**PROJECT STUDENT MANAGEMENT**

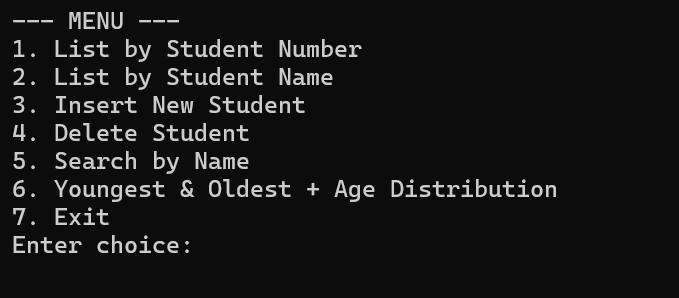
**Name: Gedeon Afofa Lombe**

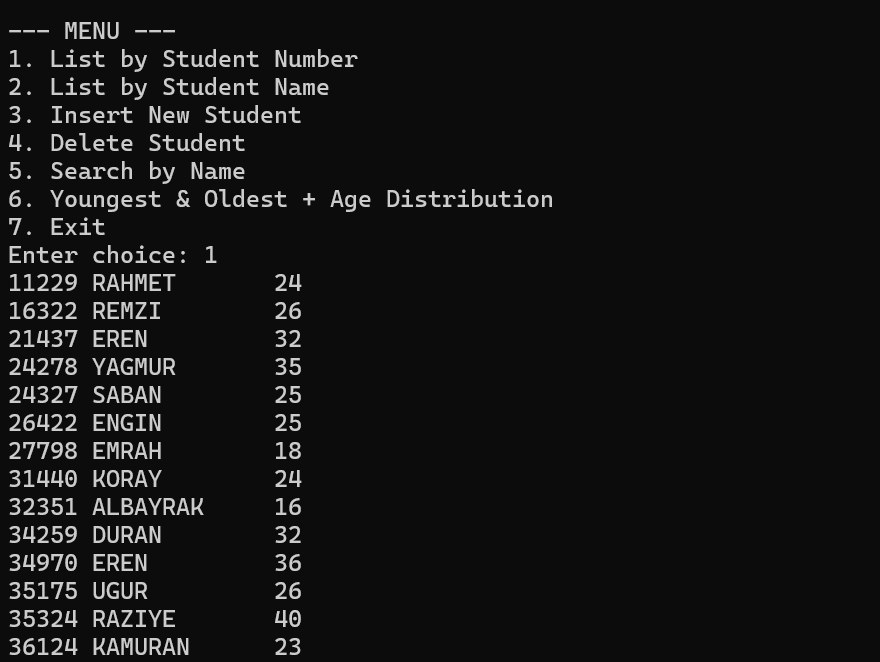
**Student of: Final International University**

**Date: 22/05/2025**

# **User documentation**

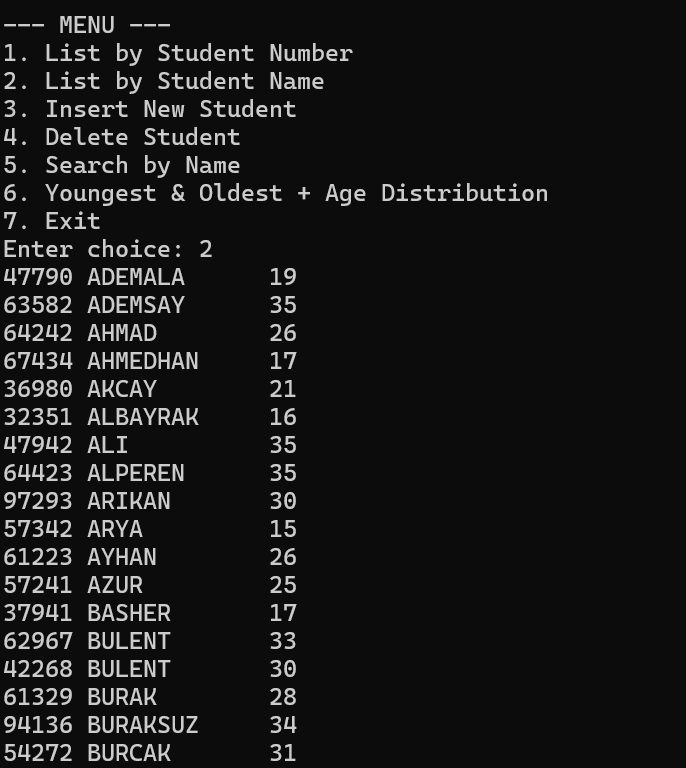
* **Start with The program:** Upon launching, The user will view a menu and pick the program’s function.

