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 primary objective of this project is to implement a fundamental **School Management System (SMS)** using core C programming constructs. This console application demonstrates data management capabilities essential for maintaining a school's operational records.

The system's scope is defined by its ability to perform **CRUD** (Create, Read, Update, Delete) operations on three distinct, critical entities: **Students**, **Teachers**, and **Courses**.

The implemented functionalities conceptually underpin four major pillars of enterprise resource planning in an academic environment:

1. Program Structure and Management:
   The application utilizes C struct declarations to define the data schemata for Students, Teachers, and Courses. This establishes the digital architecture necessary for tracking educational programs and offerings. Data manipulation functions (Add/Delete) simulate administrative actions required to maintain a current and accurate list of institutional entities.
2. Student Information Management:
   The system maintains student records within an array, serving as an in-memory Single Source of Truth for essential student demographic and academic data (ID, name, age, marks). This facilitates quick record retrieval and verification, simulating basic enrollment and record-keeping processes.
3. Academic and Faculty Management:
   Separate data arrays are managed for Teachers and Courses. This structure allows the

system to support the tracking of human resources (faculty) and academic resources (course catalog), which is crucial for managing administrative tasks such as teaching load assignments and course scheduling.

4. Role-Based Access (Conceptual Model):
   While the console application lacks explicit security features, the separation of functions (Student Menu, Teacher Menu, etc.) adheres to the conceptual requirement for role-based access. Data visibility and manipulation rights are logically compartmentalized, implying that in a production environment, different user roles would be restricted to only their relevant management sections.

2. Program Flow Chart

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

Text-Based Program Flow Diagram

This diagram illustrates the control flow of the application, emphasizing the menu hierarchy and decision points.

1. **START:** Execution of the main() function begins.
   - **Action:** Initial variables (mainChoice, choice) are declared.
2. **MAIN MENU (Persistent Loop):** A while (1) loop maintains continuous operation.
   - **Output:** Displays options: [1. Student, 2. Teacher, 3. Course, 4. Exit].
   - **Input:** Reads integer mainChoice.
   - **Decision - Validation:** Checks if the input is a valid integer (using scanf return value).
     - *If Invalid:* Display error, call clearInput() to flush the buffer, and continue the loop.
     - *If Valid:* Proceed to the **Switch Statement**.
3. **ENTITY MANAGEMENT (Switch Statement on mainChoice):**
   - **Case 1: Student Management**
     - **Action:** Displays Student Sub-Menu (Add, View, Search, Delete).
     - **Input:** Reads integer choice.
     - **Execution:** Invokes the corresponding student function (e.g., addStudent()).
     - **Return:** Flow returns to the **MAIN MENU**.
   - **Case 2: Teacher Management**
     - **Action:** Displays Teacher Sub-Menu.
     - **Input:** Reads integer choice.
     - **Execution:** Invokes the corresponding teacher function (e.g., addTeacher()).
     - **Return:** Flow returns to the **MAIN MENU**.
   - **Case 3: Course Management**
     - **Action:** Displays Course Sub-Menu.
     - **Input:** Reads integer choice.

        - **Execution:** Invokes the corresponding course function (e.g., addCourse()).
        - **Return:** Flow returns to the **MAIN MENU**.
    - **Case 4: EXIT**
        - **Action:** Program terminates with return 0.
    - **Default:** Displays "Invalid main menu choice."

3. Algorithm - Step-by-Step Logic

The system’s core functionality is governed by the Control Algorithm and the Deletion Algorithm, which requires careful array manipulation.

A. Main Program Control Algorithm (main())

This algorithm manages user interaction and application routing.

1. **START:** Initiate the application with int main().
2. **Initialization:** Declare variables, including mainChoice and choice.
3. **Execution Loop:** Begin while (1).
4. **Display:** Print the Main Menu options to standard output.
5. **Input & Validation:**
    - Attempt to read mainChoice using scanf("%d", ...).
    - **Condition Check:** If scanf fails (returns less than 1), output an error, call clearInput() to resolve buffer issues, and restart the loop (continue).
6. **Navigation (Switch):** Evaluate mainChoice using a switch construct.
7. **Sub-Menu Handling (Cases 1, 2, 3):**
    - Display the specific entity's menu (e.g., Student Menu).
    - Read the sub-menu choice.
    - Execute the appropriate function (e.g., addStudent(), viewStudents()).
8. **Exit (Case 4):** Return 0 to the operating system, terminating the program.
9. **Default:** Handle invalid input by printing an error message.
10. **Loop:** Repeat from Step 4.

B. Delete Record Algorithm (Linear Search and Data Shift)

This algorithm efficiently removes a record from the fixed-size array while ensuring the integrity of the data structure (i.e., no gaps are left).

1. **Input Acquisition:** Prompt the user for the unique identifier (ID) of the record to be deleted.
2. **Search Initialization:** Initialize a linear search loop, iterating from $i = 0$ up to $N - 1$, where N is the current record count (studentCount).
3. **Match Check:** Inside the loop, check the condition: $\text{if } (\text{records}[i].\text{id} == \text{targetID})$.
4. **Record Deletion (If Match Found):**
    - **Shift Operation:** Initiate an inner loop from $j = i$ up to $N - 2$.
    - **Assignment:** Assign the next element to the current position: $\text{records}[j] =

    \text{records}[j + 1]$. This effectively shifts all subsequent records one position to the left, overwriting the deleted record.
    - **Count Update:** Decrement the record counter: $N \leftarrow N - 1$.
    - **Confirmation & Termination:** Print a success message and exit the function (return).
5. **No Match:** If the search loop completes without finding the ID, print a "Record not found" message.

4. Challenges Encountered and Proposed Future Scope

Successful completion of this project required addressing several challenges inherent to C console development. The identified issues define the scope for future enhancements toward a production-ready system.

1. Input Buffer Contamination

- **Challenge:** C programs frequently leave residual newline characters $(\backslash n)$ in the input buffer, causing subsequent string reads to be skipped.
- **Resolution/Mitigation Strategy:** The implemented code includes the clearInput() function and utilizes the scanf(" %[^\n]s", ...) format specifier with a leading space.
- **Future Scope Enhancement:** Implement comprehensive input handling using a dedicated buffer reading function (e.g., fgets) for all inputs, followed by parsing, which offers superior control over buffer state compared to scanf.

