

STUDENT RECORD KEEPING SYSTEM

**By: Ibrahim Dayah Abdullahi
1002059744**

Table of Contents

- 1. Introduction**
- 2. System Design**
- 3. Results & Discussion**
- 4. Conclusion**

Chapter One: Introduction

This app was created as part of a UCSI university assignment for the BER2023 Object Oriented Programming course. The purpose of this application is to serve as a Student Record Keeping System, allowing teachers to keep track of their students' names, studentIDs, courseIDs, and most importantly their total marks.

1.1 Background of Study

The purpose of this software is to aid teachers in keeping track of their students' grades and records. The Student Record Keeping System keeps track of important student information by tracking the totalScore for each course/subject, which can be used to interpret an organization's success in the future.

1.2 Problem Statement

Some educational institutions continue to capture and store student record details locally, with hard copies of files for each student preserved on office shelves. This appears to be tedious and time consuming in the event that the registrar needs to locate a specific student document. Data redundancy, updating and maintaining the current manual system is onerous, inconsistent data, insecurity, imposing limitations on multiple data files is tough, and backup is difficult. As a result of these shortcomings, the Student Record Keeping System was created to address the issues listed above.

1.3 Project Objective

The main objective is to develop a robust Student Record Keeping System to be used by teachers and other educators in educational institutions.

Specific Objectives:

- i. To implement the system
- ii. To test and validate the system

1.4 Scope

This project was created as part of a UCSI assignment, but it can be used by any teacher or educator who need a reliable student record keeping system.

Data can be captured, validated, sorted, classified, modified, summarized, stored, and retrieved using this system.

1.5 Purpose of this Project

The suggested approach is designed to make life easier for educators. The project's main goal is to create a student record-keeping system that allows for easy access to student records. The Student Record Keeping System improves student information and process management.

Chapter Two: System Design

2.0 Introduction

Our system was designed using a UML Class diagram throughout the project development process. This is to ensure that we can maintain control over the project's scope throughout the development phase. We'll also go through how to set up the database, header files for each class, data structures, and the operation of each system service in this section. In the third chapter, Chapter 3 Results and Discussion, the definition of method in codes will be examined.

2.1 UML Class Diagram

As previously stated, the UML diagram is utilized to plan our project throughout its development. The Hospital Management System's Class Diagram is shown in Figure 1.1

