A REPORT

ON

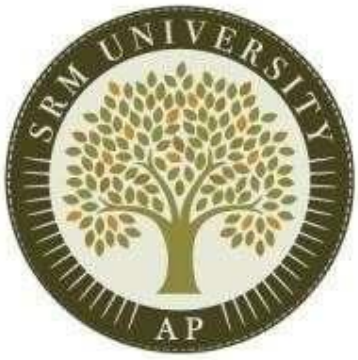
Student Record Management System

By

NAMEOFSTUDENT:	REGISTRATIONNUMBER:
R.Vidya	AP24110011915
K.Jeevana Sarayu	AP24110011926
V. Jeeshitha Sai	AP24110011931
R.Alpana	AP24110011934
G. Chaitanya Siri	AP24110011935
Rolli Gupta	AP24110012176

Prepared in the partialfulfilment of the

Project Based Learning of Course CSE201–Coding Skills-I



SRMUNIVERSITY,AP

ACKNOWLEDGEMENT

We would like to express our sincere gratitude to all those who supported and guided us throughout the successful completion of our project.First and foremost,we aredeeply gratefulto ourViceChancellor, Prof.V. S. Rao, for providing an encouraging academic environment and the necessary resources that enabled us to carry out this work effectively.

We extend our heartfelt thanks to our Faculty Mentors, for their continuous guidance, valuable suggestions, and support during every stage of this project. His expertise and motivation have been crucial in shaping our work.

We also acknowledge the cooperation, teamwork, and collective effort of all team members— whose dedication and contribution were essential to the successful completion of this project.

Finally, we express our appreciation to all others who, directly or indirectly, supported us throughout this journey.

CERTIFICATE

This is to certify that the project entitled **SRMS (Student Report Management System)** has been successfully completed by **R.Vidya (AP24110011915),K.Jeevana Sarayu(AP24110011926),V. Jeeshitha sai (AP24110011931), R.Alpana(AP24110011934), Chaitanya siri (AP24110011935),Rolli.Gupta(AP24110012176),**as a required academic component of the course *CSE 201: Coding Skills - I* during Semester 3 of Academic Year 2025-26 in the Department of Computer Science and Engineering at SRM University , AP .This work was executed under the guidance of the undersigned and is deemed to have successfully met all course requirements as of 10-12-2025.

ABSTRACT

This document details a Student Record Management System implemented in C using core procedural programming concepts, focusing on persistence and enhanced user experience via ANSI escape codes. The system is designed to manage up to 100 student records, with each record (Student struct)containing essential data such as Admission Number (validated against the "APxxxxxxxxxxx" format), Name, Age, Course, Subjects, GPA, and Attendance.

The application leverages file handling using fread() and fwrite() on a binary file (“student_records.dat”) to ensure data persistence across sessions. A key feature is the custom print_header() and color functions utilizing ANSI escape codes,providingastructured,visuallyappealing,terminal-basedinterface that mimics a web application's style.

Core functionalities include: adding new records with input validation; viewing all records in a formatted, columnar table; searching for a specific student by Admission Number; saving records manually; and permanently deleting the database file and in-memory records. This C program demonstrates robust data management, validation, and advanced terminal formatting techniques for a practical utility application.

TABLEOFCONTENTS:

1. INTRODUCTION TO PROJECT REPORT

The efficient management of educational data, particularly student records, is a fundamental requirement for any academic institution. This project addresses this need by presenting a **Student Record Management System** built entirelyin the **C programming language**. Designed for execution within a standard terminal environment, the application provides an interactive command-line interface for storing, retrieving, and organizing student information.

At its core, the system utilizes a **global array of Student structure**. Each Student entity is meticulously defined to capture critical details, including personal information (Name, Age), academic data (Course, Year of Study, Subjects, GPA), administrative identifiers (validated Admission Number), and performance metrics (Attendance Percentage).

A primary objective of this project is to achieve **data persistence**. This is accomplished through file handling mechanisms, specifically using binary files ("student_records.dat"), ensuring that data remains intact and readily available upon application restart. Furthermore, the application enhances the traditional text-based terminal experience by integrating **ANSI escape codes** toimplement colorful, structured headers and menu options. This approach improves user engagement and clarity, moving beyond simple, monochrome output to create a more intuitive and visually separated user experience.

