****

**Project-Report**

**Project Name : Student Record System**

**Course : BCA-Data Science**

**Subject : Data Structure Lab Subject code: 24CAP-152**

**Submitted by: Submitted to :**

Shlok Mahajan Mehak Bhatia

UId : 24BCD10032

Section : BCD-1 (B)

**1. Project Overview**

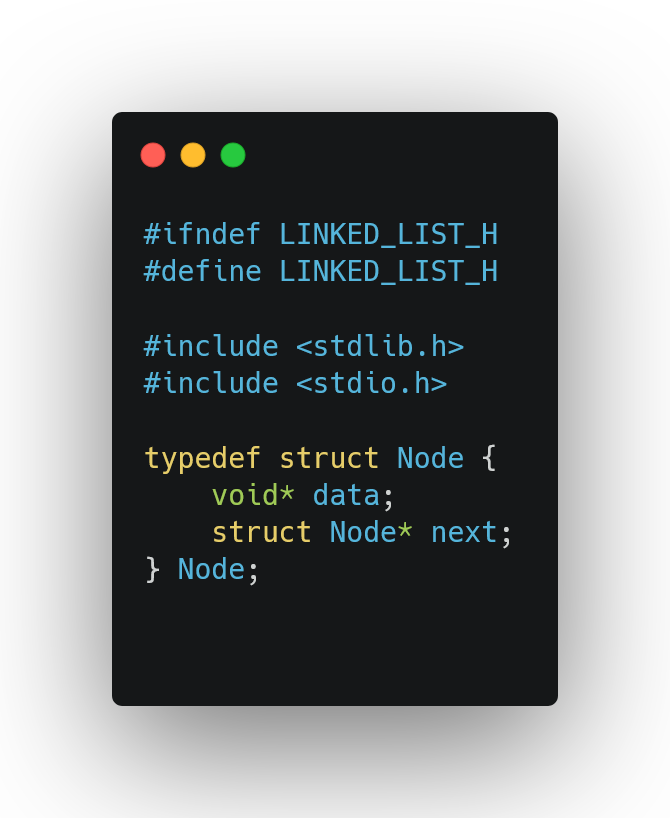
* **Name**: Student Record System
* **Description**: A console-based application to manage student records using linked lists for storage
* **Features**:
  + Add new student records
  + View all records
  + Search for specific students
  + Delete records
  + Automatic percentage calculation

**2. Project Structure**

****

**3. File Descriptions**

**1. linked\_list.h**





Student\_record.c

18

**/\*Some utility functions used are \*/**

void displayStudent(void\* data);

