A REPORT ON
STUDENT RECORD MANAGEMENT SYSTEM

BY

Name of the Student: K Subramanyam
Registration Number: AP24110011456
Section: CSE-T

Prepared in the partial fulfillment of the

Project Based Learning of Course CSE 201-CODING SKILLS-1

Submitted to:
Mr. P. Rakesh Rama Raju

TABLE OF CONTENT

ACKNOWLEDGMENT

I would like to express my sincere gratitude to everyone who supported me throughout the development of the *Student Record Management System* project.

First and foremost, I extend my heartfelt thanks to my faculty, **Mr. P. Rakesh Rama Raju**, as well as the **Campus Corporate Connect,** for their continuous guidance, encouragement, and valuable feedback. Their support played a significant role in shaping the direction and quality of this project.

I am deeply grateful to **SRMAP University** for providing the necessary resources, learning environment, and technical facilities that enabled the successful completion of this work.

ABSTRACT

The Student Record Management System (SRMS) developed in this project is a menu-driven software application designed to efficiently manage essential student information. The system enables users to perform core operations such as adding new student records, searching for students using their roll numbers, updating existing details, deleting unwanted entries, and viewing all stored data in an organized format. By incorporating structured data storage and file-handling mechanisms, the system ensures accuracy, consistency, and easy retrieval of student information.

This SRMS eliminates the limitations of manual record-keeping by automating data management tasks and reducing human error. The clear and interactive interface allows users to access and modify records quickly, making it suitable for academic administration and small educational environments. The project demonstrates effective implementation of key programming concepts such as data structures, file operations, modular design, and validation techniques.

Overall, the Student Record Management System provides a reliable, user-friendly, and scalable solution for managing student data, and it forms a strong foundation for future enhancements such as database integration, security features, and graphical user interfaces.

INTRODUCTION

Educational institutions deal with vast and continuously growing volumes of student data, such as personal information, attendance, academic records, and performance metrics. Managing these records manually can lead to inefficiencies including data duplication, difficulty in retrieval, slower processing times, and increased chances of human error. In today's digital era, institutions strive for accuracy, organization, and speed. There is a growing need for automated systems that can handle academic data effectively.

The **Student Report Management System** is designed to digitalize and streamline the process of managing student academic records. The system automates key functions such as adding student details, entering subject-wise marks, calculating total scores and grades, and generating structured performance reports. By replacing traditional manual methods with a computerized system, SRMS enhances data reliability, improves operational efficiency, and ensures the secure management of student information.

This system provides an easy-to-use interface for teachers and administrators, enabling them to update and access academic data in real time. The integrated database ensures that all information is stored centrally, making retrieval quick and hassle-free. It also allows for scalability, enabling the addition of new modules such as attendance tracking, notifications, or analytics in the future.

Moreover, the SRMS contributes significantly to academic transparency by ensuring that performance reports are consistent, accurate, and readily available. It serves as a valuable tool for institutions aiming to adopt modern educational technologies and improve their administrative workflows. Ultimately, the project demonstrates how automation can transform academic recordkeeping into a streamlined, reliable, and technologically advanced process.

PROBLEM STATEMENT

Managing student academic reports manually leads to several issues such as:

- Time-consuming data entry and retrieval
- Increased chances of human errors
- Difficulty in organizing and updating records
- Delays in report generation
- Lack of centralized and secure data storage

There is a need for a system that automates and simplifies academic data management, ensuring accuracy, speed, and security.

OBJECTIVES

- To create a system for storing student information and academic records.
- To automate the process of marks entry and report generation.
- To reduce manual errors and improve data accuracy.
- To provide quick search and retrieval of student records.
- To generate detailed academic reports efficiently.
- To ensure secure and reliable data handling.

