Data and Function Inheritance Implications for C++

Now back to inheritance. Given

```
class MyClass : public ParentClass {
    // class definition
}
```

data inheritance implies that **a `MyClass` has all fields of a `ParentClass`**, and

function inheritance implies that **we can invoke any member function for `ParentClass` on a `MyClass`**.

Compilers Implicitly Convert `Derived*` to `Base*`

What if both `MyClass` and `ParentClass` define member function aFunc?

Let's say that we have three variables

```
MyClass m;
ParentClass p;
ParentClass* ptr = &m;  // Huh?
```

Compilers can implicitly convert pointers to derived classes to pointers to base classes.

Remember that in many cases, no instructions are needed for such conversion!

Compilers Use the Type of the Object to Choose a Class

Now back to the question.

```
MyClass m;
ParentClass p;
ParentClass* ptr = &m;

m.aFunc (); // calls MyClass::aFunc
p.aFunc (); // calls ParentClass::aFunc
ptr->aFunc (); // ParentClass::aFunc
```

But `*ptr` is actually a `MyClass` instance!

Use `virtual` to Create Subtype-Specific Functions

If you **want the behavior that we implemented in C** using tables of function pointers, **mark `aFunc` as `virtual` *** in `ParentClass` ***.**

```
virtual void aFunc (void);
```

C++ compilers automatically
- define virtual function pointer tables,
- fill them for each class, and
- place a pointer to the table into each class instance (usually at the start of the structures).

(`virtual` is the default in Java, by the way.)

