**CMSC 3621**

**Lab 5: Hashing**

Complete the implementation of a hash table with chaining in the following two methods.

1)

Error\_code Hash\_table::insert(const string &new\_entry)

/\*

Post: If the Hash\_table is full, a code of overflow is returned. If the

table already contains a string that equals new\_entry, a code of

duplicate\_error is returned. Otherwise, the string new\_entry is inserted

into the Hash\_table and a code of success is returned.

\*/

2)

Error\_code Hash\_table::retrieve(const string &target, string &found) const

/\*

Post: If the table contains no string that equals target, a code of

not\_found is returned. Otherwise, the found string is placed into

found. A code of success is then returned.

\*/

Requirements:

1. Your implementation must be based on "Hashing with Chaining" under Assignment 5 in the URL <https://cs2.uco.edu/~gqian/cmsc3621/assignments>. The hash table has been partially implemented in Hash\_table.h and Hash\_table.cpp. Make sure that you read the code and the comments of the Hash\_table class to understand the implementation. To further help the understanding of hashing with chaining, the lecture notes for Hashing is provided. You can focus on slides 38-45 for the topic.
2. You are required to implement the method insert in **Hash\_table.cpp**. Note that hashing with chaining should be used in the implementation. Specifically, whether a collision occurs or not, you should add the string into the linked list at the hash address. A core statement for insertion is provided below:

table[hash(new\_entry)].insert(0, new\_entry);

1. You are required to implement the method retrieve in **Hash\_table.cpp**. Hashing with chaining should be used in the implementation. Specifically, whether a collision occurs or not, you should retrieve the string (target) from the linked list at the hash address if it exists. A core statement for retrieval is provided below:

sequential\_search(table[hash(target)]**,** target**,** position)**;**

Note that the statement above utilizes a function (sequential\_search) to search through the linked list at the hash address. sequential\_search is provided and can be found in Searches.h

1. When you finish the two methods in Hash\_table.cpp, you can use option 1 in the user interface to import the provided data file ("ol200k.txt") and then use option 2 to search for a key. To test your implementation, you are recommended to search for the key **100001**. The key should be found by both hash tables and the time they take should be shorter than that of the binary search.
2. Create a subdirectory named "lab5" under your server account on cs.uco.edu. Upload all your source code files, "makefile", and the data file ("ol200k.txt") to your server account under "lab5". The instructor should be able to use "make" to compile and test-run your program.
3. Use nano or vi to create a file named "answers" under "lab5". Add your answers to the following questions into "answers".

* For the insert method, we use

table[hash(new\_entry)].insert(0, new\_entry);

Why do we insert the new entry at position 0 of the list, not at the end of the list?

* When we search for key 100001, the time for each search methods are displayed. Rank the four methods (sequential, binary, hash table 1 and hash table 2) from the slowest to the fastest.
* Explain the ranking of the search methods based on time complexity and load factor (for hashing).

Submission:

The following files must be available under "lab5" in your server account on cs.uco.edu:

1. The source code files, "makefile", and the data file ("ol200k.txt") for "Hashing with Chaining" with your completed implementation of insert and retrieve
2. The compiled, executable program "hash"
3. The file "answers" with your answers to the questions in step 6)