CS1020E Tutorial + Lab 02

Mark NG

a0116298@u.nus.edu
http://mollymr305.github.io

September 2, 2016

Question 1: Advanced OOP

Basic Idea: `Animal` $\supseteq$ `Flyer` $\supseteq$ `Glider`

We first answer the following,

Part (d): Identify and rectify the 4 errors (in the code provided).

Question 1: Advanced OOP

Animal (Original).

```
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;
  }
};
```

Question 1: Advanced OOP

Animal (Edited).

```
class Animal { // no corrections required for Animal class
  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;
  }
};
```

Question 1: Advanced OOP

Flyer (Original).

```
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; }
};
```

Question 1: Advanced OOP

Flyer (Edited).

```
class Flyer : public Animal {
protected:
  // Correction 1: no need for shadowed variables
  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; }
};
```

Question 1: Advanced OOP

Flyer (Edited).

```
class Flyer : public Animal {
protected:
  // Correction 1: no need for shadowed variables
  bool _isFlying;
public:
  Flyer(string name, string sound)
    : Animal(name, sound), _isFlying(false) {}
    // Correction 2 (above): call the parent constructor
  void makeSound() {
    if(_isFlying) cout << getName() << " goes flap flap" << endl;
    else Animal::makeSound(); }
  void fly() { _isFlying = true; }
  void stop() { _isFlying = false; }
};
```

Question 1: Advanced OOP

Glider (Original).

```
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();
  }
};
```

Question 1: Advanced OOP

Glider (Edited).

```
class Glider : public Flyer {// Correction 3: public Inheritance
  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();
  }
};
```

Question 1: Advanced OOP

Glider (Edited).

```
class Glider : public Flyer {// Correction 3: public Inheritance
  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 Flyer::makeSound();
    // Correction 4: avoid infinite recursion
  }
};
```

Question 1: Advanced OOP

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

Question 1: Advanced OOP

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

Answer. `protected` means only subclasses can access this member, but not other classes. This is useful because we wish to encapsulate (i.e hide) whether a `Flyer` object is ‘flying’ but allow a `Glider` object (subclass) to read and modify this data.

Question 1: Advanced OOP

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

Question 1: Advanced OOP

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

Answer. The subclass inherits protected and public features of the superclass `Animal`. Note that `getName()` is equivalent to `Animal::getName()` or `this->Animal::getName()`.

Question 1: Advanced OOP

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

Question 1: Advanced OOP

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

Answer. `Flyer` and `Glider` use different implementations of `makeSound()` and `stop()`. Subclass methods will override parent methods if implemented.

Question 2: Inheritance and Polymorphism

Code Part 1 (Original).

```
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...! */
  }
```

Question 2: Inheritance and Polymorphism

Code Part 2 (Original).

```
/*
continued from previous slide ...
*/
  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");
  }
};
```

Question 2: Inheritance and Polymorphism

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

Question 2: Inheritance and Polymorphism

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

Answer. `_farm[0]` is an Animal pointer (i.e. `Animal*`). In `C++`, we can substitute a superclass-typed object with a subclass-typed object when assigning to a pointer.

Question 2: Inheritance and Polymorphism

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

Question 2: Inheritance and Polymorphism

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

Answer. `_farm[2]` is an Animal pointer. We need to downcast it into a Flyer pointer (i.e. `Flyer*`), before dereferencing it.

Question 2: Inheritance and Polymorphism

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

Question 2: Inheritance and Polymorphism

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

Answer. We need to insert `virtual` in `Animal::makeSound()`, so that `_farm[2]->makeSound()` uses `Flyer::makeSound()` when necessary.

Question 2: Inheritance and Polymorphism

Part (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

Question 2: Inheritance and Polymorphism

Part (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

Answer. First insert `virtual` in `Animal::makeSound()`, then add

```
void makeSomeNoise() {
  for (int i = 0; i < _size; i++) _farm[i]->makeSound();
}
```

End of Tutorial Discussion

Note: Detailed solutions (i.e. the file `T2_ans.pdf`) will be released soon at

`http://www.comp.nus.edu.sg/~stevenha/cs1020e.html`

Let's take a short break!

Exercise 1

(1) Intersection: 40%

- Remember to store rectangle information properly!
  - x_1, y_1: Lower-left corner.
  - x_2, y_2: Upper-right corner.
  - Note that the input coordinates might not be of this form (how to resolve this)?
- Question: can you find a simple, equivalent condition for rectangles to have an empty intersection?

Exercise 2

(2) Closest Rectangles: 40%

- How to find the center of a rectangle, given $(x_1, y_1), (x_2, y_2)$?
- The Euclidean distance d between two points (a_1, b_1) and (a_2, b_2) in $\mathbb{R}^2$ is given by

$$d = \sqrt{(a_1 - a_2)^2 + (b_1 - b_2)^2}.$$

Is there any difference in using d above, and say

$$d^2 = (a_1 - a_2)^2 + (b_1 - b_2)^2?$$

Exercise 3 & 4

(3) Combine Rectangles: 15%, (4) Nearest Rectangles: 5%

- Can't say much.
- In general, `C++` has some useful libraries (this applies to previous/future exercises as well!).
- This might make your programming task a little easier.

Useful C++ libraries.

Using `cmath` and `iomanip`.

- http://www.cplusplus.com/reference/cmath/
- http://www.cplusplus.com/reference/iomanip/
- http://www.cplusplus.com/reference/algorithm/

Challenge.

Task. Write a program that takes in an integer, and outputs the sum of all it's digits.

Example. The input 2016 should produce output 9.

Kattis Online Judge

Let's try solving these 'easy' problem(s):

`https://open.kattis.com/problems/matrix`
`https://open.kattis.com/problems/easiest`

Any Questions?

See you next week!