CMR UNIVERSITY

Private University Established in Karnataka State by Act No. 45 of 2013

SCHOOL OF ENGINEERING AND TECHNOLOGY

A Project Work Report

On

"STUDENT MANAGEMENT SYSTEM"

Submitted in partial fulfillment of the requirements for the Course

Programming in C(4CSPL1112) in

Bachelor of Technology
In
Artificial Intelligence and Machine Learning

SOET, CMR University, Bangalore

Submitted by:

Pooja Rajpurohit (24BBTCA084)
Rohini S (24BBTCA096)
Neysa Mary Pramod (24BBTCA078)
Likhita Shree S D (24BBTCA068)

Submitted to:

Prof. Anila V R
Asst. Professor
Dept. of CSE

Department of Artificial Intelligence and Machine Learning
Off Hennur - Bagalur Main Road,
Near Kempegowda International Airport, Chagalahatti,
Bangalore, Karnataka-562149

2024-2025

CMR UNIVERSITY
Private University Established in Karnataka State by Act No. 45 of 2013

SCHOOL OF ENGINEERING AND TECHNOLOGY
Chagalahatti, Bengaluru, Karnataka- 562149

Department of Artificial Intelligence and machine Learning

CERTIFICATE

Certified that the Project Work entitled "**Student Management System"** carried out by **Pooja Rajpurohit (24BBTCA084), Rohini S (24BBTCA096), Neysa Mary Pramod (24BBTCA078),** and **Likhita Shree S D (24BBTCA068)** bonafide students of **SCHOOL OF ENGINEERING AND TECHNOLGY** in partial fulfillment for the award of **BACHELOR OF TECHNOLOGY** in 2nd Semester Artificial Intelligence and Machine Learning of **CMR UNIVERSITY**, Bengaluru during the year 2024. It is certified that all corrections/suggestions indicated for the Internal Assessment have been incorporated in the report. The project has been approved as it satisfies the academic requirements in respect of project work prescribed for the said degree.

Signature of Course In-charge

……

Signature of HOD

……

TABLE OF CONTENT

ABSTRACT

The **Student Management System (SMS)** is a software application developed using the C programming language to digitize and streamline student data management processes within academic institutions. It is designed to provide a centralized platform for storing, accessing, updating, and managing critical student information such as personal details, academic performance, attendance records, and grades. The SMS aims to eliminate the inefficiencies and potential errors of manual record-keeping by automating key administrative functions, thus enhancing institutional productivity and data accuracy.

This project features a console-based interface that is intuitive and user-friendly, allowing faculty and administrative staff to perform operations like adding, searching, updating, or deleting student records. Core C programming concepts such as structures, file handling, and functions are effectively utilized to build a robust and efficient backend system. The implementation includes modular components that support future enhancements like GUI integration or database connectivity.

The SMS not only facilitates efficient data handling and secure storage through persistent file management but also supports transparency and informed decision-making. It offers a scalable solution for small to mid-sized educational institutions seeking to digitize their operations. By minimizing manual workload and improving communication, this system contributes significantly to the modernization of academic administration.

Keywords: Student Management System, C Programming, File Handling, Academic Records, Data Automation, Student Information System, Grade Tracking, Attendance Management, Education Technology

CHAPTER-1

INTRODUCTION

Student Management System Overview : A Student Management System (SMS) is a comprehensive software solution developed to streamline, automate, and centralize all aspects of student data management within educational institutions. As the complexity of academic programs grows and the number of enrolled students increases, the need for a robust system that can manage student-related data efficiently becomes essential. This system enables institutions to handle vital administrative tasks such as enrollment, attendance tracking, grade management, timetable scheduling, and communication all within a unified platform. It plays a critical role in enhancing educational efficiency by supporting educators, administrators, students, and parents with real-time access to academic records and institutional updates.

The primary goal of the Student Management System is to optimize student data handling and academic administration through digitization and automation. The system is built with the following core purposes Maintains a complete digital repository of student details including personal information, guardian contacts, academic history, and disciplinary records. Facilitates easy data retrieval, updating, and management from a single access point. Helps track academic progress by recording grades, assignments, and examination results. Supports GPA calculations, transcript generation, and progress reports to monitor student learning outcomes over time. Integrates communication tools (e.g., notifications, emails, announcements) to enhance collaboration between faculty, students, and parents. Reduces information silos and ensures all stakeholders remain informed about academic activities and updates.

