12/23/2024

Employee Management System

Project Report

Presented by:

Jamal Hassan 01-136212-018

Shazif Dildar 01-136212-042

Presented to:

Sir Saeed ur Rehman

Table of Contents

1. Introduction

The **Employee Management System** is a C++ program designed to facilitate the management of employee records. It allows users to perform essential operations such as adding, updating, deleting, and displaying employee records. Furthermore, the system uses file handling for persistent data storage, ensuring that employee information remains accessible even after the program terminates.

2. Objectives

The primary objectives of this project are:

- To create a menu-driven application for managing employee data.
- To implement file handling for saving and retrieving employee information.
- To use structures, vectors, and control statements for efficient data manipulation.

3. System Design

3.1 Data Structure

The system uses a structure (Employee) to represent employee data, which includes:

- Employee ID
- Name
- Department
- Salary

The employees are stored dynamically in a vector<Employee> to allow flexible and efficient management of records.

3.2 Operations

The following operations are implemented:

1. **Add Employee**: Add a new employee to the system.
2. **Display Employees**: Display all existing employee records.
3. **Update Employee**: Modify an existing employee's data.
4. **Delete Employee**: Remove an employee from the system.
5. **Save and Exit**: Save all records to a file and exit the program.

4. Implementation

4.1 Tools and Technology

- **Programming Language**: C++
- **Compiler**: Any standard C++ compiler (e.g., GCC, MSVC)
- **File Handling**: Text files for persistent storage

4.2 File Structure

The program uses a single text file (employees.txt) to store employee records in the following format:

ID,Name,Department,Salary

Each employee's information is stored on a new line.

4.3 Code Modules

- **Main Function**: Contains the menu-driven interface.
- **Helper Functions**: Perform specific tasks like adding, displaying, updating, and deleting records.
- **File Handling**: Save and load data to/from the text file.

5. Features

5.1 User-Friendly Interface

The system provides a clear menu-driven interface for easy navigation and operation.

5.2 Persistent Data Storage

Employee records are saved to a file, ensuring data is retained between program runs.

5.3 Dynamic Storage

The use of vector allows dynamic allocation of memory, making the system efficient and scalable.

6. Code Walkthrough

Below is an excerpt of the key sections of the program:

6.1 Employee Structure

```
struct Employee {
    int id;
    string name;
    string department;
    float salary;
};
```

6.2 Adding an Employee

```
void addEmployee(vector<Employee>& employees) {
    Employee emp;
    cout << "Enter Employee ID: ";
    cin >> emp.id;
    cin.ignore();
    cout << "Enter Name: ";
    getline(cin, emp.name);
    cout << "Enter Department: ";
    getline(cin, emp.department);
```

```
    cout << "Enter Salary: ";
    cin >> emp.salary;

    employees.push_back(emp);
    cout << "Employee added successfully.\n";
}
```

6.3 File Handling

Saving Data:

```
void saveToFile(const vector<Employee>& employees) {
    ofstream file("employees.txt");
    for (const auto& emp : employees) {
        file << emp.id << "," << emp.name << "," << emp.department << "," << emp.salary << "\n";
    }
    cout << "Data saved successfully.\n";
}
```

Loading Data:

```
void loadFromFile(vector<Employee>& employees) {
    ifstream file("employees.txt");
    string line;
    while (getline(file, line)) {
        Employee emp;
        size_t pos1 = line.find(',');
        size_t pos2 = line.find(',', pos1 + 1);
        size_t pos3 = line.find(',', pos2 + 1);

        emp.id = stoi(line.substr(0, pos1));
        emp.name = line.substr(pos1 + 1, pos2 - pos1 - 1);
        emp.department = line.substr(pos2 + 1, pos3 - pos2 - 1);
```

```
        emp.salary = stof(line.substr(pos3 + 1));


        employees.push_back(emp);
    }
}
```

7. Future Enhancements

7.1 Search Functionality

Implement a feature to search for employees by ID, name, or department.

7.2 Sorting

Add options to sort employees by salary, name, or department.

7.3 GUI Integration

Integrate the system with a graphical user interface for better usability.

7.4 Advanced File Handling

Use binary files or a database for faster and more secure data storage.

8. Conclusion

The **Employee Management System** successfully implements a basic framework for managing employee records. It demonstrates the use of structures, dynamic data storage, file handling, and menu-driven interaction in C++. The system is robust and provides a foundation for further enhancements to meet real-world requirements.