**University of Michigan – Dearborn**

**CIS 200 – Computer Science II**

**Lab# 3**

**Professor Robert Mann**

Demetrius Johnson

[meech@umich.edu](mailto:meech@umich.edu)

September 29, 2020

**Table of Contents**

Contents

[Question 1 3](#_Toc52269865)

[Source Code –see Q1\_CIS-200-LAB2-DemetriusJohnson.cpp 3](#_Toc52269866)

[Screenshots 3](#_Toc52269867)

[Question 2 4](#_Toc52269868)

[Source Code –see Q2\_CIS-200-LAB2-DemetriusJohnson.cpp 5](#_Toc52269869)

[Screenshots 5](#_Toc52269870)

[Question 3 7](#_Toc52269871)

[Source Code –see Q3\_CIS-200-LAB2-DemetriusJohnson.cpp 7](#_Toc52269872)

[Screenshots 7](#_Toc52269873)

# Question 1

// LAB 3 - CIS 200

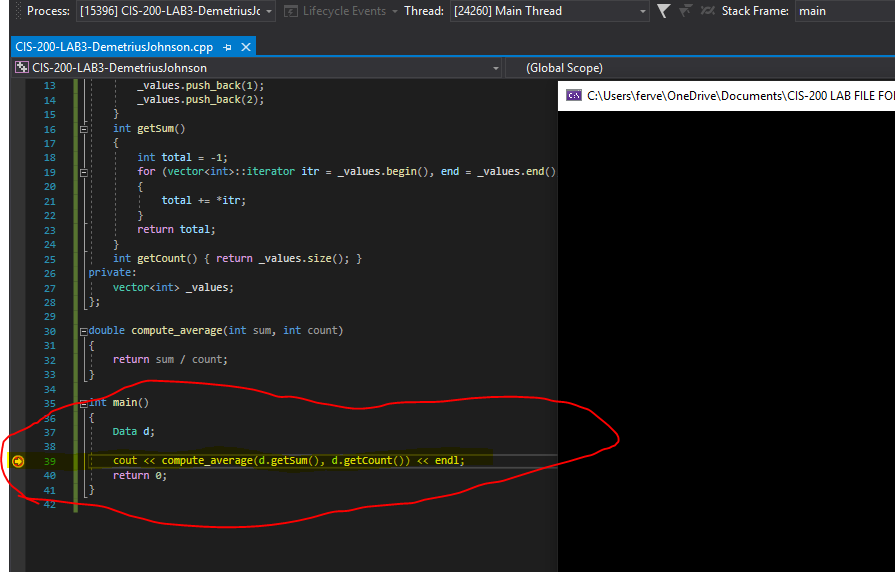
Question 1

-------------------------

1. Enter the above program without changes, compile, and execute. Does it work correctly?

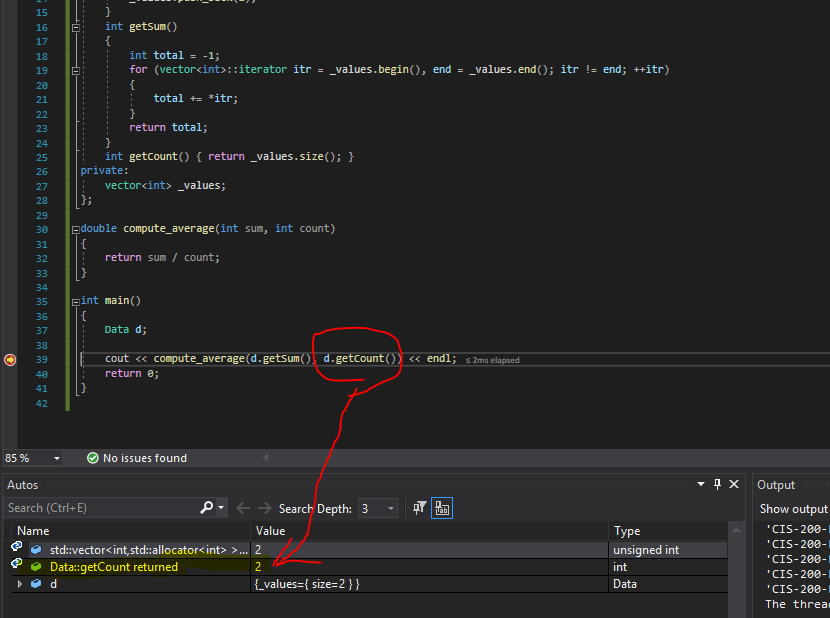
The program will compile and execute, however; it does not work properly as the calculated average is not correctly written.

