CS 121 Week 7 Worksheet Solutions - Classes (Defined in Multiple Files)

Syntax Work:

Notes: All parts needed to be completed are marked with *TO-DO* in the below syntax:

FCylinder.h and *FCylinder.cpp:*

- Method *getHeight* (needs prototype and definition); mimic *getRadius*
- Method *computeVolume* (needs prototype and definition); mimic *computeSurfaceArea*
- Method *printSelf* (needs prototype and definition); **HINT:** Read the comment on what this does

main.cpp:

- Fill in the areas that have comments (with no code below). Should all be in sequential order.

I left the solution for this in a FCylinder.cpp, FCylinder.h, and fcyl-main.cpp file (in the solutions folder).

Questions (True/False):

1. An object is an instance of a class
   True. That's pretty much its definition through-and-through.

2. Classes and structures are exactly the same
   False. Check here (stackoverflow question on c-struct versus classes)

3. A deconstructor is called when creating an object
   False. Deconstructors are called on an object when you leave the scope of the object (i.e. when they are "destroyed").
4. You can only have one constructor in a class
   False. You can have many different types (for instance, FCylinder has three).

5. Constructors and deconstructors do not need to have the same name as the class they belong to

   False. They require the same name, except destructors require a tilde (`~`) in front as well.

6. Constructors can be called at any given time like a method

   False. Constructors are only called on declaration of an object.

7. Member variables in a class are normally public

   False. They are normally private.

8. Member variables in a structure are normally public

   True. Since structures are public on default, people that use structs typically don't care if its contents are accessible by outside sources.

9. An accessor/getter method of a class should allow an instance of a class change its member variables

   False. Accessor/getter methods should only return the variables (never change).

10. A mutator/setter method of a class should allow an instance of a class change its member variables

    True. Think of "mutator" as "mutate" meaning "change" and "set" is self-explanatory.

11. Inline functions are good to use for any type of function, be it big or used very often.

    False. Since inline functions have their contents pasted onto the lines in which they are referred, it's bad for really big functions (and those called extremely frequently). Check out this link (stack overflow link on benefits of inline functions in c++).

Concept Questions:

1. Give an example of an inline function (you can use *FCylinder* with your example, however there are no inline functions in there).

   An inline function is a function that is defined in the class declaration itself.

   `// Implementing "getRadius" and "getHeight" as inline :`

```C++
  class FCylinder
  {
    private:
      //...code from before...
    public:
      //...code from before...
      double getRadius(){ return radius; }
      double getHeight(){ return height; }
      //...code from before...
  };
```

2. When passing an object through a function or method, why should it be called by reference (or const reference)? Give two reasons.
   Reasons for by-reference:
   - To have the ability to change its values
   - If we pass an object by-value, we will only affect the copy (and not the original)

   Reasons for const-reference:
   - Classes can be large (a class is a collection of data: member variables, methods, etc.)
     - This refers to the byte-size
   - When working with bigger and bigger objects the amount of time and resources it takes to pass an object by-value increases (and we don't want that).

3. Why, do you think, I named the class (from the example I gave) *FCylinder* and not just *Cylinder*?
   If we were to ever include a library with a class called *Cylinder* inside of it, there would be a naming conflict since the compiler won't know which to use in a C++ statement.

   There are ways around this (namely by using namespaces) but for typical code it's a good idea to give your class a prefix of some sort to distinguish it (e.g. I used "F" for "FCylinder").

4. What is the difference between *public* and *private* in a class/structure? Give an example supporting yourself (you can use *FCylinder* for this if you want).

   Public: Accessible outside of the class itself
   Private: Inaccessible outside of the class itself

   Example: You can call *second_cyl.printSelf()* but not *second_cyl.height* outside FCylinder.h/cpp

5. When would someone prefer using a structure over a class? What about a class over a structure?

   Structures:
   - Used typically for making small (and to quickly coded) objects.
   - Used normally if you don't care about the security of the object (i.e. public access)
   - Example: If I needed a simple object for handling a student's age, ID, and gpa

   Classes:
   - Used for more complicated objects.
   - Used normally if you need a secure object (with private or protected access)
   - Example: A marketing program that takes into account customers & store transactions,
   - Example: A level with NPCs in a video game (which holds a lot of info like number of enemies, number of power ups, locations of enemies and power ups, etc.)