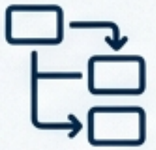
SSN Hostel Management System

A Case Study in C-Language System Architecture and Design

Language: C

Data Storage: File-based Persistence



Core Paradigms: Structured Programming, Modular Design

A Robust, Menu-Driven System for Efficient Hostel Administration

This project is a comprehensive, console-based management system built entirely in C. It is designed to handle student records and maintenance issues through a secure, dual-portal interface, ensuring a clear separation of roles and responsibilities.

Admin Portal

- Full control over student data (CRUD operations).
- Advanced search, sort, and filtering capabilities.
- System-wide dashboard and reporting.
- Complete oversight of the issue tracking system.

Student Portal

- A simplified, focused interface.
- Enables students to raise new maintenance or hostel-related issues.
- Allows students to check the status of their submitted tickets.

The Blueprint: The `Student` Data Model

A comprehensive data structure designed to be the single source of truth for all student information.

struct Student

Primary Identifier

`int id`	Unique Student / Roll ID.

Core Personal Details

`name`	Student Name
`parent_name`	Parent Name
`mother_name`	Mother Name
`dob`	Date of Birth
`gender`	Gender

Contact & Location

`phone`	Contact Number
`email`	Email Address
`address`	Permanent Address
`district`	District
`district`	District
`state`	State
`pincode`	Pincode

Contact & Location

`phone`	Contact Number
`email`	Email Address
`address`	Permanent Address
`district`	District
`state`	State
`pincode`	Pincode

Academic & Hostel Information

`room_no`	Hostel Room Number
`hostel_block`	Hostel Block
`campus`	Campus (SSN/SNU)
`department`	Department
`year`	Current Year
`admission_year`	Year of Admission

Administrative Fields

`fee_status`	Fee Payment Status (PAID / NOT_PAID)

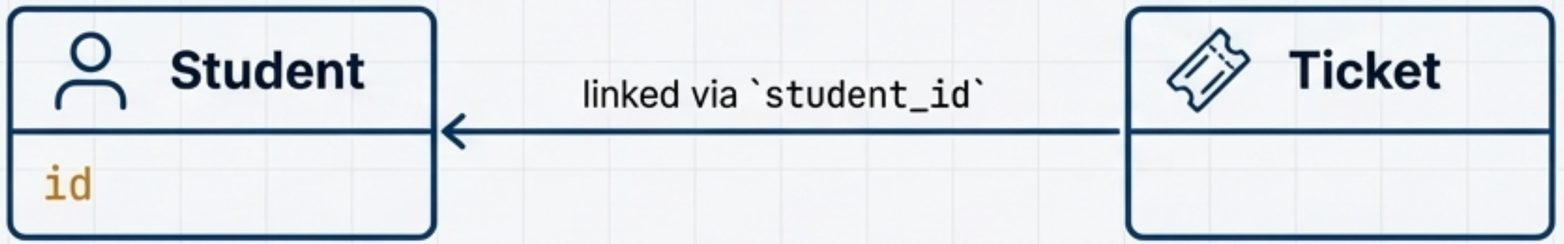
Key Design Highlight

```
int is_active; // 1 = ACTIVE, 0 = DELETED
```

This field enables ‘soft deletion’. Instead of permanently removing records, students are marked as inactive. This preserves historical data integrity and allows for easy reactivation, a crucial feature for robust data management.

The `Ticket` Model: Managing the Issue Lifecycle

A linked data structure for logging, tracking, and resolving student-raised issues.



struct Ticket	
`int ticket_id`	Unique, auto-incrementing ID.
`int student_id`	Foreign key linking to the `Student` record.
`char student_name[50]`	Denormalised for quick display without extra lookups.
`char issue[200]`	Description of the problem.
`char status[20]`	The current state of the ticket.

The Ticket Lifecycle

The `status` field tracks an issue from submission to resolution, providing clear visibility for both students and administrators.

Ensuring Data Integrity with File-Based Persistence

The system state is preserved between sessions using direct binary file I/O for efficiency and reliability.

