VARDHAMAN COLLEGE OF ENGINEERING

(AUTONOMOUS)

Affiliated to **JNTUH**, Approved by **AICTE**, Accredited by **NAAC** with **A++** Grade, **ISO 9001:2015** Certified
Kacharam , Shamshabad, Hyderabad – 501218, Telangana, India

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

A Course End Project Report towards Advanced Data Structures Laboratory titled

USER AUTHENTICATION SYSTEM

Submitted in the partial fulfillment of the requirements

for the course end project of

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE AND ENGINEERING

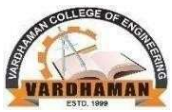
Submitted

By

Roll number	Name of the Student
22881A05G1	K. S. VADANA SRI

To

Dr. S V Vasantha
ASSOCIATE PROFESSOR



VARDHAMAN COLLEGE OF ENGINEERING

(AUTONOMOUS)

Affiliated to **JNTUH**, Approved by **AICTE**, Accredited by **NAAC** with **A++** Grade, **ISO 9001:2015** Certified

Kacharam, Shamshabad, Hyderabad – 501218, Telangana, India

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that the project titled "USER AUTHENTICATION SYSTEM"

is submitted by

Roll Number	Name of the Student
22881A05G1	K. S. VADANA SRI

in partial fulfillment of the requirements for the course end project for the course Advanced Data Structures Laboratory for the academic year 2022-23

Signature of the Instructor

Signature of the HOD

PROBLEM STATEMENT:

Develop a user authentication system in C, featuring secure registration, login, and password reset capabilities. Utilize a hash table to store user data securely, ensuring uniqueness for usernames and hashing passwords during storage. Implement account locking after consecutive failed login attempts, with a timed unlock feature. Password resets are permitted only with a matching email address.

ALGORITHM/IMPLEMENTATION:

Registration:

Users provide a unique username, password, and email.
Passwords are securely hashed.
Ensure usernames are unique.

Login:

Users enter their username and password.
Lock accounts after failed attempts, unlocking after a duration.
Update last login timestamp on successful login.

Password Reset:

Users securely reset passwords with username and email.
Verification through email is required.

Account Locking:

Lock accounts after consecutive failed login attempts.
Automatic unlock after a set time.

Hashing:

Hash usernames for uniqueness and passwords for secure storage.

Menu Interface:

Develop a user-friendly menu system for interactions.

Security Measures:

Use secure hashing algorithms for passwords.
Ensure robust password reset mechanisms.
Guard against common security threats.

Hash Function (hash):
The hash function takes a string key (e.g., username) and converts it into an integer hash value. It uses a simple sum of ASCII values modulo the hash table size.

User Creation (createUser):
The createUser function initializes a new User structure with the provided username, password, and email. It sets other fields such as lastLogin, loginAttempts, and locked to initial values.

Hash Node Creation (createHashNode):
The createHashNode function allocates memory for a new HashNode and initializes it with the given User structure.

User Insertion (insertUser):
The insertUser function inserts a user into the hash table. It calculates the hash index using the hash function and adds the new user as a node at the beginning of the linked list at that index.

User Finding (findUser):
The findUser function searches for a user in the hash table based on the provided username. It calculates the hash index and traverses the linked list at that index to find the user.

Account Locking (isAccountLocked):
The isAccountLocked function checks if a user account is currently locked. If the account is locked, it compares the time elapsed since the last login. If the time exceeds the account lock duration, it unlocks the account.

User Login (loginUser):
The loginUser function allows a user to log in by entering a username and password. It uses the isAccountLocked function to check if the account is locked. It hashes the entered password and compares it with the stored hashed password.

Password Reset (resetPassword):
The resetPassword function allows users to reset their passwords by providing their username and matching email. It hashes and updates the password if the email matches.

User Registration (registerUser):
The registerUser function allows users to register by providing a new username,

password, and email. It hashes the password and inserts the new user into the hash table.

Main Loop (main):

The main function implements a simple menu-driven interface for users to register, log in, reset their password, or exit the system. It repeatedly prompts the user for their choice until they choose to exit.

CODE:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>
#include <time.h>
#define HASH_TABLE_SIZE 100
#define MAX_LOGIN_ATTEMPTS 3
#define ACCOUNT_LOCK_DURATION 300
typedef struct {
    char username[50];
    char password[64];
    time_t lastLogin;
    int loginAttempts;
    bool locked;
    char email[50];
} User;
typedef struct HashNode {
    User user;
    struct HashNode *next;
} HashNode;
typedef struct {
    HashNode *table[HASH_TABLE_SIZE];
} Hashtable;
int hash(char *key) {
    int hash = 0;
    while (*key) {
        hash += *key;
       key++;
   }
    return hash % HASH_TABLE_SIZE;
}
User createUser(char *username, char *password, char *email) {
 User newUser;
    strcpy(newUser.username, username);
    strcpy(newUser.password, password);
    newUser.lastLogin = 0;
```

```
    newUser.loginAttempts = 0;
    newUser.locked = false;
    strcpy(newUser.email, email);
    return newUser; }
HashNode *createHashNode(User user) {
    HashNode *newNode = (HashNode *)malloc(sizeof(HashNode));
    newNode->user = user;
    newNode->next = NULL;
   return newNode; }
void insertUser(Hashtable *hashTable, User user) {
    int index = hash(user.username);
    HashNode *newNode = createHashNode(user);
    newNode->next = hashTable->table[index];
    hashTable->table[index] = newNode; }
User *findUser(Hashtable *hashTable, char *username) {
    int index = hash(username);
    HashNode *current = hashTable->table[index];
    while (current != NULL) {
        if (strcmp(current->user.username, username) == 0) {
      return &(current->user);   }
        current = current->next;     }
    return NULL; }
bool isAccountLocked(User *user) {
    if (user->locked) {
        time_t currentTime;
        time(&currentTime);
        if (currentTime - user->lastLogin < ACCOUNT_LOCK_DURATION) {
            return true;
  } else {
            user->locked = false;
            user->loginAttempts = 0;
           return false;   }    }    return false; }
void loginUser(Hashtable *hashTable) {
    char username[50];
    char password[50];
    printf("Enter your username: ");
 scanf("%s", username);
    User *user = findUser(hashTable, username);
    if (user != NULL) {
        if (isAccountLocked(user)) {
            printf("Account locked. Try again later.\n");
           return;     }
        printf("Enter your password: ");
        scanf("%s", password);
 unsigned int hash = 0;
    while (*password) {
```

