INHERITANCE AND PROTECTED MEMBERS IN CPP

Inheritance and protected Members

The **protected** keyword is included in C++ to provide greater flexibility in the inheritance mechanism. When a member of a class is declared as **protected**, that member is not accessible by other, nonmember elements of the program. With one important exception, access to a protected member is the same as access to a private member—it can be accessed only by other members of its class. The sole exception to this is when a protected member is inherited. In this case, a protected member differs substantially from a private one. As explained in the preceding section, a private member of a base class is not accessible by other parts of your program, including any derived class. However, protected members behave differently. If the base class is inherited as **public**, then the base class' protected members become protected members of the derived class and are, therefore, accessible by the derived class. By using **protected**, you can create class members that are private to their class but that can still be inherited and accessed by a derived class. Here is an example:

```
#include <iostream>
using namespace std;
class base { protected:
int i, j; // private to base, but accessible by derived
public:

 void set(int a, int b) { i=a; j=b; }
 void show() { cout << i << " " << j << "\n"; }
 };
 class derived : public base {
 int k;
 public:
 // derived may access base's i and j
 void setk() { k=i*j; }
 void showk() { cout << k << "\n"; }
 };
```

```
int main()
{
derived ob;
ob.set(2, 3); // OK, known to derived
ob.show(); // OK, known to derived
ob.setk();
ob.showk();
return 0;
}
```

In this example, because **base** is inherited by **derived** as **public** and because **i** and **j** are declared as **protected**, **derived**'s function **setk( )** may access them. If **i** and **j** had been declared as **private** by **base**, then **derived** would not have access to them, and the program would not compile.

When a derived class is used as a base class for another derived class, any protected member of the initial base class that is inherited (as public) by the first derived class may also be inherited as protected again by a second derived class. For example, this program is correct, and **derived2** does indeed have access to **i** and **j**.

```
#include <iostream>
using namespace std;
class base
{
 protected:
 int i, j;
 public:
 void set(int a, int b) { i=a; j=b; }
 void show() { cout << i << " " << j << "\n"; }
 };
 // i and j inherited as protected.
 class derived1 : public base {
 int k;
 public:
 void setk() { k = i*j; } // legal
 void showk() { cout << k << "\n"; }
 };
```

```
// i and j inherited indirectly through derived1.
class derived2 : public derived1 {
int m;
public:
void setm() { m = i-j; } // legal
void showm() { cout << m << "\n"; }
};
int main()
{
derived1 ob1;
derived2 ob2;
ob1.set(2, 3);
ob1.show();
ob1.setk();
ob1.showk();
ob2.set(3, 4);
ob2.show();
ob2.setk();

ob2.setm();
ob2.showk();
ob2.showm();
return 0;
}
```

If, however, **base** were inherited as **private**, then all members of **base** would become private members of **derived1**, which means that they would not be accessible by **derived2**. (However, **i** and **j** would still be accessible by **derived1**.) This situation is illustrated by the following program, which is in error (and won't compile).

The comments describe each error:

```
// This program won't compile.
#include <iostream>
using namespace std;
class base {
protected:
int i, j;
public:
```

```
void set(int a, int b) { i=a; j=b; }
void show() { cout << i << " " << j << "\n"; }
};
// Now, all elements of base are private in derived1.
class derived1 : private base {
int k;
public:
// this is legal because i and j are private to derived1
void setk() { k = i*j; } // OK
void showk() { cout << k << "\n"; }
};
// Access to i, j, set(), and show() not inherited.
class derived2 : public derived1 {
int m;
public:
// illegal because i and j are private to derived1
void setm() { m = i-j; } // Error
void showm() { cout << m << "\n"; }
};
int main()
{
derived1 ob1;
derived2 ob2;
ob1.set(1, 2); // error, can't use set()ob1.show(); // error, can't use show()
ob2.set(3, 4); // error, can't use set()
ob2.show(); // error, can't use show()
return 0;
}
```

*Even though **base** is inherited as **private** by **derived1**, **derived1** still has access to **base**'s **public** and **protected** elements. However, it cannot pass along this privilege.*