PROJECT LAYOUT

System Architecture Overview

1. **User Interface Layer** – Allows admin/teacher interaction with the system (forms, dashboards).
2. **Application Layer** – Business logic for storing and processing data (students, marks, reports).
3. **Database Layer** – Stores student details, subjects, marks, and report data.

```
+---------------------------+
|      User Interface       |
+-------------+-------------+
              |
+-------------v-------------+
|     Application Logic     |
+-------------+-------------+
              |
+-------------v-------------+
|      Database System      |
+---------------------------+
```

MODULES EXPLANATION

1. Login Module

- Allows authorized users (admin/teachers) to access the system.
- Ensures security and role-based access.

2. Student Management Module

- Add, update, view, and delete student information.
- Stores personal and academic details.

3. Subject Management Module

- Add and manage subjects for different classes/semesters.

4. Marks Entry Module

- Teachers can enter marks for each student.
- Ensure validation of numeric inputs.

5. Report Generation Module

- Automatically calculates results, grades, and percentages.
- Generates printable student report cards.

6. Database Management Module

- Handles data storage, updates, backup, and retrieval.

FEATURES IMPLEMENTED

- User-Friendly Dashboard
- Student Registration & Profile Management
- Subject, Examination & Marks Management
- Automatic Result Generation
- Report Card Generation
- Secure Login & Role-Based Access

DATA FLOW DIAGRAM (DFD)

Level 0 DFD (Context Diagram)

```
        +---------------------+
         |     Teacher/Admin  |
         +---------+-----------+
                   |
                   v
      +-----------------------------+
      | Student Report Management   |
      |           System          |
      +-------------+---------------+
                   |
                   v
         +----------------------+
         |    Database        |
         +----------------------+
```

Level 1 DFD

```
+-----------------+       +------------------+
|  Student Module |<-----> |   Database      |
+-----------------+       +------------------+
          |
          v
+-----------------+       +------------------+
|  Marks Module   |<-----> |   Database      |
+-----------------+       +------------------+
          |
          v
+-----------------+
| Report Module   |
+-----------------+
```

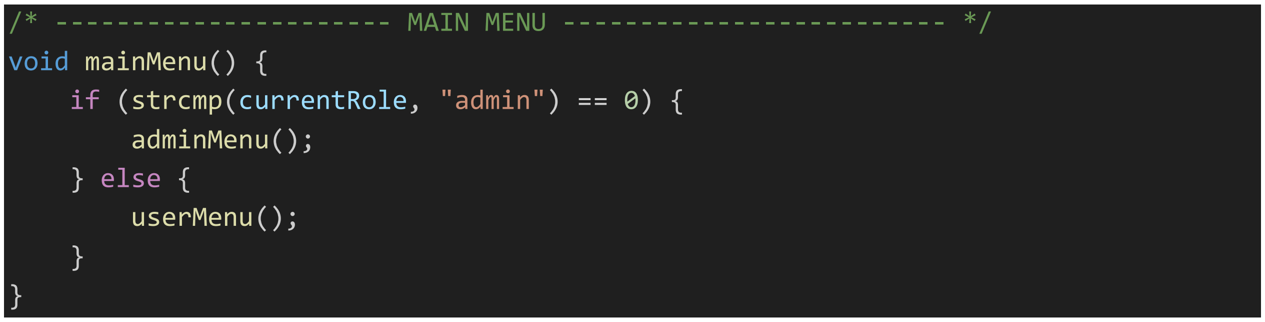
CODE EXPLANATION

1. main()

Purpose: Starting point of the program.

What it does:

- Calls loginSystem() to authenticate the user.
- If login is successful → calls mainMenu()
- Otherwise → prints "Access Denied".



```
/* --------------------- MAIN MENU ------------------------ */
void mainMenu() {
    if (strcmp(currentRole, "admin") == 0) {
        adminMenu();
    } else {
        userMenu();
    }
}
```

2. loginSystem()

Purpose: Handles login authentication.