2. Data Volatility (Lack of Persistence)

- **Challenge:** All data is stored in global arrays within volatile RAM. Upon program termination, all added records are lost.
- **Resolution/Mitigation Strategy:** The current implementation accepts this limitation given the console application environment.
- **Future Scope Enhancement (Immediate Priority):** Implement File Input/Output (I/O) using C's standard library functions (fopen, fwrite, fread). This will involve writing the entire array structure to a binary file for persistent storage and loading it upon startup.

3. Fixed Data Capacity

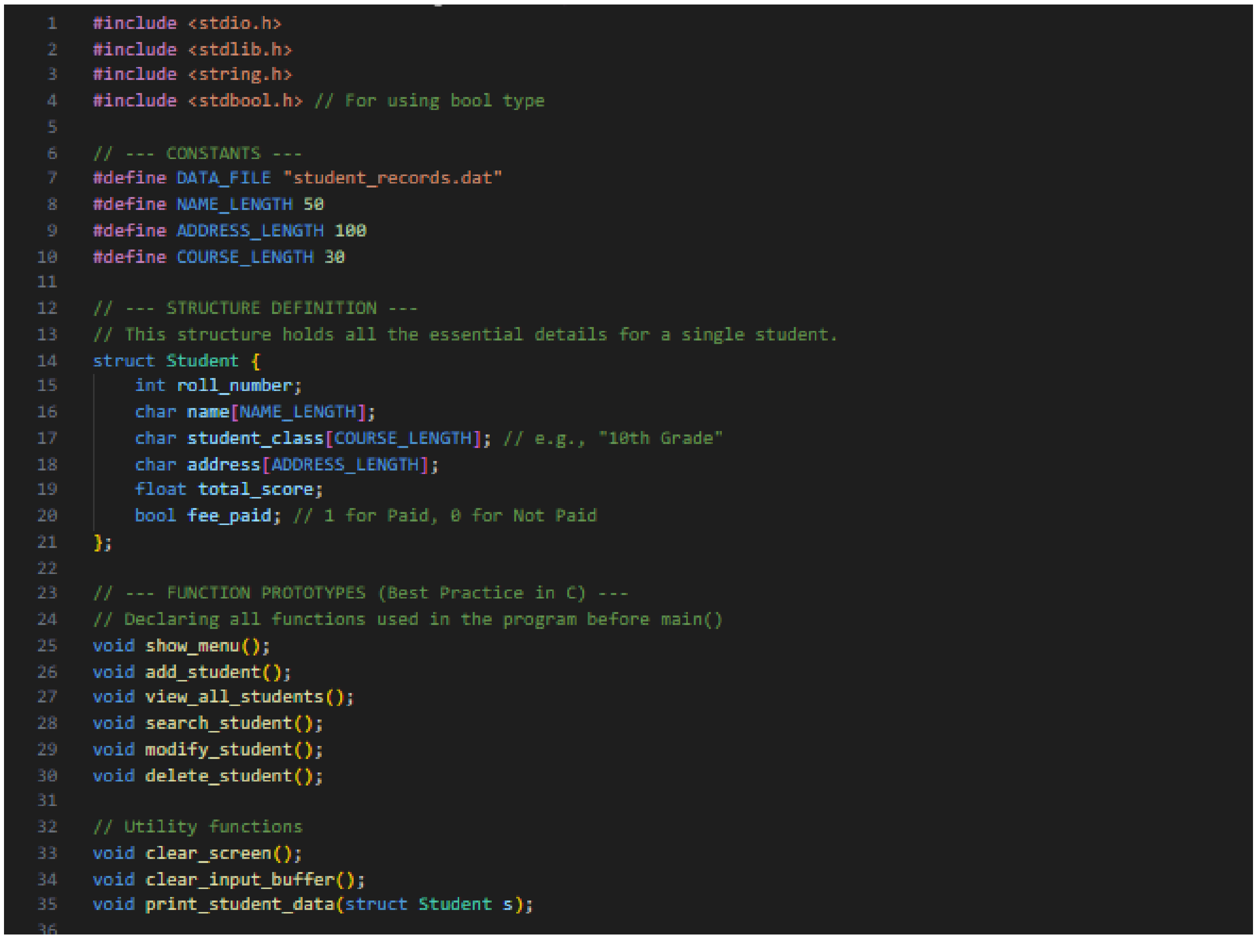
- **Challenge:** Data storage arrays (e.g., students[100]) are statically allocated, imposing a rigid, maximum capacity of 100 records per entity.
- **Resolution/Mitigation Strategy:** The maximum capacity of 100 was set as an acceptable constraint for this proof-of-concept project.
- **Future Scope Enhancement:** Transition to **Dynamic Memory Allocation** using malloc and realloc. This will allow the data structure to expand or shrink at runtime based on the actual number of records, maximizing memory efficiency and scalability.

