PROJECT BRIEF

PROJECT NAME:	College Administration System
PROJECT MANAGER:	**PROJECT DATE:**

EXECUTIVE SUMMARY

- This College Administration System aims to streamline various administrative tasks within an educational institution. It encompasses student records, faculty and staff management, and student fees management. Key features include student enrollment, academic record tracking, employee management, fee collection, and robust reporting capabilities. This system enhances efficiency, improves communication, and provides valuable insights for informed decision-making within the college.

INTRODUCTION

- This project focuses on developing a "College Administration System" – a software application designed to streamline and automate various administrative tasks within an educational institution.

OBJECTIVES

1. Automate: Streamline administrative tasks.
2. Centralize: Organize data for easy access.
3. Improve: Enhance communication and collaboration.
4. Support: Aid in informed decision-making.

DESIGN APPROACH

1. Modular Design: Divide the system into distinct modules (Student, Faculty/Staff, Fees) for better organization and maintainability.
2. User-Centric Design: Prioritize user experience with an intuitive and user-friendly interface.
3. Data-Driven Approach: Emphasize data accuracy and integrity through robust data validation and secure storage.
4. Scalability and Flexibility: Design the system to accommodate future growth and potential changes in requirements.

CONCLUSION

- This College Administration System aims to modernize college operations by automating tasks, improving data management, and enhancing communication. By implementing this system, educational institutions can streamline processes, improve efficiency, and ultimately enhance the overall educational experience for students, faculty, and staff.

