A PROJECT REPORT ON STUDENT RECORD MANAGEMENT SYSTEM (SRMS)

DONE BY

NAME: M. JITHIN VENKATA SAI

REGISTRATION NUMBER: AP24110011650

CSE 2ND YEAR THIRD SEMESTER

SECTION V

SUBMITTED TO

RAKESH RAMA RAJU

CODING SKILLS – I (CSE 201)

ABSTRACT

This project presents a Student Record Management System developed in the C programming language to offer a reliable, secure and easy-to-use solution for managing student information. The system stores key details such as roll number, name and marks in text files, ensuring that all data is safely preserved even after the program closes. It provides the essential operations required for effective record management, including adding new entries, viewing stored data, searching for specific students, updating information and deleting records when necessary.

A role-based login system forms the foundation of its security and usability. Users can log in as Admin, Staff or Guest, and each role is granted only the level of access needed for its responsibilities. Admin users can perform all operations, staff users can view and search records and guest users are limited to viewing information. This structured approach not only protects the system from unauthorized modifications but also makes the interface simple and user-friendly for every type of user.

Overall, this project demonstrates the practical importance of core C programming concepts while showing how a straightforward application can effectively represent real-world data management. It stands as a highly valuable learning tool that combines simplicity, security and smooth usability to create a dependable student information management system.

INTRODUCTION

The **Student Record Management System** is a C-based application developed to efficiently manage and maintain student information using basic file-handling techniques. The system stores essential details such as roll number, name, and marks in external text files, ensuring that the data remains preserved even after the program is terminated. To ensure secure and organized access, the application incorporates a role-based login mechanism that restricts operations based on user privileges.

The system supports three distinct user roles: **Admin, Staff, and Guest**. Admin users possess complete access and can add, update, delete, search, and view student records. Staff members are granted permissions to search and view existing records, whereas Guest users are limited to viewing the data. This layered permission structure promotes data integrity and prevents unauthorized modifications.

In addition to managing student data, the project demonstrates the practical application of key concepts in the C programming language, including structures, file operations, string handling, and conditional logic. Overall, the Student Management System serves as a foundational example of how real-world data management applications can be designed and implemented using fundamental programming principles.

OBJECTIVES

The main objectives of this project are:

- To create an easy and efficient system for managing student information using C programming.
- To add a secure login system with different roles for Admin, Staff and Guest users.
- To give each user type the right level of access so the system is used properly.
- To store student details in text files so the data can be saved for a long time and accessed easily.
- To allow Admin users to add, view, search, update and delete student records.
- To allow Staff users to only view and search student information without making any changes.
- To give Guest users read-only access to keep the data safe.
- To show the practical use of file handling, structures and conditional statements in C.
- To protect the accuracy of the data by preventing unauthorized changes.
- To provide a simple, user-friendly, menu-based interface for smooth and easy navigation.

MODULES

The system consists of the following major modules:

1. Login Module

- Takes username and password.
- Checks credentials from credentials.txt.
- Identifies role: Admin, Staff, or Guest.

2. Role-Based Menu Module

- Shows menu according to user role.
- Controls access to operations for Admin, Staff, and Guest.

3. Student Addition Module (Admin Only)

- Allows adding new student details.
- Saves roll number, name, and marks in students.txt.

4. Display Students Module

- Reads all records from students.txt.
- Displays roll number, name, and marks.

5. Search Student Module

- Searches for a student by roll number.
- Shows matching record.

