```
//HW2
//Due: 11:59PM, Friday September 29.

#include <iostream>
#include <list>
#include <map>
#include <string>
using namespace std;

class course {
public:
        string name;//CIS554
        int section;//1
        int credits;//3
        string grade;//A-
        course() {}
        course(string n, int s, int c, string g) { name = n; section = s; credits = c; grade
= g; }

        bool operator<(const course& c) const;
        bool operator==(const course& c) const;
        float num_grade();

};
float course::num_grade() {
        map<string, float> M{
                {"A", 4.0f},
                {"A-", 3.667f},
                {"B+",  3.333f},
                {"B",   3.0f},
                {"B-" ,  2.667f},
                {"C+" ,  2.333f},
                {"C",   2.0f},
                {"C-" ,  1.667f},
                {"D",   1.0f},
                {"D-" , 0.667f},
                {"F",   0.0f}
        };
        return M[grade];
}
bool course::operator<(const course& c) const {
        return (name < c.name);
}
bool course::operator==(const course& c) const {
        return (name == c.name);
}

/*
* Semester numbers:  Spring 2019: 20191; Fall 2019: 20192, etc.
Implement the following functions.
When adding a student, if the student is already in DB, then ignore the operation.
When adding a course, if the course is already in DB (even if it is in
a different semester), then ignore the operation.
All courses in a semester should be sorted according to name (increasing order)

When dropping a course, if the course does not exist, then ignore the operation.
When removing a student, if the student does not exist, then ignore the operation.
All courses in a semester need to be sorted.
When dropping or adding a course, overall GPA, semester GPA, overall credits and semester
credits all need to be updated.
If after drop_course, the list becomes empty, you don't need to remove the list.


*/
```

```
//Implement the following functions.
void add_student(map<int, pair < pair<int, float>*, map<int, tuple<int, float, list<course*>
>* >* >  >& DB, int id);
void remove_student(map<int, pair < pair<int, float>*, map<int, tuple<int, float,
list<course*> >* >* >  >& DB, int id);
void add_course(map<int, pair < pair<int, float>*, map<int, tuple<int, float, list<course*>
>* >* >  >& DB, int semester, int id, course c); //20171 Spring semester of 2017; 20172:
Fall semester of 2017
void drop_course(map<int, pair < pair<int, float>*, map<int, tuple<int, float, list<course*>
>* >* >  >& DB, int semester, int id, course c);
void print_student_semester_courses(map<int, pair < pair<int, float>*, map<int, tuple<int,
float, list<course*> >* >* >  >& DB, int semester, int id);
void print_student_all_courses(map<int, pair < pair<int, float>*, map<int, tuple<int, float,
list<course*> >* >* >  >& DB, int id);

//Implement additional functions such that you can do
//cout << DB << endl;


int main() {




        map<int, pair < pair<int, float> *, map<int, tuple<int, float, list<course*> > * > *
>  >  DB;

        add_student(DB, 11111);
        course C1("CIS554", 1, 3, "A-"), C2("CSE674", 1, 3, "B+"), C3("MAT296", 8, 4, "A"),
C4("WRT205", 5, 3, "A");

        add_course(DB, 20171, 11111, C1);
        add_course(DB, 20171, 11111, C4);
        add_course(DB, 20171, 11111, C3);
        add_course(DB, 20171, 11111, C2);
        print_student_semester_courses(DB, 20171, 11111);

        drop_course(DB, 20171, 11111, C1);
        print_student_semester_courses(DB, 20171, 11111); //sorted according to course name

        course C5("CIS351", 2, 3, "A-"), C6("PSY205", 5, 3, "B+"), C7("MAT331", 2, 3, "A"),
C8("ECN203", 4, 3, "A");
        add_course(DB, 20172, 11111, C5);
        add_course(DB, 20172, 11111, C6);
        add_course(DB, 20172, 11111, C7);
        add_course(DB, 20172, 11111, C8);
        add_course(DB, 20172, 11111, C3);
        print_student_all_courses(DB, 11111);//ID GPA

        add_student(DB, 11112);
        add_course(DB, 20171, 11112, C2);
        add_course(DB, 20171, 11112, C5);
        add_course(DB, 20171, 11112, C7);
        add_course(DB, 20171, 11112, C4);
        print_student_semester_courses(DB, 20171, 11112);

        add_course(DB, 20172, 11112, C8);
        add_course(DB, 20172, 11112, C3);
        add_course(DB, 20172, 11112, C5);
        add_course(DB, 20172, 11112, C1);
        print_student_semester_courses(DB, 20172, 11112);

        print_student_all_courses(DB, 11112);

        //Overload operator<< to allow the following cout statements.
```

```
        cout << DB << endl;
        remove_student(DB, 11111);
        cout << DB << endl;
        return 0;
}
```