AURORA DEEMED TO BE UNIVERSITY

AURORA HIGHER EDUCATION AND RESEARCH ACADEMY

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING WITH ARTIFICAL INTELLIGENCE AND MACHINE LEARNING (2024-2025)

Term – III

SUBJECT : Data Structure

TOPIC : Simple Password Storage System Using Hashing

LG – 08

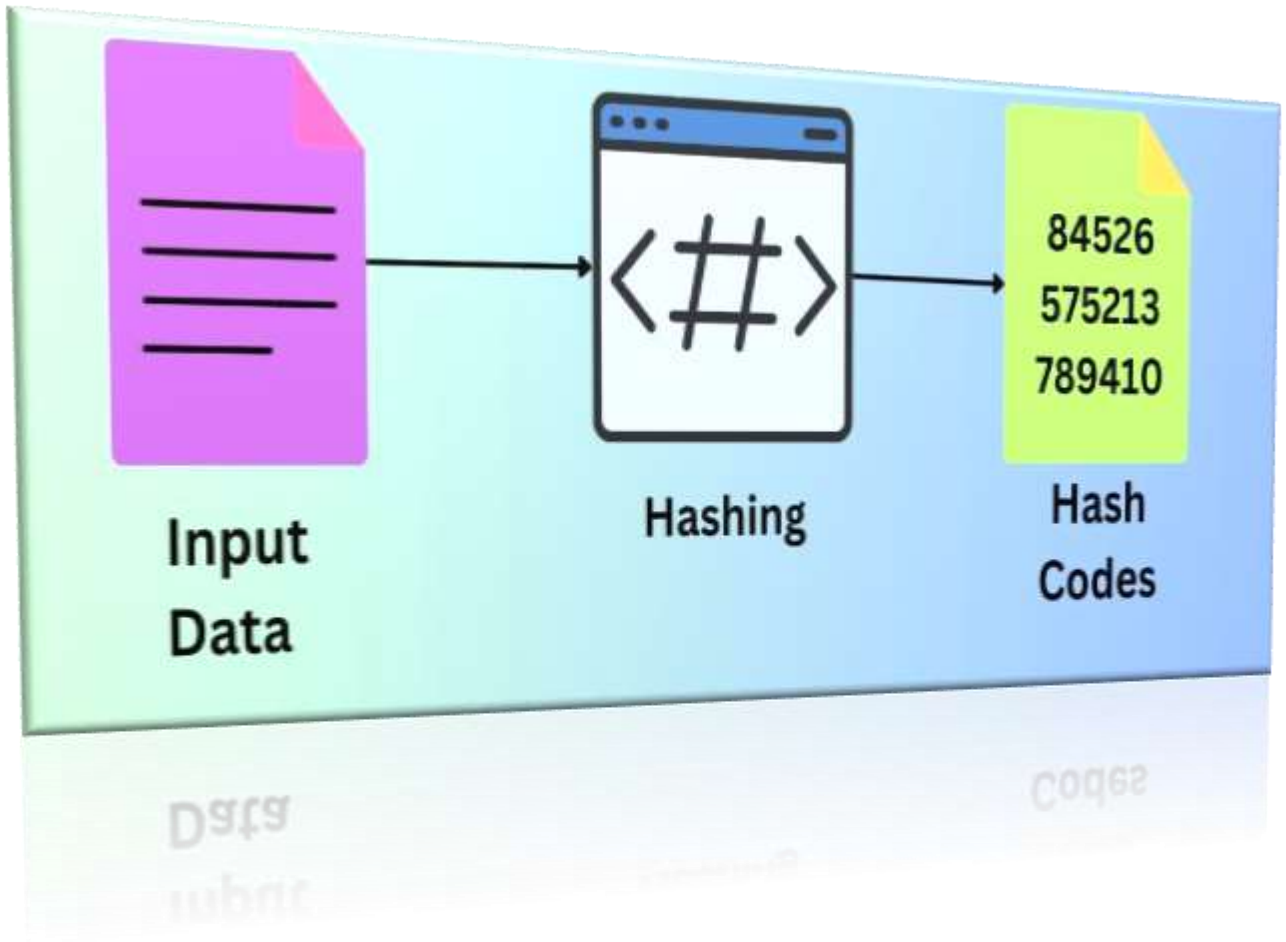
NAME	ROLL NO
P. SAI SUCHINDRA	241U1R2077
A. VIGNESH	241U1R2089

Faculty: Mr. Vemula Pranay

Assistant Professor, Department of CSE

OUTLINE :

