The Hashemite University

Faculty of Engineering

Department of Computer Engineering

Password-Based Authentication System

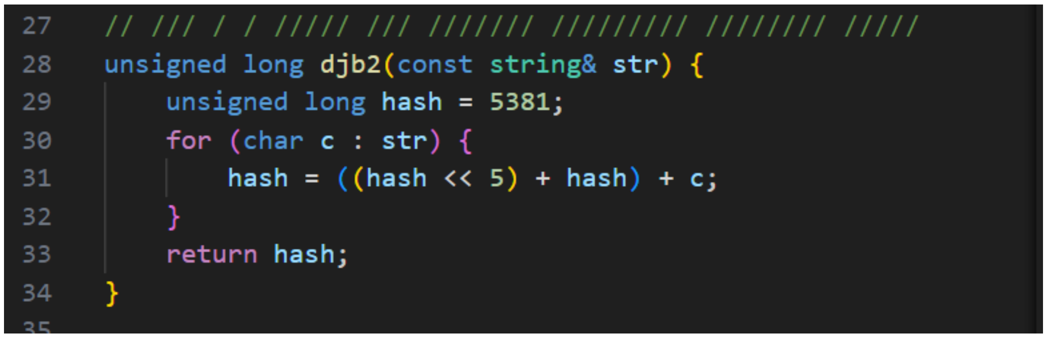
Computer Security

OVERVIEW:

A simple implementation of a password-based authentication systems designed by C++ programming language to handle any request delivered about authentication systems such as making new account, signing in to exist account, and identify the account's user based on the password.

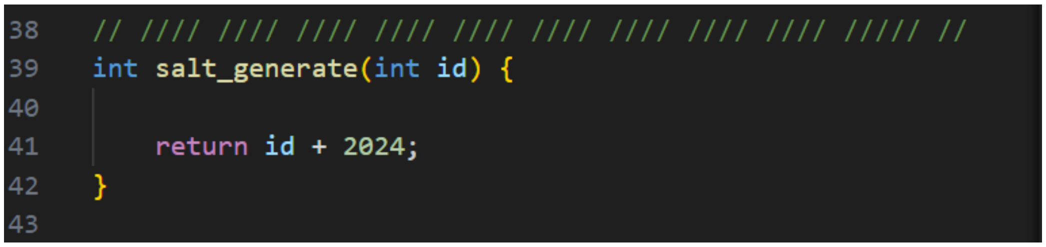
There are a lot of topics and functions that been implemented in this program, it use the idea of multithreaded programming, that makes it be able to handle more than one requests at a time, also it uses semaphores that guarantee the safety of accessing the password file by the processes, what about the password file, it contains three columns, first column implement the usernames of each account, in the second column I added the salts, and in the last column you will find the hashing code of both salt and password, this implementation makes the system solid and much secure, here I will show each part and function of the system's code and it's description:

1) **Hashing function**: this hashing algorithm DJB2 is one of the simplest hashing algorithms exists, for that reason it is widely used, the behavior of that code shown below, is to generate an unsigned long variable and affect it based on the string argument that been passed, so for each character this string contains the long variable will be shifted by five bits, added to it self and added again to that character, that makes the output random and hard to be repeated.



