24.3 — Order of construction of derived classes

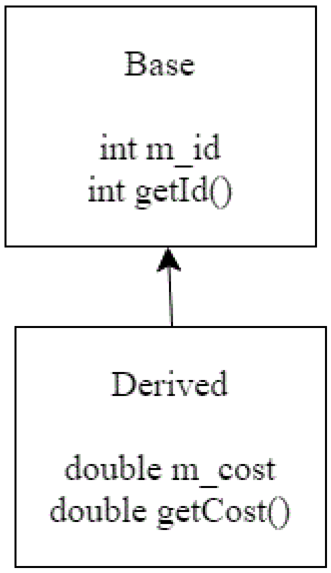
ALEX[1] APRIL 30, 2024

In the previous lesson on basic inheritance in C++ (https://www.learncpp.com/cpp-tutorial/112-basic-inheritance-in-c/)[2], you learned that classes can inherit members and functions from other classes. In this lesson, we're going to take a closer look at the order of construction that happens when a derived class is instantiated.

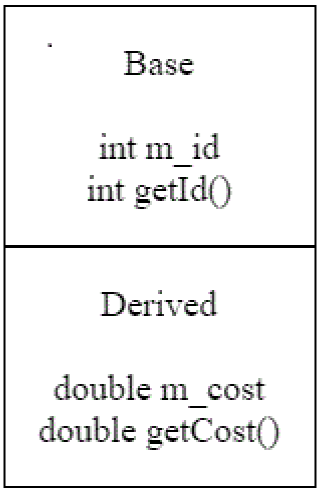
First, let's introduce some new classes that will help us illustrate some important points.

```cpp
class Base
{
public:
    int m_id {};

    Base(int id=0)
        : m_id { id }
    {
    }

    int getId() const { return m_id; }
};

class Derived: public Base
{
public:
    double m_cost {};

    Derived(double cost=0.0)
        : m_cost { cost }
    {
    }

    double getCost() const { return m_cost; }
};
```

In this example, class Derived is derived from class Base.



Because Derived inherits functions and variables from Base, you may assume that the members of Base are copied into Derived. However, this is not true. Instead, we can consider Derived as a two part class: one part Derived, and one part Base.



You've already seen plenty examples of what happens when we instantiate a normal (non-derived) class:

```
int main()
{
    Base base;

    return 0;
}
```

Base is a non-derived class because it does not inherit from any other classes. C++ allocates memory for Base, then calls Base's default constructor to do the initialization.

Now let's take a look at what happens when we instantiate a derived class:

```
int main()
{
    Derived derived;

    return 0;
}
```

If you were to try this yourself, you wouldn't notice any difference from the previous example where we instantiate non-derived class Base. But behind the scenes, things happen slightly differently. As mentioned above, Derived is really two parts: a Base part, and a Derived part. When C++ constructs derived objects, it does so in phases. First, the most-base class (at the top of the inheritance tree) is constructed. Then each child class is constructed in order, until the most-child class (at the bottom of the inheritance tree) is constructed last.

So when we instantiate an instance of Derived, first the Base portion of Derived is constructed (using the Base default constructor). Once the Base portion is finished, the Derived portion is constructed (using the Derived default constructor). At this point, there are no more derived classes, so we are done.

This process is actually easy to illustrate.

```cpp
#include <iostream>

class Base
{
public:
    int m_id {};

    Base(int id=0)
        : m_id { id }
    {
        std::cout << "Base\n";
    }

    int getId() const { return m_id; }
};

class Derived: public Base
{
public:
    double m_cost {};

    Derived(double cost=0.0)
        : m_cost { cost }
    {
        std::cout << "Derived\n";
    }

    double getCost() const { return m_cost; }
};

int main()
{
    std::cout << "Instantiating Base\n";
    Base base;

    std::cout << "Instantiating Derived\n";
    Derived derived;

    return 0;
}
```

This program produces the following result:

```
Instantiating Base
Base
Instantiating Derived
Base
Derived
```

As you can see, when we constructed Derived, the Base portion of Derived got constructed first. This makes sense: logically, a child can not exist without a parent. It's also the safe way to do things: the child class often uses variables and functions from the parent, but the parent class knows nothing about the child. Instantiating the parent class first ensures those variables are already initialized by the time the derived class is created and ready to use them.

