|  |
| --- |
|  |
| class Base  {  private:      int m\_nPrivate; // can only be accessed by Base member functions (not derived classes)  public:      int m\_nPublic; // can be accessed by anybody  }; |

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

First, there is a third access specifier that we have yet to talk about because it’s only useful in an inheritance context. The **protected** access specifier restricts access to member functions of the same class, or those of derived classes.

|  |  |
| --- | --- |
|  | class Base  {  public:      int m\_nPublic; // can be accessed by anybody  private:      int m\_nPrivate; // can only be accessed by Base member functions (but not derived classes)  protected:      int m\_nProtected; // can be accessed by Base member functions, or derived classes.  };    class Derived: public Base  {  public:      Derived()      {          // Derived's access to Base members is not influenced by the type of inheritance used,          // so the following is always true:            m\_nPublic = 1; // allowed: can access public base members from derived class          m\_nPrivate = 2; // not allowed: can not access private base members from derived class          m\_nProtected = 3; // allowed: can access protected base members from derived class      }  };    int main()  {      Base cBase;      cBase.m\_nPublic = 1; // allowed: can access public members from outside class      cBase.m\_nPrivate = 2; // not allowed: can not access private members from outside class      cBase.m\_nProtected = 3; // not allowed: can not access protected members from outside class  } |

Second, when a derived class inherits from a base class, the access specifiers may change depending on the method of inheritance. There are three different ways for classes to inherit from other classes: public, private, and protected.

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

|  |  |
| --- | --- |
|  | // Inherit from Base publicly  class Pub: public Base  {  };    // Inherit from Base privately  class Pri: private Base  {  };    // Inherit from Base protectedly  class Pro: protected Base  {  };    class Def: Base // Defaults to private inheritance  {  }; |

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).

The rest of this section will be devoted to explaining the difference between these.

Before we get started, the following should be kept in mind as we step through the examples. There are three ways that members can be accessed:

* A class can always access its own members regardless of access specifier.
* The public accesses the members of a class based on the access specifiers of that class.
* A derived class accesses inherited members based on the access specifiers of its immediate parent. A derived class can always access its own members regardless of access specifier.

This may be a little confusing at first, but hopefully will become clearer as we step through the examples.

**Public inheritance**

Public inheritance is by far the most commonly used type of inheritance. In fact, very rarely will you 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, all members keep their original access specifications. Private members stay private, protected members stay protected, and public members stay public.

|  |  |
| --- | --- |
|  | class Base  {  public:      int m\_nPublic;  private:      int m\_nPrivate;  protected:      int m\_nProtected;  };    class Pub: public Base  {      // Public inheritance means:      // m\_nPublic stays public      // m\_nPrivate stays private      // m\_nProtected stays protected        Pub()      {          // The derived class always uses the immediate parent's class access specifications          // Thus, Pub uses Base's access specifiers          m\_nPublic = 1; // okay: anybody can access public members          m\_nPrivate = 2; // not okay: derived classes can't access private members in the base class!          m\_nProtected = 3; // okay: derived classes can access protected members      }  };    int main()  {      // Outside access uses the access specifiers of the class being accessed.      // In this case, the access specifiers of cPub.  Because Pub has inherited publicly from Base,      // no access specifiers have been changed.      Pub cPub;      cPub.m\_nPublic = 1; // okay: anybody can access public members      cPub.m\_nPrivate = 2; // not okay: can not access private members from outside class      cPub.m\_nProtected = 3; // not okay: can not access protected members from outside class  } |

This is fairly straightforward. The things worth noting are:

1. Derived classes can not directly access private members of the base class.
2. The protected access specifier allows derived classes to directly access members of the base class while not exposing those members to the public.
3. The derived class uses access specifiers from the base class.
4. The outside uses access specifiers from the derived class.

To summarize in table form:

|  |  |  |  |
| --- | --- | --- | --- |
| **Public inheritance** | | | |
| **Base access specifier** | **Derived access specifier** | **Derived class access?** | **Public access?** |
| **Public** | Public | Yes | Yes |
| **Private** | Private | No | No |
| **Protected** | Protected | Yes | No |

**Private inheritance**

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

Note that this does not affect that 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.

