| **Technical Report: Class Management System (CMS) Project**  **Team ID:**  P14\_8  **Team Members:**  James Hong (2402817)  Choong Wan Qin (2402311)  Yek Yi Wei (2402288)  Brian Liew (2402342)  Wayne Lim (2402220)  **Course:**  INF1002-Programming Fundamentals  **Professor:**  Dr Frank Guan  **Submission Date:**  29/11/2024 |
| --- |

**Content**

1. Technical Description
2. Main Features
3. Program Code Structure
4. Data Structures and Rationale
5. Contribution

**1. Introduction**

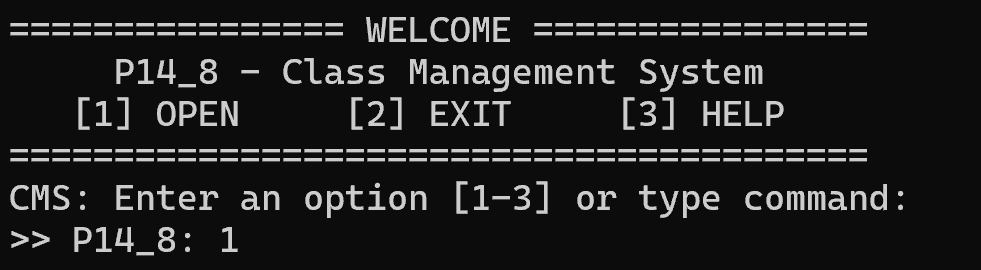
This technical report presents an overview of a Class Management System (CMS), a command-line application developed in the C programming language. The CMS is designed to interact with a text-based database file, “P14\_8-CMS.txt”, which stores student records in fields such as student ID, name, programme, marks, and grade. By combining a streamlined interface with essential operations like insertion, querying, updating, and deletion, the CMS simplifies the management of these records.

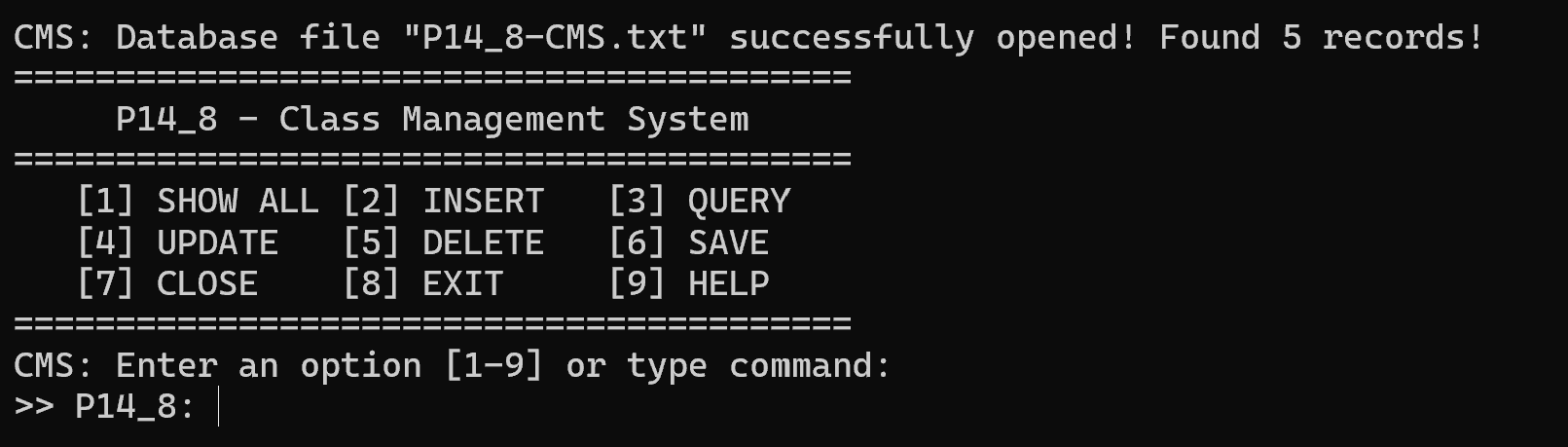
In addition, to support dynamic operations, a linked list data structure is employed, enabling efficient insertion and deletion of records during runtime compared to a standard array. This design choice enhances flexibility and adaptability when managing records in memory.

**2. Main Features**

2.1 Open Database File

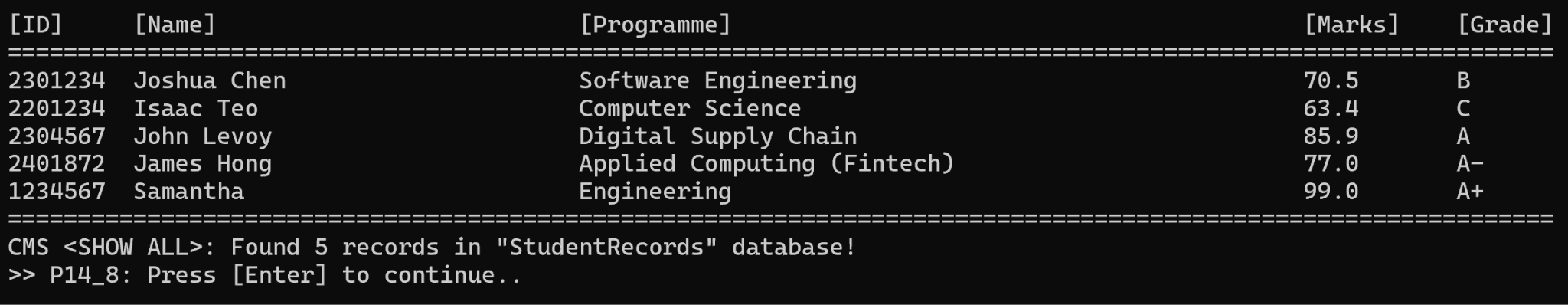
Loading student records from file into memory.





