Student Record Management using Linked List

1. Problem Statement:

In any educational institution, managing student records is a fundamental task. A robust system is required to store, retrieve, and modify student information (e.g., ID, name, GPA, courses). A static array-based system is simple but suffers from a fixed size, leading to wasted memory or an inability to add more students once full. This project aims to solve this problem by creating a flexible and dynamic database system.

2. Abstract:

The "Student Record Management System" is a console-based application designed to manage student records efficiently. The core of this project is the implementation of a singly linked list as the primary data structure for storing and manipulating student data. This approach allows for dynamic memory allocation, efficient insertion and deletion of records without the limitations of a fixed-size array. The system provides essential functionalities such as adding a new student, searching for a student by ID, updating student information, deleting a student record, and displaying the entire list of students. This report details the system's design, the rationale behind using a linked list, the implementation of its core features, and an analysis of its performance and limitations.

3. Introduction:

This project is a console-based application to manage student records. It solves the problem of static, fixed-size arrays by using a singly linked list as its core data structure. This allows the database to grow and shrink dynamically. The system provides the six essential functionalities: Add, Search, Update, Delete, Count, and Display All student records.

4. Objective(s):

The primary objectives of this project are:

- To design and implement a Student data structure to hold individual student information.
- To utilize a singly linked list to manage a collection of Student records dynamically.
- To develop a menu-driven user interface for easy interaction.
- To implement core database functionalities (Create, Read, Update, Delete - CRUD).
- To analyze the advantages and disadvantages of using a linked list for this application.

5. Methodology:

<u>Tools/technologies used-</u>

Programming Language: - C language

Compiler:- GCC, Dev-C++

Text Editor:- VS code, DEV-C++

6. Implementation Details

This section describes the logic behind the core functionalities of the "Student Record Management" system, based on the provided C code. The system uses a global head pointer, initialized to NULL, to manage the singly linked list.

i) The main() Function

The main() function serves as the primary user interface. It runs an infinite while(1) loop that displays a menu of 7 options: Insert, Delete, Display, Search, Update, Count, and Exit. It uses a switch statement based on the user's integer choice to call the appropriate functions. The loop is only terminated when the user selects option 7, which calls the exit(0) function.

ii) insert() Function

This function adds a new student record to the **beginning** of the linked list.

- A new Node is dynamically allocated using malloc().
- The user is prompted to enter the student's name, Student ID, age, year, course, and phone number.
- **Duplicate ID Validation:** A critical feature of this function is the while(1) loop that validates the StudentID. It traverses the entire list, comparing the new StudentID with all existing ones. If a duplicate is found, it prints an error and forces the user to re-enter a valid ID.
- Once all data is collected and validated, the newNode->next pointer is set to the current head.
- The global head pointer is then updated to point to this newNode, making it the new first element of the list.

iii) delete(char* id) Function

This function removes a student record from the list using their StudentID.

- It uses two pointers: temp to iterate through the list and prev to keep track of the node *before* temp.
- The list is traversed until a node with a matching StudentID is found.
- **Edge Case (Deleting Head):** If the matching node is the head (meaning prev is still NULL), the global head pointer is simply moved to the next node (head = temp->next).
- **General Case:** If the match is found in the middle or at the end of the list, the prev node's next pointer is "jumped" over the current node (prev->next = temp->next).
- Finally, the temp node (containing the deleted record) is deallocated using free(temp).

iv) display() Function

This function prints all student records currently in the database.

- It checks if the list is empty (temp == NULL). If so, it prints "No records to display" and returns.
- If the list is not empty, it uses a temp pointer to traverse the list from head to the end.
- Inside the loop, it prints all fields of each student's record in a formatted way.

v) search(char* id) Function

This function finds and displays a single student's record.

- It uses a temp pointer to traverse the list starting from the head.
- It compares the StudentID of each node with the provided id.
- If a match is found, it prints the full details of that student and returns.
- If the end of the list is reached without a match, it prints a "Record... not found" message.

vi) update(char* id) Function

This function finds a student by id and allows the user to modify specific fields.

