Karel the Robot
and Algorithms (Karel3)

Objective: To learn algorithmic design using Karel the Robot to solve problems.

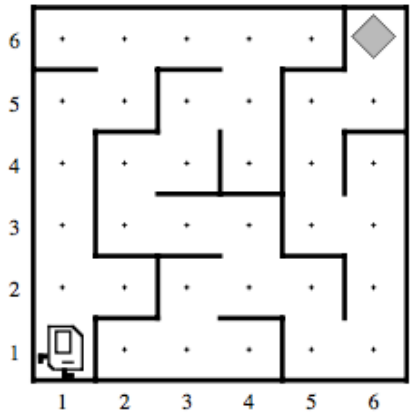
Background:
Figuring out how to solve a particular problem by computer generally requires considerable creativity. The process of designing a solution strategy is traditionally called **algorithmic design**.

The word algorithm comes from the name of a ninth-century Persian mathematician, Abu Ja'far Mohammed ibn Mûsâ al-Khowârizmî, who wrote an influential treatise on mathematics. Today, the notion of an algorithm has been formalized so that it refers to a solution strategy that meets the following conditions:

1. The strategy is expressed in a form that is clear and unambiguous.
2. The steps in the strategy can be carried out. (No wishful magical incantations!)
3. The strategy always terminates after a finite number of steps. (converges to a solution)

Download and unzip the file **Karel3.zip** from Mr Greenstein's web site. It will create the directory **Karel3** in which we will be working.

Solving a maze:
As an example of algorithmic design, suppose that you wanted to teach Karel to escape from a maze. In Karel's world, a maze might look like this:

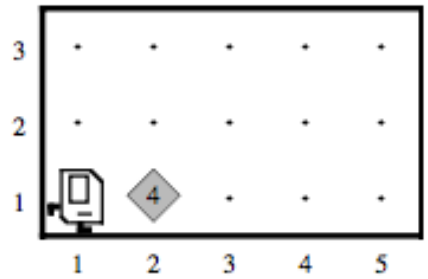


The exit to the maze is marked by a beeper, so that Karel's job is to navigate the corridors of the maze until it finds the beeper indicating the exit. The program, however, must be general enough to solve any maze, and not just the one pictured here.

For most mazes, however, you can use a simpler strategy called the **right-hand rule**, in which you begin by putting your right hand on the adjacent wall and then go through the maze without ever taking your hand off the wall. Another way to express this strategy is to proceed through the maze one step at a time, always taking the rightmost available path.

Problem #1: Now it is your turn to solve the maze problem. Open the file **MazeSolvingKarel.java** and work out a solution. When you are done, show your results to Mr Greenstein.

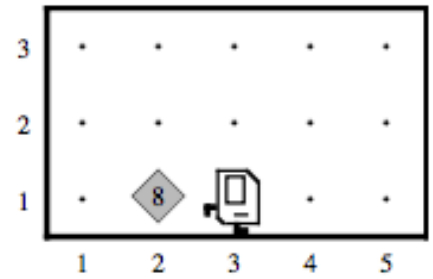
Doubling the number of beepers:
Another programming task that leads to interesting algorithmic choices is the problem of getting Karel to double the number of beepers on a corner. For example, suppose that Karel starts out in the following world:



In this case, there are four beepers at the corner of 1st Street and 2nd Avenue. The goal of this problem is to write a method doubleBeepers() that doubles the number of beepers on the current square. The run() method is as follows:

```
public void run() {
   move();
   doubleBeepers();
   move();
}
```

The final state of the world should look like this:



The program should be general enough to work for any number of beepers. For example, if there had originally been 21 beepers on the corner of 1st Street and 2nd Avenue, the program should end with 42 beepers on that corner.

WARNING: You cannot declare or use variables in the code!!! Your algorithm should work without them.

Problem #2: Solve the double beeper problem. Open the file **DoubleBeeperKarel.java** and work out a solution. Be sure to test it with the worlds **DoubleBeeperKarel7.w** and **DoubleBeeperKarel10.w**. When you are done, show your results to Mr Greenstein.