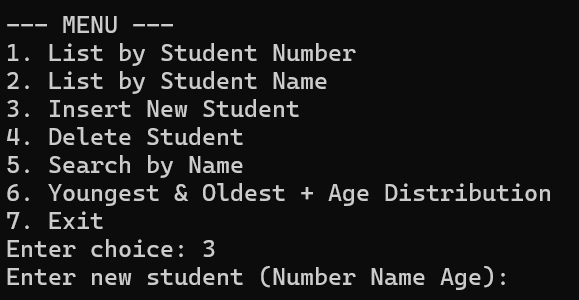
****

* **Pressing 1-sort by Student Number :** when the user presses ‘1’, the system sorts student information by student Number . ****

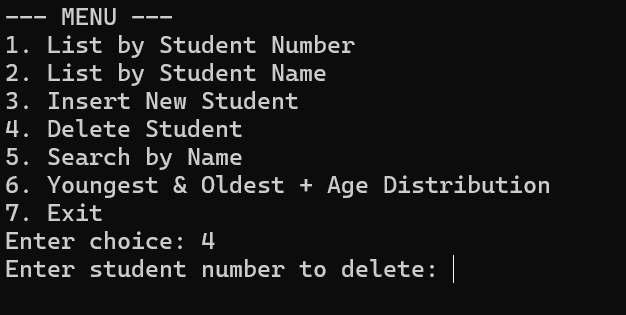
**Once the list is complete, the program will display the menu again**

* **Pressing 2-sort by Name:** when the user presses ‘2’,the program will sort student by Name .

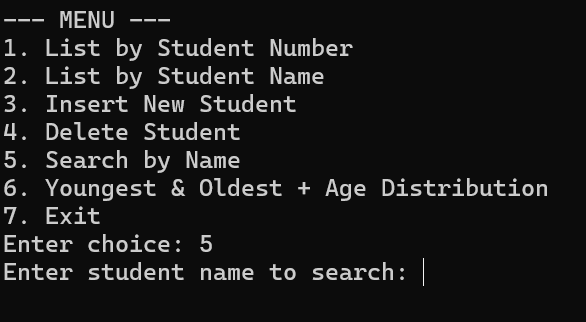
****

* **Pressing 3-Add a Insert new student:** Option ‘3’ enables manual entry of student data by the user. ****

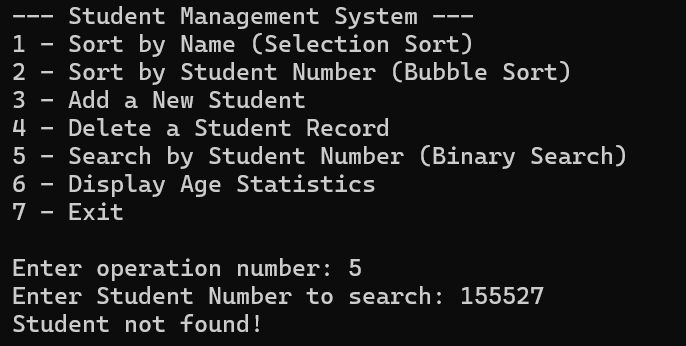
* **Pressing 4-Delete Student Information:** option ‘4’ the program will request a student ID number and delete that particular record

