/*
* @author Cole Van Verth
* @pengo cverth
* @email colevanverth@gmail.com
* @file hashtable.cpp
* @assignment 5: Hash Table
*/

#include "hashtable.h"

HashTable::HashTable(int size)
   : m_size(size) {
      m_table = new std::vector<Record>[m_size];
}

HashTable::~HashTable() {
   delete[] m_table;
}

void HashTable::insert(Record record) {
   auto& list = m_table[hash(record)];
   list.push_back(record);
}

void HashTable::clear() {
   for (int i = 0; i < m_size; ++i) {
      m_table[i].clear();
   }
}

void HashTable::write(std::string filename) {
   // Opens file for write
   std::ofstream fout;
   fout.open(filename);
   if (fout.fail()) {
      std::cerr << "Failed to open " << filename << " for write" << std::endl;
   }

   // Places data into file
   for (int i = 0; i < m_size; ++i) {
      for (auto record : m_table[i]) {
         fout << std::setfill('0') << std::setw(9) <<
         record.id << " " << record.data << std::endl;
      }
   }

   fout.close(); // Closes file
}

void HashTable::merge(std::string filename) {
   std::string input; // Temp input to hold record data when loading records

   // Opens file for read
   std::ifstream fin;
   fin.open(filename);
   if (fin.fail()) {
      std::cerr << "Failed to open " << filename << " for merge" << std::endl;
   }

   // Load data from file
   while (std::getline(fin, input)) {
      int spaceIndex = input.find(' '); // Occurs directly after id
      Record record;
      record.id = std::stoi(input.substr(0, spaceIndex));
      record.data = input.substr(spaceIndex + 1);
      insert(record);
```

```
   }

   fin.close(); // Closes file
}

Record* HashTable::remove(int id) {
   auto& chain = m_table[hash(id)]; // Vector associated with ‘id‘
   auto recordIterator = find(id);
   if (recordIterator != chain.end()) {
      auto recordCopy = new Record(*recordIterator);
      chain.erase(recordIterator);
      return recordCopy;
   }
   return nullptr;
}

Record* HashTable::search(int id) {
   auto& chain = m_table[hash(id)]; // Vector associated with ‘id‘
   auto recordIterator = find(id);
   if (recordIterator != chain.end()) {
      return new Record(*recordIterator);
   }
   return nullptr;
}

int HashTable::hash(Record record) {
   return hash(record.id);
}

int HashTable::hash(int key) {
   double base = (double)(m_hashConstant * key); // = xxxxx.yyyyy
   int integer = (int)(base); // xxxxx.00000
   double preHash = (double)(base - integer); // 00000.yyyyy
   return (int)(preHash * m_size); // Returns an index within m_size
}

std::vector<Record>::iterator HashTable::find(int id) {
   auto& chain = m_table[hash(id)];
   for (auto it = chain.begin(); it != chain.end(); it++) {
      if (it->id == id) {
         return it;
      }
   }
   return chain.end();
}
```

```
/*
* @author Cole Van Verth
* @pengo cverth
* @email colevanverth@gmail.com
* @file hashtable.h
* @assignment 5: Hash Table
*/

#pragma once

#include <vector>
#include <iomanip>
#include <fstream>
#include <string>
#include <iostream>

#include "record.h"

/**
* @brief ‘HashTable‘ implements a hashtable that stores ‘Record‘ objects.
* Hashes are calculated using a multiplication method. Collisions are resolved
* with chaining; thus, ‘Record‘ retrieval has linear time complexity
* in the best case, and O(n) time complexity in the worst case. Essential
* methods for manipulating the hashtable are provided, including reading from
* and writing to a file.
*/
class HashTable {
public:
   /**
   * ‘HashTable‘ constructor.
   * @param ‘size‘ size of hash table (default 100)
   */
   HashTable(int size = 100);

   /**
   * ‘HashTable‘ deconstructor. Deletes heap allocated array ‘m_table‘ of
   * vectors.
   */
   ~HashTable();

   /**
   * Inserts a record into the hashtable
   * @param ‘record‘ Record to be inserted into ‘m_table‘
   */
   void insert(Record record);