4. Non-Integer Sub-Menu Input

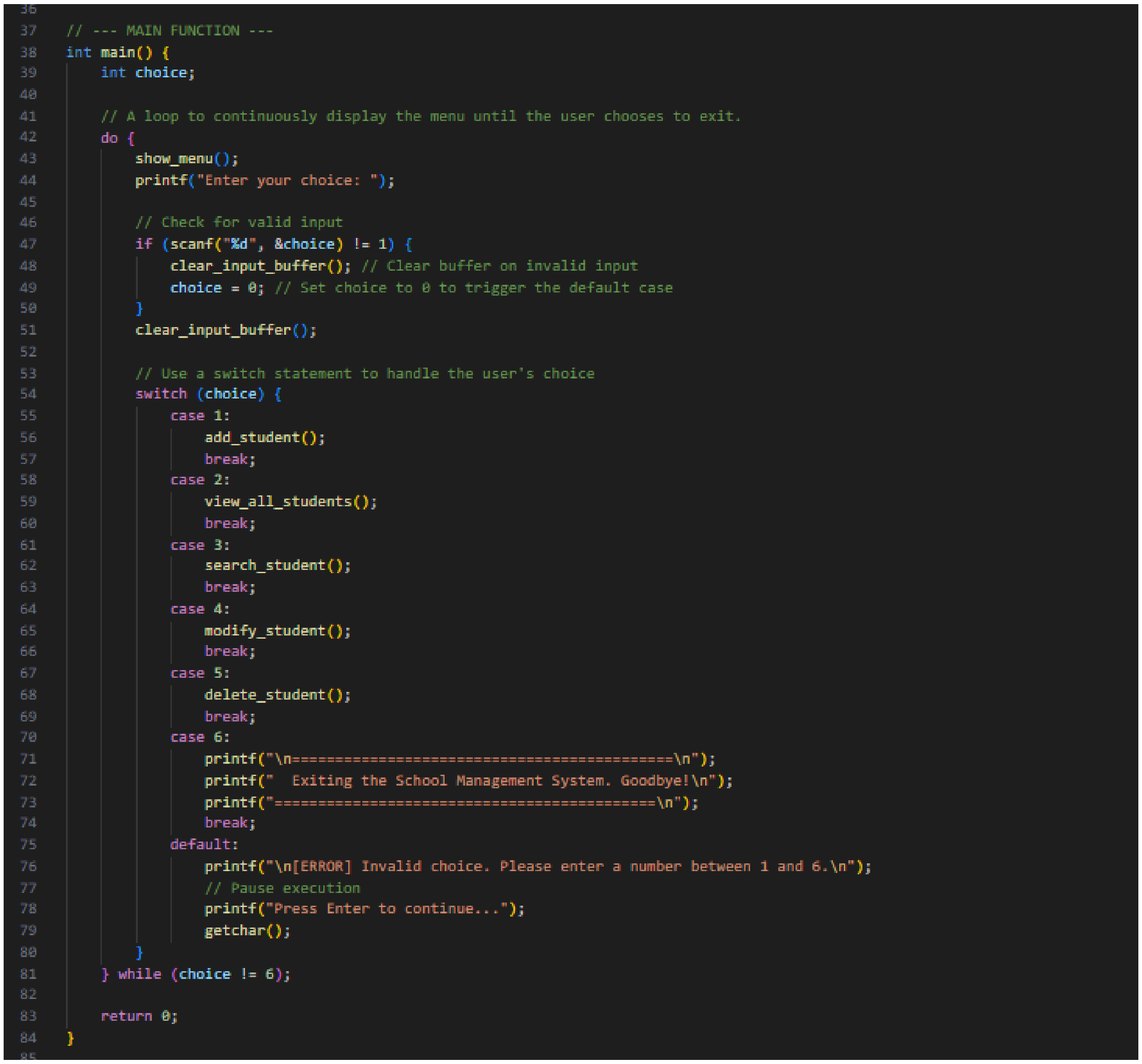
- **Challenge:** While the main menu has robust integer validation, non-integer input in sub-menus currently only prints an error and returns the user to the main menu.
- **Resolution/Mitigation Strategy:** The current flow prioritizes returning to the main loop to maintain program stability.
- **Future Scope Enhancement:** Implement immediate validation and re-prompting within each sub-menu function to force correct data type entry without returning to the main menu loop.

5. Snip of Code

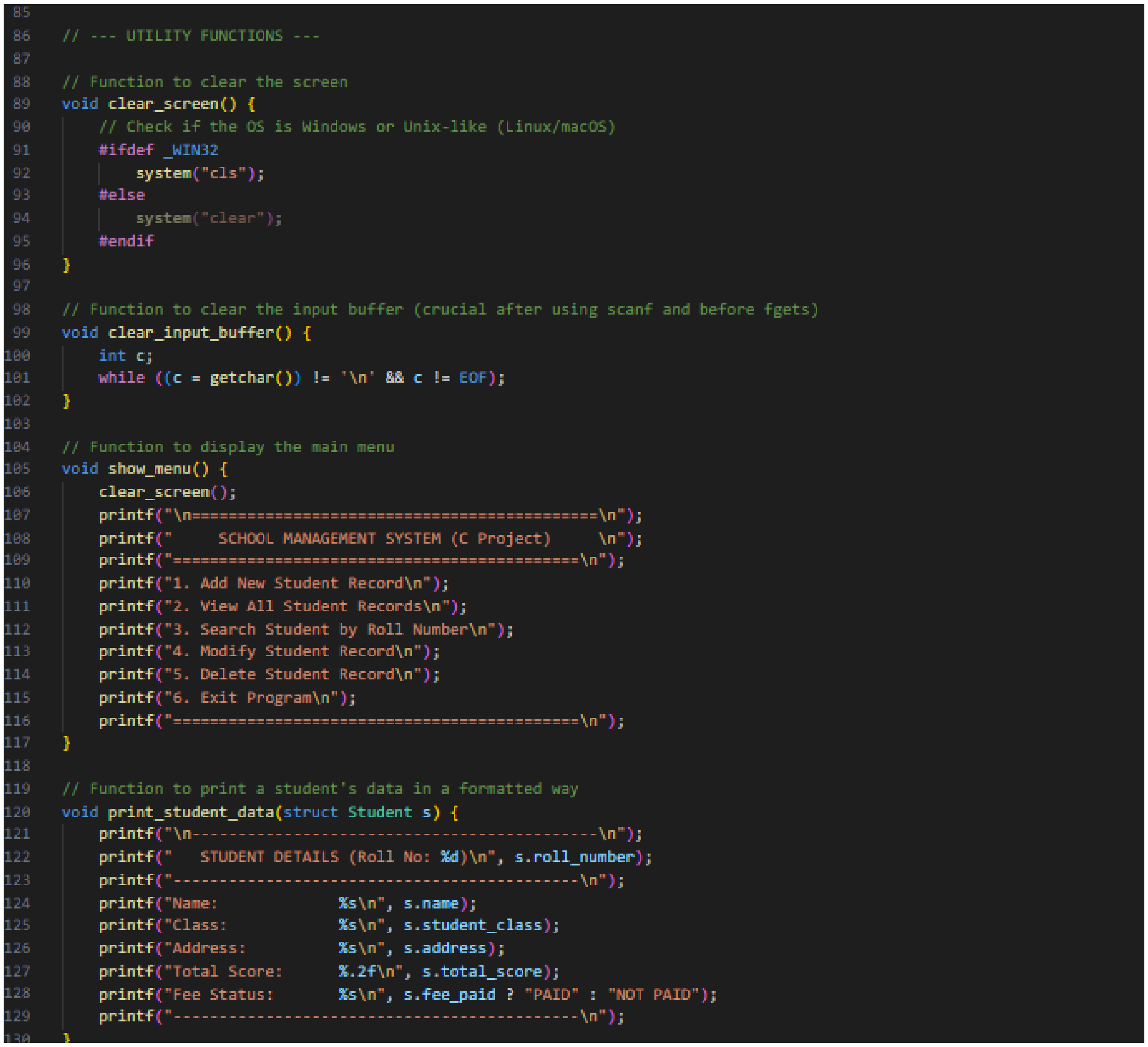
The complete C source code for the School Management System, including the input buffer management solution, is provided below.