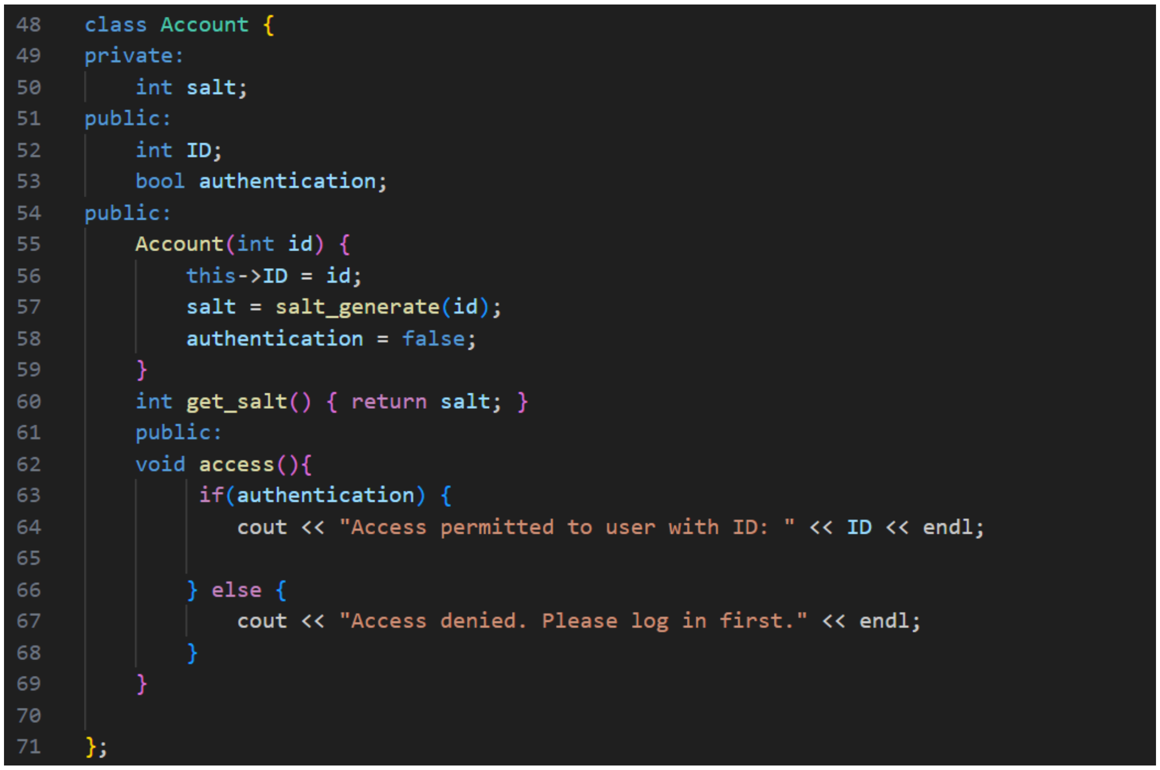
```
// /// / / ///// /// /////// ///////// //////// /////
unsigned long djb2(const string& str) {
    unsigned long hash = 5381;
    for (char c : str) {
        hash = ((hash << 5) + hash) + c;
    }
    return hash;
}
```

2) **Salt-generator**: this function simply takes the ID of the new account generated and return back this ID added with the number of (2024), the reason of adding this number rather that just passing the id it self, is to make the salt generated larger so it can affect the hash code more.



```
// //// //// //// //// //// //// //// //// //// ///// //
int salt_generate(int id) {

    return id + 2024;
}
```

3) **Class Account**: this class is just added to my code to make a small hint of what can be coming after, or how can we use these accounts, so the implementation of it is not as complex as it should be , however it still can gives some functionality of the whole system, the idea if it is to make for each signing up occur a new account object that includes the number of this account (ID) ,some information about the user, the salt this user used to be signed and so on, also there is another function (Access) that can only used if the signing in operation goes right.

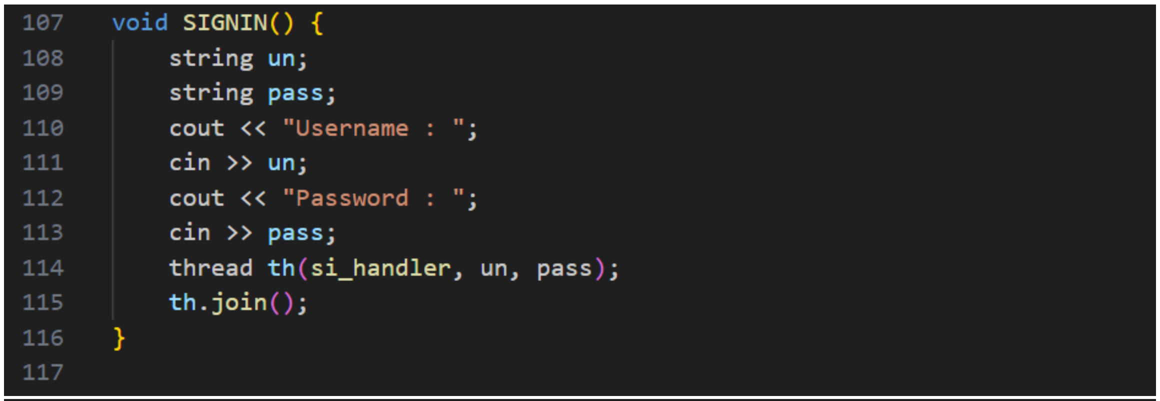


```cpp
class Account {
private:
    int salt;
public:
    int ID;
    bool authentication;
public:
    Account(int id) {
        this->ID = id;
        salt = salt_generate(id);
        authentication = false;
    }
    int get_salt() { return salt; }
    public:
    void access(){
         if(authentication) {
            cout << "Access permitted to user with ID: " << ID << endl;

         } else {
            cout << "Access denied. Please log in first." << endl;
         }
    }

};
```

