

## 24.5 — Inheritance and access specifiers

👤 [ALEX](#)<sup>1</sup> ⌚ **SEPTEMBER 11, 2023**

In the previous lessons in this chapter, you've learned a bit about how base inheritance works. In all of our examples so far, we've used public inheritance. That is, our derived class publicly inherits the base class.

In this lesson, we'll take a closer look at public inheritance, as well as the two other kinds of inheritance (private and protected). We'll also explore how the different kinds of inheritance interact with access specifiers to allow or restrict access to members.

To this point, you've seen the private and public access specifiers, which determine who can access the members of a class. As a quick refresher, public members can be accessed by anybody. Private members can only be accessed by member functions of the same class or friends. This means derived classes can not access private members of the base class directly!

```
1 | class Base
2 | {
3 |     private:
4 |         int m_private {}; // can only be accessed by Base members and friends (not derived
5 |         classes)
6 |     public:
7 |         int m_public {}; // can be accessed by anybody
   | };
```

This is pretty straightforward, and you should be quite used to it by now.

---

### The protected access specifier

When dealing with inherited classes, things get a bit more complex.

C++ has a third access specifier that we have yet to talk about because it's only useful in an inheritance context. The **protected** access specifier allows the class the member belongs to, friends, and derived classes to access the member. However, protected members are not accessible from outside the class.

```

1  class Base
2  {
3  public:
4      int m_public {}; // can be accessed by anybody
5  protected:
6      int m_protected {}; // can be accessed by Base members, friends, and derived
7  classes
8  private:
9      int m_private {}; // can only be accessed by Base members and friends (but not
10     derived classes)
11 };
12
13 class Derived: public Base
14 {
15 public:
16     Derived()
17     {
18         m_public = 1; // allowed: can access public base members from derived class
19         m_protected = 2; // allowed: can access protected base members from derived
20         class
21         m_private = 3; // not allowed: can not access private base members from
22         derived class
23     }
24 };
25
26 int main()
27 {
28     Base base;
29     base.m_public = 1; // allowed: can access public members from outside class
30     base.m_protected = 2; // not allowed: can not access protected members from
31     outside class
32     base.m_private = 3; // not allowed: can not access private members from outside
33     class
34
35     return 0;
36 }

```

In the above example, you can see that the protected base member `m_protected` is directly accessible by the derived class, but not by the public.

## So when should I use the protected access specifier?

With a protected attribute in a base class, derived classes can access that member directly. This means that if you later change anything about that protected attribute (the type, what the value means, etc...), you'll probably need to change both the base class AND all of the derived classes.

Therefore, using the protected access specifier is most useful when you (or your team) are going to be the ones deriving from your own classes, and the number of derived classes is reasonable. That way, if you make a change to the implementation of the base class, and updates to the derived classes are necessary as a result, you can make the updates yourself (and have it not take forever, since the number of derived classes is limited).

Making your members private means the public and derived classes can't directly make changes to the base class. This is good for insulating the public or derived classes from implementation changes, and for ensuring invariants are maintained properly. However, it also means your class may need a larger public (or protected) interface to support all of the functions that the public or derived classes need for operation, which has its own cost to build, test, and maintain.

In general, it's better to make your members private if you can, and only use protected when derived classes are planned and the cost to build and maintain an interface to those private members is too high.

## Best practice

Favor private members over protected members.

## Different kinds of inheritance, and their impact on access

First, there are three different ways for classes to inherit from other classes: public, protected, and private.

To do so, simply specify which type of access you want when choosing the class to inherit from:

```
1 // Inherit from Base publicly
2 class Pub: public Base
3 {
4 };
5
6 // Inherit from Base protectedly
7 class Pro: protected Base
8 {
9 };
10
11 // Inherit from Base privately
12 class Pri: private Base
13 {
14 };
15
16 class Def: Base // Defaults to private inheritance
17 {
18 };
```

If you do not choose an inheritance type, C++ defaults to private inheritance (just like members default to private access if you do not specify otherwise).

That gives us 9 combinations: 3 member access specifiers (public, private, and protected), and 3 inheritance types (public, private, and protected).

So what's the difference between these? In a nutshell, when members are inherited, the access specifier for an inherited member may be changed (in the derived class only) depending on the type of inheritance used. Put another way, members that were public or protected in the base class may change access specifiers in the derived class.

This might seem a little confusing, but it's not that bad. We'll spend the rest of this lesson exploring this in detail.

Keep in mind the following rules as we step through the examples:

- A class can always access its own (non-inherited) members.
- The public accesses the members of a class based on the access specifiers of the class it is accessing.
- A derived class accesses inherited members based on the access specifier inherited from the parent class. This varies depending on the access specifier and type of inheritance used.