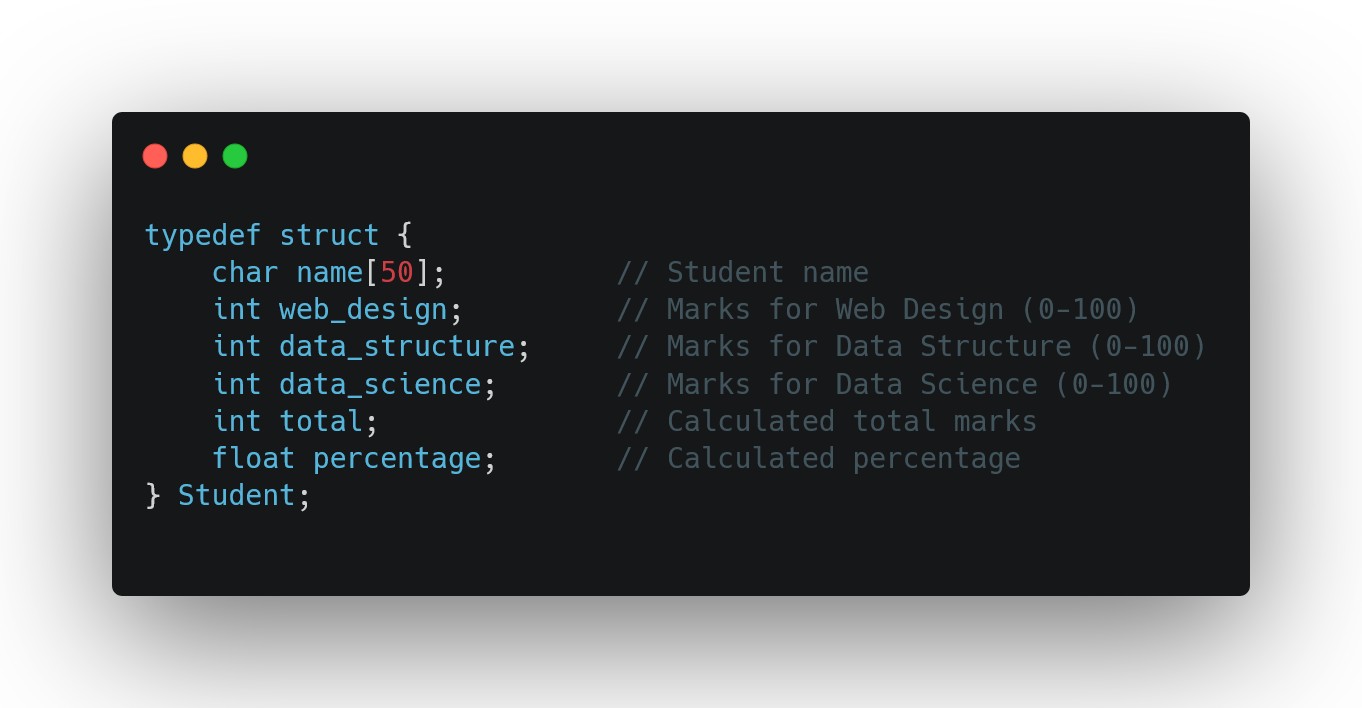
int compareStudents(void\* a, void\* b);

void freeStudent(void\* data);

void calculateStudentStats(Student\* student);

**4. Data Structures**

**Student Record:**

]

**5. Function Documentation**

**Core Functions:**

1. **createStudentRecord()**
   * Creates a new student record
   * Parameters: name, web design marks, data structure marks, data science marks
   * Returns: Pointer to new Student record
2. **insertNode()** (from linked\_list.h)
   * Inserts a new record into the linked list
   * Parameters: pointer to head pointer, data to insert
3. **searchNode()** (from linked\_list.h)
   * Searches for a record by name
   * Parameters: head pointer, search key, comparison function
   * Returns: Node pointer if found, NULL otherwise
4. **deleteNode()** (from linked\_list.h)
   * Deletes a record by name
   * Parameters: pointer to head pointer, data to delete, comparison function, free function

**Callback Functions:**

1. **displayStudent()**
   * Formats and prints a student record
   * Used by displayList()
2. **compareStudents()**
   * Compares two student records by name
   * Used by searchNode() and deleteNode()
3. **freeStudent()**
   * Frees memory allocated for a student record
   * Used by deleteNode() and freeList()

**7. Build and Run Instructions**

**8. Sample Usage**

1. Add a new record:

Enter student name: John Doe

Enter marks for Web Design: 85

Enter marks for Data Structure: 90

Enter marks for Data Science: 88

1. Search for a student:

Enter student name to search: John Doe

1. Delete a record:

Enter student name to delete: John Doe

Code and output

Linkedlist.h

#ifndef LINKED\_LIST\_H

#define LINKED\_LIST\_H

#include <stdlib.h>

#include <stdio.h>

#ifdef \_WIN32

    #define OS "Windows"

#elif \_\_linux\_\_

    #define OS "Linux"

#elif \_\_APPLE\_\_

    #define OS "macOS"

#else

    #define OS "Unknown"

#endif

#ifdef \_\_cplusplus

extern "C" {

#endif

#define LOG(msg) printf("[DEBUG] %s (%s:%d)\n", msg, \_\_FILE\_\_, \_\_LINE\_\_)

// Structure for a node (generic implementation)

typedef struct Node {

    void\* data;         // Generic pointer to hold any data type

    struct Node\* next;  // Pointer to next node

} Node;

// Function declarations

Node\* createNode(void\* data);

void insertNode(Node\*\* head, void\* data);