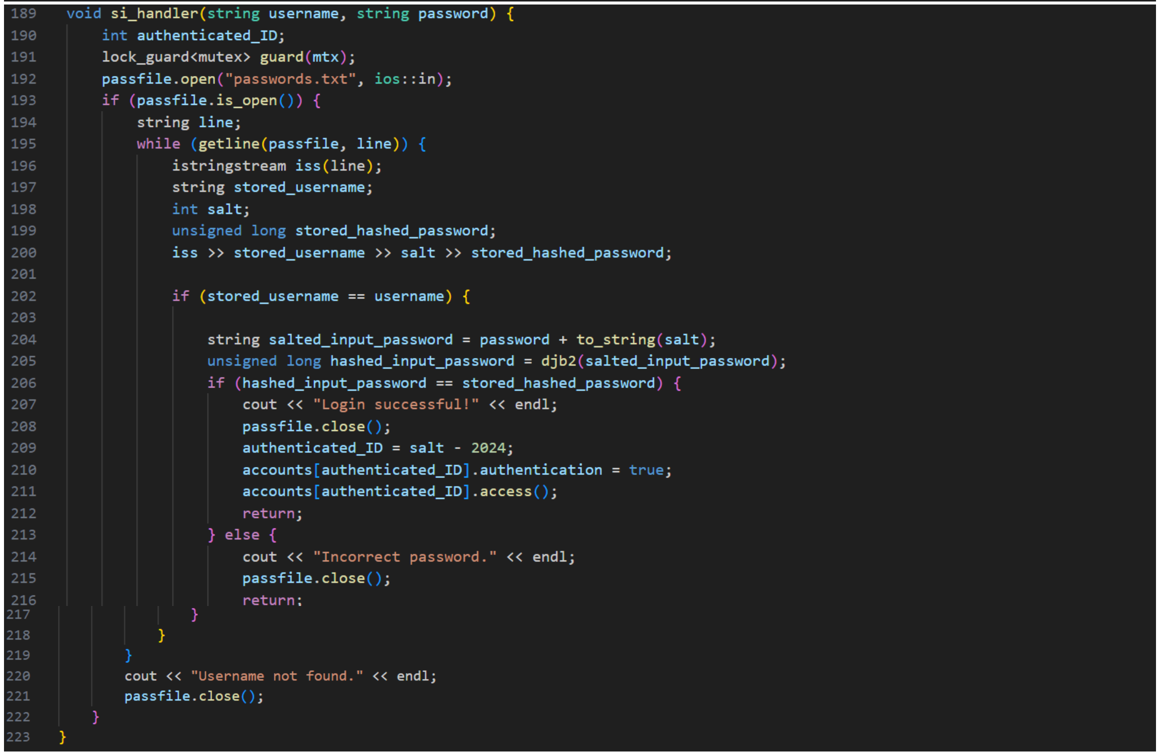
4) **Signing in functions**: there are to functions that will be used to sign in to the account , first function (SIGNIN) is the function that will be accessed by the main process, it only takes the username and the password from the user and then pass it up to a new thread generated to handle the password file accessing using the second function (si_handler), this function working by so many steps:

   A. Defining authentication id that will be searched by for the account that been authenticated if the signing in goes right.
   B. Locking the mutex with guard function that can be unlock the mutex automatically when the function goes out of the scope.
   C. Open the password file and making sure it successfully opened.
   D. Reading from each line all argument using function(iss) from the class (istringstream) and store all these values on the defined variables stored_username, salt and stored_hashing_password.
   E. If the stored username equals to that one passed, then the function will check if the hash code of the combination of salt and stored password equals to that hash code

placed in the file, if it is, the function will print that login successfully and also gives the authentication to that account , if not , then it will print Incorrect password, if the while loop finished without any succeed, the function will print that the username is not found, and closing the file anyway.