.

2. MAIN TEXT

The Student Record Management System is structures around three key pillars: Data Definition, Persistence Mechanism, and an Interactive User Interface (UI), all implemented using standard C libraries.

2.1 Data Definition and Structure

The foundation of the application is the Student structure, which serves as the blueprint for storing individual student records.

- **Structure (Student)**: Contains fixed-size character array (admn_no, name,course, subjects) and floating-point/integer primitives (age, gpa, attendance_perc, year_of_study).
- **Constants**: MAX_STUDENTS (100) dictates the capacity of the records[] array, and FILENAME (“student_records.dat”) defines the persistent storage location.
- **Validation (is_calid_admn_no)**: Ensures data integrity by strictly enforcing the Admission Number format (APfollowed by 11 digits) using the isdigit() function ctype.h.

2.2 File Handling and Data Persistence

Data persistence is managed through dedicated functions that interact with the local filesystem, treating the records[] array as a complete data payload for file operations.

1. **Loading Records (load_records)**: This function attempts to open the FILENAME in binary read mode ("rb"). It uses a single fread() call to efficiently

read the entire array of Student structs into memory, updating the global student_count based on the number of records successfully read.

2. **Saving Records (save_records)**: This function opens the file in binary write mode ("wb"), overwriting any existing data. It uses fwrite() to write only the student_count number of active records from the records[] array to the disk.
3. **Deletion (delete_all_records)**: This crucial function clears the in-memory student_count to zero and calls the standard library function remove() to permanently delete the persistent database file from the operating system, ensuring a clean slate.

2.3 User Interaction and Interface Design

The system provides a clear, menu-driven experience, enhanced by careful use of terminal styling.

- **Presentation and Styling**: The UI is visually improved using **ANSI escapecodes** (defined by macros like BOLD, GREEN, BG_BLUE). These codes enable colorful text and background effects to distinguish headers, menus, and status messages, significantly improving the readability of the console application.
    - print_header: Creates a visually prominent, centered, and colored banner for each module or screen, providing immediate context to the user.
    - Pause_and_clear: A crucial utility function that flushes output, waits for the user's ENTER input, and then uses the termial codes \033[H\033[J to

clear the screen, maintaining a clean, single-screen view for each operation.

- **Core Operation Modules**: The primary loop in main() continuously presents the display_menu() and routes the user's integer to the corresponding function.
  - Adding Records (add_records): This module handles user input using fgets() to prevent buffer overflows, followed by data type conversion (atoi() for age/year, atof() for GPA/attendance). It features loops for range validation (e.g., Age 16-99, GPA 0.0-10.0) and checks for duplicate Admission Numbers before appending the new record to the records[] array.
  - Viewing Records (view _records): Displays all current records in a clean, tabular format. Cusorm printf format specifiers are use (e.g., %-20s for left-justified names, %6.2f for right-justified GPA) to ensure align correctly and maximize readability.
  - Searching Records (search_record): Iterates through the record[] array using strcmp() to find a match for the requestedAdmission Number. When a match is fount, it presents the data in a detailed, labelled format.

3. RESULTS AND PRINCIPAL OUTCOMES

The C-based Student Record Management System successfully delivers a functional, reliable, and persistent application for managing educational data in a command-line environment. The principal outcomes demonstrate proficiency in foundational C programming, data integrity management, and enhanced user interaction.

3.1 Data Management Efficiency and Integrity

The system's most significant result is the reliable persistence of data without reliance on external libraries or complex database systems**.**

- **Reliable Data Persistence**: By employing binary file I/O (fread and fwrite), the system achieves fast and accurate storage and retrieval of the records[] array. This design ensures that all 100 data slots are handled efficiently, making the data readily available upon application startup and demonstrating effective use of the FILE stream mechanism.
- **Preventing Invalid States**: Strict input validation ensures that critical administrative identifiers, such as theAdmission Number, adhere to the mandated format (APxxxxxxxxxxx). Furthermore, numerical constraints (e.g., 0.0<= GPA<=10.0) prevent the storage of nonsensical or out-of-range data, establishinga high degree of data integrity.
- **Effective Use of Arrays**: The Student records[] array functions as a simple, in-memory cache, allowing for quick searches and views without needing to re-read the file for every operation.

