A REPORT

ON

STUDENT RECORD MANAGEMENT SYSTEM

By

Name of the Student	**Registration No.**
T.Luke Alhil	AP24110011278

Prepared in the partial fulfilment of the

Project Based Learning of Course CSE 201 – Coding Skills - I

SRM UNIVERSITY AP

Neeru Konda, Mangalagiri, Guntur

Andhra Pradesh – 522 240

DECEMBER 2025

Acknowledgements

We would like to express our sincere gratitude to our professor, **Trainer: Rakesh Rama Raju P,** for his invaluable guidance, encouragement, and support throughout the development of this project. His expertise and insightful feedback helped us shape our understanding of Coding skills, which were crucial to the successful completion of this Student's Record Management System.

We also extend our thanks to our teammates, whose collaboration, dedication, and teamwork played an essential role in overcoming challenges and refining the project. Their ideas, efforts, and commitment greatly contributed to the overall quality and functionality of the application.

Thank you, **Trainer Rakesh Rama Raju P** and our team, for making this project a rewarding and enriching experience

Table of Contents:

Contents

Abstract

This document details a **Student Record Management System (SRMS)** implemented as a console application in **plain C** using a **procedural programming** approach. The system manages student records, including Roll Number, Name, and Mark, which are persistently stored in a local text file, students.txt. A key feature is the **Role-Based Access Control (RBAC)** model, where user credentials (username, password, role) are dynamically loaded from a separate file, credentials.txt. The application supports three distinct user roles: **Admin**, **Staff**, and **Guest**. The Admin role possesses full **CRUD** (Create, Read, Update, Delete) capabilities. Staff and Guest roles have restricted access, with Staff able to add, display, search, and update, and Guest limited to displaying and searching records. Data manipulation (**Update** and **Delete**) is achieved through a standard C file-handling technique involving the creation of a temporary file, which is then renamed to ensure data integrity and persistence across sessions.

1.Introduction

This Student Record Management System (SRMS) is a C console application designed to demonstrate fundamental **procedural programming** and **file-handling** techniques for managing academic records. The system's primary goal is to provide a functional and secure environment for authorized users to perform data management operations on student records, which consist of a Roll Number, Name, and Mark.

Key Features:

- **Data Persistence:** Student records are saved to and loaded from a local text file (students.txt), ensuring that data is retained between program executions.
- **External Access Control:** User credentials and roles are read from an external file (credentials.txt), enabling flexible role management without code modification.
- **Role-Based Access Control (RBAC):** Functionality is strictly defined by user role upon login:
    - **Admin:** Full read/write access and data persistence control.
    - **Staff:** Read/limited write access (Add, Update, Display, Search).
    - **Guest:** Read-only access (Display, Search).
- **Core Functionality:** The system supports essential CRUD operations: adding new students, searching by roll number, updating records, deleting students, and displaying all records.

The system serves as an effective academic example of combining C's file I/O capabilities and procedural functions into a utility for record keeping.

2. Methodology

1. Data Management (CRUD)

All CRUD operations—Add, Search, Display, Update, and Delete—are performed directly on the students.txt file.

- **Add:** New student records are appended to the end of the file.
- **Search/Display**: The system reads the students.txt file line-by-line to locate and print the requested record or all records.
- **Update/Delete**: These complex operations utilize a temporary file (temp.txt) approach:
    1. The original students.txt is opened for reading, and temp.txt is opened for writing.
    2. The program reads records from the original file one by one.
    3. If a record matches the roll number for deletion, it is skipped.
    4. If a record matches the roll number for updating, the user is prompted for new data, and the new record is written to temp.txt.
    5. All other records are copied unchanged to temp.txt.
    6. The original file is removed, and temp.txt is renamed to students.txt, completing the persistence of the change.

2. Access Control

User access and functionality are determined by their role after a successful login, which allows up to three attempts (MAX_ATTEMPTS).

- **Authentication**: The user's input credentials are checked against the records in the credentials.txt file. The password input is not masked, displaying characters as they are typed.
- **Menu Limitation**: Upon successful login, the system directs the user to an appropriate menu (Admin, Staff, or Guest), which only offers the functions permitted for that specific role.

3. Problem statement

1. Data Management Challenge

The primary challenge is the reliable management of student records (Roll Number, Name, Marks) using procedural C programming and direct file I/O. The system must allow for CRUD operations. Since C lacks robust, built-in dynamic data structures like std::vector, operations like Update and Delete must be managed manually using file recreation. Furthermore, the C implementation does not include validation for unique Roll Numbers, presenting a challenge for data integrity.

2. Security and Access Control Challenge

A key requirement is to restrict user functionality based on their identity to ensure data security. The system needs an authentication mechanism that can read credentials from an external file (credentials.txt). It must differentiate between user roles (Admin, Staff, Guest) and present them with appropriate menus and permissions. A secondary challenge is the limitation of standard C I/O in providing a masked password input feature.

3. Data Persistence Challenge

The system must ensure that student data is not lost when the program terminates. It must implement file I/O to manage two distinct files: one for student data (students.txt) and one for security credentials (credentials.txt), ensuring that changes to student data are reliably saved back to the file using the temporary file swap method.

