Notice that a class hierarchy is not the same as an organization chart. An organization chart shows lines of command. A class hierarchy results from generalizing common characteristics. The more general the class, the higher it is on the chart. Thus a laborer is more general than a foreman, who is a specialized kind of laborer, so `laborer` is shown above `foreman` in the class hierarchy, although a foreman is probably paid more than a laborer.

Multiple Inheritance

A class can be derived from more than one base class. This is called *multiple inheritance*. Figure 9.9 shows how this looks when a class `C` is derived from base classes `A` and `B`.

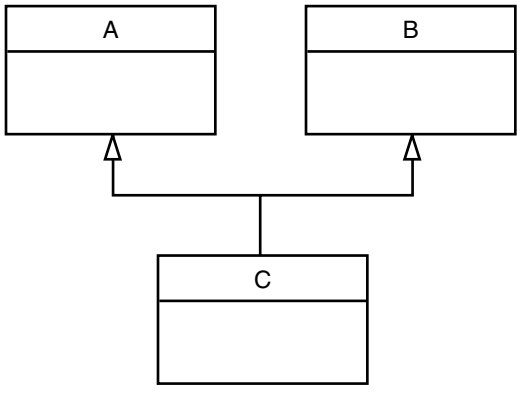

FIGURE 9.9
UML class diagram for multiple inheritance.

The syntax for multiple inheritance is similar to that for single inheritance. In the situation shown in Figure 9.9, the relationship is expressed like this:

```
class A                          // base class A
   {
   };
class B                          // base class B
   {
   };
class C : public A, public B     // C is derived from A and B
   {
   };
```

The base classes from which `C` is derived are listed following the colon in `C`'s specification; they are separated by commas.

Member Functions in Multiple Inheritance

As an example of multiple inheritance, suppose that we need to record the educational experience of some of the employees in the EMPLOY program. Let's also suppose that, perhaps in a different project, we've already developed a class called `student` that models students with different educational backgrounds. We decide that instead of modifying the `employee` class to incorporate educational data, we will add this data by multiple inheritance from the `student` class.

The `student` class stores the name of the school or university last attended and the highest degree received. Both these data items are stored as strings. Two member functions, `getedu()` and `putedu()`, ask the user for this information and display it.

Educational information is not relevant to every class of employee. Let's suppose, somewhat undemocratically, that we don't need to record the educational experience of laborers; it's only relevant for managers and scientists. We therefore modify `manager` and `scientist` so that they inherit from both the `employee` and `student` classes, as shown in Figure 9.10.

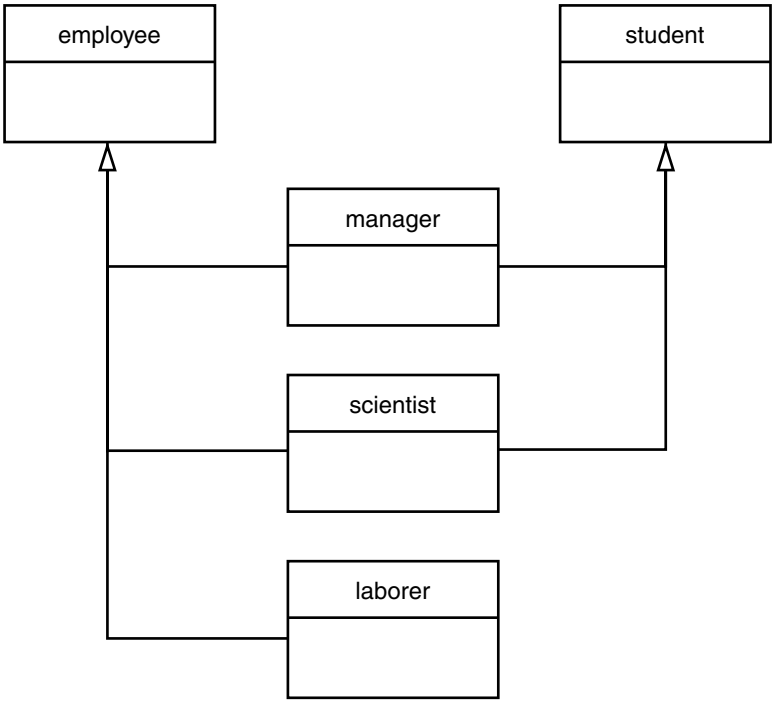

FIGURE 9.10
UML class diagram for EMPMULT.

