# School Management System (SMS) Project Documentation

## Project Details

| **Category** | **Detail** |
| --- | --- |
| **Course** | Programming in C, 1st Semester |
| **Authors** | Arnab Roy, Medha Bhatnagar |
| **SAP ID** | 590023460, 590023682 |
| **Project Title** | Basic School Management System (In-Memory Console Application) |

**1. Problem Definition and Scope**

The objective of this project is to build a School Management System (SMS) using C programming with file handling, structures, and a menu-driven interface. The system allows the school administrator to store, search, modify, delete, and retrieve student records efficiently.

### Scope of the System

The project performs complete CRUD operations on student data persisted in a binary file named student\_records.dat.

The system manages the following details for each student:

* Roll Number
* Name
* Class/Grade
* Address
* Total Score
* Fee Status (Paid/Not Paid)

### 

### The System’s Pillars:

1. **File-Based Data Storage:** All student data is stored permanently using binary file handling.
2. **Student Information Management:** Functions include add, view, search, modify, and delete student records.
3. **Structured and Modular Program Design:** Uses struct Student and separate functions for each operation.
4. **Error-Free Input Handling:** Uses clear\_input\_buffer() and input validation to avoid crashes.

## 2. Program Flow Chart (Text-Based Description)

The application operates using a single-level, menu-driven control flow, managed by a continuous do-while loop in the main function. This structure ensures users can navigate between management modules effectively.

**START (main function begins)**

1. Initialize variables (choice)
2. **Main Menu Loop (do-while loop)**
   * Display menu:
     + Add New Student Record
     + View All Student Records
     + Search Student
     + Modify Student
     + Delete Student
     + Exit
   * User inputs choice
   * Input validation
     + If invalid → print error → continue menu
     + If valid → switch(choice) executes
   * **Switch Logic**
     + Case 1 → add\_student()
     + Case 2 → view\_all\_students()
     + Case 3 → search\_student()
     + Case 4 → modify\_student()
     + Case 5 → delete\_student()
     + Case 6 → Exit program
     + Default → Invalid choice message
   * Loop continues until choice = 6

## 3. Algorithm - Step-by-Step Logic

The system’s core functionality is governed by modular algorithms dedicated to each file operation.

### A. Main Control Algorithm

1. Start program
2. Declare choice
3. do-while loop begins
4. Display menu
5. Take user input
6. Validate input
7. switch(choice):
   * Case 1 → add\_student()
   * Case 2 → view\_all\_students()
   * Case 3 → search\_student()
   * Case 4 → modify\_student()
   * Case 5 → delete\_student()
   * Case 6 → Exit
   * Default → Error message
8. Loop continues until choice = 6
9. End

### B. Add Student Algorithm

1. Open file student\_records.dat in append binary mode ("ab").
2. Take roll number input, validating it is an integer.
3. Take name input.
4. Take class input.
5. Take score input, validating it is a non-negative float.
6. Take fee status input (0 or 1), validating the choice.
7. Take address input.
8. Write struct to file using fwrite().
9. Close file.
10. Display success message.

### C. Search Student Algorithm

1. Ask for roll number to search.
2. Open file in read binary mode ("rb").
3. Set flag found = 0.
4. Loop through file using fread().
5. If roll number matches, display details and set found = 1, then break the loop.
6. If found == 0, display "Not Found".
7. Close file.

### D. Modify Student Algorithm

1. Ask for roll number to modify.
2. Open file in read+write binary mode ("r+b").
3. Loop through file using fread().
4. If record found:
   * Ask for new score and new fee status.
   * Move file pointer back one record size with fseek(file, -record\_size, SEEK\_CUR).
   * Overwrite record using fwrite().
   * Break the loop.
5. Close file.