****

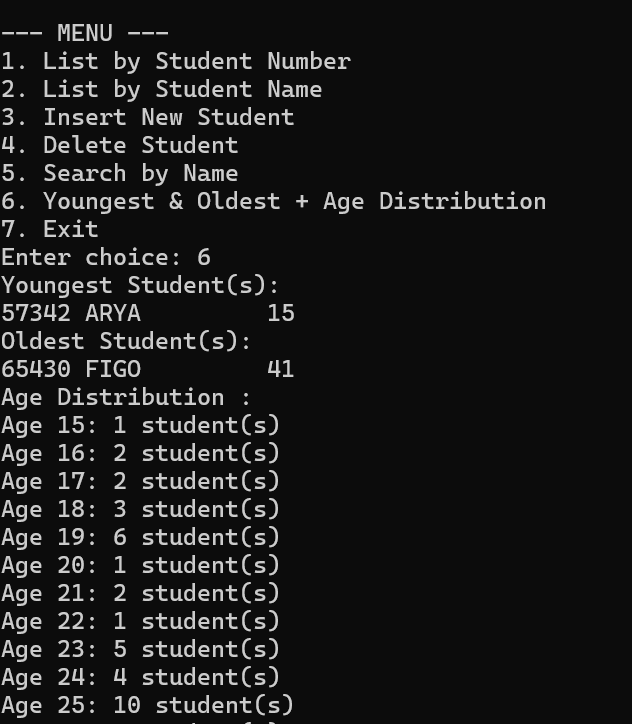
* **Pressing 5-Searching Student:** Option ‘5’retrieve and display a student’s record by entering a Student Name**.**

****

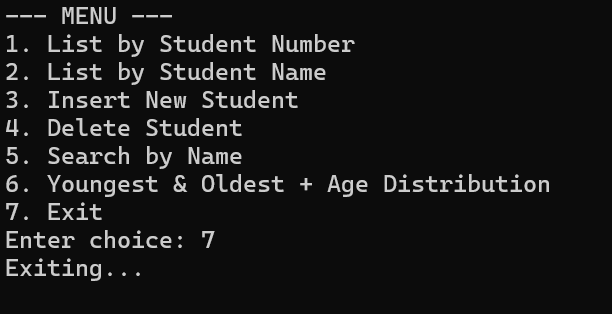
**If The program can not find the student**

****

* **Pressing 6-Display age statistics :** when the user enter ‘6’ , the program will calculates and displays age (youngest and oldest student )

****

* **Pressing 7-EXIT the program :** when the user enter ‘7’, the program will exit

****

# **Input environment**

Program will accept information by two way.

## **Data File (student.txt)**

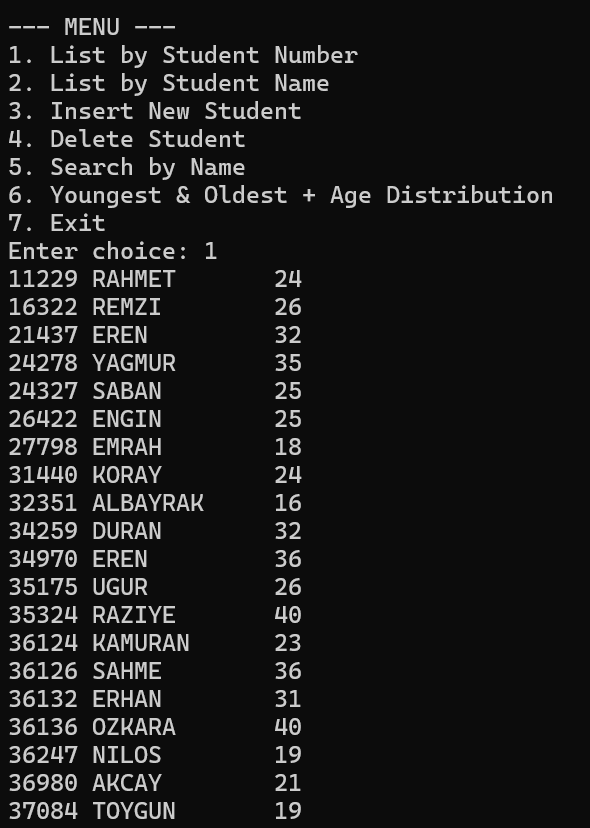
The program accepts data file as input, sequentially reads each student record, and stores them in an ordered array structure to load and display previously saved student data.

