SYSTEM SOFTWARE DEVELOPMENT

STUDENT GRADE MANAGMENT SYSTEM

MEET OUR TEAM

Swati Mehta

Divyanshi Kashyap

Muhammad Jahanzaib

INTRODUCTION

PROBLEM STATEMENT

- Managing student grades manually is inefficient, error-prone, and lacks proper data organization. Our system automates this process, ensuring accuracy and ease of access.
-

PURPOSE & USERS

- Instructors: Enter, update, and analyze student grades efficiently. They can also generate performance reports and track class progress over time.
- Students: View grades securely using their student ID, ensuring transparency and quick access to their academic records. The system also provides insights into overall class performance and individual progress.
-

SOLUTION OVERVIEW

- File Handling: Ensures persistent data storage and retrieval.
- Dynamic Memory & Linked Lists: Efficient management of student records.
- Recursion: Speeds up searching and displaying records.
- Statistical Analysis: Provides insights like highest/lowest scores and averages.
-
-
-

MAIN FUNCTIONS

Student Management
createStudentNode(): This function creates a student node, enabling dynamic memory usage and easy insertion into the list.
addStudent(): Adds students to the system, ensuring no duplicate student IDs and calculating their GPA in the process.
deleteStudent(): Allows the removal of students from the system.
modifyStudent(): Facilitates modifying student data, such as updating grades and saving them back to the system.
freeStudentList(): Frees up memory once student records are no longer needed.
searchStudent(): Enables the search of student records using their unique student ID.
calculateStudentEfficiency(): Calculates and categorizes students based on their GPA to determine efficiency levels.
sortStudents(): Sorts students based on GPA, Name, or Student ID in ascending order.

2. Data Visualization
- displayStudent(): Displays student data in a clean, tabular format, making it easy for users to compare student information.
- visualizeStudent(): Creates a bar chart to visualize students' CGPA, using color coding to show high and low GPAs.
- printCourseBarChart(): Generates a bar chart visualizing student grades for each course, showing grade performance with color-coded bars.

3. File I/O Functionalities
- ReadCSV(): Reads student data from a CSV file, loading it into the system. If the file doesn't exist, it will create an empty one.
- WriteCSV(): Writes changes from the system (additions, modifications, or deletions) back to the CSV file to keep data up-to-date.

4. GPA Calculation
- calculateGPA(): Calculates a student's GPA by converting their grades into grade points. This function is used for sorting, efficiency analysis, and display purposes.

5. User Interface
- displayTitleScreen(): Displays the title screen with project details and prompts the user to press any key to continue.
- showMenu(): Presents the user with a menu of options and prompts them to select an action.

ARCHITECTURE , RECURSION, LINKED LISTS

Architecture Overview

- The project follows a modular design. The input/output module handles CLI interactions.
- The data module stores user entries using linked lists.
- The logic module processes operations like search, sort, and traversal. The file module handles saving/loading data.
- These modules are decoupled and communicate through function calls, which improves scalability and testing.

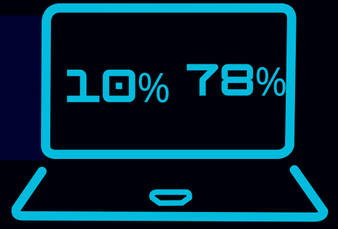
Recursion Usage

Recursion is used in several places for clean, elegant logic. For example:

- **Traversal:** Displaying the linked list recursively.
- **Sorting:** A recursive merge sort organizes data alphabetically or numerically.
- **Memory cleanup:** Recursive deallocation ensures no memory leaks. This approach makes the code concise and handles deeply nested data structures naturally.

Linked Lists

It is used to store and manage dynamic data entries—such as user inputs or task records.Each node contained the necessary data fields (e.g., name, ID, value) and a pointer to the next node in the list.
Dynamic Size: Unlike arrays, linked lists don't need a predefined size. Memory is allocated as needed—ideal for user-generated input.
Memory Control: Each node is allocated with malloc() and freed manually with free(). This allowed precise memory management, and we verified no leaks using tools like Valgrind.
Efficient Insert/Delete:In a linked list, each data item—called a node—is connected to the next using pointers.When we want to insert or delete a node, we update the pointers



file I/O

Data is saved using **fopen, fprintf**, and **fclose** for structured writing. When loading, the file is parsed using fscanf or getline. This enables persistence across program runs.

Makefile

All memory for nodes and buffers is dynamically allocated using malloc().To prevent memory leaks, every allocation is paired with a corresponding free().

memory allocation

The Makefile contains targets like all (compiles everything), clean (removes .o and binaries), and test (runs test scripts or sample inputs). It automates building and testing the system efficiently.

