INTRODUCTION

The **login screen** is a fundamental feature in modern applications that require user authentication. It serves as the gateway for users to access the system by providing their credentials, typically a username or email, combined with a password. This interface plays a critical role in securing sensitive information and ensuring that only authorized individuals can access protected resources. In addition to being the first point of interaction with the system, the login screen is essential in safeguarding user privacy and system integrity. Its goal is to verify the identity of users, thus preventing unauthorized access, data breaches, or other malicious activities.

In **C programming**, the creation of a login screen involves the use of various functions and libraries to handle user input, perform validation checks, and ensure secure processing of sensitive data. Despite being a relatively low-level language, C provides robust capabilities for handling these tasks, from basic input/output operations to more advanced file handling and encryption techniques. Developing a login screen in C not only demonstrates the language's efficiency but also highlights the importance of secure coding practices, such as validating user credentials and safely storing passwords.

This project illustrates how a basic yet secure login system can be developed using **C**, with a focus on practical security measures like password masking, encryption (if necessary), and efficient validation techniques. The simplicity and performance of C, coupled with its close-to-hardware nature, make it a powerful tool for building such essential

authentication mechanisms. Furthermore, this project emphasizes the use of secure coding practices to mitigate risks like buffer overflows and improper memory handling, which are common vulnerabilities in C-based systems.

A **login system** is a critical security component in most software applications. It allows users to create an account by signing up with details such as their full name, email, phone number, and password, which are securely stored in a binary file. Later, users can log in using their username and password combination. The file-handling feature in C is employed to read and write user data, ensuring that the login system retains user information between sessions. This makes the program not only practical but also persistent across multiple uses.

The program leverages various core concepts of the **C language**, including file handling, string manipulation, and user input validation. Through this project, users can see how a real-world application of these concepts can be implemented to solve a common problem—user authentication. It highlights the importance of basic security practices in application development and provides a stepping stone toward more advanced systems, like multi-factor authentication or cryptographic password storage. By demonstrating how C can be applied to create a simple yet effective login system, this project offers both practical value and educational insight into how essential security features are built.

THEORETICAL PART

2.1 Overview of Login Systems:

A **login system** is a fundamental security feature in software applications that restricts access to authorized users. Typically, it requires users to input credentials, such as a username and password, which are compared to pre-stored information in a secure database or file. When the credentials match, the user is authenticated and granted access; otherwise, access is denied. In addition to username and password authentication, modern login systems may include multi-factor authentication (MFA) and biometric checks, but the core concept remains validating user identity before allowing access to sensitive parts of the system.

In this project, a simple login and registration system is created using the **C language**, demonstrating how basic authentication works. The system allows new users to sign up and stores their information securely. Existing users can then log in using their credentials, which are verified against the stored data. This implementation forms the basis for many real-world applications, from desktop software to web applications.

2.2 Concepts Used in the Program:

1. File Handling:

In this project, file handling is crucial for storing and retrieving user information between sessions. The program utilizes **C's file I/O functions** to manage user data. The fopen() function is

used to open a file named Users.dat, where user credentials (such as full name, email, password, and phone number) are stored. During user registration, the program writes new user data to the file using the fwrite() function, ensuring that the data is saved in binary format for security and efficiency. When users attempt to log in, the program reads data from the file using fread() to compare credentials.

This ensures **data persistence**, meaning that even after the program is closed, the user data remains stored in the file, allowing users to log in at any time. File handling also enables the program to handle multiple users by appending new data or reading existing records.

2. String Handling:

String manipulation plays an important role in the program, particularly for handling user input and validation. Functions such as fgets() are used to take user input securely, while strcmp() is employed to compare user-entered strings (e.g., comparing passwords during login or signup). Additionally, a custom function is used to generate usernames from email addresses, which extracts the portion of the email before the @ symbol. This feature showcases how string operations can be utilized to simplify user management.

Proper string handling is vital to prevent common security issues like buffer overflows, and the program uses fgets() to ensure that input is safely limited to the buffer size.

3. Input Validation:

Input validation is critical for the security and functionality of any login system. In this program, several validation

mechanisms are implemented. During signup, the program ensures that the user's password is entered twice and checks for consistency by comparing the two password entries. If the passwords do not match, the user is alerted, and the registration is halted. Similarly, during login, the system checks both the username and password against stored data to ensure that they match before granting access.