4. Sample CODE

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define STUD_FILE "students.txt"
#define CRE_FILE  "credentials.txt"
#define MAX_ATTEMPTS 3

char currentUser[50];
char currentRole[20];

int login() {
    char u[50], p[50], r[20];
    char inUser[50], inPass[50];
    int attempt = 0;

    printf("WELCOME TO THE STUDENT MANAGEMENT SYSTEM\n");

    while (attempt < MAX_ATTEMPTS) {
        printf("\n--- Login Attempt %d of %d ---\n", attempt + 1, MAX_ATTEMPTS);
        printf("USERNAME: ");
        scanf("%49s", inUser);
        printf("PASSWORD: ");
        scanf("%49s", inPass);

        FILE *fp = fopen(CRE_FILE, "r");
        if (!fp) {
            printf("Credential file missing! Cannot proceed.\n");
            return 0;
        }

        int loggedIn = 0;
        fseek(fp, 0, SEEK_SET);

        while (fscanf(fp, "%49s %49s %19s", u, p, r) == 3) {
            if (strcmp(inUser, u) == 0 && strcmp(inPass, p) == 0) {
                strncpy(currentUser, u, sizeof(currentUser) - 1);
                currentUser[sizeof(currentUser) - 1] = '\0';
                strncpy(currentRole, r, sizeof(currentRole) - 1);
                currentRole[sizeof(currentRole) - 1] = '\0';
                fclose(fp);
                return 1;
            }
```

```
        }

        fclose(fp);
        printf("Invalid username or password.\n");
        attempt++;
    }

    printf("\nMaximum login attempts exceeded. Exiting.\n");
    return 0;
}

void searchStudent() {
    int find, roll;
    char name[50];
    float mark;

    printf("Enter roll to search: ");
    scanf("%d", &find);

    FILE *fp = fopen(STUD_FILE, "r");
    if (!fp) {
        printf("No student file!\n");
        return;
    }

    int found = 0;
    while (fscanf(fp, "%d %49s %f", &roll, name, &mark) == 3) {
        if (roll == find) {
            printf("\n--- Student Details ---\n");
            printf("Roll: %d\n", roll);
            printf("Name: %s\n", name);
            printf("Mark: %.2f\n", mark);
            printf("-----------------------\n");
            found = 1;
            break;
        }
    }
    fclose(fp);
    if (!found) {
        printf("Student with roll %d not found!\n", find);
    }
}

void addStudent() {
    int roll;
    char name[50];
```

```
    float mark;

    printf("Roll: ");
    if (scanf("%d", &roll) != 1) { return; }
    printf("Name: ");
    scanf(" %49[^\n]", name);
    printf("Mark: ");
    if (scanf("%f", &mark) != 1) { return; }

    FILE *fp = fopen(STUD_FILE, "a");
    fprintf(fp, "%d %s %.2f\n", roll, name, mark);
    fclose(fp);

    printf("Student added!\n");
}

void displayStudents() {
    FILE *fp = fopen(STUD_FILE, "r");
    if (!fp) {
        printf("No student file!\n");
        return;
    }

    int roll;
    char name[50];
    float mark;

    printf("\n--- All Students ---\n");
    printf("Roll\tName\tMark\n");
    printf("----\t----\t----\n");
    while (fscanf(fp, "%d %49s %f", &roll, name, &mark) == 3) {
        printf("%d\t%s\t%.2f\n", roll, name, mark);
    }
    printf("--------------------\n");

    fclose(fp);
}

void deleteStudent() {
    int delRoll;
    printf("Enter roll to delete: ");
    if (scanf("%d", &delRoll) != 1) { printf("Invalid input.\n"); return; }

    FILE *fp = fopen(STUD_FILE, "r");
    FILE *temp = fopen("temp.txt", "w");
    if (!fp || !temp) { return; }
```

```
    int roll;
    char name[50];
    float mark;
    int found = 0;

    while (fscanf(fp, "%d %49s %f", &roll, name, &mark) == 3) {
        if (roll != delRoll) {
            fprintf(temp, "%d %s %.2f\n", roll, name, mark);
        } else {
            found = 1;
        }
    }

    fclose(fp);
    fclose(temp);

    remove(STUD_FILE);
    rename("temp.txt", STUD_FILE);

    if (found) printf("Student deleted!\n");
    else printf("Roll not found!\n");
}

void updateStudent() {
    int updateRoll;
    printf("Enter roll to update: ");
    if (scanf("%d", &updateRoll) != 1) { printf("Invalid input.\n"); return; }