Node\* searchNode(Node\* head, void\* data, int (\*compare)(void\*, void\*));

void deleteNode(Node\*\* head, void\* data, int (\*compare)(void\*, void\*), void (\*freeData)(void\*));

void freeList(Node\*\* head, void (\*freeData)(void\*));

void displayList(Node\* head, void (\*display)(void\*));

// Function definitions (inline to avoid multiple definition errors)

 inline Node\* createNode(void\* data) {

    Node\* newNode = (Node\*)malloc(sizeof(Node));

    if (newNode == NULL) {

        fprintf(stderr, "Memory allocation failed for new node\n");

        exit(EXIT\_FAILURE);

    }

    newNode->data = data;

    newNode->next = NULL;

    return newNode;

}

 inline void insertNode(Node\*\* head, void\* data) {

    Node\* newNode = createNode(data);

    if (\*head == NULL) {

        \*head = newNode;

    } else {

        Node\* current = \*head;

        while (current->next != NULL) {

            current = current->next;

        }

        current->next = newNode;

    }

}

 inline Node\* searchNode(Node\* head, void\* data, int (\*compare)(void\*, void\*)) {

    Node\* current = head;

    while (current != NULL) {

        if (compare(current->data, data) == 0) {

            return current;

        }

        current = current->next;

    }

    return NULL;

}

 inline void deleteNode(Node\*\* head, void\* data, int (\*compare)(void\*, void\*), void (\*freeData)(void\*)) {

    if (\*head == NULL) return;

    Node\* current = \*head;

    Node\* prev = NULL;

    // If head node itself holds the data to be deleted

    if (compare(current->data, data) == 0) {

        \*head = current->next;

        freeData(current->data);

        free(current);

        return;

    }

    // Search for the node to be deleted

    while (current != NULL && compare(current->data, data) != 0) {

        prev = current;

        current = current->next;

    }

    if (current == NULL) return; // Data not found

    // Unlink the node from linked list

    prev->next = current->next;

    freeData(current->data);

    free(current);

}

 inline void freeList(Node\*\* head, void (\*freeData)(void\*)) {

    Node\* current = \*head;

    Node\* next;

    while (current != NULL) {

        next = current->next;

        freeData(current->data);

        free(current);

        current = next;

    }

    \*head = NULL;

}

 inline void displayList(Node\* head, void (\*display)(void\*)) {

    Node\* current = head;

    while (current != NULL) {

        display(current->data);

        current = current->next;

    }

}

#ifdef \_\_cplusplus

}

#endif

#endif //closing of universal header file linked list closing

Student\_record.c

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include "linked\_list.h"

// Structure for student record

typedef struct {

    char name[50];

    int web\_design;

    int data\_structure;

    int data\_science;

    int total;

    float percentage;

} Student;

// Function prototypes

Student\* createStudentRecord(char name[], int web, int ds, int dsc);

void displayStudent(void\* data);

int compareStudents(void\* a, void\* b);

void freeStudent(void\* data);

void calculateStudentStats(Student\* student);

void displayMenu();

// Main program

int main() {

    Node\* studentList = NULL;

    int choice;

    char name[50];

    int web, ds, dsc;

    printf("Student Record System using Linked List\n");

    while (1) {

        displayMenu();

        printf("Enter your choice: ");

        scanf("%d", &choice);

        switch (choice) {

            case 1: {

                printf("Enter student name: ");

                scanf("%s", name);

                printf("Enter marks for Web Design: ");

                scanf("%d", &web);

                printf("Enter marks for Data Structure: ");

                scanf("%d", &ds);

                printf("Enter marks for Data Science: ");

                scanf("%d", &dsc);

                Student\* newStudent = createStudentRecord(name, web, ds, dsc);

                insertNode(&studentList, newStudent);

                printf("Record added successfully!\n");

                break;

            }

            case 2: {

                printf("\nStudent Records:\n");

                printf("-----------------------------------------------------------------\n");

                printf("%-20s %-15s %-15s %-15s %-10s %-10s\n",

                       "Name", "Web Design", "Data Structure", "Data Science", "Total", "Percentage"); // we are alloting the space (left aligned)for all the variables