3.2 User Experience and Clarity

The intentional use of presentation utilities transforms the application from a purely functional utility into an intuitive tool.

- **Enhanced Readability via ANSI**: The strategic application of ANSI EscapeCodes (colors and bolding) results in a highly legible and visually separated interface. The print_header() function successfully contextualizes each operation, while colored prompts and status messages clearly distinguish system feedback (e.g., green for success, red for errors, yellow for warnings) from input requests.
- **Structured Output**: The view_records() function produces a clean, columnar report using precise printf formatting. This structure allows users to efficiently scan the data, a critical outcome for any data management tool.
- **Controlled Flow**: The pause_and_clear() utility ensures a smooth, single-screen-per-operation user flow, preventing visual clutter common in console applications and requiring the user's acknowledgment before proceeding.

3.3 Skills Demonstrated

Theprojectsuccessfullydemonstratesseveralkeyoutcomesrelatedto Cdevelopment:

- **Mastery of Structures andArrays**: Successful definition and manipulation of custom struct types within a globally managed fixed-size array.
- **Robust I/O Handling**: Effective use of booth standard I/O for user input (fgets for safe string input) and low-level binary I/O for persistence.
- **Modular Programming:** All major functionalities are isolated into clean, reusable functions, which improves code maintainability.

4. CONCLUSIONS AND RECOMMENDATIONS

4.1 Conclusions

The **Student Record Management System** successfully meets its objective of providing a functional,visuallyenhanced,andpersistentdatamanagementutilitywithinthe constraints of the C programming language and the terminal environment.

- **Proof of Concept**: The project successfully demonstrates the use of core C features—structs, arrays, pointers, and file handling—to build a non-trivial application. It validates the capability of standard C I/O and library functions to manage structured data and ensure application state is preserved.
- **Data Integrity Focus**: Through rigorous input validation, particularly for the Admission Number format and numerical range checks, the application proves reliable in capturing clean data, which is paramount for any administrative system.
- **Effective UI Enhancement**: The strategic implementation ofANSI escape codes provides a polished, interactive user experience that moves beyond traditional monochrome command-line interfaces, proving that powerful formatting can be achieved without external UI libraries.
- **Persistence Model**: The choice of binary file I/O is confirmed as an efficient mechanism for bulk data storage and retrieval, offering better performance than plain text parsing for structured data.

4.2 Recommendations for Future Work

To evolve the system from a demonstrative project to a more robust, industry-standard application, the following enhancements are recommended:

- **Dynamic Memory Allocation (Scalability)** : The current reliance on MAX_STUDENTS and a fixed-size array (records[]) limits scalability. Future development should transition to dynamic memory allocation using malloc and realloc to handle an arbitrary number of records, only constrained by system memory.
- **Advanced Search and Sorting**: Implement new functionalities to search by Name, Course, or GPA. Additionally, introduce sorting capabilities (e.g., sort by GPA descending) to allow users to generate ranked reports. This would require implementing sorting algorithms like Merge Sort or Quick Sort.
- **Record Modificationand Deletion byID**: The systemcurrentlylackstheability to update an existing student's details or delete a single record. A crucial enhancement would be to implement a function that locates a student byAdmission Number and allows the user to modify individual fields or remove the record from the array and file.
- **Data Security and Redundancy**: For production use, explore options to implement basic data checksums or redundant backups of the student_records.dat file to safeguard against file corruption.
- **Improved Input Handling**: Refine input functions to robustly handle non-numeric input for fields expecting integers or floats, instead of relying solely on atoi or atof, which can silently fail or return zero on invalid

5. APPENDICES

Appendix A: Source Code

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h> // New include for isdigit()
#include <unistd.h> // For sleep (optional, for visual effect)

// --- Configuration and Constants ---
#define MAX_STUDENTS 100
#define FILENAME "student_records.dat"
#define MAX_NAME_LEN 50
#define MAX_COURSE_LEN 30
#define MAX_SUBJECTS_LEN 100

