**Submission of Your Work**

You need to prepare and submit ONE SINGLE MS Word document to Canvas (in your lab section) as LastName\_FirstName\_Labxy.doc. It must contain:

* Your NAME is the only item on page 1
* For ***each*** question:
  + Specify the question number, if more than one question.
  + Source code. Copy/Paste your final source code. You must include standard “comment header” even if code is provided. *Do Not* paste a snippet of your source code, it must be copy/pasted.
  + Output results. Paste in a snippet of output showing results

**Question 1**Read the complete assignment before beginning. This is an exercise to get practice using the debugging tool. If you haven’t previously, watch this short tutorial: <https://www.youtube.com/watch?v=eEJ5xKO7c3o> No test plan is needed for this question

#include <vector>

#include <iostream>

using namespace std;

class Data

{

public:

    Data()

    {

        \_values.push\_back( 1 );

        \_values.push\_back( 2 );

    }

    int getSum()

    {

        int total = -1;

        for (vector<int>::iterator itr = \_values.begin(), end = \_values.end(); itr != end; ++itr )

        {

            total += \*itr;

        }

        return total;

    }

    int getCount() { return \_values.size(); }

private:

    vector<int> \_values;

};

double compute\_average( int sum, int count )

{

    return sum / count;

}

int main()

{

    Data d;

    cout << compute\_average( d.getSum(), d.getCount() ) << endl;  
 return 0;

}

1. Enter the above program without changes, compile, and execute. Does it work correctly?
2. 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.
   2. Run (F5) the program.
   3. When the debugger stops, step into (F11) functiongetCount.
   4. Rather than hit step over (F10) multiple times to walk through the function, step out (shift+F11) to return to the function call location.
   5. Repeat steps b & c for function getSum.
   6. Finish (F5) running the program
3. 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
4. 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
   5. Click on [1], type 4 to change its value from 2 to 4 and hit enter
   6. Continue with step 2d and finish
   7. 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.
5. 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
   3. Change the initialization value of total to zero
   4. Step out of the function and step into function compute\_average
   5. Change the code to cast count as a double in the return statement
   6. Step out of the function
   7. Execute one statement
   8. Verify the output is correct and use the snipping tool to capture an image of the output
6. 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
   3. Set next statement to be first line of for loop and continue execution up to the return statement
   4. Use snipping tool to capture an image showing local variable values

**Question 2**

This is an exercise to get practice creating classes and using inheritance. No test plan is necessary. Each class must have its own .cpp and .h files.

1st class: has one private attribute, color, which is a string, and another which is a pointer to an int. One constructor, with a string parameter with default value “Purple”. The constructor will create a new int and assign it to the pointer variable, then give it value -13. The constructor displays message “Class created with color ccc”, replacing ccc with the color variable. Destructor, which displays message “Class destroyed with color ccc and number nnn”, replacing ccc with the color variable and nnn with the value the pointer points to, and deleting the pointer. Also create appropriate get and set methods for color. You’ll have to write a copy constructor for the class as well as overloading operator=. Have the copy constructor increase the value the new pointer points to by 1 (only the copy will increase, not the original) and display message “Class created with color ccc and number nnn”, replacing ccc with the color variable and nnn with the value the pointer points to.

2nd class: inherits public from 1st class. It has private attributes of length and width, both int. One constructor, with int parameters with default values of 3 and 5. The constructor displays message “Class created with color ccc, length lll, and width www”, replacing ccc, lll, and www with the appropriate variable values. Destructor, which displays message “Class destroyed with color ccc, length lll, and width www”, replacing ccc, lll, and www with the appropriate variable values. Create appropriate get and set methods for the two attributes. Create a PrintMe method that prints “ccc object with lll x www (area = aaa)” replacing ccc, lll, and www with the appropriate variable values and aaa with length multiplied by width.

3rd class: inherits public from 2nd class. It has private int attribute of height. One constructor, with an int parameter with default value of 2. The constructor displays message “Class created with color ccc, length lll, width www, and height hhh”, replacing ccc, lll, www, and hhh with the appropriate variable values. Destructor, which displays message “Class destroyed with color ccc, length lll, width www, and height hhh”, replacing ccc, lll, www, and hhh with the appropriate variable values. Create appropriate get and set methods for the attribute. Create a PrintMe method that prints “ccc object with lll x www x hhh (volume = vvv)” replacing ccc, lll, www, and hhh with the appropriate variable values and vvv with length multiplied by width multiplied by height.

Create a function called Test that accepts a parameter to the 3rd class. Inside the function, declare an instance of 3rd class and assign the parameter to it. Call the PrintMe method of both the parameter and local variable.

**Assignment**: declare multiple instances of each object (1st class, 2nd class, and 3rd class), with and without initialization values (to show default parameter values work). Declare two more instances of 3rd class, called thing1 and thing2, using default values. Call PrintMe method of every object declared of type 2nd class and 3rd class. Call Test, passing thing2 as parameter. Call PrintMe of thing1 and thing2. Put a breakpoint on the last ‘}’ of your program so you can see final destructor messages. Capture a snippet of output and put it in your report along with all source code.

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.