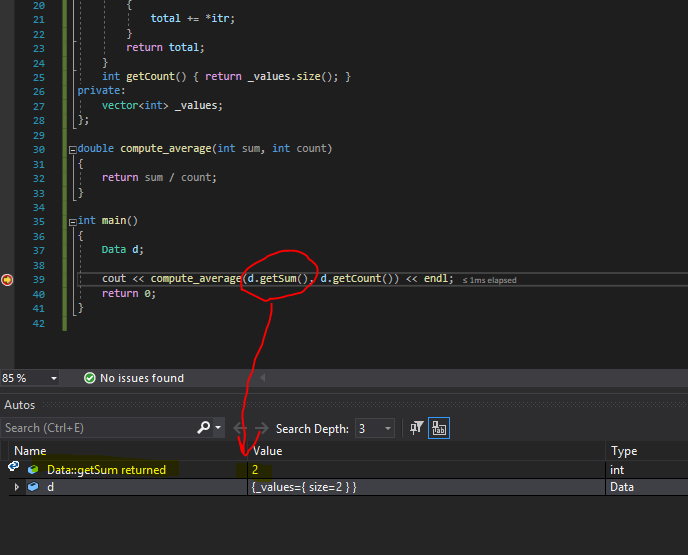
1. Set a breakpoint on the line where function compute\_average is called. In this simple program the function is only called once so putting the breakpoint inside the function itself would make more sense. But what if the function were called many times, from different locations? In this example, better to isolate the call that has the issue and trace the function with those specific values.
   1. Use the snipping tool to capture an image showing the breakpoint is set and the line is about to be executed.

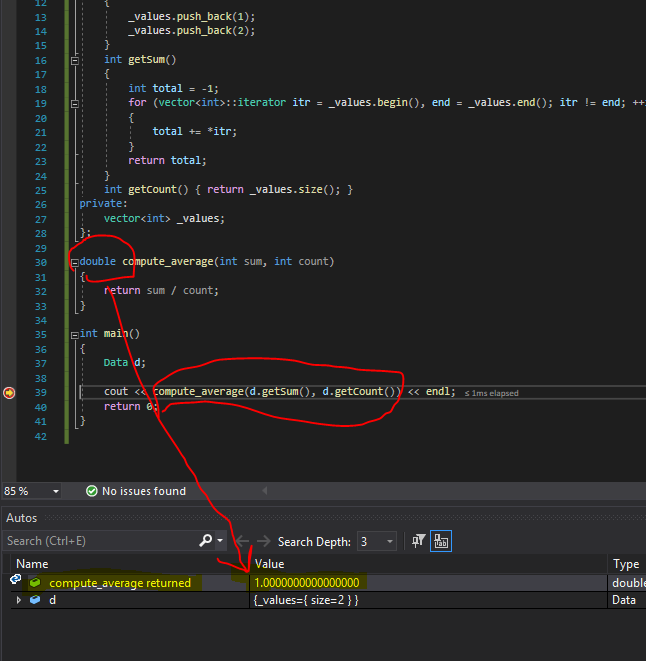


* 1. Run (F5) the program.
  2. When the debugger stops, step into (F11) functiongetCount.
  3. Rather than hit step over (F10) multiple times to walk through the function, step out (shift+F11) to return to the function call location.
  4. Repeat steps b & c for function getSum.
  5. Finish (F5) running the program

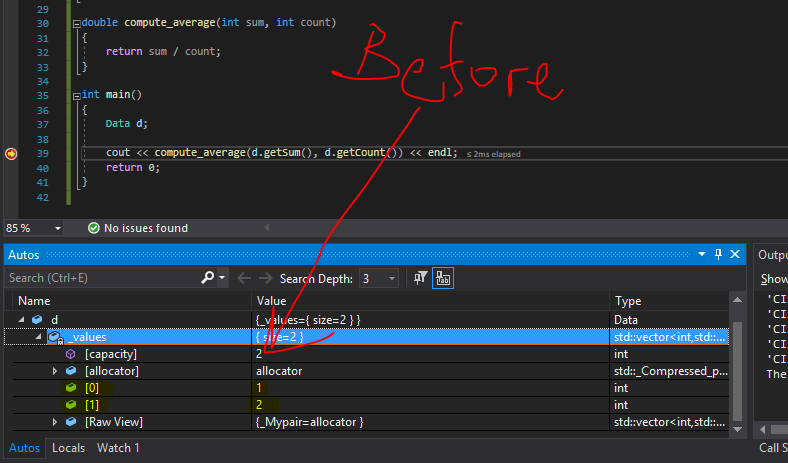
1. Sometimes, such as with the parameters to function compute\_average, we don’t store function results but we still wish to see the values. Repeat step 2, but after 2c and 2d, look at tab Autos for the function name to see the value returned by the function.
   1. This technique is better than (re-)writing your code as  
       avg = compute\_average( d.getSum(), d.getCount() );  
       cout << avg << endl;  
      just so you can see the value returned by compute\_average().
   2. Use the snipping tool to capture images of tab Autos showing the values returned by each of the three functions



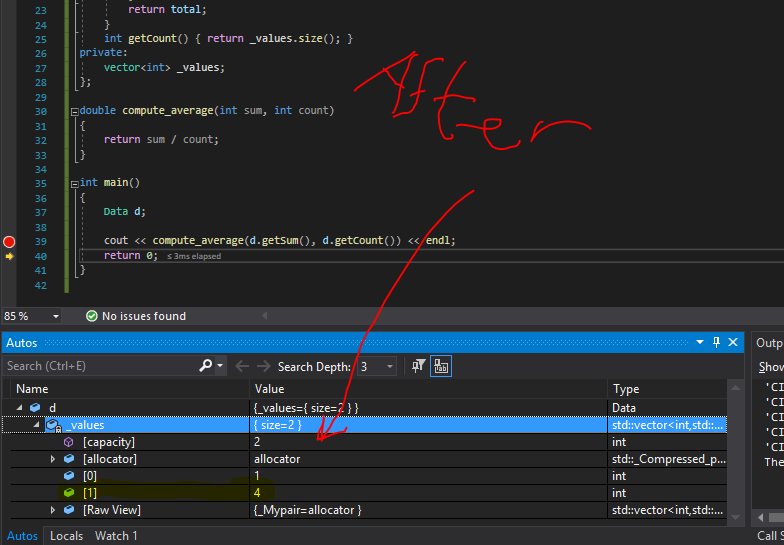




1. Sometimes we realize during debugging/testing that a value is initialized incorrectly. Variable values can be modified during execution.
   1. Repeat step 2 through 2c
   2. In autos, click the arrow to left of d to expand the class
   3. Click the arrow to left of \_values to expand the vector
   4. Use the snipping tool to capture images showing the value of variable \_values before and after the next step

