COMSC-200 Lab 2
Encapsulation Of Data

GOOD PROGRAMMING PRACTICES *show / hide*

ABOUT THIS ASSIGNMENT
In this lab session, you will modify the **GeometryHomework** solution from lab 1 so that it uses data encapsulation, by applying C `structs`. You will also practice programming with encapsulation 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 **lab2** folder provided.

Lab 2a [`MyTime.cpp`]
Write the **CPP** program as presented in figure 9.1 of our Deitel textbook, with modifications per the universal requirements listed in lab 1a, and the grading policy. Name the file **MyTime.cpp**.

*Post **MyTime.cpp** to the COMSC server for credit.*

Lab 2b Encapsulating Data [`GeometryHomework2.cpp`]
In the lab 1 version of the GeometryHomework program, an input file was read and processed line-by-line. For possible use in a GUI program later, it would be important to read all of the input without immediate processing, and to process it all afterwards -- perhaps several times (for example, in a window's paint function).

So to make this easier, the geometric shapes described in the input file should be read and stored in an array. They should be stored as `struct`-based objects. Afterwards, the array can be traversed, these `struct`-based objects recalled, and the output produced.

Copy the GeometryHomework1.cpp from lab 1 to GeometryHomework2.cpp for this lab. Do *not* use more than one source file -- that means no **.h** files, and just a single **.cpp**. To use multiple files would needlessly complicate this solution. With one exception, the output should match that of the lab 1 version. Unlike lab 1, *read all of the file data before producing any output*. Only after the input file is read and closed, produce the output. Here is the exception: any objects with invalid names will appear at the top of the output list, outputted in the loop that reads the input file and stores the valid objects in an array. Invalid objects are not stored in the array, as they as ignored in any further processing.

PROGRAM CHANGES: Modify the program so that it includes `structs` for each of the six geometric shapes that were the subject of lab 1. Name these as follows: Square, Rectangle, Circle, Cube, Prism, and Cylinder. Include data members of the `double` data type for the required dimensions for each (for example, length and width for the Rectangle). Provide a descriptive name for each of these data elements (for example, "radius" instead of "r"). Do NOT include any other data members, and do NOT include any member functions -- avoid the temptation to extend the specifications beyond what is required.

Write stand-alone output functions for each shape (for example, `void outputPrism(Prism* p)`), which should output to the `cout` standard console output, using the format from lab 1.

Use an array of generic, variable mutating pointers (for example, `void* shapes[100];`), and another array of integer codes (for example, `int shapeId[100];`), to store up to 100 objects. You will need a way to keep track of exactly how many valid objects are stored. Also, use dynamic memory allocation (for example, `Rectangle* r = new Rectangle;`) to create `struct`-based objects, and store the memory address of each in the array of generic pointers. Set the dimensions per the input file, remembering to use default values of zero for any missing inputs.

Once the input file is processed, close it. In a loop, traverse the array of objects and output each, using the new output functions (using for example, `outputSquare((Square*)shapes[i]);`. Use the array of integer codes in an if-else or switch-statement, in order to have specific code for each shape.

After outputting all of the objects, deallocate all dynamically-allocated memory in a loop (using for example, `delete (Square*)shapes[i];`).

*Post **GeometryHomework2.cpp** to the COMSC server for credit.* Save this file, because modifications will be made to this program in labs 3-4 and 11-13. The GeometryHomework series is the subject of seven (7) lab assignments.

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

GRADING POLICY *show/hide*
