COMSC-200 Lab 13
Using The MFC

GOOD PROGRAMMING PRACTICES *show / hide*

ABOUT THIS ASSIGNMENT
In this lab session, we continue with the **GeometryHomework** solution from labs 1-4 and 11-13, to attach it to a graphical user interface (GUI).

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 **lab13** folder provided.

Lab 13 Writing With The MFC [`GeometryHomework.h`, `GeometryHomework.cpp`, `GeometryHomework7Dlg.cpp`, and `GeometryHomework7.exe`]
For this exercise, you will need Microsoft Visual Studio or another compiler that can produce a Windows-compatible GUI program named **GeometryHomework7.exe**.

Using Visual Studio, create a dialog-based MFC application, using the MFC in a static library. It's okay to work in "debug" mode, but use "release" mode for the final version of your program -- it's the smaller EXE that's located in the "Release" folder, not the larger one found in the "Debug" folder.

SPECIAL SCORING RULE
EXEs submitted in "debug" mode instead of the required "release" mode will lose FIVE points with no redo possible. So get this right the first time.

Create a GUI for running the GeometryHomework program. Instead of using an input file with an unspecified number of objects, provide a GUI that can be used for any user-selected object, with user-specified dimensions. Right-click here to download a sample of how this program should work.

Name the project as you wish, except do NOT name it "GeometryHomework", because it will create files whose names will conflict with the names you are to give to your Shape class files. If you name your project "GeometryHomework7", then the EXE it produces will automatically be named correctly. Otherwise, you can simply rename your EXE before submitting it.

Here are the specifications in 9 steps. Steps 1-3 should be done BEFORE you create an MFC project. Do them as if you were writing a regular console application, and verify that it works before creating the MFC project.

1. Copy your **GeometryHomework6.cpp** file from lab 12, and separate it into **GeometryHomework.h** and **GeometryHomework.cpp** files -- removing the `main` function. Put the class definitions and stand-alone function prototypes in to the H file. Put the function definitions into the CPP. This will allow your shape classes to be used in a project that consists of multiple H and CPP files.

2. If you have not already done so, generalize the output functions so that they can write to ANY stream, and not just `cout`. To do so, modify all of the `output` functions so that they take an `ostream` object

reference in their parameter list. Name the reference `out`, and change `cout` to `out` in those functions. (Since the `main` function will be gone, there is no need to change any calls to these functions anywhere.)

3. If you have not already done so, add an overloaded `operator<<` stand-alone function for the base class, **Shape**, that calls its pure virtual output function. This allows shape objects to be sent directly for outputting, without actually calling the output function:

```
ostream& operator<<(ostream& out, const Shape* s)
{
  s->output(out);
  return out;
}
```

Note that the function specifies that a *pointer* or memory address of a Shape object (or an object of a derived class) be used in the `<<` expression. For example, if you create a Square object in a handler function, and you create an ostringstream object to convert the Square to a string, you need to send the memory address of the Square to the ostringstream, and not the Square itself.

Be sure to put only a PROTOTYPE for the stream insertion operator in the H file, with the function DEFINITION in the CPP. Otherwise you will get a compile error that references this function.

This would be a good place to ***test everything***. Create a test CPP that #includes your new **GeometryHomework.h** and copy/pastes the main function from **GeometryHomework6.cpp**. Be sure to modify the output call in main to have cout as a parameter. Compile that test CPP with your new CPP, as a 2-CPP project, and verify that it runs like it did before.

4. Create the MFC project. Confirm that you can compile and run, and then copy your new **GeometryHomework.h** and **GeometryHomework.cpp** into the project folder that contains all its other H and CPP files.

Be sure to put this as the FIRST statement in **GeometryHomework.cpp** (after any identifying comments):

```
#include "stdafx.h"
```

5. In your handler(s), provide a way for the user to select one of the six geometric objects -- it may be by pull-down menu, as in the sample program, or by typing into a text field, or by some other way of your choosing.

6. On your dialog screen, provide labeled data entry fields for input dimensions. Also include your NAME and student ID either in a CStatic control or in the window's title.

7. Provide a button to start the calculation for the selected object and dimensions.

8. Provide a text box or label for displaying the results. Use an `ostringstream` object and the shape object's new stand-alone `operator<<` function to get the results to be displayed -- do *not* add functions to the classes to accomplish this. (Refer to the lecture notes for details.) If you want multi-line output, set the multi-line property of the box to true, and use \r\n as the line break.

Note -- you may see a square box appended to the end of your text. If so, find out why it's happening, and find a way to remove that box. It's a -5 point penalty with no redo's for displaying the box.

9. In your handler(s), create a token array as you did in the original main function, with four elements. Remember that element zero is the ignored string that identifies the shape type! Set the other three

elements of the token array to const char* versions of the CStrings you get from the edit boxes. Use the token array in your constructors for the various shapes in the handler(s).

You only need to create ONE temporary object at a time in the handler. You may use either static objects or dynamically-allocated objects, but if you do the latter, be sure to deallocate the object in the handler before exiting so that you avoid memory leaks. Note that you do not need dynamically-allocated objects.

You may have a problem with naming conflicts, because the MFC does not use namespaces. There is a class named Rectangle in the MFC, which may conflict with one of your classes. There are several ways to fix this, and you may do whatever works, but the way provided in the language is to use namespaces. Here's an example, using an arbitrarily named namespace, "comsc" -- note that this effectively changes the name of your `Rectangle` class to `comsc::Rectangle`:

```
namespace comsc
{
  class Rectangle
  {
    public:
    void fun() const;
  };
}
```

```
void comsc::Rectangle::fun() const
{
}
```

```
int main()
{
  comsc::Rectangle r;
  r.fun();
}
```

Don't make this into a bigger deal than it needs to be. You do not need tabbed controls or pull-down menus or radio buttons or check boxes. You do not need buttons other than ok and cancel. You do not need handlers other than for one button, like the ok button. Use static controls for labels and instructions to the user, and edit boxes for I/O. This is an OOP exercise -- not an MFC training session.

Post *only* **GeometryHomework.h**, **GeometryHomework.cpp**, **GeometryHomework7Dlg.cpp**, and *the **Release** version of* **GeometryHomework7.exe** to the COMSC server for credit. The GeometryHomework series is the subject of seven (7) lab assignments, ending with this one.

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

GRADING POLICY *show/hide*