6. Update Student Module

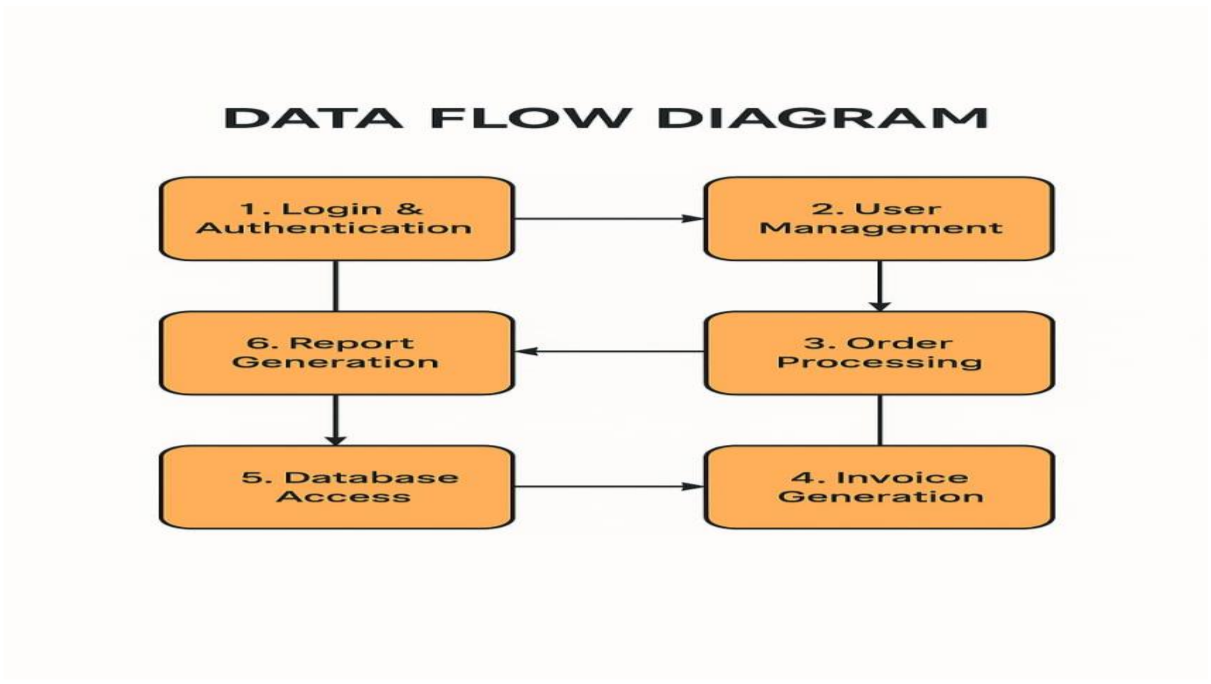
- Updates student name or marks.
- Uses a temporary file to save updated data.

7. Delete Student Module (Admin Only)

- Deletes a student record using roll number.
- Rewrites file without the deleted entry.

8. File Handling Module

- Handles reading and writing of credentials.txt and students.txt.

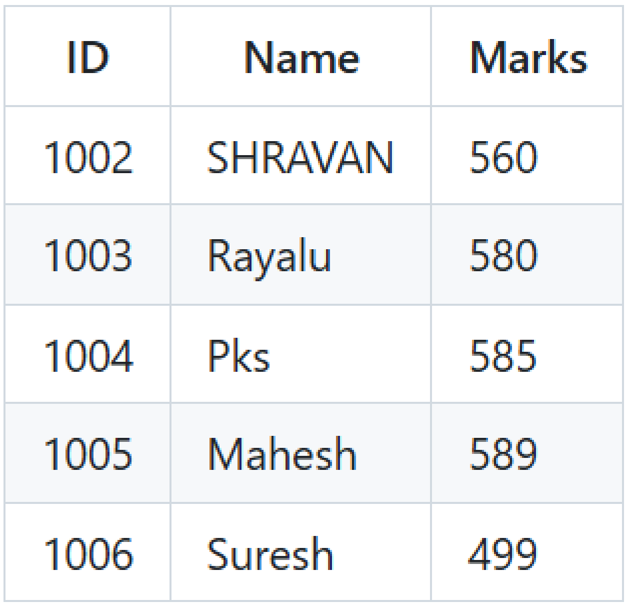


Explanation

1. Login & Authentication:
   The system checks the entered username and password with *credentials.txt* and identifies the user role.
2. Role-Based Menu:
   Based on the role (Admin, Staff, Guest), the appropriate menu is shown and the chosen option is passed to the next process.
3. Student Management:
   Handles adding, viewing, searching, updating and deleting student records. Updates and deletions use a temporary file for safe rewriting.
4. File Handling:
   All student data is stored in ***students.txt***, ensuring permanent storage and reliable read/write operations.