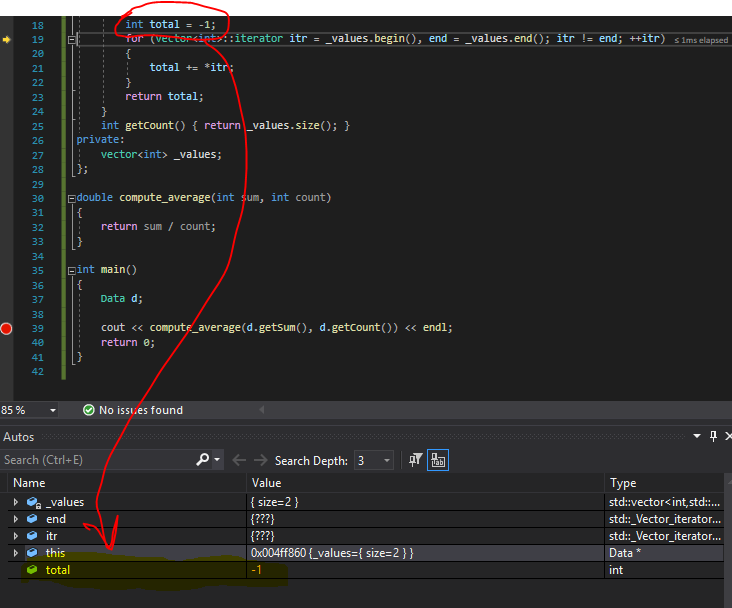


* 1. Click on [1], type 4 to change its value from 2 to 4 and hit enter

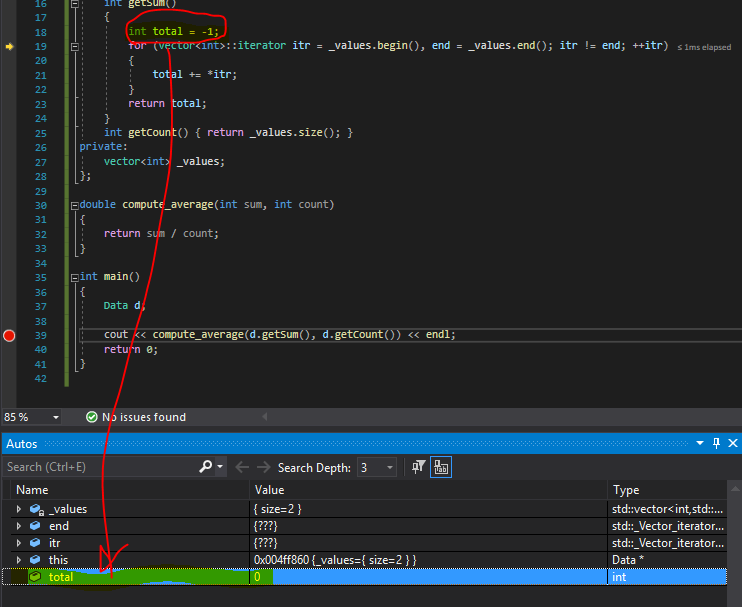


* 1. Continue with step 2d and finish
  2. This trick can also be used when a wrong value has been identified (possibly as the return from a function). The value can be changed and then continue with program execution to verify remaining code works correctly.

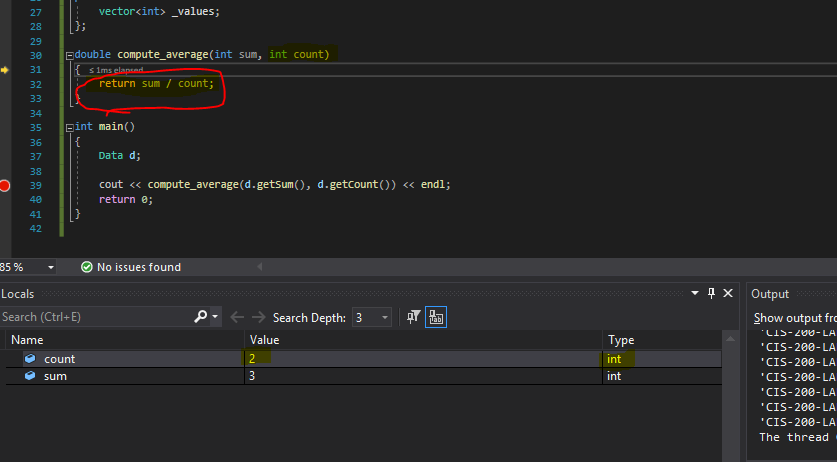
1. Code can be modified during the debugging process. Warning: be sure to save your source code before debugging in case the change doesn’t work. If it does work, save it again immediately so you don’t lose the change you just made!
   1. Repeat step 2 through 2c, then step into function getSum.
   2. Use the snipping tool to capture images showing the code before and after the next step