The Student Management System supports a broad range of academic and administrative applications, enabling educational institutions to digitize operations while ensuring secure and organized data management: Student Record Management Stores student data in a structured format. Enables easy modification and retrieval of personal and academic details. Attendance Monitoring tracks student attendance on a daily basis. Automatically generates reports (daily, weekly, monthly) for use by faculty and administrative staff. Can be integrated with hardware systems such as RFID scanners or biometric devices for seamless recording. Manages scores from tests, exams, quizzes, and assignments. Automatically calculates averages, grades, and cumulative GPAs. Offers visual tools such as charts and dashboards for performance analysis and decision support.

CHAPTER-2

PROJECT STATEMENT AND SOLUTION

2.1 Project Statement

Managing student data manually becomes increasingly difficult as the number of students grows. Important tasks such as grade calculation, report generation, attendance tracking, and data retrieval for analysis or audits can become overwhelming for administrative staff. Moreover, without a centralized system, information is often scattered across departments, leading to inefficiencies and delayed decision-making. This lack of integration hampers collaboration, reduces transparency, and affects the overall quality of education delivery.

The **Student Management System (SMS)** project addresses these issues by offering a centralized, automated solution developed in the C programming language. It is designed to streamline academic and administrative processes by enabling structured data entry, secure storage, and efficient retrieval of student records. Through the use of file handling, data structures, and a menu-driven interface, this system enables users to perform tasks such as adding new student records, viewing all data, updating or deleting information, and generating reports—all within a single application.

The SMS empowers institutions to make data-driven decisions, reduce paperwork, and save valuable time. It also lays a strong foundation for future integration with advanced technologies such as databases, mobile applications, and cloud-based platforms. Ultimately, this project aims to enhance the effectiveness and efficiency of student information management while contributing to a more transparent and organized academic environment. Furthermore, the project lays a strong technical foundation for future upgrades. Although it is currently built using the C programming language for simplicity and portability, the modular structure of the system allows it to be scaled or integrated with advanced technologies such as **relational databases (e.g., MySQL), mobile applications, or cloud-based platforms**. Such enhancements can enable remote access, improved data visualization, and real-time synchronization across multiple departments.

In conclusion, the Student Management System serves as a practical, scalable, and essential tool for modern educational institutions. It addresses critical pain points in data management, enhances administrative transparency, and promotes effective collaboration. By streamlining student information management, the SMS contributes to a more organized, efficient, and future-ready academic environment.

2.2 Proposed Solution

The proposed solution is a **Student Management System (SMS)** developed using the C programming language that provides an efficient, reliable, and easy-to-use platform for managing student data in academic institutions. The system is designed to replace traditional, paper-based data management with a digital, file-based solution that automates common administrative tasks. The console-based application provides a menu-driven interface for the following primary functionalities:

1. **Student Data Management**
    - **a.** Users can add, update, delete, and view student records.
    - **b.** Each record includes roll number, name, marks, and calculated grade.
    - **c.** Validation checks are built into input functions to ensure the integrity and format of the data (e.g., alphanumeric roll numbers, alphabetic names, and valid float marks).
1. **Grade Calculation and Academic Monitoring**
    - **a.** The system automatically assigns grades based on entered marks using predefined grading logic.
    - **b.** It allows users to search and retrieve student performance information quickly for academic review.
2. **Attendance and Performance Tracking (Extensible)**
    - **a.** Though not yet fully implemented in this version, the system is designed to be extensible for integrating attendance monitoring through structures or external modules.

    Future versions can support weekly/monthly attendance summaries and generate automated alerts or notifications.

3. **Data Persistence with File Handling**
    - **a.** Records are saved in a binary file (students.dat), ensuring efficient I/O operations and quick access to large volumes of data.
    - **b.** Temporary files (temp.dat) are used during record deletion or updates to preserve file integrity and avoid corruption.
4. **User Interface and Usability**
    - **a.** A simple command-line interface with numbered options guides users through each operation.
    - **b.** The interface design ensures the system is usable by users with minimal technical expertise.

