24.1 — Introduction to inheritance

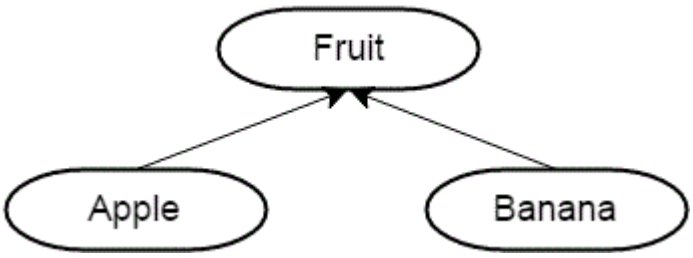
ALEX[1] JUNE 5, 2024

In the last chapter, we discussed object composition, where complex classes are constructed from simpler classes and types. Object composition is perfect for building new objects that have a "has-a" relationship with their parts. However, object composition is just one of the two major ways that C++ lets you construct complex classes. The second way is through inheritance, which models an "is-a" relationship between two objects.

Unlike object composition, which involves creating new objects by combining and connecting other objects, inheritance involves creating new objects by directly acquiring the attributes and behaviors of other objects and then extending or specializing them. Like object composition, inheritance is everywhere in real life. When you were conceived, you inherited your parents genes, and acquired physical attributes from both of them -- but then you added your own personality on top. Technological products (computers, cell phones, etc...) inherit features from their predecessors (often used for backwards compatibility). For example, the Intel Pentium processor inherited many of the features defined by the Intel 486 processor, which itself inherited features from earlier processors. C++ inherited many features from C, the language upon which it is based, and C inherited many of its features from the programming languages that came before it.

Consider apples and bananas. Although apples and bananas are different fruits, both have in common that they *are* fruits. And because apples and bananas are fruits, simple logic tells us that anything that is true of fruits is also true of apples and bananas. For example, all fruits have a name, a color, and a size. Therefore, apples and bananas also have a name, a color, and a size. We can say that apples and bananas inherit (acquire) all of the properties of fruit because they *are* fruit. We also know that fruit undergoes a ripening process, by which it becomes edible. Because apples and bananas are fruit, we also know that apples and bananas will inherit the behavior of ripening.

Put into a diagram, the relationship between apples, bananas, and fruit might look something like this:

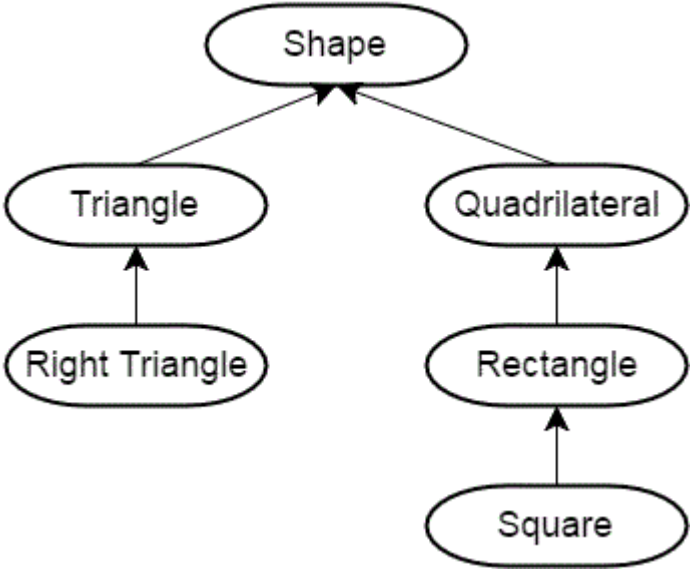


This diagram defines a hierarchy.

Hierarchies

A hierarchy is a diagram that shows how various objects are related. Most hierarchies either show a progression over time (386 -> 486 -> Pentium), or categorize things in a way that moves from general to specific (fruit -> apple -> honeycrisp). If you've ever taken biology, the famous domain, kingdom, phylum, class, order, family, genus, and species ordering defines a hierarchy (from general to specific).

Here's another example of a hierarchy: a square is a rectangle, which is a quadrilateral, which is a shape. A right triangle is a triangle, which is also a shape. Put into a hierarchy diagram, that would look like this:



This diagram goes from general (top) to specific (bottom), with each item in the hierarchy inheriting the properties and behaviors of the item above it.

