Due Date: Dec 2nd 2019 11:50pm (Monday)

User Authentication Controller

In this assignment, you are asked to implement a simple user authentication controller, where you can add/delete users, change passwords of the users, and log users in and out, list all registered and active users. Registered users are the ones added with `adduser` function, and active users are the ones that are logged in. A user must be registered to be added to the active users list with the login function.

You are given a header file (`AuthenticationController.h`) which provides the public interface (core functionalities) of `AuthenticationController` class.

Requirements:

- You are asked to create and implement `AuthenticationController.cpp` file that includes implementations for all public functions provided in the header file. You can add private methods and fields, but don't modify the public functions. You can also create other classes.
- You are asked to create two hash tables. Table1 stores `<username,password>` pairs of the registered users. Table1 will contain both old passwords and new password of the users. Table2 contains usernames of the active (logged in) users.
- You are asked to use open addressing with double hashing in the implementation of hash table. Hash function implementation is provided in the header file.
- If it is not possible to find an empty slot after a number of probes that is equals to the table size for a new entry, do not insert that entry.

- **[Optional Requirement] [+10 points]:** In case of `addUser`, `changePassword` and `login`, if the load factor of the table is greater than `MAX_LOAD_FACTOR`, you are supposed to expand the table. New table size becomes the smallest prime number greater than `2*(tablesize)`. Don't rehash the "DELETED" and "EMPTY" entries.

Sample state of Table 1 (order depends on the hash function, this is just a sample):
"Fatma", password of Fatma
"Alfred", password of Alfred
"Margo" , password of Margo
"Fatma", old password of Fatma
"Sara", password of Sara
"Margo" , old password of Margo
”Fatma", another old password of Fatma

Sample state of Table 2:
Fatma
Margo

Suggestion: Use vectors to implement tables instead of arrays. Here is a sample code.

```
#include <iostream>
#include <vector>

using namespace std;

struct user{
    string username;
    string password;
};

int main(){

    vector<user> table;
    table.resize(10); //it is easier to resize with vectors
    user aUser;
    aUser.username ="Fatma";
    aUser.password = "apassword";

    table[0] = aUser;

    cout<<"username:"<<table[0].username<<endl;
    cout<<"password:"<<table[0].password<<endl;

}
```

HOW TO SUBMIT

You are supposed to submit your source files (.cpp and .h files) as a single zip file via **CANVAS** and **codepost.**

Please use the following file format while naming the zip file: LastNameFirstnameX_Y.zip where LastNameFirstname is your last name with the first letter in capital, followed by your first name with the first letter in capital; the X is the course code; the Y is the assignment #. (ex: SerceFatmaCS300_4.zip)

HOW TO EVALUATE: The following rubric describes how your work will be evaluated.

Correctness (70 points)

- [70] Program is correct in object-oriented design and function; meets specification
- [50] Program output is correct but elements of specification missing, e.g. variable/method declarations.
- [35] Part of the specification has been implemented, e.g. one out of two required
- subprograms.
- [20] Program has elements of correct code but does not assemble/compile.

Performance Analysis and Reporting (20 points)

Readability (10 points)

- [10] Programmer name and assignment present. Sufficient comments to illustrate program logic. Well-chosen identifiers.
- [7] Programmer name present, most sections have comments. Fair choice of identifiers
- [5] Few comments, non-meaningful identifiers
- [0] No programmer name. No comments. Poor identifiers