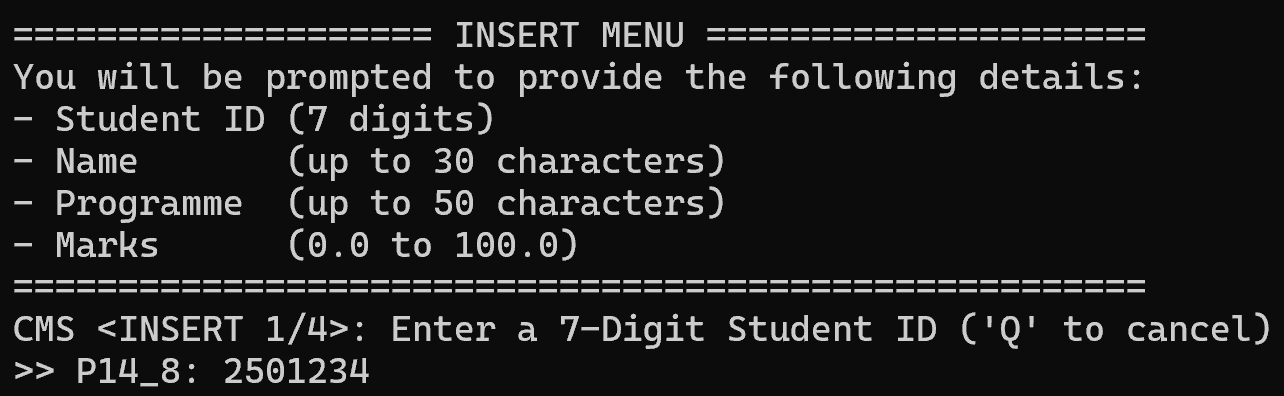
2.2 Show All Records

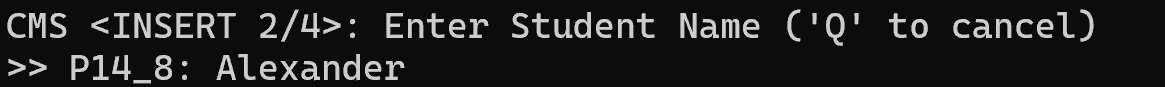
Show all student records in a well-structured, tabular format.

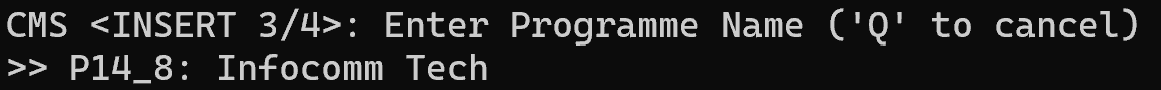


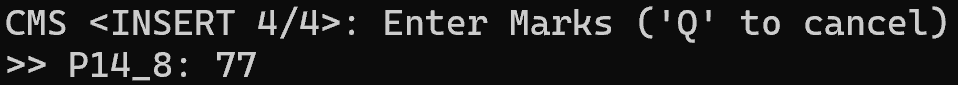
2.3 Insert Record

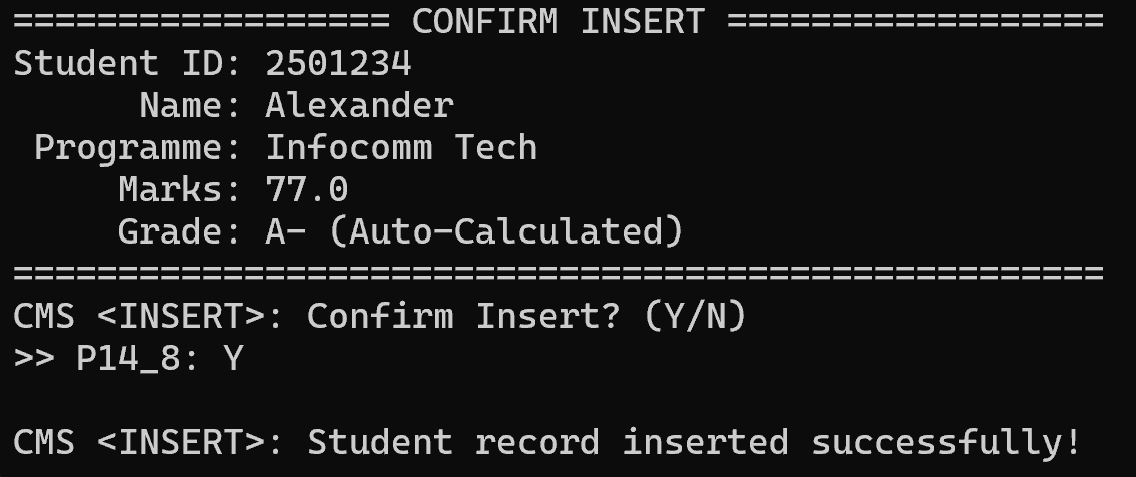
Allows addition of new student records by prompting for inputs.





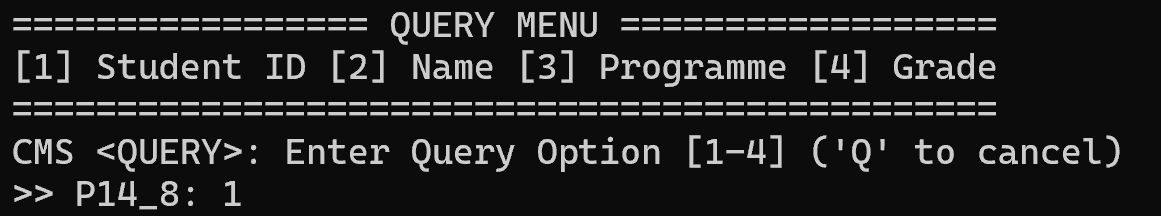


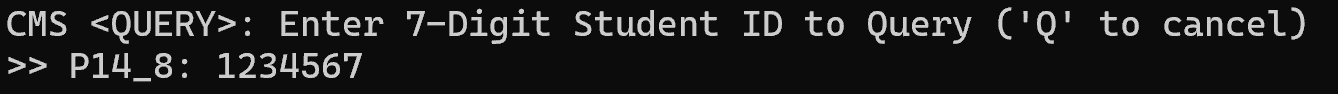


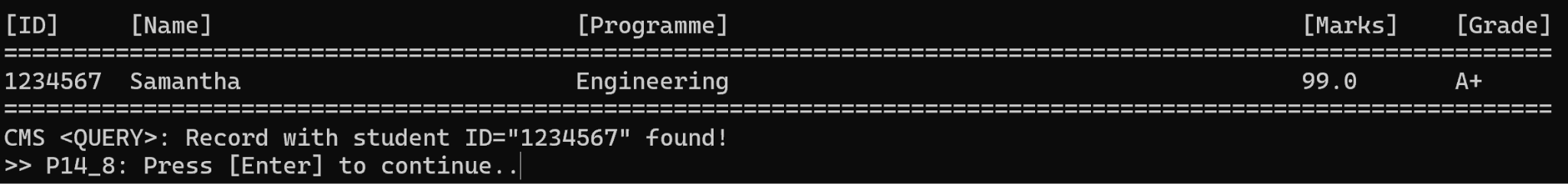


2.4 Query Records

Search for records based on student ID, name, programme or grade.