   /**
   * Deletes a record from the hashtable.
   * @param ‘id‘ int id of record in ‘m_table‘ to delete
   * @return pointer to a heap allocated copy of a record that was deleted if
   * there was a record corresponding to ‘id‘, else nullptr
   */
   Record* remove(int id);

   /*
   * Searchs for a record in the hashtable.
   * @param ‘id‘ id of record to search for in ‘m_table‘
   * @return pointer to a heap allocated copy of a record if there was a record
   * corresponding to ‘id‘, else nullptr
   */
   Record* search(int id);

   /**
   * Clears all entries in all vectors within ‘m_table‘.
   */
   void clear();
```

```

   /**
   * Merges a file containg hash entries into the hashtable
   * @param ‘filename‘ name of file to load entries from
   */
   void merge(std::string filename);

   /**
   * Writes the hashtable to a file.
   * @param ‘filename‘ name of the file to load entries into
   */
   void write(std::string filename);

private:
   /**
   * Helper function that finds the iterator associated with a record.
   * @param ‘id‘ id of record to find
   * @return iterator associated with record in vector "chain"
   * if it was found, else chain.end()
   */
   std::vector<Record>::iterator find(int id);

   /**
   * Finds the hash value for a record.
   * @param ‘record‘ pointer to record
   * @return hash corresponding to the ‘id‘ within the ‘record‘
   */
   int hash(Record record);

   /**
   * Finds the hash value for a key using ‘m_hashConstant‘ with a
   * multiplication method.
   * @param ‘key‘ key to calculate hash from
   * @return hash value calculated from ‘key‘
   */
   int hash(int key);

   const int m_size; // Capacity of ‘m_table‘

   std::vector<Record>* m_table; // Array of vector "chains"

   const double m_hashConstant = 0.618034; // Hash constant (inverse of golden rat

};
```

```
/**
* This program implements a hash table using chaining and utilizes a
* multiplication method for generating hash keys. Main provides I/O that allows
* the user to interact with the hash table class.
*
* The program works to all specifications and compiles with no warnings on
* Pengo. All methods were tested by hand. No memory leaks were found and I/O was
* tested with redirection to ensure it would work for the grading scripts.
*/
/*
* @author Cole Van Verth
* @pengo cverth
* @email colevanverth@gmail.com
* @file main.cpp
* @assignment 5: Hash Table
*/

#include <iostream>
#include <string>

#include "record.h"
#include "hashtable.h"

int main() {
   std::string input;
   HashTable table(178000);
   bool running = true;

   while (running) {
      // Print menu to user
      std::cout << "(1)load (2)insert (3)delete (4)search (5)clear"
      << " (6)save (7)quit -- Your choice? ";

      // Load user input
      std::getline(std::cin, input);

      switch(std::stoi(input)) {
         // Table merge
         case 1: {
            std::cout << "read hash table - filename? ";
            std::getline(std::cin, input);
            table.merge(input);
            break;
         }

         // Record insert
         case 2: {
            std::cout << "input new record: ";
            std::getline(std::cin, input);
            int spaceIndex = input.find(' '); // Occurs directly after id
            Record record;
            record.id = std::stoi(input.substr(0, spaceIndex));
            record.data = input.substr(spaceIndex + 1);
            table.insert(record);
            break;
         }

         // Record delete
         case 3: {
            std::cout << "delete record - key? ";
            std::getline(std::cin, input);
            auto recordCopy = table.remove(std::stoi(input));
            if (recordCopy) {
               std::cout << "Delete: " << recordCopy->id << " " << recordCopy->dat
               delete recordCopy;
            }
```

```
            else {
               std::cout << "Delete not found: " << input << std::endl;
            }
            break;
         }

         // Record search
         case 4: {
            std::cout << "search for record - key? ";
            std::getline(std::cin, input);
            Record* recordCopy = table.search(std::stoi(input));
            if (recordCopy) {
               std::cout << "Found: " << recordCopy->id << " " << recordCopy->data
               delete recordCopy;
            }
            else {
               std::cout << "Search not found: " << input << std::endl;
            }
            break;
         }

         // Table clear
         case 5: {
            table.clear();
            std::cout << "clearing hash table." << std::endl;
            break;
         }

         // Table write
         case 6: {
            std::cout << "write hash table - filename? ";
            std::getline(std::cin, input);
            table.write(input);
            break;
         }