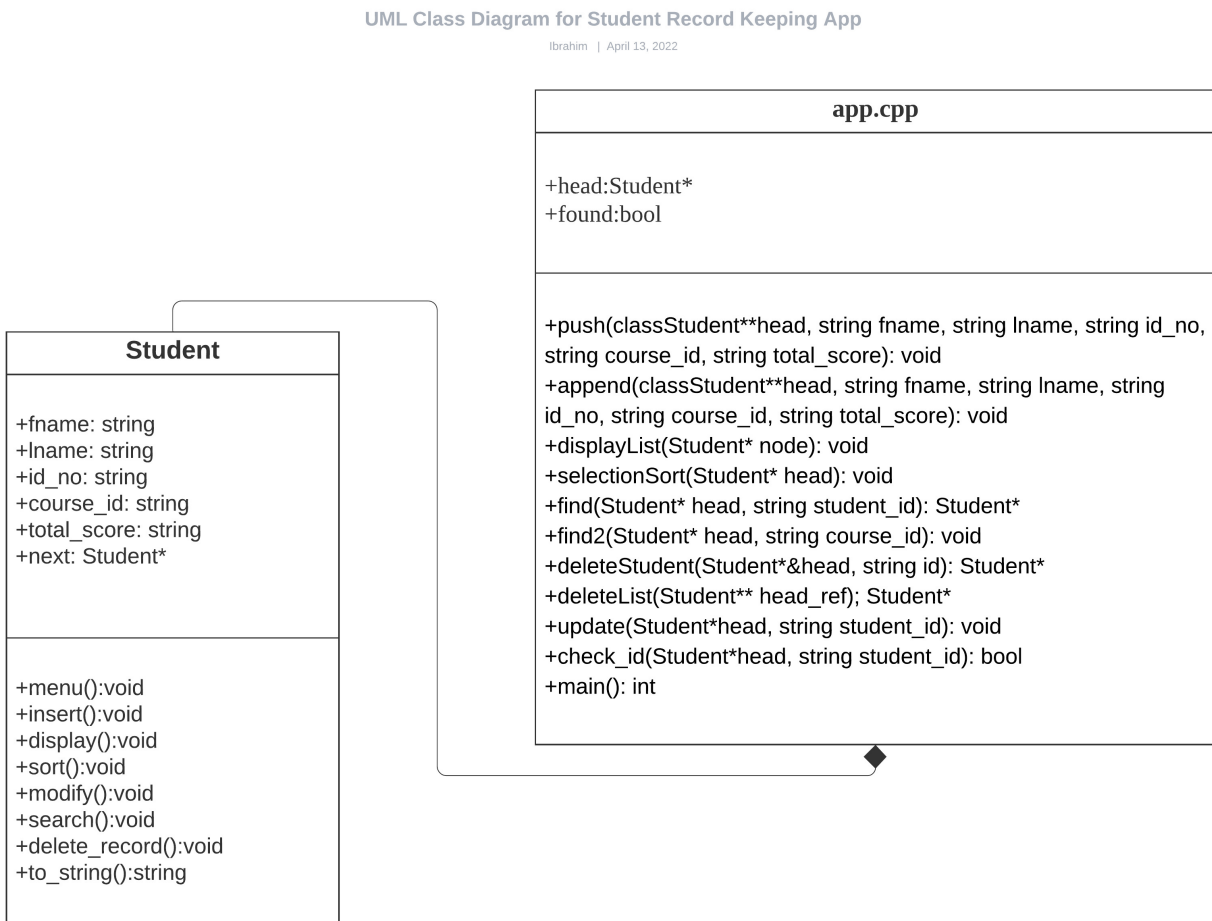


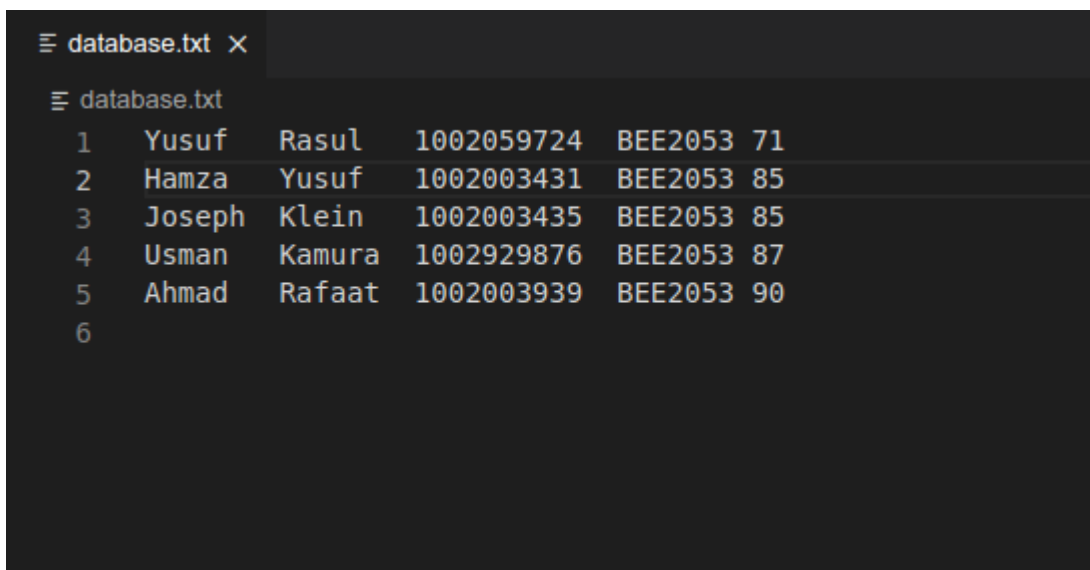
Figure 2.1: UML class diagram of our Student Record Keeping System

I used custom built linked list in our student record keeping system to manipulate data using various algorithms like searching, adding, sorting, and removing. As previously said, I used text files (or txt files) as the database, hence I used several of the 'fstream' built-in functions for file processing.