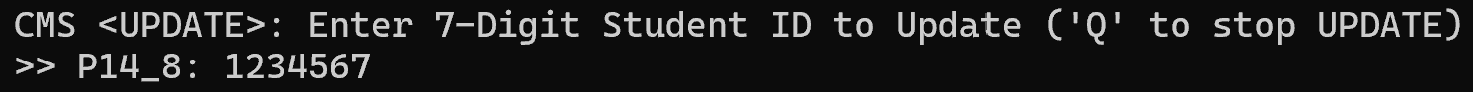


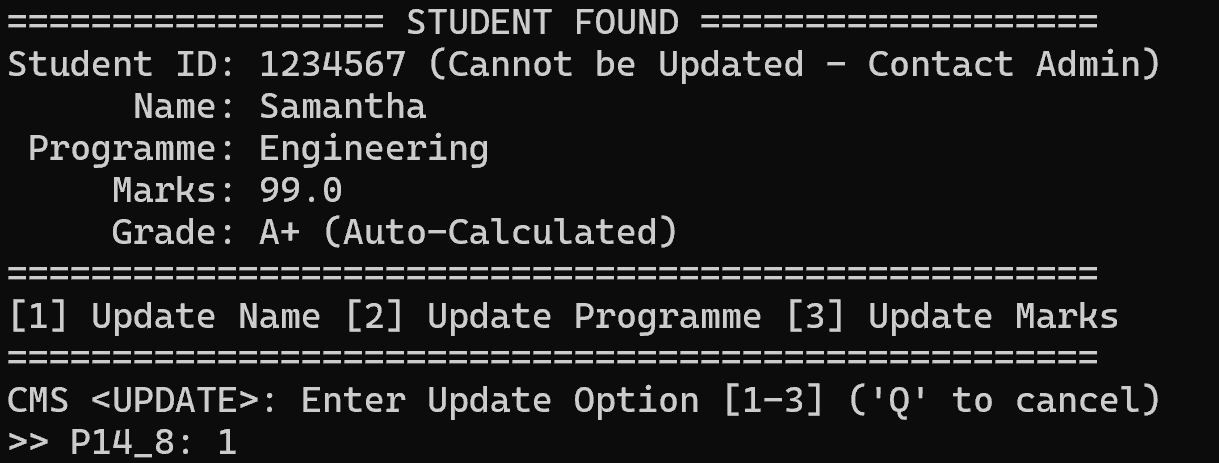


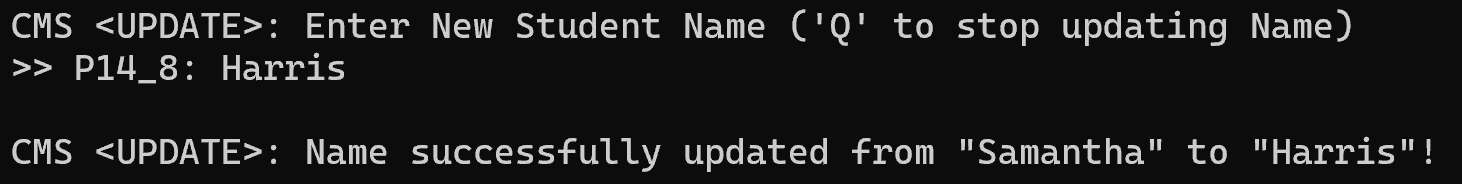


2.5 Update Record

Allows modifications to existing records, including updates to name, programme, or marks.