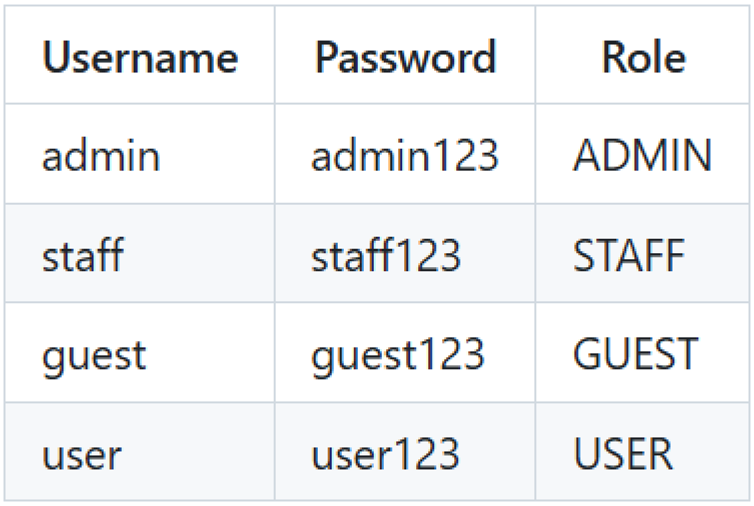
CODE & OUTPUTS

- **students.txt**



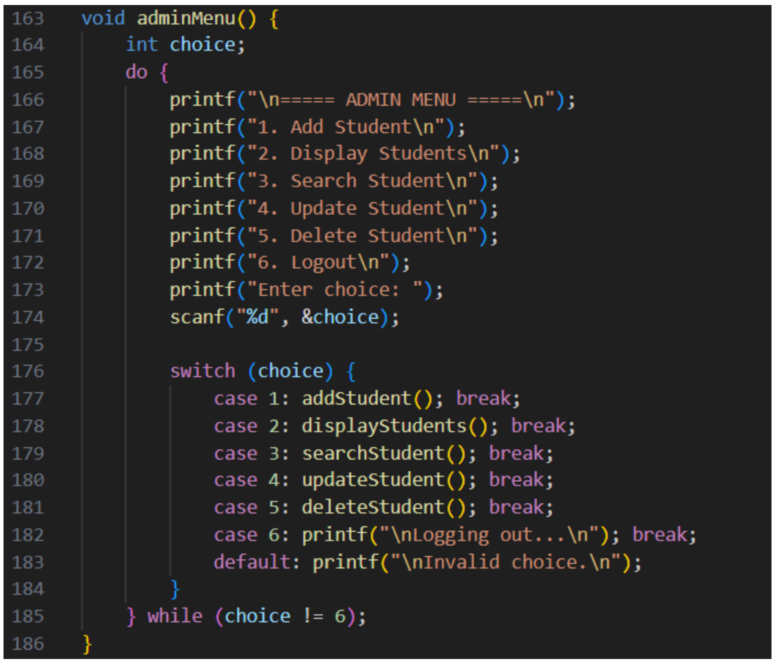
ID	Name	Marks
1002	SHRAVAN	560
1003	Rayalu	580
1004	Pks	585
1005	Mahesh	589
1006	Suresh	499

- **Credentials.txt**

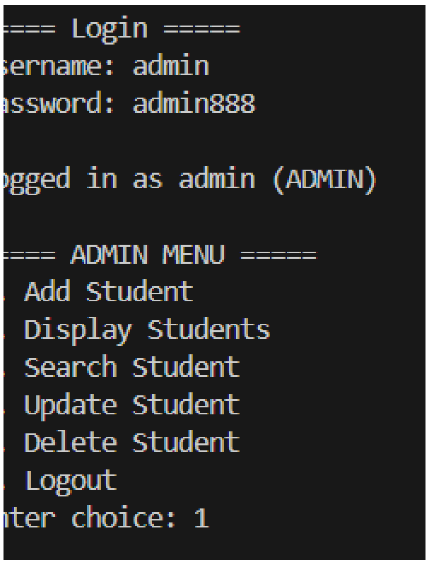


Username	Password	Role
admin	admin123	ADMIN
staff	staff123	STAFF
guest	guest123	GUEST
user	user123	USER

