# SCHOOL MANAGEMENT SYSTEM — DATA STRUCTURES IN ACTION

Course: Data Structures and Algorithms

Semester: 1 – 2024/2025

Instructor: Dismas Kitaria

Title: Modular School Management System

Due Date: 24 October 2025

---

## 1. Objective

The goal of this project is to design and implement a modular School Management System using C++. Each module shows how a specific data structure can be used to solve real problems in a school. Five main data structures were used — one for each module.

## 2. System Architecture Overview

The prototype is divided into five modules, each focusing on a different part of school operations:

|  |  |  |
| --- | --- | --- |
| Module | Core Function | Data Structure Used |
| 1 | Student Registry | Linked List |
| 2 | Course Scheduling | Circular Queue |
| 3 | Fee Tracking | AVL Tree |
| 4 | Library System | Map (Hash Map alternative) |
| 5 | Performance Analytics | Matrix |

## 3. Module-by-Module Design and Implementation

### 3.1 Student Registry — Linked List

Purpose: Used to store and manage all registered students.

Why: A linked list allows easy adding and deleting of records without moving data in memory.

|  |  |  |
| --- | --- | --- |
| Operation | Description | Time |
| Add Student | Insert new student node | O(1) |
| Remove Student | Delete node by ID | O(n) |
| Find Student | Search by ID | O(n) |
| List All | Display all students | O(n) |

### 3.2 Course Scheduling — Circular Queue

Purpose: Used to assign students to courses in the order they register.

Why: A circular queue saves space by reusing empty slots when students are removed.

|  |  |  |
| --- | --- | --- |
| Operation | Description | Time |
| Enqueue | Add student to queue | O(1) |
| Dequeue | Remove student | O(1) |
| Check Full/Empty | Verify status | O(1) |

### 3.3 Fee Tracking — AVL Tree

Purpose: Used to store and manage fee transactions in sorted order.

Why: An AVL tree keeps the data balanced, making searching and insertion faster.

|  |  |  |
| --- | --- | --- |
| Operation | Description | Time |
| Insert Transaction | Add new fee record | O(log n) |
| Search | Find record by ID | O(log n) |
| Delete | Remove a record | O(log n) |
| Traverse | List all records | O(n) |

### 3.4 Library System — Map

Purpose: Used to store and track book records by ISBN.

Why: A map allows quick lookups and updates of book information.

|  |  |  |
| --- | --- | --- |
| Operation | Description | Time |
| Add Book | Insert new book record | O(log n) |
| Borrow Book | Decrease available copies | O(log n) |
| Return Book | Increase available copies | O(log n) |
| List Books | Show all books | O(n) |

### 3.5 Performance Analytics — Matrix

Purpose: Used to store marks for several students and subjects.

Why: A matrix helps calculate averages and find the top-performing student.

|  |  |  |
| --- | --- | --- |
| Operation | Description | Time |
| Set Mark | Assign score to student | O(1) |
| Row Average | Compute per student | O(m) |
| Column Average | Compute per subject | O(n) |
| Top Performer | Find highest average | O(n × m) |

## 4. Flow Diagrams / Pseudocode

Example: Student Registration Process

START  
Input Student Details  
Create Node(Student)  
Insert Node at head of LinkedList  
Display Updated Registry  
END

## 5. Performance Summary

|  |  |  |  |
| --- | --- | --- | --- |
| Module | Structure | Avg Time | Space |
| Student Registry | Linked List | O(n) | O(n) |
| Course Scheduling | Circular Queue | O(1) | O(n) |
| Fee Tracking | AVL Tree | O(log n) | O(n) |
| Library System | Map | O(log n) | O(n) |
| Performance Analytics | Matrix | O(n × m) | O(n × m) |

## 6. Ethical Reflection

• Fairness: Course allocation follows first-come-first-served order.  
• Privacy: Student and financial data must be kept confidential.  
• Transparency: Each operation can be tracked and explained.  
• Accountability: Records are traceable for auditing.

## 7. Testing and Sample Data

Each module includes a few sample records for easy testing:  
• 3 students for the registry and performance module.  
• 3–5 transactions for fee tracking.  
• 2 books for library operations.  
• Queue size 3–5 to demonstrate rotation.

## 8. Compilation and Execution

Each module runs on its own. Use these commands to compile and test:

g++ StudentRegistry\_LinkedList.cpp -o StudentRegistry  
./StudentRegistry  
  
g++ CourseScheduling\_CircularQueue.cpp -o CourseScheduling  
./CourseScheduling  
  
g++ FeeTracking\_AVLTree.cpp -o FeeTracking  
./FeeTracking  
  
g++ LibrarySystem.cpp -o LibrarySystem  
./LibrarySystem  
  
g++ PerformanceAnalytics\_Matrix.cpp -o PerformanceAnalytics  
./PerformanceAnalytics

## 9. Conclusion

This project shows how basic data structures can be used to build a simple school management system. By using linked lists, queues, trees, maps, and matrices, we can handle data efficiently and clearly. The design is simple, modular, and easy to understand for a university-level project.