FILE I/O, MEMORY,MAKEFIE

TESTING & MEMORY VALIDATION

The application was tested using a combination of unit testing, functional testing, and edge case validation.
A dedicated test_main.c file was created to systematically validate all functionalities of the program

- Unit tests validated core functions like insert, delete, search, and sort.
- Functional tests checked how different modules interacted together.
- Edge cases included scenarios like empty input, very large inputs, duplicate values, and invalid commands

LET'S SEE SOME EXAMPLES

Unit

test_addStudent(): This test verified that a student could be added successfully. The output returned "Test Case Passed" if the student was added correctly

Functional Tests:

Adding multiple students and ensuring no duplicate IDs were allowed.
Modifying student data and confirming that changes were reflected correctly

Edge Cases

Extreme GPA values: For example, if an invalid grade was entered, the GPA calculation would assign a value of 0 for that specific grade.

Empty or corrupt CSV files: The program was tested to handle such scenarios gracefully by initializing an creating a new file.

MEMORY SAFETY CHECKS

MANUAL ALLOCATION & DEALLOCATION

We used malloc() to allocate memory for:

- New nodes in the linked list
- Buffers for input data

Every malloc() was paired with a free() to release memory after use.

VALGRIND FOR MEMORY VALIDATION

By running our program through Valgrind, we:

- Detected and fixed early memory issues
- Confirmed that all memory was freed properly
- Ensured no invalid memory operations occurred

USING RECURSION

Instead of manually freeing each node, we used recursive functions that:

- Visit each node in the list
- Free the current node
- Recurse to the next node

```
==========================================
        2263 Project: Student_Management_System
    MADE BY DIVYANSHI, Muhammad Jahanzib, Swati Mehta
==========================================
Press any key to continue...

File 'students.csv' not found! Creating a new file...

======== MENU ========
1. Add Student
2. Display Students
3. Delete Student
4. Table Visualization
5. Bar Chart Visualization
6. Sort Students
7. Search Student
8. Modify Student
10. Calculate Student Efficiency
9. Exit
======================
Enter your choice:
```

TEAM CONTRIBUTIONS

A **Divyanshi Kashyap (3769254):** Focused on implementing core functionalities such as createStudentNode(), addStudent(), and deleteStudent(). She also ensured proper dynamic memory allocation and deallocation using functions like freeStudentList() to maintain memory safety.

B **Swati Mehta (3765212):** Worked on data visualization features, including displayStudent() and visualizeStudent(), which provided a user-friendly interface for viewing student data. She also implemented the bar chart visualization for GPA and course grades

C **Muhammad Jahanzaib (3757073):** Took charge of file I/O operations (ReadCSV() and WriteCSV()) and GPA calculations (calculateGPA()). He ensured that the system could read from and write to CSV files seamlessly while maintaining data integrity.

Colour-coded Output

Implementing visualizations for GPA and individual course grades using ANSI escape codes was new to us

What we did:

We researched ANSI codes online and used color macros like GRN, RED, YEL, and CYN to give GPA bars in visualizeGPA and course-wise performance bars in printCourseBarChart.

File Handling with CSV

Reading and writing student data from/to students.csv introduced complications

What we did:

We used fscanf and fprintf carefully with fgets for name input

Dynamic Memory Management

One of the toughest parts was managing dynamic memory and ensuring proper allocation and deallocation, especially in recursive functions like sortStudents

We carefully designed recursive splitting (frontBackSplit) and merging (sortedMerge) functions, thoroughly testing edge cases to ensure correctness. The freeStudentList function was also introduced to clean up dynamically allocated memory safely.

CHALLENGES & LESSONS LEARNED

Conclusion & Future Work - PART 1

limitations or unfinished components.

- Grades: While grades >100 are set to 0, negative grades are not checked
- We don’t have a modification method to update student numbers, we only have a method to Bascially change grades for courses
- No validation for special characters for Name input in addStudent method

Possible improvements

- Work on Limitations and unfinished components
- Make sure that each input takes data in the Correct DatatType
- we can also include Department or Major role in Student list, which would give more meaning to our Data

Conclusion & Future Work - PART 2

Project accomplishments

- Linked List Implementation
- User Interface
- GPA Calculation
- Student Efficiency
- CSV Data Persistence
- Extra Functionalities like StudentEfficiency methd
- Made Test Cases for all functionalities

Future Improvements

- We can go for advanced searching and Filtering, on basis of Course name
- We can go for Role-Based Access Control
- We can also add Attendance Tracking
- We can create a functionality where we can add students to a specific class list and for that we need to make a different Class & Section list structure for the same

THANK YOU!

Thank you for watching our presenatation . Feel free to ask questions!