17. Write a revised version of the `getcrow()` member function from Question 10 that is defined outside the class definition.
18. The only technical difference between structures and classes in C++ is that _____.
19. If three objects of a class are defined, how many copies of that class's data items are stored in memory? How many copies of its member functions?
20. Sending a message to an object is the same as _____.
21. Classes are useful because they

    a. are removed from memory when not in use.

    b. permit data to be hidden from other classes.

    c. bring together all aspects of an entity in one place.

    d. can closely model objects in the real world.
22. True or false: There is a simple but precise methodology for dividing a real-world programming problem into classes.
23. For the object for which it was called, a `const` member function

    a. can modify both `const` and non-`const` member data.

    b. can modify only `const` member data.

    c. can modify only non-`const` member data.

    d. can modify neither `const` nor non-`const` member data.
24. True or false: If you declare a `const` object, it can only be used with `const` member functions.
25. Write a declaration (not a definition) for a `const void` function called `aFunc()` that takes one `const` argument called `jerry` of type `float`.

Exercises

Answers to the starred exercises can be found in Appendix G.

*1. Create a class that imitates part of the functionality of the basic data type `int`. Call the class `Int` (note different capitalization). The only data in this class is an `int` variable. Include member functions to initialize an `Int` to 0, to initialize it to an `int` value, to display it (it looks just like an `int`), and to add two `Int` values.

    Write a program that exercises this class by creating one uninitialized and two initialized `Int` values, adding the two initialized values and placing the response in the uninitialized value, and then displaying this result.

*2. Imagine a tollbooth at a bridge. Cars passing by the booth are expected to pay a 50 cent toll. Mostly they do, but sometimes a car goes by without paying. The tollbooth keeps track of the number of cars that have gone by, and of the total amount of money collected.

Model this tollbooth with a class called `tollBooth`. The two data items are a type `unsigned int` to hold the total number of cars, and a type `double` to hold the total amount of money collected. A constructor initializes both of these to 0. A member function called `payingCar()` increments the car total and adds 0.50 to the cash total. Another function, called `nopayCar()`, increments the car total but adds nothing to the cash total. Finally, a member function called `display()` displays the two totals. Make appropriate member functions `const`.

Include a program to test this class. This program should allow the user to push one key to count a paying car, and another to count a nonpaying car. Pushing the Esc key should cause the program to print out the total cars and total cash and then exit.

*3. Create a class called `time` that has separate `int` member data for hours, minutes, and seconds. One constructor should initialize this data to 0, and another should initialize it to fixed values. Another member function should display it, in 11:59:59 format. The final member function should add two objects of type `time` passed as arguments.

A `main()` program should create two initialized `time` objects (should they be `const`?) and one that isn't initialized. Then it should add the two initialized values together, leaving the result in the third `time` variable. Finally it should display the value of this third variable. Make appropriate member functions `const`.

4. Create an `employee` class, basing it on Exercise 4 of Chapter 4. The member data should comprise an `int` for storing the employee number and a `float` for storing the employee's compensation. Member functions should allow the user to enter this data and display it. Write a `main()` that allows the user to enter data for three employees and display it.

5. Start with the `date` structure in Exercise 5 in Chapter 4 and transform it into a `date` class. Its member data should consist of three `int`s: `month`, `day`, and `year`. It should also have two member functions: `getdate()`, which allows the user to enter a date in 12/31/02 format, and `showdate()`, which displays the date.

6. Extend the `employee` class of Exercise 4 to include a `date` class (see Exercise 5) and an `etype enum` (see Exercise 6 in Chapter 4). An object of the `date` class should be used to hold the date of first employment; that is, the date when the employee was hired. The `etype` variable should hold the employee's type: laborer, secretary, manager, and so on. These two items will be private member data in the `employee` definition, just like the employee number and salary. You'll need to extend the `getemploy()` and `putemploy()` functions to obtain this new information from the user and display it. These functions will probably need `switch` statements to handle the `etype` variable. Write a `main()` program that allows the user to enter data for three `employee` variables and then displays this data.

7. In ocean navigation, locations are measured in degrees and minutes of latitude and longitude. Thus if you're lying off the mouth of Papeete Harbor in Tahiti, your location is 149 degrees 34.8 minutes west longitude, and 17 degrees 31.5 minutes south latitude. This is

written as 149°34.8' W, 17°31.5' S. There are 60 minutes in a degree. (An older system also divided a minute into 60 seconds, but the modern approach is to use decimal minutes instead.) Longitude is measured from 0 to 180 degrees, east or west from Greenwich, England, to the international dateline in the Pacific. Latitude is measured from 0 to 90 degrees, north or south from the equator to the poles.