```
        hash = (hash << 5) + (*password)++; }
    snprintf(password, 11,"%u",hash);
    if (strcmp(user->password, password) == 0) {
     printf("Login successful! Welcome, %s!\n", username);
     user->loginAttempts = 0;
 } else {    printf("Invalid username or password. Please try again.\n");
            user->loginAttempts++;
            if (user->loginAttempts >= MAX_LOGIN_ATTEMPTS) {
             printf("Too many failed attempts. Your account is locked.\n");
                user->locked = true;
  time(&user->lastLogin);
  }}}
 else {        printf("User not found. Please register.\n");     } }
void resetPassword(Hashtable *hashTable) {
    char username[50];
    char email[50],str[50];
    printf("Enter your username: ");
    scanf("%s", username);
 User *user = findUser(hashTable, username);
    if (user != NULL) {     if (isAccountLocked(user)) {
            printf("Account locked. Password reset not allowed.\n");
            return;       }
        printf("Enter the email associated with your account: ");
        scanf("%s", email);
        if (strcmp(user->email, email) == 0) {
            char newPassword[50];
       printf("Enter your new password: ");
    scanf("%s", str);
          unsigned int hash = 0;
    while (*str) {
        hash = (hash << 5) + (*str)++;    }
    snprintf(newPassword, 11,"%u",hash);
          strcpy(user->password, newPassword);
           printf("Password reset successful!\n");  } else {
       printf("Invalid email. Password reset failed.\n");
        }    } else {
       printf("User not found. Please register.\n");    } }
void registerUser(Hashtable *hashTable) {
    char username[50];
    char password[50];
    char email[50];
    char str[50];
    printf("Enter a new username: ");
    scanf("%s", username);
    if (findUser(hashTable, username) != NULL) {
        printf("Username already exists. Choose a different one.\n");
```

```
        return;   }
 printf("Enter a password: ");
    scanf("%s", str);
 printf("Enter your email: ");
    scanf("%s", email);
    unsigned int hash = 0;
    while (*str) {
        hash = (hash << 5) + (*str)++; }
    snprintf(password, 11,"%u",hash);
    User newUser = createUser(username, password, email);
    insertUser(hashTable, newUser);
    printf("User registered successfully!\n");
int main() {   Hashtable userTable;
    for (int i = 0; i < HASH_TABLE_SIZE; i++) {
        userTable.table[i] = NULL;  }
    int choice;
    do {
       printf("\nUser Authentication System\n");
        printf("1. Register\n");
        printf("2. Login\n");
        printf("3. Reset Password\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        switch (choice) {
          case 1:
              registerUser(&userTable);
              break;
           case 2:      loginUser(&userTable);
              break;
           case 3:  resetPassword(&userTable);
             break;
           case 4:
              printf("Exiting...\n");
              break;
           default:
              printf("Invalid choice. Please try again.\n");
        }
    } while (choice!= 4);   return 0;
}
```

Time complexity:

Best case: O(1)

Worst case: O(n)

GITHUB LINK:

https://github.com/vadana665/ADS-

OUTPUT:

```
User Authentication System
1. Register
2. Login
3. Reset Password
4. Exit
Enter your choice: 1
Enter a new username: kiran
Enter a password: kiran32
Enter your email: kiran@gmail.com
User registered successfully!

User Authentication System
1. Register
2. Login
3. Reset Password
4. Exit
Enter your choice: 1
Enter a new username: rani
Enter a password: rani@31
Enter your email: rani23@gmail.com
User registered successfully!

User Authentication System
1. Register
2. Login
3. Reset Password
4. Exit
Enter your choice: 2
Enter your username: kirane3
User not found. Please register.

User Authentication System
1. Register
2. Login
3. Reset Password
4. Exit
Enter your choice: 2
```

```
User Authentication System
1. Register
2. Login
3. Reset Password
4. Exit
Enter your choice: 1
Enter a new username: kiran
Enter a password: kiran32
Enter your email: kiran@gmail.com
User registered successfully!

User Authentication System
1. Register
2. Login
3. Reset Password
4. Exit
Enter your choice: 1
Enter a new username: rani
Enter a password: rani@31
Enter your email: rani23@gmail.com
User registered successfully!

User Authentication System
1. Register
2. Login
3. Reset Password
4. Exit
Enter your choice: 2
Enter your username: kirane3
User not found. Please register.

User Authentication System
1. Register
2. Login
3. Reset Password
4. Exit
Enter your choice: 2
```

```
User Authentication System
1. Register
2. Login
3. Reset Password
4. Exit
Enter your choice: 2
Enter your username: rani
Enter your password: rani@31
Login successful! Welcome, rani!

User Authentication System
1. Register
2. Login
3. Reset Password
4. Exit
Enter your choice: 4
Exiting...
```

CONCLUSION

The provided C code implements a basic user authentication system using a hash table. Features include registration, login, and password reset with security measures such as password hashing and account locking. While the average-case time complexity for key operations is reasonable, improvements, such as advanced hashing algorithms, are recommended for enhanced security.