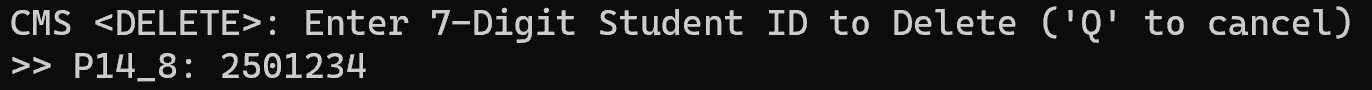


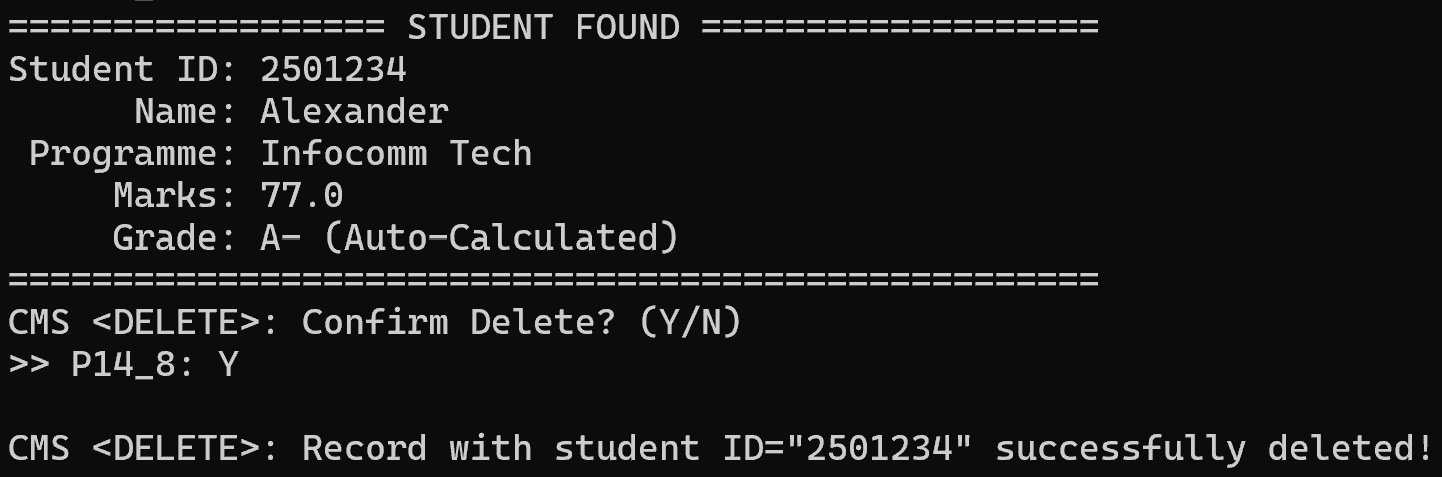




2.6 Delete Record

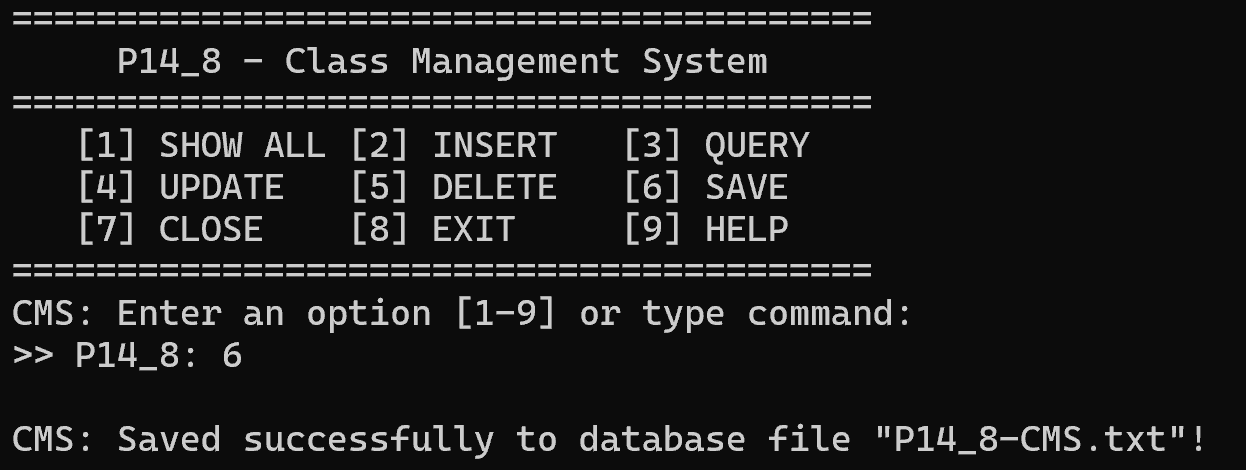
Delete existing student records using student ID.





2.7 Save Changes

Saves all in-memory changes back to the file.

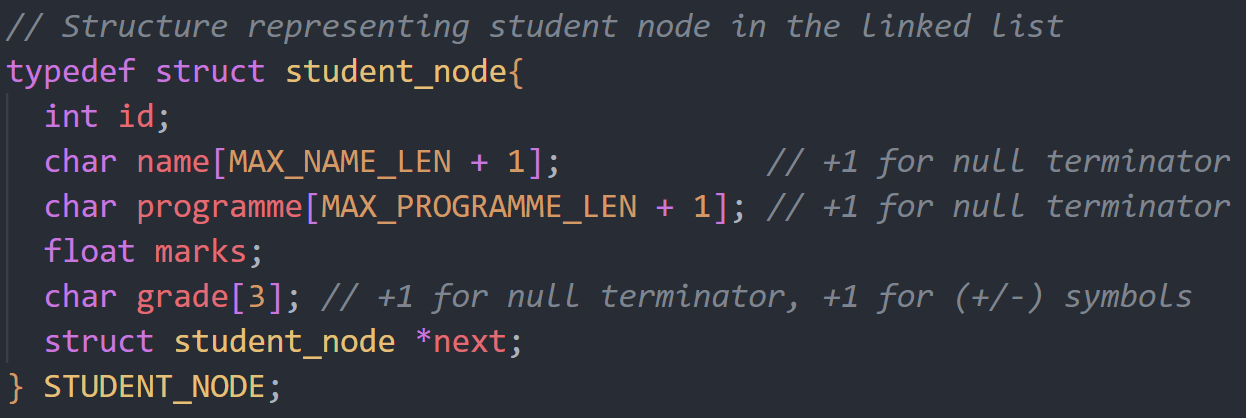


**3. Program Code Structure**

The CMS program is organized into modular components, each designed to handle specific aspects of the system. Below is the breakdown of the structure.

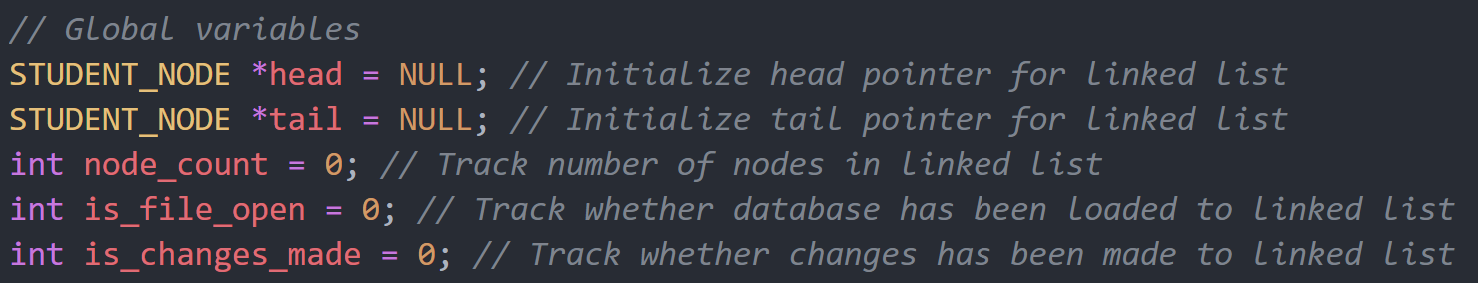
3.1 Data Structure

The program uses a linked list data structure for storage of student records during runtime. Each record is represented as a node in the list, defined by the “STUDENT\_NODE” structure.



3.2 Global Variables

The program uses several global variables to manage the state and the linked list.



