Resources: Segmentation Faults

A segmentation fault in Unix-like operating systems is the same as a memory access error in Windows-like operating systems. The basic idea is that your program has tried to read/write data from/to a part of the computer's memory that the program has not been granted access to.

Debugging a segmentation fault involves identifying the statement(s) of the program which has this behavior. There are three main strategies that I use to identify the bad line. I'm listing them from easier to do and least powerful, to harder to do and most powerful.

1- "Print debugging". Put output statements in your code. The idea is to get an output statement that occurs immediately before the bad line, so you see the output when the program runs, and an output statement write after the bad line, so you don't see it when the program runs, because the segmentation fault stops execution. Putting an output statement between every pair of lines in the program is not desirable. Especially since you have to remove them (or comment them out) when debugging has finished.

I use a binary search strategy. Put an output statement at the beginning of the program, one at the end, and one in the middle. Build and run the program. After the segmentation fault occurs, observe the last statement in the output, and the first statement that isn't in the output. Put another output statement half-way between these statements. Repeat until only one line can be blamed. Now, fix that line. This usually involves adding some more information to the debugging output statement, such as the .size() of a vector, the value of the index being used, etc., to understand how and why the index is out of range.

You may not have access to the `main()` function in your program in the case of unit tests, and some other situations. In this case, try to identify the most likely functions or methods that could be causing the segmentation fault. Put a single output statement at the start of these suspects, so you can see when the functions are entered. Also put one right before the return for these functions. Now when you run the program. The likely culprit is the function that was entered, but never returned. Continue with binary search for the line in question.

2- "Memory checker". Use a program to analyze your program as it runs. This memory checking program can watch for questionable and invalid memory usage, reporting all cases through its output. For example, I use the `valgrind` program. You use this by running valgrind and telling it to run your program. If you're doing unit tests from CodeGrinder, try "make valgrind" instead of "make test". This will run the unit test program, using valgrind, and report valgrind's findings. Start with the first reported anomaly, understand it, and fix it. Then, build and run again. Repeat until valgrind doesn't report any anomalies. As with print debugging, you may add output statements immediately before the line that is producing the error to understand the relevant data values to fix the problem.

3- "Debugger". Use a debugger program to analyze your program as it runs. A debugger is a program that can run your program one step at a time, or more, depending on your ability to communicate your desires to the debugger. The debugger can show you the value of any variable as you go. No need to add debugging output statements, just to remove them later. The downside, is that you need to learn how to use the debugger correctly to get the information and behavior you need. CS2810 students often use the debugger for their assembly programs. It would be a waste not to use debugger skills on your C++ programs too.

As stated earlier, the reason for a segmentation fault is invalid memory access by your program. Early in learning C++, the most common way to do this is by using invalid indexes into a vector, string, array, or any other sequential data type with integer indexes. The desired debugging information is 1) the actual index value by displaying it (don't trust what you think it should be, actually observe it), and 2) the actual size of the object you are indexing into (again, don't trust what you think, find the actual value).

A segmentation fault may occur when calling class methods, if the object is not actually a valid object.

Once the problem is identified, it may be that code has to change in a different part of the program to make the data structure be correctly sized, or to make the index be correctly calculated.