## **Scanner (scanf function)**

When the program runs, users can enter data directly via keyboard. The system activates a scanner utility to prompt for and capture user input in real-time .

## **Output environment**

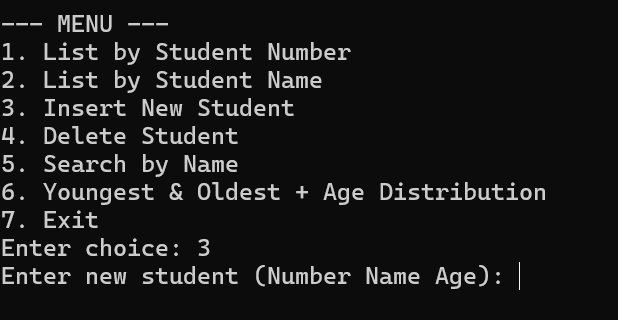
* 1- **sort by student number**: when user enter ‘1’ from keyboard, the program will sort student information by student number 0 to largest number from the Students array



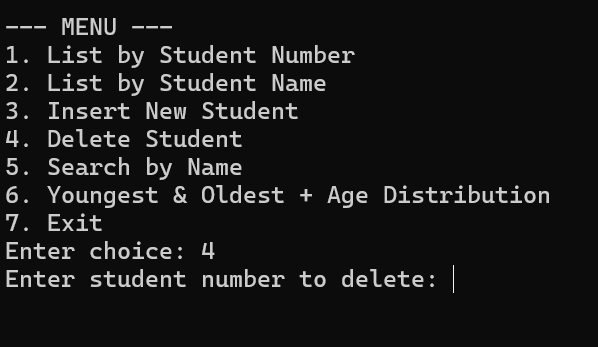
* **2-sort by Name :** when user enter ‘2 ’ from keyboard, program will sort student information by name A to Z from the Students array



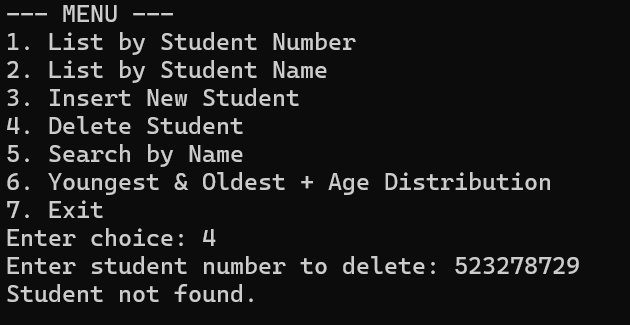
* **3-Insert a new student :** when user enter ‘3’ from keyboard, the program will add student information by user’s input after that list last sort of array



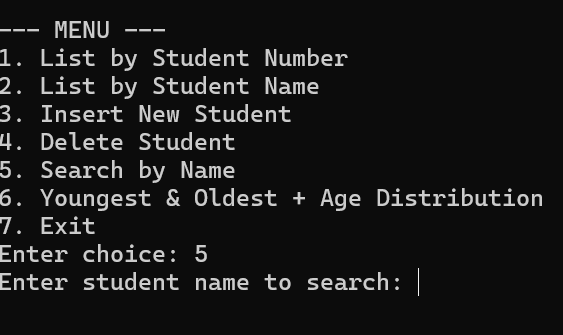
* **4-Deletion a student:** when the user enter ‘4’ from keyboard, program will delete a student base if the Student Number

****

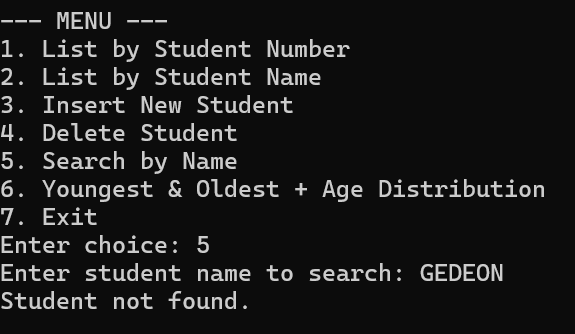
**If the Number is not in the data, program will write “student is not found.”**

