Week Lab 3

1. **Write a complete C program that inserts 10 random integers between 0 and 50 in order in a linked list. Create a separate function to calculate the floating-point average of the elements. You must use a self-referential structure to create the linked list for this program.**

**Code:**

#include<bits/stdc++.h>

using namespace std;

//Declaring Node as structure

struct Node

{

int data;

struct Node\*next; //pointer to next link

};

//creating function to insert node into the linked list/

void insert\_no(struct Node\*\* head\_ref, struct Node\* node\_new)

{

struct Node\* current;

/\*checking that node is first node in the linked list OR data of node\_new is

less or equal to the data of that node which head\_ref is pointing\*/

if (\*head\_ref == NULL || (\*head\_ref)->data >= node\_new->data)

{

//node\_new will point to that node which is pointed by head\_ref pointer/

node\_new->next = \*head\_ref;

/\* head\_ref will now point to node\_new\*/

\*head\_ref = node\_new;

}

else

{

current = \*head\_ref;

/\*if new node has to be inserted in between two node,

find its correct inorder position in the linked list \*/

while (current->next!=NULL && current->next->data < node\_new->data)

{

current = current->next;

}

//insert node to its correct position/

node\_new->next = current->next;

current->next = node\_new;

}

}

/\*function to create a new node with randomly generated data

and this will return newly created node\*/

struct Node \*newNode(int new\_data)

{

//creating a new node/

struct Node\* node\_new = new struct Node;

//assigning data to newly created node/

node\_new->data = new\_data;

//making its pointer NULL/

node\_new->next = NULL;

return node\_new;

}

// Function to find floating-point average of all the values in the linked list

float avg(struct Node\*\* head\_ref)

{

int sum=0;

Node \*h=\*head\_ref ;

/\*Checking h is not equal to NULL as we are traversing to end \*/

while(h!=NULL)

{

sum=sum+h->data;

h=h->next;

}

// returning Average

return (float)sum/10;

}

//driver function

int main()

{

// creating pointers

struct Node\*head=NULL,\*temp,\*newnode;

int d;

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

{

// generating random values from range 0 to 50

d=rand()%51;

// Calling function for inserting 10 random integer inorder(ascending)

insert\_no(&head,newNode(d));

}

temp=head;

// Loop for printing the random values

while(temp)

{

cout<<temp->data<<" ";

temp=temp->next;

}

cout<<endl;

// Calling avg function and printing the value

cout<<"Average of numbers in the above linked list is : "<<avg(&head)<<endl;

}

**Output:**



1. **Write a statement or set of statements to accomplish each of the following. Use functions wherever necessary.**

Use the following structure definition to answer questions 2.a.b.c.d and e

*struct personal{*

*char studentName[20];*

*char studentId[10];*

*};*

*struct course{*

*char courseName[20];*

*char courseId[10];*

*};*

*struct student{*

*struct personal studentInfo;*

*struct course courseInfo;*

*struct student \*next;*

*};*

*typedef struct student Student;*

*typedef Student \*StudentPtr;*

a. Write C codes to create LinkedList that contains student nodes. The program should read the values for the studentName and courseName from the user, assign a random number between 1 – 100 to the studentId and random number between 100 – 1000 to the courseId. The next pointer to be assigned NULL as the value.

b. Write C codes to create a Linked List of 10 students. Student nodes in the Linked List to be sorted based on the studentId.

c. Write C code to write the student list on to a text file.

d. Write C code to read a student name. Search for the student name in the list and if found, delete the node from the list.

e. Write C code to read a student name. Search for the student name in the list and if found, allow the user to edit the studentId. (Ensure the new ID is between 1 – 100). Reposition the student node based on the new studentId.

**Code:**

#include <stdio.h>

#include<stdlib.h>

#include<string.h>

//You can define your own numbers but keep it less than 100

#define countOfStudent 10

struct personal{

char studentName[20];

char studentId[10];

};

struct course{

char courseName[20];

char courseId[10];

};

struct student{

struct personal studentInfo;

struct course courseInfo;

struct student \*next;

};

//Global Variables

//needed to keep the randomised uniques student Ids

char vector[countOfStudent][5];

int i=0;

typedef struct student Student;

Student \*head =NULL;

//creation of new node

Student\* createNode(){

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

//copy the random value from random array to this value

strcpy(node->studentInfo.studentId,vector[i++]);

printf("Enter the Student Name: ");

scanf("%[^\n]%\*c",node->studentInfo.studentName);

printf("Enter the course name: ");

scanf("%[^\n]%\*c",node->courseInfo.courseName);

printf("\n");

int num = (rand() %(1000 - 100 + 1)) + 100;

sprintf(node->courseInfo.courseId,"%d",num);

node->next=NULL;

return node;

