Hash Table with Open Addressing

Due: 11:59:59 PM, 2/14/25, Friday

Objective

In this assignment, you will implement a hash table using open addressing in C++. You will design your hash function and determine an appropriate table size. The goal is to achieve constant-time complexity O(1) for insert, find, and delete operations.

Problem Statement

You are required to implement a hash table that supports the following operations efficiently:

- `insert(entry)`: Inserts an entry into the hash table. To handle collisions, use open addressing with linear probing by storing the entry in the next available slot.
- `find(key)`: Returns the information associated with the given key or indicates that the key is not found.
- `remove(key)`: Deletes the entry from the hash table.

Requirements

1. **Hash Function**
    - You must design your own hash function.
    - You must determine the primary key (value/field) of each entry.
    - The hash function should distribute keys uniformly across the table.
    - Consider using **modulus (%)** to ensure values fit within the table size.
    - Include comments explaining your **hash function and probing technique**.
    - Include counter(s) to keep track of the total number of collisions
2. **Hash Table Size**
    - Choose an appropriate size for the hash table.
    - Consider using a **prime number** for better distribution.
    - Implement **dynamic resizing** (optional but recommended) for extra credit. For example, resize the table when the load factor exceeds a certain threshold.
3. **Operations**
    - Implement the following methods with **O(1)** complexity:
        - `void insert(Entry entry);`
        - `string find(string key);`
        - `void remove(string key);`
    - Implement additional helper functions as needed:

        - `void display:` display the full info of one or all entries stored in the hash table
        - `int size:` returns the number of the entries stored in the hash table
4. **User Interaction**
    - Implement a **menu-based console interface** to allow users to insert/find/remove entries to test your hash table.
    - Supports a batch-processing feature to allow loading/inserting multiple entries from an external file.
5. **Testing**
    - A sample test file with 10k contacts (entries) is provided on Canvas
    - Try inserting some or all elements for testing.
    - Test your code for corner cases such as duplicate keys, table full scenarios, etc.

Starter Code Template

Below is a basic structure to help you get started. Feel free to make modifications if needed.

```
#include <iostream>
#include <vector>
using namespace std;

class HashTable {
private:
    struct Entry {
        string name;
        string phoneNumber;
        string address;
        bool occupied; // To mark the slot empty/deleted [optional]
    };

    vector<Entry> table;
    int size;

    int hashFunction(string key) {
      // TODO: a simple hash function to determine where the entry
      //       is stored inside the hash table based on the chosen key
    }

    int probe(string key, int i) {
       // TODO: it calls hashFunction and finds the (next) available
        //       spot via linear Probing
    }

public:
    HashTable(int tableSize) {
      // TODO: constructor to initialize the hash table to the given
      //    tableSize
    }

    void insert(Entry entry) {
      // TODO: insert the entry into the hash table.
    }

    string find(string key) {
      // TODO: look up the entry by its key and return the full
     // information of the record
    }

    void remove(string key) {
      // TODO: remove the entry by its key
    }
};
```

Notes

- Please keep the **Academic Integrity Policy** in mind---do not show your code to anyone and do not look at anyone else's code for this project. If you need help, please contact the instructor early.
- Include the basic header in your program as below. **Submissions without the Honor Pledge will get a zero.**

```
# Assignment: (Assignment Number):
# Author(s):
# Due Date: (Due Date)
#
# Description: Write a small blurb about what this project's goals are
#           and what tasks it accomplishes
# Comments: (Explain any known issues or questions if any)
# Honor Pledge: I have abided by the Wheaton Honor Code and
#               all work below was performed by (Your Name(s)).
```

- Make sure you either develop with or test with CLion (to be sure it reports no errors or warnings) before you submit the program. Document any known issues in the comment section above.
- First, be sure you review how the hash table works. Then, read the specifications carefully. Start the project early and ask for clarification during class or open hours.
- Feel free to use the code you find in the textbook or reuse the code the instructor posted on Canvas and you did in class. However, be sure to follow the requirements stated above and document the source.
- Make sure to document each method and adhere to the Google C++ Style Guide for documentation. When you write source code, it should be readable and

well-documented. Each method should have a comment section, including what it returns and the types of parameters that are passed.

```
/* Comment for each function should include the following items
 * A brief description of the function.
 * @param: var, datatype, description
 * @return: var, datatupe, description
 * dependencies: other major member functions invoked in this function.
 */
```

Grading

Grades will be given according to the following policy.

Required functions & test functions	**Points**
Correct implementation of the required insert, find, and remove functions.	40
Effective collision resolution (open addressing), hash function, and additional helper functions such as display and size.	20
User-friendly menu-based interface for testing with support for a file-based batch-processing feature	20
Documentation: filename, program header comment, coding style (variable naming, indentation, comment, readability, etc), a hard copy of your code (signature next to the honor pledge) **README**: include the following information ● **Project tile**: simple overview of use/purpose ● **Authors**: list of contributors for the projects ● **Description**: an in-depth paragraph about your project ● **How to Run**: step-by-step bullets to show how to run your projects with sample input(s) and sample output(s). ● **Time Complexity Analysis**: include a paragraph to explain the time complexity of insert, find, and remove functions using the counter(s) ● **Help**: any advice for common issues or problems ● **Version History**: basic git log if applicable Submit one .**zip** or .**tar** file containing all your source code and **README** for this assignment via the submission link on Canvas.	20
Bonus: dynamic resizing of the hash table	5
Total	**100 + 5**