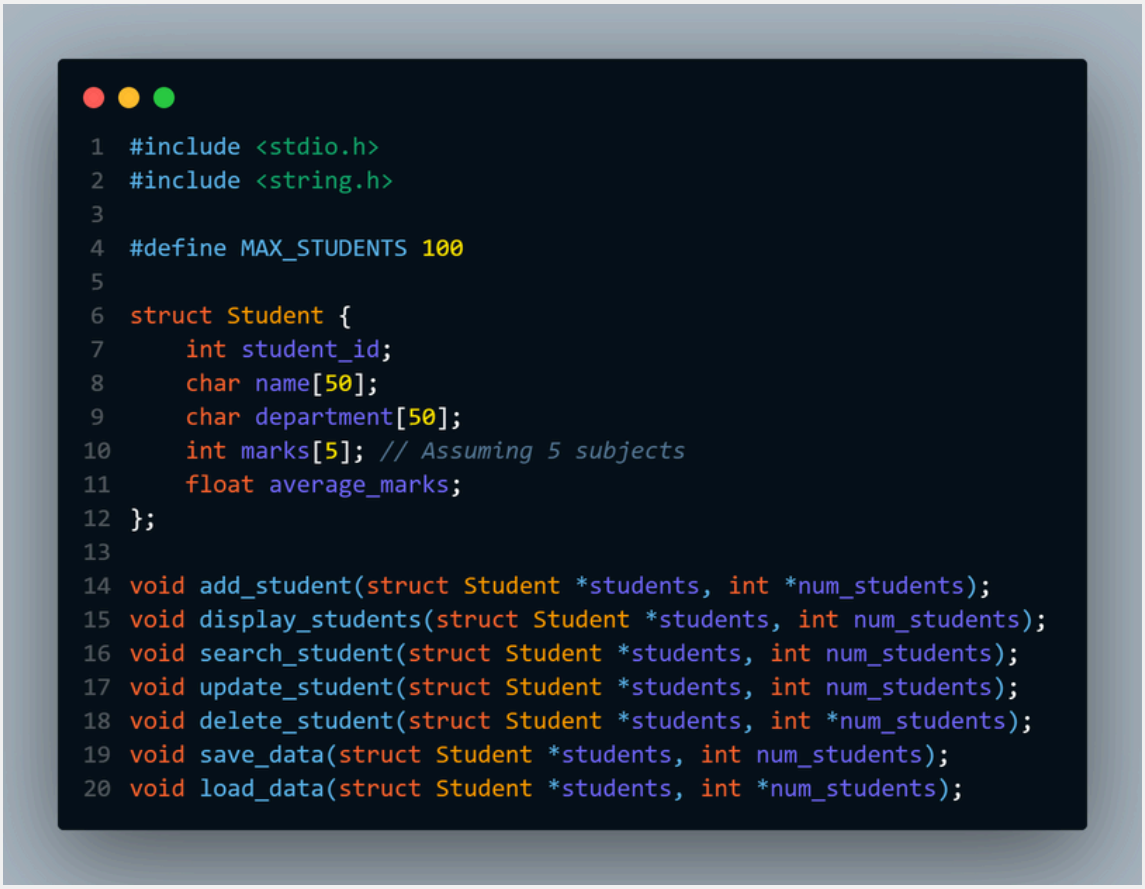
Source Codes

1. StdRecordManager.C
2. Fee_Manager.C
3. Faculty_Staff_Rec_Manager.C

Text Files Used For Storing Data

- **student_data.txt**
- **student_fees_data.txt**
- **employee_data.txt**

StdRecordManager.c



```c
#include <stdio.h>
#include <string.h>

#define MAX_STUDENTS 100

struct Student {
    int student_id;
    char name[50];
    char department[50];
    int marks[5]; // Assuming 5 subjects
    float average_marks;
};

void add_student(struct Student *students, int *num_students);
void display_students(struct Student *students, int num_students);
void search_student(struct Student *students, int num_students);
void update_student(struct Student *students, int num_students);
void delete_student(struct Student *students, int *num_students);
void save_data(struct Student *students, int num_students);
void load_data(struct Student *students, int *num_students);
```

```c
int main() {
    struct Student students[MAX_STUDENTS];
    int num_students = 0;
    int choice;

    // Load data from file on program start
    load_data(students, &num_students);

    do {
        printf("\n\nCollege Administration System\n");
        printf("1. Add Student\n");
        printf("2. Display All Students\n");
        printf("3. Search Student\n");
        printf("4. Update Student\n");
        printf("5. Delete Student\n");
        printf("6. Save Data\n");
        printf("7. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                add_student(students, &num_students);
                break;
            case 2:
                display_students(students, num_students);
                break;
            case 3:
                search_student(students, num_students);
                break;
            case 4:
                update_student(students, num_students);
                break;
            case 5:
                delete_student(students, &num_students);
                break;
            case 6:
                save_data(students, num_students);
                break;
            case 7:
                printf("Exiting...\n");
                break;
            default:
                printf("Invalid choice.\n");
        }
    } while (choice != 7);

    return 0;
}
```

```c
void add_student(struct Student *students, int *num_students) {
    if (*num_students >= MAX_STUDENTS) {
        printf("Maximum number of students reached.\n");
        return;
    }

    printf("\nEnter Student ID: ");
    scanf("%d", &students[*num_students].student_id);

    printf("Enter Student Name: ");
    scanf(" %[^\n]", students[*num_students].name); // Read full name with spaces

    printf("Enter Department: ");
    scanf(" %[^\n]", students[*num_students].department);

    printf("Enter Marks in 5 Subjects:\n");
    for (int i = 0; i < 5; i++) {
        printf("Subject %d: ", i + 1);
        scanf("%d", &students[*num_students].marks[i]);
    }

    // Calculate average marks
    float total_marks = 0;
    for (int i = 0; i < 5; i++) {
        total_marks += students[*num_students].marks[i];
    }
    students[*num_students].average_marks = total_marks / 5;

    (*num_students)++;
    printf("Student added successfully!\n");
}
```

```c
void display_students(struct Student *students, int num_students) {
    if (num_students == 0) {
        printf("No students found.\n");
        return;
    }

    printf("\nStudent Details:\n");
    printf("-------------------------------------------------\n");
    printf("ID\tName\t\tDepartment\tAverage Marks\n");
    printf("-------------------------------------------------\n");
    for (int i = 0; i < num_students; i++) {
        printf("%d\t%s\t\t%s\t\t%.2f\n", students[i].student_id, students[i].name, students[i].department, students[i].average_marks);
    }
    printf("-------------------------------------------------\n");
}
```

```c
void search_student(struct Student *students, int num_students) {
    int search_id;
    int found = 0;

    printf("Enter Student ID to search: ");
    scanf("%d", &search_id);

    for (int i = 0; i < num_students; i++) {
