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

```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);
```

```
    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
```