Project 1-Score Processing

Date: 2021-04-01

1. Introduction

This C++ program will use the given information(inputted using keyboard)of students/courses and their relation to calculate some critical statistic data then display a general report(table) in *.csv format on screen including:

- For all existing(selected by any student) courses:
  - The name of each course
  - The average score of each course
  - An average score of the average score of each course
- For all students inputted into the system:
  - The name of each student
  - The ID of each student
  - The courses this student got in each course he selected
  - The average course of this student(score 0 is also included)

Input:

- There are two types of input format:
  - Student ID+Student Name,like `3190101234, Zhang San`
  - Student ID+Course Name+Student's Score on This Course,like `3190101111, Linear Algebra, 89.5`
- These two types of format will appear without certain order, our program will do the reconstruction of students and courses information
- Comma is used to separate each key,and after the comma is always a space

Output:

- In the first line,give the title of the table,like `student id, name, Computer Architecture, Linear Algebra, Operating System, average`
- In the following lines,give each student's ID,name and grades(if any),and an average score,like `3180111435, Jessie Zhao, , , 34.5, 34.5`
- In the last line, if there are any course,put each course's average score,and an average score of all courses' average scores,like `, , 46.5, 70.8, 34.5, 50.6`
- Comma is used to separate each key,and after the comma is always a space
- All floating numbers keep one number after the floating point

2. Data Structure and Algorithm Specification

In my program, there are a main function and three global functions:

- int main(void);
  - Responsible for standard input and output
  - Call global function to process the formatted input string to get enough information

    - Call the member function `insert_student_mark()` and `insert_student_info()` of `data` instance(created with `Database` class) to finish database's construction
    - Call the member function of `data` instance to finish the output task
- int judge_input_type(string s);
    - Separate the two types of input by counting commas in the string
    - If the input is student's basic information(ID+Name),then return 0
    - If the input is student's course information(ID+Course Name+Score),then return 1
- string slice_by_comma(string s,int key_selection);
    - Slice each key in the string separated by comma
    - Return the `key_selection`'s key in the input string
- double string_to_double(string s);
    - Use `istringstream` and the overload of operator `>>` to finish the conversion from string to double
- And there are three classes with a lot of member functions:
    - Class `Student` contains a student's name,ID and a map(dictionary) of his courses and the corresponding courses' scores:

```
class Student{
private:
    string stu_name;
    string stu_id;
    map<string,double> stu_courses;
public:
    //student's name may be not given in an input
    Student(string stu_id,string stu_name="");
    void set_name(string stu_name); //set the name when given
    void add_course(string course_name,double course_mark);
    string get_id(void);
    string get_name(void);
    double get_mark(string course);
    double get_average(void);
};
```

    - Class `Course` contains a course's name,and it also stores the course's selected times and total marks gained by students for the calculation of it average score:

```
class Course{
private:
    string course_name;
    int selected_times;
    double total_mark;
public:
    Course(string course_name);
    //execute when a new student selects this course
    void add_mark(double single_mark);
    double get_average(void);
};
```

    - Class `Database` links all students and courses,maintaining the relations between student's name and his instance(with his detailed information) as well as course's name and its instance. Member function `print_csv()` is responsible for the formatted output:

```
class Database{
private:
    map<string,Student*> stu_relation;
    map<string,Course*> course_relation;
public:
    void insert_student_info(string student_id,string student_name);
    void insert_student_mark(string student_id,string course_name,double
course_mark);
    double get_courses_average(void);
    void print_csv(void);
};
```

