Assignment 7

Introduction to programming in C

Question 1

CGPA calculation

You are the new tech administrator of a college. The college offers four courses listed below.

Course Name	Course Code	Credits
Science	1001	10
Maths	1002	5
Literature	1003	5
Philosophy	1004	1

Each student has to take exactly three courses and she gets a grade from (0 - 10) on the course, depending on her performance.

The CGPA of a student is calculated as $cgpa = \frac{\sum_{i=1}^{3} credits_i * grade_i}{\sum_{i=1}^{3} credits_i}$.

That is for each course the student took, you multiply the grade obtained with the credits of the course. Sum this value over each course, and divide it by the total credits of all courses the student took.

The previous administrator wrote an incomplete C program for a CGPA calculator, using structures.

Complete the C code for the program, by writing the code for the function $float\ calculate\_gpa(student\ s)$;

This function takes as an input the struct student variable of a student s, which contains all the information about the courses the student s took and her grades in the courses. You have to return the total cgpa of the student as the output.

Input

The first line contains the number of students n. The next n lines contains the information on the students in the following order:

Name CourseCode1 Marks1 CourseCode2 Marks2 CourseCode3 Marks3

Output

The names and CGPAs (to a single decimal point) of students, line by line in the following format: Name CGPA.

Solution

```c
#include<stdio.h>

#include<stdlib.h>

struct course{
    int code;
    int credits;
};

typedef struct course course;

course science = {1001, 10};
course maths = {1002, 5};
course literature = {1003, 5};
course philosophy = {1004, 1};

struct student{
  char name[20];
  course courses[3];
  int grades[3];
};

typedef struct student student;

float calculate_gpa(student s){
    int total=0;
    int credits=0;
    for(int i = 0 ; i < 3; i++){
        total += s.grades[i] * (s.courses[i].credits);
        credits += s.courses[i].credits;
    }
    return (float)total/credits;
}

void print(student * s, int n) {
  int i;
  for (i = 0; i < n; i++) {
    printf("%s %.1f\n", s[i].name, calculate_gpa(s[i]));
  }
}


int main() {
  int i,j, n;
  scanf("%d", & n);
  student *student_info = (student * ) malloc(sizeof(student) * n);
  for (i = 0; i < n; i++) {
    scanf("%s", student_info[i].name);
    for(j = 0 ; j < 3; j ++){
        int coursecode;
        scanf("%d", &coursecode);
        switch(coursecode){
            case 1001:
            student_info[i].courses[j] = science;
            break;
            case 1002:
            student_info[i].courses[j] = maths;
            break;
            case 1003:
            student_info[i].courses[j] = literature;
            break;
            case 1004:
            student_info[i].courses[j] = philosophy;
            break;
        }
        scanf("%d", &student_info[i].grades[j]);
    }
  }
  print(student_info, n);
  return 0;
}
```

Question 2

Linked List Operations

In this question, a linked list is partially implemented where each element in the linked list is a structure of the following format:

```
struct node{
        int id;
        int priority;
        struct node *next;
};
```

The field priority is a positive integer, which denotes the priority of an element inside the list.

You have to complete the C code for performing the following operations in the linked list:

- Create and return a node e with given id and value val

  struct node * create_node(int id, int val);

- Add an node e to the beginning of the list. Return the new list.

  struct node * append(struct node * list, struct node * e);

- Search for a node e with id inside the list. Return a pointer to e if found, else return NULL.

  struct node * search(struct node * list, int id);

- Change the value of an element with given id (if found), in the list to the new value val.

  void change_priority((struct node *list, int id, int val);

Note: The code for manipulating the input as well as output is given to you. You only have to write code for the incomplete functions.

Input

A set of lines, each lines containing a character representing the operation and its inputs.

The operation can be one of the following:

- *A* $< id >< val >$

  Add an node with id and val to the list, at the start of the list.

- *C* $< id >< val >$

  Change the value of the element with id to val. If an element with this id is not found, do nothing.

- *S* $< id >$

  If an element with the id is in the list print the id and the value and a newline. Else, print the id and -1 and a newline.

- *E*

  End of input, exit from the program

Output The output of search queries.

Solution

```c
#include <stdio.h>

#include <stdlib.h>

struct node {
  int id;
  int value;
  struct node * next;
};

struct node * create_node(int id, int val) {
  struct node * new_node;
  new_node = (struct node * ) malloc(sizeof(struct node));
  new_node -> id = id;
  new_node -> value = val;
  new_node -> next = NULL;
  return new_node;
}

struct node * append(struct node * list, struct node * e) {
  if (list == NULL)
    list = e;
  else {
    e -> next = list;
  }
  return e;
}

struct node * search(struct node * list, int id) {
  while (list != NULL) {
    if (list -> id == id)
      return list;
    list = list -> next;
  }
  return NULL;
}

void change_value(struct node * list, int id, int val) {
  struct node * e = search(list, id);
  if (e != NULL)
    e -> value = val;
}

int find_value(struct node * list, int id) {
  struct node * e = search(list, id);
  if (e != NULL)
    return e -> value;
  return -1;
}


int main() {
  char op;
  int id, val;
  struct node * list = NULL;

  int flag = 1;
  do {
    scanf("%c ", & op);
    switch (op) {
    case 'A':
      scanf("%d %d", & id, & val);
      list = append(list, create_node(id, val));
      break;
    case 'S':
      scanf("%d", & id);
      printf("%d %d\n", id, find_value(list, id));
      break;
    case 'C':
      scanf("%d %d", & id, & val);
      change_value(list, id, val);
      break;
    case 'E':
      flag = 0;
    }
  } while (flag == 1);
  return 0;
}
```