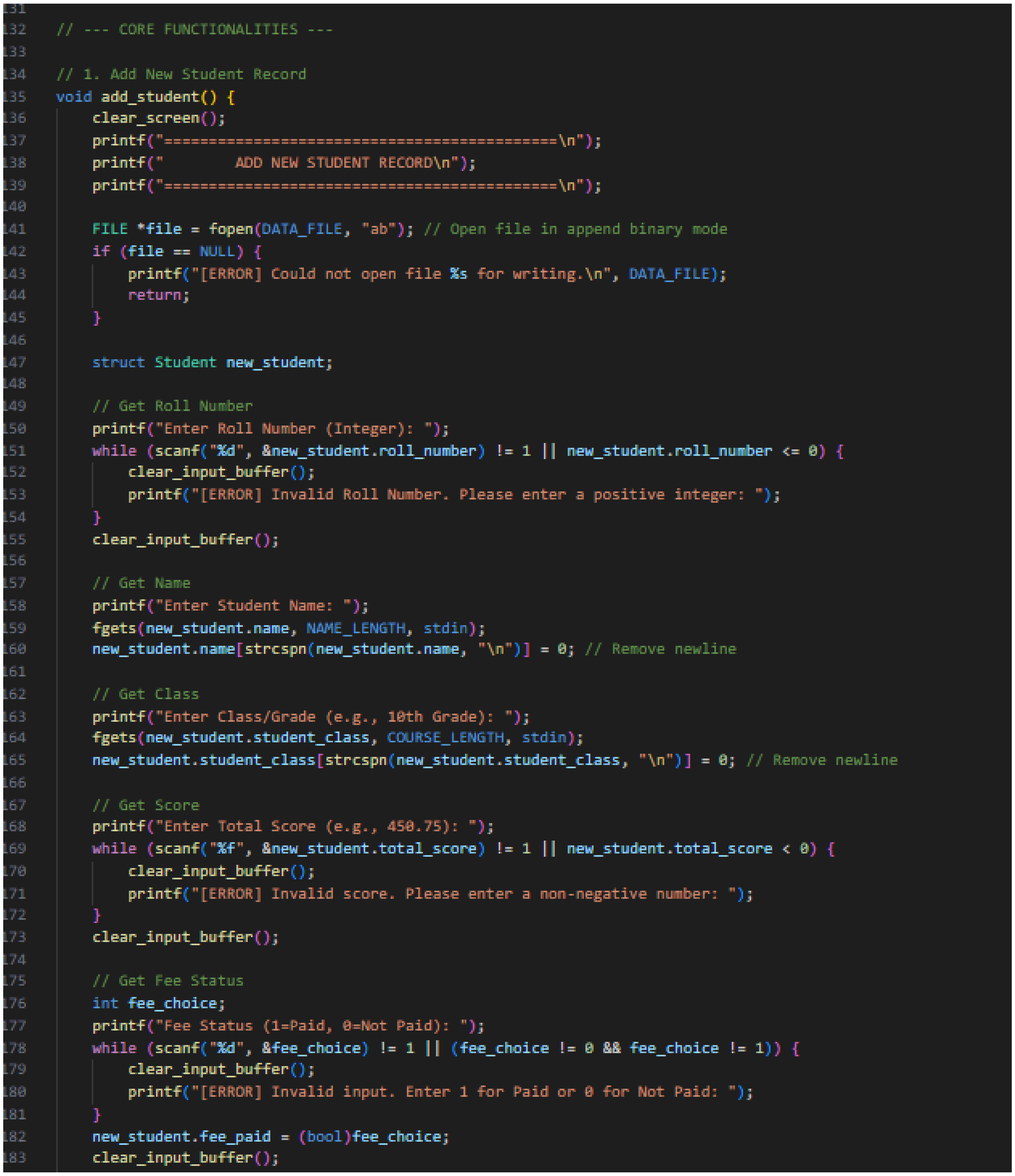
```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h> // For using bool type

// --- CONSTANTS ---
#define DATA_FILE "student_records.dat"
#define NAME_LENGTH 50
#define ADDRESS_LENGTH 100
#define COURSE_LENGTH 30

// --- STRUCTURE DEFINITION ---
// This structure holds all the essential details for a single student.
struct Student {
    int roll_number;
    char name[NAME_LENGTH];
    char student_class[COURSE_LENGTH]; // e.g., "10th Grade"
    char address[ADDRESS_LENGTH];
    float total_score;
    bool fee_paid; // 1 for Paid, 0 for Not Paid
};

// --- FUNCTION PROTOTYPES (Best Practice in C) ---
// Declaring all functions used in the program before main()
void show_menu();
void add_student();
void view_all_students();
void search_student();
void modify_student();
void delete_student();

// Utility functions
void clear_screen();
void clear_input_buffer();
void print_student_data(struct Student s);
```