        if (students[i].student_id == search_id) {
            printf("\nStudent Found:\n");
            printf("ID: %d\n", students[i].student_id);
            printf("Name: %s\n", students[i].name);
            printf("Department: %s\n", students[i].department);
            printf("Average Marks: %.2f\n", students[i].average_marks);
            found = 1;
            break;
        }
    }

    if (!found) {
        printf("Student not found.\n");
    }
}
```

```c
void update_student(struct Student *students, int num_students) {
    int update_id;
    int found = 0;

    printf("Enter Student ID to update: ");
    scanf("%d", &update_id);

    for (int i = 0; i < num_students; i++) {
        if (students[i].student_id == update_id) {
            printf("Enter New Student Name: ");
            scanf(" %[^\n]", students[i].name);

            printf("Enter New Department: ");
            scanf(" %[^\n]", students[i].department);

            printf("Enter New Marks in 5 Subjects:\n");
            for (int j = 0; j < 5; j++) {
                printf("Subject %d: ", j + 1);
                scanf("%d", &students[i].marks[j]);
            }

            // Recalculate average marks
            float total_marks = 0;
            for (int j = 0; j < 5; j++) {
                total_marks += students[i].marks[j];
            }
            students[i].average_marks = total_marks / 5;

            printf("Student updated successfully!\n");
            found = 1;
            break;
        }
    }

    if (!found) {
        printf("Student not found.\n");
    }
}
```

```c
void delete_student(struct Student *students, int *num_students) {
    int delete_id;
    int found = 0;

    printf("Enter Student ID to delete: ");
    scanf("%d", &delete_id);

    for (int i = 0; i < *num_students; i++) {
        if (students[i].student_id == delete_id) {
            for (int j = i; j < *num_students - 1; j++) {
                students[j] = students[j + 1];
            }
            (*num_students)--;
            printf("Student deleted successfully!\n");
            found = 1;
            break;
        }
    }

    if (!found) {
        printf("Student not found.\n");
    }
}
```

```c
void save_data(struct Student *students, int num_students) {
    FILE *fp;
    fp = fopen("student_data.txt", "wb"); // Open file in binary write mode
    if (fp == NULL) {
        printf("Error opening file for writing.\n");
        return;
    }

    fwrite(&num_students, sizeof(int), 1, fp); // Write number of students
    fwrite(students, sizeof(struct Student), num_students, fp); // Write student data

    fclose(fp);
    printf("Data saved successfully!\n");
}
```

```c
void load_data(struct Student *students, int *num_students) {
    FILE *fp;
    fp = fopen("student_data.txt", "rb"); // Open file in binary read mode
    if (fp == NULL) {
        printf("No existing data found. Starting with an empty database.\n");
        return;
    }

    fread(num_students, sizeof(int), 1, fp); // Read number of students
    fread(students, sizeof(struct Student), *num_students, fp); // Read student data

    fclose(fp);
    printf("Data loaded successfully!\n");
}
```

Fee_Manager.C

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

#define MAX_STUDENTS 100
#define MAX_FEES 5

struct Student {
    int student_id;
    char name[50];
    float fees_paid;
    float total_fees;
    struct Fee {
        char fee_name[50];
        float amount;
        int is_paid; // 1 for paid, 0 for unpaid
    } fees[MAX_FEES];
};

void add_student(struct Student *students, int *num_students);
void add_fee(struct Student *students, int student_index);
void pay_fee(struct Student *students, int student_index);
void view_student_fees(struct Student *students, int student_index);
void view_all_students(struct Student *students, int num_students);
void select_student(struct Student *students, int num_students, int *student_index);
void save_data(struct Student *students, int num_students);
void load_data(struct Student *students, int *num_students);
```