A look ahead

In this chapter, we'll explore the basics of how inheritance works in C++.

Next chapter, we'll explore how inheritance enables polymorphism (one of object-oriented programming's big buzzwords) through virtual functions.

As we progress, we'll also talk about inheritance's key benefits, as well as some of the downsides.

Next lesson
24.2 Basic inheritance in C++

2

Back to table of contents

3

Previous lesson
23.x Chapter 23 summary and quiz

4

5

B U URL INLINE CODE C++ CODE BLOCK HELP!

Leave a comment...

Name*

Email*

Notify me about replies:

POST COMMENT

Find a mistake? Leave a comment above!

Avatars from https://gravatar.com/[8] are connected to your provided email address.

38 COMMENTS

Newest

Nidhi Gupta
May 5, 2025 1:19 pm PDT

this inheritance examples are pretty cool!These all connect to linked lists , like something leading to another thing and then the last point is nulptr.

0 Reply

Jhon Doe
October 14, 2024 10:14 pm PDT

Shei

0 Reply

name
April 27, 2024 5:02 am PDT

thanmks!

0 Reply

Timon
March 26, 2024 3:59 pm PDT

Time to inherit this knowledge

23 Reply

ababa

November 15, 2023 10:13 pm PST

Maybe a non-problem, but I will point out anyway.
I think a lot (most?) non-native English speakers don't know that "red delicious" is a type of apple, and there is a risk of confusion (`fruit -> apple -> red delicious` is non-sensical if you think that `red` and `delicious` are somehow part of the hierarchy).
Non-native speakers would also most likely not be familiar with "granny smith", for example, but at least that would prompt them to think "what is granny smith??", and they are a Google search away from finding it out.

2 Reply

Alex Author

Reply to ababa [9] November 16, 2023 7:37 pm PST

Changed to honeycrisp, which will hopefully be less ambiguous than red delicious.

4 Reply

Suku_go

July 24, 2023 11:35 pm PDT

OH YEAH!!! INHERITANCE BABYYYY!!!

Last edited 1 year ago by Suku_go

6 Reply

Alper Ozge Gur

May 24, 2019 11:07 am PDT

The figure Rectangle<--Square caused a little shock on my side since this is the example used most frequently about how NOT to use inheritance. (I am coming from java and saw no clue if this is not the case with C++)

No offence to authors is intended, these lessons are great (in terms of both depth of insight provided and yet not so hard to comprehend)by the way.

1 Reply

Alex Author

Reply to Alper Ozge Gur [10] May 29, 2019 12:39 pm PDT

It's the case for C++ as well. Do keep in mind that the goal of this lesson isn't to illustrate good form for programming inheritance, it's to appeal to the user's understanding of hierarchy in the world around us as a baseline for understanding how inheritance allows us to model hierarchies in programming.

As a general note, it's hard to find examples that both show mechanics in a comprehensible way while simultaneously adhering to all best practices. In such cases, we'll typically favor comprehensibility for

learning purposes, then return the best practices once we have the fundamentals and vocabulary to do so effectively.

23 Reply

Avtem

Reply to Alex [11] December 14, 2022 1:19 am PST

Maybe it's worth to check Minecraft source code then. The class Entity is a class for any object that can move (Player, Arrow, Minecart). All of them have position, velocity, name, mass. Maybe another good example: class MonsterWithAI that serves as parent class for Zombie, Skeleton, Spider. All of these mobs have in common things like HP, TargetEntity, Wander(), GoAndAttackPlayer(), MakeSound()

Last edited 2 years ago by Avtem

5 Reply

Louis Cloete

April 18, 2019 3:07 am PDT

Just curious, does inheritance in the OOP sense allow for things like "a square is a rectangle and a rhombus, which are both parallelograms, which is a trapesium, which is a quadrilateral, which is a 2D geometric shape" ?

1 Reply

nascardriver

Reply to Louis Cloete [12] April 18, 2019 3:12 am PDT

yes

2 Reply

Dare

February 24, 2019 6:18 am PST

if i have 4 classes i.e., A,B,C and D And i have to call class A into class D how can i do that?

0 Reply

masterOfNothing

Reply to Dare [13] August 10, 2019 7:56 am PDT

I think you simply use a composition(described in previous chapters).

