Writing more Efficient Programs, Making Decisions, ITERATE

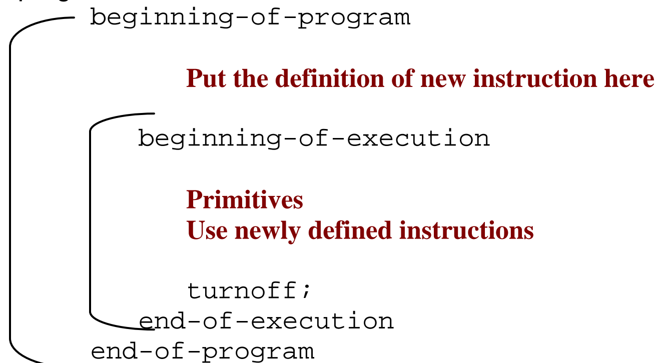
I. Reviewing Karel

1) Karel has a limited initial vocabulary of five terms:

move
pickbeeper
putbeeper
turnleft
turnoff

2) Statements in Karel's programs are separated by a semicolon (;)

3) Karel's programs have a standard format.



4) As indicated above, the programmer can define new instructions to add to Karel's vocabulary. These instructions go between the reserved words `beginning-of-program` and `beginning-of-execution`.

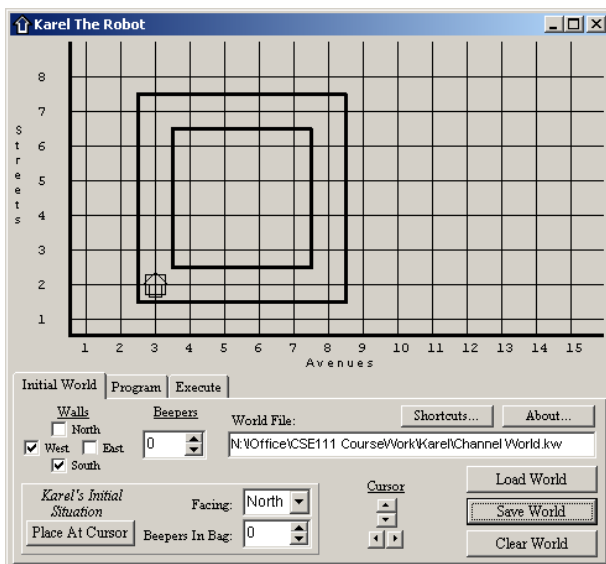
To define a new vocabulary word the programmer uses the command:

```
DEFINE-NEW-INSTRUCTION <new instruction here> AS
    BEGIN
        Instructions from primitives or;
        Instructions from other new instructions listed above this one;
    END;
```

Let's work through a review problem:

Problem Statement: Karel is to walk around the block. Karel must end up facing North.

Define Output: What will Karel's World look like?



Define Output: Karel needs to walk around the block making three turns, and each side of the street has a length of 5 streets.

Copyright © 2008 by Helene G. Kershner

Writing more Efficient Programs, Making Decisions, ITERATE

Define Input: Karel starts in the bottom left hand corner of the block facing North.

Define Initial Algorithm:

Move ahead 5 streets
Turn right
Move ahead 5 streets
Turn right
Move ahead 5 streets
Turn right
Move ahead 5 streets
Turn right

Refined Algorithm: We cannot go to code directly from this algorithm, because Karel will not understand many of the terms used. We need to define some new instructions.

```
Definitions
    Move 5 streets
    Turnright
-----------
Move ahead 5 streets
Turn right
Move ahead 5 streets
Turn right
Move ahead 5 streets
Turn right
Move ahead 5 streets
Turn right
```

Let's program this refined algorithm.

The first step is to create the new definitions.