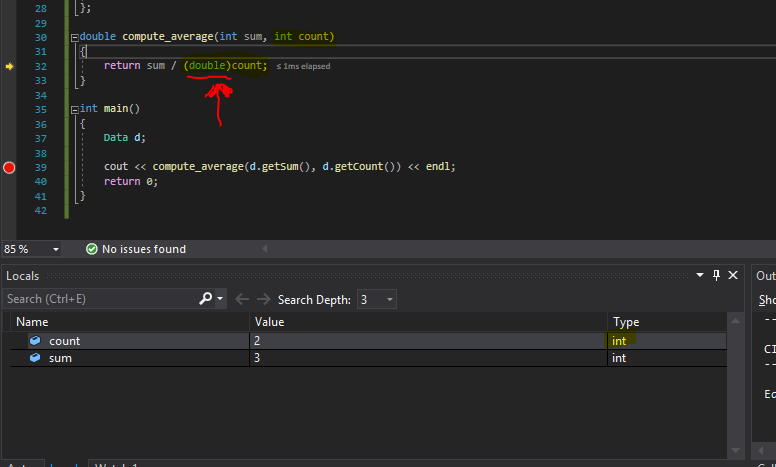


* 1. Change the initialization value of total to zero

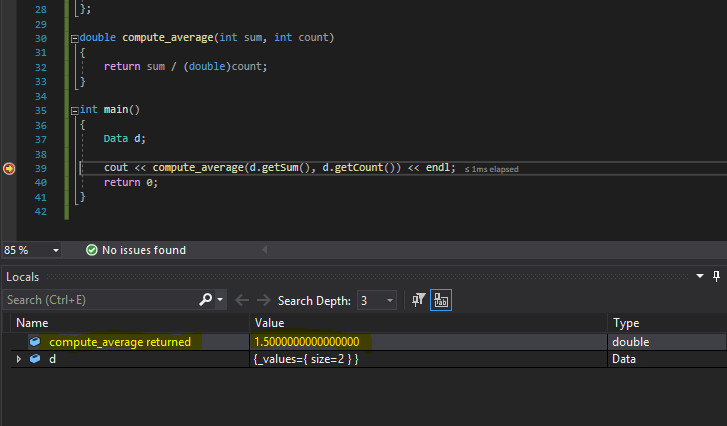


* 1. Step out of the function and step into function compute\_average
  2. Change the code to cast count as a double in the return statement

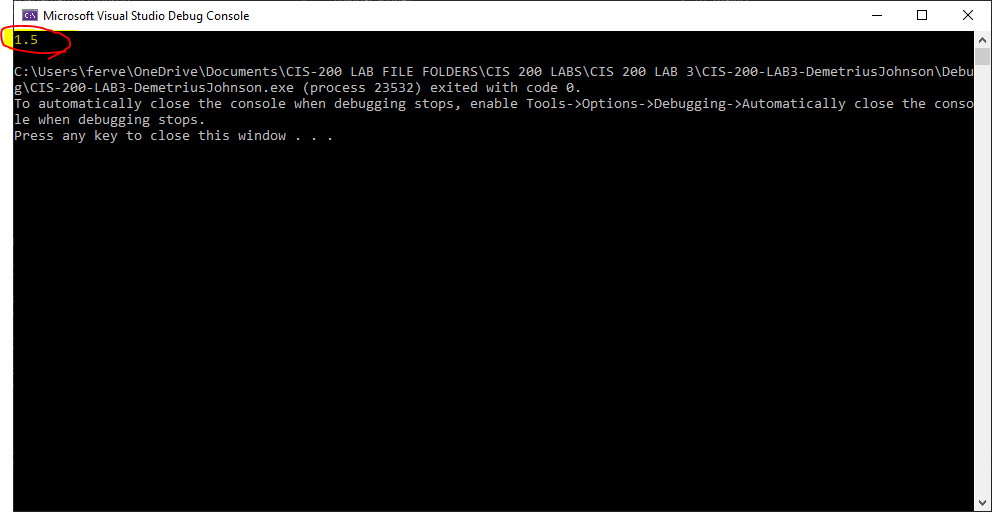




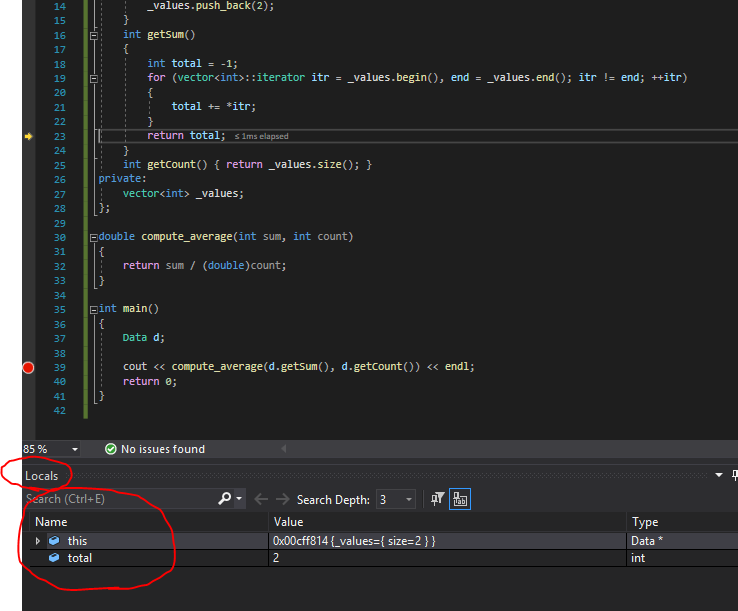
* 1. Step out of the function
  2. Execute one statement



