

## Programming Assignment #4

### Password Manager

CS 2308.251 and 252, Spring 2015

Instructor: Jill Seaman

**Due:** before class **Wednesday, 3/25/2015** (upload electronic copy by 9:30am)

---

#### Problem:

Write a C++ program that will manage passwords.

Your program will contain:

- A Class, PasswordManager, that will manage a single password.
- A main function, that will allow the user to test the PasswordManager class.

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:

    The XOR operator (^) in C++ takes two char arguments and returns a char.  
    For every character in the input string, apply the XOR operator ^ with the character '5'. For example: `str[i] ^ '5';`

Store all the resulting chars in a string to be returned as the result of the function (do not change the argument that is passed in). This is a **const** function.

**verifyPassword:** 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
- it contains at least one digit
- it contains at least one of these four characters: ! @ # \$

Otherwise it returns false. This is a **const** function.

The PasswordManager should have the following member functions that are accessible outside of the class:

**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. This is a **const** function.

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

**validatePassword:** 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. This is a **const** function.

### **Input/Output:**

The main function should create and use one instance of the PasswordManager class. It is called "the password manager" below.

Your main function will use a file "password.txt" to store the encrypted password in between executions of the program. However, the file may not yet exist the first time the program is executed. So when your main function starts, it should first try to input an encrypted password from the file "password.txt". If the file exists and contains a string, the program should set the encrypted password in the password manager. Otherwise it should set the password in the password manager to "ABC123\$\$\$".

Your program will use the following menu to prompt the user to test the implementation:

```
Password Utilities:  
A. Change Password  
B. Validate Password  
C. Quit  
Enter your choice:
```

The menu should be processed in a loop, so the user may continue testing the password operations.

The Change Password option should ask the user to enter a new password, and explain the criteria for a valid password. The main function should call the password manager to verify and change the password. It should output a message indicating whether or not the password was changed. If it was not changed, it should NOT repeat and ask the user to try again.

The Validate Password option should ask the user to input the password. Then the main function should call the password manager to validate the password, and then the main function should output whether or not the password was valid (matching the one stored by the password manager) or not. If it was not valid, it should NOT repeat and ask the user to try again.

When the user selects C to quit, the program should save the encrypted password in the file "password.txt" (overwriting anything that was previously in the file).

---

## NOTES:

- This program DOES need to be done in a Linux/Unix environment.
- 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 menu choice letters.
- Be sure to declare the non-mutator functions as const functions (encrypt, verifyPassword, getEncryptedPasword, and validatePassword).
- Create and use a **makefile** to compile the executable program. Modify the one from the lecture. I recommend calling the executable file "password".
- 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.

## 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 (the zip file) using the Assignments tool on the TRACS website for this class.
2. Submit a printout of the source files (not the makefile) at the beginning of class, the day the assignment is due. Please print your name on the front page, staple if there is more than one page.

If you are unable to turn a printout in during class, you have until 5pm on the day the assignment is due to turn it in to the computer science department office (Comal 211). They will stamp it and put it in my mailbox. DO NOT slide it under my office door.