Assignment 7: Hash Tables

Description: In this assignment you will create a Hash Table class/object (as discussed in lecture) with our standard struct (a struct with an integer id and a string for 'data').

Background

- You need to have a complete and working linked list class from Assignment 6.
- Study the notes posted on Blackboard regarding Hash Tables. It is in chapter 18 of your text, and the notes are on Blackboard. Note that only the section on "*Hashing as a Dictionary Implementation*" is relevant, and specifically the part on "*separate chaining*."
- Study the presentation *HashTables_Chaining.pdf* in the Notes section on Blackboard.
- Study the code at https://github.com/alexander-katrompas/hash-table-demo-integers-linear-probing and https://github.com/alexander-katrompas/hash-table-demo-with-linear-probing to help you understand Hash Tables, but keep in mind that code is for linear probing, you will be doing separate chaining in this assignment.

Specifications / Requirements

- Follow the Assignment Specific Instructions using this GitHub assignment invite https://classroom.github.com/a/_KJO2-bU
- You are given all the files you need for the assignment except .gitignore and your own linked list class.
    - Make a proper gitignore first and commit it.
    - Copy your own linked list files into your project directory (linkedlist.cpp, linkedlist.h, data.h).
    - Place all your comment headers and commit all the files.
    - Follow the comments in main() to get started.
- Implement a Separate Chaining (i.e. **not** Linear Probing) hash table class. To do this create a hash table class/object that contains a hash table which is an array of linked list objects (using your linked list class from the previous assignment). Do not modify the working of the linked list as per that assignment except for the printList() method if you need to.
- The OOP relationship in this assignment is a "has a" relationship. The hash table "has a" linked list (in this case, has many linked lists in an array of linked lists).
- Create your hash table array to be size 15 (use a #define in hashtable.h).
- Use modulo as your hash function.
- You are given a main.cpp that will produce test data. Note that the test data will sometimes be more than your table size, sometimes less. Make sure you run your application many times to prove it works in all cases.
- **main( )** is your testing ground for your object. You must **fully** demonstrate all hash table functionality. Look at the testing in the linear probe demo. Your testing should be **far more** exhaustive than that. Insufficient testing will result in a substantially lower grade.
- The public methods you must create are:
    - **bool insertEntry(int, string*);** pass in the id and a string pointer and dynamically allocate inside the object, and return T/F for success or failure. ids must be unique and greater than 0. Strings must be non-empty. Your hash table must account for this and reject bad ids or strings.
    - **string getData(int);** pass in the id and return the data string associated with that id. Return an empty string if the id does not exist.
    - **bool removeEntry(int);** pass in the id and return T/F for success or failure.

- **int getCount();** return the number of total entries currently in the hash table. You should keep a count internal to your object that keeps up with the number of entries (i.e. have a 'count' attribute in your object that stays up to date at all times.
- **void printTable();** traverse the table showing all locations and their contents Format it something like this (you only need to show the ids in the chain but you can show more so long as you clearly show the chain *per* hash table entry):

  Entry 1: EMPTY
  Entry 2: 3 -->14 -->5
  Entry 3: EMPTY
  Entry 4: 10
  Entry 5: 4 --> 2
  etc.

- You must make at least one private method:
  - **int hash(int);** pass in the id and return **id % HASHTABLESIZE**
- You may make more private methods at your discretion.
- All normal architecture guidelines, good programming practice, and submission guidelines apply and will be a *substantial* part of your grade.
- This assignment is essentially the first one where you are "on your own" without substantial starter code. You must show proper architecture and best practices to get a passing grade. You must show proper and ***exhaustive*** testing.

Grading: Your grade will be graded primarily on exactness to detail and specifications, architecture, and coding logic. You will also be graded on your repo and testing with the following guidelines:

- Failure to commit often, small, and smart will result in an **automatic -10% penalty** regardless of your code quality.
- Any stray files in your repo will result in an **automatic -10% penalty** regardless of your code quality.
- If you do not have proper comment headers and/or do not use the Write Submission feature (not the comment section), and/or do not submit your link correctly, it's an **automatic -10%** penalty regardless of your code quality.
- Failure to use the correct branch in your repo (main) will result in an **automatic -5% penalty** regardless of your code quality.
- Failure to test your code **thoroughly and exhaustively** will result in a maximum grade of 70% **regardless** of code quality.
- For assignments that do not compile (**for any reason**) or crash (**for any reason**), the highest grade you can achieve is 50%.

Submission: When you are ready for grading, use the write submission feature in Blackboard and submit your repo link (this is the URL in the browser, not your SSH or .git link). If you need to fix/change something after you submit but *before* I grade it, just fix/change it and push again. **Do not re-submit the assignment before getting a grade**. Only re-submit after you get a grade and want a *re*-grading.