How it works:

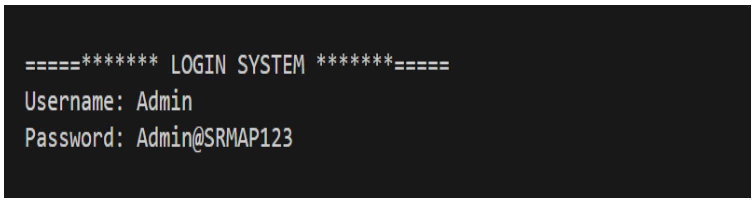
- Opens credentials.txt
- Asks user for username & password
- Reads stored username, password, and role from the file
- If match found:
    - Stores username in currentUser
    - Stores role (admin/user) in currentRole
    - Returns 1 (success)
- If no match → returns 0

This function decides whether someone can access the system and what permissions they have.

```c
char user[50], pass[50], fileUser[50], filePass[50], fileRole[10];

    printf("\n=====******* LOGIN SYSTEM *******=====\n");
    printf("Username: ");
    scanf("%s", user);
    printf("Password: ");
    scanf("%s", pass);

    while (fscanf(fp, "%s %s %s", fileUser, filePass, fileRole) != EOF) {
        if (strcmp(user, fileUser) == 0 && strcmp(pass, filePass) == 0) {
            strcpy(currentUser, fileUser);
            strcpy(currentRole, fileRole);
            fclose(fp);
            return 1;
        }
    }
```



```
=====******* LOGIN SYSTEM *******=====
Username: Admin
Password: Admin@SRMAP123
```

3. mainMenu()

Purpose: Decides which menu to show.

How it works:

- If role is "admin" → shows **adminMenu()**
- Else → shows **userMenu()**

This function acts as a router between admin and normal user modes.

```c
void mainMenu() {
    if (strcmp(currentRole, "admin") == 0) {
        adminMenu();
    } else {
        userMenu();
    }
}
```

4. adminMenu()

Purpose: Contains options available only to admin.

Options include:

1. Add Student
2. View Students
3. Search Student
4. Update Student
5. Delete Student

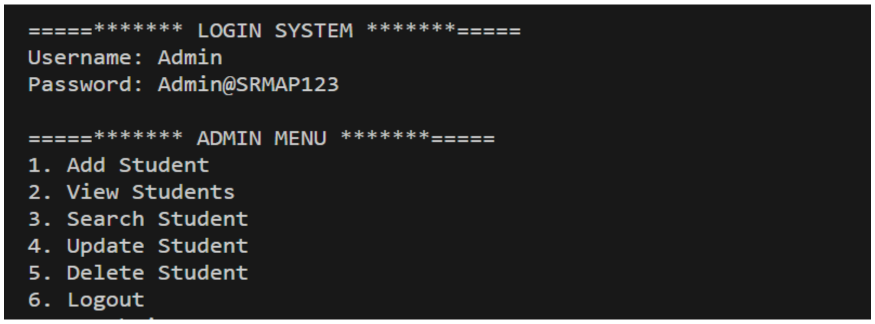
How it works:

- Takes admin input
- Calls corresponding functions

Admins can perform all CRUD operations (Create, Read, Update, Delete)

```
printf("\n=====******* ADMIN MENU *******=====\n");
        printf("1. Add Student\n");
        printf("2. View Students\n");
        printf("3. Search Student\n");
        printf("4. Update Student\n");
        printf("5. Delete Student\n");
        printf("6. Logout\n");
        printf("Enter choice: ");
        scanf("%d", &ch);

        switch (ch) {
            case 1: addStudent(); break;
            case 2: viewStudents(); break;
            case 3: searchStudent(); break;
            case 4: updateStudent(); break;
            case 5: deleteStudent(); break;
            case 6: return;
            default: printf("Invalid choice!\n");
```


```
=====******* LOGIN SYSTEM *******=====
Username: Admin
Password: Admin@SRMAP123

=====******* ADMIN MENU *******=====
1. Add Student
2. View Students
3. Search Student
4. Update Student
5. Delete Student
6. Logout
```