1. **ABSTRACT**
2. **INTRODUCTION**
3. **OBJECTIVES**
4. **METHODOLOGY**
5. **SOFTWARE REQUIREMENTS**
6. **FUNCTIONAL REQUIREMENTS**
7. **EXECUTION SCREENSHOTS**
8. **TEST CASES**
9. **CODE IN SEPARATE FILE**
10. **CREATIVITY AND INNOVATION**
11. **CONTRIBUTION TO THE SOCIETY**
12. **CONCLUSION**



ABSTRACT

Passwords are a key aspect of securing user identity in digital systems. However, storing passwords in plain text is a critical security flaw. This project presents a Simple Password Storage System, written in the C programming language, that uses a non-reversible hashing technique (djb2) to securely store and manage passwords in a file. Data is stored in a text file (users.txt), simulating a lightweight database. This approach avoids reliance on external libraries and databases, making the system portable, lightweight, and educational. The project is designed with a focus on learning, making it a valuable tool for beginners in programming and cybersecurity. It bridges theoretical knowledge with practical implementation, fostering a deeper understanding of user management systems.

INTRODUCTION

In today's digital world, keeping user passwords safe is very important. Storing passwords as plain text is risky because anyone who gets access to the data can see them. Our project offers a simple and secure way to store passwords using hashing in the C programming language.

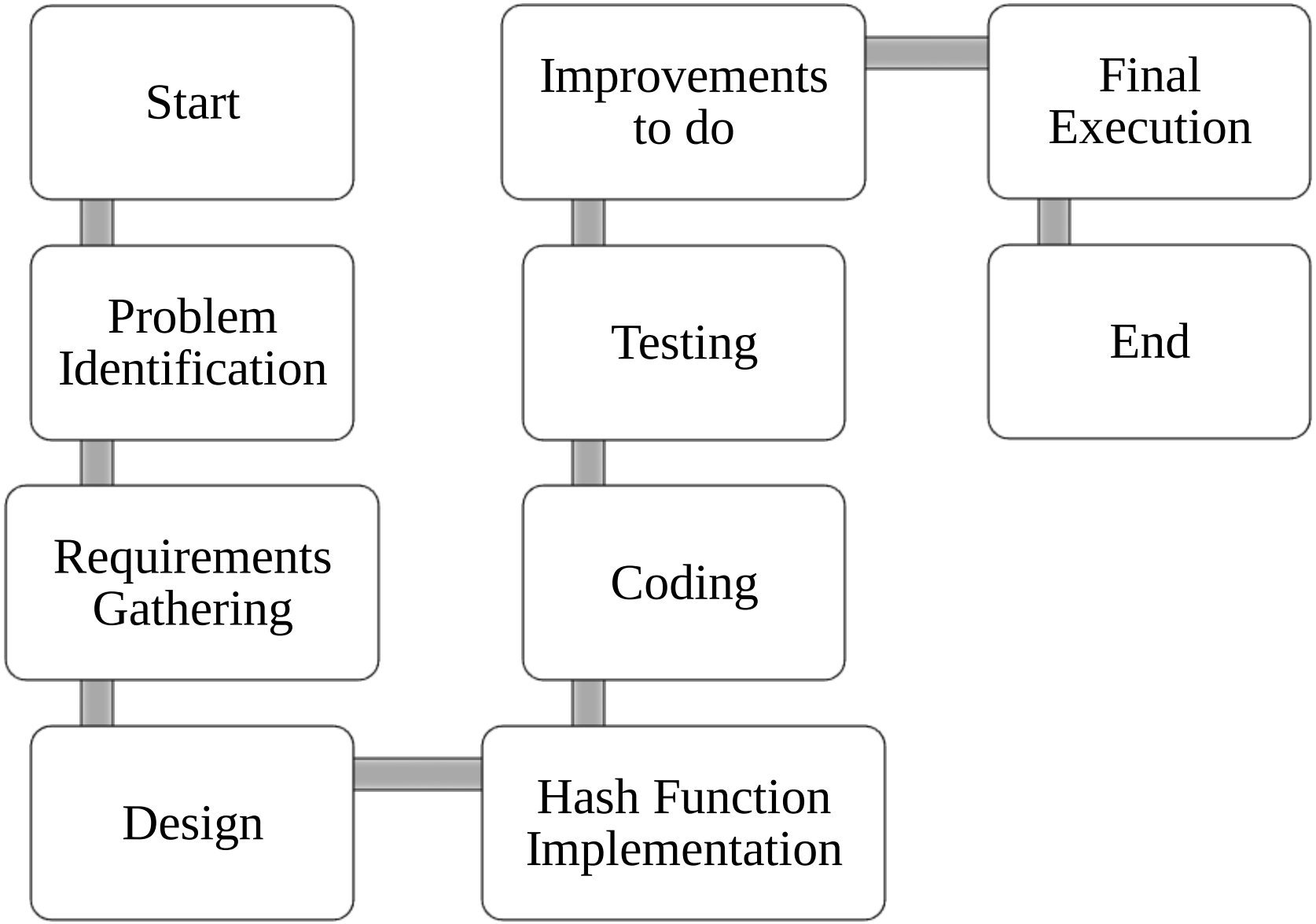
Hashing turns a password into a fixed length code that looks random and cannot be reversed. This helps protect the original password even if the data is exposed.

```
16  User users[MAX_USERS];
17  int user_count = 0;
18
19  unsigned long simple_hash(const char *str) {
20      unsigned long hash = 5381;
21      int c;
22
23      while ((c = *str++)) {
24          hash = ((hash << 5) + hash) + c;
25      }
26
27      return hash;
28  }
```

