Karel the Robot

Extending the Primitive Commands

We know how to write programs using Karel's primitive commands

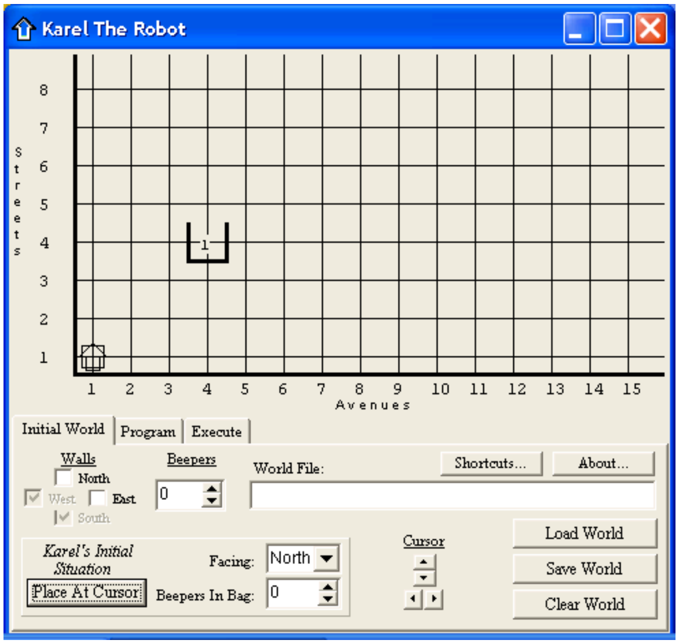
move
turnleft
pickbeeper
putbeeper
turnoff

We know have 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, writing programs with only the basic 5 primitives is hard to keep track of exactly what we are asking Karel to do. And it is tedious.

We have already seen how Karel can to go to a box with a beeper inside and retrieve the beeper and bring it home. The World view looked something like this:



☑ West □ North □ East ☑ South

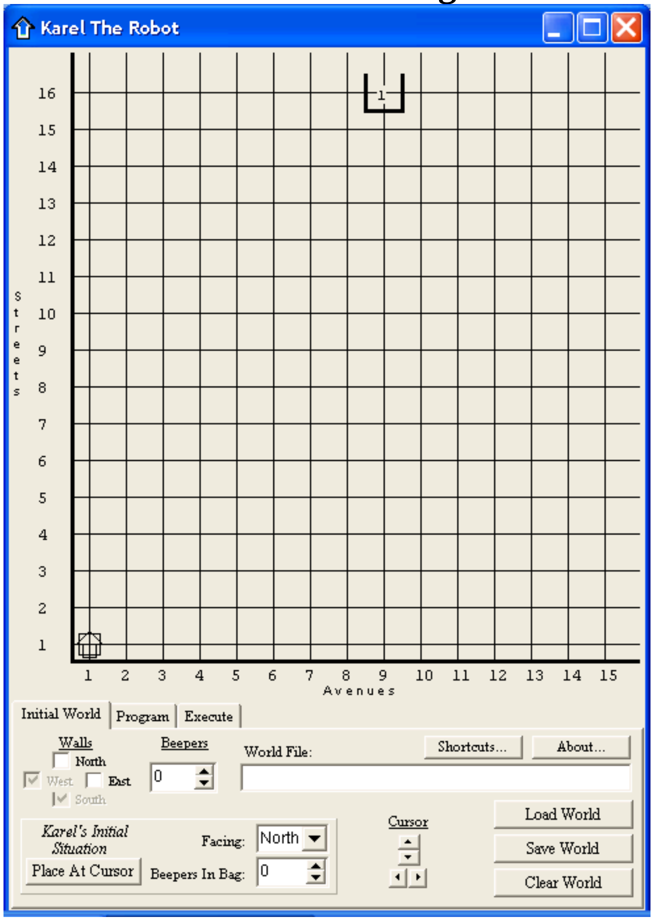
Beepers: 0

Facing: North

Beepers In Bag: 0

It was painful enough to keep track of all the move instructions and all the turnleft commands when Karel was close to the Origin (Home Base). Imagine if Karel's World looked like this:

Karel the Robot
Extending the Primitive Commands



Walls: □ North ☑ West □ East ☑ South

Beepers: 0

World File:

Karel's Initial Situation

Facing: North

Beepers In Bag: 0

We are solving essentially the same problem, retrieve the beeper from the box and come home, but now the number of move instructions makes the task incredibly tedious.

