```cpp
/**
 ********************************************************************************
 * @file lab_1.cpp
 * @author Robert Myers Jr.
 * @version V1.0
 * @brief The Main logic of the program. Does the credential check and six digit
 * verification. After verification it prompts a menu for the user to input comm
ands
 * into.
 ********************************************************************************
 */

#include "password_file.hpp"
#include "menu.hpp"
#include "user_input.hpp"
#include <cstdio>
#include <cstdlib>
#include <string>
#include <string_view>
#include <format>

std::string get_username() {
    return get_user_input("Enter username: ");
}

std::string get_password() {
    return get_user_input("Enter password: ");
}

std::string get_six_digit_code(std::string_view code) {
    std::string prompt = std::format("Enter the 6-digit code '{}' : ", code);
    return get_user_input(prompt);
}

std::string check_credentials() {
    PasswordFile passwordFile;

    auto username = get_username();
    auto password = get_password();

    if(!passwordFile.check_for_credentials(username, password)) {
        std::printf("The provided username and/or password is invalid. Please try again\n");
        exit(0);
    }
    return username;
}

unsigned int generate_code() {
    return ((unsigned int)rand()) % 1'000'000;
}

void digit_code() {
    std::string expected_code = "";
    while(expected_code.length() != 6) {
        expected_code = std::string(
                std::to_string(generate_code())
                );
    }

    auto six_digit_code = get_six_digit_code(expected_code);

    if(expected_code != six_digit_code) {
        std::printf("Incorrect code. Please try again\n");
        exit(0);
    }

}
int main() {
    // Seed the random generator at the beginning
    srand(time(0));
    auto username = check_credentials();
```

```
71      digit_code();
72      Menu menu(username);
73      menus current_menu = MAIN_MENU;
74      while(1) {
75          current_menu = menu.display_menu(current_menu);
76      };
77  }
```

```cpp
/**
*******************************************************************************
* @file menu.cpp
* @author Robert Myers Jr.
* @version V1.0
* @brief Source file of the Menu Class. See header for more information
*******************************************************************************
*/

#include "menu.hpp"
#include "password_file.hpp"
#include "user_input.hpp"
#include <cstdlib>
#include <iostream>
#include <string>
#include <string_view>

void Menu::register_newuser_menu() {
    PasswordFile password_file;

    auto users_and_passwords = password_file.get_current_users_and_passwords();

    auto username = get_user_input("Username: ");

    bool user_exists = false;

    for(auto user_credentials : users_and_passwords) {
        if(user_credentials.username_ == username) {
            user_exists = true;
            break;
        }
    }

    if(user_exists) {
        std::cout << "User is already in database. Leaving user registration menu\n";
        return;
    }

    auto password = get_user_input("Password: ");

    if(!password_file.check_if_password_is_valid(password)) {
        return;
    }
    password_file.write_encrypted_username_and_password(username, password);
    std::cout << "User has been registered!\n";
}

void Menu::change_password_menu() {
    PasswordFile password_file;

    auto password = get_user_input("Password: ");

    if(password_file.check_if_password_is_valid(password)) {
        password_file.update_user_and_password(current_user_, password);
        std::cout << "Successfully changed password\n";
    }
}

menus Menu::display_menu(menus menu) {
    switch(menu) {
        case MAIN_MENU:
            return display_main_menu();
        case REGISTER_NEW_USER_MENU:
            register_newuser_menu();
            return MAIN_MENU;
        case CHANGE_PASSWORD_MENU:
            change_password_menu();
            return MAIN_MENU;
        case EXIT_MENU:
            std::cout << "Goodbye!\n";
            exit(0);
```

```
72              default:
73                  return display_main_menu();
74          };
75      }
76
77
78      menus Menu::display_main_menu() {
79          auto input = get_user_input("Type 'newuser' to add a new user, 'changepass' to change your passw
        ord, or 'logout' to exit: ");
80
81          if(input == "newuser") {
82              return REGISTER_NEW_USER_MENU;
83          }
84          if(input == "changepass") {
85              return CHANGE_PASSWORD_MENU;
86          }
87          if(input == "logout") {
88              return EXIT_MENU;
89          }
90          std::cout << "Error: not a valid option please try again\n";
91          return MAIN_MENU;
92      }
93
94      Menu::Menu(std::string_view current_user) : current_user_(current_user) {
95      }
```

```cpp
/**
*******************************************************************************
* @file password_file.cpp
* @author Robert Myers Jr.