// --- ANSI Escape Codes for Styling (Mimicking a Website Look) ---
#define RESET     "\033[0m"
#define BOLD      "\033[1m"
#define RED       "\033[31m"
#define GREEN     "\033[32m"
#define YELLOW    "\033[33m"
#define BLUE      "\033[34m"
#define MAGENTA   "\033[35m"
#define CYAN      "\033[36m"
#define WHITE     "\033[37m"
#define BG_BLUE   "\033[44m"
#define BG_CYAN   "\033[46m"

// --- Global Variables ---
int student_count = 0;

// --- Structure Definition ---
typedef struct {
    char admn_no[14];         // APxxxxxxxxxxx (13 chars + null terminator)
    char name[MAX_NAME_LEN];
    int age;
    char course[MAX_COURSE_LEN];
    char subjects[MAX_SUBJECTS_LEN]; // e.g., "Math, Physics, English"
    float gpa;                // Out of 10.00
    float attendance_perc;    // Total attendance percentage
    int year_of_study;
} Student;

Student records[MAX_STUDENTS];
```

```c
// --- Utility Functions for Presentation ---

/**
 * @brief Prints a colored, centered, boxed header for the application.
 */
void print_header(const char *title) {
    int width = 80;
    int len = strlen(title);
    int padding = (width - len - 2) / 2;

    printf("\n");
    printf(BOLD BG_BLUE WHITE);
    for (int i = 0; i < width; i++) printf(" ");
    printf("\r");
    printf(" %*s%s%*s " RESET, padding, "", title, width - len - 2 - padding, "");
    printf(RESET "\n");
    printf("\n");
}

/**
 * @brief Draws a horizontal separator line.
 */
void print_separator() {
    printf(CYAN);
    for (int i = 0; i < 80; i++) printf("-");
    printf(RESET "\n");
}

/**
 * @brief Prints a colorful menu option.
 */
void print_menu_option(int number, const char *description) {
    printf(CYAN BOLD "  [%d]" RESET WHITE " %s\n" RESET, number, description);
}

/**
 * @brief Flushes output, waits for ENTER key, and clears the screen.
 * This ensures the user has time to read the output before the menu redraws.
 */
void pause_and_clear() {
    fflush(stdout); // Ensure all previous output is immediately rendered
    printf(YELLOW "\nPress ENTER to continue..." RESET);
    while (getchar() != '\n'); // Wait for ENTER key
    printf("\033[H\033[J"); // Clear screen
}
```

```c
// --- Validation Functions ---

/**
 * @brief Validates the Admission Number format (AP followed by 11 digits) using standard C functions.
 * @param admn_no The admission number string.
 * @return 1 on success (valid format), 0 on failure.
 */
int is_valid_admn_no(const char *admn_no) {
    // Expected format: AP + 11 digits = 13 characters total
    if (strlen(admn_no) != 13) {
        return 0;
    }

    // Check first two characters: Must be 'A' and 'P'
    // isupper() check is slightly safer, but direct comparison is fine for fixed 'AP'
    if (admn_no[0] != 'A' || admn_no[1] != 'P') {
        return 0;
    }

    // Check the remaining 11 characters: Must all be digits
    for (int i = 2; i < 13; i++) {
        // isdigit() is defined in <ctype.h>
        if (!isdigit(admn_no[i])) {
            return 0;
        }
    }

    return 1; // All checks passed
}

// --- File Handling Functions (Persistence) ---

/**
 * @brief Loads student records from the binary file.
 */
void load_records() {
    FILE *file = fopen(FILENAME, "rb");
    student_count = 0;

    if (file == NULL) {
        printf(YELLOW "\nDatabase file '%s' not found. Starting with an empty record list.\n" RESET, FILENAME);
        return;
    }

    // Read the entire array in one go
    size_t read_count = fread(records, sizeof(Student), MAX_STUDENTS, file);
    student_count = (int)read_count;
```

```
    fclose(file);
    printf(GREEN BOLD "\nSuccessfully loaded %d records from the database.\n" RESET, student_count);
}

/**
 * @brief Saves current student records to the binary file.
 */
void save_records() {
    FILE *file = fopen(FILENAME, "wb");

    if (file == NULL) {
        printf(RED BOLD "\nERROR: Could not open file '%s' for writing. Records not saved.\n" RESET, FILENAME);
        return;
    }

