9.14 *(Complex Class)* Create a class called `Complex` for performing complex-number arithmetic. Write a program to test your class. Complex numbers have the form `realPart + imaginaryPart * i` where i is $\sqrt{-1}$. Use `double` variables to represent the `private` data of the class. Provide a constructor that enables an object of this class to be initialized when it's declared. The constructor should contain default values in case no initializers are provided. Provide `public` member functions that perform the following tasks:

a) Adding two `Complex` numbers: The real parts are added together and the imaginary parts are added together.

b) Subtracting two `Complex` numbers: The real part of the right operand is subtracted from the real part of the left operand, and the imaginary part of the right operand is subtracted from the imaginary part of the left operand.

c) Printing `Complex` numbers in the form `(a, b)`, where `a` is the real part and `b` is the imaginary part.

9.20 *(Rectangle Class)* Create a class `Rectangle` with attributes `length` and `width`, each of which defaults to 1. Provide member functions that calculate the `perimeter` and the `area` of the rectangle. Also, provide *set* and *get* functions for the `length` and `width` attributes. The *set* functions should verify that `length` and `width` are each floating-point numbers larger than 0.0 and less than 20.0.

9.21 *(Enhancing Class `Rectangle`)* Create a more sophisticated `Rectangle` class than the one you created in Exercise 9.20. This class stores only the Cartesian coordinates of the four corners of the rectangle. The constructor calls a *set* function that accepts four sets of coordinates and verifies that each of these is in the first quadrant with no single *x*- or *y*-coordinate larger than 20.0. The *set* function also verifies that the supplied coordinates do, in fact, specify a rectangle. Provide member functions that calculate the `length`, `width`, `perimeter` and `area`. The length is the larger of the two dimensions. Include a predicate function `square` that determines whether the rectangle is a square.

9.22 *(Enhancing Class `Rectangle`)* Modify class `Rectangle` from Exercise 9.21 to include a `draw` function that displays the rectangle inside a 25-by-25 box enclosing the portion of the first quadrant in which the rectangle resides. Include a `setFillCharacter` function to specify the character out of which the body of the rectangle will be drawn. Include a `setPerimeterCharacter` function to specify

the character that will be used to draw the border of the rectangle. If you feel ambitious, you might include functions to scale the size of the rectangle, rotate it, and move it around within the designated portion of the first quadrant.

9.24 *(TicTacToe Class)* Create a class `TicTacToe` that will enable you to write a complete program to play the game of tic-tac-toe. The class contains as `private` data a 3-by-3 two-dimensional array of integers. The constructor should initialize the empty board to all zeros. Allow two human players. Wherever the first player moves, place a 1 in the specified square. Place a 2 wherever the second player moves. Each move must be to an empty square. After each move, determine whether the game has been won or is a draw. If you feel ambitious, modify your program so that the computer makes the moves for one of the players. Also, allow the player to specify whether he or she wants to go first or second. If you feel exceptionally ambitious, develop a program that will play three-dimensional tic-tac-toe on a 4-by-4-by-4 board. [*Caution:* This is an extremely challenging project that could take many weeks of effort!]

10.10 *(Card Shuffling and Dealing)* Create a program to shuffle and deal a deck of cards. The program should consist of class `Card`, class `DeckOfCards` and a driver program. Class `Card` should provide:

a) Data members `face` and `suit` of type `int`.
b) A constructor that receives two `int`s representing the face and suit and uses them to initialize the data members.
c) Two `static` arrays of `string`s representing the faces and suits.
d) A `toString` function that returns the `Card` as a `string` in the form "*face* `of` *suit*." You can use the `+` operator to concatenate `string`s.

Class `DeckOfCards` should contain:

a) A `vector` of `Card`s named `deck` to store the `Card`s.
b) An integer `currentCard` representing the next card to deal.
c) A default constructor that initializes the `Card`s in the deck. The constructor should use `vector` function `push_back` to add each `Card` to the end of the `vector` after the `Card` is created and initialized. This should be done for each of the 52 `Card`s in the deck.
d) A `shuffle` function that shuffles the `Card`s in the deck. The shuffle algorithm should iterate through the `vector` of `Card`s. For each `Card`, randomly select another `Card` in the deck and swap the two `Card`s.
e) A `dealCard` function that returns the next `Card` object from the deck.
f) A `moreCards` function that returns a `bool` value indicating whether there are more `Card`s to deal.

The driver program should create a `DeckOfCards` object, shuffle the cards, then deal the 52 cards.

10.11 *(Card Shuffling and Dealing)* Modify the program you developed in Exercise 10.10 so that it deals a five-card poker hand. Then write functions to accomplish each of the following:

a) Determine whether the hand contains a pair.
b) Determine whether the hand contains two pairs.
c) Determine whether the hand contains three of a kind (e.g., three jacks).
d) Determine whether the hand contains four of a kind (e.g., four aces).
e) Determine whether the hand contains a flush (i.e., all five cards of the same suit).
f) Determine whether the hand contains a straight (i.e., five cards of consecutive face values).