**N.B.K.R INSTITUTE OF SCIENCE AND**

**TECHNOLOGY**

**VIDYANAGAR -TIRUPATI**

**EMPLOYEE RECORD SYSTEM USING**

**LINKEDLIST & FILE HANDLING**

**Course: Data Structures**

**Department: Computer Science**

**Section: F**

**Year: Ⅰ**

**Semister: Ⅱ**

**Date:**

**Submitted To : Ashok Selva Kumar**

**Submitted by:**

**R.V.Nishitha(24KB1A05FJ)**

**P.Kaveri(24KB1A05EF)**

**P.Sujitha(24KB1A05DB)**

**Acknowledgement:**

I would like to express my sincere gratitude to my Computer Science teacher, [Teacher’s Name], for their valuable guidance and constant support throughout the development of this project. I am also thankful to my classmates and family members who encouraged me and provided helpful feedback during the various stages of this project.

This project has helped me gain practical understanding of data structures and file handling in C and how they are applied in real-world applications.

**Abstract:**

This project report presents a simple yet effective Employee Record System designed using C, where linked lists are used for dynamic in-memory data management, and file handling is used for permanent data storage. The system allows for adding, viewing, searching, updating, and deleting employee records. Each employee record contains an ID, name, and salary. The program also enables saving data to a text file and retrieving it later, ensuring persistence. This project demonstrates the integration of basic data structures with file operations, and serves as an introductory model for understanding how employee data can be managed in real-life HR systems.

Introduction

The **Employee Record System** is a software application developed in C to manage employee details such as ID, name, salary, etc. The system utilizes a **singly linked list** to store data in memory and **file handling** to store records permanently.

**Objective**

* To manage employee data efficiently using linked lists.
* To enable persistent data storage using files.
* To provide functionalities like add, search, delete, update, and display records.

**System Requirements**

**Hardware Requirements:**

Processor: Intel Pentium or higher

RAM: 512 MB or more

Hard Disk: Minimum 100 MB free space

Display: Standard VGA or higher

**Software Requirements:**

Operating System: Windows/Linux

IDE/Compiler: Code::Blocks, Turbo C++, Dev-C++, or GCC

Text Editor: Notepad/Notepad++/VS Code (optional)

**Methodology**

The development of this project followed the following methodology:

Planning and Analysis:

Understand the requirements of an employee management system.

Decide on the structure and data fields.

Design:

Choose a linked list for dynamic memory allocation.

Decide on file handling to save and load records.

Implementation:

Code each feature: add, delete, search, update, display, save, and load.

Use a text file (employee.txt) to store records.

Testing:

Test each function with multiple cases (valid, invalid, edge).

Ensure file operations work even after restarting the program.

Documentation:

Prepare this report, explaining the working and purpose of each part.

# Project Description

Problem Statement:

In many small organizations or academic settings, employee details are often recorded manually or in basic spreadsheets. This method is inefficient, prone to errors, and lacks features such as fast searching, updating, or deleting records. There's a need for a lightweight, file-based application to manage employee records dynamically.

Proposed Solution:

This project proposes a console-based employee record system built using C/C++, utilizing:

Singly Linked List for dynamic data storage in memory.

File Handling for persistent storage in a text file (employee.txt).

The system allows the user to:

Add new employee records.

Display all existing records.

Search for an employee by ID.

Update and delete existing records.

Key Features:

Dynamic Storage: Records are stored using linked lists to allow efficient memory usage.

Persistent Storage: Employee data is saved in a text file for retrieval on next use.

CRUD Operations: Supports Create, Read, Update, and Delete functionalities.

Simple Interface: Menu-driven CLI interface for ease of use.

Scalable Design: Easily extendable to include additional fields like department, date of joining, etc.

Algorithm:

Step 1: Initialize

Define an Employee structure (id, name, salary, next).

Set global pointer head = NULL.

Step 2: Create Employee

Allocate memory.

Assign id, name, salary; set next = NULL.

Step 3: Add Employee

Create a new node.

Insert at the beginning of the list.

Step 4: Display Employees

Traverse the list from head and print employee details.

Step 5: Save to File

Open file in write mode.

Write each employee's data.

Close file.

Step 6: Load from File

Open file in read mode.

Read records and call addEmployee().

Step 8: Main Menu

Load data from file.

Loop to provide options:

Add Employee

Display Employees

Save to File

Exit

Program code:

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

typedef struct Employee {

int id;

char name[50];

float salary;

struct Employee\* next;

} Employee;

Employee\* head = NULL;

// Function to create a new employee node

Employee\* createEmployee(int id, const char\* name, float salary) {

Employee\* newEmp = (Employee\*)malloc(sizeof(Employee));

newEmp->id = id;

strcpy(newEmp->name, name);

newEmp->salary = salary;

newEmp->next = NULL;

return newEmp;

}

// Add employee to the linked list

void addEmployee(int id, const char\* name, float salary) {

Employee\* newEmp = createEmployee(id, name, salary);

newEmp->next = head;

head = newEmp;

}

// Display all employee records

void displayEmployees() {

Employee\* temp = head;

if (temp == NULL) {

printf("No records found.\n");

return;

}

printf("Employee Records:\n");

while (temp != NULL) {

printf("ID: %d, Name: %s, Salary: %.2f\n", temp->id, temp->name, temp->salary);

temp = temp->next;

}

}

// Save all employee records to a file

void saveToFile(const char\* filename) {

FILE\* fp = fopen(filename, "w");

if (!fp) {

printf("Unable to open file.\n");

return;

