

601.220 Intermediate Programming

Spring 2023, Day 33 (April 14th)

Today's agenda

- Review exercise 32
- Day 33 recap questions
- Exercise 33

Reminders/Announcements

→ HW7 is due **this evening**

- Final project:
 - You should have access to your team repository (check by going to <https://github.com/orgs/jhu-ip/repositories>)
 - Project description and starter files are available
 - We'll introduce the final project on Monday, but please do get started by (at least) reading the project description
 - Planning times when your team can meet to work together is also highly recommended

Exercise 32 review



In B's constructor:

```
a = 27;
```

The member variable `a` is `private` in the base class `A`, so `B`'s constructor can't access it directly.

Exercise 32 review

A aptr = new B();*

In main1.cpp:

`aobj.d = 17.5;` *// d is protected in A, main cannot
// access directly*

`aptr->setb(15);` *// even though aptr is pointing to an object
// of type B, its type is A* (pointer to A)
// and A does not have a setb member function*

`bobj = a5;` *// B's assignment operator requires an
// argument of type const B& (reference to
// const B), but a5's type is A*

Exercise 32 review

After making the show member function virtual in A (note that additional output is generated):

```
--- orig_output.txt 2022-07-12 09:53:44.533488783 -0400
+++ revised_output.txt 2022-07-12 09:53:44.533488783 -0400
@@ -6,10 +6,14 @@
```

```
A is 3
test A
```

```
+B is 2
+test B
```

```
non-virtual display A
```

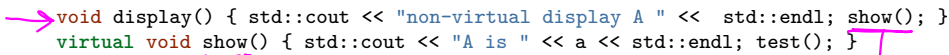
```
A is 3
test A
```

```
+B is 2
+test B
```

```
A obj killed
A is 10
```

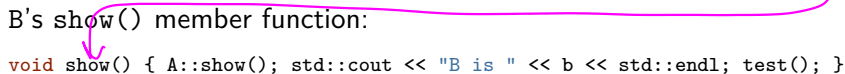
Exercise 32 review

A's `display()` and `show()` member functions:



```
void display() { std::cout << "non-virtual display A " << std::endl; show(); }  
virtual void show() { std::cout << "A is " << a << std::endl; test(); }
```

B's `show()` member function:



```
void show() { A::show(); std::cout << "B is " << b << std::endl; test(); }
```

The call to `show()` in `A::display()` now resolves to `B::show()` when called on a B object. `B::show()` calls `A::show()` directly, then prints additional output ("B is _"), then calls `test()`, which is a non-virtual member function in A, but resolves to `B::test()`, because B also defines a member function of the same name, and the call is in `B::show()`.

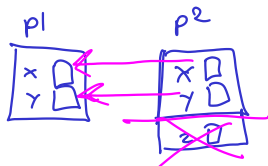
Day 33 recap questions

- ❶ Explain what object slicing is in C++.
- ❷ What is the override specifier in C++?
- ❸ Explain what function hiding is in C++?
- ❹ In C++, how do you make an abstract class?
- ❺ Can we create an object from an abstract class?

1. Explain what object slicing is in C++.

An assignment of a derived class object to a base class object will “slice off” the fields of the derived class object, because they can't be stored in the base class object.

E.g.:



// assume Point3D derives from Point2D

```
Point2D p1(3.0, 4.0);
```

```
Point3D p2(5.0, 6.0, 7.0);
```

```
p1 = p2; // only the x and y values of p2 are  
         // copied to p1
```

Value semantics and inheritance

Value semantics (copying and assigning object contents) tends not to be particularly useful in class hierarchies (where inheritance is used to define “is-a” relationships between classes.)

* It is not uncommon for base and derived classes to prohibit the use of the copy constructor and assignment operator by making them private.

2. What is the override specifier in C++?

The `override` specifier can be used in a derived class to indicate that a member function is intended to override a member function in the base class. If the derived class function (marked with `override`) does not *actually* override a base class function, the compiler reports an error.

Why override is (sometimes) useful

The problem override is designed to solve:

- base class defines a virtual member function
- derived class defines a virtual member function intended to override the base class function, but it messes up somehow (e.g., wrong number or type(s) of parameters), so that the derived class function doesn't actually override the base class function

One reason this could happen is because someone changes the definition of the member function in the base class.

Opinion about override

Virtual member functions in base classes should be pure virtual (i.e., abstract.)

If they are, then the override specifier isn't that essential, because if a derived class doesn't override all of the pure virtual member functions in the base class, it won't be instantiable.

3. Explain what function hiding is in C++?

Function hiding occurs when a derived class defines a member function with the same name as member function(s) in the base class.

This “hides” the identically-named functions in the base class.

Note that those functions could still be called using the scope resolution operator (`::`). E.g.:

```
Base::foobar(123, 'a'); // if function(s) called "foobar"  
                        // in Base would normally be hidden,  
                        // this would call call  
                        // Base::foobar(int, char)
```

4. In C++, how do you make an abstract class?

A class that has at least one pure virtual member function is an abstract class.

E.g.:

```
class Animal {  
    virtual ~Animal() {}  
    virtual void vocalize() = 0;  
};
```

Opinion: virtual functions in base classes should always be pure virtual. The reason is that if derived classes will be overriding the function to implement varying behavior, then there is probably nothing useful that the base class can do to define functionality for the member function.

5. Can we create an object from an abstract class?

No. Example:

```
class Animal {  
public:  
    virtual ~Animal() { }  
    virtual void vocalize() = 0;  
};  
  
class Dog : public Animal {  
    virtual ~Dog() { }  
    virtual void vocalize() { std::cout << "woof\n"; }  
};  
  
// ...  
  
Animal *a = new Animal(); // compile error  
Dog *d = new Dog();       // fine
```



Exercise 33

- Abstract classes, pure virtual member functions
- Talk to us if you have questions!

Notes

Notes

Notes

Notes

Notes