Validation also helps in preventing unauthorized access by ensuring that input adheres to the expected format, and that no invalid or harmful data is processed by the system.

4. Secure Input (Password Masking):

One of the user-experience and security features included in this program is **password masking**, which ensures that the user's password is not displayed on the screen while they are typing. This is achieved through the use of the getch() function, which captures each character of the password input without echoing it to the console. For each character entered, an asterisk (*) is displayed instead. This prevents onlookers from seeing the user's password, a basic but effective security measure.

5. Conditional Statements and Loops:

The program employs **if-else statements** and loops to handle the decision-making and repetitive tasks involved in the login system. For instance, if the entered username and password match the stored credentials, access is granted. Otherwise, the program prompts the user with an error message and allows them to re-enter their credentials. Loops are also used to iterate through the list of registered users stored in the file, ensuring that the login system can accommodate multiple users. This structure keeps the program efficient and easy to understand.

2.3 File Structure and Data Persistence:

The program uses a binary file, Users.dat, to store and manage user information. This file structure allows for **data persistence**, meaning that once user data is written to the file, it remains intact even after the program is closed. This is crucial for any real-world login system, as it ensures that users can log in across multiple sessions without having to re-register.

- **File Operations:** The program uses fopen() in **a+ mode** during the signup process. This mode allows the program to **append** new users to the file without overwriting existing data. For the login process, the file is opened in **read (r) mode**, which allows the system to search for and retrieve user credentials. This separation of file operations ensures that the system remains efficient and does not corrupt or lose user data.
- **Binary Storage:** Storing user data in binary format using fwrite() and fread() adds an extra layer of security, as it makes the stored information less readable than plain text files. Binary files also tend to be smaller and faster to process, making them an ideal choice for user data storage in C.

2.4 Security Considerations:

While this program provides a basic and functional login system, there are important **security limitations** that must be considered for real-world applications. Currently, the program

stores user passwords in plaintext within the binary file. This leaves user credentials vulnerable to unauthorized access, especially if the file is compromised. In real-world scenarios, best practices dictate that passwords should never be stored in plaintext.

To enhance security, the program could incorporate **password hashing algorithms**, such as **SHA-256** or **bcrypt**, to ensure that even if the user data file is accessed, the actual passwords are protected. Hashing converts passwords into a fixed-length string of characters, which cannot easily be converted back to the original password. In addition, using **salted hashing** adds further security by ensuring that even identical passwords result in different hash outputs.

Other potential improvements include encrypting the Users.dat file or implementing **multi-factor authentication** (MFA) to provide an extra layer of security. These measures would significantly reduce the risk of unauthorized access and protect user data against modern security threats, making the system far more secure for real-world use.