5. **Modularity and Maintainability**
    a. The system is designed with a modular structure, dividing each core functionality into separate functions for better readability and maintainability.
    b. This approach allows for easy debugging and enhancement, enabling future integration with databases (e.g., MySQL) or web-based dashboards.
6. **Communication & Notifications (Future Scope)**
    a. The system lays the foundation for integrating communication features that can be used to notify students or parents via email or SMS.
    b. These features can be expanded by connecting to APIs or external systems.
7. **Scalability and Adaptability**
    a. Though built in C as a lightweight application, the system is scalable to accommodate thousands of records without significant performance degradation.
    b. With codebase modularization, the system can evolve into a GUI-based or cloud-integrated application using frameworks or database layers.

The system's modular architecture ensures that different functional units—such as record management, attendance tracking, and grading—are compartmentalized. This modular design makes the application highly adaptable for future enhancements. For example:

Scalability: The current system can be expanded to support larger datasets by integrating external databases like MySQL or SQLite, ensuring faster queries and reliable data storage.

GUI Integration: The system can be adapted into a graphical user interface using libraries like GTK or Qt for better user experience and wider accessibility.

Cloud or Network Deployment: With minimal modifications, the core logic can be extended into a client-server model, allowing for cloud storage, multi-user access, and remote data synchronization.

Cross-Platform Compatibility: Since it's written in standard C, the codebase is portable and can be compiled and run on multiple operating systems with minor changes.

These features make the system not only efficient for current use but also flexible enough to evolve with technological advancements and institutional growth.

CHAPTER -3

TOOLS USED

The successful development and implementation of **the Student Management System (SMS)** required a combination of programming languages, development environments, and supporting technologies.

The following tools and technologies were used in the development of the Hospital Management System:

1. C Programming Language: Chosen for its high performance, low-level control, and robust file-handling capabilities, C was ideal for creating a lightweight HMS for resource-constrained environments like small hospitals.

2. Standard C Libraries:

stdio.h: Facilitated console input/output (e.g., printf, scanf) and file operations (fopen, fread, fwrite) for storing data in text files (patients.txt, doctors.txt, etc.).

stdlib.h: Provided utility functions like strtol for integer validation in the getValidInteger function, ensuring robust input handling.

string.h: Supported string operations (e.g., strcpy, strcmp) for managing patient names, doctor specialties, and diagnoses.

ctype.h: Enabled character validation (e.g., isdigit) to enhance input processing reliability.

3. File Handling: Used to store data persistently in text files (e.g., patients.txt, appointments.txt) using binary file operations, offering a lightweight alternative to databases.

4. Integrated Development Environment (IDE): Visual Studio Code (VS Code) was used for coding, debugging, and compilation. Its C/C++ extensions provided syntax highlighting, code completion, and an integrated terminal for running GCC, enhancing productivity and code quality.

5. Compiler: GCC (GNU Compiler Collection) compiled the C code into executables, supporting cross-platform compatibility (Windows, Linux, macOS .

6. Operating System Compatibility: The HMS was designed to run on Windows, Linux, and macOS with a C compiler, using standard C libraries and relative file paths to ensure portability across diverse hospital environments.

CHAPTER -4

SOURCE CODE

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>

struct Student {
    char rollNo[20];
    char name[50];
    float marks;
    char grade;
};

// Function prototypes
void addStudent();
void viewStudents();
void searchStudent();
void updateStudent();
void deleteStudent();
char calculateGrade(float marks);

// Helper validation functions
int isValidAlphanumeric(const char *str) {
    if (*str == '\0') return 0; // empty string not allowed
    while (*str) {
        if (!isalnum(*str)) return 0;
        str++;
    }
    return 1;
}
```