This is where the concept of creating new instructions for Karel becomes essential.

```
BEGINNING-OF-PROGRAM

    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 to the human in charge or Karel (the programmer) 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 <instruction>
```

Karel the Robot

Extending the Primitive Commands

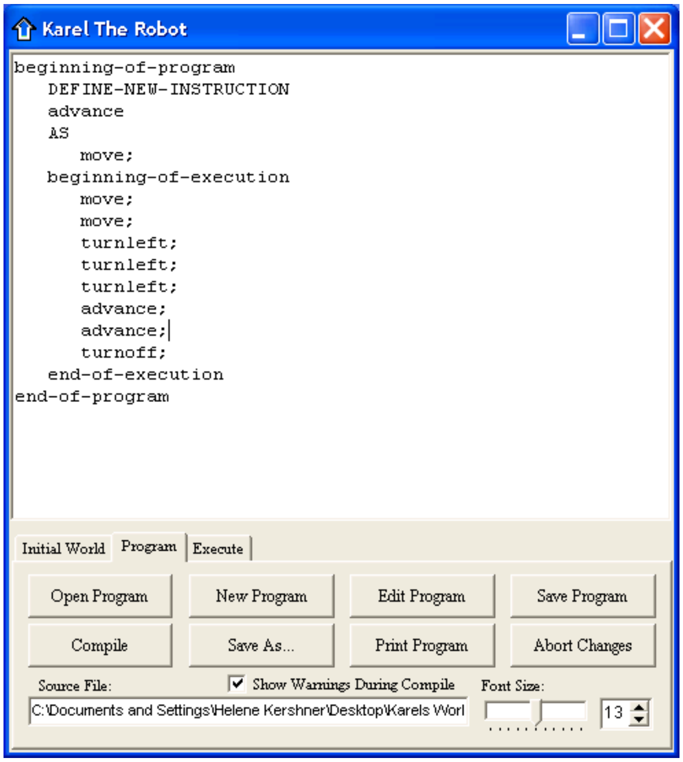
Now, technically only one single instruction can follow the AS

If that were really true, Karel would never get any easier to use.
While technically that is true, functionally there is an easy way around it.

Let’s try a simple new definition:

DEFINE-NEW-INSTRUCTION advance AS move;

Our Karel programming language now contains the five primitives and the new instruction “advance”



```
beginning-of-program
   DEFINE-NEW-INSTRUCTION
   advance
   AS
      move;
   beginning-of-execution
      move;
      move;
      turnleft;
      turnleft;
      turnleft;
      advance;
      advance;
      turnoff;
   end-of-execution
end-of-program
```

Source File: C:\Documents and Settings\Helene Kershner\Desktop\Karels Worl

☑ Show Warnings During Compile

Font Size: 13

This is not an especially useful new instruction because we could have used **move** every time we used **advance**.

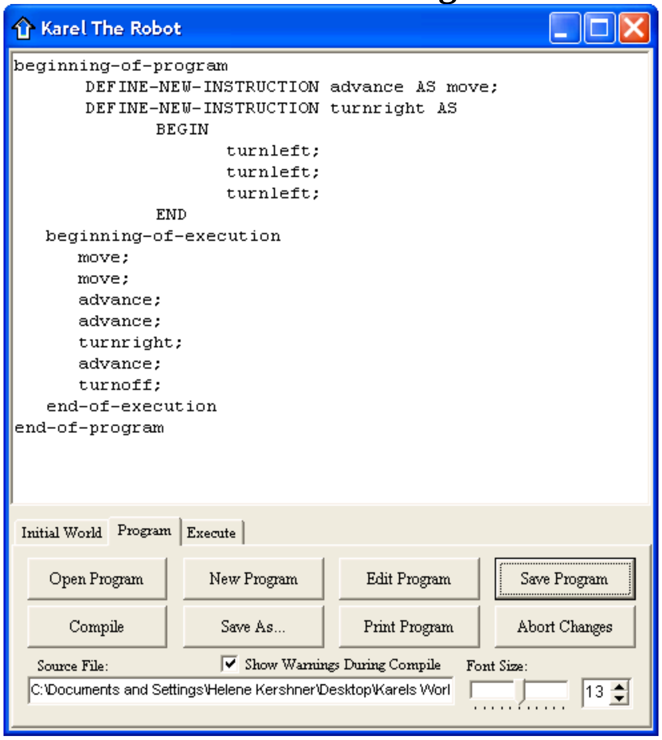
To build powerful new vocabulary for Karel we can replace the word instruction below with a group of instruction bounded by or captured between the words BEGIN and END.