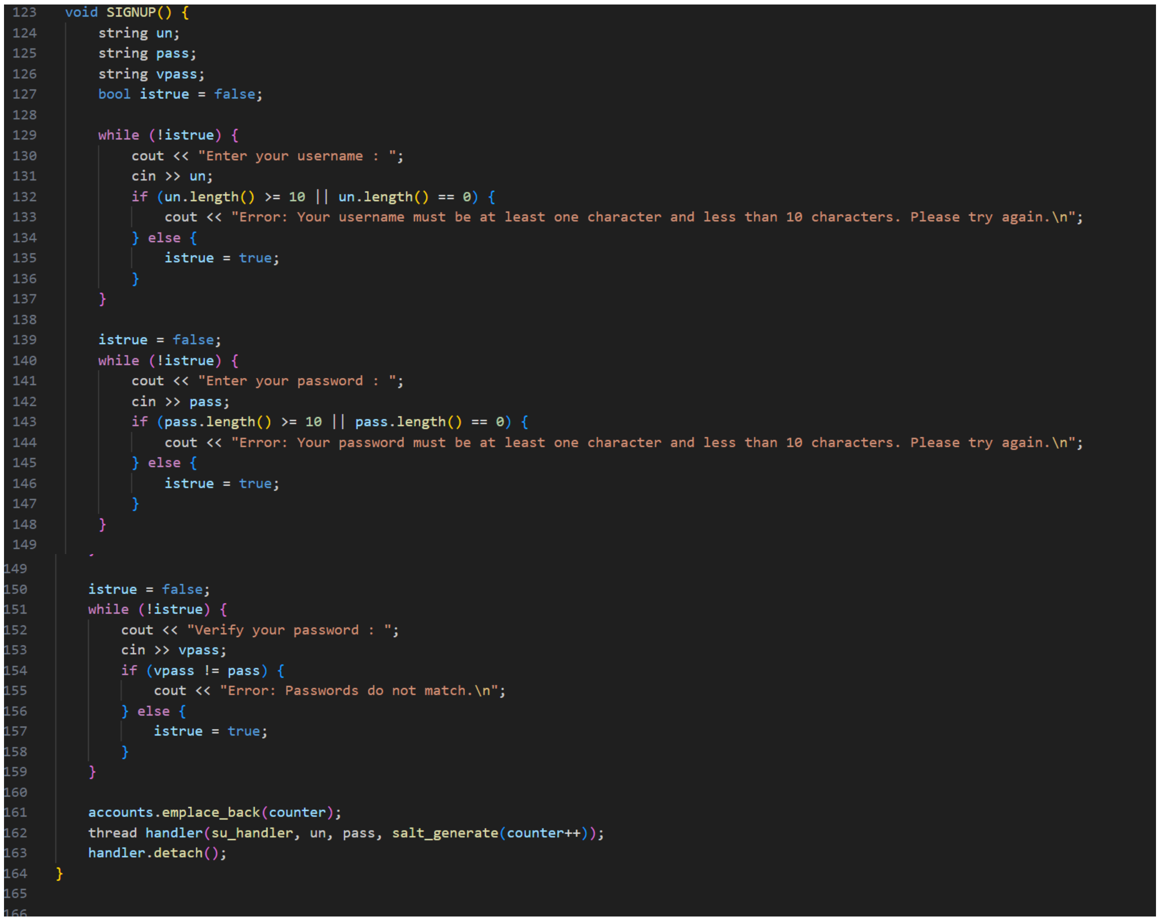
```cpp
void SIGNIN() {
    string un;
    string pass;
    cout << "Username : ";
    cin >> un;
    cout << "Password : ";
    cin >> pass;
    thread th(si_handler, un, pass);
    th.join();
}

```



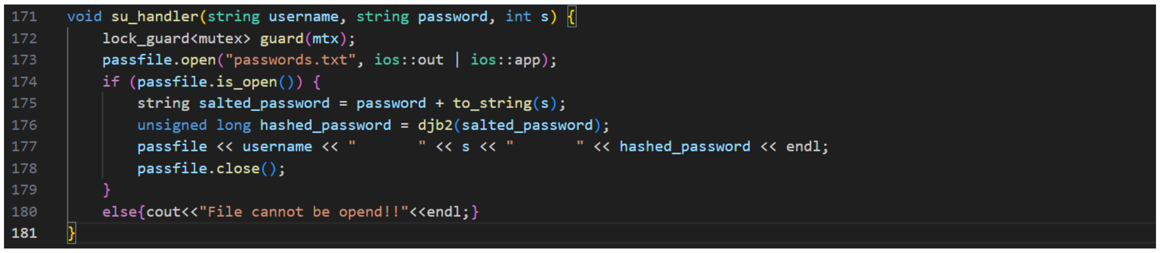
```cpp
void si_handler(string username, string password) {
    int authenticated_ID;
    lock_guard<mutex> guard(mtx);
    passfile.open("passwords.txt", ios::in);
    if (passfile.is_open()) {
        string line;
        while (getline(passfile, line)) {
            istringstream iss(line);
            string stored_username;
            int salt;
            unsigned long stored_hashed_password;
            iss >> stored_username >> salt >> stored_hashed_password;

            if (stored_username == username) {

                string salted_input_password = password + to_string(salt);
                unsigned long hashed_input_password = djb2(salted_input_password);
                if (hashed_input_password == stored_hashed_password) {
                    cout << "Login successful!" << endl;
                    passfile.close();
                    authenticated_ID = salt - 2024;
                    accounts[authenticated_ID].authentication = true;
                    accounts[authenticated_ID].access();
                    return;
                } else {
                    cout << "Incorrect password." << endl;
                    passfile.close();
                    return;
                }
            }
        }
        cout << "Username not found." << endl;
        passfile.close();
    }
}
```

5) **Signing up functions**: also two function first one (SIGNUP), is the function that used by the main process, takes the username and making sure it is valid by checking out either it is nothing or if it is oversized, same thing with password, then it ordered the user to rewrite the password, after all it create a new account that will be added in the dynamic array in its number indexed, that make it easy to search for the account by its id number passed to that array, then the function will create a new thread and detach it to allow the main process to handle another user while this thread write the information on the password file, the

function of the thread (su_handler) at first lock the mutex , then open the file and makes sure it opened successfully, after that writing all the information that been passed by the first function but with the hash code instead of password, then it closes the file.