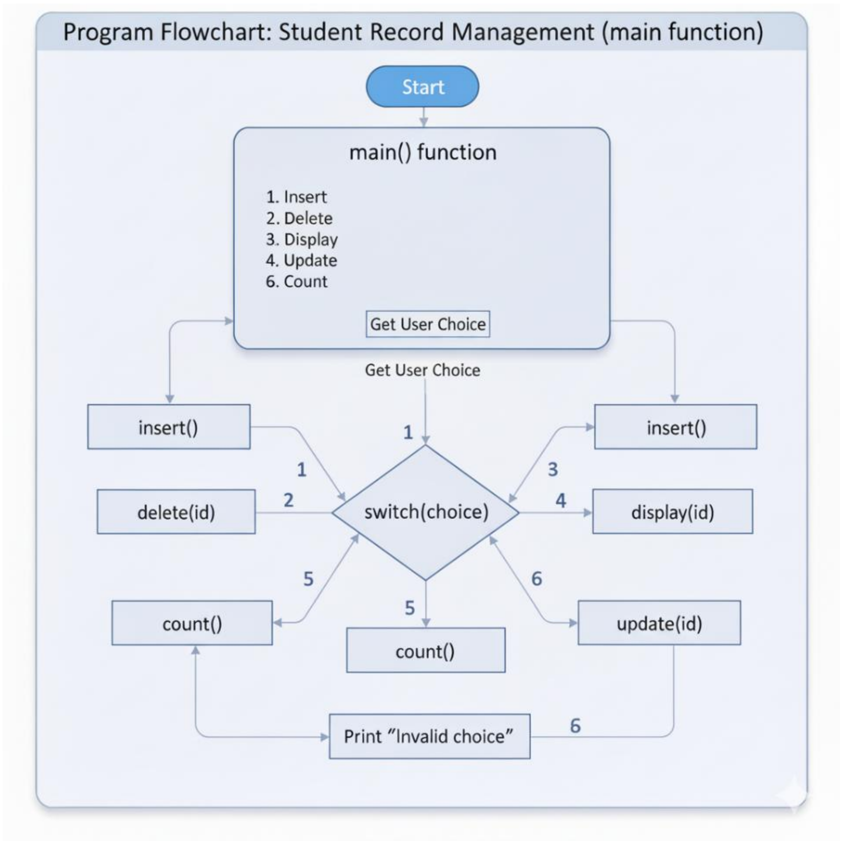
- It first traverses the list to find the node with the matching StudentID.
- If the student is not found, it prints a "not found" message.
- If the student *is* found, it displays a new sub-menu (options 1-5) asking which specific field (Name, Age, Year, Course, or Phone Number) the user wants to update.
- A switch statement handles the user's choice, prompting them to enter only the new data for that single field.

