# Employee Record System

*A Project Report by*

***Amey Ghatol 2314110016***

***Vansh Desai 2314110009***

***Simar Ahluwalia 231411004***

***Samriddhi Gupta 23140018***

*In partial fulfillment of the requirement*

*For the course of*

**BACHELOR OF TECHNOLOGY**

*In*

**COMPUTER SCIENCE AND ENGINEERING**

*Under the guidance of*

***Dr. Nisha Auti***


**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**BHARATI VIDYAPEETH (DEEMED TO BE UNIVERSITY)**

**COLLEGE OF ENGINEERING, PUNE- 43**

# BHARATI VIDYAPEETH (DEEMED TO BE UNIVERSITY)

# COLLEGE OF ENGINEERING, PUNE- 43

## CERTIFICATE

This is to certify that the Project Based Learning report titled ***Employee record system*** , submitted by **Amey Ghatol *2314110016* Vansh Desai *2314110009* Simar Ahluwalia *2314110004* Samriddhi Gupta *23140018*** , to the Bharati Vidyapeeth (Deemed to be University), College of Engineering, Pune - 43 for the award of the degree of **BACHELOR OF TECHNOLOGY** in Computer Science and Engineering is a bonafide record of the PBL work done by him/them under my supervision.

Place: Pune *Sign of Subject Teacher*

Date: 11 April 2024

# Overview

This report evaluates the Employee Management System code provided, which is designed to facilitate the management of employee records within an organization. The system allows users to perform various operations such as adding, searching, removing, modifying, and displaying employee details using a linked list data structure.

# Introduction

Employee management is vital for organizational efficiency. It involves overseeing various aspects of employee-related tasks such as record-keeping, performance monitoring, and resource allocation. Effective management of employees ensures streamlined operations, enhances productivity, and fosters a positive work environment.The Employee Management System serves the purpose of efficiently managing employee records. It provides a platform for performing key operations such as adding new employees, searching for specific records, modifying existing data, removing outdated entries, and displaying comprehensive employee information.The Employee Management Program aligns with the strategic objectives of organizations seeking to optimize their HR operations. By automating routine tasks, minimizing manual errors, and providing real-time insights into workforce dynamics, the program empowers HR professionals to focus on strategic initiatives such as talent development, succession planning, and employee engagement. Its user-friendly interface and intuitive functionalities make it an indispensable tool for HR departments striving to enhance organizational performance and employee satisfaction.The successful implementation of the Employee Management Program has the potential to significantly impact organizational efficiency and competitiveness. By streamlining HR processes, reducing administrative overheads, and enhancing data accuracy, the program enables businesses to allocate resources more effectively, respond rapidly to market changes, and maintain a competitive edge in today's dynamic business landscape. As a result, organizations can achieve greater agility, resilience, and innovation,

positioning themselves for long-term success and growth.The Employee Management Program presented here serves as a foundational tool for businesses to streamline their HR processes. Its scope encompasses a range of functionalities aimed at managing employee records efficiently.

# Use of Linked List

A linked list is a linear data structure used to store a collection of elements called nodes. Unlike arrays, which store elements in contiguous memory locations, linked lists use nodes that are dynamically allocated and connected via pointers.

Here are the key characteristics of a linked list:

**1. Nodes**

Each node in a linked list contains two parts: data and a reference (or pointer) to the next node in the sequence. The data part holds the actual value or payload, such as an integer, string, or complex data structure. The pointer part points to the next node in the sequence, forming the linkage between nodes.

**2. Head Pointer**

Linked lists typically have a head pointer, which points to the first node in the sequence. Through the head pointer, the entire linked list can be traversed sequentially by following the pointers from one node to the next.

**3. Dynamic Memory Allocation**

Linked lists allow for dynamic memory allocation, meaning nodes can be allocated or deallocated from memory as needed. This flexibility enables the linked list to grow or shrink dynamically based on the operations performed on it.

**4.Types of Linked Lists**

There are different types of linked lists, including singly linked lists, where each node has a single pointer to the next node, and doubly linked lists, where each node has pointers to both the next and previous nodes. Additionally, there are circular linked lists, where the last node points back to the first node, forming a circular structure.