DEFINE-NEW-INSTRUCTION <new name here> AS <instruction>

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

Look at the following program:

Karel the Robot
Extending the Primitive Commands



```
beginning-of-program
       DEFINE-NEW-INSTRUCTION advance AS move;
       DEFINE-NEW-INSTRUCTION turnright AS
              BEGIN
                     turnleft;
                     turnleft;
                     turnleft;
              END
   beginning-of-execution
      move;
      move;
      advance;
      advance;
      turnright;
      advance;
      turnoff;
   end-of-execution
end-of-program
```

Source File: C:\Documents and Settings\Helene Kershner\Desktop\Karels Worl

☑ Show Warnings During Compile

Font Size: 13

We have created two new instructions to increase Karel's vocabulary. As we have seen, one helps us very little. The **advance** instruction is identical to move and really adds nothing to Karel's vocabulary. But it is a perfectly legal new instruction.

On the other hand, the **turnright** instruction is very useful. It gives us, the programmers a shorthand for having to over-and-over-again type turnleft three times every time we want Karel to make a right turn.

Where do we put these new instructions and what form must they take?

```
beginning-of-program
```

Put the definition of new instruction here

```
   beginning-of-execution
```

Primitives
Use newly defined instructions

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

Let's enter our definitions.

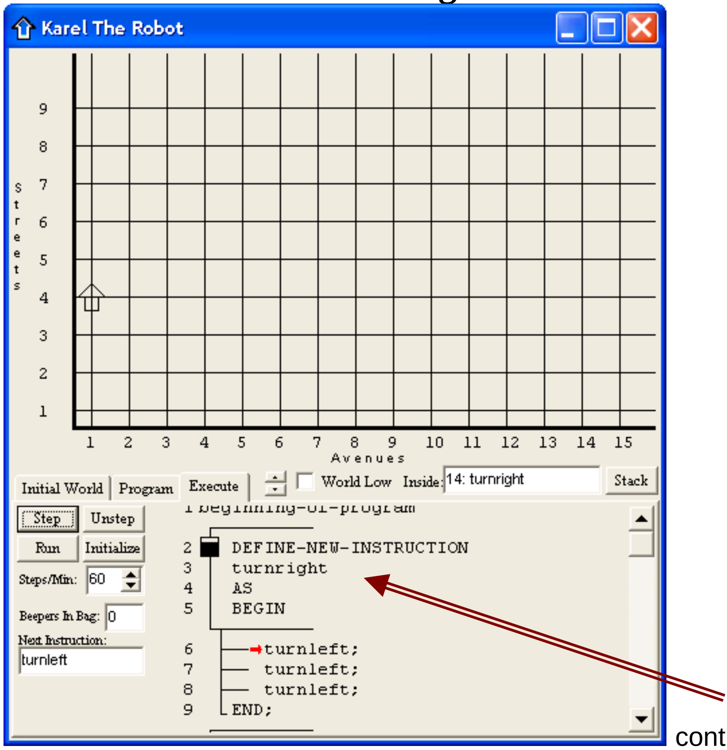
Karel the Robot

Extending the Primitive Commands

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

Now notice the instruction list as Karel is Executed. As Karel's programmer, the inclusion of the instruction turnright has made our tasks simpler. But, nothing has really changed for Karel. Every time the turnright instruction is used in our program Karel, simply looks up the new instruction, follows the definition exactly and then moves on. Control moves TEMPORARILY from our list of instructions between `beginning-of-execution` and `end-of-execution` to the definition section and then back again. Notice if our program had two turnright commands Karel would actually follow look up the definition twice and each time do exactly what it was told.