* @version V1.0
* @brief Source file of the PasswordFile Class. See header for more information
*******************************************************************************
*/
#include "password_file.hpp"
#include <array>
#include <cctype>
#include <cstddef>
#include <cstdio>
#include <fstream>
#include <iomanip>
#include <ios>
#include <iostream>
#include <string>
#include <openssl/sha.h>
#include <string_view>
#include <sstream>
#include <vector>


constexpr std::string_view USER_CREDENTIAL_FILE = {"user_credentials.txt"};
constexpr size_t SIZE_OF_CREDENTIALS_BUFFER{101};

constexpr size_t MIN_PASSWORD_SIZE = 8;

user_credential_t::user_credential_t(std::string_view username, std::string_view password) : username_{username}, password_{password} {}

void PasswordFile::write_plain_username_and_password(std::string_view username, std::string_view password) {
    std::string user_credentials = std::string(username) + " " + std::string(password);
    auto user_credentials_view = std::string_view(user_credentials);
    write_credentials_to_file(user_credentials_view);
}

void PasswordFile::write_encrypted_username_and_password(std::string_view username, std::string_view password) {
    std::string user_credentials = std::string(username) + " " + get_sha256_string(password);
    write_credentials_to_file(user_credentials);
}

void PasswordFile::write_credentials_to_file(std::string_view user_credentials) {
    std::ofstream test_file(USER_CREDENTIAL_FILE.data(), std::ios_base::app);
    test_file << user_credentials << "\n";
}

std::string PasswordFile::get_sha256_string(std::string_view str) {
    unsigned char hash_digest[SHA256_DIGEST_LENGTH];

    SHA256(reinterpret_cast<const unsigned char*>(str.data()), str.length(), hash_digest);
    std::stringstream ss;

    for(int i = 0; i < SHA256_DIGEST_LENGTH; i++) {
        ss << std::hex << std::setw(2) << std::setfill('0') << static_cast<int>(hash_digest[i]);
    }
    return std::string(ss.str());
}

bool PasswordFile::check_for_credentials(std::string_view user, std::string_view password) {
    std::array<char, SIZE_OF_CREDENTIALS_BUFFER> credentials;
    std::ifstream user_credential_file(USER_CREDENTIAL_FILE.data(), std::ios::in
```

```
    );
63
64      std::string given_credentials = std::string(user) + " " + get_sha256_string(
    password);
65
66      while(user_credential_file.getline(credentials.data(), SIZE_OF_CREDENTIALS_B
    UFFER - 1)) {
67          if(given_credentials == std::string(credentials.data())) {
68              return true;
69          }
70      }
71      return false;
72
73  }
74
75  std::vector<user_credential_t> PasswordFile::get_current_users_and_passwords() {
76      std::ifstream user_credential_file(USER_CREDENTIAL_FILE.data(), std::ios::in
    );
77
78      std::array<char, SIZE_OF_CREDENTIALS_BUFFER> credentials;
79      std::vector<user_credential_t> user_credentials;
80
81      std::string current_user;
82      std::string current_password;
83
84
85      while(user_credential_file.getline(credentials.data(), SIZE_OF_CREDENTIALS_B
    UFFER - 1)) {
86          std::stringstream credential_line(credentials.data());
87          credential_line >> current_user;
88          credential_line >> current_password;
89
90          user_credentials.push_back(user_credential_t(
91                      current_user,
92                      current_password
93                      ));
94      }
95      return user_credentials;
96  }
97
98  void PasswordFile::update_user_and_password(std::string_view username, std::stri
    ng_view password) {
99      std::fstream user_credential_file(USER_CREDENTIAL_FILE.data());
100
101     std::string file_content = "";
102
103     std::array<char, SIZE_OF_CREDENTIALS_BUFFER> credentials;
104     std::vector<user_credential_t> user_credentials;
105
106     std::string given_credentials = std::string(username) + " " + get_sha256_str
    ing(password);
107     auto position = user_credential_file.tellg();
108     while(user_credential_file.getline(credentials.data(), SIZE_OF_CREDENTIALS_B
    UFFER - 1)) {
109         auto current_line = std::string(credentials.data());
110
111         if(current_line.starts_with(username)) {
112             std::string given_credentials = std::string(username) + " " + get_sh
    a256_string(password);
113             user_credential_file.seekp(position);
114             user_credential_file << given_credentials;
115         }
116         position = user_credential_file.tellg();
117     }
118
119 }
120
121 bool PasswordFile::check_if_password_is_valid(std::string_view password) {
122     std::string symbols{"~`! @#$%^&*()_-+={[}]|\\:;\"'<,>.?/"};
123     auto password_length = password.length();
124
```

```
125     bool has_uppercase = false;
126     bool has_lowercase = false;
127     bool has_symbol = false;
128 
129     if(password_length < MIN_PASSWORD_SIZE) {
130         std::cout << "Password needs to have 8 or more characters\n";
131         return false;
132     }
133 
134     for(int i = 0; i < password_length; i++) {
135         if(std::isupper(password[i])){
136             has_uppercase = true;
137         } else if (std::islower(password[i])) {
138             has_lowercase = true;
139         } else if(symbols.find(password[i]) != std::string::npos) {
140             has_symbol = true;
141         }
142     }
143 
144     if(!has_uppercase) {
145         std::cout << "Password needs to have at least one uppercase letter\n";
146     }
147 
148     if(!has_lowercase) {
149         std::cout << "Password needs to have at least lowercase letter\n";
150     }
151 
152     if(!has_symbol) {
153         std::cout << "Password needs to have a symbol\n";
154     }
155 
156     return (has_lowercase && has_uppercase && has_symbol);
157 }
```

```cpp
/**
*****************************************************************************
* @file user_input.cpp
* @author Robert Myers Jr.