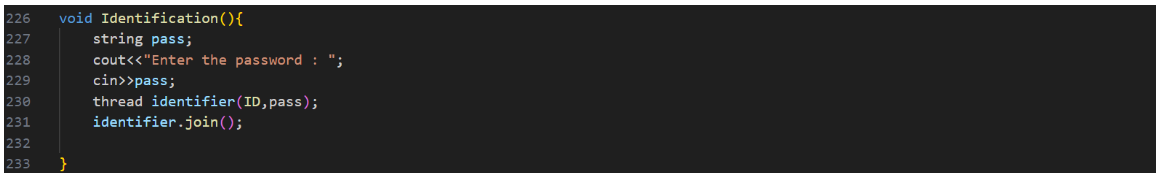
```cpp
void SIGNUP() {
    string un;
    string pass;
    string vpass;
    bool istrue = false;

    while (!istrue) {
        cout << "Enter your username : ";
        cin >> un;
        if (un.length() >= 10 || un.length() == 0) {
            cout << "Error: Your username must be at least one character and less than 10 characters. Please try again.\n";
        } else {
            istrue = true;
        }
    }

    istrue = false;
    while (!istrue) {
        cout << "Enter your password : ";
        cin >> pass;
        if (pass.length() >= 10 || pass.length() == 0) {
            cout << "Error: Your password must be at least one character and less than 10 characters. Please try again.\n";
        } else {
            istrue = true;
        }
    }


    istrue = false;
    while (!istrue) {
        cout << "Verify your password : ";
        cin >> vpass;
        if (vpass != pass) {
            cout << "Error: Passwords do not match.\n";
        } else {
            istrue = true;
        }
    }

    accounts.emplace_back(counter);
    thread handler(su_handler, un, pass, salt_generate(counter++));
    handler.detach();
}

```



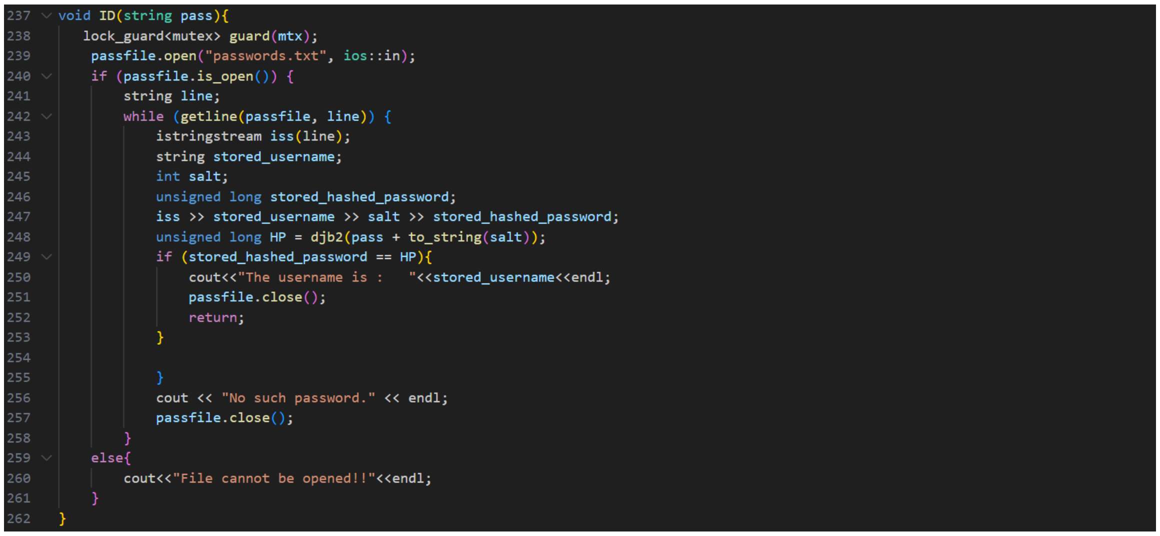
```cpp
void su_handler(string username, string password, int s) {
    lock_guard<mutex> guard(mtx);
    passfile.open("passwords.txt", ios::out | ios::app);
    if (passfile.is_open()) {
        string salted_password = password + to_string(s);
        unsigned long hashed_password = djb2(salted_password);
        passfile << username << "       " << s << "       " << hashed_password << endl;
        passfile.close();
    }
    else{cout<<"File cannot be opend!!"<<endl;}
}
```