                        //  Type        Left-Aligned (%-)   Right-Aligned (default)

                        // String     "%-10s" → "Hello "    "%10s" → " Hello"

                        // Integer      "%-5d" → "42 "      "%5d" → " 42"

                        // Float      "%-8.2f" → "3.14 "    "%8.2f" → " 3.14"

                printf("-----------------------------------------------------------------\n");

                displayList(studentList, displayStudent);

                printf("-----------------------------------------------------------------\n");

                break;

            }

            case 3: {

                printf("Enter student name to search: ");

                scanf("%s", name);

                Student searchKey;

                strcpy(searchKey.name, name);

                Node\* found = searchNode(studentList, &searchKey, compareStudents);    // a node is the part of linkedlist header file  with member next and data

                if (found != NULL) {

                    printf("\nStudent Details:\n");

                    printf("----------------------------\n");

                    displayStudent(found->data);

                    printf("----------------------------\n");

                } else {

                    printf("Student record not found.\n");

                }

                break;

            }

            case 4: {

                printf("Enter student name to delete: ");

                scanf("%s", name);

                Student deleteKey;

                strcpy(deleteKey.name, name);

                deleteNode(&studentList, &deleteKey, compareStudents, freeStudent);

                printf("Record deleted successfully.\n");

                break;

            }

            case 5: {

                freeList(&studentList, freeStudent);

                printf("Exiting program. Goodbye!\n");

                exit(0);

            }

            default:

                printf("Invalid choice. Please try again.\n");

        }

    }

    return 0;

}

// Create a new student record

Student\* createStudentRecord(char name[], int web, int ds, int dsc) {

    Student\* newStudent = (Student\*)malloc(sizeof(Student));

    if (newStudent == NULL) {

        fprintf(stderr, "Memory allocation failed for new student\n");

        exit(EXIT\_FAILURE);

    }

    strcpy(newStudent->name, name);

    newStudent->web\_design = web;

    newStudent->data\_structure = ds;

    newStudent->data\_science = dsc;

    calculateStudentStats(newStudent);

    return newStudent;

}

// Callback function to display a student record

void displayStudent(void\* data) {

    if (data == NULL) return;

    Student\* student = (Student\*)data;

    printf("%-20s %-15d %-15d %-15d %-10d %-10.2f\n",

           student->name, student->web\_design, student->data\_structure,

           student->data\_science, student->total, student->percentage);

}

// Callback function to compare two student records by name

int compareStudents(void\* a, void\* b) {

    if (a == NULL || b == NULL) return -1;

    Student\* studentA = (Student\*)a;

    Student\* studentB = (Student\*)b;

    return strcmp(studentA->name, studentB->name);

//     Return Value Meaning

// < 0  str1 is less than str2 (lexicographically)

// 0    str1 is equal to str2

// > 0  str1 is greater than str2 (lexicographically)

}

// Callback function to free a student record

void freeStudent(void\* data) {

    if (data != NULL) {

        free((Student\*)data);

    }

}

// Calculate total and percentage for a student

void calculateStudentStats(Student\* student) {

    student->total = student->web\_design + student->data\_structure + student->data\_science;

    student->percentage = (float)student->total / 3; // the quickest way to find percentage is to find the average valid for all

}

// Display menu options

void displayMenu() {

    printf("\nMenu:\n");

    printf("1. Add new student record\n");