5. userMenu()

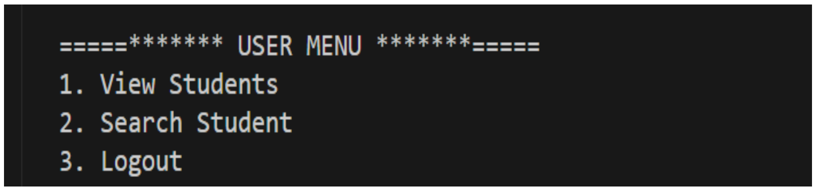
Purpose: Shows features available for normal users.

Options include:

1. View Students
2. Search Student

Users can only read data, not modify it.

```
case 1: viewStudents(); break;
            case 2: searchStudent(); break;
            case 3: return;
            default: printf("Invalid choice!\n");
```


```
=====******* USER MENU *******=====
1. View Students
2. Search Student
3. Logout
```

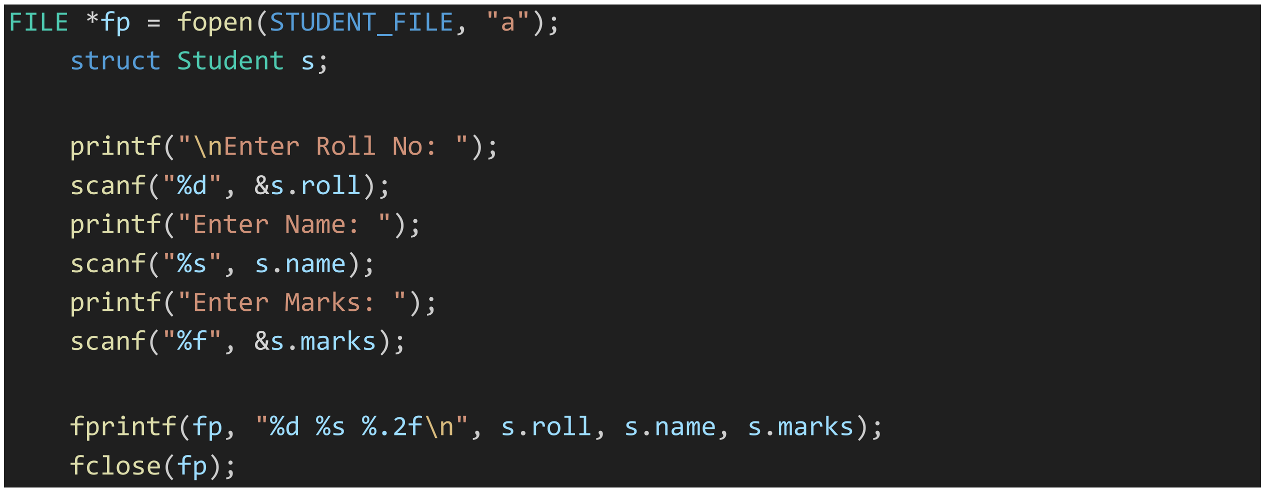
6. addStudent()

Purpose: Adds a new student record.

How it works:

- Opens students.txt in append mode
- Takes roll number, name, and marks from user
- Writes record to the file

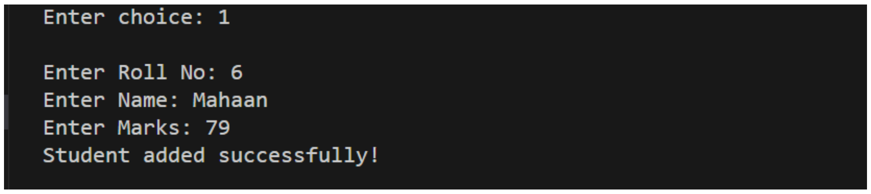
Used only by admins to insert new students.



```c
FILE *fp = fopen(STUDENT_FILE, "a");
    struct Student s;

    printf("\nEnter Roll No: ");
    scanf("%d", &s.roll);
    printf("Enter Name: ");
    scanf("%s", s.name);
    printf("Enter Marks: ");
    scanf("%f", &s.marks);

    fprintf(fp, "%d %s %.2f\n", s.roll, s.name, s.marks);
    fclose(fp);
```



```
Enter choice: 1

Enter Roll No: 6
Enter Name: Mahaan
Enter Marks: 79
Student added successfully!
```

