A Base Class pointer can point to a derived class object. Why is the vice-versa not true?

A Base Class pointer can point to a derived class object. Why is the vice-versa not true without casting? Logically a base class would not have enough information of the derived class but a derived class should have the information of the base class as well. I am missing some basics here.

c++

asked Feb 8 '11 at 19:06
Zuzu
1,048 ● 4 ● 15 ● 16

9 Answers

If I tell you I have a dog, you can safely assume that I have a pet.

If I tell you I have a pet, you don't know if that animal is a dog, it could be a cat or maybe even a giraffe. Without knowing some extra information you can't safely assume I have a dog.

similarly a derived object is a base class object (as it's a sub class), so it can be pointed to by a base class pointer. However, a base class object is not a derived class object so it can't be assigned to a derived class pointer.

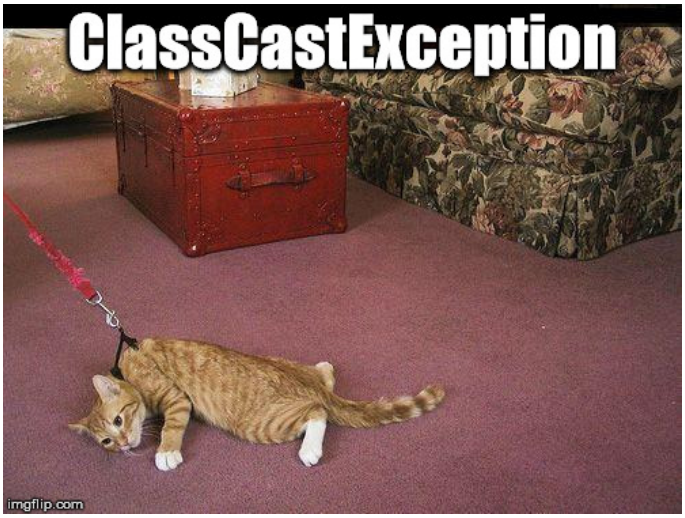
(The creaking you will now hear is the analogy stretching)

Suppose you now want to buy me a gift for my pet.

In the first scenario you know it is a dog, you can buy me a leash, everyone is happy.

In the second scenario I haven't told you what my pet is so if you are going to buy me a gift anyway you need to know information I haven't told you (or just guess), you buy me a leash, if it turns out I really did have a dog everyone is happy.

However if I actually had a cat then we now know you made a bad assumption (cast) and have an unhappy cat on a leash (runtime error).



edited May 23 '17 at 0:41
Sajuukhar
3 ● 2

answered Feb 8 '11 at 19:11
jk.
10.9k ● 4 ● 28 ● 47

3 +1: Best answer so far, IMHO. A simple analogy that intuitively illustrates the problem. – Oliver Charlesworth Feb 8 '11 at 19:43

2 "For every problem there is one solution which is simple, neat, and wrong." — H.L. Mencken. Intuitive answers often fall into this category, especially OO described in "real-world" terms. Sometimes a base class pointer does point to an object of derived class type and that is why (as the question points out) you can cast. – Fred Nurk Feb

'11 at 20:37

3 A dog is an animal, but an animal is not necessarily a dog, it might be a giraffe. What does this analogy tell? If Animal is considered to be the base class and Dog/Giraffe are the derived classes.Since Dog/Giraffe are indeed animals i.e the child class should be able to point to the base class? Should that not be the case? – Zuzu Feb 9 '11 at 11:32

1 yes if you are casting then you are explitly telling the compiler to shut up cos you have extra information it doesn't. I think I can expand the animal metaphor to add the pointers and casting in a bit - bear with me – jk. Feb 9 '11 at 13:34

We have two objects.

```
class A {
   int a;
};

class B : A {
   int b;
};
```

Allocate an instance of `B`. We can interface with that as either an `A*` or a `B*`.

Allocate an instance of `A`. If we were to cast it to a `B*`, should there be space allocated for the member `b`?

answered Feb 8 '11 at 19:10

Bill Lynch
56.2k ● 10 ● 88 ● 133

+1, I was going to post the very same example. – casablanca Feb 8 '11 at 19:11

Uh, because the base class is not a derived class.

When you have a valid pointer to a type, then you are saying that the object pointed to will have certain data in certain locations so that we can find it. If you have a pointer to a derived object, then you are guaranteeing that the pointed-to object contains all of Derived's data members- but when you point to a Base, then it infact doesn't have that and Bad Things Happen™.

