**Employee Management**

**System**

**Submitted by**

**Name of the Students: Saniya Rahman**

**Enrolment Number:** 12022002018060

**Section:** F

**Class Roll Number:** 60  
**Stream:** CSBS

**Subject:** Programming for Problem-Solving using C

**Subject Code:** ESC103 (Pr)

**Department:** Basic Science and Humanities

Under the supervision of

**Swarnendu Ghosh**

**Academic Year: 2022-26**

PROJECT REPORT SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE FIRST SEMESTER



**DEPARTMENT OF BASIC SCIENCE AND HUMANITIES**

**INSTITUTE OF ENGINEERING AND MANAGEMENT, KOLKATA**



**CERTIFICATE OF RECOMMENDATION**

We hereby recommend that the project prepared under our supervision by Saniya Rahman entitled Employee Management System accepted in partial fulfillment of the requirements for the degree of partial fulfillment of the first semester.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Head of the Department Project Supervisor

Basic Sciences and Humanities

IEM, Kolkata

# Introduction:

An employee management program using C language is a software application that helps businesses and organizations manage their employee records. The program allows administrators to enter, view, modify, and delete employee information, such as their names, IDs, positions, and salaries.  
  
Using C language, a developer can create a robust and efficient program that can handle large amounts of employee data. The program can also perform various operations on the data, such as sorting, searching, and generating reports

**Project Scope:**

The scope of an employee management program using C language can vary depending on the needs of the business or organization. However, some of the main features that could be included in such a program are:  
1.Add new employee records to the system  
2.Display all employee records in the system  
3.Find employee records by ID  
4.Modify existing employee records  
5.Delete existing employee records  
6.Sort employee records by name, ID, position, or salary  
7.Generate reports on employee data, such as salary distribution or employee turnover rate  
8.Search for employee records based on certain criteria, such as position or salary range  
  
 **Program Design:**

The program design for our scientific calculator involves several steps. These are:

1. User interface design: The user interface design should be clear and simple, with easy-to-understand instructions and prompts. The user should be able to easily select the desired operation using a menu system.
2. Function design: The program should have several functions to handle various operations. These functions should take input from the user, perform the necessary calculations, and output the result.
3. Error handling: The program should have error handling mechanisms to prevent any erroneous inputs from causing a program crash. The program should display appropriate error messages and prompt the user to enter valid input.

**Program Implementation:**

The program implementation for our scientific calculator involves several steps. These are:

1. User interface implementation: The user interface should be implemented using simple text prompts and menus. The user should be able to easily navigate through the menus and select the desired operation.
2. Function implementation: The various functions for the calculator, such as addition, subtraction, multiplication, division, exponentiation, square root, sine, cosine, tangent, and logarithm should be implemented using the appropriate mathematical formulas and programming constructs.
3. Error handling implementation: The program should have error handling mechanisms in place to prevent any erroneous inputs from causing a program crash. The program should display appropriate error messages and prompt the user to enter valid input.

**Variables:**

1. Employee structure variable: This variable would be used to hold information about a single employee, such as their name, ID, position, and salary.  
2. Array of employee structure variables: This variable would be used to hold multiple employee records.

3. Counter variable: This variable could be used to keep track of how many employees are currently stored in the array.  
4. Temporary variables: These variables would be used to hold temporary data while performing operations on the employee records. For example, when adding a new employee, you would use temporary variables to hold the input from the user before adding it to the array.  
5. Index variable: This variable would be used to keep track of the current position in the array when iterating over the employee records.

**Functions:**

1. add\_employee(): This function would be used to add a new employee record to the array. It would prompt the user to enter the employee's name, ID, position, and salary, and then add the new record to the array.  
2. display\_all\_employees(): This function would be used to display all employee records stored in the array.  
3. find\_employee\_by\_id(): This function would be used to find an employee record by their ID. It would prompt the user to enter an ID, and then search the array for a record with that ID. If found, it would return the index of the record in the array. If not found, it would return -1.  
4. modify\_employee(): This function would be used to modify an existing employee record. It would prompt the user to enter an ID, search for a record with that ID, and then allow the user to modify the name, position, or salary of the employee.

**Files:**

This program consists of a single C file, which contains the code for a employee management program. The file likely has a .c extension and could be named anything the programmer chose.

**Datasets:**

The program does not use any external datasets or files. All necessary information is contained within the code, including the operations and corresponding function definitions. The user input is provided via the command-line interface during program execution.

**Features:**

1. Employee Records Management: The program allows administrators to enter and manage employee records, such as their names, IDs, positions, and salaries.  
2. User Authentication and Authorization: The program can include a login system to ensure that only authorized users can access and modify employee data.  
3. Data Validation and Error Handling: The program can include data validation and error handling features to ensure that the data entered by users is accurate and that the program functions correctly.  
4. Employee Search and Retrieval: The program can allow administrators to search for employee records based on certain criteria, such as position or salary range, and retrieve relevant data.  
5. Employee Reports Generation: The program can generate reports on employee data, such as salary distribution or employee turnover rate, to help businesses make informed decisions about their workforce.  
6. Data Sorting and Filtering: The program can allow administrators to sort and filter employee data based on various criteria, such as name, ID, position, or salary.  
7. Backup and Restore: The program can include backup and restore features to ensure that employee data is safe and can be easily restored in case of data loss.  
8. User Interface: The program should provide a user-friendly interface that allows administrators to easily manage employee data and perform various operations.

# Programs

void add\_employee() {  
    struct Employee emp;  
    printf("Enter employee name: ");  
    scanf("%s", [emp.name](http://emp.name));  
    printf("Enter employee ID: ");  
    scanf("%d", &[emp.id](http://emp.id));  
    printf("Enter employee position: ");  
    scanf("%s", [emp.position](http://emp.position));  
    printf("Enter employee salary: ");  
    scanf("%f", &[emp.salary](http://emp.salary));  
    employees[num\_employees++] = emp;  
}  
  
void display\_all\_employees() {  
    if (num\_employees == 0) {  
        printf("No employees in the system.\n");  
        return;  
    }  
    printf("Employee records:\n");  
    for (int i = 0; i < num\_employees; i++) {  
        printf("Name: %s\n", employees[i].name);  
        printf("ID: %d\n", employees[i].id);  
        printf("Position: %s\n", employees[i].position);  
        printf("Salary: %.2f\n", employees[i].salary);  
    }  
}  
int find\_employee\_by\_id(int id) {  
    for (int i = 0; i < num\_employees; i++) {  
        if (employees[i].id == id) {  
            return i;  
        }  
    }  
    return -1;  
}  
void modify\_employee() {  
    int id;  
    printf("Enter employee ID to modify: ");  
    scanf("%d", &id);  
    int index = find\_employee\_by\_id(id);  
    if (index == -1) {  
        printf("No employee found with that ID.\n");  
        return;  
    }  
    printf("Enter new name (or press enter to keep current name): ");  
    char temp\_name[50];  
    scanf("%s", temp\_name);  
    if (strlen(temp\_name) > 0) {  
        strcpy(employees[index].name, temp\_name);  
    }  
    printf("Enter new position (or press enter to keep current position): ");  
    char temp\_position[50];  
    scanf("%s", temp\_position);  
    if (strlen(temp\_position) > 0) {  
        strcpy(employees[index].position, temp\_position);  
    }  
    printf("Enter new salary (or enter 0 to keep current salary): ");  
    float temp\_salary;  
    scanf("%f", &temp\_salary);  
    if (temp\_salary > 0) {  
        employees[index].salary = temp\_salary;  
    }  
    printf("Employee record updated.\n");  
}