Assignment #5

Due date: Friday, 17 March 2017, 5:59 pm

- For all programming questions below, write your solutions in the dialect of C++ used in class. You may use the following libraries, and no others: iostream, string, vector.
- Store your set of functions in a file named a5.cc.
- You may define your own main function, but we will use our own main to test your code.

You are to create a simple program that manages flocks of sheep. You will need to define several C++ classes to do this: Animal, Sheep, Dog, and Flock. Animal is an abstract base class with two descendants: Sheep and Dog. The Animal class should have exactly the following variables and methods:

```cpp
public:
    virtual ~Animal();
    virtual void speak() const = 0; // A "pure virtual" method
protected:
    Animal(string name);
    string getName() const;
private:
    string name;
```

Note that Animal has only one constructor, and it takes a string argument. Note also that this constructor is protected rather than public. Why? Animal is an abstract base class; no normal client will be creating instances of it, since it is abstract. The only way in which its constructor can be invoked is by a descendant class, which calls the Animal constructor to initialize the parts that are common to all Animals. This is an example of a fairly common idiom in object-oriented programming; there are many, many more as you’ll discover in CS247.

Sheep and Dog should inherit from Animal; they should each define a single constructor of one string argument (as in Animal) and should each provide an appropriate implementation of speak() (see the example output at the end). Sheep, Dog, and Animal should each define a (likely trivial) virtual destructor.

A Flock consists of a single Dog and zero or more Sheep. Flock should have exactly the following variables and methods:

```cpp
public:
    Flock(string dogName);
    virtual ~Flock();
    void addSheep(string name);
    void soundOff();
private:
    Dog *dog;
    vector<Sheep*> sheepList;
```

You create a Flock by passing in the Dog's name at instantiation. You can then add sheep one at a time by passing in each sheep’s name to addSheep(). Assume all names are distinct (i.e., don’t bother checking). We’ll discuss the variables dog and sheepList in the next section. You will also need to define a destructor.

Finally, use the following main program to test your program, and compare the output to the next section:

```cpp
int main(int argc, char* argv[]) {
    Flock *myFlock = new Flock("Spot");
    myFlock->soundOff();
}
```

---

1Yes, this is a short and easy assignment.
2Any resemblance to sheep or dogs whom you know personally is purely coincidental.
myFlock->addSheep ("Daisy");
myFlock->addSheep ("Clover");
myFlock->addSheep ("Estelle");
myFlock->soundOff();
delete myFlock;
myFlock = new Flock ("Rover");
myFlock->addSheep ("Butch");
myFlock->addSheep ("Jonno");
myFlock->soundOff();
delete myFlock;
}

Here is the output I got for my program; yours should look identical, down to the precise spacing of the characters. Note that the animal “speeches” are indented by four spaces, and that each full “flock speech” (try saying that five times quickly) is followed by a blank line.

The flock of 0 sheep speaks!
   Dog Spot says "woof".

The flock of 3 sheep speaks!
   Dog Spot says "woof".
   Sheep Daisy says "baaa".
   Sheep Clover says "baaa".
   Sheep Estelle says "baaa".

The flock of 2 sheep speaks!
   Dog Rover says "woof".
   Sheep Butch says "baaa".
   Sheep Jonno says "baaa".

Within Flock, the dog is represented via a “pointer to Dog” while the list of sheep is represented using a vector of “pointer to Sheep”. To initialize the vector of sheep pointers (the vector itself is a “direct sub-object” of the flock, even tho it contains pointers to objects on the heap), you may use an initializer “sheepList()” in the constructor of Flock, or you can just rely on the fact that sub-objects are created automatically using the default constructor of no arguments if not otherwise initialized.

When you want to add a new sheep, use a pointer and new to instantiate it, and add it to the end of the vector of sheep pointers using the push_back() method of vector.

When you want the flock to sound off, have the dog speak first, then iterate through the vector and get all of the sheep to speak.

You’ll have to think about how to design the destructor. Keep in mind that the vector is a direct subobject of your flock, whereas you have pointers to the dog and sheep. Assume that each flock “owns” its dog and sheep.