# Its advantages

Specifically, the linked list serves as a linear data structure, offering several advantages for this application:

**1. Dynamic Memory Allocation**

Linked lists allow for dynamic memory allocation, enabling the system to efficiently allocate memory for employee records as needed. This dynamic nature ensures optimal memory usage, especially when the number of employee records varies over time.

**2. Insertion and Deletion Efficiency**

The linear nature of linked lists facilitates efficient insertion and deletion operations. When adding or removing employee records, linked lists offer constant-time complexity for insertion and deletion at the beginning of the list, which is suitable for the frequent operations required in an employee management system.

**3. Traversal Flexibility**

Linked lists support linear traversal, allowing the system to easily iterate through all employee records in sequence. This traversal flexibility is essential for operations such as displaying all employee records or searching for a specific employee by traversing the list sequentially.

**4. No Fixed Size Limitation**

Unlike array-based data structures that have a fixed size, linked lists can dynamically grow or shrink based on the number of employee records. This lack of a fixed size limitation ensures scalability, accommodating any number of employee records without the need to predefine the size of the data structure.

**5. Ease of Modification**

Linked lists provide ease of modification, allowing the system to efficiently modify employee records by updating pointers within the list. This flexibility simplifies operations such as modifying employee details or rearranging the sequence of employee records.

These things make linked lists well-suited for managing employee records in a dynamic and efficient manner.

# Use of stack

Stack is a linear data structure based on the LIFO ( last in - first out ) principle , which states that the data entry last inputted in the data structure is the first one to be accessed , giving it a similar characteristic to a real life stack which functions very similarly

In our case , Stack was implemented to record the admission of employees into the company log. It is so that the most recently logged employee can be easily removed from the log due to correction reasons or other factors depending upon the logger.

However there are many more applications of stack when it comes to employee logging which can be easily added or removed from the code system , due to the highly modifiable nature of the record system. There are some niche scenarios where stacks might be used in conjunction with other data structures within an ERS:

- **Undo/redo functionality**: If an ERS allows users to make changes to employee records (e.g., update salary, change department), a stack could be used to temporarily store the previous state of the record. This would enable an "undo" feature, allowing users to revert to the previous version. This scenario is similar to how some text editors use a stack to implement undo/redo functionality.

- **Temporary storage**: A stack could be used as a temporary storage mechanism for a small set of employee records during specific operations. For instance, when performing a mass update on a group of employees (e.g., applying a salary increase to a specific department), you might temporarily store the IDs of the affected employees in a stack for processing. However, this would be a very specific use case and wouldn't be central to the core functionality of the ERS.

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

#define MAX_NAME_LENGTH 50
#define MAX_DEPARTMENT_LENGTH 50
#define MAX_DESIGNATION_LENGTH 50

// Define the employee structure
struct employee {
    int id;
    char name[MAX_NAME_LENGTH];
    char department[MAX_DEPARTMENT_LENGTH];
    char designation[MAX_DESIGNATION_LENGTH];
    float salary;
};

// Define the linked list node structure
struct node {
    struct employee emp;
    struct node *next;
};
```

```
struct node *head = NULL;


// Function prototypes
struct node *createNode(struct employee emp);
void insertNode(struct employee emp);
void removeEmployee();
void modifyEmployee(int id);
void search(int id);
void displayAll();
void displayMenu();


int main() {
    int choice;
    struct employee newEmployee; // Declare here
    int searchId; // Declare here
    int removeId; // Declare here
    int modifyId; // Declare here
    do {
        displayMenu();
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
```

```
case 1:
    // Add an employee
    printf("Enter employee ID: ");
    scanf("%d", &newEmployee.id);
    printf("Enter employee name: ");
    scanf("%s", newEmployee.name);
    printf("Enter employee department: ");
    scanf("%s", newEmployee.department);
    printf("Enter employee designation: ");
    scanf("%s", newEmployee.designation);
    printf("Enter employee salary: ");
    scanf("%f", &newEmployee.salary);
    insertNode(newEmployee);
    break;
case 2:
    // Search for an employee
    printf("Enter the employee ID to search: ");
    scanf("%d", &searchId);
    search(searchId);
    break;
case 3:
    // Remove recently added employee
    removeEmployee(removeId);
```