Database

Relational database management system (RDBMS) is an excellent tool for organizing large amount of data and defining the relationship between the datasets in a consistent and understandable way. A RDBMS provides a structure which is flexible enough to accommodate almost any kind of data. Relationships between the tables were defined by creating special columns (keys), which contain the same set of values in each table. The tables can be joined in different combinations to extract the needed data. A RDBMS also offered flexibility that enabled redesign and regeneration of reports from the database without the need to re-enter the data.

The original plan was to use a relational database management system (RDBMS) such as MySQL, but due to time restrictions and a desire for simplicity, a basic txt file was used to mimic an RDBMS, store the data, and operate as a local simple database. The figure below shows my database file with some records present inside it.



The image shows a screenshot of a text editor window titled 'database.txt'. Inside the editor, the same title is shown above a list of six lines of data. Each line represents a student record with six fields: an index number, a first name, a last name, a student ID, a department code, and a grade. The data is as follows:

	1	2	3	4	5	6
1	Yusuf	Rasul	1002059724	BEE2053	71	
2	Hamza	Yusuf	1002003431	BEE2053	85	
3	Joseph	Klein	1002003435	BEE2053	85	
4	Usman	Kamura	1002929876	BEE2053	87	
5	Ahmad	Rafaat	1002003939	BEE2053	90	
6						

Figure 2.2: my database file with some records inside it

Student Class

Based on the UML Class Diagram in figure 1.1, we can see that Student class is used to describe a patient in real life. Its instance variables are used to describe the attributes of a Student Object. From the Student Class, we know that the Student Object contains attributes of first name, last name , student id, course id and total score. The Student Class consists of methods or functions to create their academic record in the database, display information such as course id, total score and personal information from the database, update information such as total score and course id and sort the queue based on students total score in ascending order. These behaviors can be achieved by constructing the functions as per UML Class diagram in figure 2.1. The function definition will be discussed in the latter chapter which is Chapter 3: Results and Discussion. The Student class also defines the nodes of the linked list so each Student object is a node in the linked list. The figure below shows how I implemented the Student class.

```
//Student node class that defines all the properties each student record will have
//as well as defining the methods that will run this program. It also has a property
//that points to the next student in the LinkedList
class Student
{
public:
    string fname, lname, id_no, course_id, total_score;

    Student* next;//pointer to next student node in the LinkedList

    //method prototypes
    void menu();
    void insert();
    void display();
    void sort();
    void modify();
    void search();
    void delete_record();

    //this method returns the details of the Student in a neat string format
    //and makes it easier to write each record to the database file
    string to_string() {
        return fname + "\t" + lname + "\t" + id_no + "\t" + course_id + "\t" + total_score + "\n";
    }
};
```

Figure 2.3: Student class implementation

app.cpp global functions

The functions that manipulate the linked list were declared in the global scope of the app file as can be seen from the UML class diagram in figure 1.1. The figure below shows the function prototypes of all the functions that operate on the linked list. Their full definitions will be shown in the next chapter.

```
//LinkedList function prototypes
void push(class Student** head, string fname, string lname, string id_no, string course_id, string total_score);
void append(class Student** head, string fname, string lname, string id_no, string course_id, string total_score );
void displayList(class Student *node);
void selectionSort(Student* head);
Student* find(Student* head, string student_id);
Student* find2(Student* head, string course_id);
void deleteStudent(Student*& head, string id);
void deleteList(Student** head_ref);
Student* update(Student* head, string student_id);
bool check_id(Student* head, string student_id);
```

Figure 2.4: prototypes of the functions that manipulate the linked list

2.2 System Functions

Our Student Record Keeping System provides seven different options to the user every time the menu is displayed as can be seen in the figure below:

```
*****
* STUDENT RECORD KEEPING SYSTEM *
*****
1. Enter a New Record
2. Display Record
3. Sort using total score
4. Modify a record
5. Search for a record
6. Delete a record
7. Exit

*****
* Choose Options:[1/2/3/4/5/6/7] *
*****

Enter Your Choice: █
```

Figure 2.5: menu of our student record keeping app

1. The user can inset a new record into the database
2. The user can display the records already in the database
3. The user can sort the records in the database using total score
4. The user can modify the values of a record except student id
5. The user can search for a record from the database using student id
6. The user can search for a record from the database using student or course id
7. The user can exit the application

Insert a new record

The user can insert a new record into the database by entering 1. When the user clicks on that the key the user will be prompted to enter the first name last name, student id, course id and total score of the new student. If the student id the user enters already exists in the database the user will be alerted to the existence of that a record with that id and will be prompted to try again. This is done to prevent duplicate records in our database. The full implementation of this function will be shown in the next chapter.

