SFU CMPT-212 2008-1 Topic: Multiple Inheritance

Ján Maňuch

E-mail: jmanuch@sfu.ca

Friday 29th February, 2008

Multiple inheritance

- in C++ it's possible to derive a class from several base classes (they have to be different)

  Example:

```
1 class D : public A, private B, private C {
2 // same as class D : public A, B, C { !
3   ...
4 };
```

  – class `D` *is a* special type of class `A` and it *contains* classes `B` and `C`

- assume that each class `A`, `B` and `C` contains `void show();` as a public member function

  Question: Which function is called in the following code:

```
1 D object;
2 object.show();// ambiguous call
```

  – even though `B::show()` and `C::show()` are private methods of class `D` (*example:* multiple.cpp)

Solutions:

1. specify which `show()` method to call:

```
1 D object;
2 object.A::show(); // ok
3 object.C::show(); // compiler error C::show() is priva
```

2. redefine `show()` in class `D` and make it access `A::show()` (or whichever of the three methods you want):

```
1 class D : public A, private B, private C {
2  public:
3    void show() { A::show(); }
4    // hides all derived show() functions
5 };
6 ...
7 D object
8 object.show(); // calls D::show()
```

3. use containment instead

```
1 class D : public A {
2   B b;
3   C c;
4 ...
5 };
6 ...
7 D object;
8 object.show(); // calls A::show()
9 // B::show() has to be called through data member b
```

Remark:

- can be used only if there is only one publicly inherited base class

Example: vehicle.cpp

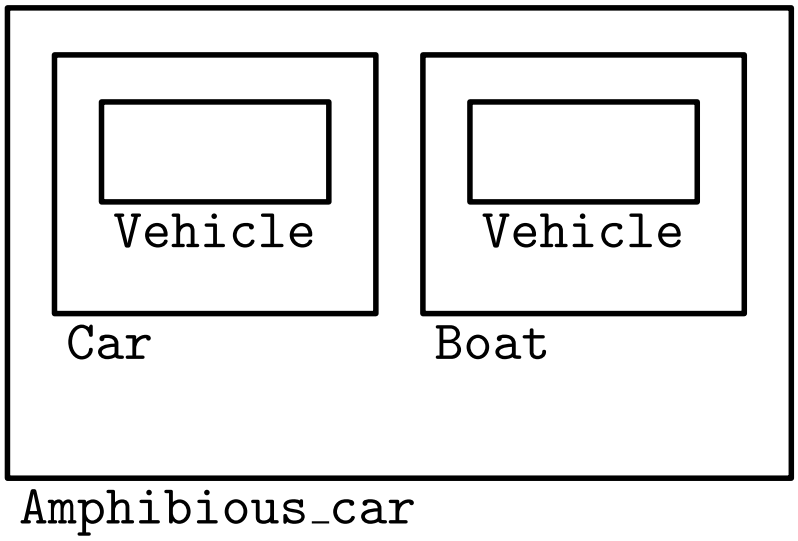
```
1   class Vehicle {
2     int weight;
3    public:
4     Vehicle(int w) : weight(w) {}
5     int getWeight() { return weight; }
6   };
7
8   class Car : public Vehicle {
9     int engine;
10   public:
11    Car(int e, int w) : engine(e), Vehicle(w) {}
12    int getEngine() { return engine; }
13  };
14
15  class Boat : public Vehicle {
16    int material;
17   public:
18    Boat(int m, int w) : material(m), Vehicle(w) {}
19    int getMaterial() { return material; }
20  };
```

```
21  class Amphibious_car : public Car, public Boat {
22   public:
23    Amphibious_car(int e, int m, int w)
24      : Car(e,w), Boat(m,w) {}
25  };
26
27  int main() {
28    Amphibious_car beta(3240,2,2000);
29    cout << beta.getEngine();   // ok
30    cout << beta.getMaterial(); // ok
31    cout << beta.getWeight();   // error - ambiguous call
32  }
```

- *Question:* what's the problem? why is the `getWeight()` call ambiguous?
- *Answer:* class `Amphibious_car` contains *2 copies* of class `Vehicle`

```
1  cout << beta.Car::getWeight(); // ok
2  // applies Vehicle::getWeight() on Vehicle-subobject
3  // contained in Car-subobject
```

Illustration:



- *Another problem:* *upcasting* is ambiguous:

```
1    Vehicle *pveh = &beta; // ambiguous
2
3    Vehicle *pveh1 = (Car *) &beta;  // copy in Car
4    Vehicle *pveh2 = (Boat *) &beta; // copy in Boat
```

- we actually want to have just *one copy* of `Vehicle` in the class `Amphibious_car`

  Question: Is it possible?

Virtual base class

Example:

```
1 class Car : public virtual Vehicle { ... };
2 class Boat : public virtual Vehicle { ... };
3 class Amphibious_car : public Car, public Boat { ... };
```

- `Vehicle` is called a *virtual base class*
- `Amphibious_car` contains just one copy of `Vehicle`
- when a class inherits a particular *base class* through several paths, the class has
  - **1** base class subobject for **all** paths inheriting the *base class* virtually (keyword `virtual` used when deriving from the base class)
  - **1** base class subobject for **each** path inheriting the *base class* non-virtually (keyword `virtual` not used)

Constructor rules for virtual base class

- according to the normal rules, the `Vehicle` subobject of class `Amphibious_car` would be constructed twice:

```
1   Amphibious_car(int e, int m, int w)
2   : Car(e,w), Boat(m,w) {}
```

  as both `Car(e,w)` and `Boat(m,w)` should call constructor `Vehicle(w)`

- to avoid this: according to the *new rules* for virtual base classes, the *virtual base* class is **not** constructed at all through the derivation paths
  Example: virtual.cpp

- that is: if no constructor for the *virtual base class* is specified in the *member initializer list*, the *default constructor* of the *virtual base class* is used

Example: vehvirtual.cpp

Supplemental material:

- `http://www.parashift.com/c++-faq-lite/multiple-inherit`

Sample questions

- What's wrong with the following code?

```
1  class A {
2   public:
3    void show() {}
4  };
5  class B {
6   public:
7    void show() {}
8  };
9  class D : public A, private B {
10 };
11 ...
12 D object;
13 object.show();
```

- Consider the following code:

```
1  class A {};
2
3  class B : public A {};
4  class C : virtual public B {};
5  class D : virtual public B {};
6  class E : virtual public A {};
7
8  class F : public C, public D, public E {};
```

  Consider an object of class `F`. How many subobjects of

  (a) class `A`;

  (b) class `B`

  it contains?

- How are constructor rules for virtual base classes different from the ordinary constructor rules?