1. System Startup (`load_data()` function)

- Reads `student_count` and `ticket_count` integers first.
- Loads the entire `students` array from `students.dat`.
- Loads the entire `tickets` array from `tickets.dat`.
- Calculates the `next_ticket_id` by scanning existing tickets, preventing ID reuse.

```
// Simplified load_data()
fp = fopen(STUDENT_FILE, "rb");
if (fp != NULL) {
    fread(&student_count, sizeof(int), 1, fp);
    fread(students, sizeof(Student), student_count, fp);
    fclose(fp);
}
```

2. Runtime & Shutdown (`save_data()` function)

- Called after every single operation in the admin menu, guaranteeing data is always current.
- Overwrites `students.dat` and `tickets.dat` with the in-memory arrays.

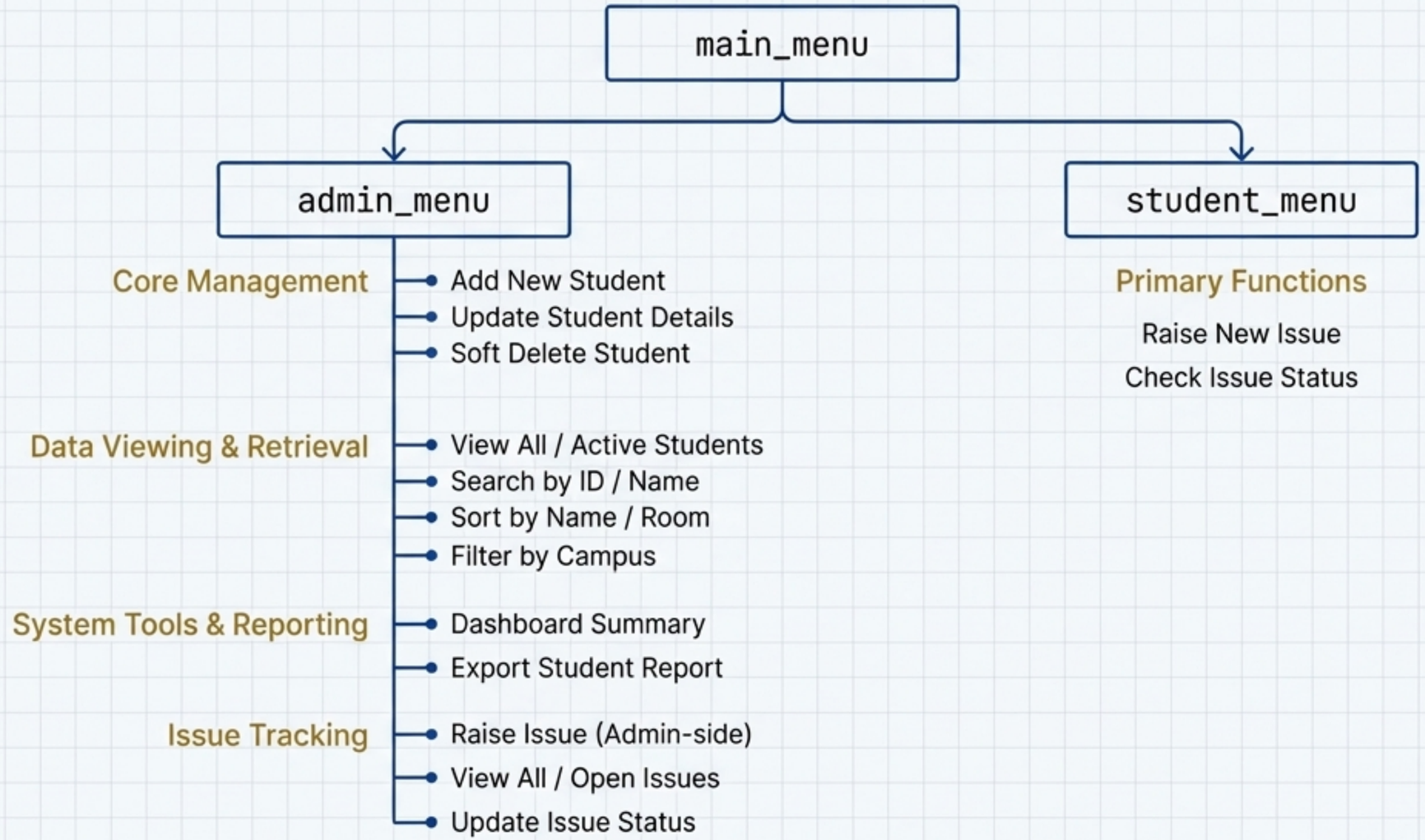
```
// Simplified save_data()
fp = fopen(STUDENT_FILE, "wb");
if (fp != NULL) {
    fwrite(&student_count, sizeof(int), 1, fp);
    fwrite(students, sizeof(Student), student_count, fp);
    fclose(fp);
}
```

File Manifest

File Name	Description
`students.dat`	Stores all student records.
`tickets.dat`	Stores all issue tickets.
`students_report.txt`	Generated on-demand for data export.

