```
/*
	// (1) THERE SHOULD BE A SPACE AFTER THE COMMA.
	Wayne,Bruce

	// (2) CLASS WAS 150, BUT 250 OK.
	CS A200
	November 30, 2018

	Lab: Course Class
*/

#ifndef COURSE_H
#define COURSE_H

#include <iostream>
#include <string>
#include <set>
#include <algorithm>

using namespace std;

class Course
{
	// (3) OSTREAM OBJECT SHOULD BE RETURNED BY REFERENCE.
	// (4) COURSE PARAMETER SHOULD BE PASSED AS CONST.
	friend ostream operator<<(ostream&, Course&);

public:
	Course();
	// (5) UNNECESSARY TO PASS THE INT PARAMETER AS REFERENCE.
	Course(int&, const string&);
	Course(const Course&);

	// (6) MUST RETURN A REFERENCE TO THE OBJECT, NOT A POINTER.
	Course* operator=(const Course&);

	// (7) FUNCTION SHOULD BE CONST.
	bool operator==(const Course&);

	int getCourseNumber() const;
	string getCourseName() const;
	int getHighestScore() const;

	// (8) UNNECESSARY TO PASS AN INT AS CONST.
	void setCourseNumber(const int);
	void setCourseName(const string&);

	void addScore(int);

	// (9) INCONSISTENT: OTHER FUNCTIONS DO NOT HAVE IDENTIFIERS
	//     FOR PARAMETERS.
	// (10) THE INT PARAMETER IDENTIFIER DIFFERS FROM THE ONE
	//      IN THE FUNCTION DEFINITION.
	void addScores(const int a[], int score);

	// (11) FUNCTION SHOULD BE CONST.
	double getAverage();
```

```
        bool isEmpty() const;

        void deleteScores();
        ~Course();

private:
        int courseNumber;
        string courseName;
        int totalScores;
        multiset<int> *scores;


        // (12) TOO MUCH WHITE SPACE.
};

#endif


#include "Course.h"

// (13) UNNECESSARY TO INCLUDE algorithm.
//      IT IS ALREADY INCLUDED IN Course.h
#include <algorithm>

ostream operator<<(ostream& out, Course& course)
{
        if (course.totalScores == 0)
                cerr << "There are no scores in the database.";
        else
        {
                // (14) NEED TO USE OUT INSTEAD OF COUT.
                cout << "Scores: ";
                for (auto iter : *(course.scores))
                        out << iter << " ";
                return out;
        }
        // (15) NOT ALL PATHS RETURN A VALUE. "IF" BODY
        //      DOES NOT HAVE A RETURN.
}

Course::Course()
{
        courseNumber = 0;
        totalScores = 0;
        // (16) REDUNDANT TO INITIALIZE A STRING TO A NULL STRING.
        //      A STRING IS ALREADY INITIALIZED TO AN EMPTY STRING WHEN
        //      IS DECLARED.
        courseName = "";
        scores = new multiset<int>;
}

Course::Course(int& newNumber, const string& newName)
{
        courseNumber = newNumber;
        courseName = newName;
        totalScores = 0;
        scores = new multiset<int>;
```

```
}

Course::Course(const Course& otherCourse)
{
	courseNumber = otherCourse.courseNumber;
	courseName = otherCourse.courseName;
	totalScores = otherCourse.totalScores;
	scores = new multiset<int>;
	// (17) THIS STATEMENT COPIES THE ADDRESSES, MAKING
	//      A SHALLOW COPY OF THE PARAMETER OBJECT.
	//      CORRECT STATEMENT: *scores = *(otherCourse.scores);
	scores = otherCourse.scores;
}

int Course::getCourseNumber() const
{
	//(18) UNNECESSARY TO CREATE A VARIABLE.
