18. State the reason that

```
enum isWord{ NO, YES };
```

is better than

```
enum isWord{ YES, NO };
```

Exercises

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

*1. A phone number, such as (212) 767-8900, can be thought of as having three parts: the area code (212), the exchange (767), and the number (8900). Write a program that uses a structure to store these three parts of a phone number separately. Call the structure `phone`. Create two structure variables of type `phone`. Initialize one, and have the user input a number for the other one. Then display both numbers. The interchange might look like this:

```
Enter your area code, exchange, and number: 415 555 1212
My number is (212) 767-8900
Your number is (415) 555-1212
```

*2. A point on the two-dimensional plane can be represented by two numbers: an `x` coordinate and a `y` coordinate. For example, (4,5) represents a point 4 units to the right of the vertical axis, and 5 units up from the horizontal axis. The sum of two points can be defined as a new point whose `x` coordinate is the sum of the `x` coordinates of the two points, and whose `y` coordinate is the sum of the `y` coordinates.

Write a program that uses a structure called `point` to model a point. Define three points, and have the user input values to two of them. Then set the third point equal to the sum of the other two, and display the value of the new point. Interaction with the program might look like this:

```
Enter coordinates for p1: 3 4
Enter coordinates for p2: 5 7
Coordinates of p1+p2 are: 8, 11
```

*3. Create a structure called `Volume` that uses three variables of type `Distance` (from the ENGLSTRC example) to model the volume of a room. Initialize a variable of type `Volume` to specific dimensions, then calculate the volume it represents, and print out the result. To calculate the volume, convert each dimension from a `Distance` variable to a variable of type `float` representing feet and fractions of a foot, and then multiply the resulting three numbers.

4. Create a structure called `employee` that contains two members: an employee number (type `int`) and the employee's compensation (in dollars; type `float`). Ask the user to fill in this data for three employees, store it in three variables of type `struct employee`, and then display the information for each employee.

5. Create a structure of type `date` that contains three members: the month, the day of the month, and the year, all of type `int`. (Or use day-month-year order if you prefer.) Have the user enter a date in the format 12/31/2001, store it in a variable of type `struct date`, then retrieve the values from the variable and print them out in the same format.
6. We said earlier that C++ I/O statements don't automatically understand the data types of enumerations. Instead, the (>>) and (<<) operators think of such variables simply as integers. You can overcome this limitation by using `switch` statements to translate between the user's way of expressing an enumerated variable and the actual values of the enumerated variable. For example, imagine an enumerated type with values that indicate an employee type within an organization:

   ```
   enum etype { laborer, secretary, manager, accountant, executive,
   researcher };
   ```

   Write a program that first allows the user to specify a type by entering its first letter (`'l'`, `'s'`, `'m'`, and so on), then stores the type chosen as a value of a variable of type `enum etype`, and finally displays the complete word for this type.

   ```
   Enter employee type (first letter only)
      laborer, secretary, manager,
      accountant, executive, researcher): a
   Employee type is accountant.
   ```

   You'll probably need two `switch` statements: one for input and one for output.
7. Add a variable of type `enum etype` (see Exercise 6), and another variable of type `struct date` (see Exercise 5) to the `employee` class of Exercise 4. Organize the resulting program so that the user enters four items of information for each of three employees: an employee number, the employee's compensation, the employee type, and the date of first employment. The program should store this information in three variables of type `employee`, and then display their contents.
8. Start with the fraction-adding program of Exercise 9 in Chapter 2, "C++ Programming Basics." This program stores the numerator and denominator of two fractions before adding them, and may also store the answer, which is also a fraction. Modify the program so that all fractions are stored in variables of type `struct fraction`, whose two members are the fraction's numerator and denominator (both type `int`). All fraction-related data should be stored in structures of this type.
9. Create a structure called `time`. Its three members, all type `int`, should be called `hours`, `minutes`, and `seconds`. Write a program that prompts the user to enter a time value in hours, minutes, and seconds. This can be in 12:59:59 format, or each number can be entered at a separate prompt ("Enter hours:", and so forth). The program should then store the time in a variable of type `struct time`, and finally print out the total number of seconds represented by this time value:

   ```
   long totalsecs = t1.hours*3600 + t1.minutes*60 + t1.seconds
   ```

10. Create a structure called `sterling` that stores money amounts in the old-style British system discussed in Exercises 8 and 11 in Chapter 3, "Loops and Decisions." The members could be called `pounds`, `shillings`, and `pence`, all of type `int`. The program should ask the user to enter a money amount in new-style decimal pounds (type `double`), convert it to the old-style system, store it in a variable of type `struct sterling`, and then display this amount in pounds-shillings-pence format.
11. Use the `time` structure from Exercise 9, and write a program that obtains two `time` values from the user in 12:59:59 format, stores them in `struct time` variables, converts each one to seconds (type `int`), adds these quantities, converts the result back to hours-minutes-seconds, stores the result in a `time` structure, and finally displays the result in 12:59:59 format.
12. Revise the four-function fraction calculator program of Exercise 12 in Chapter 3 so that each fraction is stored internally as a variable of type `struct fraction`, as discussed in Exercise 8 in this chapter.