If your class A strictly inherits various methods (member functions) and uses the same member variables as class D + additional new ones, then you probably use inheritance.

0 Reply

Alex Author

Reply to Dare [13] February 25, 2019 2:04 pm PST

I don't understand the question.

3 Reply

NordicCat

Reply to Alex [14] January 16, 2025 7:50 am PST

make the class a friend and then call as many time as you want. haha!

1 Reply

ryder

December 30, 2018 7:50 pm PST

Hi, Alex or nascardriver, I encontour an issue that I still have no idea for a long time.

In the following code snippet, I try to do a check that:

ChildA is not inherient from ParentB.

I want the program compile and print out me a message that tells me ~childA~ class is not inherient from ~ParentB~ ?

They reason why I ask this question is that I am trying to understand a piece code in Unreal Game Editor.

I will continue with my exploration but if you could add some reference to me. I would be really appreciate it. Thank you.

```cpp
#include "pch.h" // what's this headfile?
#include <iostream>
#include <string>

class ParentA {

public: ParentA() {}

std::string m_name;

std::string getName() const { return m_name; }

void printName() const {

std::cout << "I am a child of A." << std::endl;

}
};

class ParentB {
public: ParentB() {}

std::string m_name;

std::string getName() const { return m_name; }

void functionParentB() {
std::cout << "I am a child of B" << std::endl;
}
};

class childA : public ParentA {
public:

int ChildA() {}

void printNameInChild() {

std::cout << "This is a message from child A" << std::endl;
}


};

class childB : public ParentB {
public:

int ChildB() {}

void printNameInChild() {

std::cout << "This is a message from child B" << std::endl;
}


};

int main() {

childA lucifer;
childB angela;

lucifer.m_name = "Lucifer";
angela.m_name = "Angela";

//lucifer.printName();
lucifer.printNameInChild();
angela.printNameInChild();

// how to writer a if-else statement to determine one child is a child of a particular
```

```
parent?

}

/* notes

Why Shift + F11 cannot jump back out the entry point?

For example, if the cursor is on


getName(), then press 'F11', I go to its definition,

but, I want goback by pressing 'Shift+F11', but not work.

I guess, its the issue with my emacs kbd set for VS.

In exploring, I found that "Alt + F11", which is peak definition.

It is more handy.

*/
```

0 Reply

nascardriver

Reply to ryder [15] December 31, 2018 6:05 am PST

```
std::cout << std::is_base_of_v<ParentA, decltype(angela)> << '\n';
std::cout << std::is_base_of_v<ParentA, decltype(lucifer)> << '\n';
std::cout << std::is_base_of_v<ParentB, decltype(angela)> << '\n';
std::cout << std::is_base_of_v<ParentB, decltype(lucifer)> << '\n';
```

Output

```
0
1
1
0
```

0 Reply

ryder

Reply to nascardriver [16] December 31, 2018 5:42 pm PST

Thanks for pointing out this amazing std library: boost.

> It solve my problem.
>
> But, I recently have a temperation to know more details about c++. For example, after reading your code, I start to read source code of ~is_base_of_v~:

```
+BEGIN_SRC c++

#ifndef BOOST_TT_IS_BASE_OF_HPP_INCLUDED
#define BOOST_TT_IS_BASE_OF_HPP_INCLUDED

#include <boost/type_traits/is_base_and_derived.hpp>
#include <boost/type_traits/is_same.hpp>
#include <boost/type_traits/is_class.hpp>

namespace boost {

   namespace detail{
      template <class B, class D>
      struct is_base_of_imp
      {
          typedef typename remove_cv<B>::type ncvB;
          typedef typename remove_cv<D>::type ncvD;
          BOOST_STATIC_CONSTANT(bool, value = (

(::boost::detail::is_base_and_derived_impl<ncvB,ncvD>::value) ||
             (::boost::is_same<ncvB,ncvD>::value &&
::boost::is_class<ncvB>::value)));
      };
   }

   template <class Base, class Derived> struct is_base_of
      : public integral_constant<bool,
(::boost::detail::is_base_of_imp<Base, Derived>::value)> {};

   template <class Base, class Derived> struct is_base_of<Base,
Derived&> : false_type{};
   template <class Base, class Derived> struct is_base_of<Base&,
Derived&> : false_type{};
   template <class Base, class Derived> struct is_base_of<Base&,
Derived> : false_type{};

} // namespace boost

#endif // BOOST_TT_IS_BASE_AND_DERIVED_HPP_INCLUDED
```

I don't know if you are interesting to know how those code working? I mean, understanding to a degree such that a five-year-old girl could understanding if amount of time being post here.