However, Derived is guaranteed to have all of the Base data members in the same locations. That's why a pointer to Base can actually point to Derived.

answered Feb 8 '11 at 19:09

Puppy
119k ● 24 ● 183 ● 390

1 All derived classes are not "guaranteed to have all of the Base data members in the same locations." – Fred Nurk Feb 8 '11 at 20:07

Relative to the pointer in question, I believe that they are. – Puppy Feb 8 '11 at 20:18

Not if you complicate it enough with multiple inheritance and virtual bases... – Bo Persson Feb 8 '11 at 23:08

@Bo: That will only make things like casting and function calls more complex. Fundamentally, pointer-based inheritance works purely on the basis that when I have a pointer to Base, that the vtable and member variables exist where they would exist in a regular instance of Base. – Puppy Feb 9 '11 at 0:58

This is valid, because a tiger is an animal:

```
    Animal * pAnimal = new Tiger();
```

This is not valid, because it is not true that the object is a poison dart frog.

```
    PoisonDartFrog * pPoisonDartFrog = new GenericFrog();
```

answered Feb 8 '11 at 19:10

Andy Thomas
61.8k ● 9 ● 67 ● 119

1 He asks *why*, not *what*. – Puppy Feb 8 '11 at 19:11

@DeadMG: You caught me mid-edit. I think an example helps communicate the concept. – Andy Thomas Feb 8 '11 at 19:13

1 Would today's two downvoters, three years later, care to comment on what they see as flawed in this answer? – Andy Thomas Nov 11 '14 at 20:17

Because a derived class includes everything that is in the base class. But a base class does not include everything that is in the derived class.

Type casting a base class to a derived class is not recommended: What happens if you try to access members that are not part of the base class?

edited Jun 23 '16 at 22:43

answered Feb 8 '11 at 19:18

Jonathan Wood
40.7k • 51 • 177 • 285

Because C++ is a statically typed language, and allowing implicit Base-to-Derived conversions would break the type system. Bjarne Stroustrup did not want any "message not understood" runtime errors.

answered Feb 8 '11 at 19:13

fredoverflow
152k • 60 • 296 • 578

Is C statically typed? – Fred Nurk Feb 8 '11 at 20:13

At the time Bjarne Stroustrup started work on C with classes, C was not even able to type-check a function call :) But yes, C is statically typed, albeit not very strongly. – fredoverflow Feb 8 '11 at 20:17

1 That's why I don't understand justifying the prevention of the pointer conversion in terms of being statically typed. There are plenty of things that can go wrong at runtime ("message not understood"?), such as dereferencing a null pointer, and Stroustrup's goals appear to lie in the opposite direction of trusting the programmer at runtime. – Fred Nurk Feb 8 '11 at 20:24

@Fred: Maybe you have never been exposed to a dynamically typed language, I don't know. If C++ weren't statically typed, you could just say `p->any_method_name(42, "hello", test);`, and the compiler would not do any checking at compile-time if there is such a method that accepts three parameters, because he would not know that p points to an instance of a class of a statically known type. The checking would be deferred to runtime, which is more expensive and obviously less reliable (but on the plus side, more flexible -- you don't need subtyping or generics, for example). – fredoverflow Feb 8 '11 at 20:32

I know Python, among other languages. – Fred Nurk Feb 8 '11 at 20:35

```
class Base
{
public:
    int a;
}

class Derived : public Base
{
public:
    float b;
}

Base * pBase = new Base();
pBase->a = 7; // setting the value of a in the base

// make a pDerived that points to the SAME DATA as pBase
Derived * pDerived = pBase;
pDerived->a = 5; // this would be okay, base has a public member 'a'
pDerived->b = 0.2f; // error pBase has no data member b and pDerived
                    // points to the SAME DATA as pBase
```

answered Dec 11 '13 at 20:50

YoungJohn
748 • 6 • 11

Because a base class pointer can point to an instance of the base class or any derived type. A derived pointer can only point to that derived type or any subclass of it.

```
struct Base {};
struct Derived : Base {};
struct Derived2 : Base {};
Base* p = new Derived(); //Fine, Derived inherits from Base
Derived* d = new Base(); //Not fine, Base is not an instance of nor derived from
Derived.
Derived* d2 = new Derived2(); // Also not fine, Derived2 derives from Base, but
is not related to Derived.
```

As far as the why goes: In general the base pointer is more general than the derived pointer. As such it knows less about the inherited type. A derived pointer cannot be assigned a pointer to a base type

without casting simply because it cannot tell if the base pointer is of the Derived type or one of its children.

answered Feb 8 '11 at 19:10

Washu

787 ● 5 ● 6

He asks *why*, not *what*. – Puppy Feb 8 '11 at 19:12

The example was there to help communicate why, and furthermore if you had bothered to read the comments you would notice that it does say WHY. Added some clarification after the point to help. – Washu Feb 8 '11 at 19:15

2 @AndyThomas-Cramer: Trading upvotes with another user is against the spirit of the system and I'm shocked to see a 4k rep user so blatantly advocating such abuse. – Fred Nurk Feb 8 '11 at 19:49

@Fred Nurk - It was not my intent to suggest a quid pro quo. My upvote was unconditional. – Andy Thomas Feb 8 '11 at 20:19

If assign an address from a base class pointer into a derived class pointer, you can potentially assign a base class object to a derived class pointer. You run the risk of accessing derived class members when you don't have a derived class. Whereas derived class methods would work on a base class, they would only do so if the method didn't access derived class member data.

That's a huge risk.

So we force you to cast so that you have to acknowledge the disclaimer that says (you may make a stupid mistake, please be careful).

answered Feb 8 '11 at 20:21

Lee Louviere

4,150 ● 20 ● 48