Order of construction for inheritance chains

It is sometimes the case that classes are derived from other classes, which are themselves derived from other classes. For example:

```cpp
#include <iostream>

class A
{
public:
    A()
    {
        std::cout << "A\n";
    }
};

class B: public A
{
public:
    B()
    {
        std::cout << "B\n";
    }
};

class C: public B
{
public:
    C()
    {
        std::cout << "C\n";
    }
};

class D: public C
{
public:
    D()
    {
        std::cout << "D\n";
    }
};
```

Remember that C++ always constructs the “first” or “most base” class first. It then walks through the inheritance tree in order and constructs each successive derived class.

Here’s a short program that illustrates the order of creation all along the inheritance chain.

```cpp
int main()
{
    std::cout << "Constructing A: \n";
    A a;

    std::cout << "Constructing B: \n";
    B b;

    std::cout << "Constructing C: \n";
    C c;

    std::cout << "Constructing D: \n";
    D d;
}
```

This code prints the following:

```
Constructing A:
A
Constructing B:
A
B
Constructing C:
A
B
C
Constructing D:
A
B
C
D
```

Conclusion

C++ constructs derived classes in phases, starting with the most-base class (at the top of the inheritance tree) and finishing with the most-child class (at the bottom of the inheritance tree). As each class is constructed, the appropriate constructor from that class is called to initialize that part of the class.

You will note that our example classes in this section have all used base class default constructors (for simplicity). In the next lesson, we will take a closer look at the role of constructors in the process of constructing derived classes (including how to explicitly choose which base class constructor you want your derived class to use).

Next lesson
24.4 Constructors and initialization of derived classes

3

Back to table of contents

4

Previous lesson
24.2 Basic inheritance in C++

5

6

Leave a comment...

Name*

Email*

Notify me about replies:

POST COMMENT

Find a mistake? Leave a comment above!

Avatars from https://gravatar.com/[7] are connected to your provided email address.

74 COMMENTS

Newest

Nidhi Gupta
May 5, 2025 1:27 pm PDT

Construction always starts with the most base class in the inheritance hierarchy.

Each class's constructor runs in order, from base to derived.

Destruction is the reverse: derived class destructor runs first, then the base class. for example,
In the Derived class derived from Base:

cpp
Copy
Edit
Derived derived;
Output:

nginx
Copy
Edit
Base
Derived
In the multi-level inheritance example (D from C from B from A):

cpp
Copy
Edit
D d;
Output:

css
Copy
Edit
A
B

C
D
Each class constructor prints when it runs, clearly showing the order.

Would you like to explore how to use non-default base class constructors in derived classes next?

Do you like this personality?

0 Reply

Garvit
April 14, 2025 5:08 am PDT

Because Derived inherits functions and variables from Base, you may assume that the members of Base are copied into Derived. However, this is not true. Instead, we can consider Derived as a two part class: one part Derived, and one part Base.

from this line could you please confirm me are you saying that the address of data member m_id in base class and same in derived class, both have same memory address ?

1 Reply

EmtyC
December 23, 2024 5:01 am PST

Based lesson

6 Reply

Rushikesh
July 28, 2024 12:14 am PDT

So Basically every time we define a derived class we create memory for both base class and derived and we can use Base class Pointer which points to derived class object; but how does a base class pointer points to derived class because derived class can contain some addition functions and may require some more memory. so does it only points to things which are same and not additional stuff ?

2 Reply

Alex Author
Reply to Rushikesh 8 July 28, 2024 2:55 pm PDT

When a Derived object is instantiated, the Base and Derived parts are consecutive in memory.

If we set a Base pointer and a Derived pointer to point at a Derived object, they will have the same value: the address of the object.

However, the pointers have different type information. The Base pointer will only be able to access the Base part of the object, whereas the Derived pointer will be able to access both the Base and Derived parts. The compiler will resolve member access through the base pointer and derived pointer appropriately.