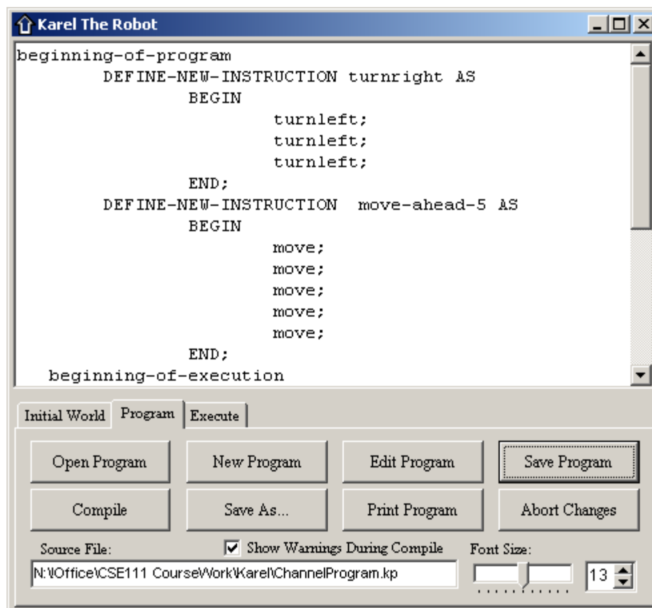
```
DEFINE-NEW-INSTRUCTION  turnright AS
    BEGIN
        turnleft;
        turnleft;
        turnleft;
    END;

DEFINE-NEW-INSTRUCTION  move-ahead-5 AS
    BEGIN
        move;
        move;
        move;
        move;
        move;
    END;
```

This part of the program has been entered into Karel's program view.

Copyright © 2008 by Helene G. Kershner

Writing more Efficient Programs, Making Decisions, ITERATE



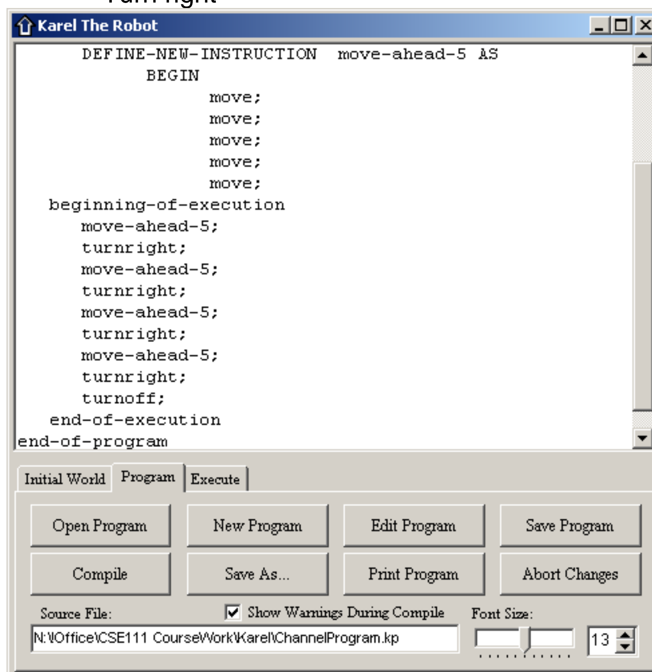
Now we can enter the rest of our code based on the refined algorithm.

Definitions

Move 5 streets

Turnright

Move ahead 5 streets
Turn right
Move ahead 5 streets
Turn right
Move ahead 5 streets
Turn right
Move ahead 5 streets
Turn right



Copyright © 2008 by Helene G. Kershner

Writing more Efficient Programs, Making Decisions, ITERATE

Now let's compile and execute our program and see what happens.

If you type the program in and correct any syntax errors that may occur (there are none in the program as shown -- I corrected them all first!)

II. Writing Efficient Code

How many instructions did we require Karel to perform?
Let's see.

```
(33)
beginning-of-program
   DEFINE-NEW-INSTRUCTION turnright AS
   BEGIN
      turnleft;
      turnleft;   (3)
      turnleft;
   END;
   DEFINE-NEW-INSTRUCTION move-ahead-5 AS
   BEGIN
      move;
      move;
      move;   (5)
      move;
      move;
   END;
   beginning-of-execution
      move-ahead-5;                                            (5)
      turnright;  →  this is three turnleft instructions (3)
      move-ahead-5;                                            (5)
      turnright;  →  this is three turnleft instructions (3)
      move-ahead-5;                                            (5)
      turnright;  →  this is three turnleft instructions (3)
      move-ahead-5;                                            (5)
      turnright;  →  this is three turnleft instructions (3)
      turnoff;                                                 (1)
   end-of-execution
end-of-program
```

Definitions are only executed when they are needed as part of the "main" program. So, those instructions, while critical to Karel's operation in this program only count when they are used. This program requires that Karel execute 33 instructions.

