

**CSci 1113: Introduction to C/C++
Programming for Scientists and Engineers
Homework 08**

Change Log

This is version 1 of this document. Changelog:

- No changes worth noting

Introduction

Due Date: Tuesday November 26th, before 5:00pm. Submit your source code files using the links on the course website.

Purpose: Practice Programming complicated user interfaces (text menu style) and writing more advanced classes.

Reminders:

1. This is **not** a collaborative assignment. You must design, implement, and test the solution to each problem on your own without substantial assistance from anyone other than the course instructor or TAs. As part of this, you may not include solutions or *portions of solutions* obtained from any source other than those provided in class: examples from the textbook, lectures, or code you and your partner write to solve lab problems. Using code obtained in other ways, or letting others view or use your code is considered academic misconduct. For more information, see the collaboration rules file on the class website, or ask the instructor or TAs if you have any questions on what is and is not allowed.
2. Homework is submitted and tested electronically. Therefore the following are important:
 - Follow the naming conventions mentioned at the end of this file.
 - Submit the correct file or files through the correct link on the course website.
 - Submit the file or files by the due date and time.
 - Follow exactly any example input and output formats given in the problem description.
 - Include any functions explicitly required by the problem description.
 - Regardless of where you develop your solution, your program must compile and run on the cselabs Linux machines.
3. Please make your source code easy to read. In particular, start each file with comments giving the program name, your name, and the date you completed the program. And use good coding style (as discussed in lecture and the textbook) to ensure that the graders can understand your code. It is also good practice to include a description of the program at the top of the file. This will help you understand the program if you come back to it later while studying for a test.

4. Include an explanatory comment before each function's declaration explaining its purpose and behavior. While good function and parameter names can often reduce the need from this, most programmers expect a short, but natural language, explanation of why a given function exists, what it accomplishes, etc.
5. Have fun! Solving computational problems and writing code can be difficult and, at times, frustrating. But it can also be fun and satisfying.

This is a multi-file assignment This homework involves three source files, (header and code for problem A, and code for problem B) While these are split into "problems" they are not separate, and all contribute to one whole "program". Like the last homework you should plan on programming this with correct file names, and testing/compiling your code on the command line terminal. Likewise you will be expected to submit a zip file for this program. If you struggled to create a zip file on the last homework plan time into your schedule to do so on this homework. As before we will happily help you create a zip file in an office hours setting, additionally you should know that searching "how to make a zip in windows" or "how to make a zip in linux" or "how to make a zip on mac" will provide you with several useful guides that can be followed.

The complete expected file list (and expected file names) will be provided at the end of this document.

Problem overview

Unlike previous assignments, this assignment focuses on one program. This section outlines the overall problem itself, we will then explain the parts in more detail in following sections.

User accounts - a common problem

Early computer systems were designed with a mind set of serving a single user for a single task. These programs would assume that the person at the keyboard is an authorized user, and without question assist that user in the program's one and only task. This is the mind-set we have taken in all our programs so far. Modern programs, however, often are developed differently. Modern programs are often designed to accomplish many related tasks, and modern programs are often designed to assist many possible users.

As we switched from single-user programs to multiple-user programs we needed to solve two problems. First we needed some way for the users to identify themselves and have different access to different personal data. The second problem was that the system now needed to protect possibly private user data. A system that provides unlimited access to any user's data should not be trusted with any private data at all.

These two problems combined to the familiar username and password system we use today. Usernames allow a person to identify themselves within a given system, both to other users of the system, and also to the system itself. Passwords (short strings that are not to be shared with other people) can be used to prove an identity. This way only a human who knows the correct username and password would be given access to private user data.

In this homework we will be writing USMASY, a simple simulated user account system for a multi-user program.

USer MAnagement SYstem (USMASY)

The USer MAnagement SYstem (referred to by its acronym USMASY (pronounced “us-mass-sea”)) is a simplified multi-user program, designed to be part of a larger programming system. USMASY plays the vital role of managing user account data, both usernames and passwords. USMASY will present itself as a menu-based text user interface. The user will be able to enter several commands to USMASY:

- help - The help command should show a brief list of the available commands to the user.
- load - The load command should prompt the user for a USMASY data file to load into memory. Each USMASY data file contains a list of users and their private passwords.
- login - The login command should allow a user to try to login, they should first type their username, then they should get three tries at entering a password. If they can enter a correct password in that much time the system should print “login successful” otherwise it should print “login failed”. In a real system, after a successful login the user should be provided a new user-specific menu. We will not be supporting that behavior in this version of USMASY.
- adduser - the add user command should allow a new user to be created by getting a username and a password. If the username is already in use the add-user should fail. Otherwise it should take a password and add a new user.
- save - the save command should take a filename and write the accounts file out of memory and back to disk. This would need to be done after an added user or else the changes would not be expected to save.
- exit - the exit command exits the program.