6) **Identification functions**: last operation this program consider, first function is the function that been used by the main process (Identification), this function asks the user to enter the password that will be identified, this passing this password to a thread and joining it to keep these processes synchronized, that we use joining function here because we need the main process to wait until the thread finishes and giving the result for the user before any new request, however, this password will go to the second function (ID), this function will lock the mutex, open the file, check the opening, reading all the values that stored in the file line by line and check if the value(hashed password) ,is equal to the hash code of the password that been passed with the salt of the same line combined, if so, the function will print the

username that exist in the line , if the whole iterations failed, the function will print that no such password exit in the file, and closing the file anyway.

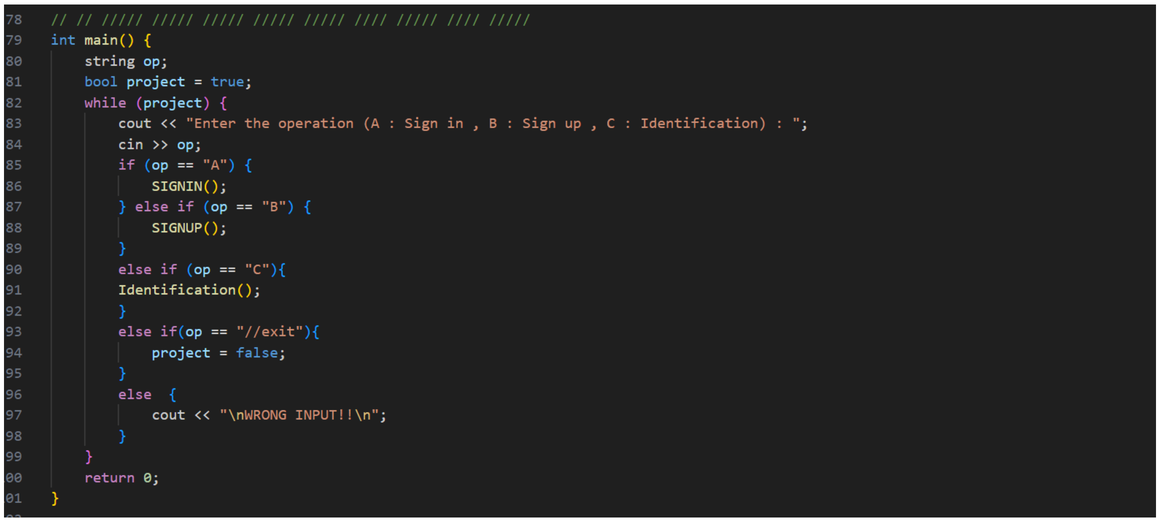


```
void Identification(){
    string pass;
    cout<<"Enter the password : ";
    cin>>pass;
    thread identifier(ID,pass);
    identifier.join();

}
```



```
void ID(string pass){
   lock_guard<mutex> guard(mtx);
    passfile.open("passwords.txt", ios::in);
    if (passfile.is_open()) {
        string line;
        while (getline(passfile, line)) {
            istringstream iss(line);
            string stored_username;
            int salt;
            unsigned long stored_hashed_password;
            iss >> stored_username >> salt >> stored_hashed_password;
            unsigned long HP = djb2(pass + to_string(salt));
            if (stored_hashed_password == HP){
                cout<<"The username is :   "<<stored_username<<endl;
                passfile.close();
                return;
            }

            }
            cout << "No such password." << endl;
            passfile.close();
        }
    else{
        cout<<"File cannot be opened!!"<<endl;
    }
}
```