```
                break;
            case 4:
                // Modify a record
                printf("Enter the employee ID to modify: ");
                scanf("%d", &modifyId);
                modifyEmployee(modifyId);
                break;
            case 5:
                // View all records
                displayAll();
                break;
            case 6:
                // Exit the program
                printf("Exiting the program...\n");
                exit(0);
            default:
                printf("Invalid choice. Please enter a valid
option.\n");
        }
    } while (1);

    return 0;
}
```

```
// Function to create a new node

struct node *createNode(struct employee emp) {

    struct node *new_node = (struct node *)malloc(sizeof(struct
node));

    if (new_node == NULL) {

        printf("Memory allocation failed.\n");

        exit(1);

    }

    new_node->emp = emp;

    new_node->next = NULL;

    return new_node;

}


// Function to insert a new node into the linked list

void insertNode(struct employee emp) {

    struct node *new_node = createNode(emp);

    if (head == NULL) {

        head = new_node;

    } else {

        new_node->next = head;

        head = new_node;

    }
```

```
}


// Function to search for an employee by ID
void search(int id) {
    struct node *ptr = head;
    while (ptr != NULL) {
        if (ptr->emp.id == id) {
            printf("Employee found:\n");
            printf("ID: %d\n", ptr->emp.id);
            printf("Name: %s\n", ptr->emp.name);
            printf("Department: %s\n", ptr->emp.department);
            printf("Designation: %s\n", ptr->emp.designation);
            printf("Salary: %.2f\n", ptr->emp.salary);
            return;
        }
        ptr = ptr->next;
    }
    printf("Employee with ID %d not found.\n", id);
}


// Function to remove an employee by ID
//Implementing stack by deleting the most recently added data to
the list
```

```
//Taking head as top pointer
void removeEmployee() {
    struct node *current = head;
    struct node *previous = NULL;
     head=head->next;
     free(current);
}

// Function to modify an employee record by ID
void modifyEmployee(int id) {
    struct node *ptr = head;
    while (ptr != NULL) {
        if (ptr->emp.id == id) {
            printf("Enter new employee name: ");
            scanf("%s", ptr->emp.name);
            printf("Enter new employee department: ");
            scanf("%s", ptr->emp.department);
            printf("Enter new employee designation: ");
            scanf("%s", ptr->emp.designation);
            printf("Enter new employee salary: ");
            scanf("%f", &ptr->emp.salary);
            printf("Employee record with ID %d modified
successfully.\n", id);
```

```
            return;
        }
        ptr = ptr->next;
    }
    printf("Employee with ID %d not found.\n", id);
}

// Function to display all employee records
void displayAll() {
    struct node *ptr = head;
    if (ptr == NULL) {
        printf("No employee records found.\n");
        return;
    }
    printf("Employee Records:\n");
    while (ptr != NULL) {
        printf("ID: %d\n", ptr->emp.id);
        printf("Name: %s\n", ptr->emp.name);
        printf("Department: %s\n", ptr->emp.department);
        printf("Designation: %s\n", ptr->emp.designation);
        printf("Salary: %.2f\n\n", ptr->emp.salary);
        ptr = ptr->next;
    }
```