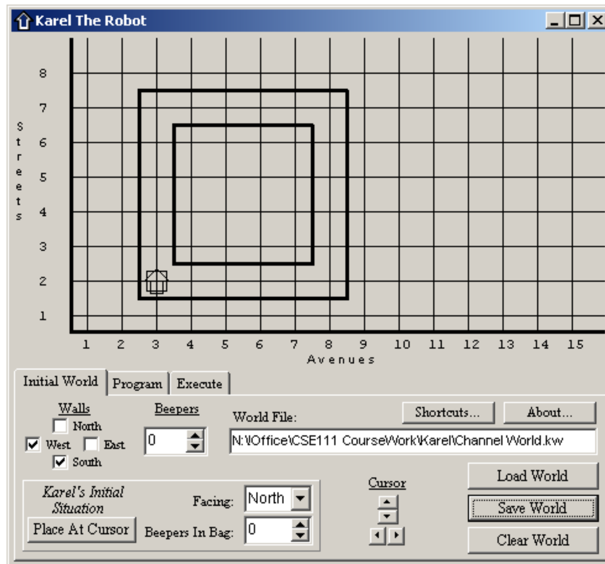
Now, notice something. Our program has Karel making four right turns. What this really means is that Karel is making 12 left turns as part of completing this task. Could we have written the program so that Karel had less work to do?

Think?

Karel is facing North. Because of this, we just had him go straight ahead and then make a right hand turn. But if Karel were facing East instead, we would have Karel go straight ahead and then make left turns which are much simpler for our Robot.

Copyright © 2008 by Helene G. Kershner

Writing more Efficient Programs, Making Decisions, ITERATE



Notice, if we had Karel make a right turn at the beginning, he could make left turns after that and have many fewer instructions to follow.

Let's go back to our initial Algorithm and make some changes.

Define Initial Algorithm:

Move ahead 5 streets
Turn right
Move ahead 5 streets
Turn right
Move ahead 5 streets
Turn right
Move ahead 5 streets
Turn right

Instead of this idea let's try another.
Remember there is more than one right way to solve a problem. Any solution that works solves the problem. But we know that some methods are easier or quicker than others.

Define Initial Algorithm:

turnright
move ahead 5 streets
turnleft
move ahead 5 streets
turnleft
move ahead 5 streets
turnleft
move ahead 5 streets
turn-around

As we've seen before, Karel won't understand most of the terms in this algorithm so it needs to be refined. However, notice that the terms turnright and turn-around, are each only used one, we could choose to define these terms, or NOT, since we only use them once. To make our program easily readable by humans we'll create them.

Copyright © 2008 by Helene G. Kershner

Writing more Efficient Programs, Making Decisions, ITERATE

Refine Algorithm

Define
 turnright
 move-ahead-5
 turn-around

----

turnright
move ahead 5 streets
turnleft
move ahead 5 streets
turnleft
move ahead 5 streets
turnleft
move ahead 5 streets
turn-around

Now let's turn this program into code in Karel's program view.