OBJECTIVES

- **To implement secure password storage** using hashing techniques (like custom hash functions) to protect user credentials from unauthorized access.
- **To develop a user-friendly system** that allows users to register and authenticate securely by comparing hashed passwords instead of storing them in plain text.
- **To demonstrate the use of hashing algorithms** in C programming for converting plaintext passwords into irreversible hash codes.
- **To ensure data integrity and privacy** by preventing password leakage even if the data file is compromised.
- **To enhance practical knowledge** of file handling, string manipulation, and hashing functions in the C language through a real-world application.

METHODOLOGY



SOFTWARE REQUIREMENTS

1.Code Editor

- Visual Studio Code (VS Code)

2.C Compiler

- **MinGW** for compiling and running C programs

3.C Standard Library

- For file handling (<stdio.h>), to provide standard functions (<stdlib.h>) ,string manipulation (<string.h>), and standard functions

4.Hashing Support

- Custom hash function implemented in C

5.Operating System

- Windows 10/11
- Alternatively: Linux or macOS (with minor changes in file path handling)

6.Text File for Data Storage

- A .txt file to store usernames and hashed passwords
- Acts as a database

FUNCTIONAL REQUIREMENTS

1. **User Registration:**

   Allows the user to create an account with a username and password.

2. **Password Hashing:**

   Converts the password into a hash before storing it.