```c
int main() {
    struct Student students[MAX_STUDENTS];
    int num_students = 0;
    int student_index = -1;
    int choice;

    load_data(students, &num_students);

    do {
        printf("\n\nStudent Fees Management System\n");
        printf("1. Add Student\n");
        printf("2. Select Student\n");
        printf("3. Add Fee\n");
        printf("4. Pay Fee\n");
        printf("5. View Student Fees\n");
        printf("6. View All Students\n");
        printf("7. Save Data\n");
        printf("8. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                add_student(students, &num_students);
                break;
            case 2:
                select_student(students, num_students, &student_index);
                break;
            case 3:
                if (student_index != -1) {
                    add_fee(students, student_index);
                } else {
                    printf("Please select a student first.\n");
                }
                break;
            case 4:
                if (student_index != -1) {
                    pay_fee(students, student_index);
                } else {
                    printf("Please select a student first.\n");
                }
                break;
            case 5:
                if (student_index != -1) {
                    view_student_fees(students, student_index);
                } else {
                    printf("Please select a student first.\n");
                }
                break;
            case 6:
                view_all_students(students, num_students);
                break;
            case 7:
                save_data(students, num_students);
                break;
            case 8:
                printf("Exiting...\n");
                break;
            default:
                printf("Invalid choice.\n");
        }
    } while (choice != 8);

    return 0;
}
```

```c
void add_student(struct Student *students, int *num_students) {
    if (*num_students >= MAX_STUDENTS) {
        printf("Maximum number of students reached.\n");
        return;
    }

    printf("\nEnter Student ID: ");
    scanf("%d", &students[*num_students].student_id);

    printf("Enter Student Name: ");
    scanf(" %[^\n]", students[*num_students].name);

    students[*num_students].fees_paid = 0.0;
    students[*num_students].total_fees = 0.0;

    // Initialize fees to zero initially
    for (int i = 0; i < MAX_FEES; i++) {
        strcpy(students[*num_students].fees[i].fee_name, "");
        students[*num_students].fees[i].amount = 0.0;
        students[*num_students].fees[i].is_paid = 0;
    }

    (*num_students)++;
    printf("Student added successfully!\n");
}
```

```c
void add_fee(struct Student *students, int student_index) {
    int fee_index = -1;
    for (int i = 0; i < MAX_FEES; i++) {
        if (strlen(students[student_index].fees[i].fee_name) == 0) {
            fee_index = i;
            break;
        }
    }

    if (fee_index == -1) {
        printf("Maximum number of fees for this student reached.\n");
        return;
    }

    printf("Enter Fee Name: ");
    scanf(" %[^\n]", students[student_index].fees[fee_index].fee_name);

    printf("Enter Fee Amount: ");
    scanf("%f", &students[student_index].fees[fee_index].amount);

    students[student_index].total_fees += students[student_index].fees[fee_index].amount;

    printf("Fee added successfully!\n");
}
```

```c
void pay_fee(struct Student *students, int student_index) {
    int fee_index;

    printf("Enter Fee Index to mark as paid (starting from 1): ");
    scanf("%d", &fee_index);

    if (fee_index < 1 || fee_index > MAX_FEES ||
        strlen(students[student_index].fees[fee_index - 1].fee_name) == 0) {
        printf("Invalid fee index.\n");
        return;
    }

    if (students[student_index].fees[fee_index - 1].is_paid == 1) {
        printf("Fee is already paid.\n");
        return;
    }

    students[student_index].fees[fee_index - 1].is_paid = 1;
    students[student_index].fees_paid += students[student_index].fees[fee_index - 1].amount;

    printf("Fee payment recorded successfully!\n");
}
```

```c
void view_student_fees(struct Student *students, int student_index) {
    printf("\nFee Details for Student ID: %d\n", students[student_index].student_id);
    printf("-------------------------------------------------\n");
    printf("Fee Name\t\tAmount\t\tPaid\n");
    printf("-------------------------------------------------\n");
    for (int i = 0; i < MAX_FEES; i++) {
        if (strlen(students[student_index].fees[i].fee_name) > 0) {
            printf("%-20s\t%.2f\t%s\n",
                   students[student_index].fees[i].fee_name,
                   students[student_index].fees[i].amount,
                   students[student_index].fees[i].is_paid ? "Yes" : "No");
        }
    }
    printf("-------------------------------------------------\n");
    printf("Total Fees: %.2f\n", students[student_index].total_fees);
    printf("Fees Paid: %.2f\n", students[student_index].fees_paid);
    printf("Outstanding Fees: %.2f\n", students[student_index].total_fees - students[student_index].fees_paid);
}
```