3. Testing Results

Test Cases	Design Purpose	Result	Status
3180111435, Operating System, 34.5 3180111430, Linear Algebra, 80 3180111435, Jessie Zhao 3180111430, Zhiwen Yang 3180111430, Computer Architecture, 46.5 3180111434, Linear Algebra, 61.5 3180111434, Anna Teng	A trivial case	student id, name, Computer Architecture, Linear Algebra, Operating System, average 3180111430, Zhiwen Yang, 46.5, 80.0, , 63.2 3180111434, Anna Teng, , 61.5, , 61.5 3180111435, Jessie Zhao, , , 34.5, 34.5 , , 46.5, 70.8, 34.5, 50.6	Pass
3180111435, Jessie Zhao 3180111430, Zhiwen Yang 3180111434, Anna Teng	A case with no course	student id, name, average 3180111430, Zhiwen Yang, 3180111434, Anna Teng, 3180111435, Jessie Zhao,	Pass
3180111435, Operating System, 34.5 3180111430, Linear Algebra, 80 3180111430, Computer Architecture, 46.5 3180111434, Linear Algebra, 61.5	A case with no student's detailed information	student id, name, Computer Architecture, Linear Algebra, Operating System, average 3180111430, , 46.5, 80.0, , 63.2 3180111434, , , 61.5, , 61.5 3180111435, , , , 34.5, 34.5 , , 46.5, 70.8, 34.5, 50.6	Pass
3180111435, Operating System, 0 3180111430, Linear Algebra, 80 3180111435, Jessie Zhao 3180111430, Zhiwen Yang 3180111430, Computer Architecture, 46.5 3180111434, Linear Algebra, 61.5 3180111434, Anna Teng	A case with zero score	student id, name, Computer Architecture, Linear Algebra, Operating System, average 3180111430, Zhiwen Yang, 46.5, 80.0, , 63.2 3180111434, Anna Teng, , 61.5, , 61.5 3180111435, Jessie Zhao, , , 0.0, 0.0 , , 46.5, 70.8, 0.0, 39.1	Pass
输入测试用例...	An empty case	student id, name, average	Pass