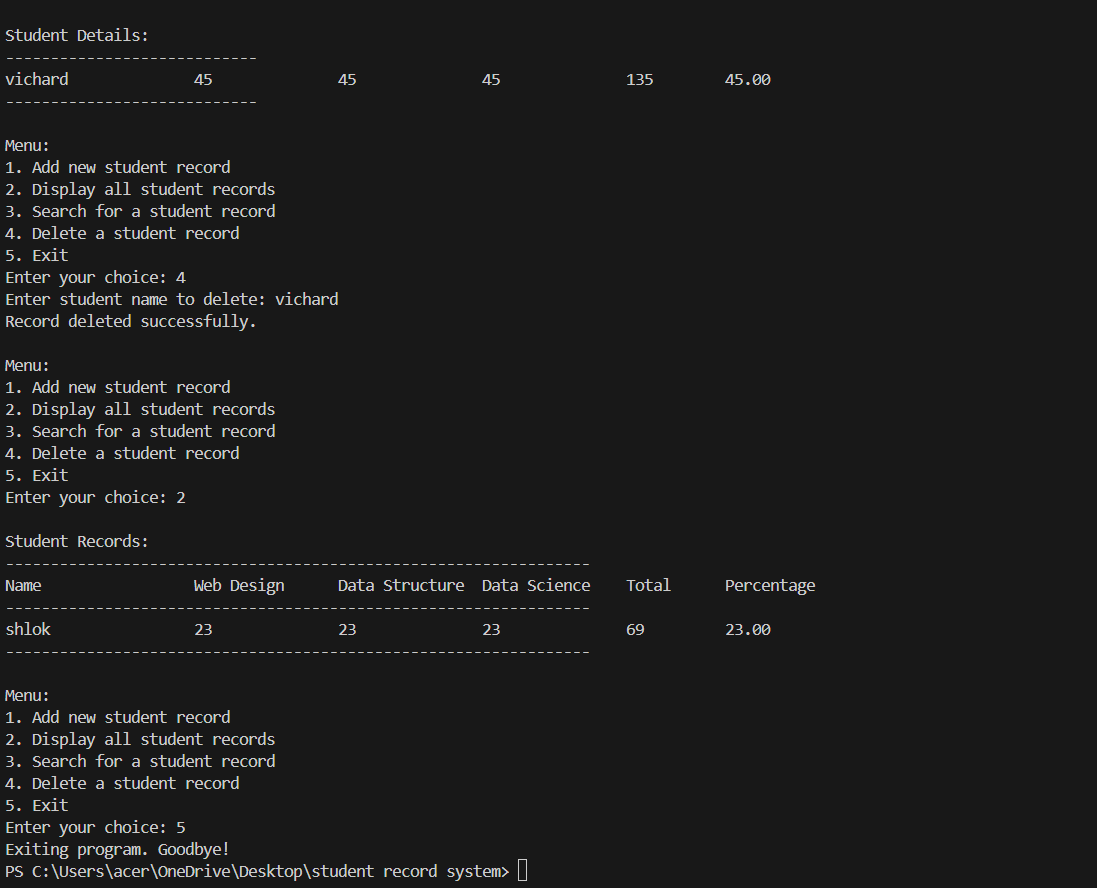
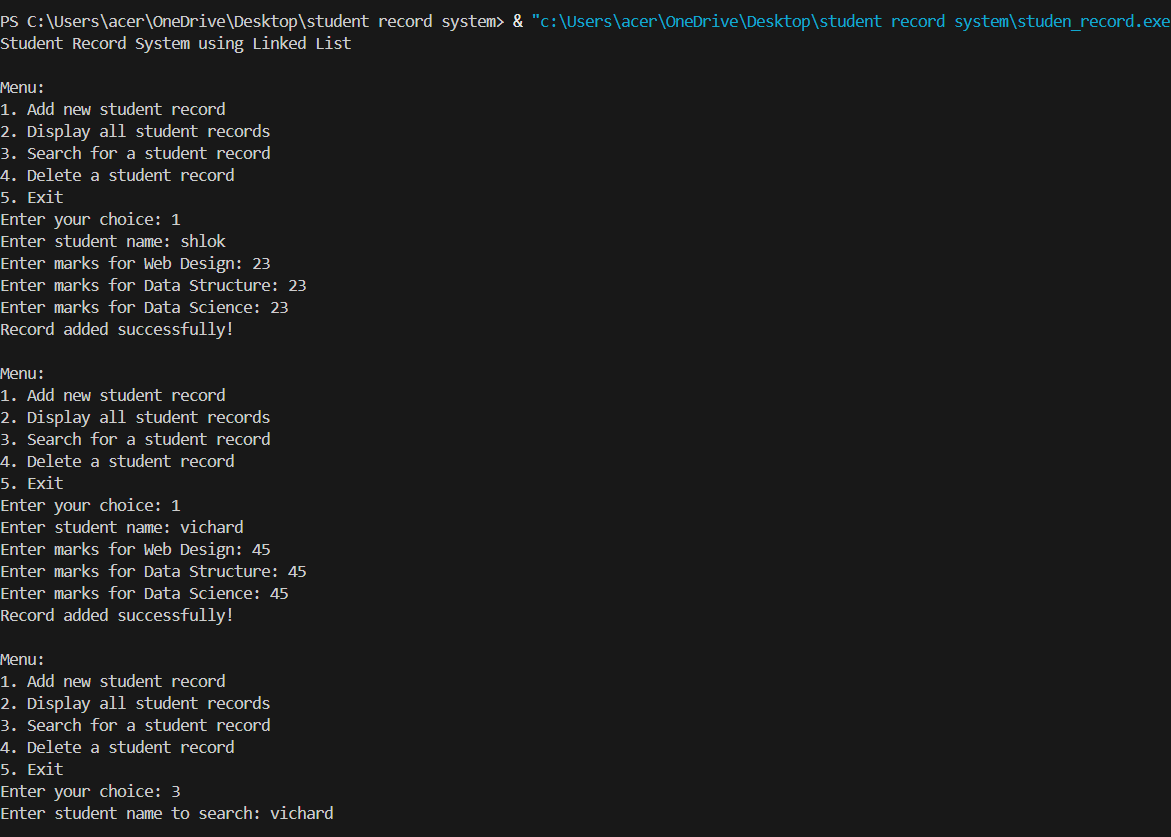
    printf("2. Display all student records\n");