➢ LOGIN & ADMIN MENU



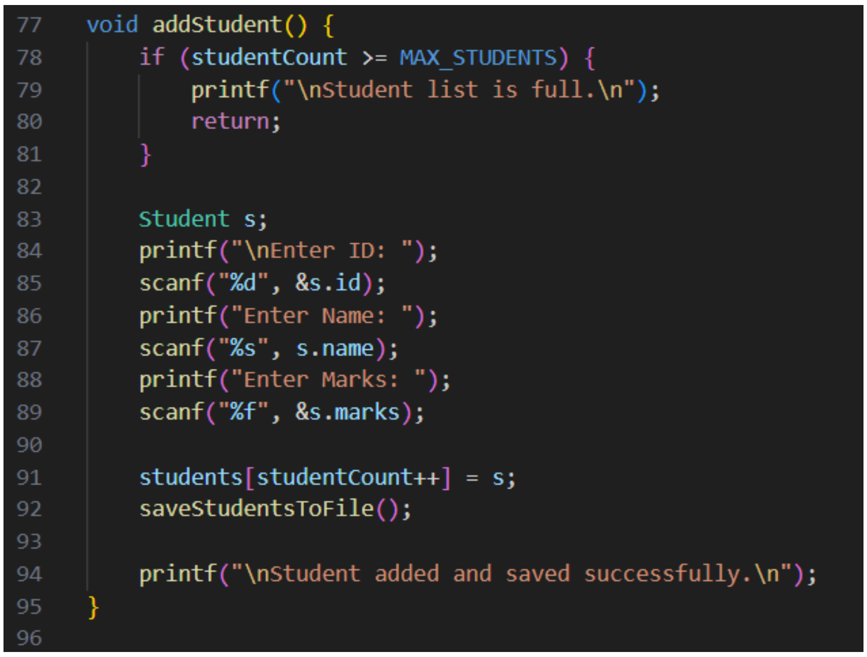
```c
void adminMenu() {
    int choice;
    do {
        printf("\n===== ADMIN MENU =====\n");
        printf("1. Add Student\n");
        printf("2. Display Students\n");
        printf("3. Search Student\n");
        printf("4. Update Student\n");
        printf("5. Delete Student\n");
        printf("6. Logout\n");
        printf("Enter choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1: addStudent(); break;
            case 2: displayStudents(); break;
            case 3: searchStudent(); break;
            case 4: updateStudent(); break;
            case 5: deleteStudent(); break;
            case 6: printf("\nLogging out...\n"); break;
            default: printf("\nInvalid choice.\n");
        }
    } while (choice != 6);
}
```



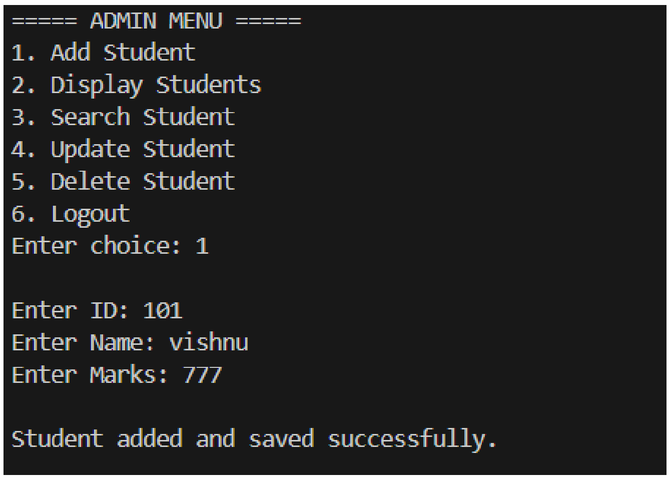
```
=== Login =====
ername: admin
ssword: admin888

gged in as admin (ADMIN)

=== ADMIN MENU =====
 Add Student
 Display Students
 Search Student
 Update Student
 Delete Student
 Logout
ter choice: 1
```

After login, the Admin Menu appears, allowing the admin to add, view, search, update, or delete student records.

