**🔷 THEORY**

**📌 Problem Statement**

Maintain a file-based system to manage employee data (ID, name, designation, and salary). The system must support:

* Adding a new employee
* Deleting an employee by ID
* Searching for an employee by ID
* Displaying all employee records

**📌 Index Sequential File**

An **index sequential file** is a file that maintains:

* **Sequential storage** of records (line by line)
* An **index** (e.g., employee ID) used for quick access

In this implementation, the index is the **Employee ID**, and operations are simulated by scanning line-by-line for the key.

**⚙️ Characteristics:**

* Efficient for **searching** and **updating** data by ID
* Suitable for simple record systems (like employee databases)
* File is managed using fstream, ifstream, and ofstream

**🔷 ALGORITHM**

**🔧 1. Add Employee**

text

CopyEdit

Input: Employee ID, Name, Designation, Salary

Steps:

1. Open "db.txt" in append mode.

2. Read employee details from user.

3. Write details in fixed-width format using `setw()`.

4. Close file and confirm success.

**🔧 2. Delete Employee**

text

CopyEdit

Input: Employee ID

Steps:

1. Open "db.txt" in read mode.

2. For each line:

a. If it does NOT match the ID → store it.

b. If it matches the ID → skip it.

3. Close read file.

4. Open "db.txt" in write mode (truncate).

5. Write back all saved lines.

6. Confirm deletion if ID was found.

**🔧 3. Search Employee**

text

CopyEdit

Input: Employee ID

Steps:

1. Open "db.txt" in read mode.

2. For each line:

a. Skip header.

b. If line starts with ID → display and mark as found.

3. If not found, show message.

4. Close file.

**🔧 4. Display All Employees**

text

CopyEdit

Steps:

1. Open "db.txt" in read mode.

2. Read and print each line.

3. Close file.

**🔷 COMPLEXITY ANALYSIS**

| **Operation** | **Time Complexity** |
| --- | --- |
| Add | O(1) (append) |
| Delete/Search | O(n) (scan file) |
| Display All | O(n) |