## Project Report: Student Record System

Overview

This report provides an overview of the Student Record System, a C-based application designed to manage student information efficiently. The system includes features like adding, displaying, searching, modifying and deleting student records.

Key Components:

1. Header Files:
    - `student_records.h`: Defines the `Student` structure and declares function prototypes for student operations.
    - `menu.h`: Declares functions for displaying the menu and getting user input.
    - `student_operations.h`: Declares functions for various student operations, such as adding, deleting, and modifying students.
2. Source Files:
    - `student_records.c`: Implements functions for adding, displaying, searching, modifying, deleting, sorting and calculating average marks of students.
    - `menu.c`: Implements functions for displaying the menu and handling user input.
    - `student_operations.c`: Implements functions for various student operations.
    - `main.c`: The main program that drives the application, handles user input, and calls functions from other modules.

Potential Problems and Solutions:

1. Memory Leaks:
    - Solution: Use `free` to release memory allocated using `malloc` and `realloc`.
    - Example: In the `deleteStudent` function, ensure that the memory of the deleted student is freed.
2. Invalid Input:
    - Solution: Implement robust input validation to handle invalid input, such as non-numeric values for roll numbers or marks.
    - Example: Use `scanf` with appropriate format specifiers and error checking to ensure valid input.
3. Array Overflow:
    - Solution: Dynamically allocate memory for the student array using `malloc` and `realloc` to handle a variable number of students.
    - Example: In the `addStudent` function, reallocate the array if it's full.
4. File I/O Errors:
    - Solution: Handle file I/O operations carefully, checking for errors like file not found, disk full, or permission issues.
    - Example: Use `fopen`, `fclose`, `fprintf`, and `fscanf` functions to read from and write to files.
5. User Interface:
    - Solution: Design a user-friendly interface with clear prompts and error messages.
    - Example: Use formatted output to display information neatly.

Testing and Debugging:

- Thoroughly test the application with various input values.
- Use a debugger to step through the code and identify errors.
- Consider using a testing framework to automate testing.

Future Enhancements:

- Implement a graphical user interface (GUI) using a library like Qt or GTK.
- Add features like exporting data to CSV or PDF formats.
- Improve the error handling and user experience.
- Explore advanced data structures like linked lists or trees for more efficient data management.

Group Member:

1. Bakare Muideen Adeleke
2. Emem Udoh
3. Glory Ighokido
4. Gold Osunde
5. Iheanyichukwu Anoruo
6. Jonathan Ibie
7. Osaretin Igiebor
8. Inemesit Gibson
9. Musa Abdulkabir
10. Boluwatife Aroyewun