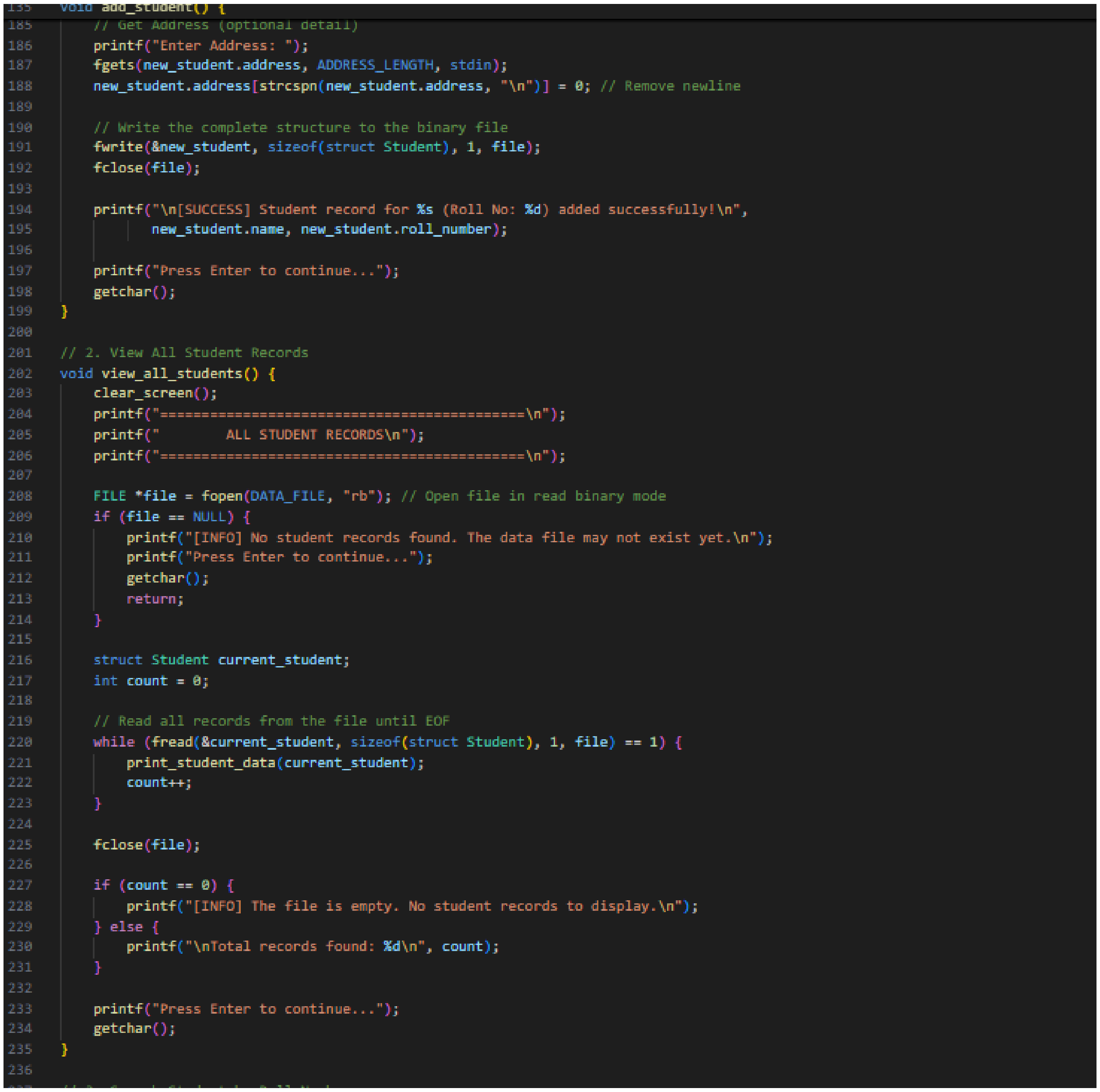
```c
// --- MAIN FUNCTION ---
int main() {
    int choice;

    // A loop to continuously display the menu until the user chooses to exit.
    do {
        show_menu();
        printf("Enter your choice: ");

        // Check for valid input
        if (scanf("%d", &choice) != 1) {
            clear_input_buffer(); // Clear buffer on invalid input
            choice = 0; // Set choice to 0 to trigger the default case
        }
        clear_input_buffer();

        // Use a switch statement to handle the user's choice
        switch (choice) {
            case 1:
                add_student();
                break;
            case 2:
                view_all_students();
                break;
            case 3:
                search_student();
                break;
            case 4:
                modify_student();
                break;
            case 5:
                delete_student();
                break;
            case 6:
                printf("\n===========================================\n");
                printf("  Exiting the School Management System. Goodbye!\n");
                printf("===========================================\n");
                break;
            default:
                printf("\n[ERROR] Invalid choice. Please enter a number between 1 and 6.\n");
                // Pause execution
                printf("Press Enter to continue...");
                getchar();
        }
    } while (choice != 6);

    return 0;
}
```