➢ ADD STUDENT



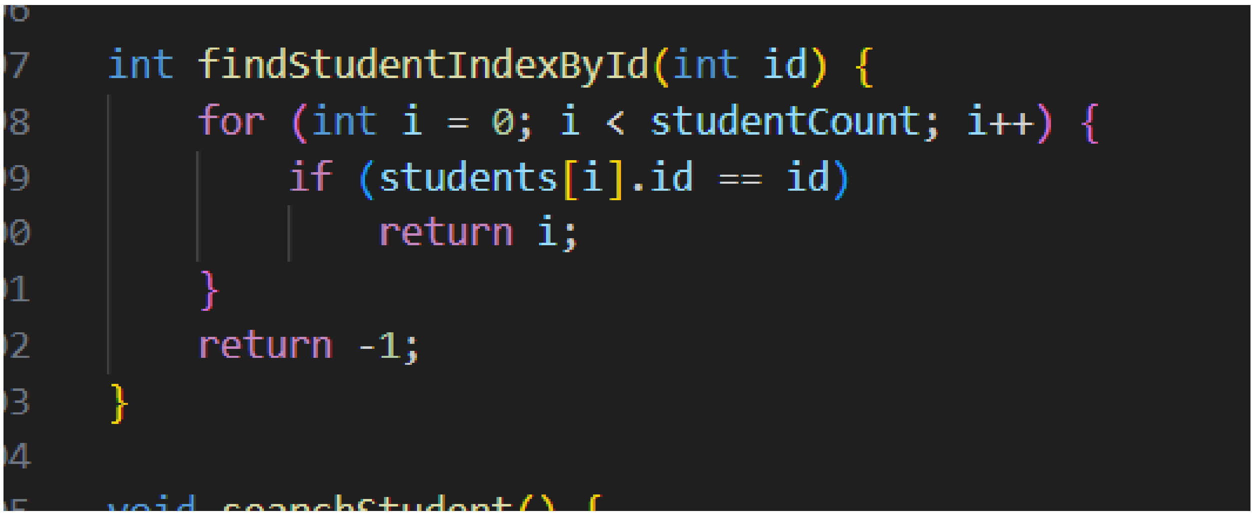
```c
void addStudent() {
    if (studentCount >= MAX_STUDENTS) {
        printf("\nStudent list is full.\n");
        return;
    }

    Student s;
    printf("\nEnter ID: ");
    scanf("%d", &s.id);
    printf("Enter Name: ");
    scanf("%s", s.name);
    printf("Enter Marks: ");
    scanf("%f", &s.marks);

    students[studentCount++] = s;
    saveStudentsToFile();

    printf("\nStudent added and saved successfully.\n");
}
```



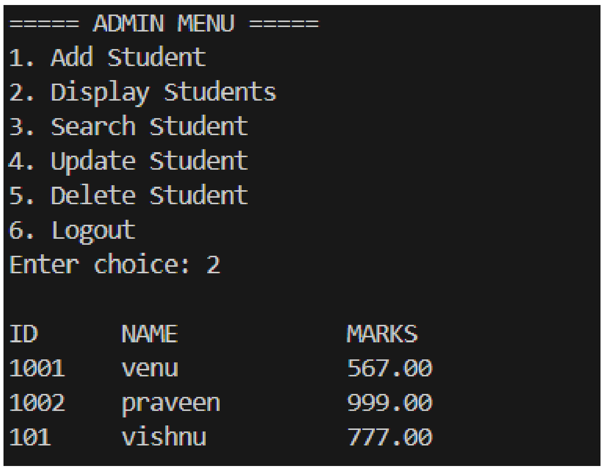
```
===== ADMIN MENU =====
1. Add Student
2. Display Students
3. Search Student
4. Update Student
5. Delete Student
6. Logout
Enter choice: 1

Enter ID: 101
Enter Name: vishnu
Enter Marks: 777

Student added and saved successfully.
```

This screen allows the admin to add a new student record by entering the roll number, name, and marks. After submission, the record is successfully stored in the system.

