Ambiguity in Multiple Inheritance

Odd sorts of problems may surface in certain situations involving multiple inheritance. Here's a common one. Two base classes have functions with the same name, while a class derived from both base classes has no function with this name. How do objects of the derived class access the correct base class function? The name of the function alone is insufficient, since the compiler can't figure out which of the two functions is meant.

Here's an example, AMBIGU, that demonstrates the situation:

```
// ambigu.cpp
// demonstrates ambiguity in multiple inheritance
#include <iostream>
using namespace std;
////////////////////////////////////////////////////////////////
class A
   {
   public:
      void show()  { cout << "Class A\n"; }
   };
class B
   {
   public:
      void show()  { cout << "Class B\n"; }
   };
class C : public A, public B
   {
   };
////////////////////////////////////////////////////////////////
int main()
   {
   C objC;            //object of class C
// objC.show();       //ambiguous--will not compile
   objC.A::show();    //OK
   objC.B::show();    //OK
   return 0;
   }
```

The problem is resolved using the scope-resolution operator to specify the class in which the function lies. Thus

```
objC.A::show();
```

refers to the version of `show()` that's in the `A` class, while

```
objC.B::show();
```

refers to the function in the B class. Stroustrup (see Appendix H, “Bibliography”) calls this *disambiguation*.

Another kind of ambiguity arises if you derive a class from two classes that are each derived from the same class. This creates a diamond-shaped inheritance tree. The DIAMOND program shows how this looks.

```
//diamond.cpp
//investigates diamond-shaped multiple inheritance
#include <iostream>
using namespace std;
////////////////////////////////////////////////////////////////
class A
   {
   public:
      void func();
   };
class B : public A
   {  };
class C : public A
   {  };
class D : public B, public C
   {  };
////////////////////////////////////////////////////////////////
int main()
   {
   D objD;
   objD.func();  //ambiguous: won't compile
   return 0;
   }
```

Classes B and C are both derived from class A, and class D is derived by multiple inheritance from both B and C. Trouble starts if you try to access a member function in class A from an object of class D. In this example objD tries to access func(). However, both B and C contain a copy of func(), inherited from A. The compiler can’t decide which copy to use, and signals an error.

There are various advanced ways of coping with this problem, but the fact that such ambiguities can arise causes many experts to recommend avoiding multiple inheritance altogether. You should certainly not use it in serious programs unless you have considerable experience.

Aggregation: Classes Within Classes

We’ll discuss aggregation here because, while it is not directly related to inheritance, both aggregation and inheritance are class relationships that are more specialized than associations. It is instructive to compare and contrast them.