3.3 Main Functions (File Handling)

1. open\_db(): Opens the database file and reads student records into the linked list. Skips metadata lines using skip\_header\_lines().
2. save\_db(): Writes the current state of the linked list to the database file. Formats records in CSV format for consistency.
3. close\_db(): Prompts the user to save unsaved changes before freeing memory and closing the database.

3.3 Main Functions (Core Operations)

#### 1.show\_all\_records: Displaying All Records

* Displays all student records in a tabular format.
* Handles cases where the linked list is empty by providing appropriate user prompts.
* Iterates through the linked list to retrieve and present all records, including a record count for user clarity.

#### 2. insert\_record: Adding New Records

* Guides the user through a step-by-step process to input details such as Student ID, Name, Programme, and Marks.
* Validates inputs to maintain data integrity:
  + Ensures Student ID is a unique 7-digit numeric value.
  + Limits Name and Programme fields to specified character limits.
  + Restricts Marks to a range of 0 to 100.
* Includes duplicate-checking logic to prevent insertion of records with existing Student IDs.
* Provides a confirmation step, allowing users to review and confirm data before insertion.
* Manages memory allocation for new nodes and updates linked list pointers to maintain structural consistency.

#### 3. query\_record: Searching for Records

* Implements a dynamic query system with multiple search criteria:
  + Student ID: Supports exact and partial matches for flexible searching.
  + Name and Programme: Enables case-insensitive and partial searches to improve usability.
  + Grade: Allows searches for specific grades (e.g., "A+") or generalized grades (e.g., "A" matches "A+", "A-", and "A").
* Includes comprehensive input validation and error handling for invalid or empty inputs.
* Displays results in a formatted table and provides clear feedback when no records match the search criteria.

#### 4. update\_record: Modifying Existing Records

* Provides functionality to locate a record by Student ID and modify its attributes (Name, Programme, Marks, or all fields).
* Ensures input validity for updated fields to maintain data integrity.
* Incorporates a confirmation system to allow users to review and approve changes before finalizing updates.
* Automates grade recalculation based on updated marks to ensure consistency.
* Handles edge cases, including when records are not found or when updates are canceled by the user.

#### 5. delete\_record: Removing Records

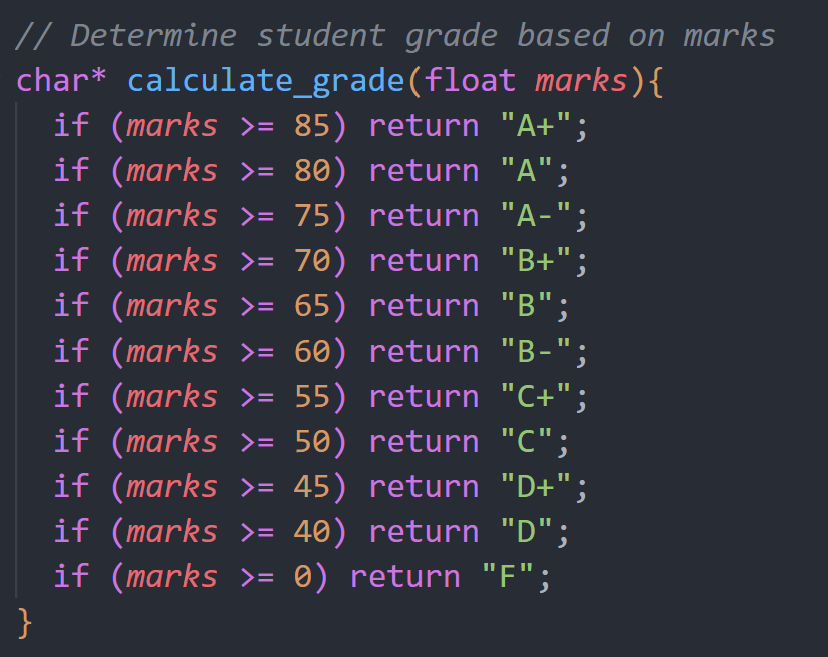
* Facilitates the deletion of records based on Student ID, with a confirmation prompt to prevent accidental deletions.
* Updates linked list pointers to maintain structural integrity after deletion.
* Frees allocated memory for removed nodes and ensures proper handling of edge cases, such as deleting the head or tail node.
* Provides clear feedback for both successful deletions and cases where no matching records are found.

3.4 Utility Functions

Utility functions are the backbone of the Class Management System (CMS), supporting core operations by managing input, memory, and database handling. These functions ensure efficient and error-free execution of the system.

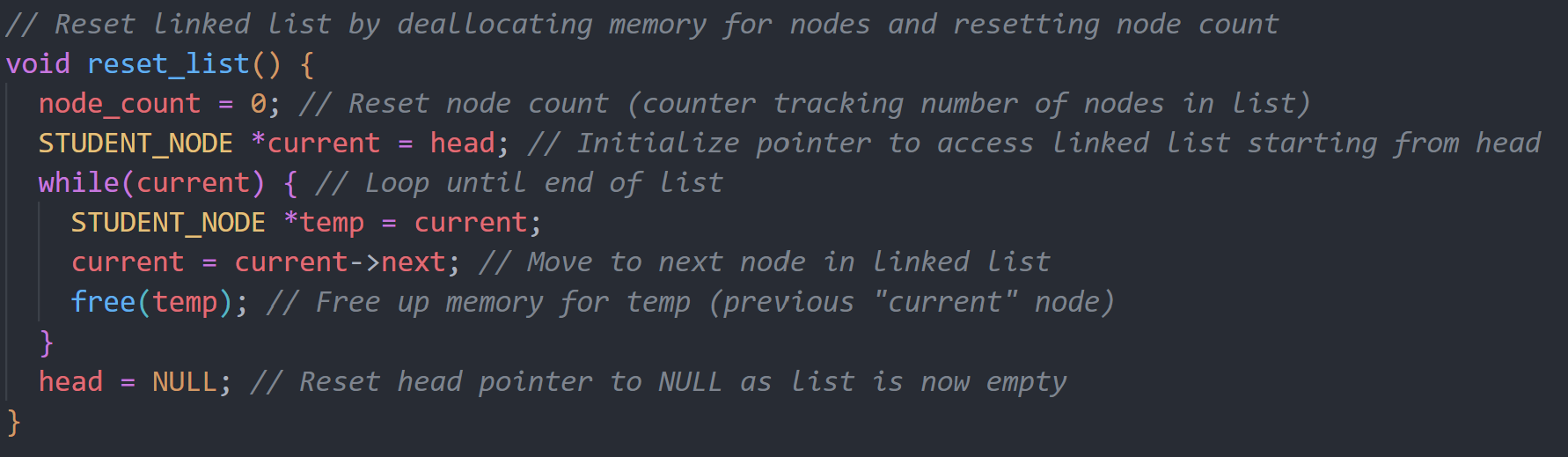
1. calculate\_grade (float marks)

* Purpose: Automatically calculates and assigns a grade based on the provided marks.
* Logic: Uses conditional checks for predefined ranges of marks and returns the corresponding grade string.
* Impact: Simplifies grade assignment for marks input, ensuring consistency and reducing manual errors.



