# High Level Requirements

**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 '2'. For example: str[i] ^ '2'; Store all the resulting chars in a string to be returned as the result of the function (do not set the member variable, do not change the argument that is passed in).

**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 letter • it contains at least one digit • it contains at least one of these four characters: , ?, ! Otherwise it returns false. 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.

**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

# Low Level Requirments.

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

2 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