CS 201, Spring 2018

Homework Assignment 1

Due: 23:59, April 5 (Thursday), 2018

In this homework, you will implement a student-review-system to keep the grades of multiple students. For each student you are going to store a record. Each student has a unique student id, name, and a list of courses. In your implementation, you **MUST** use dynamically allocated arrays.

This homework will have two parts, whose requirements are explained below.

PART A:

To take the final exam, you MUST submit at least this part and MUST get at least half of its points.

This part is a simplified version of the entire system, which keeps just the id and the name of each student without keeping his/her courses. In this system, you must keep the students in a dynamically allocated array of the `Student` objects. Thus, you must implement the `Student` class first. This class is quite simple for Part A, but you will have to extend it for Part B.

1. Below is the required part of the `Student` class. The name of the class must be `Student`. The interface for the class must be written in a file called `SimpleStudent.h` and its implementation must be written in a file called `SimpleStudent.cpp`.

   - The `Student` class keeps the id and the name of a single student as data members. Make sure to implement the get functions for these data members since we will use them to test your program.
   - Implement the default constructor, which initializes the `studentId` and `studentName` data members. Additionally, implement your own destructor and copy constructor, and overload the assignment operator. Although you may use the default ones for some of these special functions, you are advised to implement them (some may have no statements) so that it will be easier for you to extend them for Part B.
   - Do not delete or modify any part of the given data members or member functions. However, you may define additional functions and data members, if necessary.

```
#ifndef __SIMPLE_STUDENT_H
#define __SIMPLE_STUDENT_H

#include <string>
using namespace std;

class Student {
public:
    Student(const int sid = 0, const string sname = "");
    ~Student();
    Student(const Student &studentToCopy);
    void operator=(const Student &right);

    int getStudentId();
    string getStudentName();

private:
    int studentId;
    string studentName;
};
#endif
```

2. Below is the required part of the SRS class that you must write in Part A of this assignment. The name of the class must be SRS. The interface for the class must be written in a file called SimpleSRS.h and its implementation must be written in a file called SimpleSRS.cpp.

   - Implement the default constructor, which creates an empty student-review-system. Also overload the assignment operator and implement the destructor and copy constructor.
   - Implement the add and remove student functions whose details are given below:

     Add a student: This function adds a student to the system. The id of the student and his/her name are specified as parameters. In this system, student ids are unique (however, students can have the same name). Thus, if the user attempts to add a student with an already existing id, do not add the student and return false. Otherwise, if the student does not exist in the system, add him/her to the system and return true. DO NOT display any warning messages.

     Remove a student: This function removes a student from the system. The id of this student is specified as a parameter. If the student with the given id exists in the system, remove him/her from the system and return true. Otherwise, if there is no student with the given id, do not perform any action and return false. Likewise, DO NOT display any warning messages.

   - Implement the get function for the students. This getStudents function should return the number of the students in the system using the return value and should return the students in the system using the parameter list (using a pass-by-reference parameter called allStudents). Do not forget to put a deep copy of the students to the pass-by-reference parameter. Otherwise, you may encounter run-time errors.
   - Do not delete or modify any part of the given data members or member functions. However, you may define additional functions and data members, if necessary.

```
#ifndef __SIMPLE_SRS_H
#define __SIMPLE_SRS_H

#include <string>
using namespace std;
#include "SimpleStudent.h"

class SRS{
public:
    SRS();
    ~SRS();
    SRS(const SRS &srsToCopy);
    void operator=(const SRS &right);

    bool addStudent(const int studentId, const string studentName);
    bool removeStudent(const int studentId);
    int getStudents(Student *&allStudents);

private:
    Student *students;
    int studentNo;
};
#endif
```

3. To test Part A, write your own main function in a separate file. Do not forget to test your code for different cases. However, do not submit any file containing the main function. Otherwise, you may lose a considerable amount of points.

   We will write our own test codes to grade Part A of your assignment. In this test code, we will use the get functions of the Student and SRS classes. Thus do not forget to implement them.

Below is an example of the test code (we will also use different ones) and its output. In this example code, there is a global function called `displayAllStudents`. We will use it to display the students of the system, thus, to understand whether you added/removed students correctly.