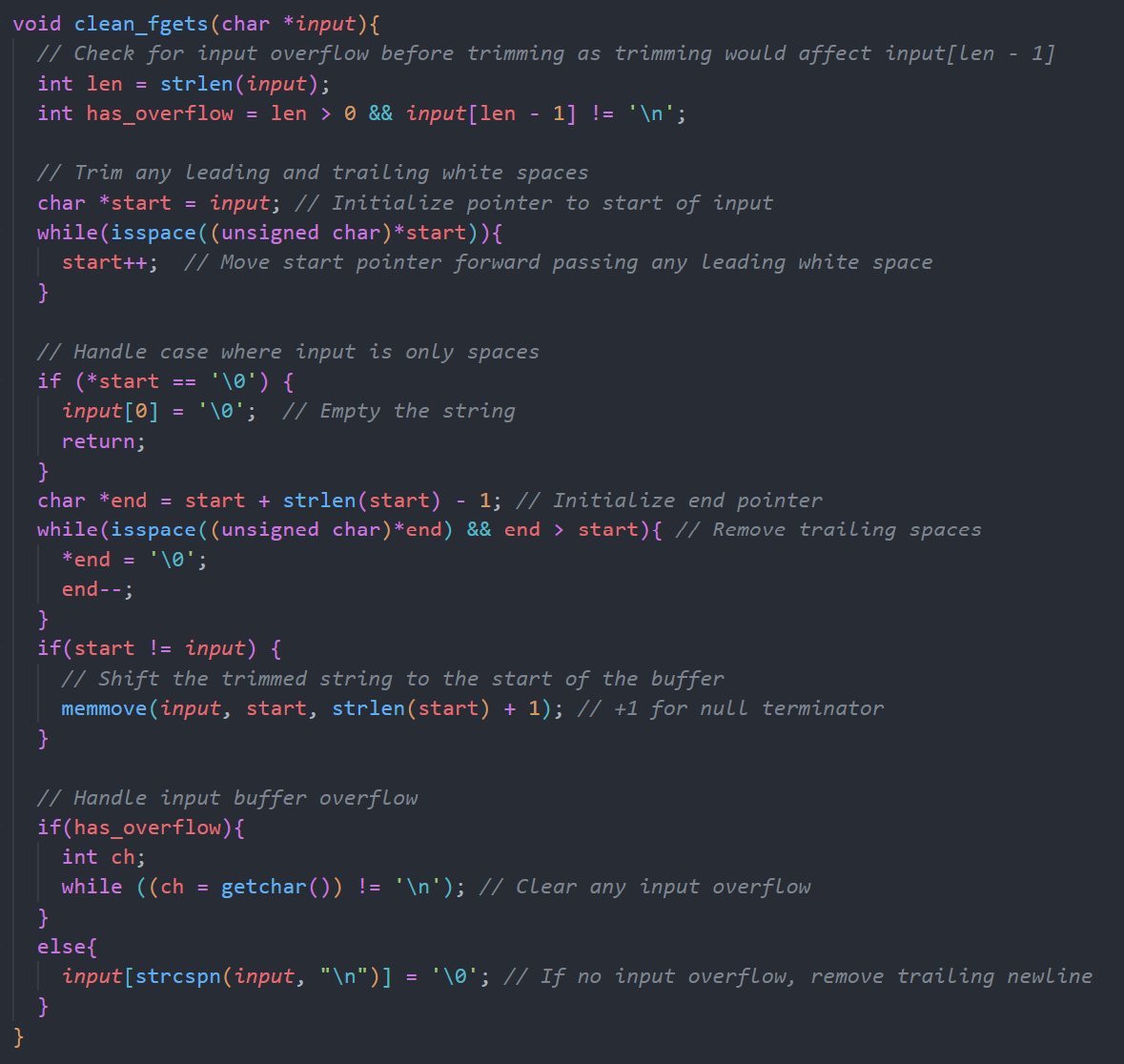
1. reset\_list()

* Purpose: Frees memory allocated for the linked list and resets all global pointers.
* Logic: Iterates through the linked list, deallocates memory for each node, and sets head and tail to NULL.
* Impact: Prevents memory leaks and ensures efficient memory management.



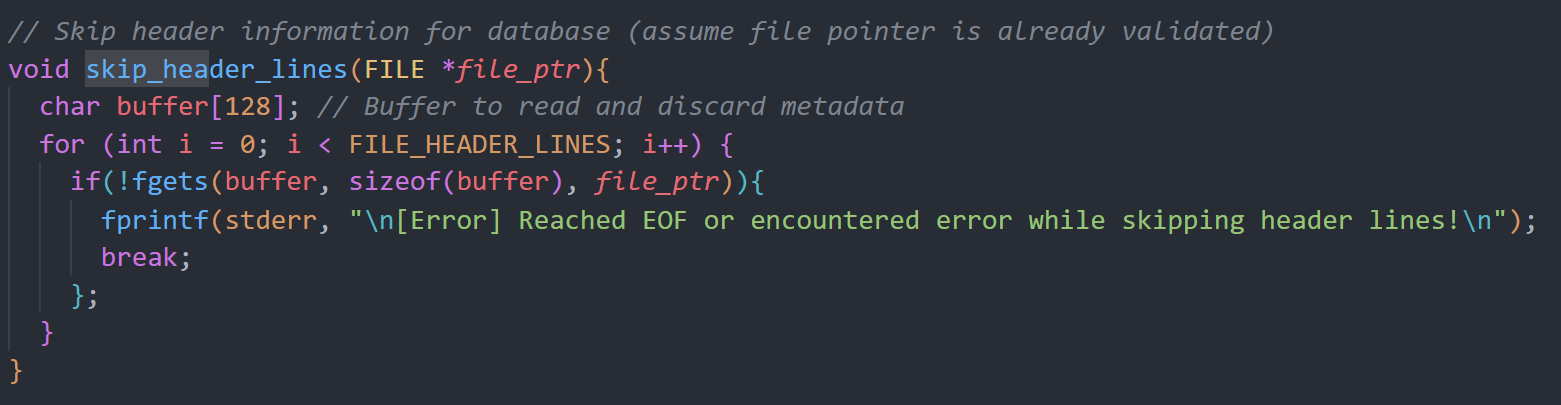
3) clean\_fgets(char \*input)