```
int isValidFloat(const char *str) {
    if (*str == '\0') return 0;
    int hasDecimal = 0;
    // Optional leading + or -
    if (*str == '+' || *str == '-') str++;
    if (*str == '\0') return 0;
    while (*str) {
        if (*str == '.') {
            if (hasDecimal) return 0; // second decimal point
            hasDecimal = 1;
        } else if (!isdigit(*str)) {
            return 0;
        }
        str++;
    }
    return 1;
}

int isValidName(const char *str) {
    if (*str == '\0') return 0; // empty string not allowed
    while (*str) {
        if (!isalpha(*str) && !isspace(*str)) return 0;
        str++;
    }
    return 1;
}

int readRollNumber(const char* prompt, char *outRollNo, size_t maxLen) {
    char buffer[100];
    while (1) {
        printf("%s", prompt);
        if (!fgets(buffer, sizeof(buffer), stdin)) {
            return 0;
        }
```

```
        buffer[strcspn(buffer, "\n")] = 0; // remove newline
        if (isValidAlphanumeric(buffer) && strlen(buffer) < maxLen) {
            strncpy(outRollNo, buffer, maxLen);
            outRollNo[maxLen - 1] = '\0';
            return 1;
        }
        printf("Invalid input. Please enter alphanumeric characters only.\n");
    }
}

int readFloat(const char* prompt, float *outValue) {
    char buffer[100];
    while (1) {
        printf("%s", prompt);
        if (!fgets(buffer, sizeof(buffer), stdin)) {
            return 0;
        }
        buffer[strcspn(buffer, "\n")] = 0; // remove newline
        if (isValidFloat(buffer)) {
            *outValue = strtof(buffer, NULL);
            return 1;
        }
        printf("Invalid input. Please enter a valid number.\n");
    }
}

int readName(const char* prompt, char *outName, size_t maxLen) {
    char buffer[100];
    while (1) {
        printf("%s", prompt);
        if (!fgets(buffer, sizeof(buffer), stdin)) {
            return 0;
        }
        buffer[strcspn(buffer, "\n")] = 0; // remove newline
```

```
        if (isValidName(buffer) && strlen(buffer) < maxLen) {
            strncpy(outName, buffer, maxLen);
            outName[maxLen - 1] = '\0';
            return 1;
        }
        printf("Invalid input. Please enter letters and spaces only.\n");
    }
}

int main() {
    int choice;
    do {
        printf("\n*** Student Record Management System ***\n");
        printf("1. Add Student\n");
        printf("2. View All Students\n");
        printf("3. Search Student by Roll Number\n");
        printf("4. Update Student\n");
        printf("5. Delete Student\n");
        printf("6. Exit\n");
        printf("Enter your choice: ");
        if (scanf("%d", &choice) != 1) {
            printf("Invalid input. Please enter a number between 1 and 6.\n");
            while (getchar() != '\n'); // clear input buffer
            continue;
        }
        while (getchar() != '\n'); // clear leftover newline

        switch (choice) {
            case 1: addStudent(); break;
            case 2: viewStudents(); break;
            case 3: searchStudent(); break;
            case 4: updateStudent(); break;
            case 5: deleteStudent(); break;
            case 6: printf("Exiting...\n"); break;
```

```
            default: printf("Invalid choice!\n");
        }
    } while (choice != 6);

    return 0;
}

char calculateGrade(float marks) {
    if (marks >= 90) return 'A';
    else if (marks >= 75) return 'B';
    else if (marks >= 60) return 'C';
    else if (marks >= 40) return 'D';
    else return 'F';
}

void addStudent() {
    FILE *fp = fopen("students.dat", "ab");
    if (fp == NULL) {
        printf("Error opening file!\n");
        return;
    }

    struct Student s;
    if (!readRollNumber("Enter Roll Number: ", s.rollNo, sizeof(s.rollNo))) {
        printf("Error reading roll number.\n");
        fclose(fp);
        return;
    }
    if (!readName("Enter Name: ", s.name, sizeof(s.name))) {
        printf("Error reading name.\n");
        fclose(fp);
        return;
    }
    if (!readFloat("Enter Marks: ", &s.marks)) {
```

```
        printf("Error reading marks.\n");
        fclose(fp);
        return;
    }
    s.grade = calculateGrade(s.marks);
    fwrite(&s, sizeof(struct Student), 1, fp);
    fclose(fp);
    printf("Student added successfully.\n");
}

void viewStudents() {
    FILE *fp = fopen("students.dat", "rb");
    if (fp == NULL) {
        printf("Error opening file!\n");
        return;
    }

    struct Student s;
    int count = 0;
    printf("\n--- All Students ---\n");
    while (fread(&s, sizeof(struct Student), 1, fp)) {
        printf("Roll No: %s\nName: %s\nMarks: %.2f\nGrade: %c\n\n",
               s.rollNo, s.name, s.marks, s.grade);
        count++;
    }
    if (count == 0) {
        printf("No records found.\n");
    }
    fclose(fp);
}

void searchStudent() {
    FILE *fp = fopen("students.dat", "rb");
    if (fp == NULL) {
```

