CS1020E: Data Structures and Algorithms I

Tutorial 2 – Advanced OO

(Week 4, starting 29 August 2016)

1. Advanced OO

Old McDonald has a farm with some animals.

Each animal has a name, and makes a sound. Some animals are flyers. These animals can fly – once they start to fly, their sound is "flap flap" till they stop. Some flyers are gliders. These animals can glide – if they are flying, once they start to glide, their movement is "whoosh" till they stop.

Your junior has written this code to describe the animals in the farm. However, his code cannot be compiled, as there are 4 errors within these 3 classes. (Only 1 of these errors affect compilation)

```
class Animal {
     string _name; // e.g. Cow
     string _sound; // e.g. moo
  public:
     Animal(string name, string sound) { _name = name; _sound = sound; }
     string getName() { return _name; }
     void makeSound() { cout << _name << " goes " << _sound << endl;}
};
class Flyer : public Animal {
  protected:
     string _name;
     string _sound;
     bool _isFlying;
  public:
     Flyer(string name, string sound)
          : _name(name), _sound(sound), _isFlying(false) {}
     void makeSound() {
          if(_isFlying) cout << getName() << " goes flap flap" << endl;
          else Animal::makeSound();
     }
     void fly() { _isFlying = true; }
     void stop() { _isFlying = false; }
};
class Glider : Flyer {
     bool _isGliding;
  public:
     Glider(string name, string sound)
          : Flyer(name, sound), _isGliding(false) {}
     void glide() { if(_isFlying) _isGliding = true; }
     void stop() { _isFlying = false; _isGliding = false; }
     void makeSound() {
          if(_isGliding) cout << getName() << " goes whoosh" << endl;
          else makeSound();
     }
};
```

(a) What does the `protected` keyword mean? In this example, how is it useful?

(b) Within a member function in the Flyer class, why can `getName()` be invoked?

(c) How is overriding demonstrated here, and how is it useful?

(d) Identify and rectify the 4 errors.

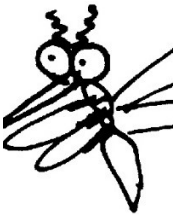
Tip: Try to compile your code! Once the compilation errors are rectified, instantiate objects and test.

2. *Inheritance & Polymorphism*

Related to Q1, Old McDonald still has a farm with some animals. You want to **make use of polymorphism** to allow 5 animals in the farm to `makeSound()`, without having to concern yourself about what type of Animal each is. To simplify things, let's ignore Gliders. There are only Flyers and non-Flyers.

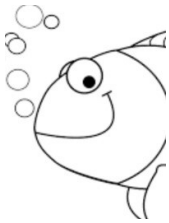
Parrot
"squak"
Perched

Cow
"moo"



Mosquito
"buzz"
Flying

Sheep
"mehh"



Fish
"blurp"

Picture credits: clipartpanda.com

```
class Animal { ... }; // Rectify the problem in (c)
class Flyer : public Animal { ... };
class OldMcDonald {
  private:
     Animal** _farm; // Old McDonald had a farm (still has now)
     const int _size; // Fixed farm size of 5
  public:
     OldMcDonald() {
          /* TODO: Create your farm, an array of Animal* elements */
     }
     ~OldMcDonald() {
          /* TODO: Old McDonald has no (more) farm... */
     }
     void makeSomeNoise() {
          /* TODO: Make sound(s) without looking out for Flyers...! */
     }

...
```

```
    void fillThisFarm() {
        _farm[0] = new Flyer("Parrot", "squak");
        _farm[1] = new Animal("Cow", "moo");
        _farm[2] = new Flyer("Mosquito", "buzz");
        ((Flyer*)_farm[2])->fly();
        _farm[3] = new Animal("Sheep", "mehh");
        _farm[4] = new Animal("Fish", "blurp");
    }
};
```

(a) What is the datatype of `_farm[0]`? Why can a pointer to Flyer be assigned to `_farm[0]`?

(b) Why can't `((Flyer*)_farm[2])->fly()` be replaced with `_farm[2]->fly()`?

(c) With Animal and Flyer classes from Q1, why will polymorphism not work? Make the necessary change.

(d) Solve the problem, ensuring that the sounds output by each animal are correct. The output of `makeSomeNoise()` should be:

Parrot goes **squak**
Cow goes moo
Mosquito goes **flap flap**
Sheep goes mehh
Fish goes blurp

- Practise consistently ☺ -

Revise a concept
Test understanding
Code to confirm