* Purpose: Cleans user input by removing trailing newline characters and trimming whitespace.
* Logic:
* Trims leading and trailing spaces.
* Removes newline characters.
* Handles input buffer overflow scenarios.
* Impact: Ensures cleaner and more robust user input, reducing the risk of invalid data.



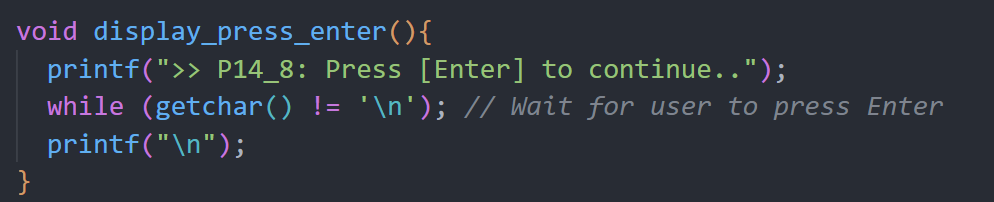
#### 4) skip\_header\_lines(FILE \*file\_ptr)

* Purpose: Skips metadata lines in the database file to start reading actual records.
* Logic: Reads a predefined number of header lines using fgets to move the file pointer forward.
* Impact: Facilitates smooth file parsing by ignoring non-essential lines.



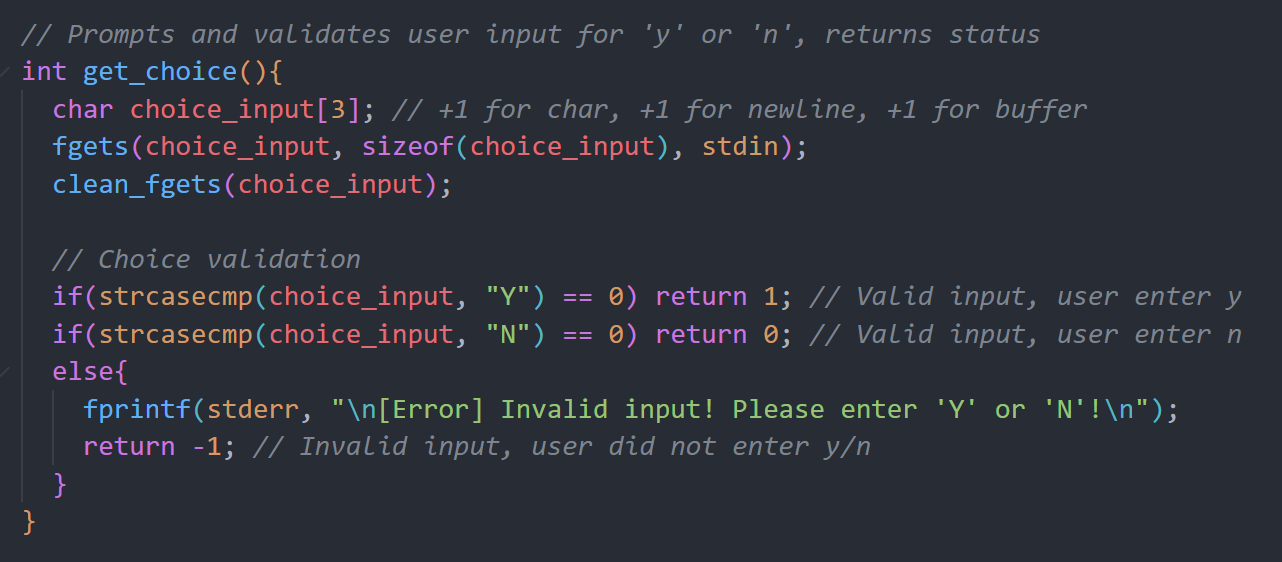
5) display\_press\_enter()

* Purpose: Prompts the user to press the Enter key to proceed.
* Logic: Pauses execution until the user presses Enter.
* Impact: Enhances user interaction by providing intuitive navigation prompts.



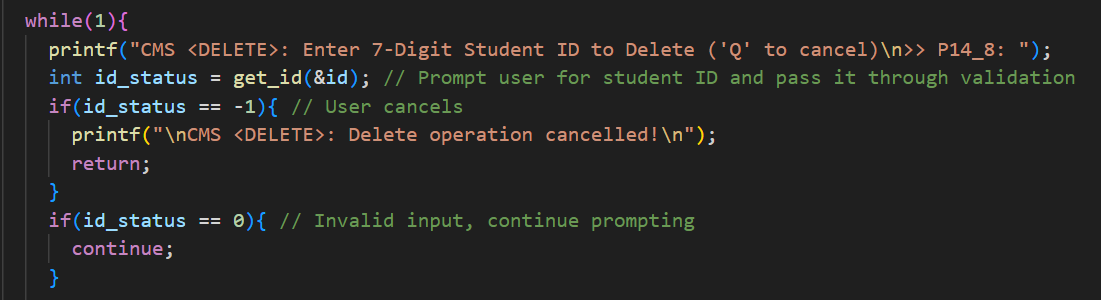
6) get\_choice()

* Purpose: Validates user input for yes or no choices.
* Logic:
* Accepts only "Y/y" or "N/n".
* Returns appropriate status codes for valid or invalid inputs.
* Impact: Ensures consistent handling of user confirmations.