CODE

```
// Standard Input Output Library
#include <stdio.h>
// Console Input Output (for getch())
#include <conio.h>
// Windows-specific functions (for system(), Beep())
#include <windows.h>

// Defining ENTER key ASCII value
#define ENTER 13
// Defining TAB key ASCII value
#define TAB 9
// Defining BACKSPACE key ASCII value
#define BCKSPC 8

// Structure to hold user information
struct user {
    char fullName[50]; // User's full name
    char email[50];    // User's email address
    char password[50]; // User's password
    char username[50]; // Username generated from the email
    char phone[50];    // User's contact number
};
```

```
// Function to take string input from the user
void takeinput(char ch[50]) {
// Get input from user with a maximum of 50 characters
    fgets(ch, 50, stdin);
// Remove the newline character at the end of the string
    ch[strlen(ch) - 1] = 0;
}

// Function to generate username from the email
char generateUsername(char email[50], char username[50]) {
    // Iterate through the email until '@' is found
    for (int i = 0; i < strlen(email); i++) {
        if (email[i] == '@') break;  // Stop when '@' is found
        else username[i] = email[i]; // Copy characters to username
    }
}

// Function to take password input and hide it with '*'
void takepassword(char pwd[50]) {
    int i = 0;
    char ch;
    while (1) {
        ch = getch(); // Read one character without displaying it on screen
// End input on ENTER or TAB
        if (ch == ENTER || ch == TAB) {
```

```
// Null terminate the password string
pwd[i] = '\0';
            break;
}
 // Handle BACKSPACE
         else if (ch == BCKSPC) {
            if (i > 0) {
// Move the index back
                i--;
// Erase the character on screen
                printf("\b \b");
            }
        } else {
// Add character to the password
            pwd[i++] = ch;
// Print '*' to hide the actual character
            printf("* \b");
        }
    }
}

int main() {
// Set console text color (light cyan on black)
    system("color 0b");
```

```
// File pointer for user data storage
FILE *fp;
// Option for menu and flag for user found
    int opt, usrFound = 0;
// Instance of struct user
    struct user user;
// Variable to store password confirmation
    char password2[50];

    // Display welcome message and options
    printf("\n\t\t\t\t----------Welcome to authentication system----------");
    printf("\nPlease choose your operation");
    printf("\n1. Signup");
    printf("\n2. Login");
    printf("\n3. Exit");

    printf("\n\nYour choice:\t");
// Take the user's choice
    scanf("%d", &opt);
// Consume the newline left by scanf
    fgetc(stdin);

    switch (opt) {
// Signup process
        case 1:
```

```
// Clear the screen
system("cls");
printf("\nEnter your full name:\t");
// Take full name input
   takeinput(user.fullName);
   printf("Enter your email:\t");

// Take email input
   takeinput(user.email);
   printf("Enter your contact no:\t");

// Take phone number input
   takeinput(user.phone);
   printf("Enter your password:\t");

// Take password input
   takepassword(user.password);
   printf("\nConfirm your password:\t");

// Confirm password
   takepassword(password2);

   // Check if passwords match
   if (!strcmp(user.password, password2)) {
```

```
        // Generate username
        generateUsername(user.email, user.username);

        // Open the file to append data
            fp = fopen("Users.dat", "a+");

        // Write user data to file
            fwrite(&user, sizeof(struct user), 1, fp);

         // Check if data is written successfully
            if (fwrite != 0)
               printf("\n\nUser registration success, Your username is %s", user.username);
            else
               printf("\n\nSorry! Something went wrong :(");

// Close the file
fclose(fp);
        } else {
            printf("\n\nPasswords do not match");
// Beep sound to indicate error
            Beep(750, 300);
        }
        break;
```

```
// Login process
case 2:
        {
// Variables for input
            char username[50], pword[50];
// Instance of struct user for login
            struct user usr;

            printf("\nEnter your username:\t");
// Take username input
            takeinput(username);
            printf("Enter your password:\t");
// Take password input
            takepassword(pword);

// Open file in read mode
fp = fopen("Users.dat", "r");
            // Read data from file and compare
            while (fread(&usr, sizeof(struct user), 1, fp)) {

// Check username
                if (!strcmp(usr.username, username)) {
```

```
// Check password

                    if (!strcmp(usr.password, pword)) {
// Clear screen on successful login
                        system("cls");
                        printf("\n\t\t\t\t\t\tWelcome %s", usr.fullName);
                        printf("\n\n| Full Name:\t%s", usr.fullName);
                        printf("\n| Email:\t%s", usr.email);
                        printf("\n| Username:\t%s", usr.username);
                        printf("\n| Contact no.:\t%s", usr.phone);
                    } else {
                        printf("\n\nInvalid Password!");
// Beep for wrong password
                        Beep(800, 300);
                    }
// Set flag if user is found
                    usrFound = 1;
                }
            }
            if (!usrFound) {
                printf("\n\nUser is not registered!");
// Beep if user not found
                Beep(800, 300);
            }
```

```
// Close the file
                fclose(fp);
            }
            break;

// Exit
        case 3:
            printf("\t\t\tBye Bye :)");
// Exit the program
            return 0;
    }

    return 0;
}
```