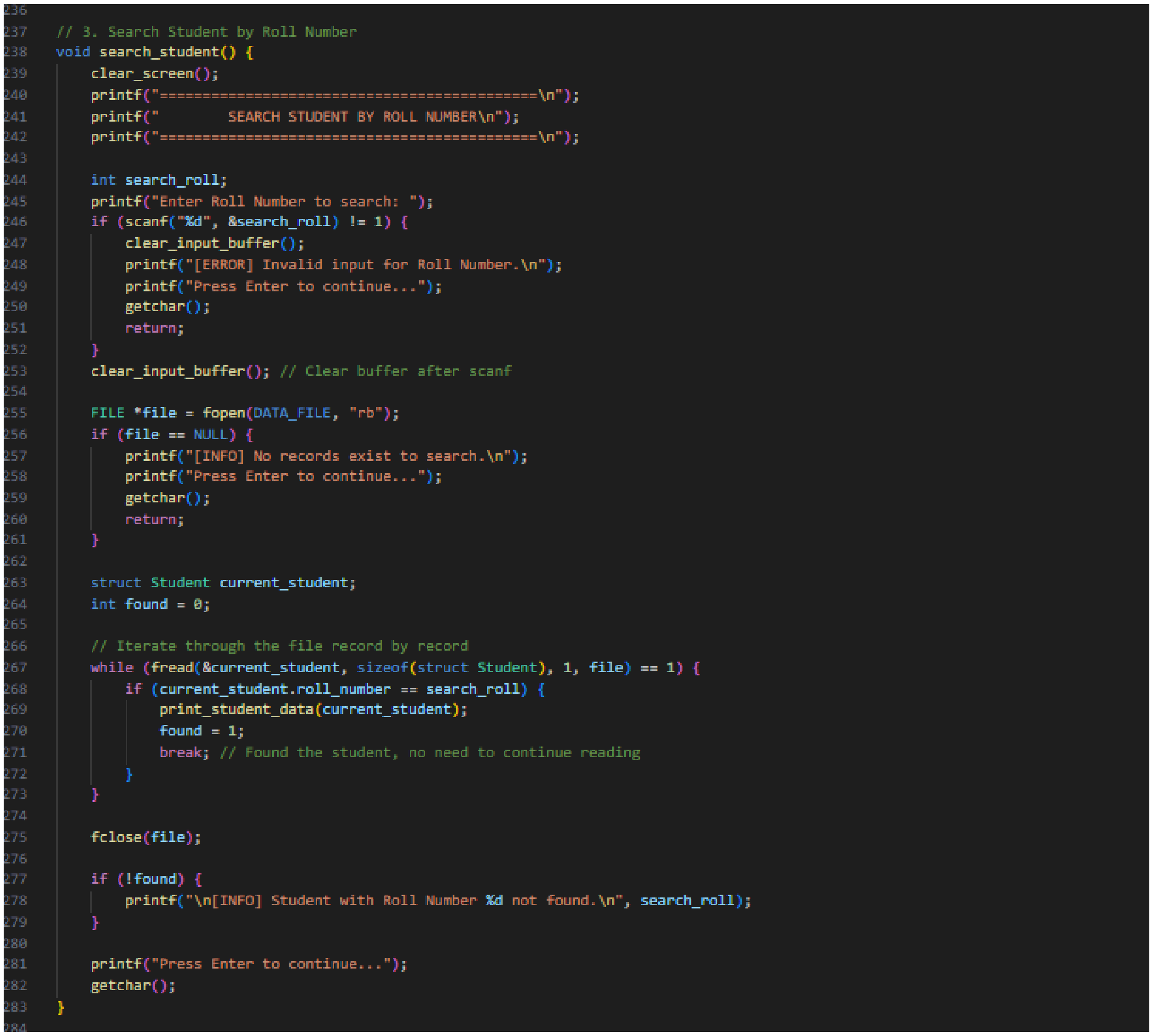
```c

// --- UTILITY FUNCTIONS ---

// Function to clear the screen
void clear_screen() {
    // Check if the OS is Windows or Unix-like (Linux/macOS)
    #ifdef _WIN32
        system("cls");
    #else
        system("clear");
    #endif
}

// Function to clear the input buffer (crucial after using scanf and before fgets)
void clear_input_buffer() {
    int c;
    while ((c = getchar()) != '\n' && c != EOF);
}

// Function to display the main menu
void show_menu() {
    clear_screen();
    printf("\n============================================\n");
    printf("     SCHOOL MANAGEMENT SYSTEM (C Project)     \n");
    printf("============================================\n");
    printf("1. Add New Student Record\n");
    printf("2. View All Student Records\n");
    printf("3. Search Student by Roll Number\n");
    printf("4. Modify Student Record\n");
    printf("5. Delete Student Record\n");
    printf("6. Exit Program\n");
    printf("============================================\n");
}

// Function to print a student's data in a formatted way
void print_student_data(struct Student s) {
    printf("\n--------------------------------------------\n");
    printf("   STUDENT DETAILS (Roll No: %d)\n", s.roll_number);
    printf("--------------------------------------------\n");
    printf("Name:             %s\n", s.name);
    printf("Class:            %s\n", s.student_class);
    printf("Address:          %s\n", s.address);
    printf("Total Score:      %.2f\n", s.total_score);
    printf("Fee Status:       %s\n", s.fee_paid ? "PAID" : "NOT PAID");
    printf("--------------------------------------------\n");
}
```