Navigating the System: A Dual-Portal Menu Architecture

A clear, hierarchical menu system guides users to the functionality they need.



Feature Deep Dive: Core Student Management

Building a reliable student database with robust CRUD operations.

`add_student()`- Adding a Record

Collects over 20 fields of data, from name to fee status.

- **Built-in Validation**: Performs a `find_student_by_id()` check before adding to prevent duplicate student IDs.
- Automatically sets `is_active` to 1.

`update_student()` - Modifying a Record

- **User-Centric Design**: Implements a nested menu system, grouping fields into logical categories (Basic, Contact, Academic, etc.). This prevents overwhelming the user with a monolithic form.
- Allows for precise, targeted updates without re-entering unchanged data.

`delete_student()`- The Soft Delete

Finds the student record by ID and sets the `is_active` flag to 0.

- **Key Benefit**: The student's data is preserved for historical records but is hidden from standard active views, representing best practice for data handling.

Feature Deep Dive: Advanced Data Retrieval

Providing administrators with flexible tools to find and inspect student information quickly.

ID	NAME	ROOM	CAMPUS	ACTIVE
101	Ananya Sharma	A-10	SSN	YES
102	Rohan Gupta	B-05	SNU	YES
103	Priya Singh	C-12	SSN	NO

Comprehensive vs. Filtered Views

`view_all_students()`

Displays a complete list, including those marked as inactive.

`view_active_students()`

The default operational view, showing only current students by checking `` `is_active == 1` ``.

Targeted Search Capabilities

`search_student_by_id()`

Precise lookup for a single record. Returns the exact match or 'Not Found'.

`search_student_by_name()`

Flexible partial search. Uses the `` `strstr()` `` function to find any student whose name contains the search substring.

Feature Deep Dive: Organising and Filtering Data

Transforming raw data into organised lists for easier analysis and administration.

In-Memory Sorting

(Bubble Sort Implementation)

- `sort_students_by_name()` : Re-orders the global `students` array alphabetically based on the `name` field.
- `sort_students_by_room()` : Re-orders the array based on the `room_no` field for logistical planning.

After sorting, the system automatically calls `view_all_students()` to display the newly ordered results.

Attribute-Based Filtering

- `view_students_by_campus()` : Prompts the admin for a campus ('SSN' or 'SNU') and displays only the matching records. This demonstrates a practical, real-world filtering requirement.

Feature Deep Dive: System Insights and Reporting

Providing high-level summaries and data export functionality for comprehensive oversight.

The Admin Dashboard (`dashboard()`)

An on-demand summary of the entire system state.

200 Total Students	**195** Active Students	**120** SSN Campus Students
75 SNU Campus Students	**85** Total Issues Raised	**12** Currently Open Issues