    // Write only the current number of active students
    fwrite(records, sizeof(Student), student_count, file);

    fclose(file);
    printf(GREEN BOLD "\nSuccessfully saved %d records to the database.\n" RESET, student_count);
}

/**
 * @brief Deletes the persistent file and clears all in-memory records.
 */
void delete_all_records() {
    print_header("DELETE ALL RECORDS");

    printf(RED BOLD "\n  WARNING: This action is permanent and cannot be undone.\n" RESET);
    printf(YELLOW "  Are you sure you want to delete ALL student records? (Type 'YES' to confirm): " RESET);

    char confirmation[10];
    // We use scanf here to read a single token, which is appropriate for a simple confirmation
    if (scanf("%9s", confirmation) != 1) {
        printf(RED BOLD "\n  Confirmation failed. Operation cancelled.\n" RESET);
        // Clear the input buffer
        while (getchar() != '\n');
        pause_and_clear();
        return;
    }
    // Clear the rest of the line after reading the integer
    while (getchar() != '\n');

    if (strcmp(confirmation, "YES") == 0) {
        // 1. Clear in-memory data
        student_count = 0;
```

```
        // 2. Delete the persistent file using the remove() function
        if (remove(FILENAME) == 0) {
            printf(GREEN BOLD "\n  [SUCCESS] All in-memory records cleared and file '%s' successfully deleted.\n" RESET, FILENAME);
        } else {
            // Error could mean the file didn't exist, which is fine, or a permissions issue
            if (remove(FILENAME) == -1) {
                // Check if file exists to give a more specific message
                FILE *file = fopen(FILENAME, "r");
                if (file != NULL) {
                    fclose(file);
                    printf(RED BOLD "\n  [ERROR] File '%s' exists but could not be deleted (Permission issue?).\n" RESET, FILENAME);
                } else {
                    printf(YELLOW BOLD "\n  [INFO] File '%s' did not exist. Records cleared from memory.\n" RESET, FILENAME);
                }
            } else {
                printf(GREEN BOLD "\n  [SUCCESS] All records cleared from memory and file handled.\n" RESET);
            }
        }
    } else {
        printf(YELLOW BOLD "\n  Confirmation failed. Operation cancelled.\n" RESET);
    }

    pause_and_clear();
```

```c
// --- Core Application Functions ---

/**
 * @brief Adds a new student record after prompting the user for details.
 */
void add_record() {
    print_header("ADD NEW STUDENT RECORD");

    if (student_count >= MAX_STUDENTS) {
        printf(RED BOLD "  Database capacity reached (%d students). Cannot add more records.\n" RESET, MAX_STUDENTS);
        pause_and_clear();
        return;
    }

    Student new_student;
    char buffer[256];

    // 1. Admission Number
    printf(YELLOW "  Enter Admission Number (APxxxxxxxxxx): " RESET);
    if (fgets(buffer, sizeof(buffer), stdin) == NULL) { pause_and_clear(); return; }
    buffer[strcspn(buffer, "\n")] = 0; // Remove newline
    strncpy(new_student.admn_no, buffer, 13);
    new_student.admn_no[13] = '\0';

    if (!is_valid_admn_no(new_student.admn_no)) {
        printf(RED BOLD "  Invalid Admission Number format. Must be AP followed by 11 digits.\n" RESET);
        pause_and_clear();
        return;
    }

    // Check for duplicate
    for (int i = 0; i < student_count; i++) {
        if (strcmp(records[i].admn_no, new_student.admn_no) == 0) {
            printf(RED BOLD "  Record with Admission Number %s already exists.\n" RESET, new_student.admn_no);
            pause_and_clear();
            return;
        }
    }

    // 2. Name
    printf(YELLOW "  Enter Student Name: " RESET);
    if (fgets(buffer, sizeof(buffer), stdin) == NULL) { pause_and_clear(); return; }
    buffer[strcspn(buffer, "\n")] = 0;
    strncpy(new_student.name, buffer, MAX_NAME_LEN - 1);
    new_student.name[MAX_NAME_LEN - 1] = '\0';
```

```c
    // 2. Name
    printf(YELLOW "  Enter Student Name: " RESET);
    if (fgets(buffer, sizeof(buffer), stdin) == NULL) { pause_and_clear(); return; }
    buffer[strcspn(buffer, "\n")] = 0;
    strncpy(new_student.name, buffer, MAX_NAME_LEN - 1);
    new_student.name[MAX_NAME_LEN - 1] = '\0';