	//      USE ONLY: return courseNumber;
	int courseNo = courseNumber;
	return courseNo;
}

Course* Course::operator=(const Course& otherCourse)
{
	// (19) INCORRECT. IT SHOULD BE:
	//      if(this == &otherCourse)
	if (*this == otherCourse)
		// (20) SHOULD BE cerr.
		cout << "Attempt to assign to self.";
	else
	{
		// (21) MUST DELETE SCORES FROM CALLING OBJECT FIRST.
		//         NEEDED STATEMENT:scores->clear();
		courseNumber = otherCourse.courseNumber;
		courseName = otherCourse.courseName;
		totalScores = otherCourse.totalScores;
		*scores = *(otherCourse.scores);
	}
	// (6a) MUST RETURN *this WHEN RETURNING A REFERENCE TO
	//      THE OBJECT.
	return this;
}

bool Course::operator== (const Course& otherCourse)
{
	return (courseNumber == otherCourse.courseNumber &&
		courseName == otherCourse.courseName &&
		totalScores == otherCourse.totalScores &&
		*scores == *otherCourse.scores);
}

string Course::getCourseName() const
{
	return courseName;
}// (22) THERE SHOULD BE AN EMPTY LINE IN BETWEEN FUNCTION DEFINITIONS.
int Course::getHighestScore() const
{
	// (23) UNNECESSARY TO CREATE ITERATORS.
```

```
	//		SIMPLY WRITE: return *(max_element(scores->begin(), scores->end()));
	// (24) UNNECESSARY TO CALL FUNCTION max_element,
	//		BECAUSE THE SET IS IN ORDER AND THE HIGHEST SCORE
	//		ELEMENT IS THE LAST ELEMENT.
	//		BETTER TO WRITE: return *(scores->rbegin());
	multiset<int>::iterator iter = scores->begin();
	multiset<int>::iterator iterEnd = scores->end();
	return *(max_element(iter, iterEnd));
}

void Course::setCourseNumber(const int newNumber)
{
	courseNumber = newNumber;
}

void Course::setCourseName(const string& newName)
{
	courseName = newName;
}

void Course::addScore(int newScore)
{
	scores->insert(newScore);
	// (25) MUST INCREMENT totalScores
	//		++totalScores;
}

void Course::addScores(const int a[], const int numOfElem)
{
	for (int i = 0; i < numOfElem; ++i)
	{
		scores->insert(a[i]);
		// (26) INEFFICIENT. SHOULD BE OUTSIDE LOOP:totalScores = numOfElem;
		++totalScores;
	}

	// totalScores = numOfElem;
}

double Course::getAverage()
{
	if (!scores->empty())
	{
		// (27) MUST ALWAYS INITIALIZE VARIABLES: int total = 0;
		int total;
		auto iter = scores->begin();
		// (28) DO NOT CALL A FUNCTION IN A LOOP WHEN IT RETURNS
		//		THE SAME VALUE AT EACH ITERATION.
		//		CORRECT CODE:
		//		auto iterEnd = scores->end();
		//		for (iter; iter != iterEnd; ++iter)
		for (iter; iter != scores->end(); ++iter)
			total += *iter;

		// (29) MUST CAST TO DOUBLE.
		//		return (static_cast<double>(total) / static_cast<double>(totalScores));
		return total / totalScores;
	}
```

```
	// (30) POOR READABILITY.
	//	return 0.0;
	return 0;
}

bool Course::isEmpty() const
{
	// (31) REDUNDANT CODE.
	//	SHOULD BE:   return (totalScores == 0);
	if (totalScores == 0)
		return true;
	else
		return false;
}

void Course::deleteScores()
{
	// (32) MUST DELETE DATA IN SET.
	//	scores->clear();
	totalScores = 0;
}

Course::~Course()
{
	// (33) THIS DOES NOT DELETE WHAT SCORES IS POINTING TO.
	//	INSTEAD OF scores->clear(), WRITE: delete scores;
	//	"DELETE" WILL CALL THE DESTRUCTOR OF THE STL CONTAINER
	//	TO WHICH SCORES IS POINTING.
	//	YOU SHOULD ALSO ADD: scores = nullptr;
	scores->clear();
}
```