Hash Table "Database" Implementation
Cpt S 223 Homework Assignment
Washington State University

Assignment Instructions (read all instructions *carefully* before you write any code):

For this assignment, you will implement a simple database app that can store (in memory), lookup, and remove clients. Think of this as a scenario where your company sells products or services and you need to maintain a client list. A "client" in this case is a structure that has a name (string), address (string), and number of purchases (int). The name is the key value and you may assume that no two clients have the exact same name. You must use the standard library unordered_map class for your hash table.

Set up a main function to read commands from stdin:

In this assignment, you will read simple commands from standard input. You will display your name and ID number on a single line to start with and then enter the command processing loop. There will be one command per line, so your main function (pseudo-code) will be something like:

```
cout << "Student Name, ID#\n";
while (true)
{
  string cmd = ReadLine();
  if (cmd.StartsWith("//")) { continue; }
  else if (cmd.StartsWith("cmd1name")) { HandleCmd1(); }
  else if (cmd.StartsWith("cmd2name")) { HandleCmd2(); }
  else if (cmd.StartsWith("quit")) { break; }
}
cout << "Done\n";
```

Most command strings will be much like C function calls in terms of syntax. You must support the following commands:

// : This is not actually a command that requires any action, but you must identify comment lines that start with the double forward slash during command processing. These are meant to allow for description of the commands and should just be ignored.

add_client(client_name,client_address) : Adds a new client to the database, provided that the client doesn't already exist. If a client with the specified name does already exist in the database then display the exact line “client already exists” (without quotes) and do not alter the database/hash table. Otherwise add the client, with number of purchases initialized to 0, to the hash table and display a line of the form:
client (name) added
where the name is the actual name of the client that was just added.

- The client name and address are both strings, separated by a comma.
- These strings may contain any characters other than commas, parentheses, line breaks, and non-displayable characters. This also applies to the client names in all commands that follow.

client_makes_purchase(client_name) : Increases the purchase count for a client that is already in the database/hash table. You must first verify that the client exists in the hash table. If a client with the specified name does NOT exist in the hash table then display the exact line “client not found” (without quotes) and do not alter the hash table. Otherwise update the purchase count for the client and display a line of the form:
client (name) has made X purchases
where the name is the actual client name and X is the number of the purchases they have made, including the one that was just added by this command. Keep the word “purchases” plural even if they've only made 1 purchase.

remove_client(client_name) : Removes the client with the specified name from the database/hash table. If a client with the specified name does not currently exist in the hash table then display the exact line “client not found” (without quotes) and take no further action for this command. Otherwise remove the client from the hash table and display a line of the form:
client (name) was removed
where the name is the actual client name.

get_client_address(client_name) : Gets and displays the address of the specified client. If a client with the specified name does not currently exist in the hash table then display the exact line “client not found” (without quotes) and take no further action for this command. Otherwise display a line for the address of the client. Do not display any information other than the client address on the line in this case.

quit() : Breaks the command processing loop and terminates your program. Display the line “Done” as the very last thing before returning from main.

Notes:

- As mentioned earlier, you must use the `unordered_map` class that is part of the standard library.
- Put comments in your code! Give a brief explanation for each function at a minimum.
- Look at the sample input/output files in the zip to get an idea of what the commands and their corresponding outputs look like. Do this before you start writing code. As usual, develop your own additional test cases on top of these, since they are samples only and not meant to rigorously test your code.
- Make sure your code compiles in g++ and runs correctly on Linux.