SCREENSHOT OF CODE

login.cpp

```
1   #include<stdio.h>
2   #include<conio.h>
3   #include<windows.h>
4   #define ENTER 13
5   #define TAB 9
6   #define BCKSPC 8
7
8   struct user{
9       char fullName[50];
10      char email[50];
11      char password[50];
12      char username[50];
13      char phone[50];
14  };
15
16  void takeinput(char ch[50]){
17      fgets(ch,50,stdin);
18      ch[strlen(ch) - 1] = 0;
19  }
20
21  char generateUsername(char email[50],char username[50]){
22      //abc123@gmail.com
23      for(int i=0;i<strlen(email);i++){
24          if(email[i] == '@') break;
25          else username[i] = email[i];
26      }
27  }
```

login.cpp

```
49
50  int main(){
51      system("color 0b");
52      FILE *fp;
53      int opt,usrFound = 0;
54      struct user user;
55      char password2[50];
56
57      printf("\n\t\t\t\t----------Welcome to authentication system----------");
58      printf("\nPlease choose your operation");
59      printf("\n1.Signup");
60      printf("\n2.Login");
61      printf("\n3.Exit");
62
63      printf("\n\nYour choice:\t");
64      scanf("%d",&opt);
65      fgetc(stdin);
66
67      switch(opt){
68          case 1:
69              system("cls");
70              printf("\nEnter your full name:\t");
71              takeinput(user.fullName);
72              printf("Enter your email:\t");
73              takeinput(user.email);
74              printf("Enter your contact no:\t");
75              takeinput(user.phone);
76              printf("Enter your password:\t");
77              takepassword(user.password);
78              printf("\nConfirm your password:\t");
79              takepassword(password2);
```