➢ DISPLAY STUDENT



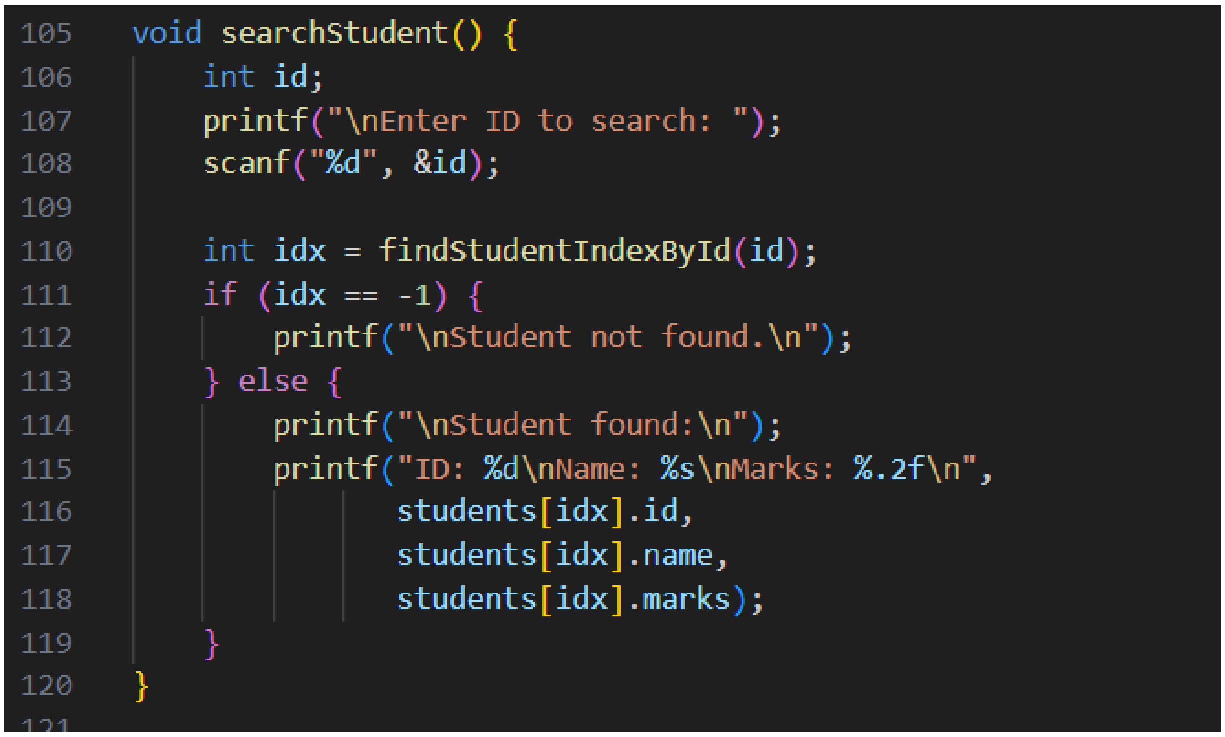
```c
int findStudentIndexById(int id) {
    for (int i = 0; i < studentCount; i++) {
        if (students[i].id == id)
            return i;
    }
    return -1;
}
```



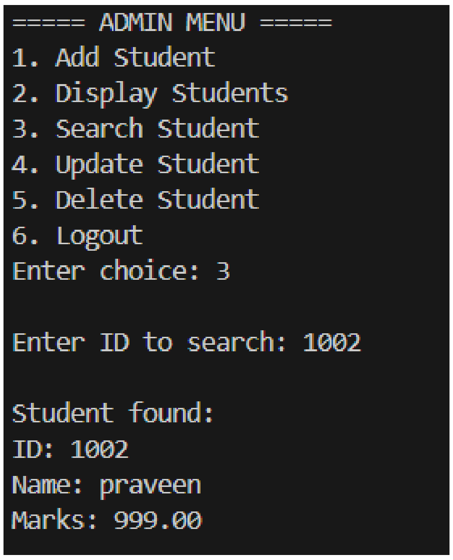
```
===== ADMIN MENU =====
1. Add Student
2. Display Students
3. Search Student
4. Update Student
5. Delete Student
6. Logout
Enter choice: 2

ID      NAME            MARKS
1001    venu            567.00
1002    praveen         999.00
101     vishnu          777.00
```

This output shows the list of all stored student records. It displays roll numbers, names, and marks in a structured table format.