If you want, you may use this global function for your tests as well. Notice that the last two lines of this global function are to deallocate the `allStudents` array. Do not remove these two lines if you get any run-time errors. If you have such run-time errors, most probably, it means you made an error implementing either the `getStudents` function or destructor/copy constructor/assignment operator.

```
#include <iostream>

#include "SimpleSRS.h"
#include "SimpleStudent.h"

using namespace std;

void displayAllStudents(SRS S){
    Student *allStudents;
    int studentNo = S.getStudents(allStudents);

    cout <<"No of students: "<< studentNo << endl;
    for (int i = 0; i < studentNo; i++){
        cout << allStudents[i].getStudentName() <<" (";
        cout << allStudents[i].getStudentId() <<")"<< endl;
    }
    if (allStudents != NULL)
        delete [] allStudents;
}
int main(){
    SRS S;

    S.addStudent(1234, "Cigdem Gunduz");
    S.addStudent(5678, "Ercument Cicek");
    if (S.addStudent(7890, "Cigdem Gunduz"))
         cout <<"Successful insertion for Cigdem Gunduz (7890)"<< endl;
    else cout <<"Unsuccessful insertion for Cigdem Gunduz (7890)"<< endl;

    if (S.addStudent(7890, "Serhan Yilmaz"))
         cout <<"Successful insertion for Serhan Yilmaz (7890)"<< endl;
    else cout <<"Unsuccessful insertion for Serhan Yilmaz (7890)"<< endl;

    if (S.removeStudent(5000))
         cout <<"Successful deletion for Student 5000"<< endl;
    else cout <<"Unuccessful deletion for Student 5000"<< endl;

    for (int i = 0; i < 1000; i++)
        S.addStudent(i, "Gozde Gunesli");

    for (int i = 1500; i > 1; i--)
        S.removeStudent(i);

    displayAllStudents(S);
    return 0;
}
```

The output should be

```
Successful insertion for Cigdem Gunduz (7890)
Unsuccessful insertion for Serhan Yilmaz (7890)
Unsuccessful deletion for Student 5000
No of students: 4
Ercument Cicek (5678)
Cigdem Gunduz (7890)
Gozde Gunesli (0)
Gozde Gunesli (1)
```

What to submit for Part A?

You should put your `SimpleStudent.h`, `SimpleStudent.cpp`, `SimpleSRS.h`, and `SimpleSRS.cpp` files into a folder and zip the folder. In this zip file, there should not be any file containing the main function. The name of this zip file should be: PartA_secX_Firstname_Lastname_StudentID.zip where X is your section number. Then follow the steps that will be explained at the end of this document for the submission of Part A.

What to be careful about implementation and submission for Part A?

You have to read "notes about implementation" and "notes about submission" parts that will be given at the end of this document.

PART B:

Now extend Part A such that each student will have a list of courses. Provide the full functionality of this extended student-review-system. In order to do that, first extend the `Student` class such that it additionally keeps the courses that a student is registered to. These courses must be kept in a dynamically allocated array of the `Course` objects. Note that the number of courses can be different from one student to another and it is not known in advance. The details of the classes are given below.

1. The requirements of the `Course` class are given below. The name of the class must be `Course`. The interface for the class must be written in a file called `Course.h` and its implementation must be written in a file called `Course.cpp`.

   - The `Course` class keeps the id of a course, its credit, and the letter grade that the student took. The course id should be unique for each student. That is, a student cannot take multiple courses with the same course id but different students may take the same course with the same course id. For different students, the credit can be different for the same course. Your system will use the following letter grades that are available in the grading system of Bilkent University: A+, A, A-, B+, B, B-, C+, C, C-, D+, D, F, FX, FZ, and W (it will not use the other letter grades of Bilkent University).
   - Implement your own default constructor, destructor, and copy constructor, and overload the assignment operator.
   - Do not delete or modify any part of the given data members or member functions. However, you may define additional functions and data members, if necessary.

```cpp
#ifndef __COURSE_H
#define __COURSE_H

#include <string>
using namespace std;

class Course{
public:
    Course();
    ~Course();
    Course(const Course &courseToCopy);
    void operator=(const Course &right);

private:
    int courseId;
    int courseCredit;
    string courseGrade;
};
#endif
```

2. The requirements of the `Student` class are given below. The name of the class must be `Student`. This time, the interface for the class must be written in a file called `Student.h` and its implementation must be written in a file called `Student.cpp`.