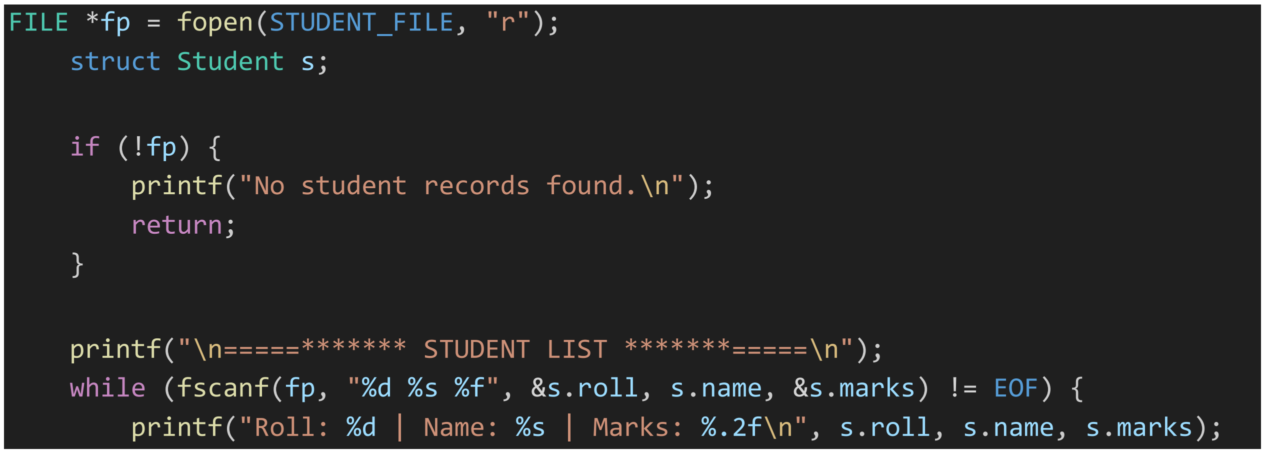
7. viewStudents()

Purpose: Displays all student records.

How it works:

- Opens students.txt in read mode
- Reads each student using fscanf
- Prints Roll, Name, Marks

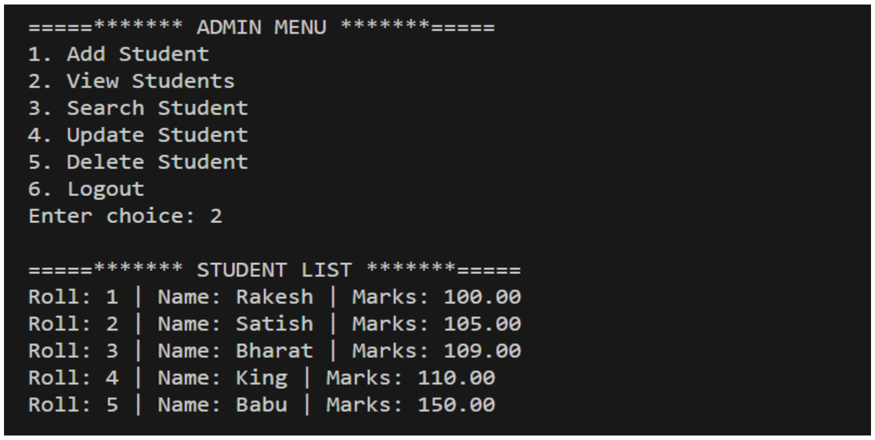
Both admin and users can view all records.



```c
FILE *fp = fopen(STUDENT_FILE, "r");
    struct Student s;

    if (!fp) {
        printf("No student records found.\n");
        return;
    }

    printf("\n=====******* STUDENT LIST *******=====\n");
    while (fscanf(fp, "%d %s %f", &s.roll, s.name, &s.marks) != EOF) {
        printf("Roll: %d | Name: %s | Marks: %.2f\n", s.roll, s.name, s.marks);
    }
```



```
=====******* ADMIN MENU *******=====
1. Add Student
2. View Students
3. Search Student
4. Update Student
5. Delete Student
6. Logout
Enter choice: 2

=====******* STUDENT LIST *******=====
Roll: 1 | Name: Rakesh | Marks: 100.00
Roll: 2 | Name: Satish | Marks: 105.00
Roll: 3 | Name: Bharat | Marks: 109.00
Roll: 4 | Name: King | Marks: 110.00
Roll: 5 | Name: Babu | Marks: 150.00
```