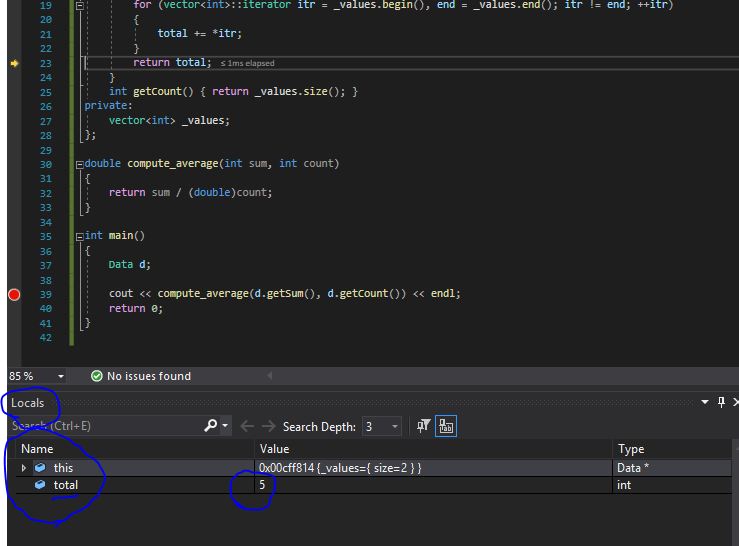
* 1. Verify the output is correct and use the snipping tool to capture an image of the output



1. As you are stepping through your code (F10), you may find you’ve gone too far and missed seeing a value. Rather than starting over, right-click on the line you’d like to execute next and select Set Next Statement. Execution will continue from that point. CAUTION: program execution point will change but values of variables will remain as they currently are so perform this operation very carefully and selectively!
   1. Execute your program up to the return statement in function getSum
   2. Use snipping tool to capture an image showing local variable values



* 1. Set next statement to be first line of for loop and continue execution up to the return statement
  2. Use snipping tool to capture an image showing local variable values



## Source Code –see lab 3 Q1 CPP FILE

## Screenshots

# Question 2

// LAB 3 - CIS 200

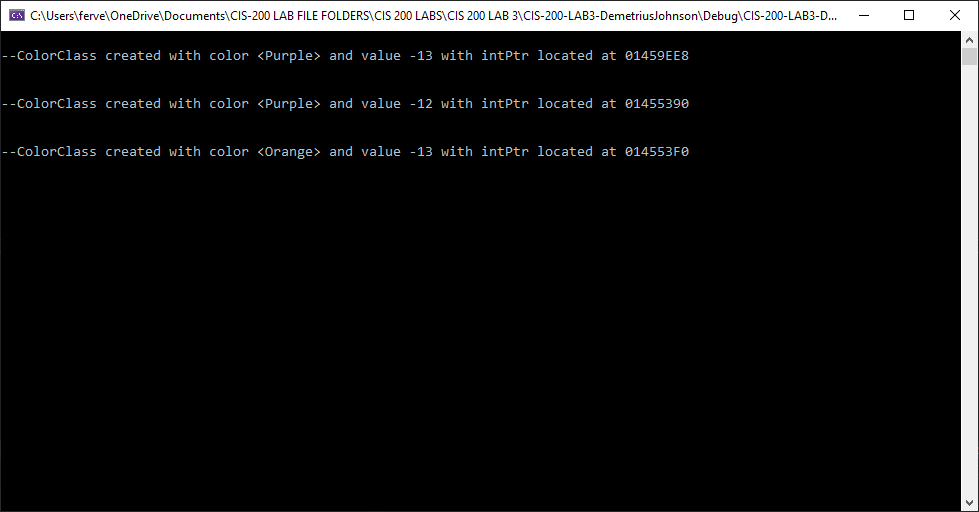
Question 2

-------------------------

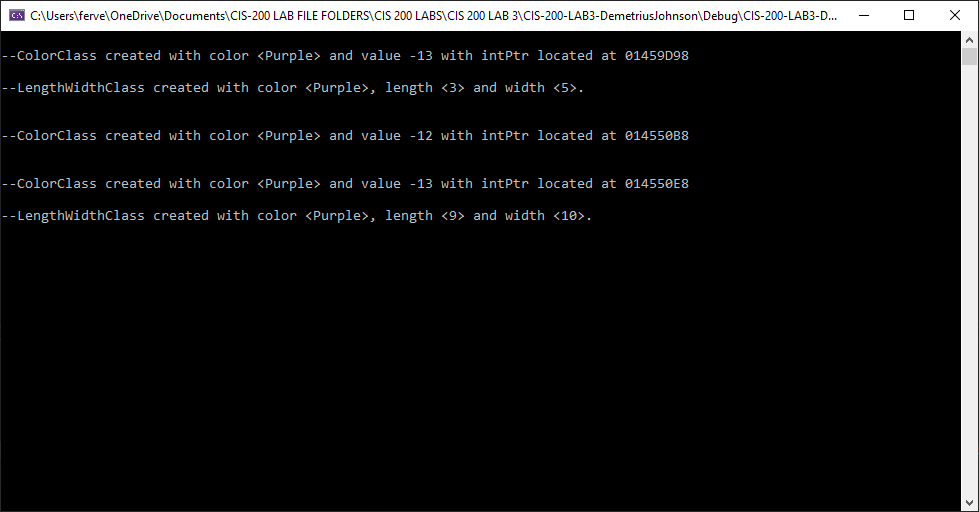
## Source Code –see lab 3 Q2 CPP and H files

