Karel the Robot
Extending the Primitive Commands

We know how to write programs using Karel's primitive commands
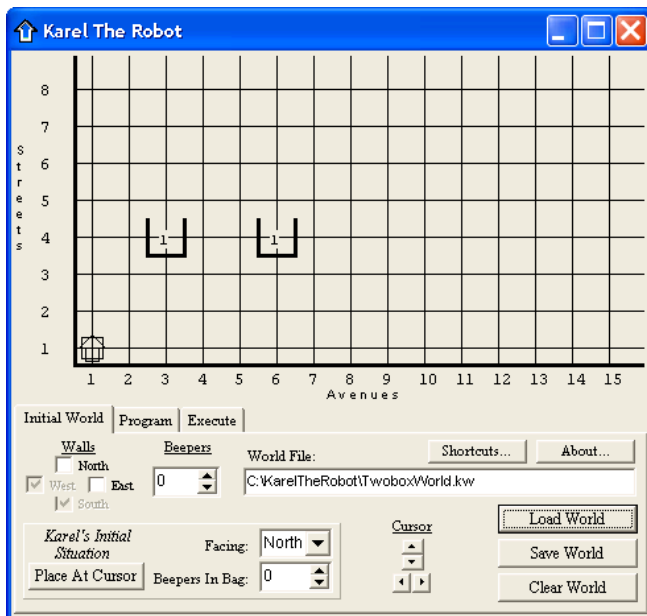
> move
> turnleft
> pickbeeper
> putbeeper
> turnoff

We know how to navigate between Karel's World view, Karel's Program view and Karel's Execution (or Run) view.

We know how to Compile a program, which means to translate it from Karel's programming language, which humans can understand as well as Karel into machine code that the computer inside Karel understands.

AND, we immediately saw that writing programs with only the basic 5 primitives is very tedious. It is hard to keep track of exactly what we are asking Karel to do.

We have already seen how Karel can to go out into the world and retrieve two beepers from two boxes, bring the beepers home and deposit them at the origin. The World view looked something like this:

It was painful to keep track of all the move instructions and all the turnleft commands.

We began the programming process with a problem statement:

Problem Statement: Karel's world contains two boxes, each with one beeper. Go out into Karel's world, retrieve the beepers from the boxes, return to the origin take beepers out of beeper bag.

We then defined the end result of the program (Output) and the starting state of our program (Input)

> Define the Output
>> Return to Origin with two beepers in beeper bag
>> Place two beepers at Origin
> Define the Input
>> Start at Origin, facing North
>> There are NO beepers in the beeper bag

Karel the Robot
Extending the Primitive Commands

Working with a problem statement, output and input we defined the initial version of our algorithm, our initial solution to Karel's problem.

        Karel starts at origin with no beepers
        Move Karel until one block North of beeper boxes
        Turn right
        Move to above first beeper box
        Turn to face down
        Pick up beeper
        Turn around
        Move one block
        Turn right
        Move 3 blocks
        Turn to face down
        Move one block
        Pick up beeper
        Turn around
        Move one block
        Go back to origin
        Put down two beepers
        Turn off.

Looking over this algorithm, we recognized that few if any of these statements were at a level that Karel could understand. Our next task was to translate the initial algorithm into Karel's primitive commands.

Our initial program looked something like this:

```
beginning-of-program
   beginning-of-execution
       move;
       move;
       move;
       move;
       turnleft;
       turnleft;
       turnleft;
       move;
       move;
       turnleft;
       turnleft;
       turnleft;
       move;
       pickbeeper;
       turnleft;
       turnleft;
       move;
       turnleft;
       turnleft;
       turnleft;
       move;
       move;
       move;
       turnleft;
       turnleft;
       turnleft;
       move;
       pickbeeper;
       turnleft;
       turnleft;
       move;
       turnleft;
```
additional instruction added here to bring Karel home

Karel the Robot
Extending the Primitive Commands

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

While this certainly solved Karel's problem statement, the constant repetition of commands to do simple tasks such as turn right or turn around made the process very tedious.

Defining New Instructions:
Since Karel is very good at following instructions, his designers included the concept of creating new instructions which simplifies the program for his human handlers but from Karel's perspective still uses just the 5 basic instructions.

Here is the structure of a Karel program. Notice that Karel's basic instructions are located between the Beginning-of-Execution and End-of-Execution. The definitions of new terms are placed between the Beginning-of-program and the Beginning-of-Execution.

BEGINNING-OF-PROGRAM

   Definitions

   BEGINNING-OF-EXECUTION

   Instructions

   END-OF-EXECUTION

END-OF-PROGRAM

Creating a new instruction is not a difficult task, and it is not complex. It **does** require care and attention to detail. The benefit is that the program can be much more natural and easy to understand.

To create a new instruction for Karel we use this command:

DEFINE-NEW-INSTRUCTION <new name here> AS
      Begin
      <instruction(s);>
      End;

The Begin and End indicate where the definition starts and finishes. **For every Begin, there must be a corresponding End.**

Let's try a simple new definition:

DEFINE-NEW-INSTRUCTION turnright AS
      Begin
                turnleft;
                turnleft;
                turnleft;
      End;

Our Karel programming language now contains the five primitives and the new instruction "turnright".
The following version of Karel's two box program uses turnright instead of three turnleft commands.