8. searchStudent()

Purpose: Searches for a student using roll number.

How it works:

- Asks for roll number
- Reads every record from students.txt
- If roll matches → prints details
- Else → shows "Record not found"

Useful for quickly finding a specific student.

```c
void searchStudent() {
    FILE *fp = fopen(STUDENT_FILE, "r");
    int r, found = 0;
    struct Student s;

    printf("\nEnter Roll Number to Search: ");
    scanf("%d", &r);

    while (fscanf(fp, "%d %s %f", &s.roll, s.name, &s.marks) != EOF) {
        if (s.roll == r) {
            printf("\nRecord Found!\n");
            printf("Roll: %d | Name: %s | Marks: %.2f\n", s.roll, s.name,
s.marks);
            found = 1;
            break;
        }
    }
```

```
Enter choice: 3

Enter Roll Number to Search: 4

Record Found!
Roll: 4 | Name: King | Marks: 110.00
```

9. updateStudent()

Purpose: Updates an existing student record.

How it works:

- Opens original file for reading (students.txt)
- Opens a temporary file (temp.txt) for writing
- Copies all records:
  - If roll matches → asks for new name & marks, writes updated record
  - Otherwise → writes original record
- Replaces original file with updated file (using remove & rename)

This is a common method to update records in text files.

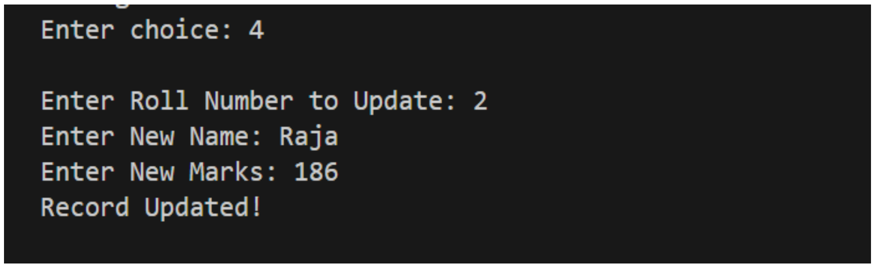
```c
void updateStudent() {
    FILE *fp = fopen(STUDENT_FILE, "r");
    FILE *tmp = fopen("temp.txt", "w");

    int r, found = 0;
    struct Student s;

    printf("\nEnter Roll Number to Update: ");
    scanf("%d", &r);

    while (fscanf(fp, "%d %s %f", &s.roll, s.name, &s.marks) != EOF) {
        if (s.roll == r) {
            found = 1;
            printf("Enter New Name: ");
            scanf("%s", s.name);
            printf("Enter New Marks: ");
            scanf("%f", &s.marks);
        }
        fprintf(tmp, "%d %s %.2f\n", s.roll, s.name, s.marks);
    }


 fclose(fp);
    fclose(tmp);
    remove(STUDENT_FILE);
    rename("temp.txt", STUDENT_FILE);
    if (found)
        printf("Record Updated!\n");
    else
        printf("Record not found.\n");
}
```



```
Enter choice: 4

Enter Roll Number to Update: 2
Enter New Name: Raja
Enter New Marks: 186
Record Updated!
```

