CS 3100 – *Data Structures and Algorithms*

Project #4 - Hash Table Indexing

Learning Objectives

- Implement a data structure to meet given specifications
- Design, implement, and use a closed hash table data structure
- Use a hash table as an index into a separate data store

Overview

Your task for this assignment is to implement a database of student records. Your database will consist of two parts: an unsorted vector of student records, and an index of student IDs implemented as a closed hash table.

The Record Store

Student records will be stored in an unsorted vector of class Record. The definition of class Record will be provided for you in the file `Record.h`.

The HashTable class

Your hash table should be implemented as an array of `MAXHASH` objects of class Slot. A Slot contains an integer key, and an integer value. The definition of class Slot will be provided for you in the file `Slot.h`. The value of `MAXHASH` should initially be `#defined` to 1000.

To implement the hash table, you should create a class called HashTable, implemented in the files HashTable.h and HashTable.cpp. Your class should support the following operations:

- **`bool HashTable::insert(int key, int value, int& collisions)`** – Insert a new key/value pair into the table. Duplicate keys are not allowed. This function should return **`true`** if the key/value pair is successfully inserted into the hash table, and **`false`** if the pair could not be inserted (for example, due to a duplicate key already found in the table). If the insertion is successful, the number of collisions occuring during the insert operation should be returned in **`collisions`**.

- **`bool HashTable::remove(int key)`** – If there is a record with the given key in the hash table, it is deleted and the function returns **`true`**; otherwise the function returns **`false`**.

- **`bool HashTable::find(int key, int& value)`** – If a record with the given key is found in the hash table, the function returns **`true`** and a copy of the value is returned in `value`. Otherwise the function returns **`false`**.

- **`float HashTable::alpha()`** - returns the current loading factor, α, of the hash table.

- Your hash table should also overload **`operator<<`** such that `cout << myHashTable` prints all the key/value pairs in the table, along with their hash table slot number, to cout. *operator<< should not print empty or tombstone slots!*

The hash and probe functions

Because this is a closed hash, you will need both a hash function and a probe function. A hash function will be provided for you in the file `hashfunction.h`. Your hash must use **pseudo-random probing** to resolve collisions.

Main program

You should use your student database in a main program that allows a user to insert, search, and delete from the database. Searching the database by UID should be done using the hash table, and should report the number of collisions encountered during the search.

Deleting from the database removes both the hash table (index) entry, and the record store (vector) entry. **You must delete records from the record store in a way that does not waste memory, and does not break the index**. One way to do this is to copy the last record in the vector into the position holding the record to be deleted, and then use pop_back() to remove the last element of the vector.

Example program operation

```
Would you like to (I)nsert or (D)elete a record, or (S)earch for a record, or (Q)uit?
Enter action:  I
Inserting a new record.
Last name:  Doe
First name:  Jane
UID: 1234
Year:  Junior
Record inserted.

Would you like to (I)nsert or (D)elete a record, or (S)earch for a record, or (Q)uit?
Enter action:  S
Enter UID to search for: 1234
Searching... record found (3 collisions during search)
----------------------------
Last name:  Doe
First name:  Jane
UID: 1234
Year:  Junior
----------------------------

Would you like to (I)nsert or (D)elete a record, or (S)earch for a record, or (Q)uit?
Enter action:  S
Enter UID to search for: 2345
```

```
Searching... record not found

Would you like to (I)nsert or (D)elete a record, or (S)earch for a record, or (Q)uit?
Enter action:  Q
Exiting.
```

Turn in and Grading

Please zip your entire project directory into a single file called Project4.zip.

This project is worth 50 points, distributed as follows:

Task	**Points**
`HashTable::insert` stores key/value pairs in the hash table using appropriate hashing and collision resolution, and correctly rejects duplicate keys and reports correct collision counts. Insert correctly re-uses space from previously-deleted records.	10
`HashTable::find` correctly finds records in the hash table using appropriate hashing and collision resolution, and returns either the value associated with the search key or **`false`** when the requested key is not present in the hash table. Database searches are performed using the HashTable::find() member function.	10
`HashTable::remove` correctly finds and deletes records from the table, without interfering with subsequent search or insert operations. Also, removing records does not waste memory in the record store.	10
`AVLTree::alpha` correctly calculates and returns the load factor of the table.	5
`operator<<` is correctly overloaded to print the hash table slot, key, and value of all non-empty, non-tombstone records in the table	5
Code is well organized, well documented, and properly formatted. Variable names are clear, and readable. Classes are declared and implemented in seperate (.cpp and .h) files.	5
Appropriate use of public and private class member data. No global variables or unnecessary member variables. Efficient and well-designed code. No memory leaks when creating and deleting hash tables.	5