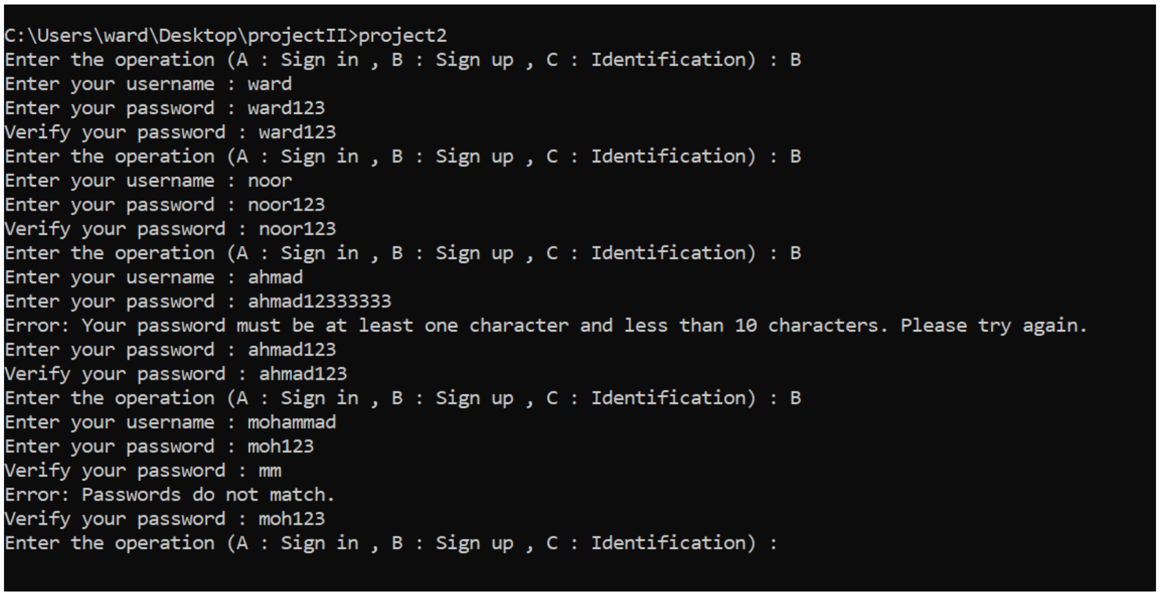
7) **The main function**: the main function is an infinite loop function that will stay handling any request of all services showed before, by a simple multiple choices format, A for signing in, B for signing up and C for identification, this loop will end only if the user pass (//exit) operation, this will breaking up the loop and terminate the process.



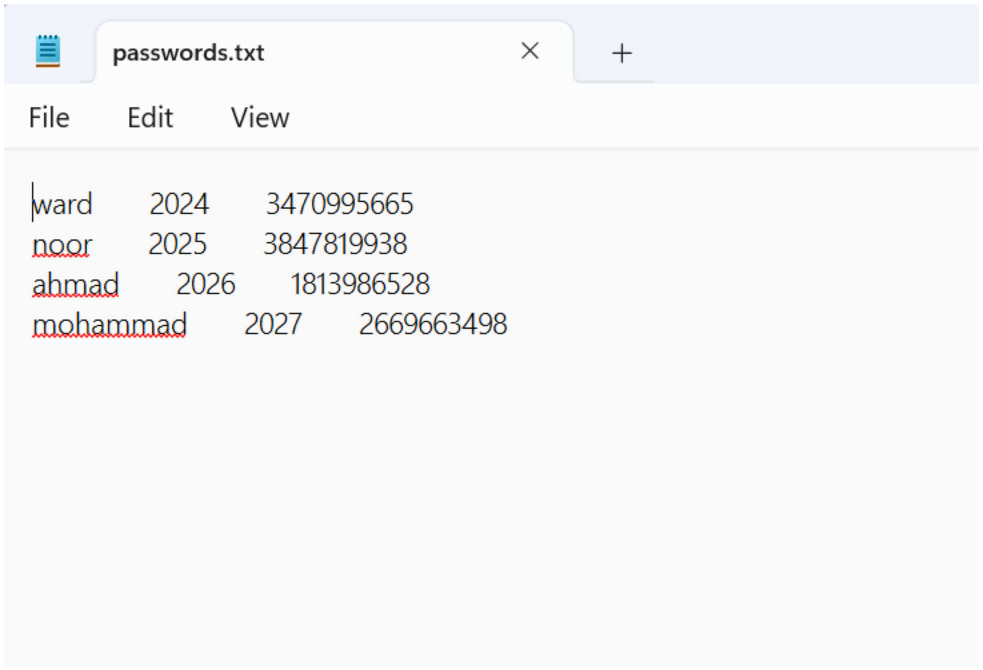
```
// // ///// ///// ///// ///// ///// //// ///// //// /////
int main() {
    string op;
    bool project = true;
    while (project) {
        cout << "Enter the operation (A : Sign in , B : Sign up , C : Identification) : ";
        cin >> op;
        if (op == "A") {
            SIGNIN();
        } else if (op == "B") {
            SIGNUP();
        }
        else if (op == "C"){
        Identification();
        }
        else if(op == "//exit"){
            project = false;
        }
        else  {
            cout << "\nWRONG INPUT!!\n";
        }
    }
    return 0;
}
```

TESTING:

1) Signing up:

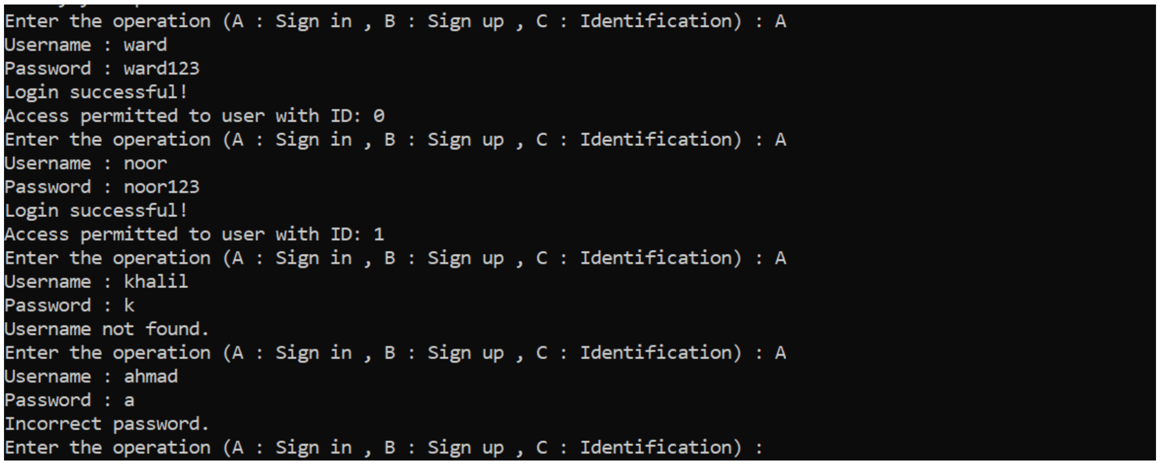


```
C:\Users\ward\Desktop\projectII>project2
Enter the operation (A : Sign in , B : Sign up , C : Identification) : B
Enter your username : ward
Enter your password : ward123
Verify your password : ward123
Enter the operation (A : Sign in , B : Sign up , C : Identification) : B
Enter your username : noor
Enter your password : noor123
Verify your password : noor123
Enter the operation (A : Sign in , B : Sign up , C : Identification) : B
Enter your username : ahmad
Enter your password : ahmad12333333
Error: Your password must be at least one character and less than 10 characters. Please try again.
Enter your password : ahmad123
Verify your password : ahmad123
Enter the operation (A : Sign in , B : Sign up , C : Identification) : B
Enter your username : mohammad
Enter your password : moh123
Verify your password : mm
Error: Passwords do not match.
Verify your password : moh123
Enter the operation (A : Sign in , B : Sign up , C : Identification) :
```



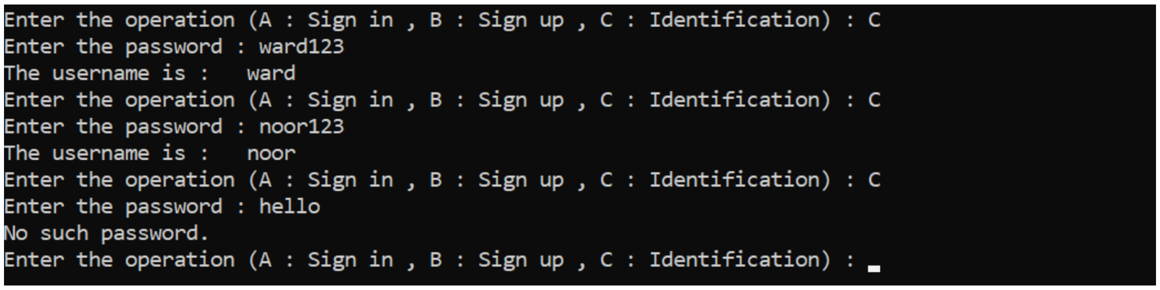
```
ward      2024      3470995665
noor      2025      3847819938
ahmad      2026      1813986528
mohammad      2027      2669663498
```

2) Signing in:



```
Enter the operation (A : Sign in , B : Sign up , C : Identification) : A
Username : ward
Password : ward123
Login successful!
Access permitted to user with ID: 0
Enter the operation (A : Sign in , B : Sign up , C : Identification) : A
Username : noor
Password : noor123
Login successful!
Access permitted to user with ID: 1
Enter the operation (A : Sign in , B : Sign up , C : Identification) : A
Username : khalil
Password : k
Username not found.
Enter the operation (A : Sign in , B : Sign up , C : Identification) : A
Username : ahmad
Password : a
Incorrect password.
Enter the operation (A : Sign in , B : Sign up , C : Identification) :
```

3) Identification:



```
Enter the operation (A : Sign in , B : Sign up , C : Identification) : C
Enter the password : ward123
The username is :   ward
Enter the operation (A : Sign in , B : Sign up , C : Identification) : C
Enter the password : noor123
The username is :   noor
Enter the operation (A : Sign in , B : Sign up , C : Identification) : C
Enter the password : hello
No such password.
Enter the operation (A : Sign in , B : Sign up , C : Identification) :
```

Eng. Ward Mohammad Ghnaim.

Eng. Noor Aldden Shamroukh.