vii) count() Function

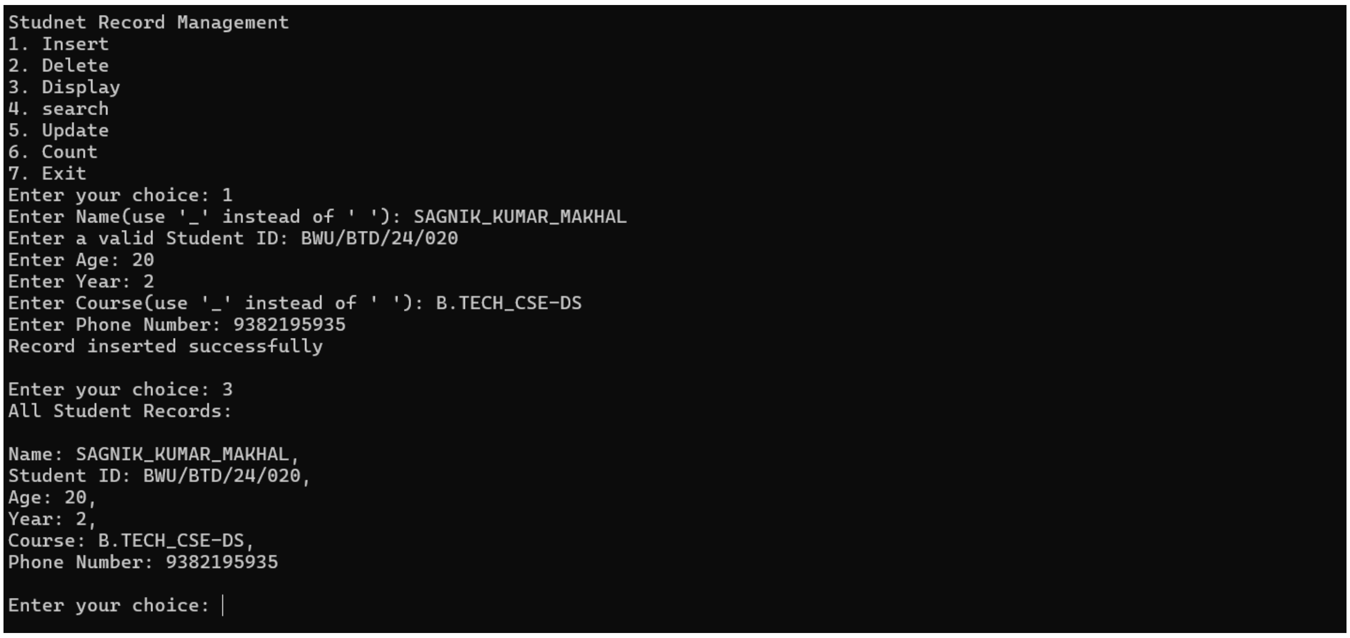
This is a utility function that counts the total number of records in the list.

- It initializes a counter cnt to 0.
- It uses a temp pointer to traverse the entire list.
- For every node it visits, it increments cnt.
- After the loop finishes, it returns the final cnt value.

7. Flowchart:-



8. Results and Discussion:



```
Studnet Record Management
1. Insert
2. Delete
3. Display
4. search
5. Update
6. Count
7. Exit
Enter your choice: 1
Enter Name(use '_' instead of ' '): SAGNIK_KUMAR_MAKHAL
Enter a valid Student ID: BWU/BTD/24/020
Enter Age: 20
Enter Year: 2
Enter Course(use '_' instead of ' '): B.TECH_CSE-DS
Enter Phone Number: 9382195935
Record inserted successfully

Enter your choice: 3
All Student Records:

Name: SAGNIK_KUMAR_MAKHAL,
Student ID: BWU/BTD/24/020,
Age: 20,
Year: 2,
Course: B.TECH_CSE-DS,
Phone Number: 9382195935

Enter your choice: |
```

9. Learning Outcomes:

- **Data Structures:** Implemented a singly linked list from scratch.
- **Memory Management:** Used malloc() and free() to dynamically manage memory.
- **Algorithms:** Implemented list traversal, insertion (at head), deletion (with edge-case handling), and linear search.
- **Core C:** Used structs, pointers, strcmp(), and switch statements.

- **Software Design:** Built a modular, menu-driven application.
- **Validation:** Added critical input validation to prevent duplicate StudentIDs.

10. Conclusion:

This project successfully demonstrated the creation of a functional Student Record Management system using a singly linked list in C. All core functionalities—including inserting (with duplicate ID checks), deleting, searching, updating, displaying, and counting records—were implemented and tested successfully.

The key takeaway is the fundamental trade-off between the **flexibility** of a linked list (dynamic memory, easy insertions) and its primary **limitation** (O(n) linear search time). While excellent for small-scale applications, this project clearly illustrates why more advanced data structures like hash tables are the preferred choice for large, performance-critical databases.

11. Reference:

i) **Ansi C, E. balaguruswamy, Mc, GrowHill Publications.**

ii) **Data Structure using C and C++, R. K. Shukla, Wiley Publications.**

iii) **Data Structure using C, Reema Thareja, Oxford Publications**

iv) **Let us C, Yashavant Kanetkar, BPB Publications.**