Low-Level Design (LLD) — EMS Project

Overview

This document describes the low-level design for the EMS (Employee Management System) implemented in the provided codebase. It covers class responsibilities, key data structures, method summaries, interactions, memory/ownership notes, and known issues.

Modules / Files

- `EmployeeIF.H` — interface (pure virtual) specifying the public API for employees.
- `Employee.H` / `Employee.C` — concrete base class implementing common behavior for employees.
- `FullEmp.H` / `FullEmp.C` — full-time employee specialization.
- `InternEmp.H` / `InternEmp.C` — intern specialization.
- `ContEmp.H` / `ContEmp.C` — contractual specialization.
- `Manager.H` / `Manager.C` — singleton-like manager controlling employees and collections.
- `Deque.H`, `EDLL.H`, `Node.H` — container and double-linked list utilities used by `Manager` to store employee pointers.
- `EMS.C`, `main.C` — application entrypoints and CLI orchestration.

Class Responsibilities

EmployeeIF (interface)

- Pure virtual methods for getters and setters common to all employee types.
- Enums: `EmpGender`, `EmpType`, `EmpStatus`, `Agency`, `College`, `Branch`, `InputType`, `WidthPrint`.
- Purpose: define the contract for employee objects used by the rest of the system.

Employee (concrete base)

- Stores common fields: name, id, gender, DOB, DOJ, DOL, type, status.
- Implements many `EmployeeIF` methods (getters, setters, I/O operators, ID generation, DOJ/DOL helpers).
- Friend functions: stream operators and `convertIntern2FullTime` helper.

Key methods implemented in `Employee.C`: - Constructors to create random or user-supplied employees. - `setEmployeeType`, `setEmployeeStatus`, `setDOL`, `setDOJ`, `addLeavesToAll` (no-op default), etc. - I/O operators: `operator>>` reads Name/Gender/DOB via `std::getline(std::cin >> std::ws, ...)`.

Ownership: `Employee` is allocated on heap by `Manager` and stored by pointer in containers (raw pointers used throughout).

FullEmp / InternEmp / ContEmp

- Each derives from `Employee` and override specific methods: `getEmployeeCount`, `getCurrentLeaves`, and leave-management methods.
- `FullEmp` adds `mCurrentLeaves`, `mLeaveApplied`, and static `mMaxLeaves`.
- `InternEmp` stores college and branch.
- `ContEmp` stores agency.

Deque / EDLL / Node

- `Node<T>`: doubly-linked node structure with `mNodeData`, `mNextNode`, `mPrevNode`.
- `Deque<T>`: generic deque built on `Node<T>` with push/pop front/back, resize, operator[], iteration, size, clear.
- `EDLL<T>`: extends `Deque<T>` with middle insertion/removal (uses `this->` to access dependent base members inside template).

Memory: nodes allocated with `new` and freed appropriately in destructors; containers hold raw pointers for data.

Manager

- Singleton-style manager stored in `Manager::mOwnPtr` (single instance created with `getInstance`).
- Manages collections: `mEmployeeList`, `mResignedEmployeeList` (both `EDLL<EmployeeIF*>*`).
- Provides methods to add/remove employees, display, search, and manage leaves.
- Allocates employees (using `new FullEmp()`, `new InternEmp()`, `new ContEmp()`) and stores them in container of `EmployeeIF*`.

Ownership: Manager owns the containers and is responsible for deleting them in destructor; employees appear to be deleted upon removal in some code paths but also moved to resigned list in others (inconsistent).

Class diagrams

Below is the UML that models the primary classes and relationships. The original StarUML project file is included in the repo; see the UML section below for viewing instructions.

*** UML file provided ***

I provided a StarUML project file with the system UML. I've kept the original file in the repository at:

- `code/UML/EMSAss2.mdj`

How to view/edit the UML:

1. Install StarUML (https://staruml.io/) or another MDJ-capable tool.
2. Open `code/UML/EMSAss2.mdj` in StarUML to view the class diagrams, sequence diagrams, etc.

Removing an employee

- `Manager::removeEmployee(...)` searches, marks status RESIGNED, moves to `mResignedEmployeeList` (creating it if null), removes from `mEmployeeList` (via `remElementMiddle`).

Printing

- Manager iterates container and uses `operator<<` (friend) for the specific derived class by casting `EmployeeIF*` to `FullEmp*`, `InternEmp*`, or `ContEmp*` and calling `std::cout << static_cast<const FullEmp*>(sEmp)`.

File->Class mapping

- `EmployeeIF.H` — EmployeeIF
- `Employee.H/.C` — Employee implementation
- `FullEmp.H/.C` — FullEmp
- `InternEmp.H/.C` — InternEmp
- `ContEmp.H/.C` — ContEmp
- `Manager.H/.C` — Manager singleton (templated earlier but now non-template)
- `Deque.H`/`EDLL.H`/`Node.H` — container utilities
- `EMS.C`, `main.C` — application entrypoints

Sequence diagram (add employee)

```
User -> Manager::getInstance -> Manager::addEmployee(type)
Manager -> new DerivedEmployee()
Manager -> EDLL::pushBack
EDLL::pushBack -> Node allocation
```