## Public inheritance

Public inheritance is by far the most commonly used type of inheritance. In fact, very rarely will you see or use the other types of inheritance, so your primary focus should be on understanding this section.

Fortunately, public inheritance is also the easiest to understand. When you inherit a base class publicly,

inherited public members stay public, and inherited protected members stay protected. Inherited private members, which were inaccessible because they were private in the base class, stay inaccessible.

Access specifier in base class	Access specifier when inherited publicly
Public	Public
Protected	Protected
Private	Inaccessible

Here's an example showing how things work:

```
1  class Base
2  {
3  public:
4      int m_public {};
5  protected:
6      int m_protected {};
7  private:
8      int m_private {};
9  };
10
11 class Pub: public Base // note: public inheritance
12 {
13     // Public inheritance means:
14     // Public inherited members stay public (so m_public is treated as public)
15     // Protected inherited members stay protected (so m_protected is treated as
16 protected)
17     // Private inherited members stay inaccessible (so m_private is inaccessible)
18 public:
19     Pub()
20     {
21         m_public = 1; // okay: m_public was inherited as public
22         m_protected = 2; // okay: m_protected was inherited as protected
23         m_private = 3; // not okay: m_private is inaccessible from derived class
24     }
25 };
26
27 int main()
28 {
29     // Outside access uses the access specifiers of the class being accessed.
30     Base base;
31     base.m_public = 1; // okay: m_public is public in Base
32     base.m_protected = 2; // not okay: m_protected is protected in Base
33     base.m_private = 3; // not okay: m_private is private in Base
34
35     Pub pub;
36     pub.m_public = 1; // okay: m_public is public in Pub
37     pub.m_protected = 2; // not okay: m_protected is protected in Pub
38     pub.m_private = 3; // not okay: m_private is inaccessible in Pub
39
40     return 0;
41 }
```

This is the same as the example above where we introduced the protected access specifier, except that we've instantiated the derived class as well, just to show that with public inheritance, things work identically in the base and derived class.

Public inheritance is what you should be using unless you have a specific reason not to.

## Best practice

Use public inheritance unless you have a specific reason to do otherwise.

## Protected inheritance

Protected inheritance is the least common method of inheritance. It is almost never used, except in very particular cases. With protected inheritance, the public and protected members become protected, and private members stay inaccessible.

Because this form of inheritance is so rare, we'll skip the example and just summarize with a table:

Access specifier in base class	Access specifier when inherited protectedly
Public	Protected
Protected	Protected
Private	Inaccessible

## Private inheritance

With private inheritance, all members from the base class are inherited as private. This means private members are inaccessible, and protected and public members become private.

Note that this does not affect the way that the derived class accesses members inherited from its parent! It only affects the code trying to access those members through the derived class.

```
1 class Base
2 {
3 public:
4     int m_public {};
5 protected:
6     int m_protected {};
7 private:
8     int m_private {};
9 };
10
11 class Pri: private Base // note: private inheritance
12 {
13     // Private inheritance means:
14     // Public inherited members become private (so m_public is treated as private)
15     // Protected inherited members become private (so m_protected is treated as
16 private)
17     // Private inherited members stay inaccessible (so m_private is inaccessible)
18 public:
19     Pri()
20     {
21         m_public = 1; // okay: m_public is now private in Pri
22         m_protected = 2; // okay: m_protected is now private in Pri
23         m_private = 3; // not okay: derived classes can't access private members in
24 the base class
25     }
26 };
27
28 int main()
29 {
30     // Outside access uses the access specifiers of the class being accessed.
31     // In this case, the access specifiers of base.
32     Base base;
33     base.m_public = 1; // okay: m_public is public in Base
34     base.m_protected = 2; // not okay: m_protected is protected in Base
35     base.m_private = 3; // not okay: m_private is private in Base
36
37     Pri pri;
38     pri.m_public = 1; // not okay: m_public is now private in Pri
39     pri.m_protected = 2; // not okay: m_protected is now private in Pri
40     pri.m_private = 3; // not okay: m_private is inaccessible in Pri
41
42     return 0;
43 }
```

To summarize in table form:

Access specifier in base class	Access specifier when inherited privately
Public	Private
Protected	Private
Private	Inaccessible

Private inheritance can be useful when the derived class has no obvious relationship to the base class, but uses the base class for implementation internally. In such a case, we probably don't want the public interface of the base class to be exposed through objects of the derived class (as it would be if we inherited publicly).

In practice, private inheritance is rarely used.