Create a class `angle` that includes three member variables: an `int` for degrees, a `float` for minutes, and a `char` for the direction letter (N, S, E, or W). This class can hold either a latitude variable or a longitude variable. Write one member function to obtain an angle value (in degrees and minutes) and a direction from the user, and a second to display the angle value in 179°59.9' E format. Also write a three-argument constructor. Write a `main()` program that displays an angle initialized with the constructor, and then, within a loop, allows the user to input any angle value, and then displays the value. You can use the hex character constant `'\xF8'`, which usually prints a degree (°) symbol.

8. Create a class that includes a data member that holds a "serial number" for each object created from the class. That is, the first object created will be numbered 1, the second 2, and so on.

   To do this, you'll need another data member that records a count of how many objects have been created so far. (This member should apply to the class as a whole; not to individual objects. What keyword specifies this?) Then, as each object is created, its constructor can examine this count member variable to determine the appropriate serial number for the new object.

   Add a member function that permits an object to report its own serial number. Then write a `main()` program that creates three objects and queries each one about its serial number. They should respond `I am object number 2`, and so on.

9. Transform the `fraction` structure from Exercise 8 in Chapter 4 into a `fraction` class. Member data is the fraction's numerator and denominator. Member functions should accept input from the user in the form `3/5`, and output the fraction's value in the same format. Another member function should add two fraction values. Write a `main()` program that allows the user to repeatedly input two fractions and then displays their sum. After each operation, ask whether the user wants to continue.

10. Create a class called `ship` that incorporates a ship's number and location. Use the approach of Exercise 8 to number each `ship` object as it is created. Use two variables of the `angle` class from Exercise 7 to represent the ship's latitude and longitude. A member function of the `ship` class should get a position from the user and store it in the object; another should report the serial number and position. Write a `main()` program that creates three ships, asks the user to input the position of each, and then displays each ship's number and position.

11. Modify the four-function fraction calculator of Exercise 12 in Chapter 5 to use a `fraction` class rather than a structure. There should be member functions for input and output, as well as for the four arithmetical operations. While you're at it, you might as well install the capability to reduce fractions to lowest terms. Here's a member function that will reduce the `fraction` object of which it is a member to lowest terms. It finds the greatest common divisor (gcd) of the fraction's numerator and denominator, and uses this gcd to divide both numbers.

```
void fraction::lowterms()     // change ourself to lowest terms
   {
   long tnum, tden, temp, gcd;

   tnum = labs(num);          // use non-negative copies
   tden = labs(den);          //    (needs cmath)
   if(tden==0 )   // check for n/0
      { cout << "Illegal fraction: division by 0"; exit(1); }
   else if( tnum==0 )         // check for 0/n
      { num=0; den = 1; return; }

   // this 'while' loop finds the gcd of tnum and tden
   while(tnum != 0)
      {
      if(tnum < tden)         // ensure numerator larger
          { temp=tnum; tnum=tden; tden=temp; }  // swap them
      tnum = tnum - tden;     // subtract them
      }
   gcd = tden;                // this is greatest common divisor
   num = num / gcd;           // divide both num and den by gcd
   den = den / gcd;           // to reduce frac to lowest terms
   }
```

You can call this function at the end of each arithmetic function, or just before you perform output. You'll also need the usual member functions: four arithmetic operations, input, and display. You may find a two-argument constructor useful.

12. Note that one advantage of the OOP approach is that an entire class can be used, without modification, in a different program. Use the `fraction` class from Exercise 11 in a program that generates a multiplication table for fractions. Let the user input a denominator, and then generate all combinations of two such fractions that are between 0 and 1, and multiply them together. Here's an example of the output if the denominator is 6:

```
       1/6     1/3    1/2    2/3    5/6
----------------------------------------
1/6    1/36    1/18   1/12   1/9    5/36
1/3    1/18    1/9    1/6    2/9    5/18
1/2    1/12    1/6    1/4    1/3    5/12
2/3    1/9     2/9    1/3    4/9    5/9
5/6    5/36    5/18   5/12   5/9    25/36
```