Display existing records

The user can display the existing records in database by entering 2. When the user clicks on that the function will check if the database file exists and if it's not empty. If the database file exists and it's not empty then the contents of it will be displayed in a neat form but if it does not exist or is empty a message will be displayed to the user informing him of that. The full implementation of this function will be shown in the next chapter.

Sort existing records

The user can sort the existing records in database in ascending order using total score by entering 3. When the user clicks on that the function will check if the database file exists and if it's not empty. If the database file exists and it's not empty then the records will be sorted in ascending order using total score but if it does not exist or is empty a message will be displayed to the user informing him of that. The full implementation of this function will be shown in the next chapter.

Modify existing record

The user can modify an existing record in database by first searching for it using student id by entering 4. When the user clicks on that the function will check if the database file exists and if it's not empty. If the database file exists and it's not empty then the record will be searched for using student id and if it is found the user will be able to modify all of it's values except for student id, but if the record does not exist a message will be displayed to the user informing him of that. The full implementation of this function will be shown in the next chapter.

Search existing records

The user can search for an existing record in the database by entering 5. The user will then be prompted to choose between searching using student id and course id. If the database file exists and it's not empty, then depending on the choice of the user the database records will be searched using either student id or course id. If the record is then found it will be displayed but if it's not found a message will be displayed informing the user that the record was not found. The full implementation of this function will be shown in the next chapter.

Delete existing record

The user can delete an existing record in database by first searching for it using student id by entering 6. When the user clicks on that key the function will check if the database file exists and if it's not empty. If the database file exists and it's not empty then the record will be searched for using student id and if it is found then the record will be deleted and a message will be shown informing the user that the record has been successfully deleted, but if the record does not exist a message will be displayed to the user informing him that the record does not exist. The full implementation of this function will be shown in the next chapter.

Exit the application

The user can exit the application by entering 7.

Chapter Three: Results & Discussion

3.0 Introduction

As mentioned in the Chapter Introduction, our Student Record Keeping System is a console application. So, in this section, we will discuss about how the main menu is generated, how the records in the database are easily manipulated by copying them to a linked list and then applying various algorithms to the linked list.

3.1 Main Menu

```
*****
* STUDENT RECORD KEEPING SYSTEM *
*****
1. Enter a New Record
2. Display Record
3. Sort using total score
4. Modify a record
5. Search for a record
6. Delete a record
7. Exit

*****
* Choose Options:[1/2/3/4/5/6/7] *
*****

Enter Your Choice: █
```

Figure 3.1: main menu of our app

In the figure above (3.1), there are six services provided by the system that are put inside a menu together with 'Exit'. These services have their own functions to achieve the operation that has been discussed in Chapter 2. Figure 3.2 below shows the menu appears again along with a message saying invalid choice when the user tries to enter number or selection that is not included in the menu which is '8'.

```

*****
* STUDENT RECORD KEEPING SYSTEM *
*****
1. Enter a New Record
2. Display Record
3. Sort using total score
4. Modify a record
5. Search for a record
6. Delete a record
7. Exit

*****
* Choose Options:[1/2/3/4/5/6/7] *
*****

Enter Your Choice: 8

Invalid Choice. Please Try Again.

*****
* STUDENT RECORD KEEPING SYSTEM *
*****
1. Enter a New Record
2. Display Record
3. Sort using total score
4. Modify a record
5. Search for a record
6. Delete a record
7. Exit

*****
* Choose Options:[1/2/3/4/5/6/7] *
*****

Enter Your Choice: █

```

Figure 3.2: shows what happens when the user enters wrong input

This can be achieved by using a goto statement nested in a switch statement which acts as a loop and as long as the user does not select one of the seven options, the program will keep running. If the user entered the correct input, then it will bring user to the specific service or the operation that was chosen by the user then after the service is done it will display the main menu again as can be seen from the figure below.

```
*****  
* STUDENT RECORD KEEPING SYSTEM *  
*****
```