To be honest with you, I don't know how ~is_base_of_v~ works. The things I can do is to google any things I don't understand.

I am not sure it is very effective, but the key point is that I do not have any burdon on it. It's not like a semester that I have to finish it within some time. In my current case, I can study it as much as possible I'd like to.

0 Reply

nascardriver

Reply to ryder [17] January 1, 2019 4:18 am PST

> Thanks for pointing out this amazing std library: boost
I did not and will not recommend using boost in my replies. boost is a feature-rich library

of which many features have already been implemented into the standard library. You don't need boost. @std::is_base_of_v can be used after including the <type_traits> header.

I don't think you read the chapter about templates yet, which is what this is all about. I'll try to keep it simple:
I'm following the possible implementation shown at cppreference (https://en.cppreference.com/w/cpp/types/is_base_of). Actual implementations might differ.

C++ has a feature, SFINAE. When the compiler has to construct types during compile time (Types that where not explicitly specified by the coder), an error might occur, because the type cannot be constructed (Don't confuse "construct" with constructors of classes, I mean "build" or "come up with"). This error isn't treated as a compiler error, instead the function/type it occurred at is ignored and cannot be used.
@std::is_base_of first tests if both types are classes by attempting to create a pointer to a data member

```
1 // @T is your class (or non-class) type
2 // @p is a pointer to an int-member of a @T
3 int T::*p{ nullptr };
```

@T will be filled in by the compiler. If it can't be filled in, @T is not a class type.
Once the compiler knows both types are classes, it checks if they are unions (I couldn't find an implementation for @std::is_union). If a type is a class and not a union, it continues.
The compiler then tries creates a pointer to an object of the derived class and tries to call a function that has been overloaded to take either a pointer to the base class, or a void*. Base class pointer can implicitly be cast to parent class pointers. Depending on the function that has been called, the compiler knows if the type is a base of the other type.

Once you're done with chapter 12, you can also use

```
1 std::cout << !!dynamic_cast<ParentA *>(&angela) << '\n';
2 std::cout << !!dynamic_cast<ParentA *>(&lucifer) << '\n';
3 std::cout << !!dynamic_cast<ParentB *>(&angela) << '\n';
4 std::cout << !!dynamic_cast<ParentB *>(&lucifer) << '\n';
```

as opposed to the code I posted earlier. This should be easier to understand for now.

1 Reply

Links

1. https://www.learncpp.com/author/Alex/
2. https://www.learncpp.com/cpp-tutorial/basic-inheritance-in-c/
3. https://www.learncpp.com/
4. https://www.learncpp.com/cpp-tutorial/chapter-23-summary-and-quiz/

5. https://www.learncpp.com/introduction-to-inheritance/
6. https://www.learncpp.com/cpp-tutorial/a-few-common-cpp-problems/
7. https://www.learncpp.com/site-news/happy-holidays-to-everyone/
8. https://gravatar.com/
9. https://www.learncpp.com/cpp-tutorial/introduction-to-inheritance/#comment-589868
10. https://www.learncpp.com/cpp-tutorial/introduction-to-inheritance/#comment-410280
11. https://www.learncpp.com/cpp-tutorial/introduction-to-inheritance/#comment-410911
12. https://www.learncpp.com/cpp-tutorial/introduction-to-inheritance/#comment-405509
13. https://www.learncpp.com/cpp-tutorial/introduction-to-inheritance/#comment-396133
14. https://www.learncpp.com/cpp-tutorial/introduction-to-inheritance/#comment-396409
15. https://www.learncpp.com/cpp-tutorial/introduction-to-inheritance/#comment-387829
16. https://www.learncpp.com/cpp-tutorial/introduction-to-inheritance/#comment-387889
17. https://www.learncpp.com/cpp-tutorial/introduction-to-inheritance/#comment-387965
18. https://g.ezoic.net/privacy/learncpp.com