    // 3. Age (Input validation loop)
    do {
        printf(YELLOW "  Enter Age (16-99): " RESET);
        if (fgets(buffer, sizeof(buffer), stdin) == NULL) { pause_and_clear(); return; }
        new_student.age = atoi(buffer);
    } while (new_student.age < 16 || new_student.age > 99);

    // 4. Course/Major
    printf(YELLOW "  Enter Course/Major: " RESET);
    if (fgets(buffer, sizeof(buffer), stdin) == NULL) { pause_and_clear(); return; }
    buffer[strcspn(buffer, "\n")] = 0;
    strncpy(new_student.course, buffer, MAX_COURSE_LEN - 1);
    new_student.course[MAX_COURSE_LEN - 1] = '\0';

    // 5. Subjects Opted
    printf(YELLOW "  Enter Subjects Opted (Comma separated): " RESET);
    if (fgets(buffer, sizeof(buffer), stdin) == NULL) { pause_and_clear(); return; }
    buffer[strcspn(buffer, "\n")] = 0;
    strncpy(new_student.subjects, buffer, MAX_SUBJECTS_LEN - 1);
    new_student.subjects[MAX_SUBJECTS_LEN - 1] = '\0';

    // 6. GPA (Input validation loop)
    do {
        printf(YELLOW "  Enter GPA (0.00-10.00): " RESET);
        if (fgets(buffer, sizeof(buffer), stdin) == NULL) { pause_and_clear(); return; }
        new_student.gpa = atof(buffer);
    } while (new_student.gpa < 0.0 || new_student.gpa > 10.0);

    // 7. Attendance Percentage (Input validation loop)
    do {
        printf(YELLOW "  Enter Attendance Percentage (0.0-100.0): " RESET);
        if (fgets(buffer, sizeof(buffer), stdin) == NULL) { pause_and_clear(); return; }
        new_student.attendance_perc = atof(buffer);
    } while (new_student.attendance_perc < 0.0 || new_student.attendance_perc > 100.0);

    // 8. Year of Study (Input validation loop)
    do {
        printf(YELLOW "  Enter Year of Study (1-4): " RESET);
        if (fgets(buffer, sizeof(buffer), stdin) == NULL) { pause_and_clear(); return; }
        new_student.year_of_study = atoi(buffer);
    } while (new_student.year_of_study < 1 || new_student.year_of_study > 4);


    // Add the new student to the array
    records[student_count] = new_student;
    student_count++;

    printf(GREEN BOLD "\n  [SUCCESS] Record for %s added successfully! Current total: %d\n" RESET, new_student.name, student_count);

    // Prompt to save immediately
    printf(YELLOW "  Do you want to save the changes to file now? (Y/N): " RESET);
    char choice;
    if (scanf(" %c", &choice) == 1 && (choice == 'Y' || choice == 'y')) {
        save_records();
    }
    // Clear the input buffer
    while (getchar() != '\n');

    pause_and_clear();
}
```

```c
/**
 * @brief Displays all student records in a structured, neat table format.
 */
void view_records() {
    print_header("ALL STUDENT RECORDS");

    if (student_count == 0) {
        printf(YELLOW BOLD "  No records found in the database. Add a new record first (Option 1).\n" RESET);
        pause_and_clear();
        return;
    }

    // --- Table Header ---
    print_separator();
    // ALIGNMENT FIX: The COURSE column width is increased from 10 to 20 to prevent data overflow.
    printf(BOLD BG_CYAN WHITE);
    printf("| %-13s | %-20s | %-4s | %-20s | %-4s | %-6s | %-11s |\n" RESET,
        "ADMN NO.", "NAME", "AGE", "COURSE", "YR", "GPA", "ATTENDANCE");
    print_separator();

    // --- Table Rows ---
    for (int i = 0; i < student_count; i++) {
        Student *s = &records[i];
        // The COURSE column width is 20.
        // Numerical fields (AGE, YR, GPA, ATTENDANCE) are now RIGHT-JUSTIFIED
        // by removing the '-' flag, which ensures the decimal points and '%' symbol align correctly.
        printf("| %-13s | %-20s | %4d | %-20s | %4d | %6.2f | %10.2f%% |\n",
            s->admn_no, s->name, s->age, s->course, s->year_of_study, s->gpa, s->attendance_perc);
    }