****

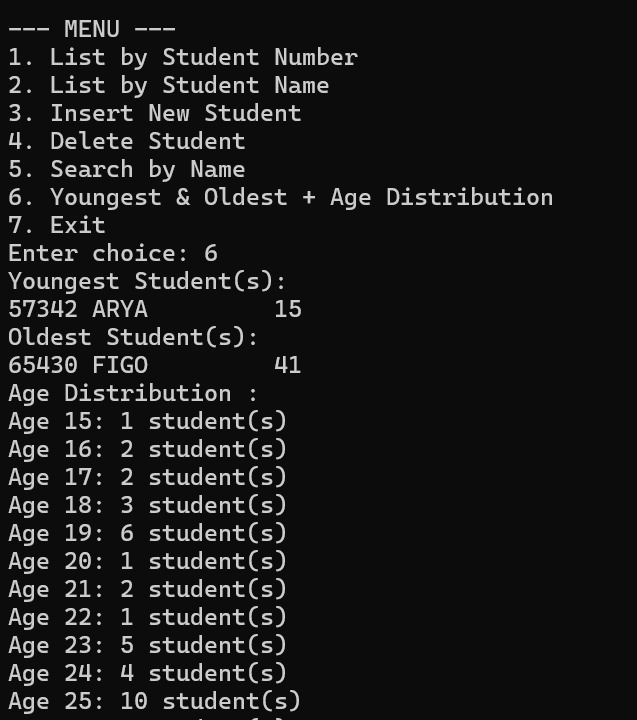
* **5-searching:** when user enter ‘5’ from keyboard, program will search student by Name



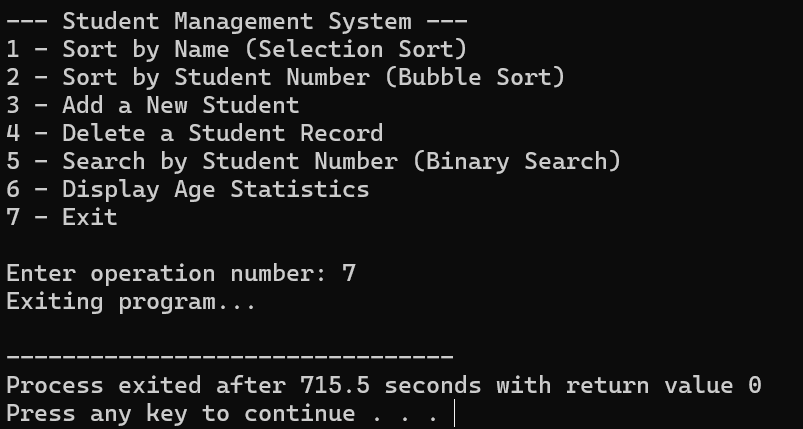
**If the Name is not in the Data**

****

* **6-Display age :** when the user enter ‘6’ the program will display the youngest and oldest student



* **7-EXIT**:when the user enter ‘7’ from the keyboard the program will show exiting program



# **Software design issues**

This program is a menu-driven system for managing student records using linked lists. The data is initially loaded from a student.txt file into a structure array and sorted into two linked lists:

- headNum: Sorted by student number.

- headName: Sorted by student name.

Below is a detailed explanation of each menu option:

* Option 1: List by Student Number (30-by-30)

Displays all students sorted by their student numbers from headNum. Shows 30 students at a time. After every 30 records, the user must press Enter to continue. Useful for administrators reviewing records by ID order.

* Option 2: List by Student Name (25-by-25)

Displays all students sorted alphabetically by their names from headName. Shows 25 students at a time. Uses getchar() to pause after each group. Helps find students in name directories or alphabetical listings.

* Option 3: Insert New Student

Prompts the user to enter a new student’s:

- Student Number (integer)

- Name (string)

- Age (integer)

The student is added into both headNum and headName in their correct sorted position. Ensures that lists stay ordered after insertion.

* Option 4: Delete Student

Asks for a student number to remove.

If the student exists:

- Removes the student from headNum (by number).

- Removes the student from headName (by matching name).