Karel the Robot
Extending the Primitive Commands

```
beginning-of-program
    define-new-instruction turnright as
    begin
        turnleft;
        turnleft;
        turnleft;
    end;
    beginning-of-execution
        move;
        move;
        move;
        move;
        turnright;
        move;
        move;
        turnright;
        move;
        pickbeeper;
        turnleft;
        turnleft;
        move;
        turnright;
        move;
        move;
        move;
        turnright;
        move;
        pickbeeper;
        turnleft;
        turnleft;
        move;
        turnleft;
  additional instruction added here to bring Karel home
        putbeeper;
        putbeeper;
        turnoff;
    end-of-execution
end-of-program
```

Not only is this program shorter than the earlier version but it is easier to understand.
Definitions are not standardized. The programmer can create them however the programmer chooses.
For example, the programmer might have decided to define the term turn-180 (which means turn-around)
and then use turn-180 to define turnaround. The basic program would be unchanged, but the definitions
section would contain two definitions one of which uses the other. This is perfectly legal. The definitions
section would look like:

```
beginning-of-program
    define-new-instruction turn-180 as
    begin
        turnleft;
        turnleft;
    end;
    define-new-instruction turnright as
    begin
        turn-180;
        turnleft;
    end;
    beginning-of-execution
```

Notice that the definition of turnright makes use of the turn-180 command. The most important concept here
is that a definition must be defined before it is used. So turn-180 had to be defined before turnaround.

Karel the Robot
Extending the Primitive Commands

Using the turn-180 definition we can simplify our initial program further making it easier to read and understand. From Karel's point of view nothing has changed. Karel is still using only his primitive commands. From the programmer's view the programming process is becoming much simpler.

```
beginning-of-program
    define-new-instruction turn-180 as
    begin
        turnleft;
        turnleft;
    end;
    define-new-instruction turnright as
    begin
        turn-180;
        turnleft;
    end;
    beginning-of-execution
        move;
        move;
        move;
        move;
        turnright;
        move;
        move;
        turnright;
        move;
        pickbeeper;
        turn-180;
        move;
        turnright;
        move;
        move;
        move;
        turnright;
        move;
        pickbeeper;
        turn-180;
        move;
        turnleft;
```
additional instruction added here to bring Karel home
```
        putbeeper;
        putbeeper;
        turnoff;
    end-of-execution
end-of-program
```

Reducing Repeated Instructions:
Looking at the program above, there is still a lot of repetition.
The Iterate (which means repeat) command can be used to simplify code where instructions are repeated over and over again. In the above program, if we were to bring Karel home, our instructions would look like this.
```
beginning-of-program
    define-new-instruction turn-180 as
    begin
        turnleft;
        turnleft;
    end;
    define-new-instruction turnright as
    begin
        turn-180;
        turnleft;
    end;
```

Karel the Robot
Extending the Primitive Commands

```
beginning-of-execution
    move;
    move;
    move;
    move;
    turnright;
    move;
    move;
    turnright;
    move;
    pickbeeper;
    turn-180;
    move;
    turnright;
    move;
    move;
    move;
    turnright;
    move;
    pickbeeper;
    turn-180;
    move;
    turnleft;
    move;
    move;
    move;
    move;
    move;
    turnleft;
    move;
    move;
    move;
    move;
    putbeeper;
    putbeeper;
    turnoff;
  end-of-execution
end-of-program
```

Notice, there are many places in the code where commands are repeated.

The Iterate command has the following structure:

Iterate <some number> times
        Begin
           Instruction(s);
        End;

The Iterate command can be used within a define a new instruction command or directly within the code. Here is an example of the program we have been using where the Iterate command replaces the many repetitions of the move command.

Looking at the compiled version of the code we can see that Karel's designers recognized the Iterate command as a repeat or loop and show us this visually in the compiled code.

Karel the Robot
Extending the Primitive Commands

```
1 beginning-of-program

   2  define-new-instruction turn-180 as
   3  begin

   4    turnleft;
   5    turnleft;
   6  end;

   7  define-new-instruction turnright as
   8  begin

   9    turn-180;
  10    turnleft;
  11  end;

  12  beginning-of-execution

  13    Iterate
  14    4 times
  15    begin
  16      move;
  17    end;
  18    turnright;
  19    move;
  20    move;
  21    turnright;
  22    move;
  23    pickbeeper;
  24    turn-180;
  25    move;
  26    turnright;
  27    Iterate
  28    3 times
  29    begin
  30      move;
  31    end;
  32    turnright;
  33    move;
  34    pickbeeper;
  35    turn-180;
  36    move;
  37    turnleft;
  38    Iterate
  39    5 times
  40    begin
  41      move;
  42    end;
  43    turnleft;
  44    Iterate
  45    4 times
  46    begin
  47      move;
  48    end;
  49    putbeeper;
  50    putbeeper;
  51    turnoff;
  52  end-of-execution
  53 end-of-program
```

Notice the circle or loop wherever the Iterate Command appears.