    printf("3. Search for a student record\n");

    printf("4. Delete a student record\n");

    printf("5. Exit\n");

}

Output

**9. Limitations**

* Console-based interface only
* No persistent storage (data lost when program exits)
* Basic error handling

**10. Future Enhancements**

* Add file I/O for data persistence
* Implement sorting functionality
* Add more student fields (ID, contact info, etc.)
* Graphical user interface

# ****Case Study: Student Record Management System Using Linked List in C****

## ****1. Introduction****

This case study explores a **Student Record Management System** implemented in **C** using a **generic linked list** data structure. The system allows educational institutions to manage student records efficiently, including storing student details, marks in different subjects, and calculating total marks and percentages.

### ****Key Features****

* **Dynamic Student Record Management** (Add, Search, Delete, Display)
* **Modular Design** (Separate Linked List Implementation)
* **Generic Data Handling** (Supports any data type via void\*)
* **Interactive Menu-Driven Interface**

## ****System Architecture****

### ****Components****

1. **Linked List Module (linked\_list.h)**
   * Provides reusable linked list operations (insert, search, delete, display).
   * Uses **generic void\* pointers** for flexibility.
   * Supports **callback functions** for custom data handling.
2. **Student Record Module (student\_record.c)**
   * Defines the Student structure (name, marks, total, percentage).
   * Implements functions for:
     + Creating/deleting student records.
     + Calculating total marks and percentage.
     + Displaying records in a formatted table.
3. **Main Program**
   * Interactive menu for user operations.
   * Integrates linked list functions with student-specific logic.

## ****Workflow Example****

1. **User Adds a Student**:
   * Input: Name = "Alice", Marks = (90, 85, 95).
   * System calculates total (270) and percentage (90.00).
   * Record stored in the linked list.
2. **User Searches for "Alice"**:
   * System traverses the list and displays Alice’s details.
3. **User Deletes "Alice"**:
   * System removes the node and deallocates memory.
4. **User Exits**:
   * freeList() cleans up all remaining nodes.

## ****Conclusion****

This system demonstrates how **generic data structures** (like linked lists) can power real-world applications (like student record management). By decoupling the linked list logic from student-specific operations, the code becomes **reusable, maintainable, and extensible**.

**Potential Extensions**:

* Add a database backend (SQLite).
* Implement sorting by marks/names.
* Extend to a multi-user system.