Troubleshooting

These are things I expect you to do before you email me. I won't help you write your code or troubleshoot programming logic problems. You should be able to do that at this point. But C poses some tricky hard to find issues, so ask if you need to. The worst I can say is you should be able to find that yourself.

Make sure your program compiles and runs main0.cc **on student**. This is a minimum before I will help you. If it doesn't run figure out why. I can only hunt and guess if it doesn't run. And I don't see why you expect me to do that for you when you can do it just as easily. You must run it on student because it is more likely to show you issues with uninitialized variables that your own machines.

Make sure your program compiles and runs with main1.cc. If it has issues, you must fix those issues before continuing.

Make sure your program compiles and runs with main2.cc (for part 2). If it has issues, you must fix those issues before continuing.

Make your own test files. You can very easily make your own. You should test MORE than what I have given you in test1.asm and test2.asm. Make a file with no origin and test. Make a file with no end and test. Make a file that is a blank file. Make a file that is only comments and blank lines. Make a file with the origin on the very first line. Make a file with a blank line for the very first line. Make a file with a comment as the very first line. Put tabs on the same line before your origin. Put spaces before your origin. Put spaces and tabs before your origin.

Some techniques for troubleshooting:

When programming, go in small steps. DO NOT write the whole thing and then troubleshoot later. That isn't a good technique and will never work for bigger programs. The errors could be any where and it becomes a mammoth task to track them down. Do small pieces. Get them working. Troubleshoot any errors there and when something doesn't work you know exactly where to look. Add features slowly. Get the new feature working. Save your file. Then move onto the next. For example, have findOrigin simply print ***line*** to start with. Then create your remove spaces function. Pass it ***line*** and make sure it works. Test it thoroughly. Make sure it works so that you wont ever have to worry that remove spaces is the source of your trouble. Then move on to capitals.

Make a very simple ASM file for testing. With only an origin, an add, and a halt. For part 2 change the add to whatever you are currently testing.

ALWAYS INITIALIZE ALL VARIABLES. One of the main reasons your program runs fine for you but doesn't run on student is uninitialized variables. This causes unpredictable errors. Set string char arrays to empty strings by setting first char to '\0'.

ALWAYS RETURN VALUES. This one doesn't happen as often, but it crops up from time to time. C will not tell you if you forgot to return something or if you had a path through your code that didn't return something. Make sure ALL paths through if statements have a return. If returning out of a loop, make sure there is a return in case the loop exits without the return. The easiest way to never fail this is to ONLY have one return at the end.

Print things.

Print ***line*** inside of the find origin loop at the top before it gets modified.

Print ***line*** (or its equivalent if you used another array) just after your call to remove spaces to make sure it actually removed the spaces.

Print ***line*** (or its equivalent if you used another array) just after your call to make all characters upper or lower case to make sure it actually changed the case.

Print sentinels. That is print 1, 2, 3, 4, etc... at various places throughout your program inside of loops. Inside of if statements to make sure the code you think should get executed actually did get executed. This is very good for segmentation faults. If it print 1, 2, 3 then there is a segmentation fault, you know it was something after it printed 3.

Print variables, especially after you set them. For example in getOrigin, once you determine there IS an origin and you find the address, print that value to make sure you captured it properly.

Segmentation faults.

Check your scanf, fscanf, and sscanf statements. Make sure you passed a POINTER for the storage variable.

Check any library functions you used. Make sure you got the parameters correct and in the right order.

Print sentinels as described above.

Comment things out. Comment out the call to firstPass and see if that clears your segmentation fault. If it does, uncomment the call and start commenting out things inside of firstPass until the segmentation fault goes away. Do this for other function calls if needed.