Here's a miniprogram that shows these relationships (but leaves out everything else):

```
class student
   { };
class employee
   { };
class manager : private employee, private student
   { };
class scientist : private employee, private student
   { };
class laborer : public employee
   { };
```

And here, featuring considerably more detail, is the listing for EMPMULT:

```
//empmult.cpp
//multiple inheritance with employees and degrees
#include <iostream>
using namespace std;
const int LEN = 80;           //maximum length of names
////////////////////////////////////////////////////////////
class student                 //educational background
   {
   private:
      char school[LEN];       //name of school or university
      char degree[LEN];       //highest degree earned
   public:
      void getedu()
         {
         cout << "   Enter name of school or university: ";
         cin >> school;
         cout << "   Enter highest degree earned \n";
         cout << "   (Highschool, Bachelor's, Master's, PhD): ";
         cin >> degree;
         }
      void putedu() const
         {
         cout << "\n   School or university: " << school;
         cout << "\n   Highest degree earned: " << degree;
         }
   };
////////////////////////////////////////////////////////////
class employee
   {
   private:
      char name[LEN];         //employee name
      unsigned long number;   //employee number
```

```
   public:
      void getdata()
         {
         cout << "\n   Enter last name: "; cin >> name;
         cout << "   Enter number: ";      cin >> number;
         }
      void putdata() const
         {
         cout << "\n   Name: " << name;
         cout << "\n   Number: " << number;
         }
   };
////////////////////////////////////////////////////////////////
class manager : private employee, private student  //management
   {
   private:
      char title[LEN];        //"vice-president" etc.
      double dues;            //golf club dues
   public:
      void getdata()
         {
         employee::getdata();
         cout << "   Enter title: ";          cin >> title;
         cout << "   Enter golf club dues: "; cin >> dues;
         student::getedu();
         }
      void putdata() const
         {
         employee::putdata();
         cout << "\n   Title: " << title;
         cout << "\n   Golf club dues: " << dues;
         student::putedu();
         }
   };
////////////////////////////////////////////////////////////////
class scientist : private employee, private student  //scientist
   {
   private:
      int pubs;     //number of publications
   public:
      void getdata()
         {
         employee::getdata();
         cout << "   Enter number of pubs: "; cin >> pubs;
         student::getedu();
         }
```

```
      void putdata() const
         {
         employee::putdata();
         cout << "\n   Number of publications: " << pubs;
         student::putedu();
         }
   };
////////////////////////////////////////////////////////////////
class laborer : public employee             //laborer
   {
   };
////////////////////////////////////////////////////////////////
int main()
   {
   manager m1;
   scientist s1, s2;
   laborer l1;

   cout << endl;
   cout << "\nEnter data for manager 1";    //get data for
   m1.getdata();                            //several employees

   cout << "\nEnter data for scientist 1";
   s1.getdata();

   cout << "\nEnter data for scientist 2";
   s2.getdata();

   cout << "\nEnter data for laborer 1";
   l1.getdata();

   cout << "\nData on manager 1";           //display data for
   m1.putdata();                            //several employees

   cout << "\nData on scientist 1";
   s1.putdata();

   cout << "\nData on scientist 2";
   s2.putdata();

   cout << "\nData on laborer 1";
   l1.putdata();
   cout << endl;
   return 0;
   }
```

The `getdata()` and `putdata()` functions in the `manager` and `scientist` classes incorporate calls to functions in the `student` class, such as

```
student::getedu();
```

and

```
student::putedu();
```

These routines are accessible in `manager` and `scientist` because these classes are descended from `student`.

Here's some sample interaction with EMPMULT:

```
Enter data for manager 1
   Enter last name: Bradley
   Enter number: 12
   Enter title: Vice-President
   Enter golf club dues: 100000
   Enter name of school or university: Yale
   Enter highest degree earned
   (Highschool, Bachelor's, Master's, PhD): Bachelor's

Enter data for scientist 1
   Enter last name: Twilling
   Enter number: 764
   Enter number of pubs: 99
   Enter name of school or university: MIT
   Enter highest degree earned
   (Highschool, Bachelor's, Master's, PhD): PhD

Enter data for scientist 2
   Enter last name: Yang
   Enter number: 845
   Enter number of pubs: 101
   Enter name of school or university: Stanford
   Enter highest degree earned
   (Highschool, Bachelor's, Master's, PhD): Master's

Enter data for laborer 1
   Enter last name: Jones
   Enter number: 48323
```

As we saw in the EMPLOY and EMPLOY2 examples, the program then displays this information in roughly the same form.