**EMPLOYEE MANAGEMENT SYSTEM**

Project submitted to the

SRM University – AP, Andhra Pradesh

for the partial fulfillment of the requirements to award the degree of

**Bachelor of Technology**

In

**Computer Science and Engineering**

**School of Engineering and Sciences**

Submitted by

**K. MUTHYA PRASADH**(AP23110010339)

**M.VIGENSH VARMA** (AP23110010308)

**V.KARTHIK**(AP23110010298)

**D.PRANEETH VARMA** (AP23110010322)



Under the Guidance of

Prof. Kavitha rani karnena.

**SRM University–AP**

**Neerukonda, Mangalagiri, Guntur Andhra Pradesh – 522 240**

**[November, 2024]**

# Certificate

Date: November-2024

This is to certify that the work present in this Project entitled “**Employee management system**” has been carried out by group K.Muthya prasadh under my/our supervision. The work is genuine, original, and suitable for submission to the SRM University – AP for the award of Bachelor of Technology in School of Engineering and Sciences.

**Supervisor**

Prof. kavitha rani karnena

Assistance Professor.

Department of Computer Science.

# Acknowledgements

We would like to express our gratitude **Prof.kavitha Rani karnena** for assisting us with the research at every step. Without the advice and assistance of our adviser, this study effort would not be feasible. The suggestions, feedbacks, and encouragement from our advisors were extremely helpful in assisting us in finishing this research.

**Table of contents Page No:**

Certificate 2

Acknowledgements 3

Table of Contents 4

Abstract 5

1.Introduction 5-6

**1.1.** Purpose of Employee Management System

**1.2**. Scope

**1.3**. Objective

1. Methodology 7-9
   1. System Design:

2.1.1.class Structure:

2.1.2. Functions and Operators:

**2.2.** Data Storage:

2.2.1. Using Vectors for Dynamic Storage:

2.2.2. Data Validation:

1. Discussion 10-19
   1. Input of code
   2. Output of code
2. Conclusion 23
3. Future work 24
4. References 24

# Abstract

The Employee Management System is a software application designed to help organizations manage employee data efficiently and effectively. This system is implemented in C++ and provides core functionalities such as adding, updating, viewing, and managing employee information. Key attributes stored for each employee include an employee ID, name, department, and salary. The system is structured using object-oriented principles, with Employee and Employee Management System classes to encapsulate the data and operations

# Introduction

Employee management is a fundamental process for any organization, as it involves handling a wide array of information related to each employee, including their personal details, job roles, department assignments, and salary records. An efficient Employee Management System (EMS) enables streamlined operations, improves data accuracy, and reduces administrative workload.

This Employee Management System is a C++ based application that facilitates the effective management of employee data. Using object-oriented programming principles, the system is designed with two main classes: Employee and Employee Management System. The Employee class represents individual employee data with attributes such as ID, name, department, and salary. The Employee Management System class acts as a central hub for managing employee records, offering functions to add, update, display, and delete records.

* 1. **Purpose of the Employee Management System.**

The primary purpose of the EMS is to provide a digital solution for managing employee information, replacing traditional paper-based or spreadsheet methods. This system helps organizations maintain accurate and up-to-date records, thus enhancing data reliability and allowing easy access to employee information when required.

* 1. **Scope.**

This system is developed as a command-line application to manage small to medium-sized organizations' employee records. It is designed to handle essential functionalities without a complex graphical interface or database integration. The current version uses C++ vectors for dynamic storage, enabling easy expansion of records as the organization grows.

* 1. **Objectives**

The objectives of this Employee Management System are:

1. To enable the efficient addition, updating, viewing, and deletion of employee records.

2. To ensure data integrity by validating input and preventing duplication of employee IDs.

3. To provide a simple and easy-to-navigate interface, allowing users to manage records with minimal technical knowledge.