}

Employee\* temp = head;

while (temp != NULL) {

fprintf(fp, "%d,%s,%.2f\n", temp->id, temp->name, temp->salary);

temp = temp->next;

}

fclose(fp);

printf("Data saved to file.\n");

}

// Load employee records from a file

void loadFromFile(const char\* filename) {

FILE\* fp = fopen(filename, "r");

if (!fp) {

printf("No previous records found.\n");

return;

}

int id;

char name[50];

float salary;

while (fscanf(fp, "%d,%49[^,],%f\n", &id, name, &salary) == 3) {

addEmployee(id, name, salary);

}

fclose(fp);

printf("Data loaded from file.\n");

}

int main() {

int choice, id;

char name[50];

float salary;

loadFromFile("employees.txt");

while (1) {

printf("\nEmployee Record System\n");

printf("1. Add Employee\n");

printf("2. Display Employees\n");

printf("3. Save to File\n");

printf("4. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

getchar(); // consume newline

switch (choice) {

case 1:

printf("Enter ID: ");

scanf("%d", &id);

getchar(); // consume newline

printf("Enter Name: ");

fgets(name, sizeof(name), stdin);

name[strcspn(name, "\n")] = 0; // remove newline

printf("Enter Salary: ");

scanf("%f", &salary);

addEmployee(id, name, salary);

break;

case 2:

displayEmployees();

break;

case 3:

saveToFile("employees.txt");

break;

case 4:

saveToFile("employees.txt");

printf("Exiting...\n");

return 0;

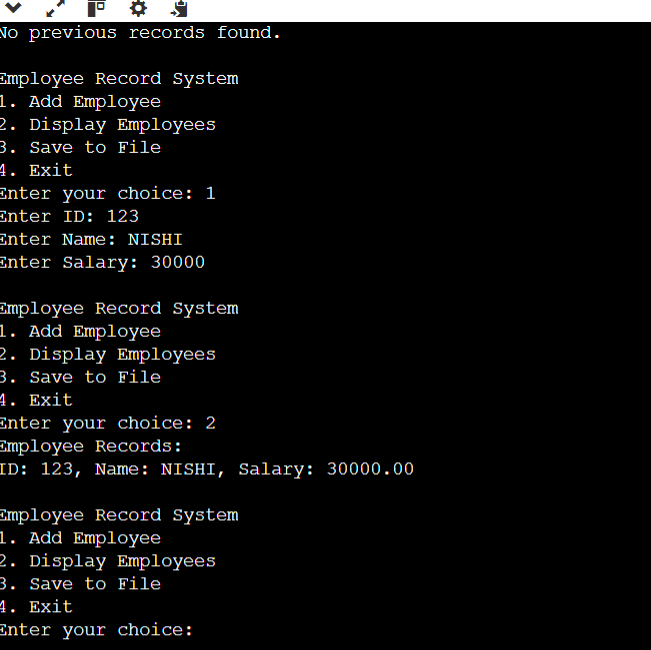
default:

printf("Invalid choice!\n");

}

}

}



A screen shot of a computer

AI-generated content may be incorrect.

Testing & Validation:

No Previous record found

Employee Record System

1.Add Employee.

.2.Display Employees

3.Save to File

4.Exit

Enter your Choice:1

Enter ID: 123

Enter Name: NISHI

Enter Salary: 3000

Employee Record System

1. Add Employee
2. Display Employees
3. Save to File
4. Exit

Enter your Choice: 2

Employee Records:

ID:123, Name: NISHI, Salary: 30000.00

Employee Record System

1..Add Employee

2. Display Employees

1. Save to File
2. Exit

Enter your Choice: 3

Data saved to file.

Employee Record System

1. Add Employee
2. Display Employees
3. Save to File
4. Exit

Enter your Choice:4

Data Saved to file

Exiting….

LIMITATIONS:

Despite being a functional and educational project, the Employee Record System has the following limitations:

1. **Lack of Security**
   * The system does not include any form of authentication or data encryption.
   * Employee data stored in plain text files can be easily accessed or modified.
2. **No GUI (Graphical User Interface)**
   * The system operates entirely through a command-line interface, which is not user-friendly for non-technical users.
3. **Data Volatility (In-memory Operations)**
   * . All operations are performed on data loaded into memory (linked list). If the system crashes before saving, unsaved changes are lost.
4. **No Input Validation**
   * Limited input validation may cause the program to behave unexpectedly if incorrect data types are entered.
5. **No Data Sorting or Filtering**
   * The system lacks options to sort or filter employees by name, salary, or other attributes.
6. **Text File Limitation**
   * Uses a plain text file (employee.txt) which is not optimized for large data storage or complex queries compared to a database.
7. **Scalability**
   * Not suitable for large-scale applications. Performance degrades with a high number of employee records due to linear search in linked lists.
8. **No Error Logging**
   * The system does not maintain any error or activity log for debugging or auditing purposes.
9. **Single-User System**
   * It is designed for use by a single user. No multi-user access or concurrency is supported
10. **Hardcoded File Path**

* The file path for storing data is fixed in code, making it less flexible across different systems.

**11. Future Enhancements**

* GUI-based frontend using C++ with Qt.
* Encryption for secure data storage.
* Sorting employee records.
* Integration with databases like MySQL.

Bottom of Form

Bottom of Form

**10. Conclusion**

This project demonstrates efficient use of dynamic memory and persistent file storage for managing employee records. It highlights real-world use of linked lists for memory efficiency and file handling for data persistence.