         // End program
         case 7: {
            running = false;
            break;
         }
      }
   }
}
```

```
p5: main.o hashtable.o record.o
   g++ -o p5 main.o hashtable.o record.o

main.o: main.cpp
   g++ -c main.cpp

hashtable.o: hashtable.cpp hashtable.h
   g++ -c hashtable.cpp

record.o: record.cpp record.h
   g++ -c record.cpp

clean:
   rm -f p5 *.o *~
```

```
/*
* @author Cole Van Verth
* @pengo cverth
* @email colevanverth@gmail.com
* @file record.cpp
* @assignment 5: Hash Table
*/

#include "record.h"

Record::Record() {}

Record::Record(int id, std::string data)
   : id(id), data(data) {}

Record::Record(const Record& other) {
   id = other.id;
   data = other.data;
}
```

```
/*
* @author Cole Van Verth
* @pengo cverth
* @email colevanverth@gmail.com
* @file record.h
* @assignment 5: Hash Table
*/

#pragma once

#include <string>

/**
* @brief ‘Record‘ objects are data containers that store an int id and string
* data. ‘Record‘ objects are the unit of storage for ‘HashTable‘ objects.
*/
struct Record {
   /**
   * ‘Record‘ default constructor.
   */
   Record();

   /**
   * ‘Record‘ constructor with parameters.
   * @param ‘id‘ id of the record
   * @param ‘data‘ data stored in the record
   */
   Record(int id, std::string data);

   /**
   * ‘Record‘ copy constructor.
   * @param ‘other‘ reference of Record to copy
   */
   Record(const Record& other);

   int id; // Record key

   std::string data; // Record data
};
```

```
------ RUN ---------
(1)load (2)insert (3)delete (4)search (5)clear (6)save (7)quit -- Your choice?
search for record - key? Search not found: 925525955 ✓
(1)load (2)insert (3)delete (4)search (5)clear (6)save (7)quit -- Your choice?
read hash table - filename? (1)load (2)insert (3)delete (4)search (5)clear
(6)save (7)quit -- Your choice? delete record - key? Delete: 925525955 is one ✓
plus 925525954 copy one
(1)load (2)insert (3)delete (4)search (5)clear (6)save (7)quit -- Your choice?
search for record - key? Found: 925525955 is one plus 925525954 copy two ✓
(1)load (2)insert (3)delete (4)search (5)clear (6)save (7)quit -- Your choice?
delete record - key? Delete: 925525955 is one plus 925525954 copy two ✓
(1)load (2)insert (3)delete (4)search (5)clear (6)save (7)quit -- Your choice?
search for record - key? Found: 925525955 is one plus 925525954 copy three ✓
(1)load (2)insert (3)delete (4)search (5)clear (6)save (7)quit -- Your choice?
delete record - key? Delete: 925525955 is one plus 925525954 copy three ✓
(1)load (2)insert (3)delete (4)search (5)clear (6)save (7)quit -- Your choice?
search for record - key? Search not found: 925525955 ✓
(1)load (2)insert (3)delete (4)search (5)clear (6)save (7)quit -- Your choice?
read hash table - filename? (1)load (2)insert (3)delete (4)search (5)clear
(6)save (7)quit -- Your choice? write hash table - filename? (1)load (2)insert
(3)delete (4)search (5)clear (6)save (7)quit -- Your choice? ----- OUT DIFF ETC
```

```
-----
Files hout.s and out.s are identical     ✓
 170099  850495 5424292 hout.s
 170099  850495 5424292 out.s
----- VALGRIND -----
==3021840== Memcheck, a memory error detector
==3021840== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==3021840== Using Valgrind-3.18.1 and LibVEX; rerun with -h for copyright info
==3021840== Command: ./p5
==3021840==
==3021840==
==3021840== HEAP SUMMARY:
==3021840==     in use at exit: 0 bytes in 0 blocks
==3021840==   total heap usage: 1,090,256 allocs, 1,090,256 frees, 41,630,184
bytes allocated
==3021840==
==3021840== All heap blocks were freed -- no leaks are possible     ✓
==3021840==
==3021840== For lists of detected and suppressed errors, rerun with: -s
==3021840== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```