3. **Data Storage:**

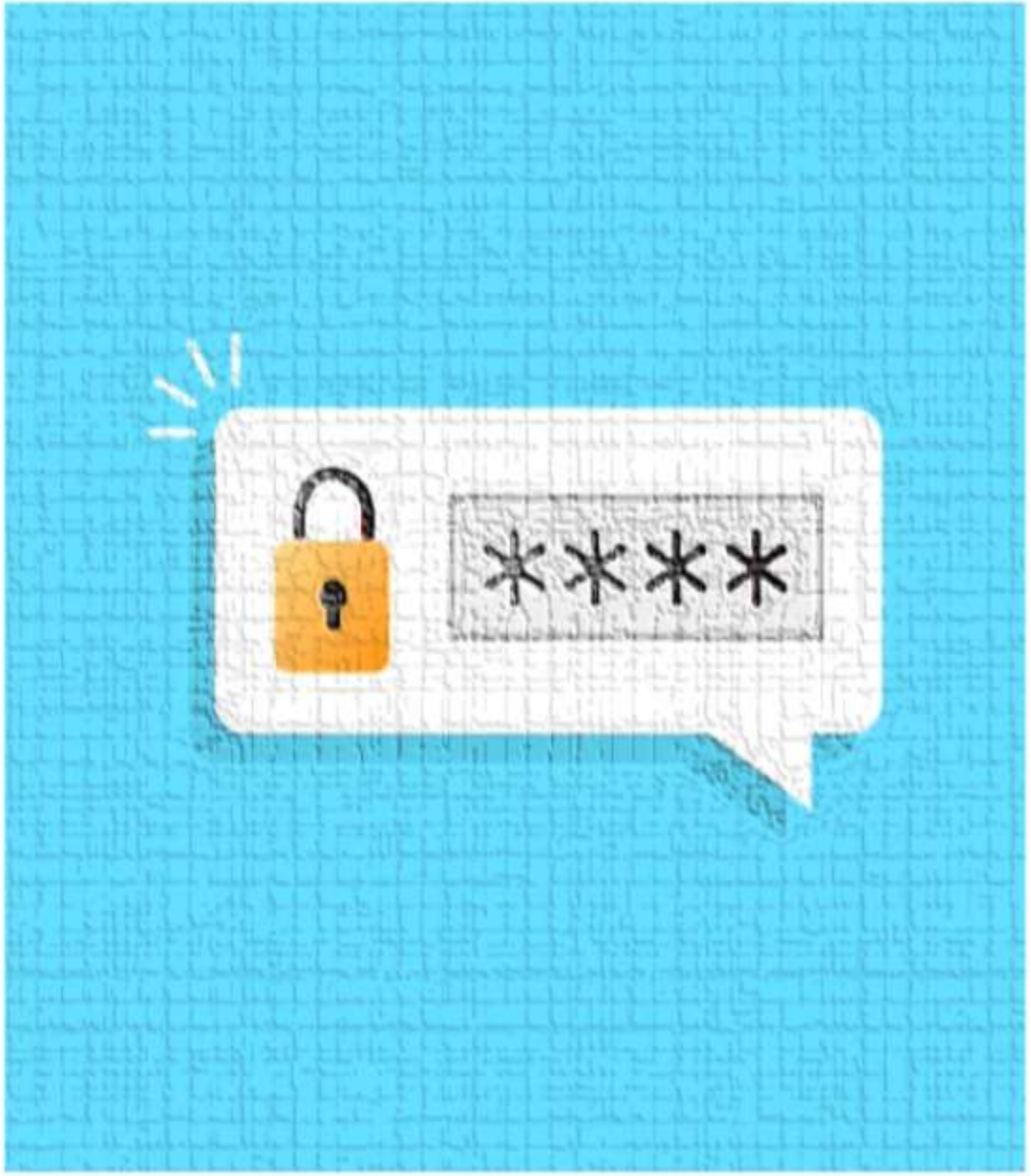
   Stores the username and hashed password in a text file.

4. **User Login:**

   Lets the user log in by comparing the entered password's hash with the stored one.

5. **Input Validation & Errors:**

   Shows messages for duplicate usernames, wrong passwords, or empty inputs.



EXECUTION SCREENSHOTS

```
--- Password Storage System ---
1. Register new user
2. Login user
3. View stored users
4. Delete user
5. Search user
6. Exit
Choose an option: 1
Enter username: Vignesh
Enter password: 124578
User registered and saved successfully!
```

(i)In this we a registering a new user

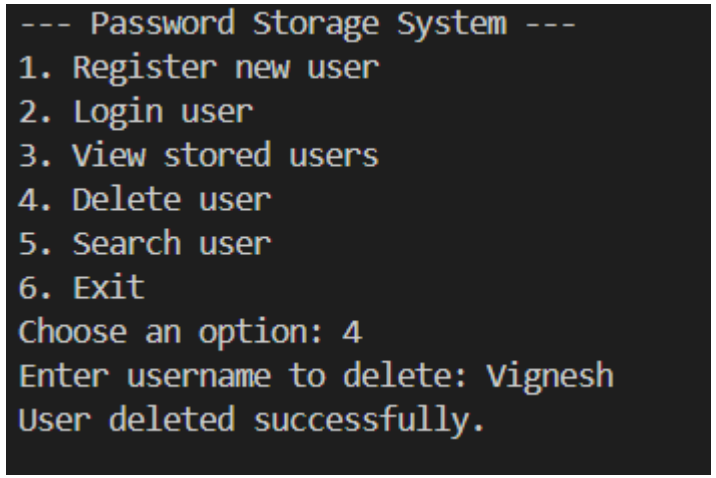
```
--- Password Storage System ---
1. Register new user
2. Login user
3. View stored users
4. Delete user
5. Search user
6. Exit
Choose an option: 2
Enter username: Vignesh
Enter password: 124578
Login successful!
```