## A final example

```

1 class Base
2 {
3 public:
4     int m_public {};
5 protected:
6     int m_protected {};
7 private:
8     int m_private {};
9 };

```

Base can access its own members without restriction. The public can only access m\_public. Derived classes can access m\_public and m\_protected.

```

1 class D2 : private Base // note: private inheritance
2 {
3     // Private inheritance means:
4     // Public inherited members become private
5     // Protected inherited members become private
6     // Private inherited members stay inaccessible
7 public:
8     int m_public2 {};
9 protected:
10    int m_protected2 {};
11 private:
12    int m_private2 {};
13 };

```

D2 can access its own members without restriction. D2 can access Base's m\_public and m\_protected members, but not m\_private. Because D2 inherited Base privately, m\_public and m\_protected are now considered private when accessed through D2. This means the public can not access these variables when using a D2 object, nor can any classes derived from D2.

```

1 class D3 : public D2
2 {
3     // Public inheritance means:
4     // Public inherited members stay public
5     // Protected inherited members stay protected
6     // Private inherited members stay inaccessible
7 public:
8     int m_public3 {};
9 protected:
10    int m_protected3 {};
11 private:
12    int m_private3 {};
13 };

```

D3 can access its own members without restriction. D3 can access D2's m\_public2 and m\_protected2 members, but not m\_private2. Because D3 inherited D2 publicly, m\_public2 and m\_protected2 keep their access specifiers when accessed through D3. D3 has no access to Base's m\_private, which was already private in Base. Nor does it have access to Base's m\_protected or m\_public, both of which became private when D2 inherited them.

## Summary

The way that the access specifiers, inheritance types, and derived classes interact causes a lot of confusion. To try and clarify things as much as possible:


First, a class (and friends) can always access its own non-inherited members. The access specifiers only affect whether outsiders and derived classes can access those members.

Second, when derived classes inherit members, those members may change access specifiers in the derived class. This does not affect the derived classes' own (non-inherited) members (which have their own access specifiers). It only affects whether outsiders and classes derived from the derived class can access those inherited members.

Here's a table of all of the access specifier and inheritance types combinations:

Access specifier in base class	Access specifier when inherited publicly	Access specifier when inherited privately	Access specifier when inherited protectedly
Public	Public	Private	Protected
Protected	Protected	Private	Protected
Private	Inaccessible	Inaccessible	Inaccessible


As a final note, although in the examples above, we've only shown examples using member variables, these access rules hold true for all members (e.g. member functions and types declared inside the class).

[Next lesson](#)  
24.6 [Adding new functionality to a derived class](#)

2

[Back to table of contents](#)

3

[Previous lesson](#)  
24.4 [Constructors and initialization of derived classes](#)

4

5



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

Leave a comment...

 Name\*

Notify me about replies: 



🔍 Find a mistake? Leave a comment above!?

👤 Avatars from <https://gravatar.com/><sup>7</sup> are connected to your provided email address.

## 167 COMMENTS

Newest ▼

**Nidhi Gupta**

🕒 May 6, 2025 9:04 am PDT

- There are 3 access specifiers: public, protected, and private.
- public: accessible by all
- protected: accessible by the class, friends, and derived classes
- private: accessible only by the class and its friends
- There are 3 inheritance types: public, protected, and private.
- Public inheritance: retains access levels (public stays public, protected stays protected)
- Protected inheritance: public and protected become protected
- Private inheritance: public and protected become private

👍 1    ➡ Reply

**Seb**

🕒 January 17, 2025 7:30 am PST

Is it possible to have mixed parent member access? Like if the parent class has two public members `m_var1` and `m_var2`, and in the child class you want `m_var1` to become private while `m_var2` stays public

👍 2    ➡ Reply

**Seb**🗨️ Reply to [Seb](#)<sup>8</sup> 🕒 January 17, 2025 7:34 am PST

After reading a bit further into the next lesson, I guess you could make both members private and then add a public access function in the child class for `m_var2`. But is there a way to do what I said without doing stuff like adding access functions?

👍 1    ➡ Reply

**Seb**🗨️ Reply to [Seb](#)<sup>9</sup> 🕒 January 17, 2025 8:00 am PST

After reading a bit further into further lessons, I guess you could inherit the parent class publicly, and then do `private: using Parent::m_var1` to privatize `m_var1`. So that answers my question

👍 1    ➡ Reply



**RaveN**

Reply to [Seb](#)<sup>10</sup> February 10, 2025 9:44 am PST

Thank you for leaving an answer to your question))

Last edited 4 months ago by RaveN

0

Reply



**Hector**

September 9, 2024 11:25 pm PDT

Thanks for this valuable content!

0

Reply



**Raven**

August 23, 2024 5:45 pm PDT

A good quiz for this section would be to provide the reader with a blank version of the last table, and have them fill it in themselves.

Cheers for the tutorials!

1

Reply



**wangzhichao**

July 15, 2024 9:25 am PDT

this is a very confusing feature of cpp

4

Reply



**rafal**

April 1, 2024 5:35 am PDT

I like to think of it as hierarchy going like:

public > protected > private

And then access specifier says what is maximum access level of symbols from base class.

