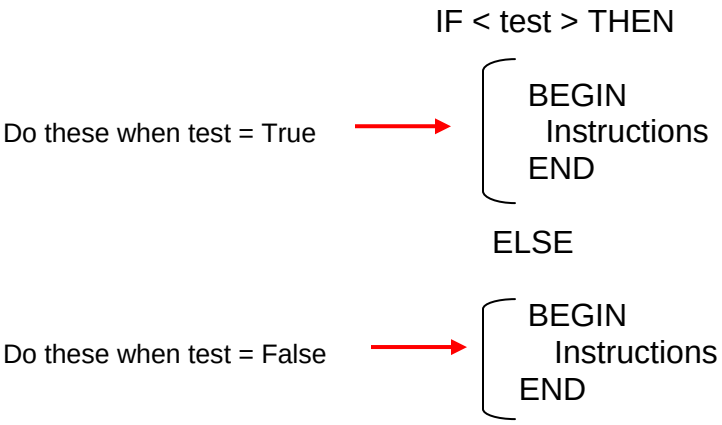
Karel the Robot – Making More Complex Decisions

IF / THEN / ELSE



Remember the possible <test> options are:

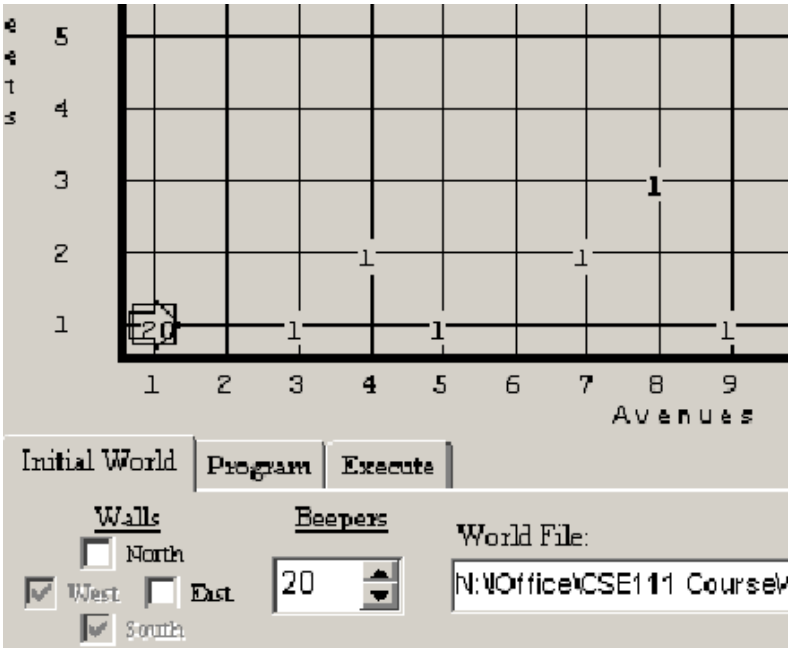
front-is-clear	front-is-blocked
left-is-clear	left-is-blocked
right-is-clear	right-is-blocked
next-to-a-beeper	not-next-to-a-beeper
any-beepers-in-beeper-bag	no-beepers-in-beeper-bag

Problem Statement:
Karel has been told to place two beepers on each street corner between 1st street and 2nd avenue and 1st street and 10th avenue. This would be a very easy task except there was a beeper party last night and there are beepers scattered around on random street corners.
Define Output: There will be two beepers on every corner on 1st street between 2nd avenue and 10th avenue.
Define Input: Karel is next to 20 beepers. He is at the Origin facing east.

Initial Algorithm

```
Pickup 18 beepers
Move
If next to a beeper, put a beeper down
Otherwise put down 2 beepers
Move
If next to a beeper, put a beeper down
Otherwise put down 2 beepers
Move
If next to a beeper, put a beeper down
Otherwise put down 2 beepers
. . .
turnoff
```

Karel the Robot – Making More Complex Decisions



Revised Algorithm

Pickup 18 beepers
Repeat 9 times
    Move
    If next to a beeper, put a beeper down
    Otherwise put down 2 beepers
turnoff

Why is the problem solved this way? Well, if there is already a beeper on the corner, Karel only needs to put one beeper down. If there are no beepers on the corner, then Karel needs to put 2 beepers down.

Let's examine the IF statement. Anytime we use a word like, OTHERWISE, we know we need to use an IF / THEN / ELSE.

    If next to a beeper, put a beeper down
    Otherwise put down 2 beepers

In Karel's language this would become:

```
IF next-to-a-beeper THEN
        BEGIN
                putbeeper;
        END
ELSE  (another word for Otherwise)
        BEGIN
                putbeeper;
                putbeeper;
        END;
```

Karel the Robot – Making More Complex Decisions

Given what we have above, we can write the program without further refining the algorithm.

```
beginning-of-program
   beginning-of-execution

      ITERATE 18 TIMES pickbeeper;

      ITERATE 9 TIMES
      BEGIN
         move;
         IF next-to-a-beeper THEN
            BEGIN
               putbeeper;
            END
         ELSE
            BEGIN
               putbeeper;
               putbeeper;
            END;
      END;

      turnoff;
   end-of-execution
end-of-program
```

Notice:
1) When Karen is next-to-a-beeper (the test is TRUE), one beeper get put down.
2) When Karen is NOT next-to-a-beeper (the test is FALSE), two beepers get put down.

3) BEGIN / END statements must match up or Karel (not to mention the programmer) gets very confused.