(ii)In this we are Login in with the user name and password

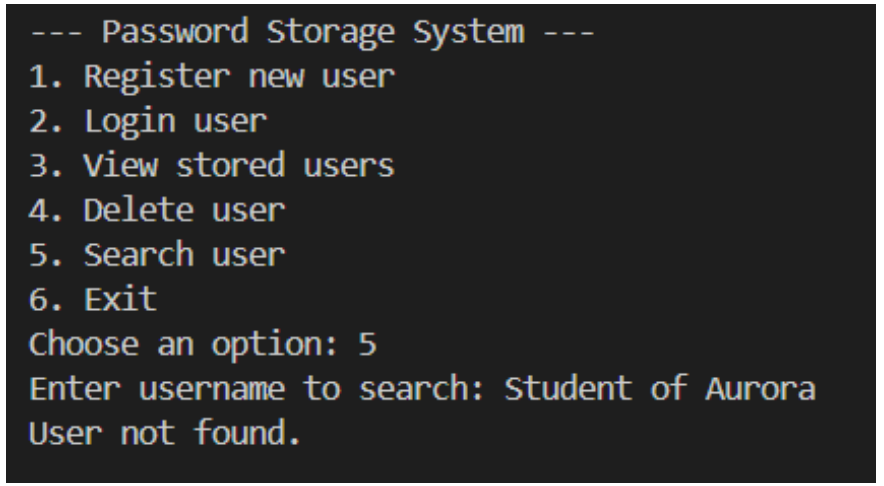
```
--- Password Storage System ---
1. Register new user
2. Login user
3. View stored users
4. Delete user
5. Search user
6. Exit
Choose an option: 3
Enter master password to view users: admin123

--- Stored Users ---
1. Username: Aurora | Password Hash: 2088290737
2. Username: Vignesh | Password Hash: 2110914880
```

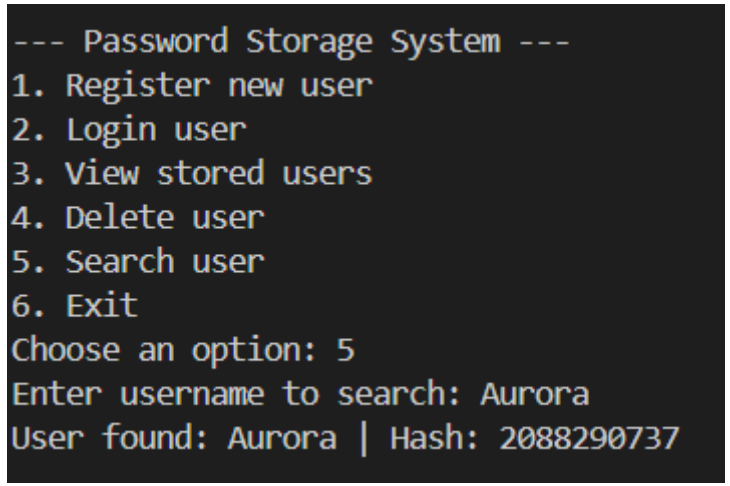
(iii)In this case we can see the stored users with a master password kept in the code