### E. Delete Student Algorithm

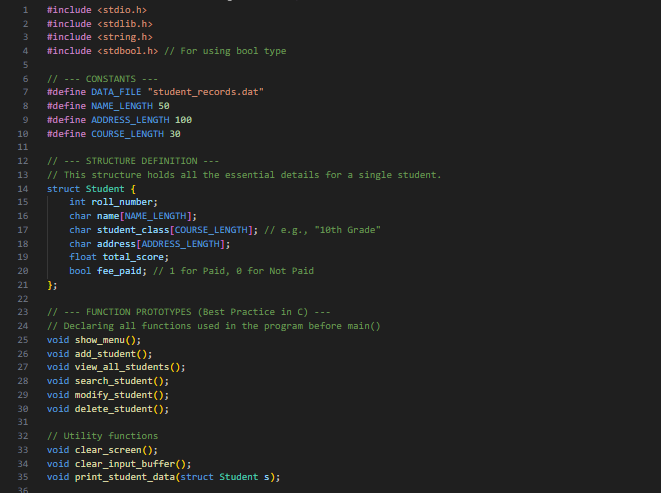
1. Ask for roll number to delete.
2. Open original file (student\_records.dat) in read binary mode ("rb").
3. Open temporary file (temp\_records.dat) in write binary mode ("wb").
4. Loop through original file, copying all records **except** the one to delete to the temporary file.
5. Close both files.
6. Delete original file using remove(DATA\_FILE).
7. Rename temporary file to original file name using rename().
8. Display success or "Not Found" message.

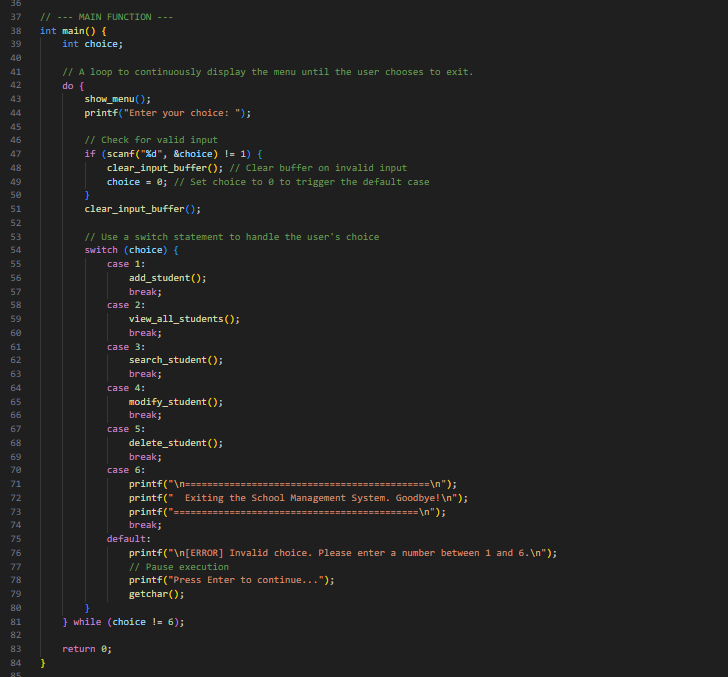
## 4. Problems Faced

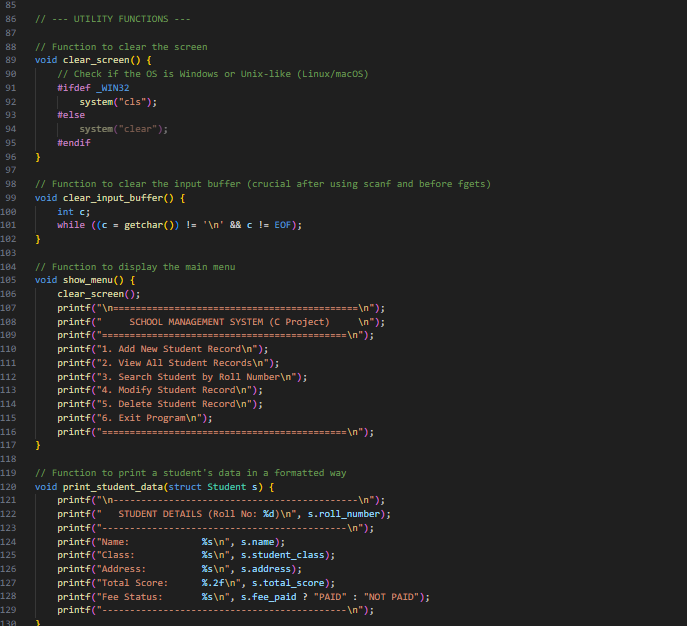
Successful completion of this project required addressing several challenges inherent to C console and file development.