➢ SEARCH STUDENT



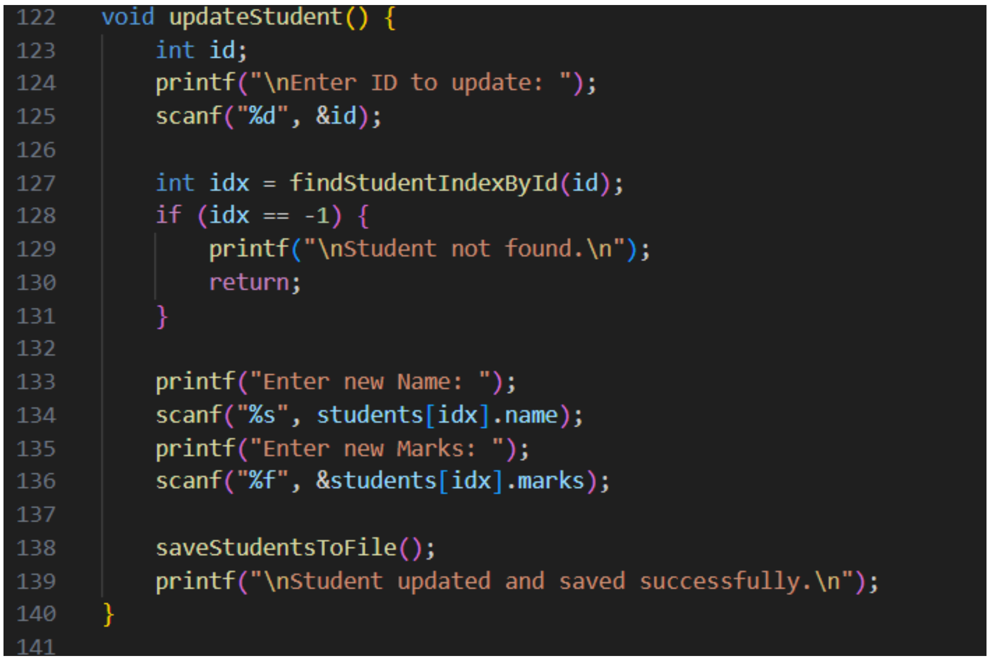
```c
void searchStudent() {
    int id;
    printf("\nEnter ID to search: ");
    scanf("%d", &id);

    int idx = findStudentIndexById(id);
    if (idx == -1) {
        printf("\nStudent not found.\n");
    } else {
        printf("\nStudent found:\n");
        printf("ID: %d\nName: %s\nMarks: %.2f\n",
                students[idx].id,
                students[idx].name,
                students[idx].marks);
    }
}
```



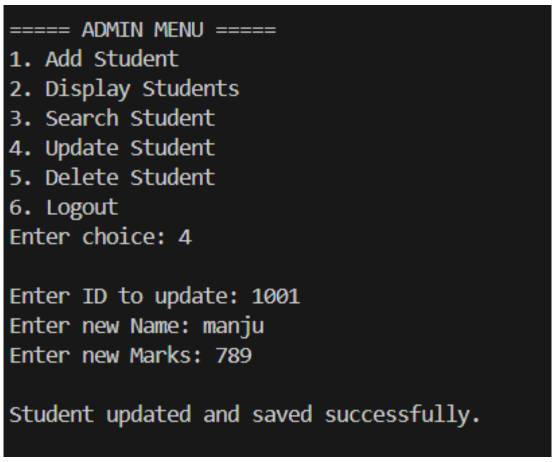
```
===== ADMIN MENU =====
1. Add Student
2. Display Students
3. Search Student
4. Update Student
5. Delete Student
6. Logout
Enter choice: 3

Enter ID to search: 1002

Student found:
ID: 1002
Name: praveen
Marks: 999.00
```

This screen lets the user search for a student using the roll number. If the record exists, the student's details are displayed.

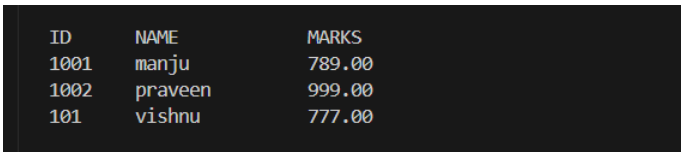
➢ UPDATE STUDENT



```c
void updateStudent() {
    int id;
    printf("\nEnter ID to update: ");
    scanf("%d", &id);

    int idx = findStudentIndexById(id);
    if (idx == -1) {
        printf("\nStudent not found.\n");
        return;
    }

    printf("Enter new Name: ");
    scanf("%s", students[idx].name);
    printf("Enter new Marks: ");
    scanf("%f", &students[idx].marks);

    saveStudentsToFile();
    printf("\nStudent updated and saved successfully.\n");
}
```