```
}


// Function to display the menu options
void displayMenu() {
    printf("\nMenu:\n");
    printf("1. Add an employee\n");
    printf("2. Search for an employee\n");
    printf("3. Remove the recently added employee\n");
    printf("4. Modify a record\n");
    printf("5. View all records\n");
    printf("6. Exit\n");
}
```

# Code Overview:

The code is written in C and structured into several components:

Header Files and Definitions:

- Standard C header files like <stdio.h>, <stdlib.h>, and <string.h> are included.
- Preprocessor directives define maximum lengths for employee name, department, and designation.

Structures:

- Two structures are defined: employee to represent individual employees and node to create nodes for the linked list.

Global Variables:

- The global variable head is declared to serve as the head of the linked list.

Function Prototypes:

- Prototypes for functions such as createNode(), insertNode(), search(), removeEmployee(), modifyEmployee(), and displayAll() are declared.

Main Function:

- The main function provides a menu-driven interface for users to interact with the system.
- Users can select options to perform various operations on employee records.

Employee Management Functions:

- Functions like createNode(), insertNode(), search(), removeEmployee(), modifyEmployee(), and displayAll() implement functionalities to manage employee records.

Menu Display Function:

- The displayMenu() function displays the menu options for user selection.

# Functions

1. createNode(struct employee emp):

- This function creates a new node in the linked list to store an employee's data.
- It allocates memory for the node using `malloc()` and initializes its `emp` field with the provided employee data.
- If memory allocation fails, it prints an error message and exits the program.

2. insertNode(struct employee emp):

- This function inserts a new node representing an employee into the linked list.
- It first creates a new node using `createNode()` and then inserts it at the beginning of the list.
- If the list is empty, the new node becomes the head of the list; otherwise, it is linked to the existing head.

3. search(int id):

- This function searches for an employee record in the linked list based on the provided employee ID.
- It traverses the list sequentially, comparing each node's ID with the target ID until a match is found.
- If a matching record is found, it prints the details of the employee; otherwise, it notifies the user that the employee with the specified ID was not found.

4. removeEmployee(int id):

- This function removes a recently added employee record from the linked list.
- This function is an example of a stack where it follows the LIFO principle so that the last inserted data of an employee is removed at first.

5. modifyEmployee(int id):

- This function modifies an existing employee record in the linked list based on the provided employee ID.
- It traverses the list to find the node containing the specified ID and allows the user to input new details for the employee.
- After updating the employee's details, it prints a confirmation message; otherwise, it notifies the user that the employee with the specified ID was not found.

6. displayAll():

- This function displays details of all employee records stored in the linked list.
- It traverses the entire list sequentially, printing the details of each employee node.
- If the list is empty, it prints a message indicating that no employee records were found.

# Features:

***Detailed Structure Definitions:***

The code defines comprehensive structures for employees and linked list nodes, encompassing essential attributes and pointers for efficient data storage and retrieval. These structures serve as the backbone of the program, providing a robust framework for organizing and managing employee data effectively.

***Extensive Functionality Implementation:***

The program implements a wide array of functions to handle various aspects of employee data management. These functions encompass functionalities such as creating nodes, inserting new records, searching for specific employees, removing outdated entries, modifying existing records, and displaying comprehensive employee information, ensuring comprehensive coverage of essential operations.

***Thorough User Interaction Design:***

User interaction is facilitated through a meticulously designed menu-driven interface, which presents users with a range of options to choose from. The interface guides users through available functionalities, provides clear instructions, and ensures seamless navigation, enhancing user experience and promoting efficient utilization of the system's features.

***Strategic Code Modularization:***

The program adopts a strategic approach to code modularization, dividing related functionalities into separate modules for clarity and maintainability. Each module encapsulates a specific set of functionalities, promoting code reusability, facilitating future modifications, and simplifying debugging and troubleshooting efforts, thereby enhancing the overall robustness and scalability of the program.

# Space and time complexity

The space and time complexity of the provided Employee Management System code can be analyzed as follows:

**1. Space Complexity:**

- The space complexity of the code primarily depends on the number of employee records stored in the linked list and any additional memory used by variables and function calls.
- Each employee record consists of several fields, including ID, name, department, designation, and salary. Therefore, the space complexity of storing employee records is O(n), where n is the number of employees.
- Additionally, the space complexity may include the overhead associated with the linked list nodes, pointers, and any other variables used within functions.

Overall, the space complexity of the code is O(n), where n is the number of employee records stored in the linked list.

**2. Time Complexity:**

- The time complexity of the code varies depending on the specific operations performed on the employee records.
- Insertion and deletion operations in a linked list typically have a time complexity of O(1) when performed at the beginning or end of the list. Therefore, adding or removing an employee record is generally efficient.