Karel The Robot

```
beginning-of-program
   DEFINE-NEW-INSTRUCTION turnright AS
   BEGIN
      turnleft;
      turnleft;
      turnleft;
   END;
   DEFINE-NEW-INSTRUCTION  move-ahead-5 AS
   BEGIN
      move;
      move;
      move;
      move;
      move;
   END;
   DEFINE-NEW-INSTRUCTION turn-around AS
   BEGIN
      turnleft;
      turnleft;
   END;
   beginning-of-execution
      turnright;
      move-ahead-5;
      turnleft;
      move-ahead-5;
      turnleft;
      move-ahead-5;
      turnleft;
      move-ahead-5;
      turn-around;
      turnoff;
   end-of-execution
end-of-program
```

Initial World | Program | Execute

Open Program | New Program | Edit Program | Save Program
Compile | Save As... | Print Program | Abort Changes

Source File: N:\\Office\CSE111 CourseWork\Karel\quickerchannelProgram.kp
Show Warnings During Compile
Font Size: 13

If we count the number of instructions Karel has to execute in this version of the program, you will find that Karel is performing less instructions so this program will run faster.

Copyright © 2008 by Helene G. Kershner

Writing more Efficient Programs, Making Decisions, ITERATE

```
(29)
beginning-of-program
   DEFINE-NEW-INSTRUCTION turnright AS
   BEGIN
      turnleft;
      turnleft;      (3)
      turnleft;
   END;
   DEFINE-NEW-INSTRUCTION  move-ahead-5 AS
   BEGIN
      move;
      move;
      move;          (5)
      move;
      move;
   END;
   DEFINE-NEW-INSTRUCTION turn-around AS
   BEGIN
      turnleft;      (2)
      turnleft;
   END;
   beginning-of-execution
      turnright;     (3)
      move-ahead-5;  (5)
      turnleft;      (1)
      move-ahead-5;  (5)
      turnleft;      (1)
      move-ahead-5;  (5)
      turnleft;      (1)
      move-ahead-5;  (5)
      turn-around;   (2)
      turnoff;       (1)
   end-of-execution
end-of-program
```

While the difference between 33 instructions and 29 is not huge, as we watch Karel on the screen it is clear that even this savings matters.

III. New Language Tool for Karel -- IF/THEN and IF/THEN/ELSE Or -- How to get Karel to Make Decisions!

Earlier in the semester we spent quite a bit of time working with Logic. The first part of our discussion of Logic began with being able to decide if a statement is True or False. Karel's programming language has a way to have Karel make some decisions based on his world. The IF/THEN (or IF/THEN/ELSE) structure is the statement Karel's language uses to make choices.

The IF/THEN Instruction:

```
IF <test condition> THEN
     <instruction>
```

As with or definitions, even though only one instruction TECHNICALLY follows the word THEN, by using the BEGIN/END idea we can have multiple instructions wrapped up to look like one.

Copyright © 2008 by Helene G. Kershner

Writing more Efficient Programs, Making Decisions, ITERATE

What is really going on with this instruction? How does it reflect a decision?

```
IF <test condition> THEN
        <instruction>
```

Karel looks at the <test condition>. Whenever the <test condition> is TRUE, Karel does whatever instructions follow the THEN, Afterwards Karel goes back to following instructions one at a time.

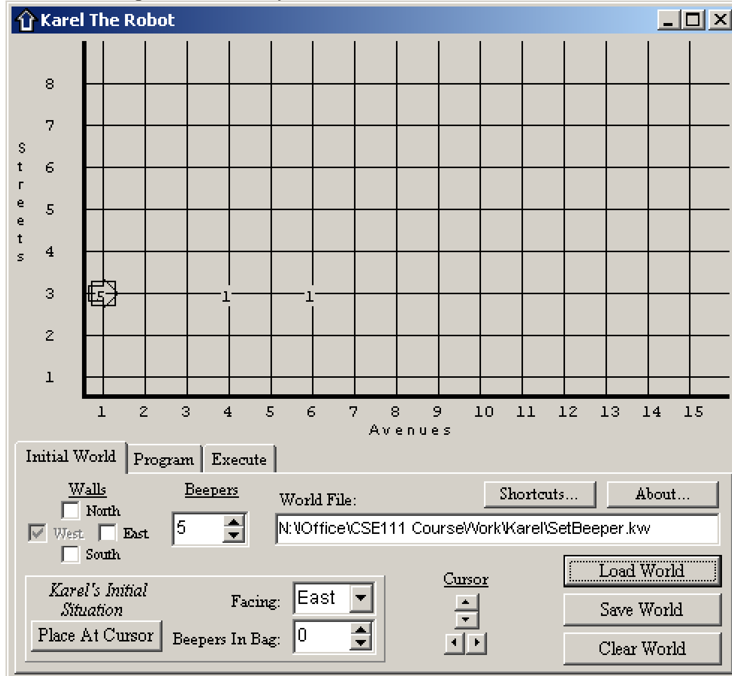
On the other hand, if the <test condition> is FALSE, Karel ignores the instructions after the THEN and continues to follow the rest of the instructions in the program one at a time.