4 Reply

DANGER
July 1, 2024 2:52 am PDT

Hey,
I have a doubt so, How do I put it like I tried deleting the default constructor for both parent class as well as derived class and make a parameterized one and I got an error in derived class saying "the default constructor of <class_name> cannot be referenced" so do we need a default constructor for inheritance to work and what about parameterized constructor of a base class like can we call it from the derived class object somehow my explanation is bad I can't explain my doubt clearly like I wanna know the details of what's actually happening.
Thank You

0 Reply

Alex Author
Reply to DANGER 9 July 2, 2024 1:29 pm PDT

You need to make sure your derived class non-default constructor is explicitly calling the base class non-default constructor (as part of the member initializer list). Otherwise it will implicitly call the base class default constructor, which is deleted.

```
#include <iostream>

class Foo
{
public:
   Foo() = delete;
   Foo(int) {};
};

class Goo: public Foo
{
public:
    Goo() = delete;
    Goo(int x): Foo(x) {}; // calls base class Foo(int)
};

int main()
{
    Goo g { 5 };
}
```

0 Reply

Timo
October 1, 2023 2:40 am PDT

"This makes sense: **logically**, a child can not exist without a parent"

That's because the makers made it that way. It's just an analogy you use.
Everything could have gone different.

Sorry, triggered cuz "this makes sense" lel

Maybe you could have said "analogical"

1 Reply

Alex Author
Reply to Timo 10 October 2, 2023 1:15 pm PDT

I am a native speaker of English and I don't think I've ever run across the word "analogical" before. You're correct that this probably makes more sense analogically than logically, but I try to avoid non-technical terms that people are going to have to dictionary to understand. :)

13 Reply

learnccp lesson reviewer
July 23, 2023 6:18 pm PDT

Cool lesson!

2 Reply

yousef elsayed
June 21, 2023 11:43 am PDT

Thank you very helpful

1 Reply

Kamran Malik
February 14, 2023 3:55 am PST

Two cute questions.
1.Does Derived class inherit Constructor of Base class(if provided) or not?

2.In inheritance chain when we instantiate object of most derived class then constructors starts executing from most base class until reaches the most derived class. How much this impact performance of program?

Last edited 2 years ago by Kamran Malik

1 Reply

Alex Author
Reply to Kamran Malik 11 February 15, 2023 1:54 pm PST

1. Nope

2. No idea, you'd have to measure.

2 Reply

learnccp lesson reviewer

Reply to Alex 12 July 23, 2023 6:19 pm PDT

can u elaborate on the first one? the constructor is not inherited in anyway?

0 Reply

Alex Author

Reply to learnccp lesson reviewer 13 July 23, 2023 11:39 pm PDT

Nope.

The constructor of a base class builds the base class (allows it to be created, does initialization, sets virtual pointer, etc...). The constructor of a derived class does the same for the derived class.

If a derived class inherited a base constructor, calling that constructor on a derived would try to construct a base object, not a derived object.

From a design standpoint, it's much easier to explicitly define the constructors we want in Derived than to have all Base constructors inherited by default and have to explicitly remove the ones we don't want (which in most cases is all of them since they probably don't do what we need)!

1 Reply

Kamran Malik

Reply to Alex 12 February 16, 2023 2:40 am PST

Thanks for the feedback!

0 Reply

Waldo

July 16, 2022 4:07 am PDT

`: m_id(id)`

`: m_cost(cost)` - should use uniform initialization

> things happen slightly **different**

*differently

0 Reply

Links

1. https://www.learncpp.com/author/Alex/
2. https://www.learncpp.com/cpp-tutorial/112-basic-inheritance-in-c/
3. https://www.learncpp.com/cpp-tutorial/constructors-and-initialization-of-derived-classes/
4. https://www.learncpp.com/
5. https://www.learncpp.com/cpp-tutorial/basic-inheritance-in-c/
6. https://www.learncpp.com/order-of-construction-of-derived-classes/
7. https://gravatar.com/
8. https://www.learncpp.com/cpp-tutorial/order-of-construction-of-derived-classes/#comment-600188
9. https://www.learncpp.com/cpp-tutorial/order-of-construction-of-derived-classes/#comment-599085
10. https://www.learncpp.com/cpp-tutorial/order-of-construction-of-derived-classes/#comment-588013
11. https://www.learncpp.com/cpp-tutorial/order-of-construction-of-derived-classes/#comment-577458
12. https://www.learncpp.com/cpp-tutorial/order-of-construction-of-derived-classes/#comment-577508
13. https://www.learncpp.com/cpp-tutorial/order-of-construction-of-derived-classes/#comment-584348
14. https://g.ezoic.net/privacy/learncpp.com