```c
void view_all_students(struct Student *students, int num_students) {
    if (num_students == 0) {
        printf("No students found.\n");
        return;
    }

    printf("\nStudent List:\n");
    printf("-----------------------------------------------------------------------\n");
    printf("ID\tName\t\t\tTotal Fees\t\tFees Paid\n");
    printf("-----------------------------------------------------------------------\n");
    for (int i = 0; i < num_students; i++) {
        printf("%d\t%-20s\t%.2f\t\t%.2f\n",
               students[i].student_id,
               students[i].name,
               students[i].total_fees,
               students[i].fees_paid);
    }
    printf("-----------------------------------------------------------------------\n");
}
```

```c
void select_student(struct Student *students, int num_students, int *student_index) {
    int search_id;
    int found = 0;

    printf("Enter Student ID to select: ");
    scanf("%d", &search_id);

    for (int i = 0; i < num_students; i++) {
        if (students[i].student_id == search_id) {
            *student_index = i;
            printf("Student selected successfully!\n");
            found = 1;
            break;
        }
    }

    if (!found) {
        printf("Student not found.\n");
        *student_index = -1; // Reset student_index if not found
    }
}
```

```c
void save_data(struct Student *students, int num_students) {
    FILE *fp;
    fp = fopen("student_fees_data.txt", "wb");
    if (fp == NULL) {
        printf("Error opening file for writing.\n");
        return;
    }

    fwrite(&num_students, sizeof(int), 1, fp);
    fwrite(students, sizeof(struct Student), num_students, fp);

    fclose(fp);
    printf("Data saved successfully!\n");
}
```

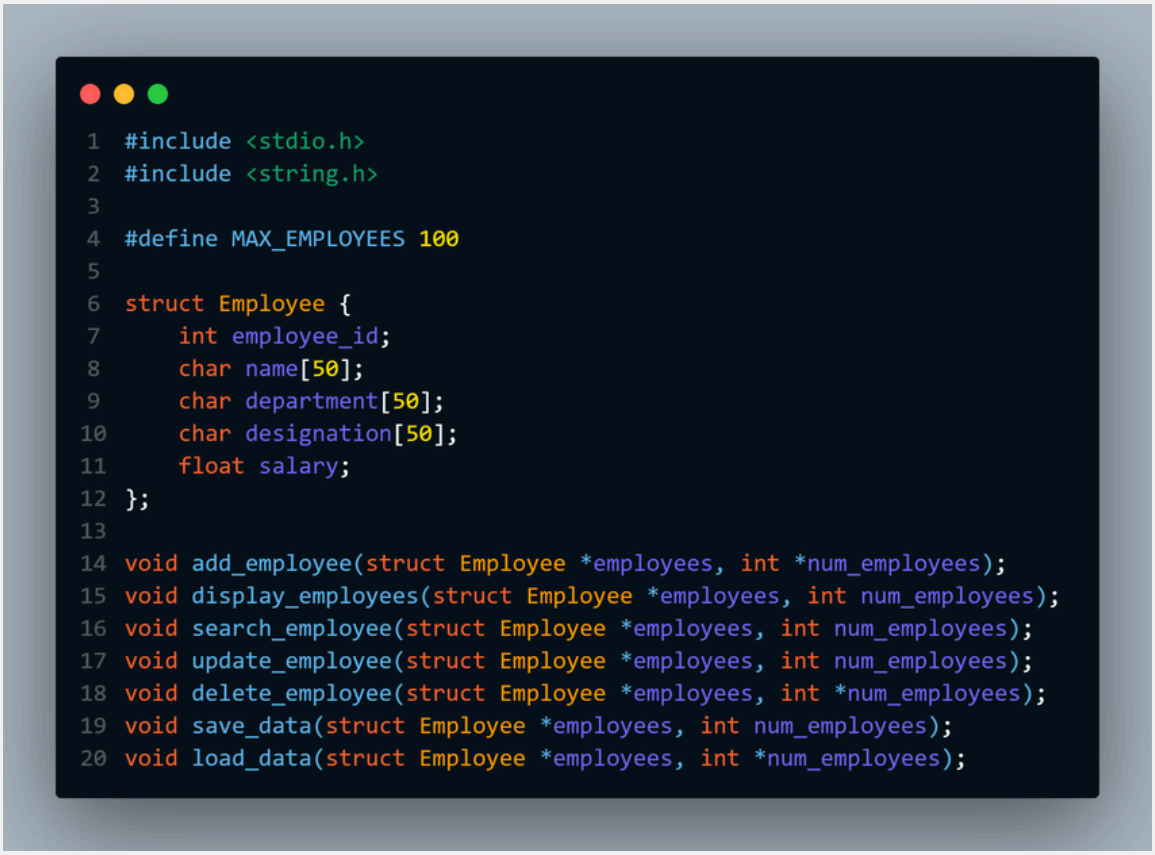
```c
void load_data(struct Student *students, int *num_students) {
    FILE *fp;
    fp = fopen("student_fees_data.txt", "rb");