If the student doesn’t exist, shows an appropriate message: "Student not found."

* Option 5: Search by Name

Asks the user to enter a student name.

Searches in headName (sorted by names).

If a match is found, displays the student’s number, name, and age.

If no match, prints "Student not found."

* Option 6: Youngest & Oldest + Age Distribution

Traverses headNum to:

- Identify and display all youngest students (lowest age).

- Identify and display all oldest students (highest age).

- Count and display how many students exist in each age from 15 to 40.

Reflects real-time data including newly inserted or deleted students.

* Option 7: Exit

Safely exits the program.

No files are saved (the program runs in-memory after initial file load).

Input Requirements

Input file: student.txt

Format: StudentNumber Name Age

Example:

47790 MUHSIN 23

63582 AYSE 28

How to Use

1. Compile the program:

Dev c++ -Project2.dev Student\_management.c

2. Ensure student.txt is in the same folder.

3. Run the program:

./student\_management

4. Use the menu to manage students interactively.

# **Program list**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MAX 150

struct studInfo {

int studNr;

char name[12];

int age;

};

struct node {

struct studInfo info;

struct node \*link;

};

typedef struct node\* NODEPTR;

struct studInfo allStudent[MAX];

int studentCount = 0;

NODEPTR headNum = NULL;

NODEPTR headName = NULL;

void fileIntoArray();

NODEPTR insertSortedByNumber(NODEPTR head, struct studInfo student);

NODEPTR insertSortedByName(NODEPTR head, struct studInfo student);

NODEPTR deleteByNumber(NODEPTR head, int studNr);

NODEPTR deleteByName(NODEPTR head, char name[]);

void listStudents(NODEPTR head, int batchSize);

void insertNewStudent();

void deleteStudent();

void searchByName();

void listOldYoung();

void listAgeDistribution();

void menu();

int main() {

fileIntoArray();

for (int i = 0; i < studentCount; i++) {

headNum = insertSortedByNumber(headNum, allStudent[i]);

headName = insertSortedByName(headName, allStudent[i]);

}

menu();

return 0;

}

void fileIntoArray() {

FILE \*fp = fopen("student.txt", "r");

if (!fp) {

printf("Cannot open student.txt\n");

exit(1);

}

while (fscanf(fp, "%d %s %d", &allStudent[studentCount].studNr,

allStudent[studentCount].name, &allStudent[studentCount].age) == 3) {

studentCount++;

}

fclose(fp);

}

NODEPTR insertSortedByNumber(NODEPTR head, struct studInfo student) {

NODEPTR newNode = (NODEPTR)malloc(sizeof(struct node));

newNode->info = student;

newNode->link = NULL;

if (head == NULL || student.studNr < head->info.studNr) {

newNode->link = head;

return newNode;

}

NODEPTR current = head;

while (current->link != NULL && current->link->info.studNr < student.studNr)

current = current->link;

newNode->link = current->link;

current->link = newNode;

return head;

}

NODEPTR insertSortedByName(NODEPTR head, struct studInfo student) {

NODEPTR newNode = (NODEPTR)malloc(sizeof(struct node));

newNode->info = student;

newNode->link = NULL;

if (head == NULL || strcmp(student.name, head->info.name) < 0) {

newNode->link = head;

return newNode;

}

NODEPTR current = head;

while (current->link != NULL && strcmp(current->link->info.name, student.name) < 0)

current = current->link;

newNode->link = current->link;

current->link = newNode;

return head;

}

void listStudents(NODEPTR head, int batchSize) {

int count = 0;

while (head != NULL) {

printf("%5d %-12s %2d\n", head->info.studNr, head->info.name, head->info.age);

count++;

if (count % batchSize == 0) {

printf("Press Enter to continue...\n");

getchar(); getchar();

}

head = head->link;

}

}

void insertNewStudent() {

struct studInfo newStud;

printf("Enter new student (Number Name Age): ");

scanf("%d %s %d", &newStud.studNr, newStud.name, &newStud.age);

headNum = insertSortedByNumber(headNum, newStud);

headName = insertSortedByName(headName, newStud);

printf("\nUpdated list sorted by Number:\n");

listStudents(headNum, 30);