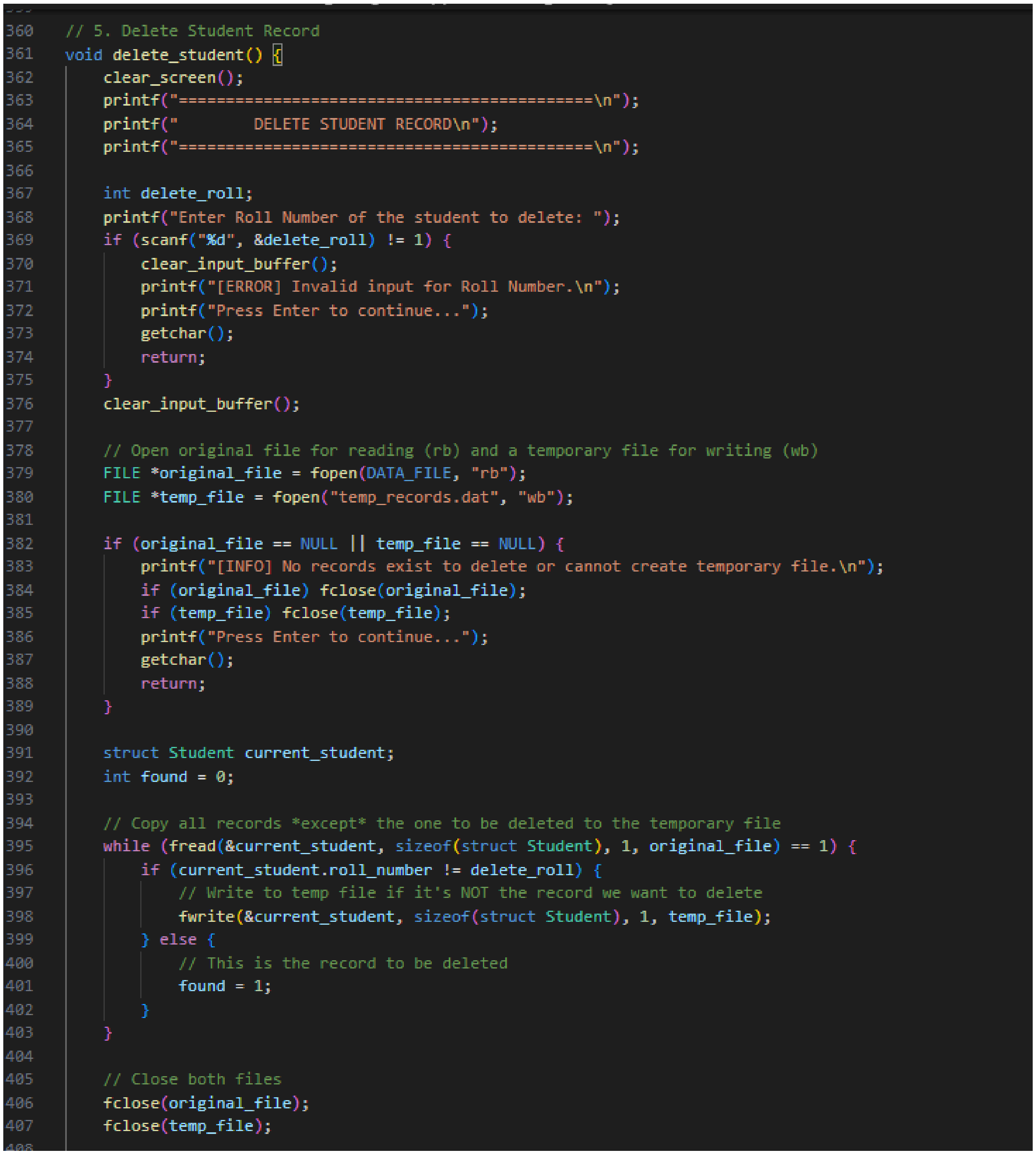
```c

// --- CORE FUNCTIONALITIES ---

// 1. Add New Student Record
void add_student() {
    clear_screen();
    printf("============================================\n");
    printf("        ADD NEW STUDENT RECORD\n");
    printf("============================================\n");

    FILE *file = fopen(DATA_FILE, "ab"); // Open file in append binary mode
    if (file == NULL) {
        printf("[ERROR] Could not open file %s for writing.\n", DATA_FILE);
        return;
    }

    struct Student new_student;

    // Get Roll Number
    printf("Enter Roll Number (Integer): ");
    while (scanf("%d", &new_student.roll_number) != 1 || new_student.roll_number <= 0) {
        clear_input_buffer();
        printf("[ERROR] Invalid Roll Number. Please enter a positive integer: ");
    }
    clear_input_buffer();

    // Get Name
    printf("Enter Student Name: ");
    fgets(new_student.name, NAME_LENGTH, stdin);
    new_student.name[strcspn(new_student.name, "\n")] = 0; // Remove newline

    // Get Class
    printf("Enter Class/Grade (e.g., 10th Grade): ");
    fgets(new_student.student_class, COURSE_LENGTH, stdin);
    new_student.student_class[strcspn(new_student.student_class, "\n")] = 0; // Remove newline

    // Get Score
    printf("Enter Total Score (e.g., 450.75): ");
    while (scanf("%f", &new_student.total_score) != 1 || new_student.total_score < 0) {
        clear_input_buffer();
        printf("[ERROR] Invalid score. Please enter a non-negative number: ");
    }
    clear_input_buffer();

    // Get Fee Status
    int fee_choice;
    printf("Fee Status (1=Paid, 0=Not Paid): ");
    while (scanf("%d", &fee_choice) != 1 || (fee_choice != 0 && fee_choice != 1)) {
        clear_input_buffer();
        printf("[ERROR] Invalid input. Enter 1 for Paid or 0 for Not Paid: ");
    }
    new_student.fee_paid = (bool)fee_choice;
    clear_input_buffer();
```

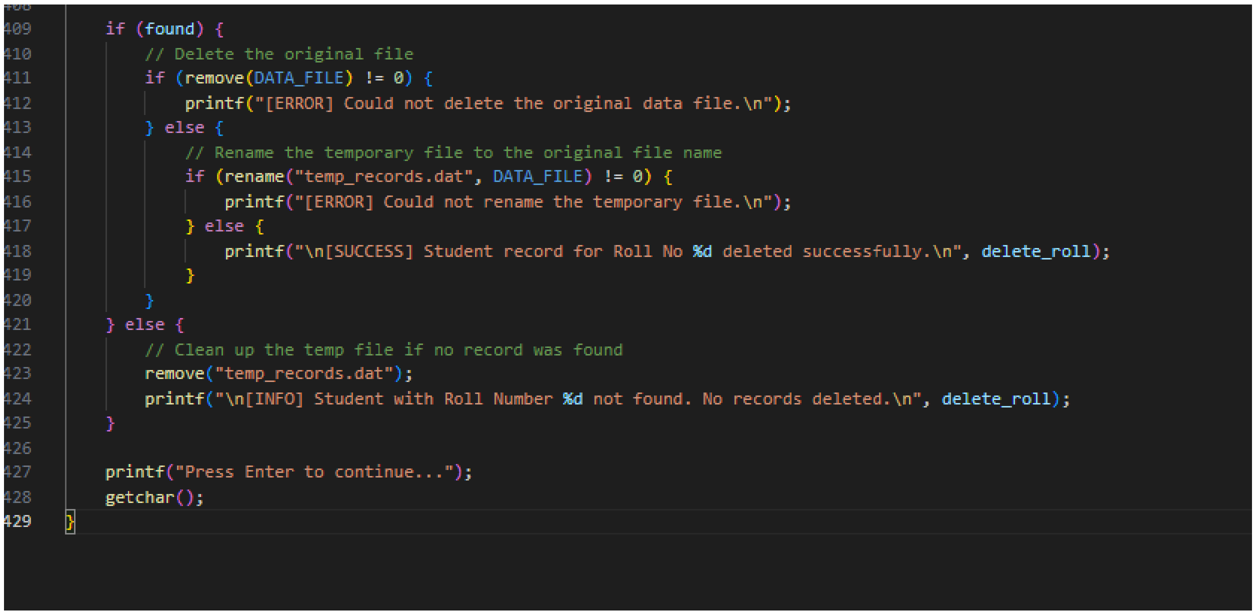


```c
void add_student() {
    // Get Address (optional detail)
    printf("Enter Address: ");
    fgets(new_student.address, ADDRESS_LENGTH, stdin);
    new_student.address[strcspn(new_student.address, "\n")] = 0; // Remove newline

    // Write the complete structure to the binary file
    fwrite(&new_student, sizeof(struct Student), 1, file);
    fclose(file);

    printf("\n[SUCCESS] Student record for %s (Roll No: %d) added successfully!\n",
           new_student.name, new_student.roll_number);

    printf("Press Enter to continue...");
    getchar();
}

// 2. View All Student Records
void view_all_students() {
    clear_screen();
    printf("============================================\n");
    printf("        ALL STUDENT RECORDS\n");
    printf("============================================\n");

    FILE *file = fopen(DATA_FILE, "rb"); // Open file in read binary mode
    if (file == NULL) {
        printf("[INFO] No student records found. The data file may not exist yet.\n");
        printf("Press Enter to continue...");
        getchar();
        return;
    }

    struct Student current_student;
    int count = 0;

    // Read all records from the file until EOF
    while (fread(&current_student, sizeof(struct Student), 1, file) == 1) {
        print_student_data(current_student);
        count++;
    }

    fclose(file);

    if (count == 0) {
        printf("[INFO] The file is empty. No student records to display.\n");
    } else {
        printf("\nTotal records found: %d\n", count);
    }

    printf("Press Enter to continue...");
    getchar();
}
```