* @version V1.0
* @brief Implementation of the get_user_input function to get user input
* verification.
*****************************************************************************
*/
#include "user_input.hpp"

#include <iostream>

std::string get_user_input(std::string_view prompt) {
    std::string response;
    std::cout << prompt;
    std::cin >> response;
    return response;
}
```

```cpp
/**
*****************************************************************************
* @file menu.hpp
* @author Robert Myers Jr.
* @version V1.0
* @brief The Menu Class works similar to a state machine where the main loop simply
* feeds the state back into the display_menu function which determines what state it
* needs to put the menu
*****************************************************************************
*/
#pragma once

#include <string>
#include <string_view>
enum menus {
    MAIN_MENU,
    REGISTER_NEW_USER_MENU,
    CHANGE_PASSWORD_MENU,
    EXIT_MENU
};
class Menu {
    public:
        Menu(std::string_view current_user);
        /**
         * @brief Displays the given menu.
         *
         * @param menu The menu to display
         * @return The menu to display next.
         */
        menus display_menu(menus menu);
    private:
        /**
         * @brief Displays main menu
         */
        menus display_main_menu();
        /**
         * @brief Displays register new user menu
         */
        void register_newuser_menu();
        /**
         * @brief Displays change password menu
         */
        void change_password_menu();
        std::string current_user_;
};
```

```cpp
/**
*****************************************************************************
* @file password_file.hpp
* @author Robert Myers Jr.
* @version V1.0
* @brief Class used to read and write the password file.
*****************************************************************************
*/
#pragma once
#include <vector>
#include <string>
#include <string_view>

/**
 * @class user_credential_t
 * @brief A struct that represents a user and their password read from the text file
 */
struct user_credential_t {
    user_credential_t(std::string_view username, std::string_view password);
    std::string username_;
    std::string password_;
};

class PasswordFile {
    public:
        /**
         * @brief Writes a username and password in plain text to the textfile
         */
        void write_plain_username_and_password(std::string_view username, std::string_view password);
        /**
         * @brief Writes a username and password in plain text to the textfile where the password is
         * encrypted
         */
        void write_encrypted_username_and_password(std::string_view username, std::string_view password);
        /**
         * @brief Check if username and password are in the text file.
         */
        bool check_for_credentials(std::string_view user, std::string_view password);

        /**
         * @brief Check if a password is valid. I.e. upper and lower case, 8+ characters, and has a symbol
         */
        bool check_if_password_is_valid(std::string_view password);

        /**
         * @brief Gets a list of users
         */
        std::vector<user_credential_t> get_current_users_and_passwords();

        /**
         * @brief Update a specific user's password
         */
        void update_user_and_password(std::string_view username, std::string_view password);
    private:
        /**
         * @brief Write credentials to file
         */
        void write_credentials_to_file(std::string_view user_credentials);
        /**
         * @brief Helper method for getting the sha256 of a string
         */
        std::string get_sha256_string(std::string_view str);
};
```

```cpp
/**
*********************************************************************************
* @file user_input.hpp
* @author Robert Myers Jr.
* @version V1.0
* @brief Header file that declares the get_user_input helper function
*********************************************************************************
*/
#pragma once
#include <string>

/**
 * @brief Get input from user
 *
 * @param prompt The prompt to be printed out
 */
std::string get_user_input(std::string_view prompt);
```

```
1  ##############################################################################
2  # @file password_file.cpp
3  # @author Robert Myers Jr.
4  # @version V1.0
5  # @brief CMakeList file to compile Lab 1
6  ##############################################################################
7  cmake_minimum_required(VERSION 3.15)
8
9  project(Lab1 LANGUAGES CXX)
10
11 # Use C++ 20 features
12 set(CMAKE_CXX_STANDARD 20)
13 set(CMAKE_CXX_STANDARD_REQUIRED True)
14
15 # Generates a compile commands file for LSP
16 set(CMAKE_EXPORT_COMPILE_COMMANDS True)
17
18 add_executable(lab1
19     lab_1.cpp
20     menu.cpp
21     password_file.cpp
22     user_input.cpp
23 )
24
25 # Link OpenSSL
26 find_package(OpenSSL REQUIRED)
27 target_link_libraries(lab1 PRIVATE OpenSSL::Crypto)
```