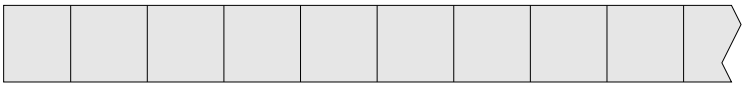
## 3.10 Laboratory: The Silver Dollar Game

**Objective.** To implement a simple game using `Vectors` or arrays.

**Discussion.** The Silver Dollar Game is played between two players. An arbitrarily long strip of paper is marked off into squares:



The game begins by placing silver dollars in a few of the squares. Each square holds at most one coin. Interesting games begin with some pairs of coins separated by one or more empty squares.

The goal is to move all the $n$ coins to the leftmost $n$ squares of the paper. This is accomplished by players alternately moving a single coin, constrained by the following rules:

1. Coins move only to the left.
2. No coin may pass another.
3. No square may hold more than one coin.

The last person to move is the winner.

**Procedure.** Write a program to facilitate playing the Silver Dollar Game. When the game starts, the computer has set up a random strip with 3 or more coins. Two players are then alternately presented with the current game state and are allowed to enter moves. If the coins are labeled $0$ through $n-1$ from left to right, a move could be specified by a coin number and the number of squares to move the coin to the left. If the move is illegal, the player is repeatedly prompted to enter a revised move. Between turns the computer checks the board state to determine if the game has been won.

Here is one way to approach the problem:

1. Decide on an internal representation of the strip of coins. Does your representation store *all* the information necessary to play the game? Does your representation store *more* information than is necessary? Is it easy to test for a legal move? Is it easy to test for a win?
2. Develop a new class, `CoinStrip`, that keeps track of the state of the playing strip. There should be a constructor, which generates a random board. Another method, `toString`, returns a string representation of the coin strip. What other operations seem to be necessary? How are moves performed? How are rules enforced? How is a win detected?

3. Implement an application whose `main` method controls the play of a single game.

**Thought Questions.** Consider the following questions as you complete the lab:

*Hint: When flipped, the Belgian Euro is heads 149 times out of 250.*

1. How might one pick game sizes so that, say, one has a 50 percent chance of a game with three coins, a 25 percent chance of a game with four coins, a $12\frac{1}{2}$ percent chance of a game with five coins, and so on? Would your technique bias your choice of underlying data structure?
2. How might one generate games that are not immediate wins? Suppose you wanted to be guaranteed a game with the possibility of $n$ moves?
3. Suppose the computer could occasionally provide good hints. What opportunities appear easy to recognize?
4. How might you write a method, `computerPlay`, where the computer plays to win?
5. A similar game, called Welter's Game (after C. P. Welter, who analyzed the game), allows the coins to pass each other. Would this modification of the rules change your implementation significantly?

**Notes:**