The figure below shows the code used to achieve this program loop. It's essentially just a goto statement nested inside a switch statement.

```

{
    //this is where the application starts
start:
    int first_choice;
    char second_choice;
    cout << "\n\t\t\t*****" << "\n";
    cout << "\t\t\t* STUDENT RECORD KEEPING SYSTEM *" << "\n";
    cout << "\t\t\t*****" << "\n";
    cout << "\t\t\t 1. Enter a New Record" << "\n";
    cout << "\t\t\t 2. Display Record" << "\n";
    cout << "\t\t\t 3. Sort using total score" << "\n";
    cout << "\t\t\t 4. Modify a record" << "\n";
    cout << "\t\t\t 5. Search for a record" << "\n";
    cout << "\t\t\t 6. Delete a record" << "\n";
    cout << "\t\t\t 7. Exit\n"
    << "\n";

    cout << "\t\t\t*****" << "\n";
    cout << "\t\t\t* Choose Options:[1/2/3/4/5/6/7] *" << "\n";
    cout << "\t\t\t*****" << "\n";
    cout << " Enter Your Choice: ";
    cin >> first_choice;
    switch (first_choice)
    {
    case 1:
        do
        {
            insert();
            cout << "\n\n\t\t\t Add Another Student Record (Y, N) : ";
            cin >> second_choice;
        } while (second_choice == 'y' || second_choice == 'Y');
        break;
    case 2:
        display();
        break;
    case 3:
        sort();
        break;
    case 4:
        modify();
        break;
    case 5:
        search();
        break;
    case 6:
        delete_record();
        break;
    case 7:
        cout << "\n\t\t\t The Program has been terminated succesfully \n";
        exit(0);
    default:
        cout << "\n\n\t\t\t Invalid Choice. Please Try Again. \n";
    }

    //use goto statement as a while loop to keep the program running until
    //the user decides to quit
    goto start;
}

```

Figure 3.6: code used to achieve program loop

3.2 Linked List and Algorithms

Here we further discuss the implementation of the algorithms that we used to manipulate the data in the linked list. The function algorithms are all declared and defined in our application file (app.cpp). As we mentioned earlier in Chapter 2, the Student class is the node of our linked list as can be seen in the figure below.

```
//Student node class that defines all the properties each student record will have
//as well as defining the methods that will run this program. It also has a property
//that points to the next student in the LinkedList
class Student
{
public:
    string fname, lname, id_no, course_id, total_score;

    Student* next; //pointer to next student node in the LinkedList

    //method prototypes
    void menu();
    void insert();
    void display();
    void sort();
    void modify();
    void search();
    void delete_record();

    //this method returns the details of the Student in a neat string format
    //and makes it easier to write each record to the database file
    string to_string() {
        return fname + "\t" + lname + "\t" + id_no + "\t" + course_id + "\t" + total_score + "\n";
    }
};
```

Figure 3.7: shows Student class which is the node of our linked list

The Student node class in the app.cpp file acts as a node that contains data and pointer. The data is declared as string data types and consists of all the fields of each record as can be seen in the figure above. The pointer is named as 'next' as shown in figure 3.7 and it is used to point to the next element or in another words, the pointer will store the value of next node's address. Figure 3.8 shows the diagram of a linked list.



Add a Student to the linked list

The function in the figure below adds a Student object to the tail of the linked list.

```
/* insert new node at the end of the linked list */
void append(class Student** head, string fname, string lname, string id_no, string course_id, string total_score )
{
    /* 1. create and allocate node */
    class Student* newStudent = new Student;

    class Student *last = *head; /* used in step 5*/

    /* 2. assign data to the node */
    newStudent->fname = fname;
    newStudent->lname = lname;
    newStudent->id_no = id_no;
    newStudent->course_id = course_id;
    newStudent->total_score = total_score;

    /* 3. set next pointer of new node to null as its the last node*/
    newStudent->next = NULL;

    /* 4. if list is empty, new node becomes first node */
    if (*head == NULL)
    {
        *head = newStudent;
        return;
    }

    /* 5. Else traverse till the last node */
    while (last->next != NULL)
        last = last->next;

    /* 6. Change the next of last node */
    last->next = newStudent;
    return;
}
```

