Group members:

Erik Williams

epwilliams@csu.fullerton.edu

## Functional Overview

The system is structured around the GlassesDisplay class, which contains all functionality related to glasses management. This class uses seven different hash tables, each with a unique hash function. These hash functions (hashfct1 to hashfct7) are designed to extract specific digits from a barcode, thereby deciding the hash table in which a particular pair of glasses will be stored. This design allows the system to leverage the properties of hash tables for quick data access while distributing the data effectively to minimize collisions and maximize retrieval efficiency.

File Reading and Data Addition: The system can read data from a file containing glasses attributes and barcodes, parsing each entry and adding it to all hash tables. This redundancy across multiple tables ensures that searches can be optimized based on the operational context or query type.

Data Removal and Verification: Glasses can be removed from all tables based on their barcode, demonstrating the system's ability to maintain synchronization across multiple data structures. Additionally, the tables can be verified to ensure that they are synchronized in size.

Optimization and Analysis: The best_hashing function provides a mechanism to evaluate and determine which hash table currently offers the most balanced data distribution, measured by the size difference between the largest and smallest buckets. This function is critical for assessing the effectiveness of the hashing mechanisms and can guide adjustments or optimizations.

## Helper Function

The provided bucket_count function in the GlassesDisplay class is designed to evaluate the distribution of data within a given hash table. This function essentially measures the "balance" of a hash table by calculating the difference between the sizes of the largest and smallest buckets.

```cpp
// Helper function to return the hash value based on the first digit
tabnine: test | explain | document | ask
int GlassesDisplay::bucket_count(const CustomHashTable & table) {
  int max_count = 0;   // Tracks the maximum size of the observed buckets.
  int min_count = INT_MAX;  // Tracks the minimum size of the observed buckets.

  // Determine the number of buckets to iterate over. Assumes the hash table can provide this info.
  size_t num_buckets = std::min(table.bucket_count(), size_t(10));

  for (size_t i = 0; i < num_buckets; ++i) {
    auto first = table.begin(i);
    auto last = table.end(i);
    int count = std::distance(first, last);      You, 2 days ago • all tests pass

    // Update the maximum and minimum with the current bucket size.
    max_count = std::max(max_count, count);
    min_count = std::min(min_count, count);
  }

  // Return the difference between the largest and smallest bucket sizes.
  return max_count - min_count;
}
```

```
Erik_Williams ~/project-3-glasses-display-hashtable-erik-williams$ make
g++ -std=c++17 -Wall GlassesDisplay.cpp main.cpp -o hashing_test
./hashing_test
Successfully opened file in1.txt
Successfully opened file in2.txt
hash function 1 item 1234567: passed, score 1/1
hash function 2 item 1234567: passed, score 1/1
hash function 3 item 1234567: passed, score 1/1
hash function 4 item 1234567: passed, score 1/1
hash function 5 item 1234567: passed, score 1/1
hash function 6 item 1234567: passed, score 1/1
hash function 7 item 1234567: passed, score 1/1
hash function 1 item 6789012: passed, score 1/1
hash function 2 item 6789012: passed, score 1/1
hash function 3 item 6789012: passed, score 1/1
hash function 4 item 6789012: passed, score 1/1
hash function 5 item 6789012: passed, score 1/1
hash function 6 item 6789012: passed, score 1/1
hash function 7 item 6789012: passed, score 1/1
size after adding two bows: passed, score 1/1
size after reading in1.txt: passed, score 1/1
bestHashing() for in1.txt: Best hash table: 2
passed, score 1/1
size after reading in2.txt: passed, score 1/1
bestHashing() for in2.txt: Best hash table: 3
passed, score 1/1
size after removing 8890123: passed, score 1/1
hash function 1: Best hash table: 4
passed, score 1/1
TOTAL SCORE = 21 / 21
```