## Screenshots

TEST 1 – Declare three 1st class objects

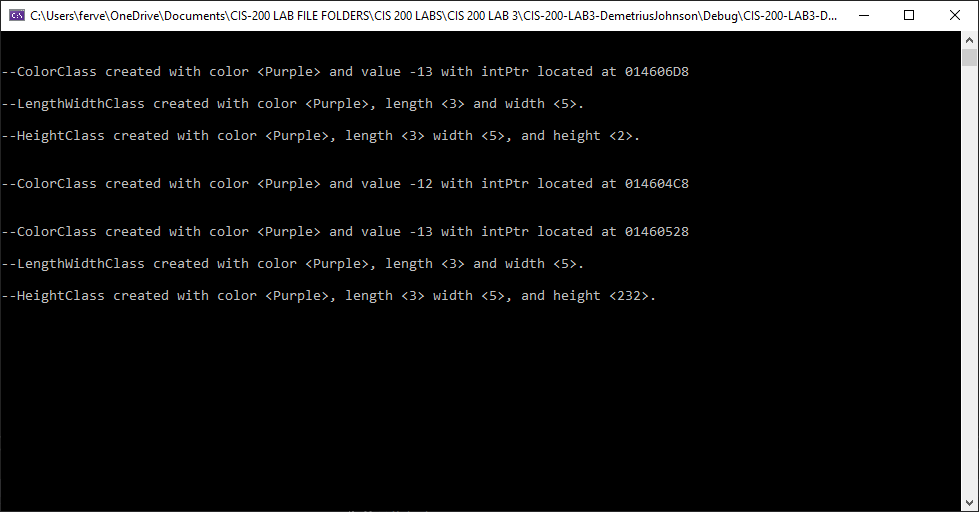


TEST 2 – declare three 2nd class objects

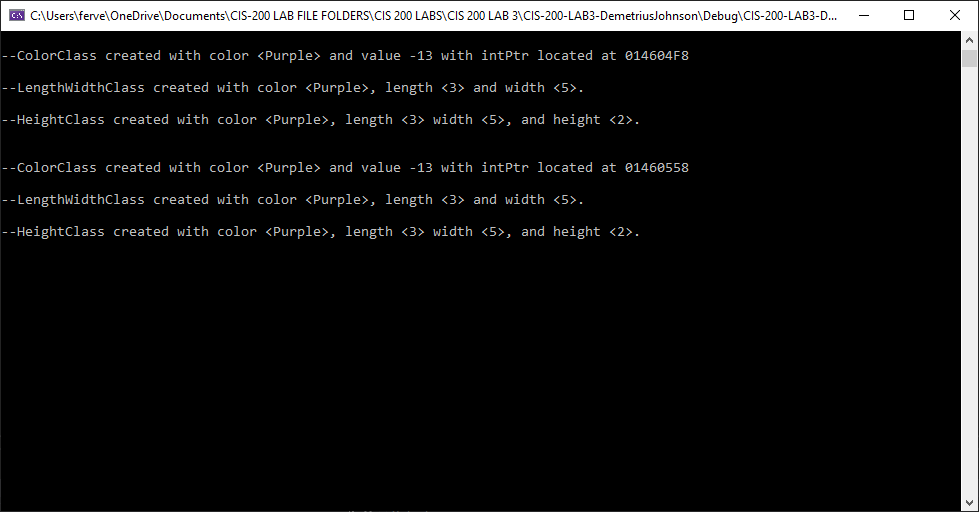


\*Notice how LengthWidth class only appears twice, this is because the default copy constructor is called for the LengthWidth class attribute assignments.

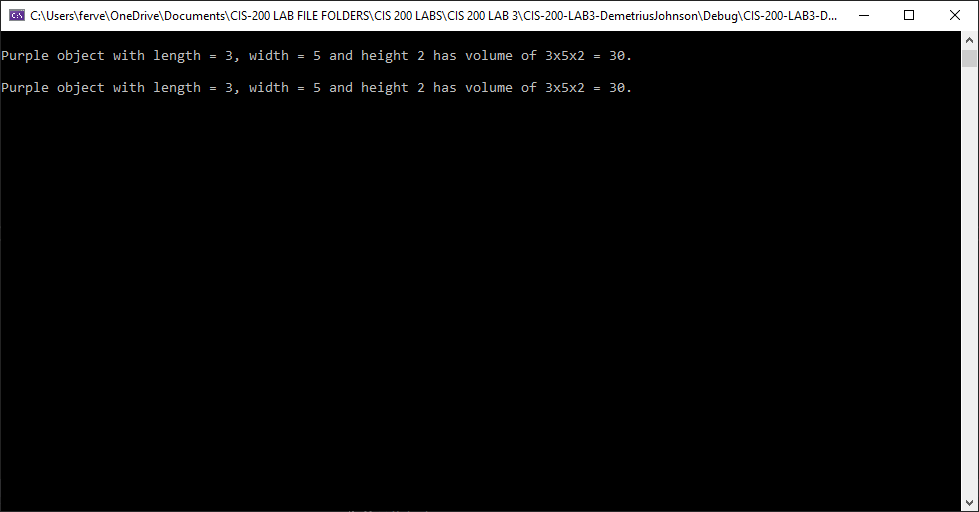
TEST 3 – declare three 3rd class objects