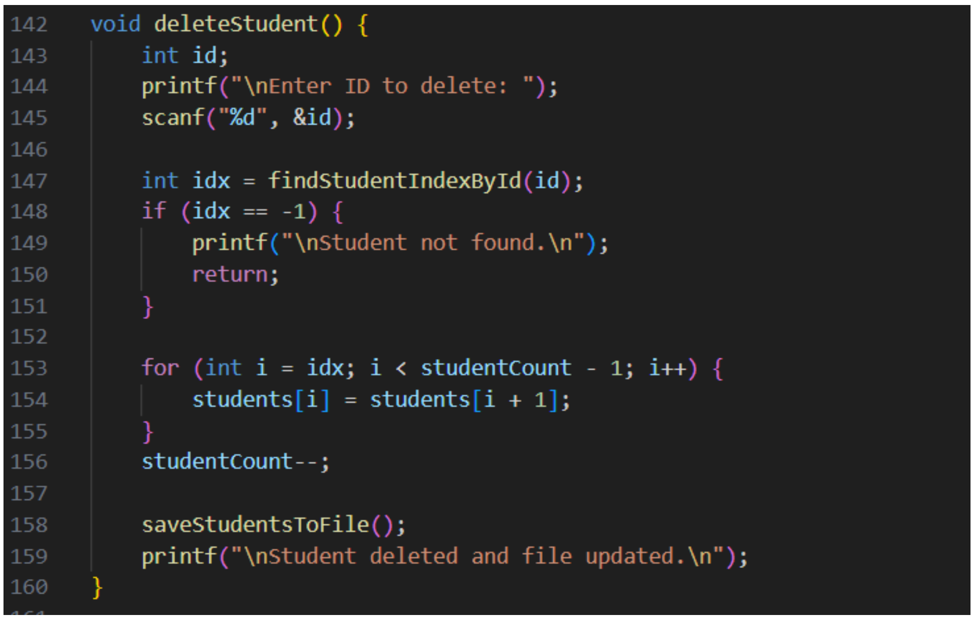
```
===== ADMIN MENU =====
1. Add Student
2. Display Students
3. Search Student
4. Update Student
5. Delete Student
6. Logout
Enter choice: 4

Enter ID to update: 1001
Enter new Name: manju
Enter new Marks: 789

Student updated and saved successfully.
```



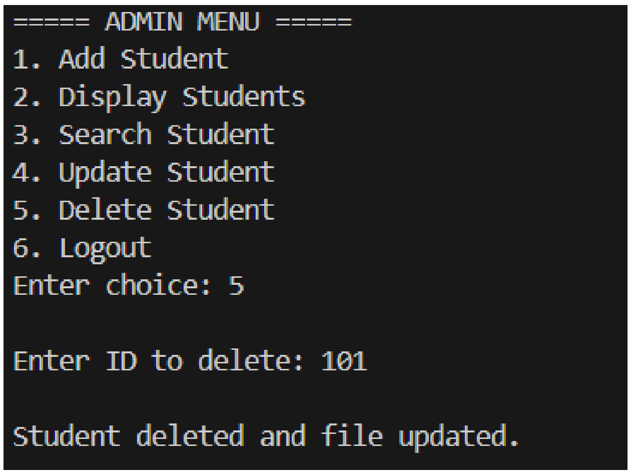
```
ID      NAME            MARKS
1001    manju           789.00
1002    praveen         999.00
101     vishnu          777.00
```

The update screen enables modification of an existing student's details. The admin can update the student's name and marks after entering the roll number.

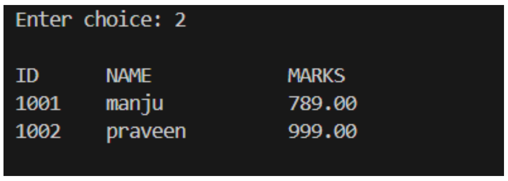
➢ DELETE STUDENT



```c
void deleteStudent() {
    int id;
    printf("\nEnter ID to delete: ");
    scanf("%d", &id);

    int idx = findStudentIndexById(id);
    if (idx == -1) {
        printf("\nStudent not found.\n");
        return;
    }

    for (int i = idx; i < studentCount - 1; i++) {
        students[i] = students[i + 1];
    }
    studentCount--;

    saveStudentsToFile();
    printf("\nStudent deleted and file updated.\n");
}
```



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

Enter ID to delete: 101

Student deleted and file updated.
```



```
Enter choice: 2

ID      NAME            MARKS
1001    manju           789.00
1002    praveen         999.00
```

This output shows the result of deleting a student record. The system removes the matching roll number from the database and confirms the action.

CONCLUSION

The Student Record Management System is a simple program that helps store and manage student information in an easy and organised way. Users can add, view, search, update, and delete student records depending on their role. Admin, Staff, and Guest users have different levels of access, which helps keep the records safe and prevents unwanted changes. The system is menu-driven and easy to understand, making it simple for anyone to use.

This project shows how basic C programming concepts like structures, functions, loops, conditions, and simple file storage can be used to build a small working application. It performs all the main tasks needed for handling student records in a clear and reliable way. Overall, the Student Record Management System is a good example of how useful applications can be created using simple and fundamental C programming techniques.