    FILE *fp = fopen(STUD_FILE, "r");
    FILE *temp = fopen("temp.txt", "w");
    if (!fp || !temp) { return; }

    int roll;
   char name[50];
    float mark;
    int found = 0;

    while (fscanf(fp, "%d %49s %f", &roll, name, &mark) == 3) {
        if (roll == updateRoll) {
            found = 1;
            char newName[50];
            float newMark;

            printf("New Name: ");
            scanf(" %49[^\n]", newName);
```

```
            printf("New Mark: ");
            if (scanf("%f", &newMark) != 1) { return; }

            fprintf(temp, "%d %s %.2f\n", roll, newName, newMark);
        } else {
            fprintf(temp, "%d %s %.2f\n", roll, name, mark);
        }
    }

    fclose(fp);
    fclose(temp);

    remove(STUD_FILE);
    rename("temp.txt", STUD_FILE);

    if (found) printf("Student updated!\n");
    else printf("Roll not found!\n");
}

void adminMenu() {
    int c;
    while (1) {
        printf("\nADMIN MENU\n");
        printf("1.Add Student\n2.Display All Students\n3.Search Student by Roll\n4.Update Student\n5.Delete
Student\n6.Logout\n");
        if (scanf("%d",&c) != 1) { c = 0; }

        if(c==1)addStudent();
        else if(c==2)displayStudents();
        else if(c==3)searchStudent();
        else if(c==4)updateStudent();
        else if(c==5)deleteStudent();
        else return;
    }
}

void staffMenu() {
    int c;
    while (1) {
        printf("\nSTAFF MENU\n");
        printf("1.Add Student\n2.Display All Students\n3.Search Student by Roll\n4.Update Student\n5.Logout\n");
        if (scanf("%d",&c) != 1) { c = 0; }

        if(c==1)addStudent();
        else if(c==2)displayStudents();
        else if(c==3)searchStudent();
```

```
        else if(c==4)updateStudent();
        else return;
    }
}

void guestMenu() {
    int c;
    while (1) {
        printf("\nGUEST MENU\n");
        printf("1.Display All Students\n2.Search Student by Roll\n3.Logout\n");
        if (scanf("%d",&c) != 1) { c = 0; }

        if(c==1)displayStudents();
        else if(c==2)searchStudent();
        else return;
    }
}

int main() {
    if (!login()) {
        return 0;
    }
    printf("\nSuccessfully logged in as: %s (%s)\n", currentUser, currentRole);
    if (strcmp(currentRole,"admin")==0) adminMenu();
    else if (strcmp(currentRole,"staff")==0) staffMenu();
    else if (strcmp(currentRole,"guest")==0) guestMenu();
    else printf("Unknown role. Exiting.\n");
    printf("Logged out. Goodbye!\n");
    return 0;
}
```

Functions present in the code:

Authentication and Security Functions

login(): Handles the user authentication process, checks credentials against credentials.txt, and sets the current user's role and username .

main(): The program's entry point; it calls login() and then directs control to the appropriate role menu (adminMenu, staffMenu, or guestMenu) .

Menu and Navigation Functions

adminMenu(): Displays the menu and handles choices for the Admin role, providing access to all CRUD functions (Add, Display, Search, Update, Delete) .

staffMenu(): Displays the menu and handles choices for the Staff role (Add, Display, Search, Update) .

guestMenu(): Displays the menu and handles choices for the Guest role (Display, Search)

Student Data Management (CRUD) Functions

addStudent(): Prompts the user for a Roll Number, Name, and Mark, and appends the new record to the students.txt file .

displayStudents(): Reads and prints all student records from students.txt in a formatted table .

searchStudent(): Prompts for a roll number, then reads the students.txt file sequentially to find and display the details of the matching student record .

updateStudent(): Prompts for a roll number, asks for new name and marks, and modifies the record in the file using the temporary file swap technique (temp.txt) .

deleteStudent(): Prompts for a roll number and removes the corresponding record from the file using the temporary file swap technique (temp.txt)

5.Result(Output)

```
WELCOME TO THE STUDENT MANAGEMENT SYSTEM

--- Login Attempt 1 of 3 ---
USERNAME: admin
PASSWORD: admin123
```

```
WELCOME TO THE STUDENT MANAGEMENT SYSTEM

--- Login Attempt 1 of 3 ---
USERNAME: admin
PASSWORD: admin123

Successfully logged in as: admin (admin)

ADMIN MENU
1.Add Student
2.Display All Students
3.Search Student by Roll
4.Update Student
5.Delete Student
6.Logout
```

```
--- All Students ---
Roll    Name    Mark
----    ----    ----
12      yash    100.00
757     sai     100.00
--------------------

ADMIN MENU
1.Add Student
2.Display All Students
3.Search Student by Roll
4.Update Student
5.Delete Student
6.Logout
```

```
Enter roll to delete: 757
Student deleted!

ADMIN MENU
1.Add Student
2.Display All Students
3.Search Student by Roll
4.Update Student
5.Delete Student
6.Logout
```

s

6. Conclusion

The developed **Student Record Management System (SRMS)** in **plain C** successfully demonstrates the implementation of a console application for record management using a **procedural programming** model. The project effectively showcases robust **file I/O** techniques, utilizing fopen, fprintf, fscanf, remove, and rename to achieve **data persistence**.

The C implementation successfully integrates a **Role-Based Access Control (RBAC)** model, where permissions are strictly enforced through separate menu functions for Admin, Staff, and Guest users. A key technical achievement is the file-based implementation of **Update** and **Delete** operations, which relies on the creation and substitution of a temporary file, a standard and effective method in C for modifying records stored on disk.