```
/*
* Author: Linh Phan
* Course: CS1
* Assignment: Assignment 4
* Date: 4/20/2025 Spring 2025
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

typedef struct Student {
      char first[30], last[30];
      int score;
      struct Student* next;
} Student;

//function prototypes
Student* addStudent(char* lastName, char* firstName, int score);
void displayStudents(Student* head);
void processStudentData(char* fileName, Student** head);
Student* mergeSortedLists(Student* stuOne, Student* stuTwo);
void splitList(Student* originalHead, Student** frontRef, Student** backRef);
void mergeSort(Student** originalHead);

//create a node and allocate memory for it
Student* addStudent(char* lastName, char* firstName, int score) {
      Student* newStudent = (Student*)malloc(sizeof(Student));
      if (newStudent == NULL) {
            printf("Memory allocation failed!\n");
            return NULL;
      }

      //setting the data + setting last 'letter' to '\0'
      strncpy(newStudent->last, lastName, sizeof(newStudent->last) - 1);
      newStudent->last[sizeof(newStudent->last) - 1] = '\0';

      strncpy(newStudent->first, firstName, sizeof(newStudent->first) - 1);
      newStudent->first[sizeof(newStudent->first) - 1] = '\0';

      newStudent->score = score;
      newStudent->next = NULL;

      return newStudent;
}

//printing output
void displayStudents(Student* head) {
      //variables
      float sum = 0;
      int index = 0;
      Student* temp = head;
      Student* lastNode = NULL;
      float medianScore = 0;
      float secondMedianScore = 0;

      //prints the sorted list while keeping track of sum and index
      printf("\nSorted List by Score:\n");
      while (temp != NULL) {
            printf("%s %s - %d\n", temp->last, temp->first, temp->score);
            sum = sum + temp->score;
            index++;
            lastNode = temp;
            temp = temp->next;
      }

      //calculate average and made a new Student to calculate median
      sum = sum/index;
      Student* median = head;

      //if the amount of students is odd, sets the median to middle
      if((index/2)%2 == 0) {
            for(int i = 0; i < (index/2); i++) {
                  median = median->next;
            }
            medianScore = median->score;
      }
      //if amount of students is even, set median to average of middle 2
      else if ((index/2)%2 == 1) {
            for(int i = 0; i < (index/2); i++) {
                  medianScore = median->score;
                  median = median->next;
                  secondMedianScore = median->score;
            }
            medianScore = (medianScore + secondMedianScore)/2;
      }

      //prints scores
      printf("\nHighest Score: %d", head->score);
      printf("\nLowest Score: %d", lastNode->score);
      printf("\nMedian Score: %.2f", medianScore);
      printf("\nAverage Score: %.2f\n", sum);

      //prints top 5
      temp = head;
      printf("\nTop 5 Students:\n");
      for (int i = 0; i < 5; i++) {
            printf("%s %s - %d\n", temp->last, temp->first, temp->score);
            temp = temp->next;
      }
}

//read input.txt and append to linked list
void processStudentData(char* fileName, Student** head) {
      //open files
      FILE* file = fopen(fileName, "r");
      if (file == NULL) {
            printf("Error: Could not open the file %s\n", fileName);
            return;
      }

      //variables
      char firstName[30], lastName[30];
      int score;
      Student* tail = *head;

      //append each student to a node
      while (fscanf(file, "%s %s %d", lastName, firstName, &score) != EOF) {
```

```c
            Student* newStudent = addStudent(lastName, firstName, score);
            //sets the head
            if (*head == NULL) {
                  *head = newStudent;
                  tail = newStudent;
            } else {
                  //if there's alr a head then update tail
                  tail->next = newStudent;
                  tail = newStudent;
            }
      }

      fclose(file);
}

//merge the two sorted linked list
Student* mergeSortedLists(Student* stuOne, Student* stuTwo) {
      //returns other student if one is null
      if (!stuOne) return stuTwo;
      if (!stuTwo) return stuOne;

      Student* result = NULL;

      //compare scores and then merge
      if (stuOne->score >= stuTwo->score) {
            result = stuOne;
            result->next = mergeSortedLists(stuOne->next, stuTwo);
      } else {
            result = stuTwo;
            result->next = mergeSortedLists(stuOne, stuTwo->next);
      }

      return result;
}

//split orignal link list in half
void splitList(Student* originalHead, Student** frontRef, Student** backRef) {
      Student* middle = originalHead;
      Student* last = originalHead->next;

      //finding middle and last in the list
      while (last) {
            last = last->next;
            if (last) {
                  middle = middle->next;
                  last = last->next;
            }
      }

      //splitting or setting list in half
      *frontRef = originalHead;
      *backRef = middle->next;
      middle->next = NULL;
}

//initiate merge sort on linked list
void mergeSort(Student** originalHead) {
      Student* head = *originalHead;
      //stops if linked list is null or has one element
```

```
	if (head == NULL || head->next == NULL) {
		return;
	}

	//variables
	Student* stuOne;
	Student* stuTwo;

	//calls function to split and then recursively sort it
	splitList(head, &stuOne, &stuTwo);
	mergeSort(&stuOne);
	mergeSort(&stuTwo);

	//merge it back together and then sets the head to the linked list
	*originalHead = mergeSortedLists(stuOne, stuTwo);
}

int main() {
	//variables
	char fileName[100];
	Student* head = NULL;

	//check for input name
	printf("Enter input file name: ");
	scanf("%s", fileName);

	processStudentData(fileName, &head);

	//if processStudentData worked or if there were input
	//then continue but if not then ends program
	if(head != NULL) {
		mergeSort(&head);
		displayStudents(head);
		//free list
		while (head != NULL) {
			Student* temp = head;
			head = head->next;
			free(temp);
		}
	}
	return 0;
}
```