    if (fp == NULL) {
        printf("No existing data found. Starting with an empty database.\n");
        *num_students = 0;
        return;
    }

    fread(num_students, sizeof(int), 1, fp);
    fread(students, sizeof(struct Student), *num_students, fp);

    fclose(fp);
    printf("Data loaded successfully!\n");
}
```

Faculty_Staff_Rec_Manager.C



```c
#include <stdio.h>
#include <string.h>

#define MAX_EMPLOYEES 100

struct Employee {
    int employee_id;
    char name[50];
    char department[50];
    char designation[50];
    float salary;
};

void add_employee(struct Employee *employees, int *num_employees);
void display_employees(struct Employee *employees, int num_employees);
void search_employee(struct Employee *employees, int num_employees);
void update_employee(struct Employee *employees, int num_employees);
void delete_employee(struct Employee *employees, int *num_employees);
void save_data(struct Employee *employees, int num_employees);
void load_data(struct Employee *employees, int *num_employees);
```

```c
int main() {
    struct Employee employees[MAX_EMPLOYEES];
    int num_employees = 0;
    int choice;

    // Load data from file on program start
    load_data(employees, &num_employees);

    do {
        printf("\n\nFaculty and Staff Management System\n");
        printf("1. Add Employee\n");
        printf("2. Display All Employees\n");
        printf("3. Search Employee\n");
        printf("4. Update Employee\n");
        printf("5. Delete Employee\n");
        printf("6. Save Data\n");
        printf("7. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                add_employee(employees, &num_employees);
                break;
            case 2:
                display_employees(employees, num_employees);
                break;
            case 3:
                search_employee(employees, num_employees);
                break;
            case 4:
                update_employee(employees, num_employees);
                break;
            case 5:
                delete_employee(employees, &num_employees);
                break;
            case 6:
                save_data(employees, num_employees);
                break;
            case 7:
                printf("Exiting...\n");
                break;
            default:
                printf("Invalid choice.\n");
        }
    } while (choice != 7);

    return 0;
}
```

```c
void add_employee(struct Employee *employees, int *num_employees) {
    if (*num_employees >= MAX_EMPLOYEES) {
        printf("Maximum number of employees reached.\n");
        return;
    }

    printf("\nEnter Employee ID: ");
    scanf("%d", &employees[*num_employees].employee_id);

    printf("Enter Employee Name: ");
    scanf(" %[^\n]", employees[*num_employees].name); // Read full name with spaces

    printf("Enter Department: ");
    scanf(" %[^\n]", employees[*num_employees].department);

    printf("Enter Designation: ");
    scanf(" %[^\n]", employees[*num_employees].designation);

    printf("Enter Salary: ");
    scanf("%f", &employees[*num_employees].salary);

