CSE-165 Lab 05

Points: 100

Write a separate file for each of the following tasks.

Zip all your files together and submit the zip file to CatCourses.

----

1. **Getters and Setters (20 points)**

Create a C++ class named Circle in a file called Circle.h. Your class should have three private double variables (x, y, and r). The variables x and y will store the Cartesian coordinates of the center of the circle and the variable r will store the radius.

There should be two constructors, a default constructor that takes no arguments and instantiates a unit circle centered at the origin, and a second constructor that takes in three arguments (for x, y and r) and instantiates a circle of radius r centered at (x, y).

There should be getters and setters for all the private variables, and a getArea() method that calculates and returns the area of the circle. Your code will be tested with the circles.cpp file.

Expected Output:

```
Center:     (0, 0)
Radius:     1
Area:  3.14159

Center:     (1, 2)
Radius:     3
Area:  28.2743

Center:     (5.5, 10.1)
Radius:     20.2
Area:  1281.9
```

CSE-165 Lab 05

Points: 100

2. **Inheritance in C++ (20 points)**

The file Animal.h contains an Animal class that stores the name and age of a generic animal. Besides the appropriate constructors, getters, and setters, it has a function called feed() which prints out the message "Some food, please!"

Dogs are one kind of animal, so we can extend the Animal class to produce a Dog class. Create a Dog class which inherits from the Animal class and change its constructor and destructor to print more appropriate messages and change the feed() function to print a message saying "Dog food, please!"

Your class should be stored in a file called Dog.h. Your solution will be tested with the file animals.cpp.

Expected Output:

```
Creating Generic Animal
Creating Dog
Snoopy is 4 years old.
Dog food, please!
Deleting Dog
Deleting Generic Animal
```

---

3. **More on Inheritance (20 points)**

Suppose we wish to store several different animals in an array. We do not know ahead of time what kind of animals they will be. C++ allows us to create a vector of pointers to Animal, and store in that vector any object of type Animal or a descendant of Animal.

Reuse the Animal.h and Dog.h files from the previous exercise and get the display.h file. Write a C++ program that reads in an integer N. This is followed by N lines, where each line contains a character, a string, and an integer, separated by spaces. The character will either have the value A or D, indicating whether the animal described on this line is a generic animal (A) or a dog (D). The string and the integer describe the animal's name and age, respectively.

CSE-165 Lab 05

Points: 100

For each line of input, instantiate the appropriate object and push it to the vector of pointers to Animal that you have created, using upcasting if necessary. Once you have pushed all the animals to the vector, call the display function, found in display.h, with your vector passed as an argument.

Example Input/Output:

```
2
A Rex 7
Creating Generic Animal
D Snoopy 4
Creating Generic Animal
Creating Dog
Rex
Snoopy
```

---

4. **Counting Objects (10 points)**

Modify the Animal class used in previous exercises by adding a static variable named count. This variable should store the number of instances of the Animal object that have been created in the program countingAnimals.cpp.

Expected Output:

```
Creating Generic Animal
Creating Generic Animal
2
Deleting Generic Animal
Deleting Generic Animal
```

CSE-165 Lab 05

Points: 100

5. **Constructors/Destructors (30 points)**

Extend the Stack class given in the book and used in lab 4 so that it stores doubles (instead of void* pointers).

Add two constructors: the default one that will create an empty stack, and another one that will receive an integer N and will build a stack of N elements, such that the first element is 1.0 and every subsequent element is incremented by 0.1.

Add a destructor that will delete the whole stack by making calls to pop(), and prints as output each element destroyed.

Submit your updated Stack class, which will be tested for correctness with stacks.cpp.

Expected Output:

```
s1
2 1.5 1 0.5
s2
1.3 1.2 1.1 1
end
```