//COS10007 – Technical Developing Software  
Student ID – 103234103  
Lab 3 – Week 3//  
  
1/   
#include <stdio.h>

#include <stdlib.h>

#include <time.h>

// define the structure for a linked list node

struct node {

int data;

struct node \*next;

};

typedef struct node Node; // define a shorthand for the struct

// function prototypes

void insert(Node \*\*head, int data);

double average(Node \*head);

int main() {

Node \*head = NULL; // initialize the linked list as empty

srand(time(NULL)); // seed the random number generator with the current time

// insert 10 random integers between 0 and 50 in order

for (int i = 0; i < 10; i++) {

int data = rand() % 51; // generate a random integer between 0 and 50

insert(&head, data); // insert the integer in the correct order in the linked list

}

// print the linked list

printf("Linked list: ");

Node \*current = head;

while (current != NULL) {

printf("%d ", current->data);

current = current->next;

}

printf("\n");

// calculate and print the average of the elements in the linked list

printf("Average: %.2f\n", average(head));

return 0;

}

// function to insert a new node in the linked list in order

void insert(Node \*\*head, int data) {

Node \*new\_node = (Node\*) malloc(sizeof(Node)); // allocate memory for a new node

new\_node->data = data; // assign the data to the new node

new\_node->next = NULL; // set the next pointer of the new node to NULL

if (\*head == NULL) { // if the linked list is empty, set the new node as the head

\*head = new\_node;

} else if (data <= (\*head)->data) { // if the new node is less than or equal to the head, insert it at the beginning

new\_node->next = \*head;

\*head = new\_node;

} else { // otherwise, insert the new node in the correct order

Node \*current = \*head;

while (current->next != NULL && data > current->next->data) {

current = current->next;

}

new\_node->next = current->next;

current->next = new\_node;

}

}

// function to calculate the average of the elements in the linked list

double average(Node \*head) {

int sum = 0;

int count = 0;

Node \*current = head;

while (current != NULL) {

sum += current->data;

count++;

current = current->next;

}

if (count == 0) { // handle the case where the linked list is empty

return 0.0;

} else {

return (double) sum / count;

}

}  
\*\*Example Output\*\*  
Text

Description automatically generated

2/  
#include<stdio.h>  
#include<stdlib.h>

a/

//open file for reading

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

if(fp == NULL){

printf("Error opening file.\n");

exit(1);

}

//create a student node

StudentPtr head = (StudentPtr) malloc(sizeof(Student));

if(head == NULL){

printf("Memory allocation failed.\n");

exit(1);

}

//read data from file and assign to student node

fscanf(fp, "%s %d %d %s", head->studentInfo.studentName, &head->courseInfo.courseId, &head->studentInfo.studentId, head->courseInfo.courseName);

head->nextPtr = NULL;

//close file

fclose(fp);  
  
b/  
//assuming file format is as follows: studentName courseId studentId courseName

//open file for reading

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

if(fp == NULL){

printf("Error opening file.\n");

exit(1);

}

//create a student node

StudentPtr head = (StudentPtr) malloc(sizeof(Student));

if(head == NULL){

printf("Memory allocation failed.\n");

exit(1);

}

//read data from file and assign to first student node

fscanf(fp, "%s %d %d %s", head->studentInfo.studentName, &head->courseInfo.courseId, &head->studentInfo.studentId, head->courseInfo.courseName);

head->nextPtr = NULL;

//create 9 more student nodes and add to linked list in ascending order of studentId

int i;

StudentPtr currPtr, prevPtr;

for(i = 1; i < 10; i++){

StudentPtr newPtr = (StudentPtr) malloc(sizeof(Student));

if(newPtr == NULL){

printf("Memory allocation failed.\n");

exit(1);

}

fscanf(fp, "%s %d %d %s", newPtr->studentInfo.studentName, &newPtr->courseInfo.courseId, &newPtr->studentInfo.studentId, newPtr->courseInfo.courseName);

newPtr->nextPtr = NULL;

currPtr = head;

prevPtr = NULL;

while(currPtr != NULL && newPtr->studentInfo.studentId > currPtr->studentInfo.studentId){

prevPtr = currPtr;

currPtr = currPtr->nextPtr;

}

if(prevPtr == NULL){

newPtr->nextPtr = head;

head = newPtr;

}

else{

prevPtr->nextPtr = newPtr;

newPtr->nextPtr = currPtr;

}

}

//close file

fclose(fp);

//print the student list

currPtr = head;

while(currPtr != NULL){

printf("%s %d %d %s\n", currPtr->studentInfo.studentName, currPtr->courseInfo.courseId, currPtr->studentInfo.studentId, currPtr->courseInfo.courseName);

currPtr = currPtr->nextPtr;

}  
  
c/  
//assuming the linked list is already created

//open file for writing

FILE \*fp = fopen("output.txt", "w");

if(fp == NULL){

printf("Error opening file.\n");

exit(1);

}

//write each student node to file

StudentPtr currPtr = head;

while(currPtr != NULL){

fprintf(fp, "%s %d %d %s\n", currPtr->studentInfo.studentName, currPtr->courseInfo.courseId, currPtr->studentInfo.studentId, currPtr->courseInfo.courseName);  
  
d/  
void deleteStudent(StudentPtr \*listPtr, char name[]) {

StudentPtr previousPtr = NULL;

StudentPtr currentPtr = \*listPtr;

while (currentPtr != NULL) {

if (strcmp(currentPtr->studentInfo.studentName, name) == 0) {

if (previousPtr == NULL) {

\*listPtr = currentPtr->nextPtr;

} else {

previousPtr->nextPtr = currentPtr->nextPtr;

}

free(currentPtr);

return;

}

previousPtr = currentPtr;

currentPtr = currentPtr->nextPtr;

}

printf("Student %s not found in the list.\n", name);

}  
e/  
void editStudent(StudentPtr \*listPtr, char name[]) {

StudentPtr previousPtr = NULL;

StudentPtr currentPtr = \*listPtr;

while (currentPtr != NULL) {

if (strcmp(currentPtr->studentInfo.studentName, name) == 0) {

int newId;

printf("Enter new student ID (1-100): ");

scanf("%d", &newId);

currentPtr->studentInfo.studentId = newId;

if (previousPtr == NULL) {

\*listPtr = currentPtr->nextPtr;

} else {

previousPtr->nextPtr = currentPtr->nextPtr;

}

StudentPtr tmpPtr = \*listPtr;

while (tmpPtr != NULL && tmpPtr->studentInfo.studentId < newId) {

previousPtr = tmpPtr;

tmpPtr = tmpPtr->nextPtr;

}

if (previousPtr == NULL) {

currentPtr->nextPtr = \*listPtr;

\*listPtr = currentPtr;

} else {

previousPtr->nextPtr = currentPtr;

currentPtr->nextPtr = tmpPtr;

}

return;

}

previousPtr = currentPtr;

currentPtr = currentPtr->nextPtr;

}

printf("Student %s not found in the list.\n", name);

}