    (*num_employees)++;
    printf("Employee added successfully!\n");
}
```

```c
void display_employees(struct Employee *employees, int num_employees) {
    if (num_employees == 0) {
        printf("No employees found.\n");
        return;
    }

    printf("\nEmployee Details:\n");
    printf("-----------------------------------------------------------\n");
    printf("ID\tName\t\tDepartment\tDesignation\tSalary\n");
    printf("-----------------------------------------------------------\n");
    for (int i = 0; i < num_employees; i++) {
        printf("%d\t%s\t\t%s\t\t%s\t%.2f\n", employees[i].employee_id, employees[i].name, employees[i].department, employees[i].designation, employees[i].salary);
    }
    printf("-----------------------------------------------------------\n");
}
```

```c
void search_employee(struct Employee *employees, int num_employees) {
    int search_id;
    int found = 0;

    printf("Enter Employee ID to search: ");
    scanf("%d", &search_id);

    for (int i = 0; i < num_employees; i++) {
        if (employees[i].employee_id == search_id) {
            printf("\nEmployee Found:\n");
            printf("ID: %d\n", employees[i].employee_id);
            printf("Name: %s\n", employees[i].name);
            printf("Department: %s\n", employees[i].department);
            printf("Designation: %s\n", employees[i].designation);
            printf("Salary: %.2f\n", employees[i].salary);
            found = 1;
            break;
        }
    }

    if (!found) {
        printf("Employee not found.\n");
    }
}
```

```c
void update_employee(struct Employee *employees, int num_employees) {
    int update_id;
    int found = 0;

    printf("Enter Employee ID to update: ");
    scanf("%d", &update_id);

    for (int i = 0; i < num_employees; i++) {
        if (employees[i].employee_id == update_id) {
            printf("Enter New Employee Name: ");
            scanf(" %[^\n]", employees[i].name);

            printf("Enter New Department: ");
            scanf(" %[^\n]", employees[i].department);

            printf("Enter New Designation: ");
            scanf(" %[^\n]", employees[i].designation);

            printf("Enter New Salary: ");
            scanf("%f", &employees[i].salary);

            printf("Employee updated successfully!\n");
            found = 1;
            break;
        }
    }

    if (!found) {
        printf("Employee not found.\n");
    }
}
```

```c
void delete_employee(struct Employee *employees, int *num_employees) {
    int delete_id;
    int found = 0;

    printf("Enter Employee ID to delete: ");
    scanf("%d", &delete_id);

    for (int i = 0; i < *num_employees; i++) {
        if (employees[i].employee_id == delete_id) {
            for (int j = i; j < *num_employees - 1; j++) {
                employees[j] = employees[j + 1];
            }
            (*num_employees)--;
            printf("Employee deleted successfully!\n");
            found = 1;
            break;
        }
    }

    if (!found) {
        printf("Employee not found.\n");
    }
}
```

```c
void save_data(struct Employee *employees, int num_employees) {
    FILE *fp;
    fp = fopen("employee_data.txt", "wb"); // Open file in binary write mode
    if (fp == NULL) {
        printf("Error opening file for writing.\n");
        return;
    }

    fwrite(&num_employees, sizeof(int), 1, fp); // Write number of employees
    fwrite(employees, sizeof(struct Employee), num_employees, fp); // Write employee data

    fclose(fp);
    printf("Data saved successfully!\n");
}
```

```c
void load_data(struct Employee *employees, int *num_employees) {
    FILE *fp;
    fp = fopen("employee_data.txt", "rb"); // Open file in binary read mode
    if (fp == NULL) {
        printf("No existing data found. Starting with an empty database.\n");
        return;
    }

    fread(num_employees, sizeof(int), 1, fp); // Read number of employees
    fread(employees, sizeof(struct Employee), *num_employees, fp); // Read employee data

    fclose(fp);
    printf("Data loaded successfully!\n");
}
```