A Note on Computer Security

It should be noted that Computer Security is a massively complicated field of study. While we will be exploring some of the highest level ideas of this field in this homework with our program, you should be aware that the solutions we will use for this problem are **not security best standards**. In an actual computer system where the privacy or security of real people is on the line the code you will write here would not be considered secure. A secure user password management system:

- would not compare two literal strings (this type of password checking is prone to “side channel attacks” in which the computer can be observed to learn about a user’s password.)
- string passwords should not be literally stored on disk (this type of storage is susceptible to many forms of hacking, once hacked the passwords can be taken and used on other systems. While an ideal user would not reuse passwords, most due and even a low-security system being compromised can rapidly lead to much more dangerous or problematic violations)
- string passwords should not be stored in memory (see above, it’s harder to get passwords out of computer memory, but it’s quite possible.)

Any system that has usernames and passwords owes it to the system’s users to maintain best practice standards for password safety and security. What we are programming quite simply, is not that. Do not store any real or meaningful username or password information into this program, it’s not safe. Do not use this as the basis of any login system that you would need to program in the future. it’s not safe.

Problem A: SystemUser class (35 pts)

For this problem you should start by writing a SystemUser class to represent one user to the USMASY system. We've designed this class with a few principals in mind, most importantly: there should be no way to access a password and updating the password should only be possible with the password. These rules are slightly broken by the streaming operators << and >>, but should otherwise be maintained in all methods and friend functions.

The SystemUser class should have the following private member variables.

- username: the name of the user
- password: the user's password.

The class should have the following public member functions:

- A default constructor that sets the username and password to "empty" (the literal 5 letter word "empty" not an empty string)
- A two argument constructor that takes an initial username and password.
- A getUsername method that allows access to the username.
- A setUsername method that allows modification of the username.
- A checkPassword method that takes an input password and checks if it is correct or not, returning a boolean.
- An updatePassword method that takes as input the old password (as entered by the user) and the new password. This should update the password member variable to the new password, but only if the old password was entered correctly. This should return a boolean indicating true (the password was updated) or false (the password was not updated)
- Operator overloads for operators == and !=. Equality between two accounts should be based only on the username, not the password. (so two SystemUser objects that share a username would be considered equal even if they have different passwords). You can do this either as a function, a friend function, or as a method, this is your choice.
- Operator overloads for the streaming operators << and >>. These should follow best standards for these functions as discussed in class, and should work with cin/cout or file input/output streams.

There is no main method for part A. You should, however, independently create your own tests for these methods. Going forward onto part B without confidence in the correctness of your part A is foolishness that will cost you precious time in this assignment.

Problem B: USMASY main method (65 pts)

Part B asks you to write the USMASY main method. The general outline of this is given above, with examples being shown below to help explain. Here we will give some general hints/requirements, and then specific details about each command.

Requirements

- You should use an array of type `SystemUser` in the main method to store user data read in from the USMASY file.
- Your program repeatedly prompt users for a command until they enter “exit”
- If an invalid command is given it should print an appropriate error (See examples)
- The load command should load data from a USMASY file into the `SystemUser` array, all other commands should simply use this array.
- While you can assume that the USMASY file is less than 10,000 users long, you cannot assume any specific size for the file, and therefore should be tracking the current size of the system user database alongside the array you store the system users in.

HINTS

- Start by writing the command loop and getting that correct, then focus on saving and loading.
- For each command write a separate function that handles the command. Pay attention to what inputs this function might need, and what outputs it might provide.
- Make heavy use of the class and methods from part A. Always be checking to see if the `SystemUser` class has a way to accomplish the goal you currently have. Don’t forget to consider operator overloads as well.
- This program is probably one of the longest ones we’ve written, but it shares many parts in common with problem’s we’ve already tackled, and it has lots of “filler”. **Make Sure to start early!** so that you have a little extra time for the extra typing that will be involved.

help command

The help command should simply print information about USMASY and return to the main loop. Format for this can be found in the examples.

load command

Requirements

- The load command should fail with an error if user data has already been loaded, either by adding users or the load command. If you are tracking the current number of users this is easy to accomplish.
- The load command should start by prompting the user for a USMASY file name.
 - We will store user accounts in a text file with usernames like “accounts.usmasy” your computer will not recognize this file name, but you should still be able to load this file in any text editor as it will be text formatted.
 - The user account file will have one user per line, listing their username and then their password.
 - A single test account file has been provided, it should initially have 2000 users.
- This command should use the overloaded `>>` operator that you wrote in part A to read from the file. This loop can be done in three lines of code (five counting braces).

save command

Requirements

- The save command should start by prompting the user for a USMASY file name.
 - We will store user accounts in a text file with usernames like “accounts.usmasy” your computer will not recognize this file name, but you should still be able to load this file in any text editor as it will be text formatted.
 - The user account file will have one user per line, listing their username and then their password.
- This command should write the currently loaded user account array to a file as a valid usmasy file. You can test if it’s a valid file by restarting the program and trying to load it.
- This command should use the overloaded << operator that you wrote in part A to read from the file. This loop can be done in three lines of code (five counting braces).

login command

Requirements

- This function should start by prompting for a username.
- If an account with the username cannot be found output an error and return to the main loop
- If an account with the username can be found it should prompt the user for a password.
- If the password is correct, it should print “login successful” and control should return to the main loop.
- If the password is incorrect, it should print “ERROR: incorrect password”, and the user should be allowed to try again.
- After 3 failed login attempts, an error should be printed and control should return to the main loop.

add user command

Requirements

- The add user command should prompt for a username and password
- If the username is already being used by a previous user, the add should fail, an appropriate message should be printed, and control should return to the main loop.
- If no other user is using the username, the user should be added and control should return to the main loop.

