Data Structures and Algorithms

Classes:

PasswordManager: This class manages the data associated with a user of a specific account or service: a username and a password, and it has member functions that access and mutate the data. The private members are the username and password string variables, and the encrypt() and meetsCriteria() methods that encrypt password inputs using Caesar's Cipher that are checked for validity with meetsCriteria(). The criteria are that the password must contain 8 or more characters, and 1 uppercase & lowercase character, and 1 digit. The public members are the constructor & destructor, the mutator methods setUsername(), setEncryptedPassword(), and setNewPassword(), the accessor methods getUsername(), getEncryptedPassword(), and the helper method authenticate(). The constructor, destructor, setUsername(), setEncryptedPassword(), getUsername(), and getEncryptedPassword() do exactly what their name says: access and set private member variables. The method setNewPassword(string) takes an input string that represents the new password, which is checked to see if it meets the criteria using meetsCriteria(). If the result is positive, the new password is set and encrypted. Lastly, the authenticate(string) method checks whether the input string (representing the current password) from the user matches the password that the class has stored, thus serving as a basic for of authentication.

Private:

- Variables: username, password
- Methods:
  - encrypt(string): encrypts an input password with Caesar's cipher
  - meetsCriteria(string): checks if password has 8 characters, 1 upper & lowercase letter, and 1 digit

Public:

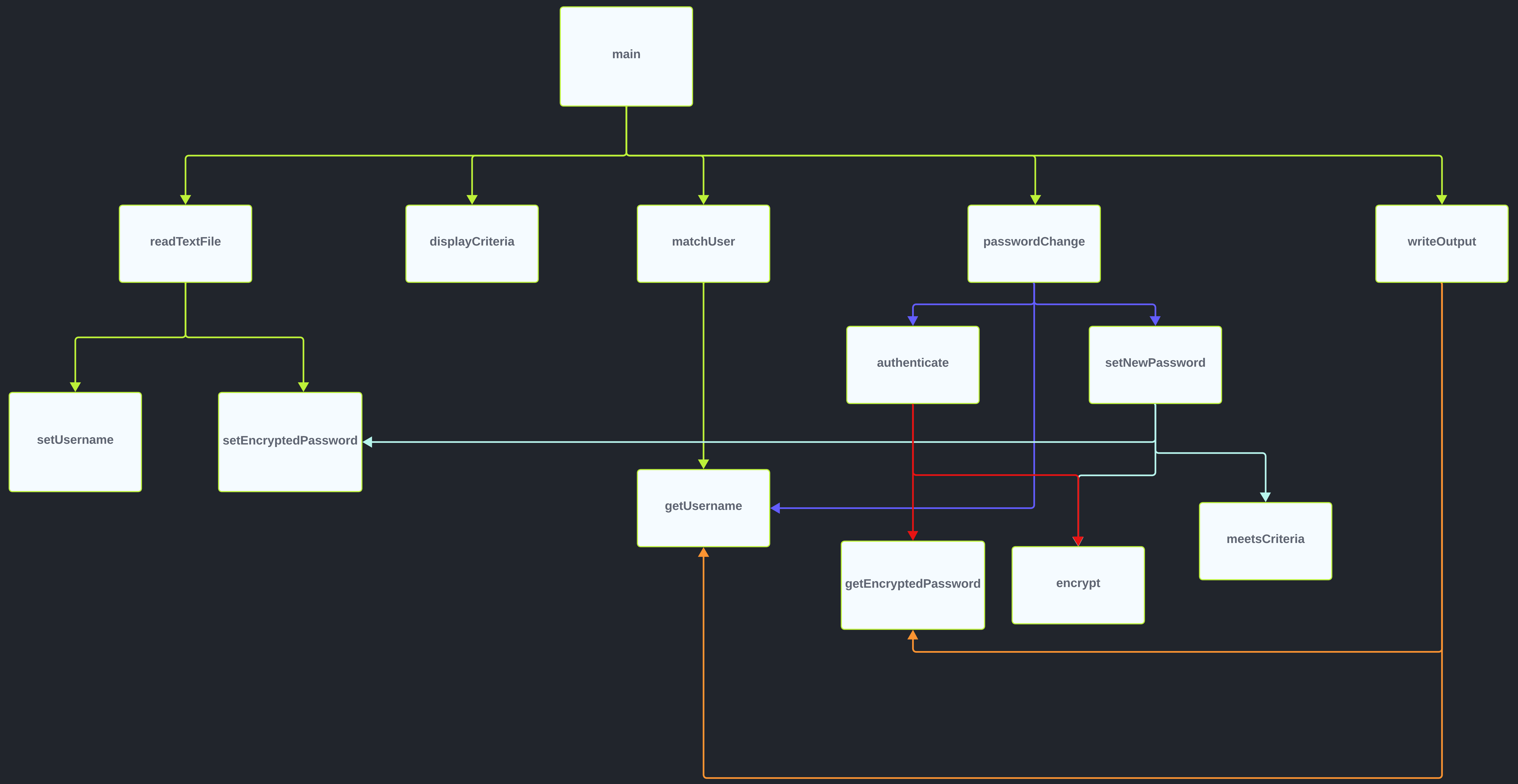
- PasswordManager(): constructor that initializes member variables to empty strings
- ~PasswordManager(): PasswordManager object destroyed
- void setUsername(string): mutator for username
- void setEncryptedPassword(string): mutator for encrypted password
- string getUsername() const: accessor for username
- string getEncryptedPassword() const: accessor for encrypted password
- bool setNewPassword(string): takes a string and attempts setting it as object password
- bool authenticate(string): checks if input string matches encrypted password on file

Design Choices:

The code logic in the driver file uses 5 functions to execute the logic: readTextfile(), displayCriteria(), matchUser(), passwordChange(), and writeOutput(). readTextfile() reads “passwords.txt” and stores the username/password combinations in an array. displayCriteria() requests the user for their credentials and the new updated password to change, and then matchUser() finds a match from the username input with the array from readTextfile(), and returns the index where the match was found. Using this, we are able to call passwordChange(), which checks if the current password was correct and then checks the if the new password meets the criteria before changing the password. Lastly, writeOutput() writes the changes to the “passwords.txt” file. Some changes that could be done would be to include the matchUser() function in displayCriteria() to have less functions. Also, we passed values by pointer in the passwordChange() and matchUser() functions instead of passing by value since it is more efficient (doesn’t create a copy of the argument).

Instructions for Compiling:

Get all the code submitted (PasswordDriver.cpp, PasswordManager.cpp, PasswordManager.h, makefile) in one folder. In your terminal, go to the folder where everything is saved (ex: cd “Project 2”), and then type “make” in your terminal. This will create a link all the files and create a.out, which is the executable file. Then, run “./a.out” in the terminal. This should execute the linked file.


main
readTextFile
displayCriteria
matchUser
passwordChange
writeOutput
authenticate
setNewPassword
setUsername
setEncryptedPassword
getUsername
meetsCriteria
getEncryptedPassword
encrypt