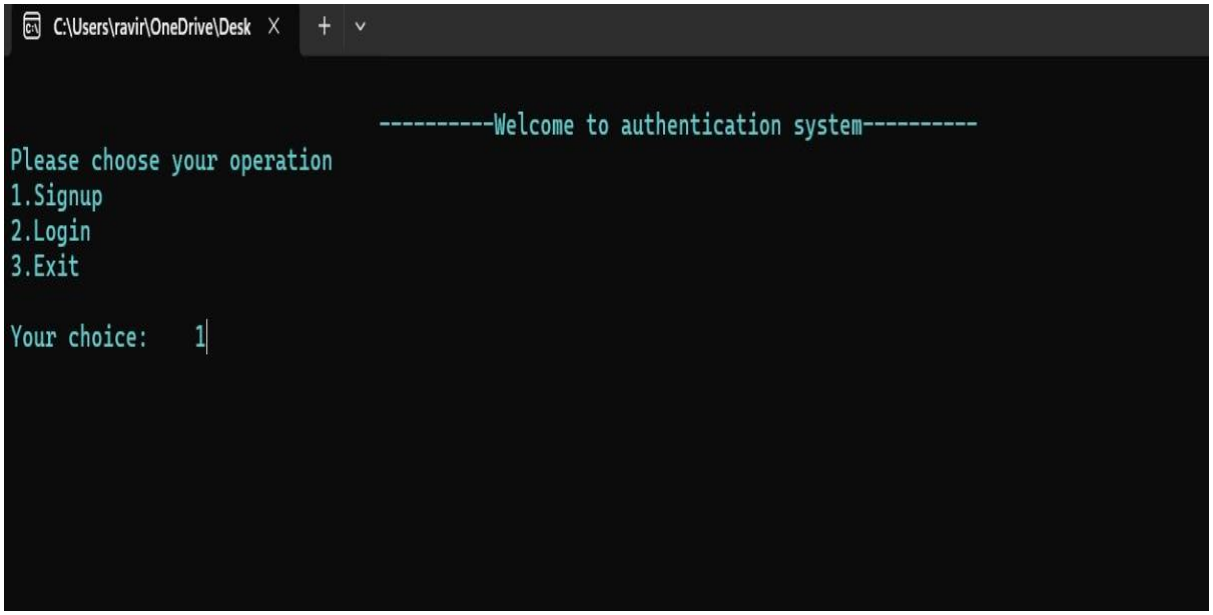
login.cpp

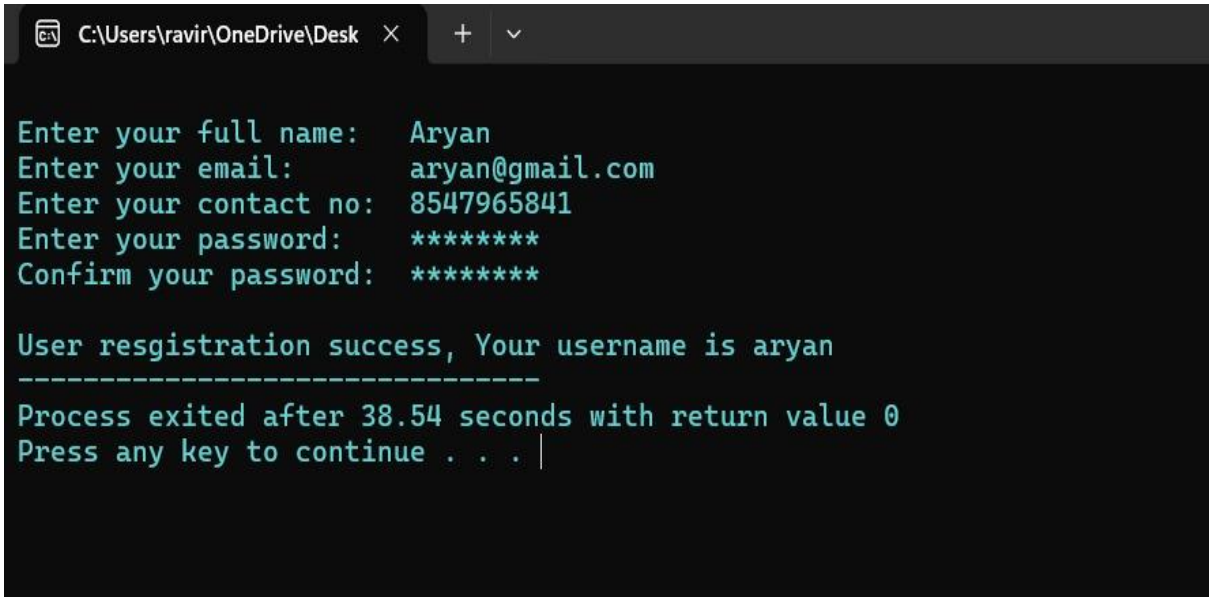
```
79              takepassword(password2);
80
81              if(!strcmp(user.password,password2)){
82                  generateUsername(user.email,user.username);
83                  fp = fopen("Users.dat","a+");
84                  fwrite(&user,sizeof(struct user),1,fp);
85                  if(fwrite != 0) printf("\n\nUser resgistration success, Your username is %s",user.username);
86                  else printf("\n\nSorry! Something went wrong :(");
87                  fclose(fp);
88              }
89              else{
90                  printf("\n\nPassword donot matched");
91                  Beep(750,300);
92              }
93          break;
94
95          case 2:
96
97              char username[50],pword[50];
98              struct user usr;
99
100             printf("\nEnter your username:\t");
101             takeinput(username);
102             printf("Enter your password:\t");
103             takepassword(pword);
104
105             fp = fopen("Users.dat","r");
106             while(fread(&usr,sizeof(struct user),1,fp)){
107                 if(!strcmp(usr.username,username)){
108                     if(!strcmp(usr.password,pword)){
109                         system("cls");
110                         printf("\n\t\t\t\t\t\tWelcome %s",usr.fullName);
```

