Exercise 1: Classes w/ Dynamic Memory

Initial Due Date: Thursday, June 17, 11:55 pm (23:55)

Final Deadline: Sunday, June 20, 11:55 pm (23:55)

For this assignment, you are asked to implement a class `csci152_student` to represent students who are taking 152, which calculates their final grades based on assignment scores, quizzes, and the final exam. This class should work for different numbers of quizzes and assignments, using the grading rubric from the course syllabus.

The fields and interface for this class are provided in the header file **csci152_student.h** – you should not edit this file! Your job is to create a separate implementation file **csci152_student.cpp**, which provides bodies for the constructors, functions, and the destructor of the class following the descriptions (and hints) given in the header. To help you out, we have also provided a simple testing file main.cpp, but be sure to come up with more thorough tests… we will be using a more substantial testing file for final grading!

This assignment requires that you do some memory management – notice that there are two variable-length arrays of unsigned ints held onto by pointer fields. We cannot use fixed-length arrays for this, since we need to handle the different cases where we have a different number of quizzes and assignments. Because of this complication, you really should run Valgrind on your own testing class and `csci152_student` class implementation before submitting.

To make things a little easier, you don't have to worry about cases where a user might provide bad inputs, for example, scores that are out of the specified range. This approach is considered strict, old-school Design by Contract… if a user provides bad inputs, it is their own fault when things go bad!

You may also notice the use of the type `std::initializer_list` in the main constructor to take in the assignment and quiz scores as parameters. Initializer lists in C++ are somewhat like arrays, but they allow us to use convenient syntax such as `{75, 88, 92, 81, 100}` to directly provide a list of constants when we construct new objects. You can see this in the provided **main.cpp** when we want to simply list the assignment and quiz scores without declaring additional variables:

```
csci152_student agood("Agood Stutante",
                      {100, 100, 100, 100},
                      {20, 20, 20, 20, 20, 20},
                      100);
```

You will need to use and access the items in these initializer lists in the main constructor implementation. It will be sufficient to use the `ilist.size()` function to get the number of items in the initializer list, and to iterate through the items, you can use the following loop structure:

```
for (unsigned int item : ilist) {
    // code that does something with item
}
```

You could also use the keyword `auto` in place of the type `unsigned int` here if you are feeling ~~lazy~~ clever – the compiler can directly infer the item type from the iterator list.

Do the assignment yourself! Don't use other's code, whether it is from online or from another student. For this assignment, submit only the **csci152_student.cpp** file to Moodle by the deadline. The name of this file must be **csci152_student.cpp**, or it will not be recognized by our testing system, and it will get a score of 0.

A Note About Deadlines…

This summer, every assignment will have two deadlines: The initial deadline, and the final deadline. Whatever you submit by the initial deadline will be immediately scored through the autograder, and we will give you this preliminary score very soon after the deadline. If you are happy with this score, then there is nothing more to do… that will be your final score*. However, if you don't like your initial score, you will have a second opportunity to submit your work again before the final deadline. Whatever you have submitted by this final deadline will be your final score for the assignment.

The main reason why we use this "two phased" grading approach is that often, students make small mistakes in their submissions that prevent their code from compiling or properly running, which would normally result in a score of zero. With two phases, students have a chance to make sure things are working properly on our testing framework right after the first deadline and won't be immediately penalized if it isn't.

Note that although you are strongly encouraged to complete and submit your assignment solution by the first deadline, you are not required to do so – submitting by the second is sufficient for receiving a grade. However, if a student does not take advantage of this opportunity to submit their assignment by the first deadline, they cannot complain if some small mistake prevents their code from compiling and running.

For more details about our approach to programming assignments and grading, please refer to the "Programming Assignment General Instructions" document on the Programming Assignments page in WordPress.

Questions?

If you have questions about the assignment, please post your questions on Piazza. HOWEVER, never post code publicly! If you have problems that you feel requires that someone look at your code, post it "private to instructors" in Piazza.

Good luck!

---

* Unless it is later determined that you committed academic misconduct, in which case you will get a zero for the assignment, and probably worse.