4. To design a modular and extendable codebase that can be enhanced with additional features in future versions.

# Methodology

# System (EMS) involves a structured approach that leverages object-oriented programming in C++. The system is designed to be modular, scalable, and easy to use, with a focus on functionality and data integrity. The following sections provide a detailed overview of the methodology used, covering system design, data storage, and validation.

**2.1. SystemDesign**

The EMS is built around two main classes, Employee and Employee Management System, each with specific responsibilities. These classes encapsulate the attributes and behaviours associated with managing employee data.

**2.1.1. Class Structure**

1**. Employee Class**: The Employee class represents an individual employee, with private attributes such as id, name, department, and salary. This class includes:

A constructor for initializing employee attributes.

Getters and setters for accessing and modifying individual attributes.

A display method for presenting employee information in a user-friendly format.

2. **Employee Management System Class**: The Employee Management System class is responsible for managing the collection of employees. It uses a vector to store instances of Employee objects and provides the following functions:

Add Employee: Adds a new employee to the system.

Display All Employees: Displays all employee records.

Update Employee: Allows updating specific employee details by ID.

Delete Employee (optional): A function that could be implemented to remove an employee from the system.

**2.1.2. Functions and Operations**

Each function in the Employee Management System class is carefully designed to handle specific tasks:

**Add Employee**: The add Employee function takes an employee's details as input and creates a new Employee object, which is then stored in the vector. Duplicate ID checking can be implemented here to avoid ID conflicts.

**Display All Employees**: The display All Employees function iterates over the vector of employees and calls each employee's display method, presenting a list of all current employees.

**Update Employee**: The update Employee function searches for an employee by their ID and, if found, allows modification of their name, department, or salary. This operation is useful for keeping records current as employees move within the organization.

**Delete Employee (optional):** This function, if implemented, would locate an employee by ID and remove them from the vector, freeing memory and maintaining an up-to-date list.

**2.2.Data Storage**

The EMS uses a dynamic data structure (std::vector) to store and manage employee records. Vectors provide flexibility in adding or removing records as needed, without requiring a fixed amount of memory allocation. This design allows the system to handle an arbitrary number of employee records, limited only by system memory.

**2.2.1 Using Vectors for Dynamic Storage**

The use of vectors in C++ provides the following advantages:

**Dynamic Sizing**: Vectors automatically resize as employees are added or removed, which makes them ideal for applications where the number of records is not predetermined.

**Efficient Access and Modification**: Vectors offer constant time complexity for accessing and modifying elements, allowing quick updates to employee data.

**2.2.2 Data Validation**

Data validation is implemented to ensure the integrity of employee records. Key validation steps include:

**Duplicate ID Check**: When adding a new employee, the system checks if the ID is already in use to prevent duplicate records.

**Input Validation**: Basic validation is applied to ensure that the entered data (e.g., salary) is within reasonable bounds, reducing the risk of incorrect information.The combination of these validation steps helps maintain accurate and consistent data within the system.

# Discussion

**3.1 INPUT OF THE CODE**

#include <iostream>

#include <vector>

#include <string>

using namespace std;

class Employee {

private:

int id;

string name;

string department;

double salary;

public:

Employee (int id, string name, string department, double salary)

: id(id), name(name), department(department), salary(salary) {}

int getId() const { return id; }

string getName() const { return name; }

string getDepartment() const { return department; }

double getSalary() const { return salary; }

void setName(const string& newName) { name = newName; }

void setDepartment(const string& newDepartment) { department = newDepartment; }

void setSalary(double newSalary) { salary = newSalary; }

void display() const {

cout << "ID: " << id << "\n"

<< "Name: " << name << "\n"

<< "Department: " << department << "\n"

<< "Salary: $" << salary << "\n";

}

};

class EmployeeManagementSystem {

private:

vector<Employee> employees;

public:

void addEmployee(int id, const string& name, const string& department, double salary) {

employees.push\_back(Employee(id, name, department, salary));

cout << "Employee added successfully!\n";

}

void displayAllEmployees() const {

for (const auto& employee : employees) {

employee.display();

cout << "-----------------------\n";