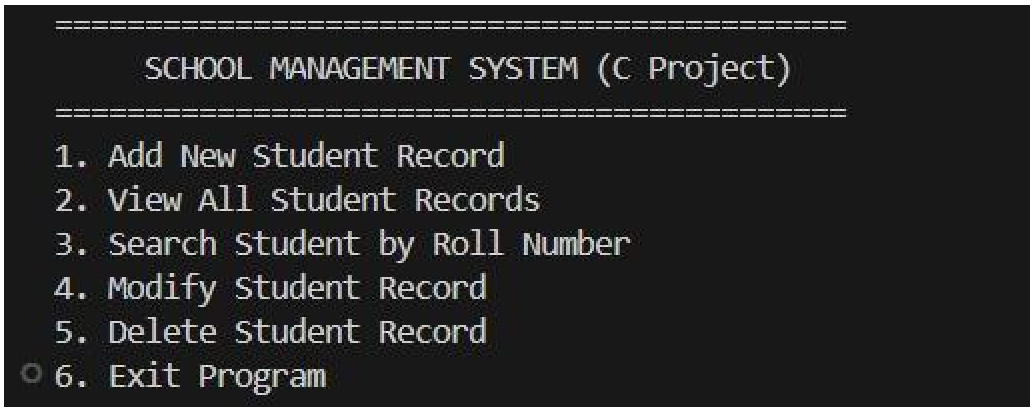
```c

// 3. Search Student by Roll Number
void search_student() {
    clear_screen();
    printf("============================================\n");
    printf("        SEARCH STUDENT BY ROLL NUMBER\n");
    printf("============================================\n");

    int search_roll;
    printf("Enter Roll Number to search: ");
    if (scanf("%d", &search_roll) != 1) {
        clear_input_buffer();
        printf("[ERROR] Invalid input for Roll Number.\n");
        printf("Press Enter to continue...");
        getchar();
        return;
    }
    clear_input_buffer(); // Clear buffer after scanf

    FILE *file = fopen(DATA_FILE, "rb");
    if (file == NULL) {
        printf("[INFO] No records exist to search.\n");
        printf("Press Enter to continue...");
        getchar();
        return;
    }

    struct Student current_student;
    int found = 0;

    // Iterate through the file record by record
    while (fread(&current_student, sizeof(struct Student), 1, file) == 1) {
        if (current_student.roll_number == search_roll) {
            print_student_data(current_student);
            found = 1;
            break; // Found the student, no need to continue reading
        }
    }

    fclose(file);

    if (!found) {
        printf("\n[INFO] Student with Roll Number %d not found.\n", search_roll);
    }

    printf("Press Enter to continue...");
    getchar();
}

```


```c
// 5. Delete Student Record
void delete_student() {
    clear_screen();
    printf("=============================================\n");
    printf("        DELETE STUDENT RECORD\n");
    printf("=============================================\n");

    int delete_roll;
    printf("Enter Roll Number of the student to delete: ");
    if (scanf("%d", &delete_roll) != 1) {
        clear_input_buffer();
        printf("[ERROR] Invalid input for Roll Number.\n");
        printf("Press Enter to continue...");
        getchar();
        return;
    }
    clear_input_buffer();

    // Open original file for reading (rb) and a temporary file for writing (wb)
    FILE *original_file = fopen(DATA_FILE, "rb");
    FILE *temp_file = fopen("temp_records.dat", "wb");

    if (original_file == NULL || temp_file == NULL) {
        printf("[INFO] No records exist to delete or cannot create temporary file.\n");
        if (original_file) fclose(original_file);
        if (temp_file) fclose(temp_file);
        printf("Press Enter to continue...");
        getchar();
        return;
    }

    struct Student current_student;
    int found = 0;

    // Copy all records *except* the one to be deleted to the temporary file
    while (fread(&current_student, sizeof(struct Student), 1, original_file) == 1) {
        if (current_student.roll_number != delete_roll) {
            // Write to temp file if it's NOT the record we want to delete
            fwrite(&current_student, sizeof(struct Student), 1, temp_file);
        } else {
            // This is the record to be deleted
            found = 1;
        }
    }

    // Close both files
    fclose(original_file);
    fclose(temp_file);
```



```c
    if (found) {
        // Delete the original file
        if (remove(DATA_FILE) != 0) {
            printf("[ERROR] Could not delete the original data file.\n");
        } else {
            // Rename the temporary file to the original file name
            if (rename("temp_records.dat", DATA_FILE) != 0) {
                printf("[ERROR] Could not rename the temporary file.\n");
            } else {
                printf("\n[SUCCESS] Student record for Roll No %d deleted successfully.\n", delete_roll);
            }
        }
    } else {
        // Clean up the temp file if no record was found
        remove("temp_records.dat");
        printf("\n[INFO] Student with Roll Number %d not found. No records deleted.\n", delete_roll);
    }

    printf("Press Enter to continue...");
    getchar();
}
```

6. Output



```
==============================================
     SCHOOL MANAGEMENT SYSTEM (C Project)
==============================================
1. Add New Student Record
2. View All Student Records
3. Search Student by Roll Number
4. Modify Student Record
5. Delete Student Record
6. Exit Program
```