- Searching for an employee record by ID involves traversing the linked list sequentially until the desired record is found. In the worst case, when the record is not found or is located at the end of the list, the time complexity is O(n), where n is the number of employees. Modifying an employee record also involves searching for the record by ID, followed by updating the record's fields. Therefore, the time complexity of modification is also O(n) in the worst case.

- Displaying all employee records requires traversing the entire linked list, resulting in a time complexity of O(n), where n is the number of employees.
- The time complexity of the main menu operations depends on the specific implementation of input/output operations, which typically have a constant time complexity.

Overall, the time complexity of the code is dominated by the sequential traversal of the linked list, resulting in O(n) time complexity for most operations involving employee records.

Therefore,the space complexity of the code is O(n), where n is the number of employee records, and the time complexity varies depending on the specific operation performed, with most operations having a time complexity of O(n) due to sequential traversal of the linked list.

# Strengths

*Ease of Use and Accessibility:*

The Employee Management System prioritizes simplicity and accessibility, making it suitable for users with varying levels of technical expertise. Its intuitive user interface, clear instructions, and straightforward functionalities empower users to navigate the system effortlessly and perform tasks with ease, thereby promoting user adoption and satisfaction.

*Efficient Resource Utilization:*

Leveraging linked list data structures, the program optimizes resource utilization by dynamically allocating memory and facilitating efficient insertion, deletion, and traversal operations. This ensures optimal utilization of system resources, minimizes memory overhead, and enables seamless scalability to accommodate growing datasets, thereby enhancing system performance and responsiveness.

*Flexibility and Customization Options:*

The program offers flexibility and customization options to meet diverse organizational requirements and preferences. Users can tailor the system to their specific needs by configuring settings, defining custom fields, and adapting functionalities to align with organizational workflows, thereby enhancing user satisfaction, productivity, and adoption rates.

*Scalability and Adaptability:*

Designed with scalability and adaptability in mind, the Employee Management System can accommodate varying database sizes and organizational growth trajectories. Its modular architecture, coupled with efficient data structures and algorithms, ensures scalability without compromising performance, enabling organizations to scale their operations seamlessly and adapt to evolving business needs.

# Limitations

**Insufficient Input Validation:**

The Employee Management System lacks comprehensive input validation mechanisms, leaving it vulnerable to errors resulting from invalid user inputs. Without robust validation, the program may accept incorrect or malformed data, leading to data corruption or unexpected behavior. Strengthening input validation is essential to ensure data integrity and prevent erroneous operations that could compromise the reliability of the system.

**Lack of Robust Error Handling:**

The program's error handling capabilities are limited, which can result in unexpected behavior or program crashes in error scenarios. Inadequate error handling makes it challenging for users to identify and address issues effectively, leading to frustration and decreased usability. Implementing robust error handling mechanisms, such as error logging and informative error messages, is essential to enhance the system's resilience and user experience.

**Memory Management Concerns:**

While the program utilizes dynamic memory allocation, it lacks sophisticated memory management mechanisms to deallocate memory when records are removed. This oversight can result in memory leaks over time, leading to inefficient resource utilization and potential performance degradation. Improving memory management strategies, such as implementing garbage collection or manual memory deallocation, is necessary to mitigate memory-related issues and ensure the program's long-term stability.

**Limited Scalability with Large Datasets:**

As the size of the employee database increases, the program's performance may degrade due to limitations in scalability. The use of a linked list data structure, while efficient for small to moderate-sized datasets, may become inefficient for handling large datasets. Without proper optimization techniques or alternative data structures, such as balanced trees or hash tables, the program may struggle to maintain acceptable performance levels with increasing data volumes.

**Dependency on User Input Accuracy:**

The effectiveness of the Employee Management System is heavily reliant on the accuracy and consistency of user inputs. In the absence of comprehensive input validation, the program is susceptible to errors caused by incorrect or inconsistent data entry. Users may unintentionally input invalid data or omit required fields, leading to data integrity issues and inaccurate results. Enhancing input validation mechanisms and incorporating data validation checks are imperative to mitigate the impact of user input errors and improve the reliability of the system.

# Conclusion

In conclusion, the report evaluates the Employee Management System code provided, highlighting its structure, functionality, and areas for improvement.

The code effectively utilizes a linked list data structure to manage employee records, providing dynamic memory allocation, efficient insertion and deletion operations, and linear traversal. However, there are areas for enhancement, including input validation, error handling, and data persistence.

Despite these areas for improvement, the code serves as a solid foundation for an Employee Management System, demonstrating effective use of a linked list to meet the requirements of managing employee records. By addressing the recommended improvements, such as implementing input validation and enhancing error handling, the system can be further optimized for robustness and usability.

Overall, the code presents a practical solution for managing employee records, showcasing the benefits of utilizing a linked list data structure in a real-world application. With continued refinement and development, the Employee Management System has the potential to become a valuable tool for organizations in efficiently managing their workforce.