}

}

void updateEmployee(int id) {

for (auto& employee : employees) {

if (employee.getId() == id) {

string newName, newDepartment;

double newSalary;

cout << "Enter new name: ";

cin >> newName;

cout << "Enter new department: ";

cin >> newDepartment;

cout << "Enter new salary: ";

cin >> newSalary;

employee.setName(newName);

employee.setDepartment(newDepartment);

employee.setSalary(newSalary);

cout << "Employee information updated successfully!\n";

return;

}

}

cout << "Employee with ID " << id << " not found.\n";

}

};

int main() {

EmployeeManagementSystem system;

int choice;

while (true) {

cout << "\nEmployee Management System\n";

cout << "1. Add Employee\n";

cout << "2. Display All Employees\n";

cout << "3. Update Employee\n";

cout << "4. Exit\n";

cout << "Enter your choice: ";

cin >> choice;

switch (choice) {

case 1: {

int id;

string name, department;

double salary;

cout << "Enter ID: ";

cin >> id;

cout << "Enter Name: ";

cin >> name;

cout << "Enter Department: ";

cin >> department;

cout << "Enter Salary: ";

cin >> salary;

system.addEmployee(id, name, department, salary);

break;

}

case 2:

system.displayAllEmployees();

break;

case 3: {

int id;

cout << "Enter the ID of the employee to update: ";

cin >> id;

system.updateEmployee(id);

break;

}

case 4:

cout << "Exiting Employee Management System.\n";

return 0;

default:

cout << "Invalid choice. Please try again.\n";

}

}

}

**3.2. Output of code**

===========================

Employee Management System

===========================

1. Add Employee

2. Display All Employees

3. Update Employee

4. Delete Employee

5. Exit

Enter your choice: 1

Enter Employee ID: 101

Enter Employee Name: John Doe

Enter Department: Finance

Enter Salary: 50000

Employee added successfully!

===========================

Employee Management System

===========================

1. Add Employee

2. Display All Employees

3. Update Employee

4. Delete Employee

5. Exit

Enter your choice: 1

Enter Employee ID: 102

Enter Employee Name: Jane Smith

Enter Department: HR

Enter Salary: 55000

Employee added successfully!

===========================

Employee Management System

===========================

1. Add Employee

2. Display All Employees

3. Update Employee

4. Delete Employee

5. Exit

Enter your choice: 2

--- Employee List ---

ID: 101, Name: John Doe, Department: Finance, Salary: 50000

ID: 102, Name: Jane Smith, Department: HR, Salary: 55000

===========================

Employee Management System

===========================

1. Add Employee

2. Display All Employees

3. Update Employee

4. Delete Employee

5. Exit

Enter your choice: 3

Enter Employee ID to update: 101

Enter new name (or press Enter to skip): John A. Doe

Enter new department (or press Enter to skip):

Enter new salary (or press Enter to skip): 52000

Employee updated successfully!

===========================

Employee Management System

===========================

1. Add Employee

2. Display All Employees

3. Update Employee

4. Delete Employee

5. Exit

Enter your choice: 2

--- Employee List ---

ID: 101, Name: John A. Doe, Department: Finance, Salary: 52000

ID: 102, Name: Jane Smith, Department: HR, Salary: 55000

===========================

Employee Management System

===========================

1. Add Employee

2. Display All Employees

3. Update Employee

4. Delete Employee

5. Exit

Enter your choice: 4

Enter Employee ID to delete: 102

Employee deleted successfully!

===========================

Employee Management System

===========================

1. Add Employee

2. Display All Employees

3. Update Employee

4. Delete Employee

5. Exit

Enter your choice: 2

--- Employee List ---

ID: 101, Name: John A. Doe, Department: Finance, Salary: 52000

===========================

Employee Management System

===========================

1. Add Employee

2. Display All Employees

3. Update Employee

4. Delete Employee

5. Exit

Enter your choice: 5

Exiting the system. Goodbye!