Just like the rest of Karel's world, there are a limited number of things that Karel can test for. After all his is a dimwitted Robot.   These 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

Let's look at a Problem where we might want to use this new statement.

Problem Statement: Karel has 5 beepers in his beeper bag. His task is to make sure that the five corners between 3rd Street and 3rd Avenue and 3rd Street and 7th Avenue all have one beepers. Then Karel is to go Home. If you look at the World below, two of these corners already have beepers on them.



This is a perfect problem for Karel's programmer to uses the IF/THEN statement. If the corner already has a beeper, skip it. Otherwise put a beeper on the corner. The programmer doesn't have to know the details of the corners. In fact, if the beepers were magically moved to other corners on 3rd Street between 3rd Avenue and 7th Avenue, the same program should still work.

Copyright © 2008 by Helene G. Kershner

Writing more Efficient Programs, Making Decisions, ITERATE

We also want to make our program as general as possible so we can create definitions where possible. In addition, if Karel can do some of the work for us, we want to allow Karel to do so.

Define Output: There must be a beeper at every corner on 3rd Street between 3rd Avenue and 7th Avenue.
Define Input: Karel has to put 5 beepers in his bag. He is located on 3rd Street and 1st Avenue facing East.
Initial Algorithm:

Put 5 beepers in the beeper bag
Move 2 avenue blocks
Put down beeper
Move 1 avenue block
Already a beeper so skip
Move 1 avenue block
Already a beeper so skip
Move 1 avenue block
Put down beeper
Move 1 avenue block
Put down beeper
Turn around
Move 6 avenue blocks
Turn left
Move 2 avenue blocks
Turn around

We know from previous problems that many of the words used ahead, Karel cannot understand. In addition we don't have to do the work for Karel of checking to see if there is a beeper. So, let's Refine the program and use the If/Then to have Karel check the corners for us.

Refine the algorithm:

Put 5 beepers in the beeper bag
Move 2 avenue blocks
If no beeper, put one down
Move 1 avenue block
If no beeper, put one down
Move 1 avenue block
If no beeper, put one down
Move 1 avenue block
If no beeper, put one down
Move 1 avenue block
If no beeper, put one down
Turn around
Move 6 avenue blocks
Turn left
Move 2 avenue blocks
Turn around

Well, this is a simpler program, and we will learn how to repeat the same set of instructions later this semester.

What might this program look like?
We already know how to create a turn around statement so we should do that.

Our program would begin something like this.

Copyright © 2008 by Helene G. Kershner

Writing more Efficient Programs, Making Decisions, ITERATE

```
beginning-of-program
DEFINE-NEW-INSTRUCTION turn-around AS
        BEGIN
                turnleft;
                turnleft;
        END;
    beginning-of-execution
        move;
        move;
        IF not-next-to-a-beeper THEN putbeeper;
        move;
        IF not-next-to-a-beeper THEN putbeeper;
        …
```

Let's write this out completely in Karel's program world and see what happens.