}

void deleteStudent() {

int studNr;

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

scanf("%d", &studNr);

NODEPTR current = headNum;

int found = 0;

char name[12];

while (current != NULL) {

if (current->info.studNr == studNr) {

strcpy(name, current->info.name);

found = 1;

break;

}

current = current->link;

}

if (!found) {

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

return;

}

headNum = deleteByNumber(headNum, studNr);

headName = deleteByName(headName, name);

printf("\nList after deletion (Sorted by Number):\n");

listStudents(headNum, 30);

printf("\nList after deletion (Sorted by Name):\n");

listStudents(headName, 25);

}

NODEPTR deleteByNumber(NODEPTR head, int studNr) {

if (head == NULL) return NULL;

if (head->info.studNr == studNr) {

NODEPTR temp = head->link;

free(head);

return temp;

}

NODEPTR current = head;

while (current->link != NULL && current->link->info.studNr != studNr)

current = current->link;

if (current->link != NULL) {

NODEPTR temp = current->link;

current->link = temp->link;

free(temp);

}

return head;

}

NODEPTR deleteByName(NODEPTR head, char name[]) {

if (head == NULL) return NULL;

if (strcmp(head->info.name, name) == 0) {

NODEPTR temp = head->link;

free(head);

return temp;

}

NODEPTR current = head;

while (current->link != NULL && strcmp(current->link->info.name, name) != 0)

current = current->link;

if (current->link != NULL) {

NODEPTR temp = current->link;

current->link = temp->link;

free(temp);

}

return head;

}

void searchByName() {

char name[12];

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

scanf("%s", name);

NODEPTR current = headName;

while (current != NULL) {

if (strcmp(current->info.name, name) == 0) {

printf("Found: %d %s %d\n", current->info.studNr, current->info.name, current->info.age);

return;

}

current = current->link;

}

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

}

void listOldYoung() {

if (headNum == NULL) return;

NODEPTR current = headNum;

int youngestAge = current->info.age;

int oldestAge = current->info.age;

while (current != NULL) {

if (current->info.age < youngestAge)

youngestAge = current->info.age;

if (current->info.age > oldestAge)

oldestAge = current->info.age;

current = current->link;

}

printf("Youngest Student(s):\n");

current = headNum;

while (current != NULL) {

if (current->info.age == youngestAge)

printf("%5d %-12s %2d\n", current->info.studNr, current->info.name, current->info.age);

current = current->link;

}

printf("Oldest Student(s):\n");

current = headNum;

while (current != NULL) {

if (current->info.age == oldestAge)

printf("%5d %-12s %2d\n", current->info.studNr, current->info.name, current->info.age);

current = current->link;

}

}

void listAgeDistribution() {

int ageDist[41] = {0};

NODEPTR current = headNum;

while (current != NULL) {

if (current->info.age >= 15 && current->info.age <= 40)

ageDist[current->info.age]++;

current = current->link;

}

printf("Age Distribution :\n");

for (int i = 15; i <= 40; i++) {

if (ageDist[i] > 0)

printf("Age %2d: %d student(s)\n", i, ageDist[i]);

}

}

void menu() {

int choice;

do {

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

printf("1. List by Student Number \n");

printf("2. List by Student Name \n");

printf("3. Insert New Student\n");

printf("4. Delete Student\n");

printf("5. Search by Name\n");

printf("6. Youngest & Oldest + Age Distribution\n");

printf("7. Exit\n");

printf("Enter choice: ");

scanf("%d", &choice);

switch (choice) {

case 1: listStudents(headNum, 30); break;

case 2: listStudents(headName, 25); break;

case 3: insertNewStudent(); break;

case 4: deleteStudent(); break;

case 5: searchByName(); break;

case 6: listOldYoung();

listAgeDistribution();

break;

case 7: printf("Exiting...\n");

break;

default: printf("Invalid choice.\n");

}

} while (choice != 7);

}

# **Programming Time**

**Analyze and design : 3hours**

**Coding: 8 hours**

**Debugging: 10hours**

**Documentation : 3 hours**

**Total : 24hours**