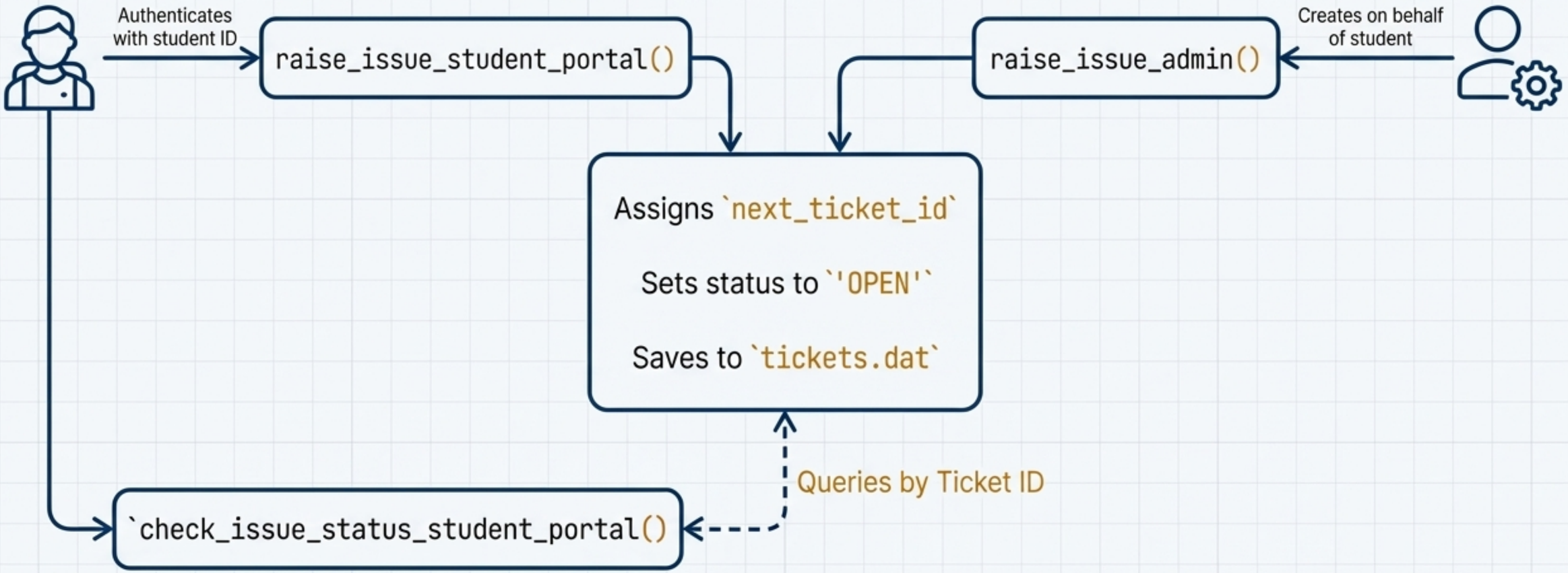
Text File Export (`export_students_report()`)

Generates a detailed, human-readable report named `students_report.txt`.

- Iterates through every student and writes all 20+ fields in a clean, formatted layout.
- Ideal for printing, archiving, or sharing data outside the system.

The Issue Tracking System: From Submission to Resolution

A streamlined workflow for managing student-reported maintenance and hostel issues.



Administrative Tools for Issue Management

Empowering administrators to view, filter, and update the status of all raised tickets.

Viewing the Queue

- `view_all_tickets()`: Provides a complete historical log of all issues ever submitted.
- `view_open_tickets()`: The primary administrative view, filtering the list to show only tickets with the status `'OPEN'`.

update_ticket_status()

Admin selects a ticket by its ID. A menu provides options to change the status.

Reflects the real-world workflow of addressing a maintenance task.

Highlights of Thoughtful Implementation

Design choices that prioritise robustness, user experience, and data integrity.

Data Preservation via Soft Deletes

Using an `is_active` flag instead of `DELETE` operations prevents data loss and maintains relational integrity. A mature approach to data management.

Robust Input Handling

Custom helper functions like `read_int()` and `clear_input_buffer()` manage input and validate data types, preventing common crashes from invalid user input.

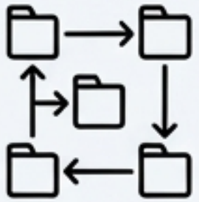
User-Centric Update Menu

The nested `switch` statement in `update_student()` is a deliberate UX choice. It groups 20+ editable fields into manageable categories, making the interface intuitive.

Transactional Data Saving

The `save_data()` function is called after *every* single administrative action. This "commit-after-change" strategy ensures maximum data durability, even in an unexpected program termination.

System Synthesis: A Complete and Cohesive Solution



Structured Data Models

Clear, comprehensive `structs` form a solid foundation.

Reliable Persistence

Binary file I/O ensures data is always saved and available.

Clear Role Separation

Distinct Admin and Student portals provide secure, tailored user experiences.

Comprehensive Feature Set

From CRUD and advanced search to dashboarding and issue tracking, the system provides a full suite of management tools.

The SSN Hostel Management System demonstrates the power of structured C programming to create a robust, feature-rich, and reliable administrative application from the ground up.