7) get\_id(int \* id)

* Purpose: Ensures the input is exactly 7 numeric characters, user can also cancel the operation by entering ‘Q’.
* Logic: Validates that the input is numeric and exactly 7 characters long (strspn(id\_input, "0123456") == MAX\_ID\_LEN, and converts valid input to an integer using atoi and assigns it to the id pointer.
* Impact: Prevents invalid IDs from being processed, ensuring database integrity.
* Simplifies validation logic within the delete\_record function by centralizing it in this utility.



**4. Data Structures and Rationale**

The Class Management System (CMS) employs a linked list as the primary data structure for storing student records during runtime. Arrays were also considered but were deemed less suitable for the CMS's needs.

4.1 Consideration for application

The CMS allows dynamic management of student records, with frequent operations such as adding new records, deleting records, and updating existing ones. These operations benefit from a flexible data structure that can adapt to changes in the dataset size without the need for reallocation or restructuring.

4.2 Why Linked List was chosen

The decision to use a linked list was based on its ability to dynamically allocate memory as needed. Unlike arrays, which require a predefined size, a linked list grows or shrinks based on the number of records. This makes it ideal for applications like the CMS, where the dataset size is variable and cannot be predicted beforehand.

In addition, insertion and deletion operations are particularly efficient in a linked list, as they involve updating pointers rather than shifting elements. For instance, adding a new student record to the end of the list or removing a record from the middle can be done in constant time O(1), making these operations faster than in an array, where elements need to be shifted. This efficiency aligns well with the CMS’s requirement to frequently modify the dataset.

4.3 Why an Array was not chosen

An array offers advantages over linked lists such as random access, where accessing an element by its index takes constant time O(1). This would make an array more suitable for applications focused primarily on querying records, where the dataset remains relatively static. However, the CMS’s primary requirements involve dynamic operations like adding and deleting records, which are not efficiently handled by arrays. Adding a new record to an array requires shifting elements, especially when inserting in the middle, resulting in an O(n) time complexity. Similarly, deleting a record requires shifting subsequent elements to maintain order, which is computationally expensive. These inefficiencies make arrays less suitable for the CMS.

**5. Contribution**

James Hong (2402817)

* open\_db(): Contributed to loading records from the database file into memory as a linked list. Implemented file handling logic, including skipping metadata lines, parsing CSV fields, and error handling for malformed lines or file issues.
* save\_db(): Wrote the linked list back to the database file in a structured format, ensuring all records are saved with consistent formatting and handling edge cases such as empty datasets.
* show\_all(): Implemented functionality to traverse the linked list and display all student records in a tabular format. Included formatting for alignment and clear output, along with feedback when no records are available.
* User Interface:
* Developed display\_menu() to present a dynamic interface based on the system’s current state (e.g., whether the database file is open).
* Created run\_cmd() to map user inputs to the appropriate functions, handling command validation and error messages for unknown commands.
* Utility Functions:
* reset\_list(): Frees all nodes in the linked list to prevent memory leaks.
* clean\_fgets(): Sanitizes user input to remove extra spaces and trailing newlines, ensuring input consistency across the program.

Choong Wan Qin (2402311)

* Contributed to the development of the insert function by implementing comprehensive validation checks to ensure data integrity. These checks included restricting the ID field to accept only numeric values, ensuring the Name and Program fields contain only alphabetic characters, and validating the Marks field to allow numeric values with up to two decimal places. These measures were implemented to maintain consistency and prevent invalid data entry into the system.

Brian Liew (2402342)

* Contributed to the implementation of the Update Record functionality in the Class Management System (CMS). This involved creating the logic to locate student records using their unique ID and providing options to update fields such as name, programme, or marks. Additionally, I integrated automatic grade recalculation when marks are updated, enhancing the system's efficiency and usability. My contribution ensures that users can modify records effectively while maintaining data integrity.

Yek Yi Wei (2402288)

query\_record(): a key feature enabling dynamic and user-friendly querying of student records based on various criteria. This function supports multiple search options, including:

* Dynamic Linked List Traversal: Implemented logic to traverse the linked list and retrieve records matching user criteria. Substring matching and case-insensitive comparisons were utilized for flexible searches.
* Input Validation and Error Handling: Developed robust validation for numeric inputs (e.g., Student ID) and text fields (e.g., Name, Programme) to ensure data integrity. Provided clear error messages and retry prompts for invalid or empty inputs.
* Interactive Query Menu: Created a loop-based menu system enabling users to navigate query options seamlessly or cancel searches with intuitive commands.
* Result Presentation: Designed formatted table outputs for query results, improving clarity and user experience.
* Flexible Search Options: Supported partial matches for Student ID, Name, and Programme queries, as well as specific and generalized grade searches.

Efficient, reliable, and user-friendly access to student records. Its robust design and comprehensive validation ensure accuracy, while the flexible query options and clear interface improve usability for administrative and academic purposes.

remove\_extra\_spaces(): ensures clean and properly formatted input by removing unnecessary spaces from user-provided strings. The function is critical for maintaining data consistency and improving query accuracy.

* Whitespace Trimming: Removes leading and trailing spaces.
* Consecutive Space Collapse: Reduces multiple spaces between words to a single space.
* Efficient In-Place Processing: Optimized memory usage without additional storage.

This utility enhances query accuracy, improves data presentation, and ensures robust handling of user input, supporting features like the query record system effectively.

User Interface:

* Developed display\_menu() to present a dynamic interface based on the system’s current state.
* Developed the table design to show the database.
* Developed display\_press\_enter() for better UI.

Wayne Lim (2402220)

* I contributed to the development of the Delete Record function. This involved designing and implementing logic to locate and remove student records from a linked list data structure based on a unique student ID. The function was developed to ensure efficient operations while maintaining data integrity and a user-friendly interface. Some key aspects of my contributions include:
  + Validation Mechanism: Integrated input validation to ensure the Student ID is a valid 7-digit numeric value. This allowed users to cancel the operation at any point by entering a specific cancel command, preventing accidental deletions.
  + Confirmation Prompt: Incorporated a confirmation step where detailed information about the identified record is displayed to the user. Users can review and confirm the deletion, safeguarding against accidental or unintended modifications.
  + Memory Management: Implemented safe deletion by updating the pointers in the linked list and freeing the memory allocated to the removed node and addressed scenarios where the list becomes empty after deletion by resetting both the head and tail pointers.
  + System Updates: Adjusted global variables, including node\_count and is\_changes\_made, to reflect the current state of the CMS after a successful deletion.