10. deleteStudent()

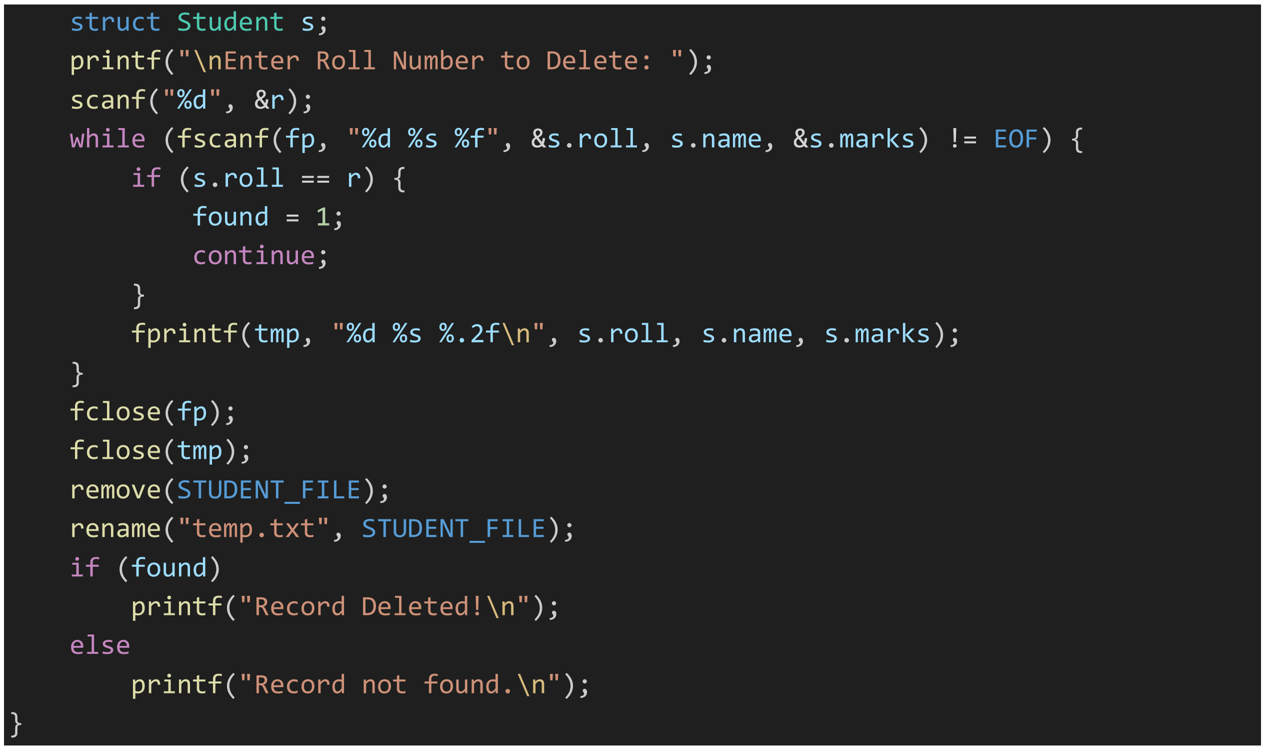
Purpose: Deletes a student record.

