Programming Assignment #4

Password Manager

CS 2308.001 and 002 Fall 2016
Instructor: Jill Seaman

Due: Tuesday, 11/1/2016: upload electronic copy by 11:55pm.

Problem: Write a C++ program that will simulate a Change Password Utility.

Your program will contain:

- One class, PasswordManager, that will manage a single password.
- A main function that will allow the user to change their password.

Your program should consist of the following files:

PasswordManager.h
PasswordManager.cpp
PasswordDriver.cpp (containing the main function)

You should also have a **makefile** that can be used to build the executable program.

PasswordManager Class:

The PasswordManager class should have just one member variable, which will store the encrypted password (a string). Do **not** store the password unencrypted!

The PasswordManager class should have the following two internal member functions (not accessible outside of the class):

encrypt: this takes a password (a string) and returns the encrypted form of the password. Note: there is no decrypt function (there is no need to decrypt passwords). We will use the following VERY simple encryption algorithm (a Caesar Cipher):

> For every character in the input string, add 15 to the ascii value of the character. The encrypted character's ascii value must stay in the range of printable, non-whitespace characters: 33 to 126. This can be enforced using this formula:
> `ascii value of encrypted char =`
> `((ascii value of ch – 33) + 15) % 94 + 33`

Store all the resulting chars in a string to be returned as the result of the function.

meetsCriteria: this takes a string (a password) and returns true if it meets the following criteria:

- it is at least 8 characters long
- it contains at least one Uppercase letter and one Number.

Otherwise it returns false.

The PasswordManager should have the following member functions that are accessible from outside of the class (from the driver):

setEncryptedPassword: (a setter function) takes a string (an encrypted password) and stores it in the member variable.

getEncryptedPassword: (a getter function) returns the value of the encrypted password stored in the member variable.

setNewPassword: takes a string (a proposed password). If it meets the criteria in meetsCriteria, it encrypts the password and stores it in the member variable and returns true. Otherwise returns false.

authenticate: takes a string (a password) and returns true if, once encrypted, it matches the encrypted string stored in the the member variable. Else returns false.

Input/Output:

The main function should create an array of 3 instances of the PasswordManager class.

Your main function will use a file "password.txt" to store the **encrypted** passwords in between executions of the program. However, the file may not yet exist the first time the program is executed. When your main function starts, it should try to open the file "password.txt". If the file exists, you should assume it contains 3 encrypted passwords, one per line, and the program should use these to set the encrypted passwords in the password manager array. Otherwise it should set all the passwords in the password manager array to "ABC123***" (this one needs to be encrypted). Note: close the input file here! Do not open the file for output until after you have closed it for input.

Your program should explain the criteria for new passwords (see meetsCriteria above) and then ask the user to enter their netID, old password, and new password:

```
Please enter your netID: 1
Please enter your old password: ABC123***
Please enter your new password: AA44bb55
```

For simplicity, the allowed netIDs will be 0, 1, and 2, and we will use these as indexes into the password manager array.

Your program should give one of the following responses:

```
NetID is invalid, password not changed.
Old password is incorrect.
New Password does not meet criteria.
Password has been changed for netID: 1
```

If any of the input data is invalid or causes an error message, do NOT ask the user to re-enter the data.

After outputting one of the above responses to the screen, the program should output the three **encrypted** passwords to the file "password.txt", one per line (overwriting anything that was previously in the file), then exit.

NOTES:

- This program must be done in a **Linux or Unix** environment, using a command line compiler like g++. Do not use codeblocks, eclipse, or Xcode!
- A password does not contain any whitespace. When a password is being entered by the user, assume a whitespace character indicates the end of the password.
- Do NOT change the names of the functions! Use the exact same function names, and do not change the case (uppercase/lowercase). DO NOT change the input order of the three values.
- Create and use a **makefile** to compile the executable program. Modify the one from the TimeDemo (on the class website).
- Put the Class declaration in the header file, the implementation of the class member functions in PasswordManager.cpp and the main function in PasswordDriver.cpp. Put a header comment at the top of each file.
- ALL of the input and output must be done by the driver. **The password manager class should not do ANY input/output**, not to the screen OR the file!
- constructor functions are NOT required, but allowed.
- Your program **must compile** and run, otherwise you will receive a score of 0.
- Your program must pass **Test Case 0** or you will receive a score of 30 or less with no credit for the other grading categories (correctness/constraints/style). The input values and expected output are in a file called **TC0.txt** on the class website. Your program does not need to input from or output to a file to pass TC0. But your program must contain and use a PasswordManager class in order to pass TC0.

Logistics:

Since there are multiple files for this assignment, you need to combine them into one file before submitting them. You should use the zip utility from the Linux/Unix command line:

```
[...]$zip assign4_xxxxx.zip PasswordDriver.cpp
PasswordManager.cpp PasswordManager.h makefile
```

This combines the 4 files into one zip file, **assign4_xxxxx.zip** (where xxxxx is your NetID). Then you should submit only assign4_xxxxx.zip.

There are two steps to the turn-in process:

1. Submit an electronic copy using the Assignments tool on the TRACS website for this class.

2. Submit a printout of the source files only (not the makefile) at the beginning of class, the day after the assignment is due. Please **print your name on top of the front page**, and staple if there is more than one page.

Note: Each member of a group must submit their own electronic copy and their own printout!! Make sure your name is written or circled on your printout.

See the assignment turn-in policy on the course website (cs.txstate.edu/~js236/cs2308) for more details.