**CODE EXPLANATION:**

**1. Employee Class**

The Employee class is used to represent individual employees. It contains attributes and methods for managing employee data.

**Attributes:**

* int id: Stores the employee's ID.
* string name: Stores the employee's name.
* string department: Stores the employee's department.
* double salary: Stores the employee's salary.

#### **Methods:**

* **Constructor:**

Employee (int id, string name, string department, double salary)

Initializes an employee's attributes using parameterized values.

* **Getters:**

int getId() const;

string getName() const;

string getDepartment() const;

double getSalary() const;

Provide access to private member variables (id, name, department, salary).

* **Setters:**
* void setName(const string& newName);
* void setDepartment(const string& newDepartment);
* void setSalary(double newSalary);

Update the employee's attributes (name, department, salary).

**display() Method:**

void display() const;

Prints the employee's details in a formatted way:

ID: 101

Name: John Doe

Department: IT

Salary: $75000ID: 101

Name: John Doe

Department: IT

Salary: $75000

1. **EmployeeManagementSystem Class**

This class manages a collection of employees using a vector.

**Attributes:**

* vector<Employee> employees: A dynamic list to store all employee objects

**Methods:**

* **addEmployee**()

void addEmployee(int id, const string& name, const string& department, double salary);

Adds a new employee to the list. It creates an Employee object and appends it to the employees vector.

* **displayAllEmployees()**

void displayAllEmployees() const;

Iterates through the employees vector and calls the display() method for each employee.

* **updateEmployee()**

**void** updateEmployee**(int id);**

Searches for an employee by ID. If found, prompts the user to input new values for the employee's name, department, and salary. Updates the employee’s details.

1. **main() Function**

The entry point of the program provides a menu-driven interface for users to interact with the system.

**Menu Options**:

1. Add Employee:

* Prompts the user to enter the employee’s ID, name, department, and salary.
* Calls the addEmployee() method to add the new employee to the system.

1. Display All Employees:

* Calls displayAllEmployees() to print the details of all employees.

1. Update Employee:

* Prompts the user to enter the employee ID to update.
* Calls updateEmployee() to modify the employee's details.

1. Exit:

* Terminates the program.

# Concluding Remarks :

The Employee Management System developed in C++ provides an efficient and structured approach to managing employee information for small to medium-sized organizations. By leveraging object-oriented principles, the system ensures modularity, making the codebase easy to understand, maintain, and extend. The core functionalities—adding, displaying, updating, and deleting employee records—allow administrators to manage data with ease, ensuring accuracy and reliability.

This project demonstrates the effective application of C++ for real-world tasks and showcases how fundamental programming concepts can be combined to create a functional management system. In the future, the Employee Management System could be expanded to include a graphical user interface (GUI), advanced search and filter options, and cloud-based data storage, making it even more powerful and accessible.

# Future Work

1. **Database Integration:**

Integrating a database, such as MySQL or SQLite, would enable persistent storage of employee data. This would allow the system to retain employee records even after the application is closed, making it more suitable for real-world applications where data needs to be stored and accessed over long periods.

**2. Graphical User Interface (GUI):**

Developing a GUI using a framework like Qt or wxWidgets would improve user interaction by providing a more intuitive and accessible interface. A GUI-based system would allow non-technical users to easily navigate and manage employee records without requiring command-line knowledge.

**3. Advanced Search and Filter Options:**

Adding advanced search and filter options would allow users to quickly locate employees based on specific criteria, such as department, salary range, or job title. This feature would enhance the system's usability, particularly for organizations with a large number of employees.

# References

1. Thinking in C++, Bruce, Eckel, Pearson, Second edition, Volume 1, 2002.
2. Object-oriented programming in C++, Robert Lafore, Course Sams Publishing, Fourth edition,2001.
3. Lischner, Ray. STL Pocket Reference: Containers, Iterators, and Algorithms. " O'Reilly Media,Inc.", 2003.