    // --- Table Footer ---
    print_separator();
    printf(BOLD WHITE "\n  Total Records: %d\n" RESET, student_count);

    pause_and_clear();
}

/**
 * @brief Searches for a student by Admission Number and displays details.
 */
void search_record() {
    print_header("SEARCH STUDENT RECORD");
    char search_admn_no[14];
    printf(YELLOW "  Enter Admission Number to search (APxxxxxxxxxx): " RESET);
    if (fgets(search_admn_no, sizeof(search_admn_no), stdin) == NULL) { pause_and_clear(); return; }
    search_admn_no[strcspn(search_admn_no, "\n")] = 0;

    if (!is_valid_admn_no(search_admn_no)) {
        printf(RED BOLD "  Invalid Admission Number format.\n" RESET);
        pause_and_clear();
        return;
    }

    for (int i = 0; i < student_count; i++) {
        if (strcmp(records[i].admn_no, search_admn_no) == 0) {
            Student *s = &records[i];

            printf(GREEN BOLD "\n  [MATCH FOUND]\n" RESET);
            print_separator();

            printf(BOLD WHITE "  Admission No: " RESET CYAN "%s\n" RESET, s->admn_no);
            printf(BOLD WHITE "  Name:         " RESET "%s\n", s->name);
            printf(BOLD WHITE "  Age:          " RESET "%d\n", s->age);
            printf(BOLD WHITE "  Course/Major: " RESET "%s\n", s->course);
            printf(BOLD WHITE "  Year of Study:" RESET "%d\n", s->year_of_study);
            printf(BOLD WHITE "  Subjects:     " RESET "%s\n", s->subjects);
            printf(BOLD WHITE "  GPA:          " RESET YELLOW "%.2f / 10.00\n" RESET, s->gpa);
            printf(BOLD WHITE "  Attendance:   " RESET MAGENTA "%.2f%%\n" RESET, s->attendance_perc);

            print_separator();
```

```c
            print_separator();

            printf(BOLD WHITE "  Admission No: " RESET CYAN "%s\n" RESET, s->admn_no);
            printf(BOLD WHITE "  Name:         " RESET "%s\n", s->name);
            printf(BOLD WHITE "  Age:          " RESET "%d\n", s->age);
            printf(BOLD WHITE "  Course/Major: " RESET "%s\n", s->course);
            printf(BOLD WHITE "  Year of Study:" RESET "%d\n", s->year_of_study);
            printf(BOLD WHITE "  Subjects:     " RESET "%s\n", s->subjects);
            printf(BOLD WHITE "  GPA:          " RESET YELLOW "%.2f / 10.00\n" RESET, s->gpa);
            printf(BOLD WHITE "  Attendance:   " RESET MAGENTA "%.2f%%\n" RESET, s->attendance_perc);

            print_separator();
            pause_and_clear(); // Pause to allow reading the result
            return;
        }
    }

    printf(RED BOLD "\n  [NOT FOUND] No student found with Admission Number: %s\n" RESET, search_admn_no);
    pause_and_clear(); // Pause to allow reading the result
}

/**
 * @brief Displays the main application menu.
 */
void display_menu() {
    print_header("STUDENT RECORD MANAGEMENT SYSTEM");

    printf(BOLD WHITE "  Welcome! Select an option from the menu below:\n\n" RESET);

    print_menu_option(1, "Add New Student Record");
    print_menu_option(2, "View All Student Records");
    print_menu_option(3, "Search Record by Admission Number");
    print_menu_option(4, "Save Records to File");
    print_menu_option(5, "DELETE ALL RECORDS (Start Fresh)");
    print_menu_option(6, "Exit Application (Unsaved data will be lost!)");

    print_separator();
    printf(YELLOW "  Enter your choice: " RESET);
}
```

```c
// --- Main Function ---

int main() {
    // Attempt to load existing data immediately on startup
    load_records();
    // Clear screen for a neat start (optional)
    printf("\033[H\033[J");

    int choice;

    while (1) {
        display_menu();

        if (scanf("%d", &choice) != 1) {
            printf(RED BOLD "\n  Invalid input. Please enter a number.\n" RESET);
            // Clear the input buffer for the next loop iteration
            while (getchar() != '\n');
            sleep(1);
            printf("\033[H\033[J"); // Clear screen after error message
            continue;
        }
        // Clear the rest of the line after reading the integer
        while (getchar() != '\n');