|  |  |
| --- | --- |
|  | class Base  {  public:      int m\_nPublic;  private:      int m\_nPrivate;  protected:      int m\_nProtected;  };    class Pri: private Base  {      // Private inheritance means:      // m\_nPublic becomes private      // m\_nPrivate stays private      // m\_nProtected becomes private        Pri()      {          // The derived class always uses the immediate parent's class access specifications          // Thus, Pub uses Base's access specifiers          m\_nPublic = 1; // okay: anybody can access public members          m\_nPrivate = 2; // not okay: derived classes can't access private members in the base class!          m\_nProtected = 3; // okay: derived classes can access protected members      }  };    int main()  {      // Outside access uses the access specifiers of the class being accessed.      // Note that because Pri has inherited privately from Base,      // all members of Base have become private when access through Pri.      Pri cPri;      cPri.m\_nPublic = 1; // not okay: m\_nPublic is now a private member when accessed through Pri      cPri.m\_nPrivate = 2; // not okay: can not access private members from outside class      cPri.m\_nProtected = 3; // not okay: m\_nProtected is now a private member when accessed through Pri        // However, we can still access Base members as normal through Base:      Base cBase;      cBase.m\_nPublic = 1; // okay, m\_nPublic is public      cBase.m\_nPrivate = 2; // not okay, m\_nPrivate is private      cBase.m\_nProtected = 3; // not okay, m\_nProtected is protected  } |

To summarize in table form:

|  |  |  |  |
| --- | --- | --- | --- |
| **Private inheritance** | | | |
| **Base access specifier** | **Derived access specifier** | **Derived class access?** | **Public access?** |
| **Public** | Private | Yes | No |
| **Private** | Private | No | No |
| **Protected** | Private | Yes | No |

**Protected inheritance**

Protected inheritance is the last 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 private.

To summarize in table form:

|  |  |  |  |
| --- | --- | --- | --- |
| **Protected inheritance** | | | |
| **Base access specifier** | **Derived access specifier** | **Derived class access?** | **Public access?** |
| **Public** | Protected | Yes | No |
| **Private** | Private | No | No |
| **Protected** | Protected | Yes | No |

Protected inheritance is similar to private inheritance. However, classes derived from the derived class still have access to the public and protected members directly. The public (stuff outside the class) does not.

**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, the base class sets its access specifiers. The base class can always access its own members. The access specifiers only affect whether outsiders and derived classes can access those members.

Second, derived classes have access to base class members based on the access specifiers of the immediate parent. The way a derived class accesses inherited members is not affected by the inheritance method used!

Finally, derived classes can change the access type of inherited members based on the inheritance method used. This does not affect the derived classes’ members, which have their own access specifiers. It only affects whether outsiders and classes derived from the derived class can access those inherited members.

A final example:

|  |  |
| --- | --- |
|  | class Base  {  public:      int m\_nPublic;  private:      int m\_nPrivate;  protected:      int m\_nProtected;  }; |

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

|  |  |
| --- | --- |
|  | class D2: private Base  {  public:      int m\_nPublic2;  private:      int m\_nPrivate2;  protected:      int m\_nProtected2;  } |

D2 can access its own members without restriction. D2 can access Base’s members based on Base’s access specifiers. Thus, it can access m\_nPublic and m\_nProtected, but not m\_nPrivate. Because D2 inherited Base privately, m\_nPublic, m\_nPrivate, and m\_nProtected are now private when accessed through D2. This means the public can not access any of these variables when using a D2 object, nor can any classes derived from D2.

|  |  |
| --- | --- |
|  | class D3: public D2  {  public:      int m\_nPublic3;  private:      int m\_nPrivate3;  protected:      int m\_nProtected3;  }; |

D3 can access its own members without restriction. D3 can access D2’s members based on D2’s access specifiers. Thus, D3 has access to m\_nPublic2 and m\_nProtected2, but not m\_nPrivate2. D3’s access to Base members is controlled by the access specifier of its immediate parent. This means D3 does not have access to any of Base’s members because they all became private when D2 inherited them.

You can declare a derived class from a base class with different access control, i.e., public inheritance, protected inheritance or private inheritance.

class base

{

.... ... ....

};

class derived : access\_specifier base

{

.... ... ....

};

/\* Note: Either keyword public, protected or private is used in place of access\_specifier. \*/

## Things to remember while Using Public, Protected and Private Inheritance

1. Protected and public members(data and function) of a base class are accessible from a derived class(for all three: public, protected and private inheritance).
2. Objects of derived class with private and protected inheritance cannot access any data member of a base class.
3. Objects of derived class with public inheritance can access only public member of a base class.

## Summary of Public, Protected and Private Inheritance

### Accessibility in Public Inheritance

| **Accessibility** | **private** | **protected** | **public** |
| --- | --- | --- | --- |
| Accessible from own class? | yes | yes | yes |
| Accessible from dervied class? | no | yes | yes |
| Accessible outside dervied class? | no | no | yes |

### Accessibility in Protected Inheritance

| **Accessibility** | **private** | **protected** | **public** |
| --- | --- | --- | --- |
| Accessible from own class? | yes | yes | yes |
| Accessible from dervied class? | no | yes | yes |
| Accessible outside dervied class? | no | no | no |

### Accessibility in Private Inheritance

| **Accessibility** | **private** | **protected** | **public** |
| --- | --- | --- | --- |
| Accessible from own class? | yes | yes | yes |
| Accessible from dervied class? | no | yes | yes |
| Accessible outside dervied class? | no | no | no |