Kevin Wang

UID: 205209507

**Project 4 Report**

Data Structure :

* ChatTrackerImpl has the given struct Info that takes in a string user and string chat and has an int value for count
* Within the private data:
  + int m\_bucketSize;– used for construction of hash table implementation to set its max bucket size
  + unsigned int keyGen(string& name) const; -- hash function that takes in a string and return the bucket number in which to store data
  + vector < list< Info >> m\_usersWhoLeft;– hash table hashed by chat room name, keeps track of all the users that have left the chat
  + vector < list<Info>> m\_info;– hash table hashed by user’s name and is the current ‘active’ data structure with users and chats
  + vector < list<Info>> m\_chatrooms;– hash table hashed by chat name and each list has all the user’s that have ever joined it
    - purpose of this data structure is for the terminate function to work quicker by finding each user’s keygen bucket number instead of iterating through all the users in m\_info and checking if it is associated with the given chat.

Algorithms & Pseudocode

* void ChatTrackerImpl::join(string user, string chat)
  + takes the user’s name and generate bucket number using keyGen(string & name) to add new info into m\_info
  + within the specified bucket number generated, checks for collision by starting at the end of the list and iterating to the front, if the user and chat information already exists, don’t change anything
    - otherwise, push\_back into m\_info[bucket] the Info
      * note: the last item in the list is the user’s current chat
  + generate the bucket number hashed by the chat’s name = cBucket
  + add to the m\_chatrooms[cBucket] the information for easier lookup of all users in the chat
* int ChatTrackerImpl::contribute(string user)
  + bucket = keyGen(user)
  + within m\_info[bucket] iterate from the end of the list (current chat) and increment user’s count by 1
  + return the count of the user, otherwise return 0
* int ChatTrackerImpl::terminate(string chat)
  + initialize int total;
  + since m\_chatrooms contain all the users that were ever in that specific chat room, go to the cBucket of m\_chatrooms to find all the users, while all of the user’s within a chat have not been looked at:
    - take the user’s name and find the bucket that it exists in m\_info
    - within m\_info[bucket] add to total the amount of contributions the user has to that specific chat
  + since m\_usersWhoLeft is also hashed by chat name, use the same bucket generated at first and add to total for the users that have left
  + return the total
* int ChatTrackerImpl::leave(string user, string chat)
  + initialize int count;
  + keyGen with both user and chat
  + within m\_info[bucket] erase the Info if user and chat matches
    - count = user’s count
    - delete the information
  + add to m\_usersWhoLeft hashed by chat and add Info to the keyGen bucket of chat
  + return -1 if the chat is not associated with user
* int ChatTrackerImpl::leave(string user)
  + Given only the user, go its bucket generated by keyGen(user) and delete the item at the end of the list – that is the user’s current chat, the item before the deleted one will be the user’s new current chat
    - If there are no items left, then the user is not associated with any chat
  + Add to the m\_usersWhoLeft vector by taking the user’s chat name and generate a bucket hashed by chat name
  + Delete the information in m\_info and return the count of the user

Notes:

* Runtime speed for the given commands.txt file was around ~85 ms on my windows laptop(clock speed @ 2.75 Ghz) in visual studios release mode
  + Did not test on g32 fast on my laptop
* Runtime speed on my desktop PC (clock speed @ 3.15 Ghz) visual studios release mode was around ~60ms
* Runtime speed on my desktop PC compiled under g32fast was around ~55ms-60ms
* Runtime speed on macbook pro g32fast was around ~55ms