EXECUTION SCREENSHOTS



(iv) In this we can delete the stored used by entering user name



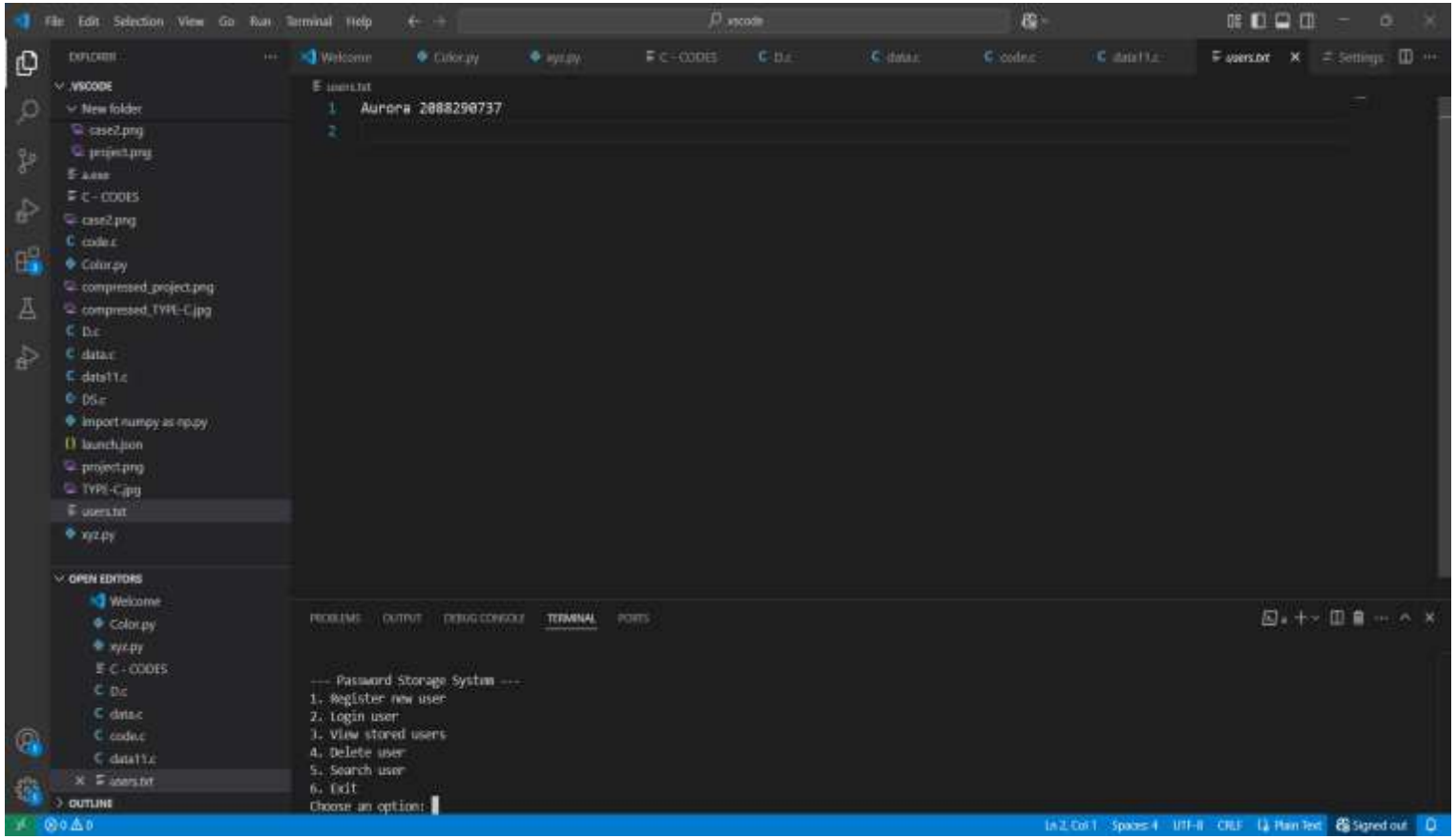


(v)In this the 1st image if we are entering the user which is not stored in the file then it is showing user not found,
In the 2nd image we can that if the enter the stored user name then it is giving the Hash Password instead of original password which was stored

EXECUTION SCREENSHOTS

```
--- Password Storage System ---
1. Register new user
2. Login user
3. View stored users
4. Delete user
5. Search user
6. Exit
Choose an option: 6
Exiting...
```

(vi)In this we can simply exit the program



(vii) Here we can see that the data which we are entering is stored in a text file with a hash value

TEST CASES

Test Case ID	Scenario	Input	Expected Output	Result
TC01	Register new user	test1 / pass1	User registered successfully	✓
TC02	Register existing user	test1 / newpass	Username already exists!	✓
TC03	Login with correct details	test1/pass1	Login successful!	✓
TC04	Login with wrong password	test1/wrongpass	Invalid username or password	✓
TC05	Delete existing user	test1	User deleted successfully	✓
TC06	Delete non-existent user	User which has not stored	User not found	✓

CODE IN SEPARATE FILE

CREATIVITY AND INNOVATION

In our project, creativity is shown through the custom hashing algorithm used to securely store passwords. We added features like input validation to catch errors, and clear feedback messages such as "Login Successful" or "Wrong Password" to make the system easier to use. In this we have added a Master password to see the stored data in the txt file . The simple menu interface makes the system straight forward and user-friendly, improving the overall experience.

CONTRIBUTION TO THE SOCIETY

The Password Storage System Using Hashing project helps individuals understand the importance of data security in the digital world. It encourages people to adopt secure practices, such as password hashing, to protect their personal information. This system promotes cybersecurity awareness by showing how simple techniques can improve security. It is designed for students and beginners, offering an easy way to learn about data protection. The project also highlights how technology can be used to create safer, more secure systems, fostering a deeper understanding of digital privacy and security.

CONCLUSION

The Password Storage System Using Hashing successfully demonstrates how to store user credentials securely using a custom hash function in C. It provides a basic but effective way to protect sensitive data, avoiding the risks of storing plaintext passwords. The project also highlights the importance of cybersecurity, user-friendly design, and secure programming practices. The project serves as a practical foundation for understanding password security and can be further improved with more advanced features in the future.

THANK YOU