HINTS

- use the overloaded >> operator to read the user input for username and password

- use the overloaded == or != operators to check for duplicate usernames.
- While it's possible to do this task without using these operators/overloads, it's a bit of a menace, and I don't recommend it.
- Make sure to adjust the count of users as necessary.

exit command

Requirements This should print “goodbye!” before the program exits.

Examples

The following examples are to help illustrate the expected formatting of this program, and the expected behavior.

Example 1

This example does not load or save, simply making users as needed for the program.

```
Welcome to USMASY the user management system
enter command: hel
ERROR: unknown command: hel type help for a command list
enter command: help
Welcome to USMASY the user management system
The USMASY commands are:
* help
* load
* login
* addUser
* save
* exit
enter command: login
username: doug
ERROR: unable to find user data
enter command: login
username: empty
ERROR: unable to find user data
enter command: addUser
username and password: doug |)OuG
enter command: login
username: doug
password: doug
ERROR: incorrect password
password: douggie
ERROR: incorrect password
password: passowrd
ERROR: incorrect password
```

```
ERROR: login failed after 3 incorrect attempts
enter command: login
username: doug
password: |)OuG
login successful
enter command: save
enter a USMASY file name: test1.usmasy
enter command: exit
Goodbye!
```

After this example, the file “test1.usmasy” should have the following contents:

```
doug |)OuG
```

Example 2

This example starts after Example 1 and uses the test1.usmasy file.

```
Welcome to USMASY the user management system
enter command: load
enter a USMASY file name: test1.usmasy
enter command: login
username: doug
password: |)OuG
login successful
enter command: addUser
username and password: doug douggie1
ERROR: username in use
enter command: addUser
username and password: douggie douggie1
enter command: login
username: douggie
password: |)OuG
ERROR: incorrect password
password: douggie1
login successful
enter command: save
enter a USMASY file name: test2.usmasy
enter command: exit
Goodbye!
```

After this example, the file “test2.usmasy” should have the following contents:

```
doug |)OuG
douggie douggie#1
```

Example 3

In this example our user loads the wrong file and finds they cannot load after having already loaded

```
Welcome to USMASY the user management system
enter command: load
enter a USMASY file name: test1.usmasy
enter command: load
ERROR: user data already loaded.
enter command: exit
Goodbye!
```

Example 4

In this example the user tries to load the wrong file name, then adds a user and finds they cannot load a file after adding a user.

```
Welcome to USMASY the user management system
enter command: load
enter a USMASY file name: wrongFile.usmasy
unable to open wrongFile.usmasy
enter command: addUser
username and password: doug doug
enter command: load
ERROR: user data already loaded.
enter command: accounts.usmasy
ERROR: unknown command: accounts.usmasy type help for a command list
enter command: exit
Goodbye!
```

Compilation instructions

This is a multiple-file project, as such you should expect to be compiling the program manually, on the command line.

For this project, if you follow the file name guidelines below, you should be able to use the following command (on linux machines) (Note, you will need to use the cd command to move to where the files are stored first.)

```
g++ HW8.cpp SystemUser.hpp SystemUser.cpp -o HW8
```

You could then type

```
./HW8
```

followed by enter, to run your program (Assuming no compilation errors).

Submission Preparation

For this homework assignment, you should submit three files, named as follows, using the Homework 8 Submission link on the class website.

- HW8.cpp - contains the usmasy program for part B.
- SystemUser.hpp - contains declarations for the SystemUser class.
- SystemUser.cpp - contains definitions for the SystemUser class.

CHANGE IN TYPICAL SUBMISSION POLICIES: Since there are now quite a few more files than on previous homeworks, and the names of these files are quite strictly unchangeable, we are asking you to submit a single zip file containing all of the files above. This will make it easier to keep files separate and also to avoid issues that could arise due to renaming files.

Name your zip file: <YourName>_8.zip (as usual replace your name with your email name, so mine would be kluve018_8.zip). On the linux machines you can make this zip file with the following command:

```
zip <YourName_8.zip> HW8.cpp SystemUser.hpp SystemUser.cpp
```

You can also create a zip file like this through most computer interfaces, for instructions on making zip simply google search a term such as “how to make a zip file on windows” or “how to make a zip file on mac”. Once you have created the zip file you will only need to submit that zip file. (Although I would recommend checking that the zip file has the correct contents.