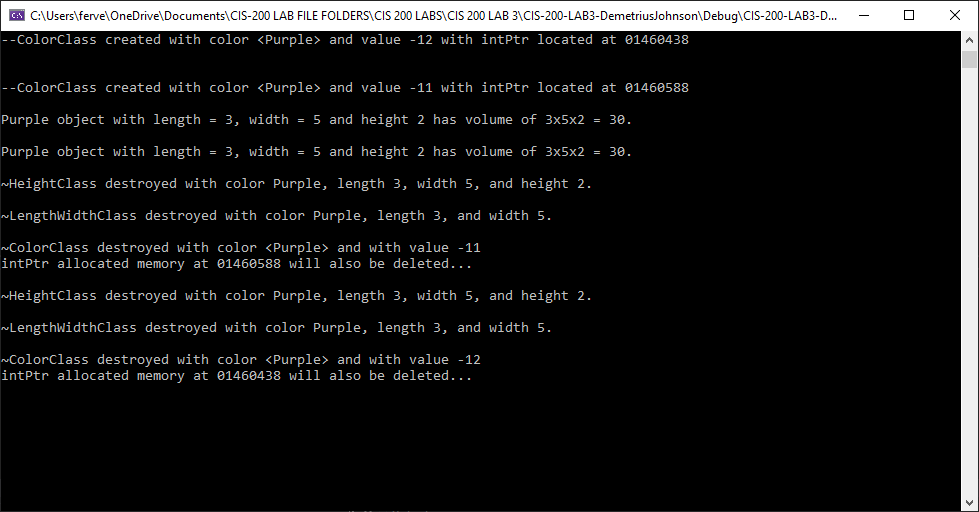
TEST 4 – Declare thing1 and thing2 of type 3rd class



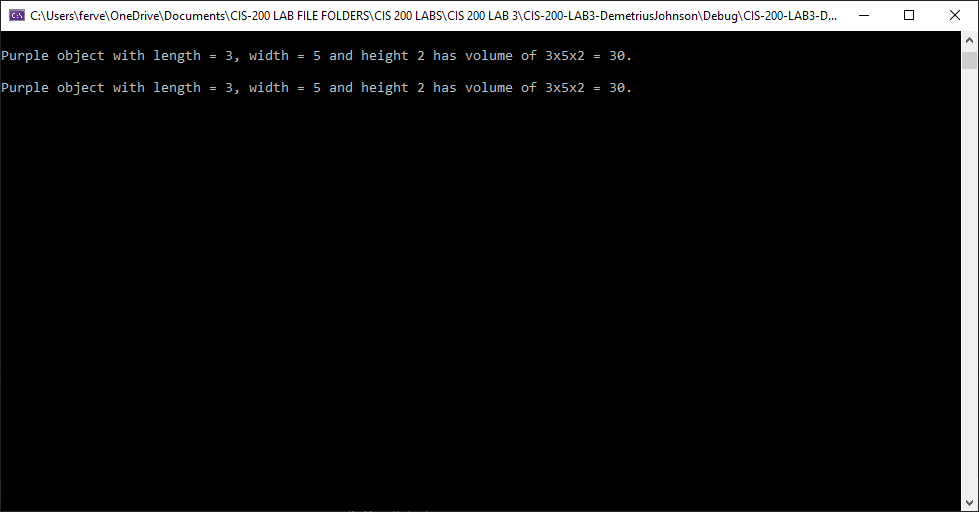
TEST 5 - PrintMe function of thing1 and thing2 objects



