```
cout << "  Description: " << todays_special.description << endl;
cout << "     Quantity: " << todays_special.quantity_on_hand << endl;
cout << "Regular Price: " << setprecision(2)
     << todays_special.retail_price << endl;
cout << "   Sale Price: " << todays_special.retail_price * 0.8
     << endl;
```

3. Compile and run the program to see the output from the structure.
4. Save the source code file and close.

NESTED STRUCTURES

A structure can include enumerated data types and even other structures as members. Consider the program in Figure 10-9. The program sets up a structure to be used to store vital data about blood donors. The structure named **donor_info** includes two enumerated data types (**blood_type** and **rh_factor**) and a structure (**blood_pressure**) among its members.

```
enum blood_type { unknown, A, B, AB, O };
enum rh_factor { negative, positive };

struct blood_pressure
  {
   int systolic;
   int diastolic;
  };

struct donor_info
  {
   blood_type type;
   rh_factor rh;
   blood_pressure bp;
   int heart_rate;
  };

int main()
{
  donor_info current_donor;

  current_donor.type = A;
  current_donor.rh = positive;
  current_donor.bp.systolic = 130;
  current_donor.bp.diastolic = 74;
  current_donor.heart_rate = 69;
  return 0;
}
```

FIGURE 10-9
This program uses nested structures to store blood pressure values.

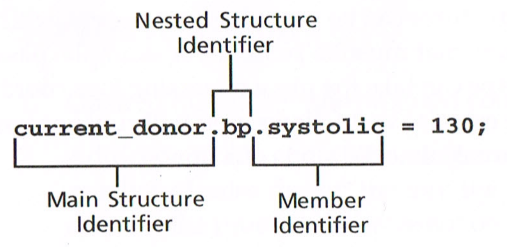
Nested Structure Identifier
current_donor.bp.systolic = 130;
Main Structure Identifier
Member Identifier

FIGURE 10-10
This assignment stores the value 130 in the systolic variable which is in the **bp** structure which is in the **current_donor** structure.

The **blood_type** and **rh_factor** data are good candidates for an enumerated data type because only a few values are possible. Because blood pressure is actually two values, a structure is used to group the two values into one variable. When a structure appears within a structure the resulting data structure is called a ***nested structure.***

Accessing the nested structure requires that two periods be used. As the statement in Figure 10-10 illustrates, initializing the blood pressure values requires that the first structure be accessed by name, then the structure variable within the first structure, and finally, the variable within the nested structure.

EXERCISE 10-8 NESTED STRUCTURES

1. Enter the program shown in Figure 10-9. Save the source code as *DONORS.CPP*.
2. Add the following statement to the main function.

```
cout << "The donor's blood pressure is "
     << current_donor.bp.diastolic << " over "
     << current_donor.bp.systolic << ".\n";
```

3. Add the appropriate directive to include the code necessary for the cout statements.
4. Compile and run the progra1m. Save and close the source code.

It is easy to get locked into thinking about structures in terms of database applications. Structures, however, have many other applications. For example, some mathematical or graphical applications use coordinates such as (x,y) in calculations. You can use a structure like the one below to group the x and y into a structure that represents a graphical point.

```
struct point
  {
   float x;
   float y;
  };
```

You might then want to use nested structures to create a data type that defines two points, that when connected, form a line, as shown below.

```
struct line
  {
   point p1;
   point p2;
  };
```