Refine the Algorithm Again:

Define

Turn-around

If no beeper, put one down

Put 5 beepers in the beeper bag
Move 2 avenue blocks
Check-beeper
Move
Check-beeper
Move
Check-beeper
Move
Check-beeper
Move
Check-beeper
Turn-around
Move 6 avenue blocks
Turnleft
Move 2 avenue blocks
Turn-around

Write the Program:

```
beginning-of-program
   DEFINE-NEW-INSTRUCTION turnaround AS
   BEGIN
      turnleft;
      turnleft;
   END;
   DEFINE-NEW-INSTRUCTION check-beeper AS
   BEGIN
      IF
      not-next-to-a-beeper
      THEN
         putbeeper;
   END;
```

Copyright © 2008 by Helene G. Kershner

```
 beginning-of-execution
     pickbeeper;
     pickbeeper;
     pickbeeper;
     pickbeeper;
     pickbeeper;
     move;
     move;
     check-beeper;
     move;
     check-beeper;
     move;
     check-beeper;
     move;
     check-beeper;
     move;
     check-beeper;
     turnaround;
     move;
     move;
     move;
     move;
     move;
     move;
     turnleft;
     move;
     move;
     turnaround;
     turnoff;
   end-of-execution
end-of-program
```

IV: ITERATE

Whenever there are instructions that are repeated, they can be reduced by using the Iterate command.

- The ITERATE command enables the programmer to have Karel repeat an instruction or set of instructions a fixed number of times.

ITERATE <positive number> TIMES <instruction>

- The <instruction> can actually be a group of instructions enclosed by a BEGIN and END

The program above can be rewritten making significant use of the ITERATE command.

Copyright © 2008 by Helene G. Kershner

Writing more Efficient Programs, Making Decisions, ITERATE

```
beginning-of-execution
      pickbeeper;
      pickbeeper;
      pickbeeper;
      pickbeeper;
      pickbeeper;
      move;
      move;
      check-beeper;
      move;
      check-beeper;
      move;
      check-beeper;
      move;
      check-beeper;
      move;
      check-beeper;
      turnaround;
      move;
      move;
      move;
      move;
      move;
      move;
      turnleft;
      move;
      move;
      turnaround;
      turnoff;
   end-of-execution
end-of-program
```

(Beside the five pickbeeper instructions:) this is a great place to use ITERATE
ITERATE 5 TIMES pickbeeper;
;

(Beside the six move instructions after turnaround:) this is another really good place to use ITERATE
ITERATE 6 TIMES move;

So, minimally this program can be reduced as follows:

```
      beginning-of-program
         DEFINE-NEW-INSTRUCTION turnaround AS
         BEGIN
            turnleft;
            turnleft;
         END;
         DEFINE-NEW-INSTRUCTION check-beeper AS
         BEGIN
            IF not-next-to-a-beeper THEN
               putbeeper;
         END;
         beginning-of-execution
            ITERATE 5 TIMES pickbeeper;
            move;
            move;
            check-beeper;
            move;
            check-beeper;
            move;
            check-beeper;
            move;
            check-beeper;
            move;
```

Notice that the set of instructions
move;
check-beeper;
are repeated a number of times, 4 to be exact these can also be replaced by an ITERATE statement. See below.

Copyright © 2008 by Helene G. Kershner

```
      check-beeper;
      turnaround;
      ITERATE 6 TIMES move;
      turnleft;
      move;
      move;
      turnaround;
      turnoff;
   end-of-execution
end-of-program
```

More than just individual instructions can be used in the ITERATE statement. When we put BEGIN and END around a group of statements, they can be used as part of the ITERATE statement. Notice in this revision of the program that the ITERATE statement is used to simplify the code even more.

```
beginning-of-program
   DEFINE-NEW-INSTRUCTION turnaround AS
   BEGIN
      turnleft;
      turnleft;
   END;
   DEFINE-NEW-INSTRUCTION check-beeper AS
   BEGIN
      IF
      not-next-to-a-beeper
      THEN
         putbeeper;
   END;
   beginning-of-execution
      ITERATE 5 TIMES pickbeeper;
      move;
      move;
      ITERATE 4 TIMES
            BEGIN
                  check-beeper;
                  move;
            END;
            check-beeper;
      turnaround;
      ITERATE 6 TIMES move;
      turnleft;
      move;
      move;
      turnaround;
      turnoff;
   end-of-execution
end-of-program
```

Copyright © 2008 by Helene G. Kershner