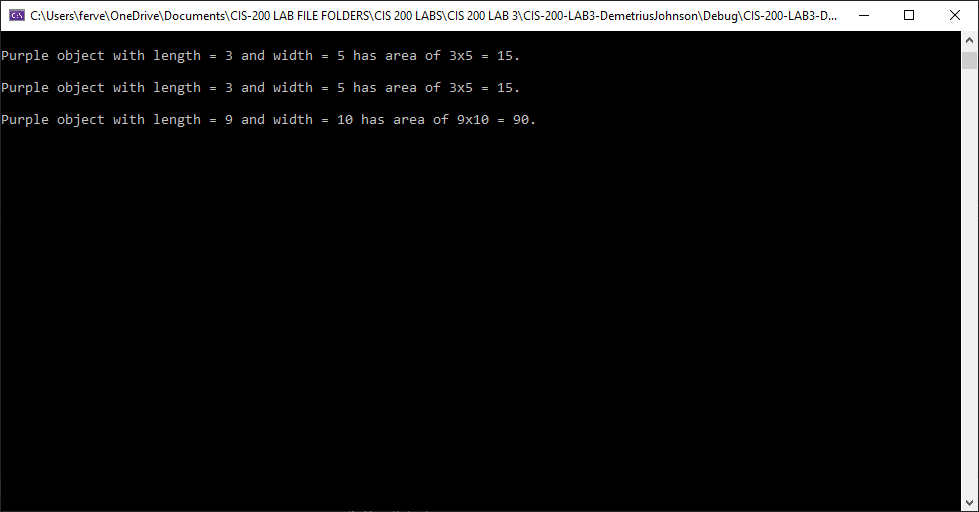
TEST 6 – test function



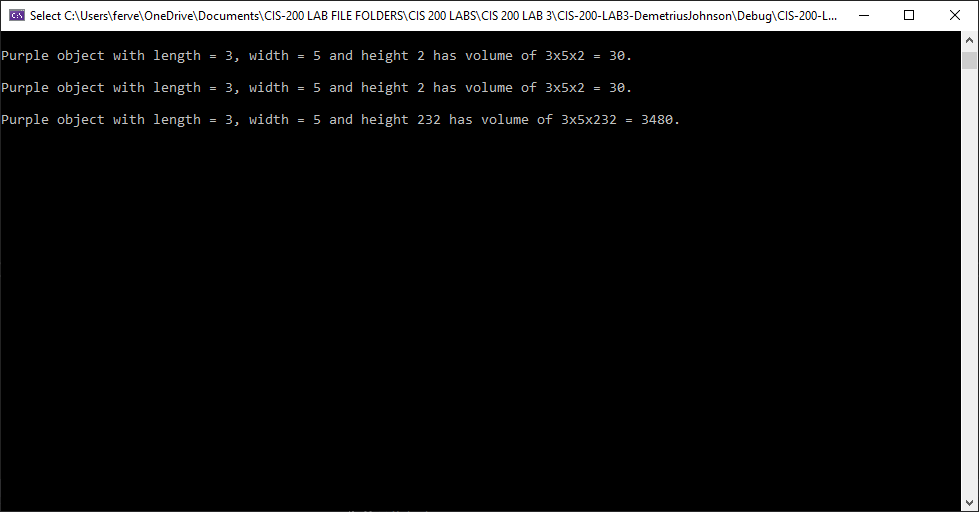
TEST 7 – PrintMe function of thing1 and thing2 (again)



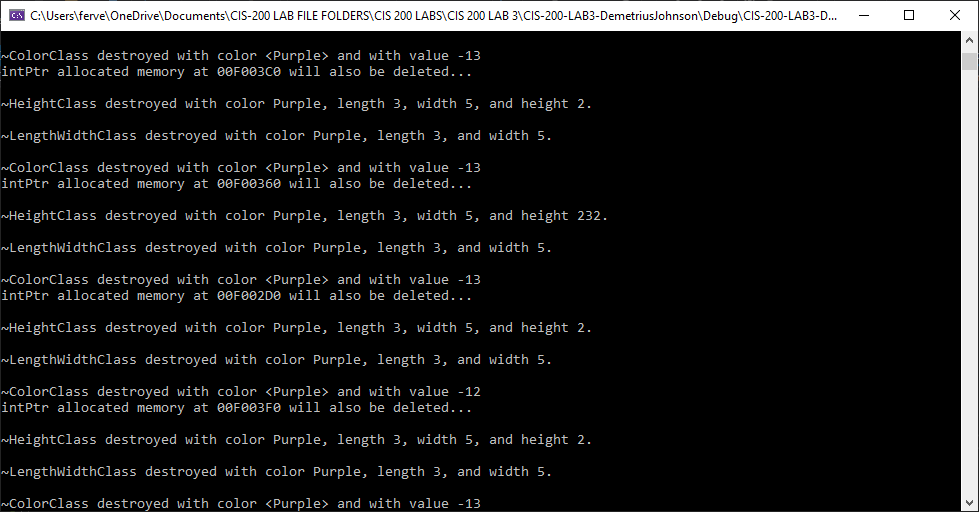
TEST 8 – PrintMe function for 2nd class objects (print me for all three declared objects)



TEST 9 - PrintMe function for 3rd class objects (print me for all three declared objects)



TEST 10 – DESTRUCTOR MESSAGES AT END OF PROGRAM



TEST 11

Go back to all classes and change private attributes to protected. What inheritance type should be used now in 2nd class and 3rd class? Put a comment in your report as question 2A describing whether you need to change inheritance from public to something else. Modify your classes to directly access inherited attributes where possible. Re-run instructions listed in paragraph Assignment, above, using the new versions of your classes.

Response:

Output is exactly the same. When you change variables in a class from private to protected, it only makes it easier for upper level derived classes to access the variables. Inheritance type need not change. After changing all classes private data members to protected, I simply replaced all getter functions inside of the derived classes with the direct variable that the function served to get since now all the derived classes pass on the access of the protected variables to all derived classes, and with using Public inheritance, the protected variables continue to funnel down all the way up to the highest-level derived class.

For example, in the highest derived class (the 3rd class), you can now access all public and protected data members off all of the previous 2 classes below it. Below are the screenshots showing the program executes perfectly fine even after the changes were made. It is also reflected in the latest revision of the code.