}

//Inserting the node value according to their student id

void insert(Student \*node){

Student \*temp=head;

if(head==NULL){

head=node;

}

else if(strcmp(node->studentInfo.studentId,head->studentInfo.studentId)<0){

node->next=head;

head=node;

}

else

{

while(temp->next!=NULL && strcmp(temp->next->studentInfo.studentId,node->studentInfo.studentId)<0 ){

temp=temp->next;

}

node->next=temp->next;

temp->next=node;

}

}

/\* Given a reference (pointer to pointer) to the head of a list and a key, deletes the first occurrence of key in linked list \*/

int deleteNode( char key[])

{

Student \* temp = head, \*prev;

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

if (temp != NULL && strcmp(head->studentInfo.studentName,key)==0)

{

head= temp->next; // Changed head

free(temp); // free old head

return 1;

}

// Search for the key to be deleted, keep track of the

// previous node as we need to change 'prev->next'

while (temp != NULL && strcmp(temp->studentInfo.studentName,key)!=0)

{

prev = temp;

temp = temp->next;

}

// If key was not present in linked list

if (temp == NULL) {

return -1;

}

// Unlink the node from linked list

prev->next = temp->next;

free(temp);

return 1;

}

void printLL(){

if(head==NULL){

printf("Empty List\n");

}

else{

Student \*temp=head;

printf("%-20s%-20s%-20s%-20s\n","Student ID","Student Name","Course Id","Course Name");

while (temp!=NULL){

printf("%-20s%-20s%-20s%-20s\n",temp->studentInfo.studentId,temp->studentInfo.studentName,temp->courseInfo.courseId,temp->courseInfo.courseName);

temp=temp->next;

}

printf("\n");

}

}

void delAndRep(char key[]){

Student \* temp = head, \*prev;

char newkey[5];

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

node->next=NULL;

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

if (temp != NULL && strcmp(head->studentInfo.studentId,key)==0)

{

printf("Key Found!!! Enter the new key: ");

gets(newkey);

head= temp->next;

strcpy(node->studentInfo.studentId,newkey);

strcpy(node->studentInfo.studentName,temp->studentInfo.studentName);

strcpy(node->courseInfo.courseName,temp->courseInfo.courseName);

strcpy(node->courseInfo.courseId,temp->courseInfo.courseId);

// Changed head

free(temp);

insert(node);

return;// free old head

}

// Search for the key to be deleted, keep track of the

// previous node as we need to change 'prev->next'

while (temp != NULL && strcmp(temp->studentInfo.studentId,key)!=0)

{

prev = temp;

temp = temp->next;

}

// If key was not present in linked list

if (temp == NULL) {

printf("The Given ID is not available \n");

return;

}

printf("Key Found!!! Enter the new key: ");

gets(newkey);

//copying the old values to new values

strcpy(node->studentInfo.studentId,newkey);

strcpy(node->studentInfo.studentName,temp->studentInfo.studentName);

strcpy(node->courseInfo.courseName,temp->courseInfo.courseName);

strcpy(node->courseInfo.courseId,temp->courseInfo.courseId);

insert(node);

prev->next = temp->next;

free(temp);

}

void writeInFile(){

Student \*temp=head;

FILE \*fptr;

fptr=fopen("listOfStudent.txt","w+");

fprintf(fptr,"%-20s%-20s%-20s%-20s\n","Student ID","Student Name","Course Id","Course Name");

while(temp!=NULL){

fprintf(fptr,"%-20s%-20s%-20s%-20s\n",temp->studentInfo.studentId,temp->studentInfo.studentName,temp->courseInfo.courseId,temp->courseInfo.courseName);

temp=temp->next;

}

fclose(fptr);

}

//here generating random number is very easy but generating unique random numbers is a difficult one

void uniqueRandomNum(){

int list[100];

for(int i=0;i<100;i++){

list[i]=i;

}

for(int i=0;i<100;i++){

int j=i+rand()%(100-i);

int temp=list[i];

list[i]=list[j];

list[j]=temp;

}

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

sprintf(vector[i],"%d",list[i]);

}

}

int main() {

//calling unique number creator

uniqueRandomNum();

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

insert(createNode());

}

printLL();

//writting the list in the file

writeInFile();

printf("Enter the name student to be deleted:");

char name[20];

gets(name);

int verify=deleteNode(name);

if(verify==1){

printf("Student Deleted \n");

}

else if(verify==-1){

printf("Student Not Present in the List\n");

}

printf("After Deletion Operation:\n");

printLL();

printf("Enter the student Id to be replaced: ");

char idtoreplace[5];

gets(idtoreplace);

delAndRep(idtoreplace);

printf("After replacement: \n");

printLL();

}

**Output:**