How it works:

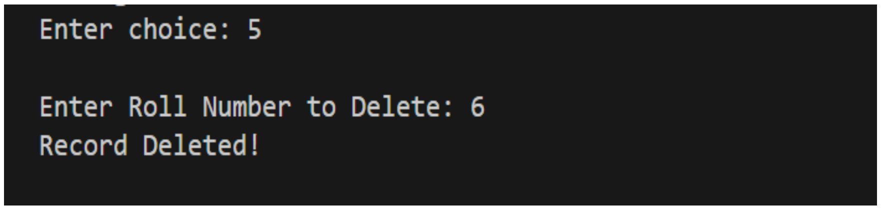
- Opens original file and temp file
- Reads each record:
    - If roll matches → skip writing (deleting)
    - Otherwise → write record to temp file
- Replace old file with temp file

Deletes by recreating file without the selected student.

```
void deleteStudent() {
    FILE *fp = fopen(STUDENT_FILE, "r");
    FILE *tmp = fopen("temp.txt", "w");
    int r, found = 0;
```



```
    struct Student s;
    printf("\nEnter Roll Number to Delete: ");
    scanf("%d", &r);
    while (fscanf(fp, "%d %s %f", &s.roll, s.name, &s.marks) != EOF) {
        if (s.roll == r) {
            found = 1;
            continue;
        }
        fprintf(tmp, "%d %s %.2f\n", s.roll, s.name, s.marks);
    }
    fclose(fp);
    fclose(tmp);
    remove(STUDENT_FILE);
    rename("temp.txt", STUDENT_FILE);
    if (found)
        printf("Record Deleted!\n");
    else
        printf("Record not found.\n");
}
```



```
Enter choice: 5

Enter Roll Number to Delete: 6
Record Deleted!
```

.txt files used in the project
Credentials.txt:
Admin Admin@SRMAP123 admin
Student student@SRMAP123 user
Guest Guest123 guest

Students.txt:
1 Rakesh 100.00
2 Satish 105.00
3 Bharat 109
4 King 110
5 Babu 150
6 Mahaan 79.00

Note: Students txt files changes everytime you update or add students in the project.

CONCLUSION

The **Student Record Management System (SRMS)** successfully demonstrates how digital automation can significantly improve the efficiency and accuracy of academic data management within educational institutions. By replacing traditional manual methods, the system provides a streamlined approach to handling student records, marks, and performance reports. Throughout the development of this project, various software engineering principles, database management techniques, and user-interface design strategies were applied to create a system that is both functional and user-friendly.

The system's modular architecture ensures that each component—student registration, marks entry, subject and record generation—functions cohesively while remaining easy to modify or expand. This modularity also supports future scalability, making it possible to integrate new features as institutional needs evolve. The automated computation of grades and report generation reduces the likelihood of human errors and saves valuable time for teachers and administrators. Moreover, the centralized database ensures secure storage, quick retrieval, and easy maintenance of academic records.

Implementing the SRMS enhances overall academic transparency and provides more reliable insights into student performance, which can help institutions make data-driven decisions. The system's emphasis on user experience ensures that even users with minimal technical background can operate it efficiently. Additionally, the project highlights the importance of digital transformation in education and showcases the benefits of adopting modern software solutions for administrative processes.

In conclusion, the Student Record Management System fulfills its goal of simplifying, organizing, and modernizing academic record management. It stands as a practical, scalable, and efficient solution for educational institutions seeking to enhance their administrative capabilities and move toward a more technology-driven environment.

FUTURE ENHANCEMENTS

- Integration of SMS/Email alerts for parents.
- Online student/parent portal for viewing results.
- Mobile application for quick access.
- Advanced analytics for student performance tracking.
- Cloud-based storage and multi-campus support.
- More advanced validation (e.g., duplicate roll numbers, incorrect formats) and robust error handling can improve reliability.
- Automatic backup features ensure that records are not lost due to accidental deletion or file corruption.
- The system can generate PDF or Excel reports for attendance, marks, student lists, or performance analysis.

REFERENCES

- Online documentation of programming languages used c c++.
- Database reference manuals (MySQL/PostgreSQL).
- Educational software requirement standards.
- Research papers on academic management systems.