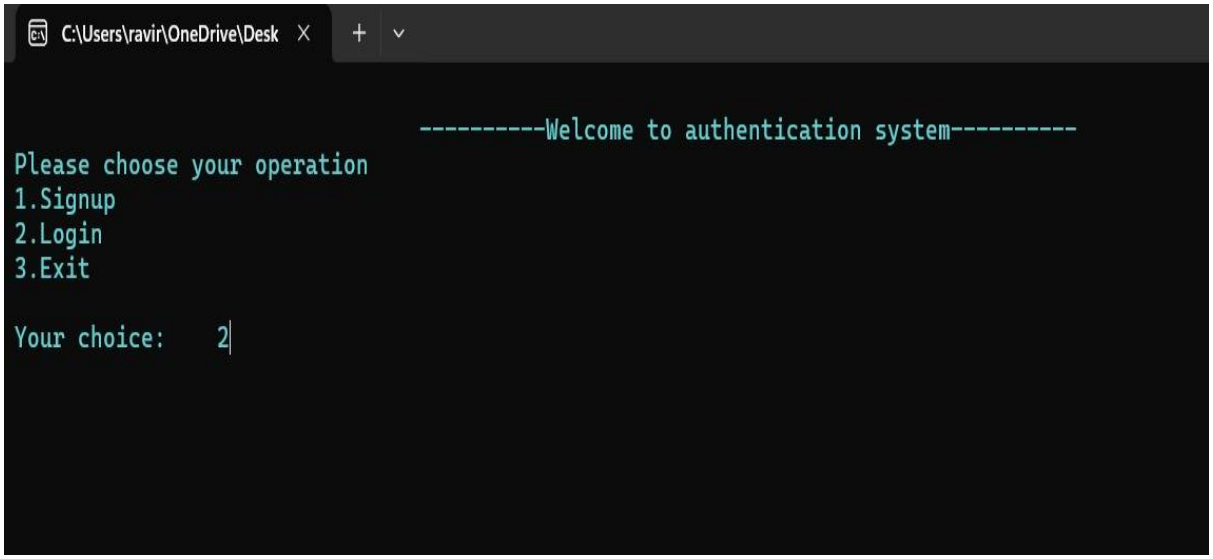
login.cpp

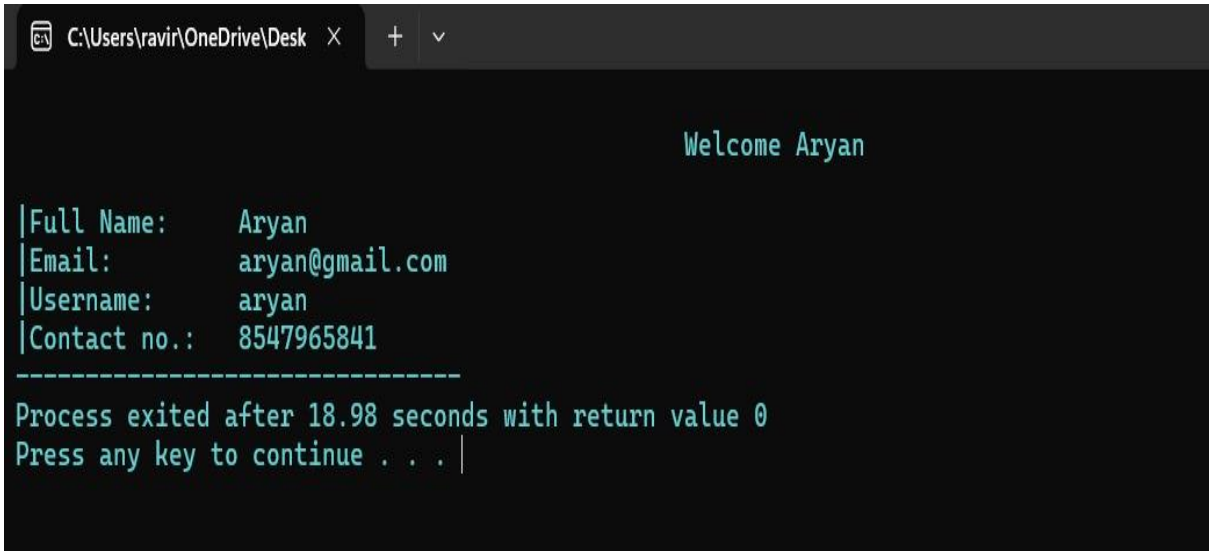
```
108                 if(!strcmp(usr.password,pword)){
109                     system("cls");
110                     printf("\n\t\t\t\t\t\tWelcome %s",usr.fullName);
111                     printf("\n\n|Full Name:\t%s",usr.fullName);
112                     printf("\n|Email:\t\t%s",usr.email);
113                     printf("\n|Username:\t%s",usr.username);
114                     printf("\n|Contact no.:\t%s",usr.phone);
115                 }
116                 else {
117                     printf("\n\nInvalid Password!");
118                     Beep(800,300);
119                 }
120                 usrFound = 1;
121             }
122         }
123         if(!usrFound){
124             printf("\n\nUser is not registered!");
125             Beep(800,300);
126         }
127         fclose(fp);
128         break;
129
130     case 3:
131         printf("\t\t\tBye Bye :)");
132         return 0;
133
134 }
135
136
137
138 return 0;
139 }
```