1. **Input Buffer Issues:** scanf leaves newline characters, causing fgets to skip input when reading names or addresses immediately after a numeric input. **Solved using clear\_input\_buffer()** after every scanf call to consume the residual newline.
2. **File Pointer Misalignment:** During record modification, moving the file pointer precisely back to the start of the record is essential to overwrite the old data without corrupting the sequential fixed-size structure. This was resolved using **fseek(file, -record\_size, SEEK\_CUR)**.
3. **Binary File Not Found:** If the data file does not exist when trying to read or modify, fopen() returns NULL. This is **handled with explicit checks** for NULL file pointers, displaying informative error or information messages instead of crashing.
4. **Handling Mixed Input Types:** Careful and consistent **clearing of the input buffer** is needed between scanf (for numbers) and fgets (for strings) to maintain program stability and ensure all fields are correctly populated.
5. **Deletion Logic:** Direct deletion is impossible in a binary file structure. We implemented the **temp file mechanism** to safely delete records without corrupting the original file by copying all desired records to a temporary location and then replacing the original.

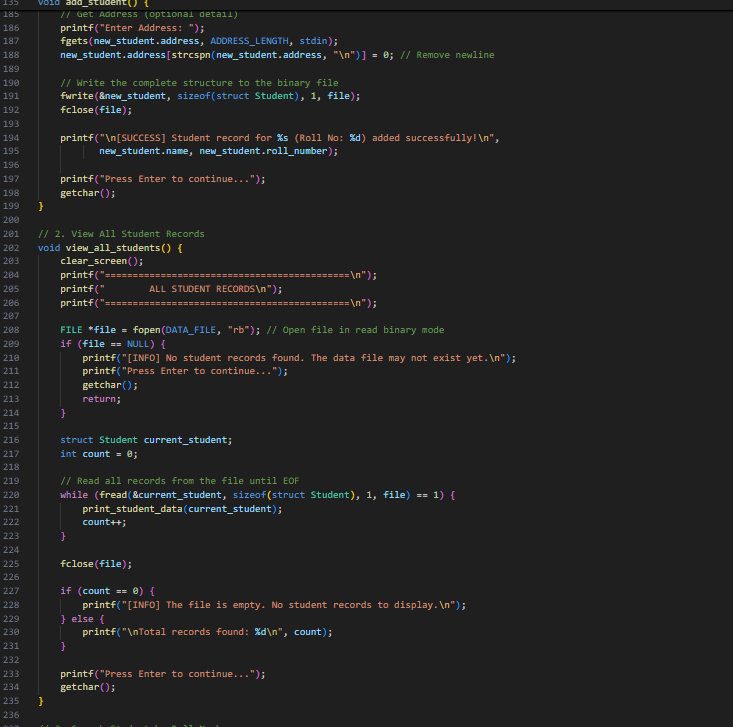
## 5. SNIP OF CODE

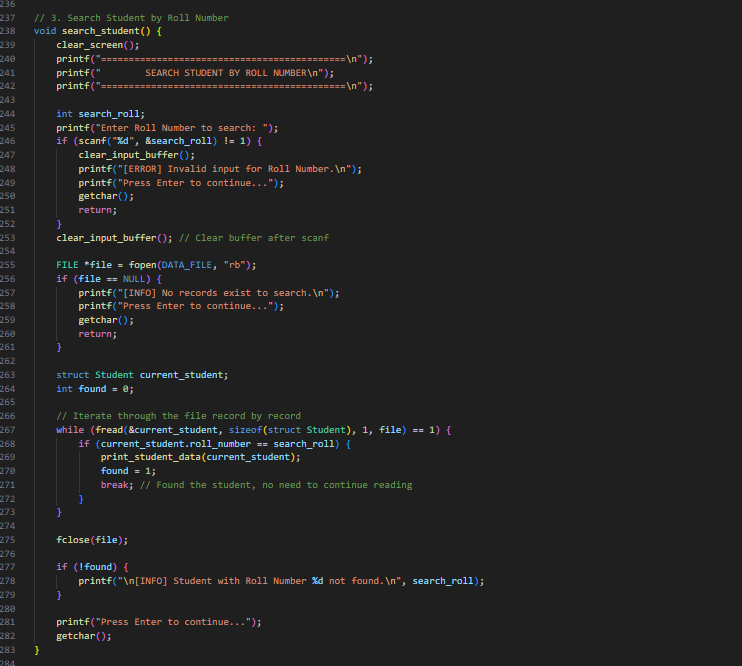


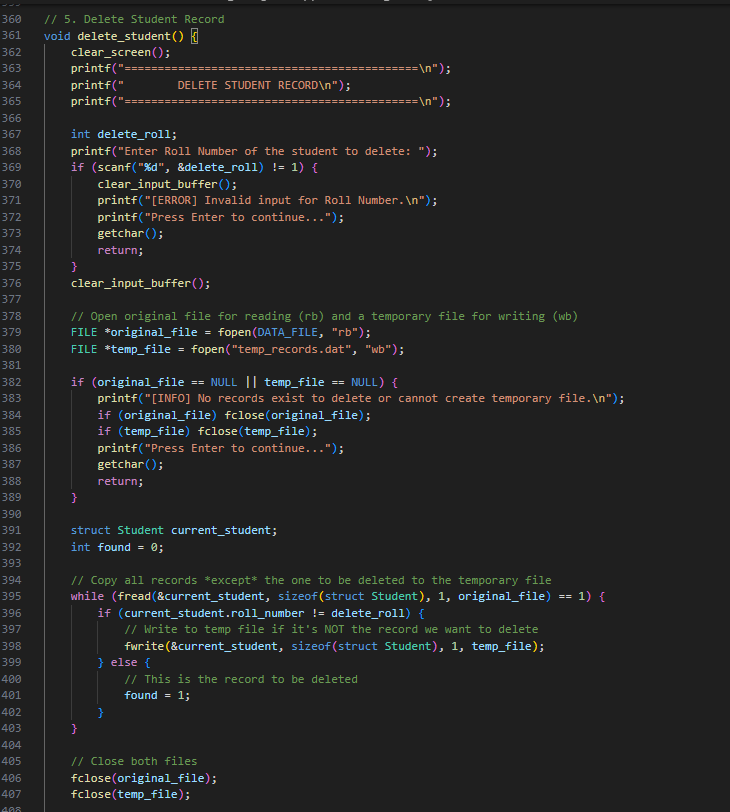


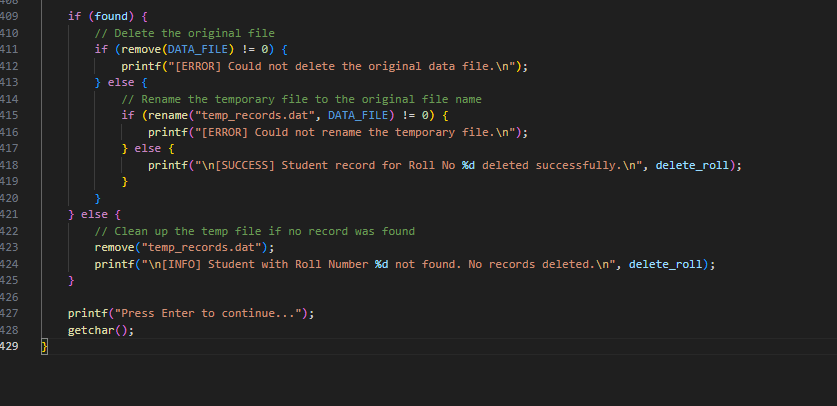












## 6. Output