- The Student class is similar to the one given in Part A. However, now it keeps a dynamic array of Course objects as well. Similar to Part A, implement your own default constructor, destructor and copy constructor, and do not forget to overload the assignment operator. Also, make sure to implement the get functions for the studentId and studentName data members (we will use them for testing).
- Your class must have a public member function called calculateStudentCurrentGPA (we will also use it for testing). This function calculates the current GPA of the student. Make sure that this function returns the most recent GPA of the student since the courses that a student takes may change (since it is possible to add or remove courses).

  Use the grading system of Bilkent University for calculating the GPAs. The grade point equivalents of the letter grades are below. Note that the grade W does not have any grade point equivalent and should not be considered in the GPA calculation.

A+	4.0	B+	3.3	C+	2.3	D+	1.3	F	0.0
A	4.0	B	3.0	C	2.0	D	1.0	FX	0.0
A-	3.7	B-	2.7	C-	1.7			FZ	0.0

  Note 1: This function should return the **weighted** average of the courses that the student takes. For example, a student that takes a 3-credit course with A and a 4-credit course with C, has a GPA of 2.86 (not 3.00).

  Note 2: This function does not include the letter grade W in the calculation. For example, a student that has three 4-credit courses with A+, B and W grades has a GPA of 3.5 (not 2.33).
- Do not delete or modify any part of the given data members or member functions. However, you may define additional functions and data members, if necessary.

```
#ifndef __STUDENT_H
#define __STUDENT_H

#include <string>
using namespace std;
#include "Course.h"

class Student{
public:
    Student(const int sid = 0, const string sname = "");
    ~Student();
    Student(const Student &studentToCopy);
    void operator=(const Student &right);

    int getStudentId();
    string getStudentName();
    double calculateStudentCurrentGPA();

private:
    int studentId;
    string studentName;
    Course *courses;
    int courseNo;
};
#endif
```

3. Below is the required part of the SRS class. The name of the class must be SRS. This time, the interface for the class must be written in a file called SRS.h and its implementation must be written in a file called SRS.cpp.
   - The SRS class is similar to the one given in Part A. However, now it also keeps the list of courses for each student. Similar to Part A, implement your own default constructor, destructor and copy constructor, and overload the assignment operator. Also make sure to implement the getStudents function, as explained before (we will use it for testing).

- Implement the following functions that your extended system should support.

  Add a student: This function adds a student to the system. The id of the student and his/her name are specified as parameters. In this function, the course list is not specified; the course(s) will be added later. In this system, student ids are unique (however, students can have the same name). Thus, if the user attempts to add a student with an already existing id, do not add the student and return false. Otherwise, if the student does not exist in the system, add him/her to the system and return true. DO NOT display any warning messages. (This function is very similar to what you will implement in Part A. But in Part B, you will need to create an empty course list for the student when you add him/her to the system.)

  Remove a student: This function removes a student from the system. The id of the student is specified as a parameter. If the student with the given id exists in the system, remove him/her from the system and return true. Otherwise, if there is no student with the given id, do not perform any action and return false. Likewise, DO NOT display any warning messages. Note that this function also clears the course list of the specified student. (This function is very similar to what you will implement in Part A. But now, for Part B, you will need to remove his/her course list when you remove the student from the system.)

  Add a course to the student: This function adds a course to the course list of a student. The id of the student as well as the course id, the course credit, and the letter grade of this student are specified as parameters. In this function, you should take care of the following issues:
  - If the student with the specified id does not exist in the system, or if the course with the specified id exists in the course list of the specified student, or if the letter grade is invalid, do not perform any action and return false. DO NOT display any warning messages.
  - All course ids are unique for a student. Thus, if the user attempts to add a course with an existing course id for the specified student, do not perform any action and return false. DO NOT display any warning messages. (However, different students may take the same course.)
  - Otherwise, add the course to the course list of the specified student and return true.

  Remove a course from the student: This function removes a course from the course list of a student. The student id and the course id are given as parameters. If there is no student with the specified student id or if there are not any courses with the specified course id in the course list of the specified student, do not perform any action and return false. DO NOT display any warning messages. Otherwise, remove the specified course from the course list of the specified student and return true.