```
        printf("Error opening file!\n");
        return;
    }

    struct Student s;
    char roll[20];
    int found = 0;
    if (!readRollNumber("Enter Roll Number to Search: ", roll, sizeof(roll))) {
        printf("Error reading roll number.\n");
        fclose(fp);
        return;
    }
    while (fread(&s, sizeof(struct Student), 1, fp)) {
        if (strcmp(s.rollNo, roll) == 0) {
            printf("\n--- Student Found ---\n");
            printf("Roll No: %s\nName: %s\nMarks: %.2f\nGrade: %c\n",
                   s.rollNo, s.name, s.marks, s.grade);
            found = 1;
            break;
        }
    }
    if (!found) {
        printf("Student not found.\n");
    }
    fclose(fp);
}

void updateStudent() {
    FILE *fp = fopen("students.dat", "rb+");
    if (fp == NULL) {
        printf("Error opening file!\n");
        return;
    }
```

```
    struct Student s;
    char roll[20];
    int found = 0;
    if (!readRollNumber("Enter Roll Number to Update: ", roll, sizeof(roll))) {
        printf("Error reading roll number.\n");
        fclose(fp);
        return;
    }
    while (fread(&s, sizeof(struct Student), 1, fp)) {
        if (strcmp(s.rollNo, roll) == 0) {
            fseek(fp, -sizeof(struct Student), SEEK_CUR);

            if (!readName("Enter New Name: ", s.name, sizeof(s.name))) {
                printf("Error reading name.\n");
                fclose(fp);
                return;
            }
            if (!readFloat("Enter New Marks: ", &s.marks)) {
                printf("Error reading marks.\n");
                fclose(fp);
                return;
            }
            s.grade = calculateGrade(s.marks);
            fwrite(&s, sizeof(struct Student), 1, fp);
            found = 1;
            printf("Student updated successfully.\n");
            break;
        }
    }
    if (!found) {
        printf("Student not found.\n");
    }
    fclose(fp);
}
```

```
void deleteStudent() {
    FILE *fp = fopen("students.dat", "rb");
    if (fp == NULL) {
        printf("Error opening file!\n");
        return;
    }

    FILE *temp = fopen("temp.dat", "wb");
    if (temp == NULL) {
        printf("Error opening temporary file!\n");
        fclose(fp);
        return;
    }

    struct Student s;
    char roll[20];
    int found = 0;
    if (!readRollNumber("Enter Roll Number to Delete: ", roll, sizeof(roll))) {
        printf("Error reading roll number.\n");
        fclose(fp);
        fclose(temp);
        return;
    }
    while (fread(&s, sizeof(struct Student), 1, fp)) {
        if (strcmp(s.rollNo, roll) != 0) {
            fwrite(&s, sizeof(struct Student), 1, temp);
        } else {
            found = 1;
        }
    }
    fclose(fp);
    fclose(temp);
    remove("students.dat");
```

```
    rename("temp.dat", "students.dat");
    if (found) {
        printf("Student deleted successfully.\n");
    } else {
        printf("Student not found.\n");
    }
}
```

CHAPTER – 5

OUTPUT

```
*** Student Record Management System ***
1. Add Student
2. View All Students
3. Search Student by Roll Number
4. Update Student
5. Delete Student
6. Exit
Enter your choice: 1
Enter Roll Number: 24bbca054054
Enter Name: harshit
Enter Marks: 98
Student added successfully.
```

Fig. 5.1 Choice Menu (Add Student)

```
*** Student Record Management System ***
1. Add Student
2. View All Students
3. Search Student by Roll Number
4. Update Student
5. Delete Student
6. Exit
Enter your choice: 2