        // Clear screen before executing the action, ensuring the action's output is clean and visible
        printf("\033[H\033[J");

        switch (choice) {
            case 1:
                add_record();
                break;
            case 2:
                view_records();
                break;
            case 3:
                search_record();
                break;
            case 4:
                save_records();
                // We add a manual pause here since save_records doesn't use pause_and_clear
                // due to its internal Y/N input loop logic.
                pause_and_clear();
                break;
            case 5:
                delete_all_records();
                break;
            case 6:
                printf(MAGENTA BOLD "\n  Thank you for using the Student Management System. Goodbye!\n" RESET);
                return 0;
            default:
                printf(RED BOLD "\n  Invalid choice. Please enter a number between 1 and 6.\n" RESET);
                pause_and_clear();
                break;
        }
    }

    return 0;
}
```

Appendix B:Data File Structure Specification

The student_records.dat file is a binary file composed of a contiguous sequence of Student structures. It is not human-readable but is optimized for speed when loading and saving the entire dataset

DataField	**DataType (C)**	**Size (Bytes)**	**Description**
admn_no	char[14]	14	Admission Number (e.g., AP12345678901)
name	char[50]	50	Student'sfullname
age	int	4	Student'sage
course	char[30]	30	CourseorMajor
subjects	char[100]	100	Comma-separated list of subjects
gpa	float	4	GradePointAverage (0.00-10.00)
attendance_perc	float	4	Attendance Percentage (0.0-100.0)
year_of_study	int	4	Yearof enrollment(1-4)
TOTALPERRECORD	-	**206**	**sizeof(Student)**

Appendix C:Nomenclature and ANSIColor Mapping

This table provides the mapping for the ANSI Escape Codes used throughout the application to enhance the terminal UI. These sequences are critical for achieving the "web-like" presentation.

MacroName	ANSI Code	Color/Style	UsageContext
RESET	\033[0m	Resetallattributes	Always placed atthe end of a colored string.
BOLD	\033[1m	Bold/Brighttext	Emphasizing key words, headings.
GREEN	\033[32m	ForegroundGreen	Successmessages ([SUCCESS]).
RED	\033[31m	ForegroundRed	ErrorandWarningmessages (ERROR, WARNING).
YELLOW	\033[33m	Foreground Yellow	User prompts,general warnings.
CYAN	\033[36m	ForegroundCyan	Menuoptions,separators.
BG_BLUE	\033[44m	BackgroundBlue	Main menu and function headers.
BG_CYAN	\033[46m	BackgroundCyan	Tableheadersin view_records().

6. REFERENCES

Since this project is an independent software implementation based solely on standard C libraries and system functionalities, the references primarily cite the programming language standards and the relevant operating system specifications that govern the terminal features used.

External Data and Inspiration

1. **Kernighan, B. W. & Ritchie, D. M**., *The C Programming Language* 2nd edition, Prentice Hall, Upper Saddle River, NJ (1988).
2. **ISO/IEC 9899:1999 (C99**), *Programming languages — C*, International Organization for Standardization and the International Electro technical Commission (1999).
3. **POSIX.1-2008 Standard (IEEE Std 1003.1™-2008),** *Portable Operating System Interface: Shells and Utilities*, The Open Group (2008). (Referenced for remove(), stdio.h functions, and standard environment behavior.)
4. **ANSI Standard X3.64-1979 (Dec. 1979),** *ANSI Escape Sequences*, American National Standards Institute. (Referenced for the terminal control codes used for text coloring and screen clearing, e.g., \033[H\033[J and \033[31m.)
5. **Stroustrup, B.,** *Programming: Principles and Practice Using C++* 2nd edition, Addison-Wesley, Upper Saddle River, NJ (2014). (Referenced for general procedural design principles and robust I/O handling practices, such as using fgets for safer input.)