Appendix: Source Code (in C++)

```
#include <iostream>
#include <sstream>
#include <iomanip>
#include <string>
#include <map>
using namespace std;

class Student{
private:
    string stu_name;
```

```
    string stu_id;
    map<string,double> stu_courses;
public:
    Student(string stu_id,string stu_name="");
    void set_name(string stu_name);
    void add_course(string course_name,double course_mark);
    string get_id(void);
    string get_name(void);
    double get_mark(string course);
    double get_average(void);
};

class Course{
private:
    string course_name;
    int selected_times;
    double total_mark;
public:
    Course(string course_name);
    void add_mark(double single_mark);
    double get_average(void);
};

class Database{
private:
    map<string,Student*> stu_relation;
    map<string,Course*> course_relation;
public:
    void insert_student_info(string student_id,string student_name);
    void insert_student_mark(string student_id,string course_name,double
course_mark);
    double get_courses_average(void);
    void print_csv(void);
};

int judge_input_type(string s);
string slice_by_comma(string s,int key_selection);
double string_to_double(string s);

int main(void){
    Database data;
    string temp_str;
    while(getline(cin,temp_str)){
        if(judge_input_type(temp_str))

 data.insert_student_mark(slice_by_comma(temp_str,0),slice_by_comma(temp_str,1),
string_to_double(slice_by_comma(temp_str,2)));
        else
data.insert_student_info(slice_by_comma(temp_str,0),slice_by_comma(temp_str,1));
    }
    data.print_csv();
    return 0;
}

int judge_input_type(string s){
    int comma_count=0;
    while(s.find_first_of(",")!=string::npos){
        comma_count++;
```

```
        s=s.substr(s.find_first_of(",")+1);
    }
    if(comma_count==1) return 0;
    else return 1;
}

string slice_by_comma(string s,int key_selection){
    int comma_count=0;
    string result="";
    while(s.length()>0){
        if(comma_count==(key_selection+1)) break;
        else if(s[0]==','){
            s.erase(0,2);
            comma_count++;
        }else if(comma_count==key_selection){
            result+=s[0];
            s.erase(0,1);
        }else s.erase(0,1);
    }
    return result;
}

double string_to_double(string s){
    double result;
    istringstream temp_stream(s);
    temp_stream>>result;
    return result;
}

Student::Student(string stu_id,string stu_name){
    this->stu_name=stu_name;
    this->stu_id=stu_id;
    return;
}

void Student::set_name(string stu_name){
    this->stu_name=stu_name;
    return;
}

void Student::add_course(string course_name,double course_mark){
    stu_courses[course_name]=course_mark;
    return;
}

string Student::get_id(void){
    return stu_id;
}
string Student::get_name(void){
    return stu_name;
}

double Student::get_mark(string course){
    if(stu_courses.find(course)!=stu_courses.end())
        return stu_courses[course];
    else return -1;
}
```

```
double Student::get_average(void){
    map<string,double>::iterator p;
    double marks_sum=0;
    int courses_count=0;
    for(p=stu_courses.begin();p!=stu_courses.end();p++){
        marks_sum+=p->second;
        courses_count++;
    }
    if(courses_count==0) return -1;
    else return marks_sum/courses_count;
}

Course::Course(string course_name){
    this->course_name=course_name;
    this->selected_times=0;
    this->total_mark=0;
    return;
}

void Course::add_mark(double single_mark){
    this->selected_times++;
    this->total_mark+=single_mark;
    return;
}

double Course::get_average(void){
    if(selected_times==0) return -1;
    else return total_mark/selected_times;
}

void Database::insert_student_info(string student_id,string student_name){
    if(stu_relation.find(student_id)!=stu_relation.end()){
        stu_relation[student_id]->set_name(student_name);
        return;
    }else{
        stu_relation[student_id]=new Student(student_id,student_name);
        return;
    }
}

void Database::insert_student_mark(string student_id,string course_name,double
course_mark){
    if(stu_relation.find(student_id)!=stu_relation.end())
        stu_relation[student_id]->add_course(course_name,course_mark);
    else{
        stu_relation[student_id]=new Student(student_id);
        stu_relation[student_id]->add_course(course_name,course_mark);
    }
    if(course_relation.find(course_name)!=course_relation.end()){
        course_relation[course_name]->add_mark(course_mark);
        return;
    }else{
        course_relation[course_name]=new Course(course_name);
        course_relation[course_name]->add_mark(course_mark);
        return;
    }
}
```

```
double Database::get_courses_average(void){
    map<string,Course*>::iterator p;
    int courses_count=0;
    double courses_average_sum=0;
    for(p=course_relation.begin();p!=course_relation.end();p++){
        courses_average_sum+=p->second->get_average();
        courses_count++;
    }
    if(courses_count==0) return -1;
    else return courses_average_sum/courses_count;
}

void Database::print_csv(void){
    map<string,Course*>::iterator p_course;
    map<string,Student*>::iterator p_student;
    cout<<"student id, name";

 for(p_course=course_relation.begin();p_course!=course_relation.end();p_course++
)
        cout<<", "<<p_course->first;
    cout<<", average"<<endl;

 for(p_student=stu_relation.begin();p_student!=stu_relation.end();p_student++){
        cout<<p_student->second->get_id()<<", "<<p_student->second->get_name();

 for(p_course=course_relation.begin();p_course!=course_relation.end();p_course++
){
            if(p_student->second->get_mark(p_course->first)!=-1)
                cout<<fixed<<setprecision(1)<<", "<<p_student->second-
>get_mark(p_course->first);
            else cout<<", ";
        }
        if(p_student->second->get_average()==-1) cout<<", "<<endl;
        else cout<<fixed<<setprecision(1)<<", "<<p_student->second-
>get_average()<<endl;
    }
    if(course_relation.begin()!=course_relation.end()){
        cout<<", ";

 for(p_course=course_relation.begin();p_course!=course_relation.end();p_course++
)
            cout<<", "<<p_course->second->get_average()<<fixed<<setprecision(1);
        if(get_courses_average()==-1) cout<<", "<<endl;
        else cout<<fixed<<setprecision(1)<<", "<<get_courses_average()<<endl;
    }
}
```

Declaration

I hereby declare that all the work done in this project is of my independent effort.