--- All Students ---
Roll No: 3b
Name: pooja
Marks: 80.00
Grade: B

Roll No: 24bbca054054
Name: harshit
Marks: 98.00
Grade: A
```

Fig. 5.2 View All Students

```
*** Student Record Management System ***
1. Add Student
2. View All Students
3. Search Student by Roll Number
4. Update Student
5. Delete Student
6. Exit
Enter your choice: 3
Enter Roll Number to Search: 24bbca054054

--- Student Found ---
Roll No: 24bbca054054
Name: harshit
Marks: 98.00
Grade: A
```

Fig. 5.3 Search Student by Roll Number

```
*** Student Record Management System ***
1. Add Student
2. View All Students
3. Search Student by Roll Number
4. Update Student
5. Delete Student
6. Exit
Enter your choice: 4
Enter Roll Number to Update: 24bbca054054
Enter New Name: harsh
Enter New Marks: 99
Student updated successfully.
```

Fig. 5.4 Update Student

```
*** Student Record Management System ***
1. Add Student
2. View All Students
3. Search Student by Roll Number
4. Update Student
5. Delete Student
6. Exit
Enter your choice: 5
Enter Roll Number to Delete: 24bbca054054
Student deleted successfully.
```

Fig. 5.5 Delete Student

CHAPTER – 6

CONCLUSION

The Student Record Management System developed in C successfully demonstrates how core programming concepts like structures, file handling, functions, arrays, and strings can be integrated to create a functional and efficient application. By structuring student data with fields such as name, roll number, marks, and grade, the system allows for clear and organized data representation. The use of arrays and strings simplifies input handling, while functions ensure modularity and code reusability throughout the program. This system provides essential features like adding, updating, deleting, and searching student records, making it practical for basic academic or administrative use. File handling plays a crucial role by enabling data persistence, allowing student records to be stored and retrieved even after the program is closed. This ensures that the system mimics real-world applications where data integrity and long-term storage are vital.

The Student Management System is an essential tool for modern educational institutions aiming to improve their administrative and academic processes. It not only helps in maintaining accurate student records but also simplifies tasks such as attendance tracking, grade management, and communication. By reducing manual work and paperwork, the system saves valuable time for teachers and administrative staff. Moreover, it provides easy access to information, which helps in better decision-making and academic planning. As education becomes more digital and data-driven, such systems are becoming increasingly important for ensuring smooth, transparent, and efficient management. Overall, the Student Management System contributes to a smarter, more organized, and future-ready learning environment.

Overall, this project not only fulfills the requirements of a simple database management system but also enhances the understanding of fundamental programming techniques in C. It serves as a solid foundation for building more advanced systems in the future, possibly integrating GUI elements, databases, or web-based interfaces. This experience reinforces the importance of combining logic, data handling, and structured programming in solving real-world problems effectively.

CHAPTER – 7

REFERENCE

1. Kernighan, B. W., & Ritchie, D. M. (1988). The C Programming Language (2nd ed.). Prentice Hall.
2. Kanetkar, Y. (2018). Let Us C (16th ed.). BPB Publications.
3. GeeksforGeeks. (n.d.). C Programming. Retrieved from https://www.geeksforgeeks.org
4. TutorialsPoint. (n.d.). C Language Tutorials. Retrieved from https://www.tutorialspoint.com/cprogramming/
5. Neso Academy. (n.d.). C Programming Tutorials [YouTube Channel]. Retrieved from https://www.youtube.com/user/nesoacademy
6. https://www.onlinegdb.com/online_c_compiler
7. OnlineGDB – Online C Compiler & Debugger https://www.onlinegdb.com/online_c_compiler
   (Run, debug, and share your code when the college lab PCs rebel.)
8. GitHub – "Student-Management-System-in-C" Open-Source Projects Example repo: https://github.com/safwan-mohammed/Student-Management-System-C
   (Reference for version control structure and README formatting.)
9. Stack Overflow – "C File Handling" Tag https://stackoverflow.com/questions/tagged/c+file-io
   (Crowd-sourced solutions to hair-tearing segmentation faults.)
10. IEEE Xplore Digital Library – "Student Information Systems" Research Papers https://ieeexplore.ieee.org
    (Great for sprinkling peer-reviewed authority over your abstract and conclusion.)