AP Computer Science Name: _____
Karel Chapter 6 Notes (2)

ERRORS TO AVOID W/WHILE LOOPS

QUESTION: If we order five fence sections, how many fence posts do we need?

Example 1:

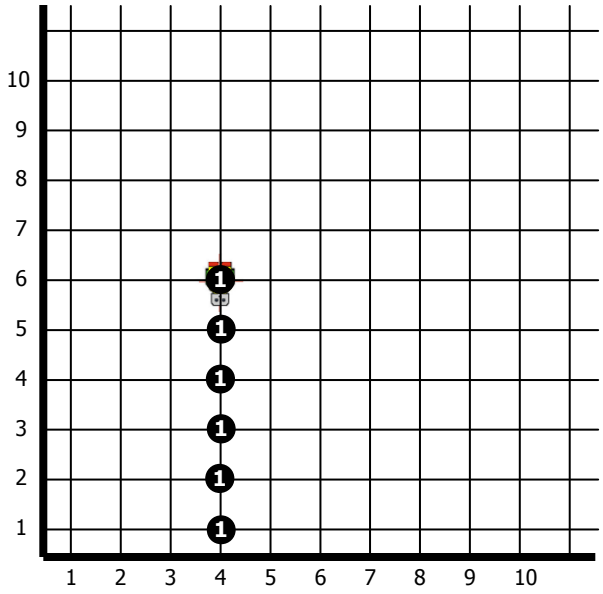
A robot named karel is somewhere in the world facing south. One beeper is on each corner between karel's current position and the southern boundary wall. There is a bepper on the corner on which karel is currently standing. Fix the method below, `clearAllBeepersToTheWall`, so that is correctly solves the problem.

```
public void clearAllBeepersToWall()
{

      while ( frontIsClear() )
      {
            pickBeeper();
            move();
      }




}
```



Four-step Process for Building a WHILE Loop:

Step 1: Identify the one test that must be true when karel is finished with the loop.

Step 2: Use the opposite form of the test identified in step 1 as the loop <test>.

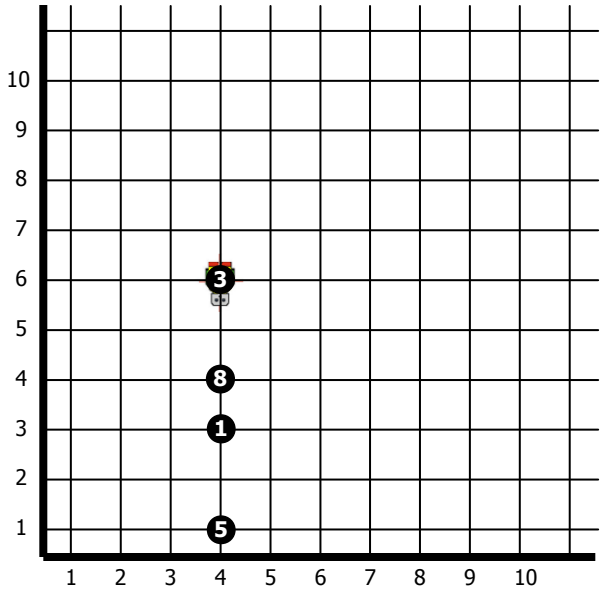
Step 3: Do the minimum needed to ensure that the test eventually evaluates to false so that the WHILE loop stops.

Step 4: Do whatever is required before or after the WHILE is executed to ensure we solve the given problem.

Nested WHILE Loops:

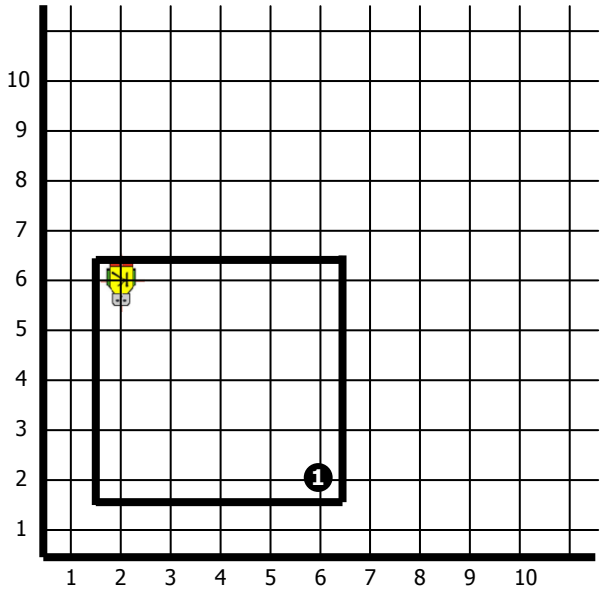
Example 2:

A robot named karel is somewhere in the world facing south. Between its current location and the southern boundary wall are beepers. We do not know how many beepers are on each corner (some corners may even have no beepers). Write a new method that will direct karel to pick all the beepers between its current location and the southern boundary wall.



Example 3:

A robot named karel is facing south in the northwest corner of a room that has no doors or windows. Somewhere in the room, next to a wall, is a simple beeper. Instruct karel to find the beeper by writing the new method, `findBeeper`.



How to Reason the Correctness of WHILE loops:

1. Show that the instruction works correctly when the initial situation results in the test being false.

2. Show that each time the loop body is executed, the robot's new situation is a simpler and similar version of the old situation.

We'd like to spend a little more time discussing this last concept of correctness. Remember that a robot will do exactly what it is told and only what it is told. It is up to us to make sure that it is provided with a correct way to solve a given problem. At the time, we need a way to "prove" (in some sense) that our solution is correct. It will be impossible for us to simulate every possible situation, so we need a way to think through our solution in a formal way to verify that it does what it is supposed to do.

In order to reason about WHILE loops, we will need to understand a key concept called a **loop invariant**. A loop invariant is an assertion (something that can be proven true or false) which is true after each iteration of the loop. For our purposes, loop invariants will be assertions about the robot's world. In particular, the items that we need to be concerned about after ONE iteration are the following:

* How has the robot's direction changed, if at all?

* How has the robot's relative position in the world changed, if at all (this may involve thinking about wall segments as well)?

* How has the number of beepers in the robot's beeper-bag changed, if at all?

* How has the number of beepers in the robot's beeper-bag changed, if at all?

Example 4:

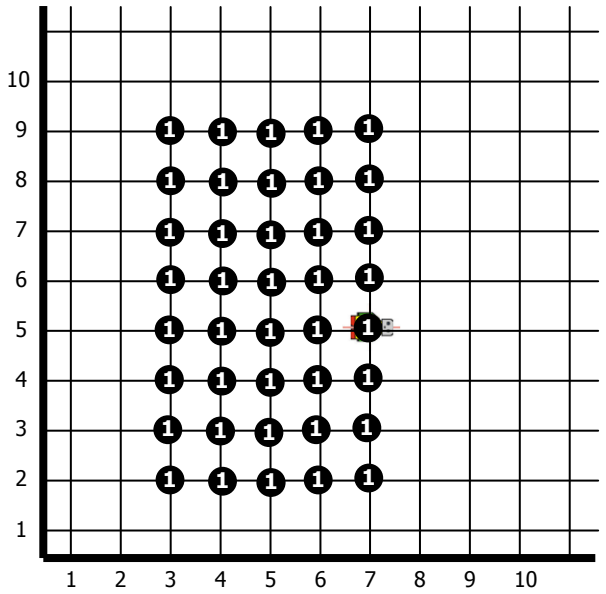
What is the loop invariant for the first attempt at solving the fencepost problem from Example 1?

```
while ( frontIsClear() )
{
     pickBeeper();
     move();
}
```

Example 5:

Let's write a complex program using stepwise refinement. Suppose that we need to patrol the perimeter of a rectangular field of beepers. Imagine that there has been theft of the beepers and we need a robot guard to walk around the edge of the field. Build a class of Guard robots with a method `walkPerimeter`. The robot will initially be positioned somewhere in the field, but not necessarily on an edge.

To make the problem more definite, let's suppose that the field is at least two beepers wide and two beepers long. The path we want the robot to follow is one that walks along corners that actually contain the beepers marking the outer edge of the field. Below is a sample situation.



Example 6:

Will your program from above work in the situation below? If not, fix it so that it does.