Like:

```

1 class A
2 {
3     public int m_pub{};
4 };
5
6 class B: protected A // public > protected -> reduce to protected
7 {
8     // m_pub is protected within B;
9 };

```

And privates from base are just inaccessible.

👍 3    ➡ Reply



**Yolanda**

🔄 Reply to [rafal](#) <sup>11</sup> ⌚ July 31, 2024 2:14 am PDT

Yeah, this is the most succinct way I can come up with to remember it:

The access specifier sets the maximum level of access the derived class can have to public and protected members in the base class. Private members in the base class will always be inaccessible.

👍 1    ➡ Reply



**eklektos**

⌚ August 1, 2023 8:32 am PDT

**That gives us 9 combinations: 3 member access specifiers (public, private, and protected), and 3 inheritance types (public, private, and protected).**

Hey Alex!

What do you mean by the 9 combinations? Can you name the combinations? **3x3 = 9?**

✎ Last edited 1 year ago by eklektos

👍 1    ➡ Reply



**Alex**

Author

🔄 Reply to [eklektos](#) <sup>12</sup> ⌚ August 2, 2023 1:22 pm PDT

public members inherited publicly  
 public members inherited privately  
 public members inherited protectedly  
 private members inherited publicly  
 private members inherited privately  
 private members inherited protectedly  
 protected members inherited publicly  
 protected members inherited privately  
 protected members inherited protectedly

👍 18    ➡ Reply



Ajit Mali

Reply to [Alex](#)<sup>13</sup> ⌚ July 15, 2024 2:08 am PDT

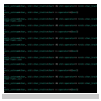
actually only 6 variations

#### **member - inheritance**

1. public - public
2. public - protected
3. public - private
4. protected - public == protected - protected
5. protected - private
6. private remains inaccessible in all cases

👍 0

➡ Reply



learnccp lesson reviewer

⌚ July 24, 2023 12:22 pm PDT

analogy to understand better:

public: It's like memories that can be accessed and shared freely by anyone, including the children.

protected: It's like memories that can be accessed and shared by children (derived classes) but not by others.

private: It's like private thoughts or feelings that can only be accessed and known by the person (class) itself and not by anyone else, including the children (derived classes).

👍 1

➡ Reply



yusef elsayed

⌚ June 21, 2023 10:09 pm PDT

Thank you very much for your efforts

👍 4

➡ Reply



Emeka Daniel

⌚ April 27, 2023 2:55 pm PDT

So how the public accesses inherited members from a derived class is fully dependent on the type of inheritance. That's nice.

But isn't private inheritance a bit redundant for members that are already protected in the base's class?

👍 0

➡ Reply



Alex

Author

Reply to [Emeka Daniel](#)<sup>14</sup> ⌚ May 2, 2023 1:27 pm PDT

Not really. Using private inheritance, protected members of Base become private in Derived, so that if Derived is then inherited by some other derived class, the Base member will be totally inaccessible in that derived class.

```
1 class Base
2 {
3     protected:
4         int m_x{};
5 };
6
7 class Derived : private Base
8 {
9     public:
10         // m_x is now private in this class
11         Derived()
12         {
13             m_x = 5; // can still access here
14         }
15 };
16
17 class D2 : public Derived
18 {
19     public:
20         D2()
21         {
22             m_x = 6; // can't access here
23         }
24 };
25
26 int main()
27 {
28 }
```



2



Reply

## Links

1. <https://www.learncpp.com/author/Alex/>
2. <https://www.learncpp.com/cpp-tutorial/adding-new-functionality-to-a-derived-class/>
3. <https://www.learncpp.com/>
4. <https://www.learncpp.com/cpp-tutorial/constructors-and-initialization-of-derived-classes/>
5. <https://www.learncpp.com/inheritance-and-access-specifiers/>
6. <https://www.learncpp.com/wordpress/alexsthreadedcomments/>
7. <https://gravatar.com/>
8. <https://www.learncpp.com/cpp-tutorial/inheritance-and-access-specifiers/#comment-606664>
9. <https://www.learncpp.com/cpp-tutorial/inheritance-and-access-specifiers/#comment-606665>
10. <https://www.learncpp.com/cpp-tutorial/inheritance-and-access-specifiers/#comment-606667>
11. <https://www.learncpp.com/cpp-tutorial/inheritance-and-access-specifiers/#comment-595305>
12. <https://www.learncpp.com/cpp-tutorial/inheritance-and-access-specifiers/#comment-584879>
13. <https://www.learncpp.com/cpp-tutorial/inheritance-and-access-specifiers/#comment-584980>

14. <https://www.learncpp.com/cpp-tutorial/inheritance-and-access-specifiers/#comment-579872>
15. <https://g.ezoic.net/privacy/learncpp.com>