CptS 223 – Advanced Data Structures in C++

Programming Assignment 3: Hash Tables

I. Learner Objectives:

At the conclusion of this programming assignment, participants should be able to:

- Design and implement a hash table with separate chaining and linear probing

II. Prerequisites:

Before starting this programming assignment, participants should be able to:

- Apply and implement class templates
- Design, implement, and test medium programs in C++
- Edit, build, debug, and run programs through a Linux environment
- Describe the operations of a hash table

III. Overview & Requirements:

What is required?

For this project, you will implement functions for TWO hash table implementations. The starting code base includes abstract class called “Hash”, which the two concrete classes “ChainingHash” and “ProbingHash” are derived from. The file Hash.h includes a description of the interface and what you need to implement with it. Please start with the code provided on Canvas and implement the logic of those functions in “ChainingHash” and “ProbingHash”. **Those public interfaces cannot be changed, but you could add more public and private variables or methods as you see fit**.

For both HashTables, you need to implement the rehashing logic. Rehashing is automatically performed at a load factor of 0.75. Rehashing will re-distribute all existing values to the new buckets. During the rehashing, you should find a new prime number as the new num of buckets to make the load factor < 0.75.

(40 pts) *Chaining:* This is a hash table where the data is stored in a vector of lists. Feel free to use the C++ STL `std::vector` and `std::list` classes. You will also need to keep track of the total number of elements stored in the hash for calculating "size."

Constructor (5 pts)
7 functions (5 pts per function)

(40 pts) *Linear Probing:* This is a hash table with a single vector of elements, but it MUST do lazy deletion (a value is marked as deleted but only physically deleted when a new value is inserted at the same place). This HashTable needs to keep track if a bucket is empty, full, or deleted and your code must use that information when probing and rehashing. You'll need to keep track of the total size of the hash table (excluded those marked "deleted" values). This is doing linear probing: probe(i) = i.

Constructor (5 pts)
7 functions (5 pts per function)

(10 pts) Test cases:
The main.cpp already has the following test cases. Don't change the main.cpp. You need to make sure these test cases can run properly.

1. Create a HashTable (initial num of buckets: 101)
2. Insert 100,000 records (rehash will automatically take place according to the load factor)
3. Search for key 97
4. Erase key 97
5. Search for key 10000
6. Clear the hash table

The main.cpp will also print the time taken by operation 2,3,4.
In the project directory, create a file called "output.txt" and put the output of main.cpp to this file when submitting.

(10 points) Other requirements:

Correctly use cmake to build. The given project already has a correct CMakeLists.txt. You can change it if needed.

IV. Submitting Assignments:

1. Option 1: Canvas

Zip all source files (including CMakeLists.txt, etc., if any) and upload it to Canvas.

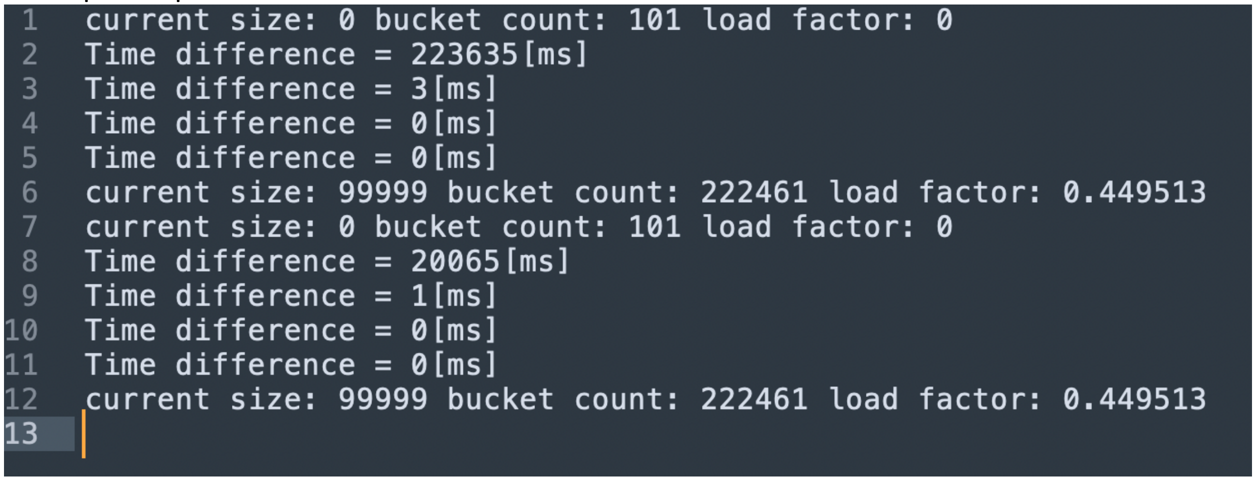
2. Option 2: Github

1) On your local file system, and inside of your git repo for the class, create a new branch called PA3. Available to the branch create a new directory called PA3. Place all PA3 files in the directory. All files for PA3must be added, committed, and pushed to the PA3 branch of your private GitHub repo created in PA1.
2) Submission: You must submit a URL link to the branch of your private GitHub repository. Please make sure the instructor and TAs (GitHub account listed in Syllabus) are the collaborators of your repository. Otherwise, we won't be able to see your repository. DO NOT CREATE NEW REPO.
3) Do not push new commits the branch after you submit your URL to Canvas otherwise it might be considered as late submission.

V. Grading Guidelines:

This assignment is worth 100 points.

* Sample output



```
current size: 0 bucket count: 101 load factor: 0
Time difference = 223635[ms]
Time difference = 3[ms]
Time difference = 0[ms]
Time difference = 0[ms]
current size: 99999 bucket count: 222461 load factor: 0.449513
current size: 0 bucket count: 101 load factor: 0
Time difference = 20065[ms]
Time difference = 1[ms]
Time difference = 0[ms]
Time difference = 0[ms]
current size: 99999 bucket count: 222461 load factor: 0.449513
```