  Calculate the course GPA: This function calculates the GPA of the course whose id is given as a parameter. It excludes W, FX, and FZ grades while calculating the average. That is, it should calculate the average grade over all students who took this course and got a letter grade except W, FX, and FZ. For example, the average GPA of a course that has 4 students with B, F, W and FZ grades, is 1.50 since both W and FZ grades are not included in the calculations whereas B and F grades are included. If there are not any students that took the specified course, or if all students that took this course got W, FX, or FZ grades, this function should return 0. Calculate the average course GPA using the grade point equivalents given above.

- Do not delete or modify any part of the given data members or member functions. However, you may define additional functions and data members, if necessary.

```cpp
#ifndef __SRS_H
#define __SRS_H

#include <string>
using namespace std;
#include "Student.h"

class SRS{
public:
    SRS();
    ~SRS();
    SRS(const SRS &srsToCopy);
    void operator=(const SRS &right);
    int getStudents(Student *&allStudents);
    bool addStudent(const int studentId, const string studentName);
    bool removeStudent(const int studentId);
    bool addCourse(const int studentId, const int courseId,
                   const string courseGrade, const int courseCredit);
    bool removeCourse(const int studentId, const int courseId);
    double calculateCourseGPA(const int courseId);

private:
    Student *students;
    int studentNo;
};
#endif
```

4. To test Part B, write your own main function in a separate file. Do not forget to test your code for different cases. However, do not submit this file or any file containing the main function. Otherwise, you may lose a considerable amount of points.

   Likewise, we will write our own test codes to grade Part B of your assignment. In this test code, we will use the get functions of the `Student` and `SRS` classes. Thus do not forget to implement them.

   Below is an example of the test code (we will also use different ones) and its output. In this example code, there are two global functions, `displayAllStudents` and `displayStatistics`. The first one is the same with the one implemented for Part A. The purpose of the second one is to get some aggregate statistics about the grades of the students. We will use them to understand whether you added/removed students and courses correctly.

   If you want, you may use these global functions for your tests. Again the last two lines are included to deallocate the `allStudents` array. Do not remove these two lines if you get any run-time errors while using these functions. If you have such run-time errors, most probably, you have an error regarding the implementation of either your `getStudents` function or destructor/copy constructor/assignment operator.

```
#include <iostream>
using namespace std;
#include "SRS.h"
#include "Student.h"

void displayAllStudents(SRS S){
    Student *allStudents;
    int studentNo = S.getStudents(allStudents);
    cout <<"No of students: "<< studentNo << endl;
    for (int i = 0; i < studentNo; i++){
        cout << allStudents[i].getStudentName() <<" (";
        cout << allStudents[i].getStudentId() <<")"<< endl;
    }
    if (allStudents != NULL)
        delete [] allStudents;
}
void displayStatistics(SRS S){
    Student *allStudents;
    int studentNo = S.getStudents(allStudents);
    int noUnsatisfactory = 0;
    int noProbation = 0;
    int noSatisfactory = 0;
    int noHonor = 0;
    int noHighHonor = 0;

    for (int i = 0; i < studentNo; i++){
        double gpa = allStudents[i].calculateStudentCurrentGPA();
        if (gpa < 1.80)
            noUnsatisfactory++;
        else if (gpa < 2.00)
            noProbation++;
        else {
            noSatisfactory++;
            if (gpa >= 3.00 && gpa < 3.50)
                noHonor++;
            if (gpa >= 3.50)
                noHighHonor++;
        }
    }
    cout <<"Number of high honor: "<< noHighHonor << endl;
    cout <<"Number of honor: "<< noHonor << endl;
    cout <<"Number of satisfactory: "<< noSatisfactory << endl;
    cout <<"Number of probation: "<< noProbation << endl;
    cout <<"Number of unsatisfactory: "<< noUnsatisfactory << endl;
    if (allStudents != NULL)
        delete [] allStudents;
}
int main(){
    SRS S;
    S.addStudent(1234, "Cigdem Gunduz");
    S.addStudent(5678, "Ercument Cicek");
    S.addStudent(7890, "Cigdem Gunduz");
    S.addStudent(7890, "Serhan Yilmaz");

    for (int i = 0; i < 1000; i++){
        S.addStudent(i, "Gozde Gunesli");
        S.addCourse(i, 201, "B", 4);
        S.addCourse(i, 101, "C", 3);
        S.addCourse(i, 102, "A+", 4);
        S.addCourse(i, 103, "F", 3);
        S.addCourse(i, 204, "D", 2);
        S.addCourse(i, 503, "C", 3);
    }
    S.addCourse(1234, 201, "B", 4);
    S.addCourse(5678, 202, "B", 1);
    S.addCourse(5678, 201, "C-", 4);
    S.addCourse(7890, 410, "D+", 4);
    S.addCourse(7890, 201, "W", 4);

    S.addStudent(3782, "Onur Karakaslar");
    S.addCourse(3782, 201, "A", 2);
    S.addStudent(3782, "Cigdem Demir");
    S.addCourse(3782, 201, "F", 1);

    S.removeCourse(5678, 202);
    S.removeCourse(5678, 210);
    for (int i = 1500; i > 0; i--)
        S.removeStudent(i);

    displayAllStudents(S);
    displayStatistics(S);
    cout <<"Course GPA for 201 is "<< S.calculateCourseGPA (201) << endl;
    return 0;
}
```