Figure 3.8: shows function to add new Student node to the tail of the linked list

Sort the linked list

The function in the figure below uses the selection sort algorithm to sort the linked list in ascending order using total score.

```
void selectionSort(Student* head)
{
    Student* temp = head;

    // Traverse the List
    while (temp) {
        Student* min = temp;
        Student* r = temp->next;

        // Traverse the unsorted sublist
        while (r) {
            if (stoi(min->total_score) > stoi(r->total_score))
                min = r;
            r = r->next;
        }

        // Swap all the data
        string a = temp->fname;
        string b = temp->lname;
        string c = temp->id_no;
        string d = temp->course_id;
        string e = temp->total_score;

        temp->fname = min->fname;
        temp->lname = min->lname;
        temp->id_no = min->id_no;
        temp->course_id = min->course_id;
        temp->total_score = min->total_score;

        min->fname = a;
        min->lname = b;
        min->id_no = c;
        min->course_id = d;
        min->total_score = e;

        temp = temp->next;
    }
}
```

Figure 3.9: shows function that implements selection sort algorithm on the linked list

Search the linked list

The functions in the figure below search the linked list using student id and course

id respectively.

```
//this function implements the serial searching algorithm on our LinkedList
//and uses Student ID as the search key
Student* find(Student* head, string student_id)
{
    // Base case
    if (head == NULL) {
        cout << "Record not found\n";
        return head;
    }

    // If the Student ID is present in current node, return current node
    if (head->id_no == student_id) {
        cout << head->to_string();
        return head;
    }

    // Use recursion to go thru the remaining records in the list
    return find(head->next, student_id);
}

//this function implements the serial searching algorithm on our LinkedList
//and uses Course ID as the search key
bool found = false;
Student* find2(Student* head, string course_id)
{
    // Base case
    if (head == NULL) {
        if(!found)
            cout << "Record not found\n";
        return head;
    }

    // If the Course ID is present in current node, keep looking for more students with
    // the same Course ID in the list until you iterate thru the entire list
    if (head->course_id == course_id) {
        found = true;
        cout << head->to_string();
    }

    // Use recursion to go thru the remaining records in the list
    return find2(head->next, course_id);
}
```

Figure 3.10: shows functions that search the linked list

Delete a Student from the linked list

The function in the figure below searches a Student in the linked list using student

id and removes it from the linked list if it finds it.

```
//this function deletes the Student node containing the same student ID as the
//student id passed as an argument and alters the head of the linked list using recursion
void deleteStudent(Student*& head, string id)
{
    // Check if list is empty or we
    // reach at the end of the
    // list.
    if (head == NULL) {
        cout << "Record not present in the list\n";
        return;
    }
    // If current node is the node to be deleted
    if (head->id_no == id) {
        Student* t = head;
        head = head->next; // If it's start of the Student node head
                           // Student node points to second node
        delete (t); // Else changes previous Student node's next to
                   // current node's next
        cout << "Record has been successfully deleted\n";
        return;
    }
    deleteStudent(head->next, id);
}
```

Figure 3.11: function that can delete a Student node from the linked list

Delete the entire linked list

The function in the figure below when called can delete the entire linked list when it's not needed anymore to free up memory.

```
//this function will delete the entire linked list when it's not needed to save memory
void deleteList(Student** head_ref)
{
    /* deref head_ref to get the real head */
    Student* current = *head_ref;
    Student* next = NULL;

    while (current != NULL)
    {
        next = current->next;
        free(current);
        current = next;
    }

    /* deref head_ref to affect the real head back
    in the caller. */
    *head_ref = NULL;
}
```

Figure 3.12: shows function that can delete the entire linked list

Modify a Student in the linked list

The function in the figure below searches the linked list using student id passed in

as an argument and returns the Student with that student id. It is used in the Student::modify method to return a record that is to be modified.

```
//this function implements the serial searching algorithm on our LinkedList
//and uses Student ID as the search key and then returns the Student node to be modified
Student* update(Student* head, string student_id)
{
    // Base case
    if (head == NULL) {
        return head;
    }

    // If the Student ID is present in current node, return current node
    if (head->id_no == student_id) {
        return head;
    }

    // Use recursion to go thru the remaining records in the list
    return update(head->next, student_id);
}
```