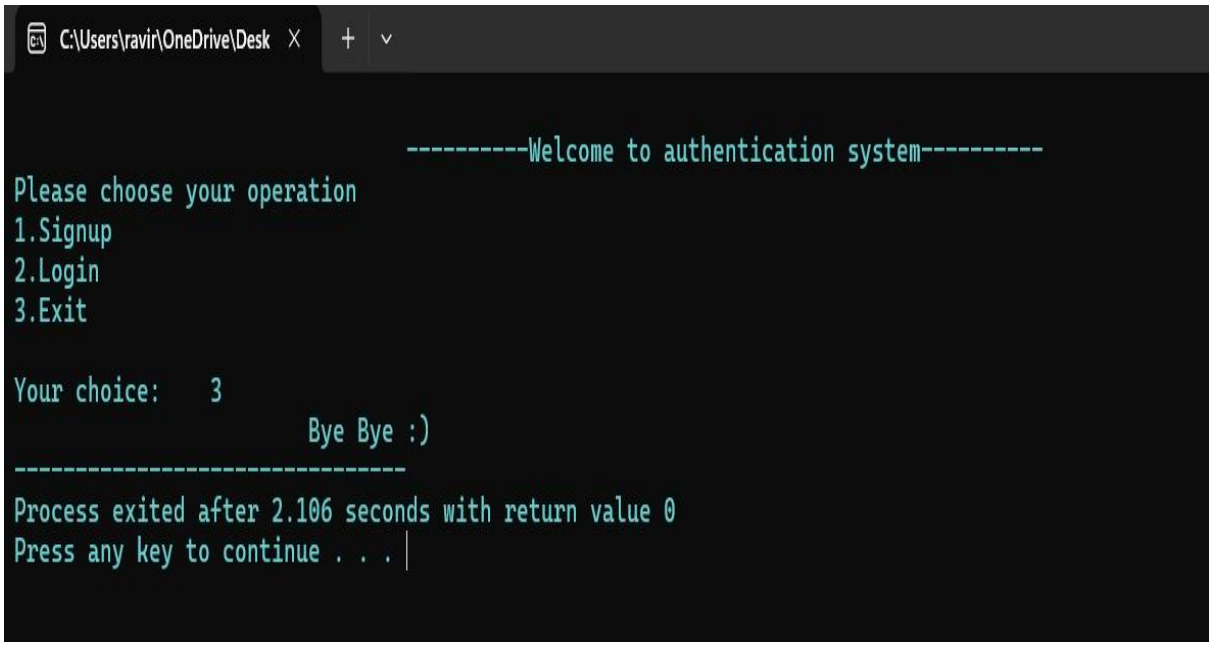
SCREENSHOT OF OUTPUT






C:\Users\ravir\OneDrive\Desk
----------Welcome to authentication system----------
Please choose your operation
1.Signup
2.Login
3.Exit
Your choice: 2


C:\Users\ravir\OneDrive\Desk
Welcome Aryan
|Full Name: Aryan
|Email: aryan@gmail.com
|Username: aryan
|Contact no.: 8547965841
Process exited after 18.98 seconds with return value 0
Press any key to continue . . .


C:\Users\ravir\OneDrive\Desk
----------Welcome to authentication system----------
Please choose your operation
1.Signup
2.Login
3.Exit
Your choice: 3
Bye Bye :)
Process exited after 2.106 seconds with return value 0
Press any key to continue . . .

USES

The login system developed in this project has several potential applications, particularly for educational and small-scale projects. Some possible uses include:

3.1 Desktop Applications:

This program can serve as the foundational authentication module for simple desktop software where user access control is required. For example, it can be implemented in local database management systems, file access systems, or employee management software that requires users to log in before accessing sensitive information.

3.2 Educational Tool:

The login system is an excellent example for teaching basic programming concepts in C. It demonstrates core topics like file handling, string manipulation, and input/output operations, which are crucial for beginners learning the C language. Students can modify and extend the code to add more functionality, such as password recovery or user roles.

3.3 Command-Line Utilities:

For command-line programs where a login mechanism is needed to control access to specific functionalities, this system can be adapted and implemented. It can serve as a secure interface for accessing admin-level commands in a utility program or protecting specific operations in system utilities.

3.4 System-Level Authentication:

This login system can be expanded for use in embedded systems or lightweight operating system utilities where user authentication is required. The simplicity and efficiency of C make it well-suited for systems with limited resources.

3.5 Prototype for Larger Systems:

While this is a basic login system, it can act as a prototype for more complex systems. By adding features like encryption, multi-factor authentication, or database integration, this project can be evolved into a more secure and robust user authentication system for web services, mobile apps, or enterprise applications.