Eric Roberts
CS 106A

Handout #11
January 8, 2010

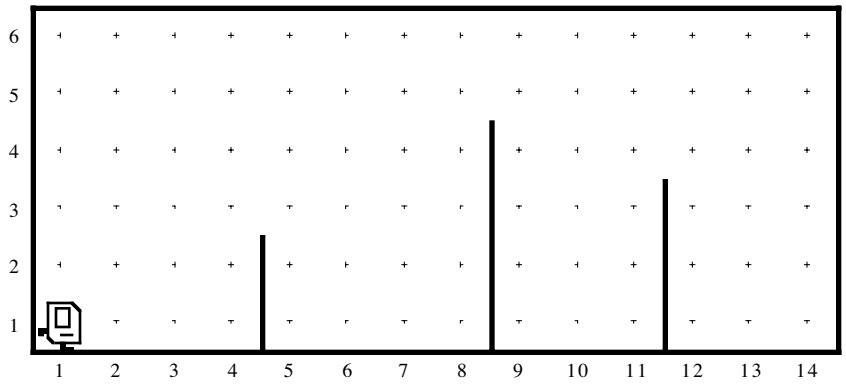
An Example of Stepwise Refinement

"sweet spring is your
time is my time is our
time for springtime is lovetime
and viva sweet love"

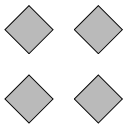
—e. e. cummings

For those who are not fortunate enough to live in California, winter maintains a solid hold on the world. The trees have lost their leaves and stand as empty monuments to the ravages of the season. And, even here, the recent cold snap has us longing for the coming of spring.

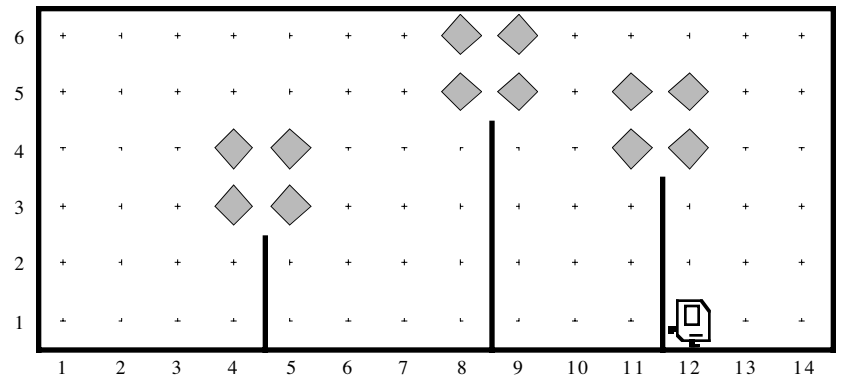
As our class example today, we'll have Karel act out the coming of springtime to a winter world that looks something like this:



In this sample world, the vertical wall sections represent barren tree trunks. In an attempt to add some semblance of life back to the world, Karel has taken on the mission of adorning the barren trees with a new set of leaves represented by beepers. Karel's plan is to climb each of the trees and adorn the top of each tree with a cluster of four leaves arranged in a square like this:



Thus, when Karel is done, the winter scene will look like this:



As in most Karel problems, the situation that Karel faces need not match exactly the one shown in the diagram. There may be more trees; Karel simply continues the process until there are no beepers left in the beeper bag. The trees may also be of different heights or spaced differently than the ones shown in the diagram. Your task is to design a program that is general enough to solve any such problem, subject to the following assumptions:

- Karel starts at the origin facing east, somewhere west of the first tree.
- The trees are always separated by at least two corners, so that the leaves at the top don't interfere with one another.
- The trees always end at least two corners below the top, so that the leaf cluster will not run into the top wall.
- Karel has just enough beepers to outfit all of the trees. The original number of beepers must therefore be four times the number of trees.
- Karel should finish facing east at the bottom of the last tree.

Think hard about what the parts of this program are and how you could break it down into simpler subproblems. What if there were only one tree? How does that simplify the problem, and how can you use the one-tree solution to help solve the more general case?

The solutions will be available on the web as Handout #11A.