The output should be

```
No of students: 4
Ercument Cicek (5678)
Cigdem Gunduz (7890)
Gozde Gunesli (0)
Onur Karakaslar (3782)
Number of high honor: 1
Number of honor: 0
Number of satisfactory: 2
Number of probation: 0
Number of unsatisfactory: 2
Course GPA for 201 is 2.9
```

What to submit for Part B?

You should put your `Student.h`, `Student.cpp`, `Course.h`, `Course.cpp`, `SRS.h`, and `SRS.cpp` into a folder and zip the folder. In this zip file, there should not be any file containing the main function. The name of this zip file should be: PartB_secX_Firstname_Lastname_StudentID.zip where X is your section number. Then follow the steps explained at the end of this document for the submission of Part B.

What to be careful about implementation and submission for Part B?

You have to read "notes about implementation" and "notes about submission" parts that are given just below.

NOTES ABOUT IMPLEMENTATION (for both Part A and Part B):

1. You MUST use dynamically allocated arrays. You will get no points if you use fixed-sized arrays, linked-lists or any other data structures such as `vector/array` from the standard library.
2. You ARE NOT ALLOWED using any global variables or any global functions to implement the classes. However, you may use global functions to test your program (however, do not submit them).
3. Your code must not have any memory leaks. You will lose points if you have memory leaks in your program even though the outputs of the operations are correct.

NOTES ABOUT SUBMISSION (for both Part A and Part B):

1. Conform to the rules given separately for Part A and Part B. That is, the name of the classes, the name of the .h and .cpp files, and the name of the zip files should conform to the specifications separately given for Part A and Part B. Otherwise, you may lose a considerable amount of points.
2. **Before 23:59 on April 5, you need to send an email with a subject line CS 201 HW1 to Serhan Yılmaz, by attaching two zip files (one for Part A and the other for Part B). Read "what to submit for Part A" and "what to submit for Part B" sections very carefully.**
3. No hardcopy submission is needed. The standard rules about late homework submissions apply.
4. Do not submit any files containing the main function.
5. You are free to write your programs in any environment (you may use either Linux or Windows). On the other hand, we will test your programs on "`dijkstra.ug.bcc.bilkent.edu.tr`" and we will expect your programs to compile and run on the `dijkstra` machine. If we could not get your program properly work on the `dijkstra` machine, you would lose a considerable amount of points. Therefore, we recommend you to make sure that your program compiles and properly works on "`dijkstra.ug.bcc.bilkent.edu.tr`" before submitting your assignment.
6. This assignment will be graded by your TA Serhan Yılmaz (serhan.yilmaz at bilkent edu tr). Thus, you may ask your homework related questions directly to him.

VERY IMPORTANT: We expect all of you to comply with academic integrity. The honor code statement, which has been sent you by email, was prepared to clarify our expectations about the academic integrity principles. Please study the part of this statement related with "individual assignments" very carefully.

If you submit this homework assignment, we will assume that you all read this honor code statement and comprehensively understood its details. Thus, please make sure that you understood it. If you have any questions (any confusions) about the honor code statement, consult with the course instructors.

We will check your codes for plagiarism.