Figure 3.13: shows function that helps in modifying records

Check if a record already exists in the linked list

The function in the figure below searches the linked list using student id passed in as an argument and returns a Boolean value indicating the result of that search. It is used to prevent duplicate entries into the database.

```
//this function uses serial searching algorithm to avoid duplicate Student IDs
bool check_id(Student* head, string student_id)
{
    // Base case
    if (head == NULL) {
        return false;
    }

    //If the Student ID is present in current node, return true
    if (head->id_no == student_id) {
        return true;
    }

    // Use recursion to go thru the remaining records in the list
    return check_id(head->next, student_id);
}
```

Figure 3.14: shows function that checks if a record already exists in the database

3.3 Implementation of the functions that run the system

Menu method:

This function displays the main menu, collects the user input and calls the appropriate methods based on the user input. It's implementation can be seen in the figure below.

```
//this method will start the program and display the menu
void Student::menu()
{
    //this is where the application starts
    start:
    int first_choice;
    char second_choice;
    cout << "\n\t\t\t\t\t*****" << "\n";
    cout << "\t\t\t\t\t* STUDENT RECORD KEEPING SYSTEM *" << "\n";
    cout << "\t\t\t\t\t*****" << "\n";
    cout << "\t\t\t\t\t1. Enter a New Record" << "\n";
    cout << "\t\t\t\t\t2. Display Record" << "\n";
    cout << "\t\t\t\t\t3. Sort using total score" << "\n";
    cout << "\t\t\t\t\t4. Modify a record" << "\n";
    cout << "\t\t\t\t\t5. Search for a record" << "\n";
    cout << "\t\t\t\t\t6. Delete a record" << "\n";
    cout << "\t\t\t\t\t7. Exit\n";
    cout << "\n";

    cout << "\t\t\t\t\t*****" << "\n";
    cout << "\t\t\t\t\tChoose Options:[1/2/3/4/5/6/7] *" << "\n";
    cout << "\t\t\t\t\t*****" << "\n";
    cout << "Enter Your Choice: ";
    cin >> first_choice;
    switch (first_choice)
    {
        case 1:
            do
            {
                insert();
                cout << "\n\n\t\t\t\t\tAdd Another Student Record (Y, N) : ";
                cin >> second_choice;
            } while (second_choice == 'y' || second_choice == 'Y');
            break;
        case 2:
            display();
            break;
        case 3:
            sort();
            break;
        case 4:
            modify();
            break;
        case 5:
            search();
            break;
        case 6:
            delete_record();
            break;
        case 7:
            cout << "\n\t\t\t\t\tThe Program has been terminated succesfully \n";
            exit(0);
        default:
            cout << "\n\n\t\t\t\t\tInvalid Choice. Please Try Again. \n";
    }

    //use goto statement as a while loop to keep the program running until
    //the user decides to quit
    goto start;
}
```

Figure 3.15: menu method definition

Insert method:

This function collects the fields of a new record from the user and adds the new record to the database. It's implementation can be seen in the figure below.

```
void Student::insert()
{
    //read the records from the database file and add them all to the LinkedList
```

Display method:

This function displays the records in the database. It's implementation can be seen in the figure below.

```

//this method will read from the database.txt file and display the results
void Student::display() // Display data of student
{
    fstream file;
    string record; // this string will be used to store a record from the database
    cout << "\n*****" << "\n";
    cout << "***** Student Record Table *****" << "\n";
    //cout << labels();
    file.open("database.txt", ios::in);
    if (!file)
    {
        cout << "\n\t\t\t Database file doesn't exist!";
        file.close();
    }
    else
    {
        int total = 0; //this variable represents the total number of records in the database
        string record; // this string will be used to store every record in the database

        //Read the records from the database line by line and print them
        while(getline(file, record)) {
            total++;
            cout << total << " " << record << "\n";
        }
    }

    file.close();
}

```

Figure 3.17: display method definition

Sort method:

This function sorts the records in the database using total score. It's implementation can be seen in the figure below.

```
//this method will sort the records in the database in ascending based on total_score  
//using the selection sort algorithm
```

Search method:

This function searches the database for a particular record using either student id or course id. It's implementation can be seen in the figure below.

```
//this method will search the records using Student ID or Course ID  
//depending on what the user wants and then display the results of the search  
void Student::search() {
```


Modify method:

This function can search for a record in the database using student id and if found can modify it's values apart from student id. It's implementation can be seen in the figure below.

```

//this method will modify the contents of any user selected record
void Student::modify() {
    //read the records from the database file and add them to the LinkedList
    //to make searching and modifying the records easier
    fstream file;
    file.open("database.txt", ios::in);
    if (!file) {
        cout << "Database doesn't exist or is empty. Please add some data first \n";
    } else {
        //delete the contents of the LinkedList before using it
        deleteList(&head);
        file >> fname >> lname >> id_no >> course_id >> total_score;

        while (!file.eof())
        {
            //append the record to the LinkedList
            append(&head, fname, lname, id_no, course_id, total_score);

            file >> fname >> lname >> id_no >> course_id >> total_score;
        }

        file.close();

        cout << "Enter the Student ID of the record you want to modify: ";
        string id;
        cin >> id;
        //search for the record using the user provided student id
        //and if the record is found modify it
        Student* temp = update(head, id);
        if(temp != NULL) {
            cout << "Update the student's first Name: ";
            cin >> temp->fname;
            cout << "Update the student's last name: ";
            cin >> temp->lname;
            cout << "Update the student's Course ID: ";
            cin >> temp->course_id;
            cout << "Update the student's Total Score: ";
            cin >> temp->total_score;
            cout << "Record has been successfully updated.\n";

            //now clear the database file then write the updated LinkedList to it
            file.open("database.txt", ios::out);
            file << "";
            file.close();

            //iterate over the updated LinkedList and write it's data to the database file
            file.open("database.txt", ios::out);
            while (head != NULL) {
                file << head->to_string();
                head = head->next;
            }
            file.close();
        } else {
            cout << "Record not found.\n";
        }
    }
}

```

Figure 3.20: modify method definition

Delete method:

This function can search for a record in the database using student id and if found delete it from the database. It's implementation can be seen in the figure below.

```

//this method is used to delete a record from the databse using a student ID
void Student::delete_record() {

    //read the records from the database file and add them to the LinkedList
    //to make searching and deleting easier
    fstream file;
    file.open("database.txt", ios::in);
    if (!file) {
        cout << "Database doesn't exist or is empty. Please add some data first \n";
    } else {
        //delete the contents of the LinkedList before using it
        deleteList(&head);
        file >> fname >> lname >> id_no >> course_id >> total_score;

        while (!file.eof())
        {
            //append the record to the LinkedList
            append(&head, fname, lname, id_no, course_id, total_score);

            file >> fname >> lname >> id_no >> course_id >> total_score;
        }

        file.close();

        //remove the record based on id given by user from the list
        cout << "Enter the ID number of the Student record you want to delete: ";
        string id;
        cin >> id;
        deleteStudent(head, id);

        //now clear the database file then write the updated LinkedList to it
        file.open("database.txt", ios::out);
        file << "";
        file.close();

        //iterate over the updated LinkedList and write it's data to the database file
        file.open("database.txt", ios::out);
        while (head != NULL) {
            file << head->to_string();
            head = head->next;
        }
        file.close();
    }
}

```

Figure 3.21: delete method definition

Chapter Four: Conclusion

Many companies rely on information to help them make better decisions. On a daily basis, a large amount of student data is generated, either manually or electronically. When a school's student population is less than a hundred, the manual approach can be effective, but it is not the ideal way for keeping track of student records. Because these methods of capturing and managing data about students are prone to data inconsistency, data redundancy, difficult to update and maintain data, bad security, difficult to impose constraints on various data files, and difficult to backup, manual and disintegrated electronic systems have numerous disadvantages.

An integrated student database system offers practical solutions to problems that can arise with a manual approach. It is necessary to use historical student records with no missing data in order to measure the school's and students' performance over time. The integrated student database system, which gathers and keeps track of students' longitudinal data, would give accurate and reliable information about present and former students.

Because of the attention given in its development, the system is error-free, highly efficient, and time-saving. All parts of the software development cycle are utilized, and it is worth noting that the system is extremely reliable. The system is designed to accommodate future growth.