CS 121 Week 7 Worksheet - 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.

FCylinder.h:

```
#include <iostream>

using namespace std;

// required for volume and area methods
const double PI = 3.141592;

class FCylinder
{
  private:
    // member variables
    double radius, height;              // doubles for radius & height

  public:
    FCylinder();                        // default constructor
    FCylinder(double r, double h);      // overloaded constructor
    FCylinder(const FCylinder& fc);     // copy constructor; optional as C++
                                        // defines this automatically (good to make)
    ~FCylinder();                       // deconstructor

    // getter Methods
    double getRadius();                 // return radius
                                        // return height (TO-DO)

    // setter Methods
    void setRadius(double r);           // set radius (with bounds-checking)
    void setHeight(double h);           // set height (with bounds-checking; TO-DO)

    // misc. Methods
    double computeSurfaceArea();        // returns surface area of FCylinder

                                        // returns volume of FCylinder (TO-DO)

                                        // print member variables of FCylinder (TO-DO)
};
```

FCylinder.cpp:

```
#include "FCylinder.h"

// default constructor
FCylinder::FCylinder()
{
  // default initialization
  radius = height = 1;
}

// overloaded constructor
FCylinder::FCylinder(double r, double h)
{
  // first assign default value
  radius = height = 1;

  // afterwards, attempt to assign the values of r, h to radius, height
  setRadius(r);
  setHeight(h);

  // NOTE: If setRadius does not work, then radius retains the default value of 1.
  //       If setHeight does not work, then height retains the default value of 1.
}

// copy constructor; I included this to show what a copy constructor does
FCylinder::FCylinder(const FCylinder& fc)
{
  // we're allowed to use the dot operator here because we're within
  // the "same" class

  radius = fc.radius;
  height = fc.height;
}

// deconstructor
FCylinder::~FCylinder()
{
  cout << "Destroyed a FCylinder.\n";
}

//---------------------Getter Methods---------------------//
// returns radius
double FCylinder::getRadius()
{
  return radius;
}

// returns height (TO-DO)
```

```
//----------------------Setter Methods----------------------//
// set radius (as long as r > 0, else keep original)
void FCylinder::setRadius(double r)
{
  if(r > 0)
  {
    radius = r;
  }
  else
  {
    cout << "Attempted to assign invalid radius size! Retaining current radius value.\n";
  }
}

// set height (as long as h > 0, else keep original)
void FCylinder::setHeight(double h)
{
  //TO-DO









}

//----------------------Misc. Methods----------------------//
// returns surface area of FCylinder
double FCylinder::computeSurfaceArea()
{
  return (2 * PI * radius * height) + (2 * PI * radius * radius);
}

// returns volume of FCylinder (TO-DO)









// prints member variables of FCylinder class to console output (TO-DO)
```

main.cpp:

```
/*
  NOTE: When we include "FCylinder.h", we're also including the libraries and namespaces
        it includes (that is: the library "iostream" and the namespace "std")
*/

#include "FCylinder.h"

void class_area();

int main()
{
  cout << "Start of program.";
  cout << "\n-------------------------\n";

  // function made solely to show when the deconstructor is called
  class_area();

  cout << "\n-------------------------\n";

  // custom exit prompt
  cout << "End of program. Press ENTER to continue.";
  cin.ignore(1000, '\n');

  return 0;
}
void class_area()
{
  // example of each type of constructor in use
  FCylinder first_cyl;             //default constructor
  FCylinder second_cyl(7.5, 4.0);  //overloaded constructor
  FCylinder third_cyl(first_cyl);  //copy constructor

  // print out each cylinder (before operations)
  first_cyl.printSelf();
  second_cyl.printSelf();
  third_cyl.printSelf();

  // assign first_cyl's radius and height equal to second_cyl's (use either method)



  // set the radius of third_cyl equal to 14 and height to 13



  //print out each cylinder's surface area and volume (NOTE: Each method returns doubles)




  // print out each cylinder again (after operations)


}
```

Questions (True/False):

1. An object is an instance of a class
2. Classes and structures are exactly the same
3. A deconstructor is called when creating an object
4. You can only have one constructor in a class
5. Constructors and deconstructors do not need to have the same name as the class they belong to
6. Constructors can be called at any given time like a method
7. Member variables in a class are normally public
8. Member variables in a structure are normally public
9. An accessor/getter method of a class should allow an instance of a class change its member variables
10. A mutator/setter method of a class should allow an instance of a class change its member variables
11. Inline functions are good to use for any type of function, be it big or used very often.

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).

2. When passing an object through a function or method, why should it be called by reference (or const reference)? Give two reasons.

3. Why, do you think, I named the class (from the example I gave) *FCylinder* and not just *Cylinder*?

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).

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

Bunch of notes I came up with on classes/structures and general OOP:

- Classes and structures are grouped in a category of programming called Object-Oriented Programming (OOP in short). Here are some major parts to OOP:
  - **Abstraction:** A definition that captures general characteristics without details
  - **Data Hiding:** Restricting access to certain members of an object. The intent is to allow only member functions to directly access and modify the object's data (think of the purpose of *public* and *private*).
  - **Encapsulation:** The bundling of an object's data and procedures into a single entity
- Classes are equivalent to structures in the sense that you can make objects from them
- In a more formal approach: A **class** is a programmer-defined data type used to define objects
- An **object** is an instance of a class. Formally, an object is a software entity that combines data and functions that act on the data in a single unit. The "data in a single unit" part can be described as a class.
- When a function is declared as **inline**, it means the compiler will replace all instances of the function call with the code's body (instead of usually referring to an address).
  - Typically, this works well with small functions/methods used frequently (e.g. getters/setters)
  - Also, whenever you define a method inside of a class, the method is automatically seen as being inline (this means methods made outside of the class cannot be inline)
- To **instantiate** a class/structure means to create an object/instance of it.
- **Method** is a synonym for **class function**
- When stating a member variable or function of a class/structure as *public*, that means that the member variable or method can be accessed by dot operator **outside of the class/structure**.
- When stating a member variable or function of a class/structure as *private*, that means the member variable or method can only be used **within the class/structure**.
- Although there are other differences between structures and classes, the primary difference you must concern yourself with is that structures are **public on default**, whereas classes are **private on default**.
  - This means that, if you were not to specifically type *public* or *private* in the structure/class you're working with, the member variables and methods will be either **public** or **private** (depending on what you're working with)
  - This is the reason we're able to use the dot operator when working with structures (without having to specify that the member variable or method is private/public).
- Although it's typically better to implement your own, C++ implicitly makes the following in a class/structure:
  - Default constructor (NOTE: This **DOES NOT** mean that all member variables will be initialized; only objects inside of the class will be initialized if they exist)
  - Copy constructor (will copy every member variable from one class to another)
  - Assignment operator (same as above except after declaration)
  - Deconstructor (will call any deconstructors of any objects in the class if they exist)