COMSC-200 Lab 11
Using Inheritance

GOOD PROGRAMMING PRACTICES *show / hide*

ABOUT THIS ASSIGNMENT
In this lab session, we continue with the **GeometryHomework** solution from labs 1-4, so that it uses inheritance. You will also practice programming with classes by doing exercises from our textbook.

As you complete this assignment, post the required files to the COMSC server. You may post them all at once, or one-at-a-time as you complete them, as you prefer. The individual labs are graded in order, starting with "a", which must be fully correct before "b" is graded, and so on. This assignment is worth 50-points, to be awarded after all labs are successfully completed. Use the "Submit your SP2015 work" link on the class website to post your file for this lab, using the **lab11** folder provided.

Lab 11 Using Inheritance [`GeometryHomework5.cpp`]
Make substantial structural changes to the GeometryHomework program from lab 4, applying C++ technologies that we have learned since lab 4. The first change is *optional*. The changes include:

1. OPTIONAL: Use a `vector` to replace the fixed-size `void*` array. Use the "linked list" implementation, starting with zero-length, and not the "array" implementation that starts pre-sized. You may use either `const void*`'s or `const Shape*`'s as the data type in the vector. For example, `vector<const void*> shapes;` or `vector<const Shape*> shapes;`. *Replace the int array with a vector, too!*

In your vectors, do *not* store constants! No "`const void* const`" or "`const Shape* const`" or "`const int`". Some compiler implementations don't allow assignment of values to constants in vectors. NOTE: This applies to the vector declarations only. You will still need to use constant, read-only pointers in the loops in main, as in the previous version of GeometryHomework. It's okay to copy constant pointers.

If you want to use `const Shape*`'s, define the base class like this: `class Shape{};`. The class needs no members -- it just exists so that we can have a vector of read-only `const Shape*`'s. The 2-dimensional shapes (square, rectangle, and circle) would derive from `Shape`.

If you do *not* choose to implement the vector option at this time, then you may continue to use the arrays as in the lab 4 version.

Iterators: With vectors, you can use either `operator[]` and the `size()` function, or you can use "iterators". (Note: do NOT use `vector::at(int)`, because it is not well-supported among compilers, while it's equivalent, `vector::operator[]` is.) If you use iterators, declare them something like this: `vector<const Shape*>::iterator it;` and `vector<int>::iterator it2;`. *Hint: Use a traversal loop in the output and deallocation structures that traverses both iterators in tandem.*

Remember that `it` returns *pointer to a pointer* -- you have to dereference it to get a `Shape*` like this: `*it`. To cast the pointers, you'll have to do something like this: `((const Circle*)(*it))`, which resolves to a `const Circle*`. To cast the shape ID `int`s, do something like this: `*it2`, which resolves to an `int`.

2. Modify the output functions to include an `ostream&` object as their first (and only) parameter. Call the

output functions with `cout` for that parameter. Inside the output functions, do NOT send to `cout`. Instead, output to the aliased object name as it appears in the parameter list. (I suggest using `out` for that name.) In this way, the output functions will be made generic, able to output to ANY stream -- console (`cout`), file (`fout`), network (`nout`), or memory (`sout`).

3. Use stand-alone, programmer-defined output stream manipulators. *(See lecture notes.)*

4. Do NOT use friend relationships, and do NOT write getters or setters other than those specified. If you have any remaining stand-alone or member functions to support outputting, remove them, and move their coding into your output functions.

5. Apply inheritance, so that the three 3-dimensional shapes (cube, prism, and cylinder) derive from their 2-dimensional relatives (square, rectangle, and circle, respectively). The derived classes should use their inherited dimensions, and add a dimension if needed. Do NOT declare new versions of the 2-D dimension data members in the derived 3-D classes.

Further, if you did not already do so in **GeometryHomework4.cpp**, make `const` every data member, non-constructor member function, local variable, and parameter variable as possible.

Do *not* use more than one source file -- that means no **.h** files, and just a single **.cpp**. *Post **GeometryHomework5.cpp** to the COMSC server for credit.* Save this file, because modifications will be made to this program in labs 12-13. The GeometryHomework series is the subject of seven (7) lab assignments.

Do NOT write getters to access the private members of a base class from a derived class, and do NOT use friend relationships.

How to pause a console program: *show / hide*

GRADING POLICY *show/hide*