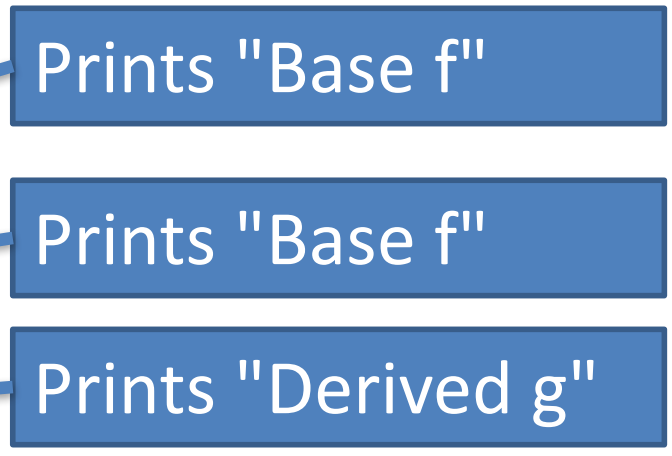
Inheritance wrapup, polymorphism

- Warmup: Get Dropbox code (apr-21-start.cpp).
  Add two new car types to the race by defining two new classes that inherit from car:
- A `racecar`:
  - can accelerate at 10 mph every second, rather than 5 mph every second
  - all race cars have a top speed of 200 mph.
- A `clunker`:
  - still accelerates at 5 mph per second.
  - top speed of 50 mph.
  - But after calling drive() 3 times, the car dies, immediately stops, can't be fixed, and you have to call your parents to pick you up.
- Optional: `truck`
  - Accelerates 2 mph per second, but only on every other call to drive.

- When a class is a particular kind of another class, use **inheritance**.

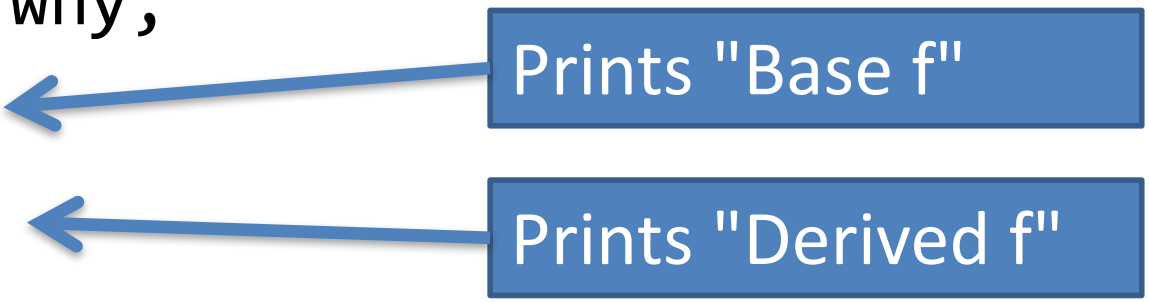
```
class X { void f(); };
class Y : public X { void g(); };
void X::f() { cout << "Base f"; }
void Y::g() { cout << "Derived g"; }
```

```
X ex; Y why;
ex.f();
why.f();
why.g();
```



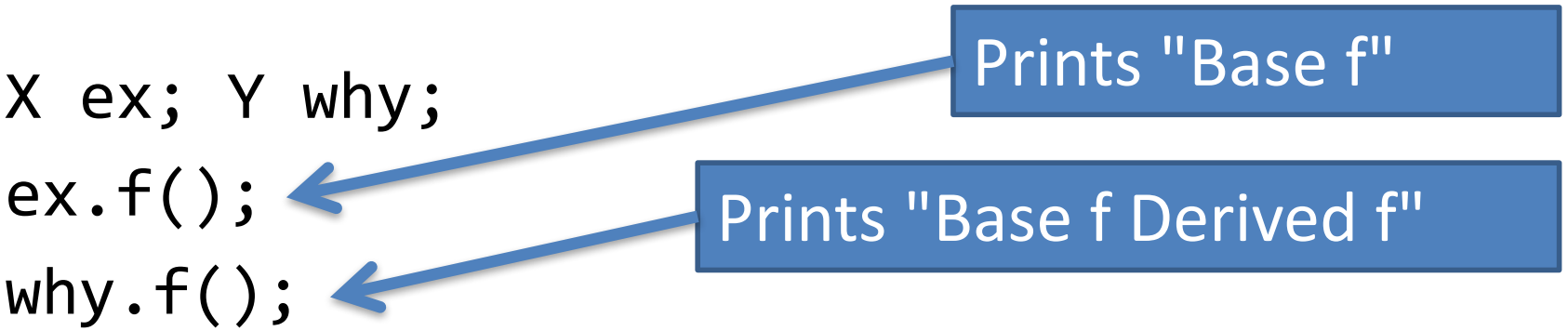
- A derived class is allowed to **override** methods in the base class.

```
class X { void f(); };
class Y : public X { void f(); };
void X::f() { cout << "Base f"; }
void Y::f() { cout << "Derived f"; }

X ex; Y why;
ex.f();
why.f();
```



- If a derived class overrides a method, the overridden method code can still call the base class version of the method if needed.

```
class X { void f(); };
class Y : public X { void f(); };
void X::f() { cout << "Base f"; }
void Y::f() { X::f(); cout << "Derived f"; }

X ex; Y why;
ex.f();
why.f();
```



Polymorphism

- From Greek πολύς, polys, "many, much" and μορφή, morphē, "form, shape."

- ***The ability for a derived class to substitute in code where a base class is used.***

- This concept is not new:

```
void f(double x) {
  /* do something */;
}

int main() {
  int y = 3;
  f(y);
}
```

C++ will automatically convert a derived class object to a base class object when required.

Typical situations:

- Variable assignment
- Calling a function

Caveat emptor

- When C++ automatically converts a derived-class object to a base-class object, the converted object loses all extra abilities the derived class had.

```
class A {
  public:
  void f() { cout << "base f"; }
};
class B : public A {
  public:
  void f() { cout << "derived f"; }
  void g() { cout << "derived g"; }
};
int main() {
  A a;  a.f();
  B b;  b.f();  b.g();
  A copy = b;  copy.f();  copy.g();
}
```

Caveat emptor

- When C++ automatically converts a derived-class object to a base-class object, the converted object loses all extra abilities the derived class had.
- **Copying** the derived-class object into a base-class object means the copy only has the abilities of the base class.
- **How do we avoid making copies?**

Step 1: Use Pointers

- ***A base-class pointer can point to a derived-class object.***
- Because no copy is made, the pointer still points at an object that has all the abilities of the derived class.
- The base-class pointer will still only let you (directly) call functionality specified by the base class.

Step 2: Use virtual methods

- Class methods can be tagged with the keyword "virtual."
- When a virtual method is called using a pointer, C++ uses the version of the method that belongs to **the type of the object being pointed at**, not **the type of the pointer**.