Karel the Robot
Extending the Primitive Commands



Inside: 14: turnright

Steps/Min: 60

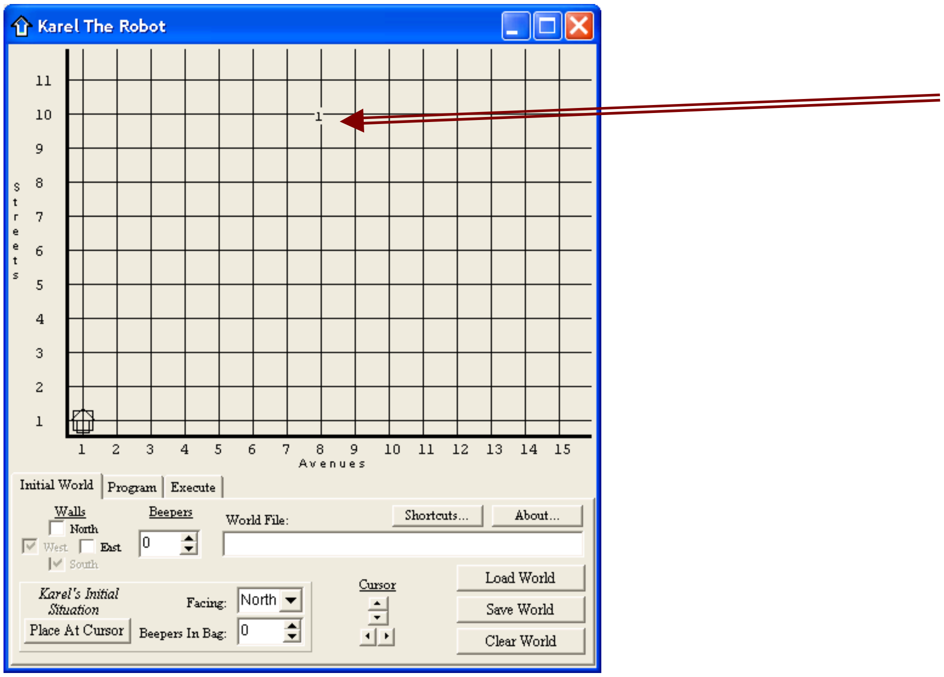
Beepers In Bag: 0

Next Instruction: turnleft

```
beginning-of-program
DEFINE-NEW-INSTRUCTION
turnright
AS
BEGIN
    turnleft;
    turnleft;
    turnleft;
END;
```

control has moved from the list of instruction to the turnright definition. When Karel finishes that instruction by turning left three times, control will move back to the main part of the program.

Has Karel really learned a new instruction?
No, every time Karel encounters the turnright command, he simply does turnsleft three times. However, as programmers we have gained flexibility. Now our programs can better reflect what we really want Karel to do. Let's look at another Example.



Walls: □ North ☑ West □ East ☑ South

Beepers: 0

World File:

Facing: North

Beepers In Bag: 0

Karel the Robot

Extending the Primitive Commands

In this World, we want Karel to retrieve a beeper and bring it home.
Karel is fairly far away from the beeper. The beeper is at 10th Street and 8th Avenue. Karel is at the Origin (Home Base) 1st Street and 1st Avenue. We need to use lot of move instructions to get Karel to the beeper and back.

Problem Statement: Karel starts at the Origin (Home Base) and needs to travel to 10th Street and 8th Avenue to retrieve a beeper. Karel then needs to come home, drops beeper at HomeBase and face North.

Define the Output: Get to 10th Street and 8th Avenue and pick up beeper. Then get home.
Define the Input: Start at Origin(HomeBase) with empty beeper-bag.

Initial Algorithm:

```
9 move instructions
turnright
7 moves
Pickup beeper
turn-around
7 moves
turnleft
9 move instructions
turn-around
drop off beeper
```

If we wrote out all the move instructions, and we could teach Karel what turnright and turn-around means. This program would work.

Refine Algorithm:

```
Define turnright as 3 turnleft instructions
Define turn-around as 2 turnleft instruction
------
9 move instructions
turnright
7 moves
Pickup beeper
turn-around
7 moves
turnleft
9 move instructions
turn-around
drop off beeper
```

The following program is exactly our refined algorithm written very carefully, using all the appropriate punctuation and headings required by Karel's language. AND, as instructed by our refined algorithm, we have defined turnright and turn-around.

Karel the Robot

Extending the Primitive Commands

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

This “main” program has **39** statements

This set of instructions solves our problem. It is as straightforward as it can get. It also has so many move instructions that it is easy to type in either too many or too few and then have to keep fixing out code.

Karel the Robot

Extending the Primitive Commands

We used 9 move instructions, followed by 7 move instructions, followed by 9 more and then 7 more. If we created a move4 instruction, our code would be little different for Karel, but much simpler for us.

What would our move4 instruction look like?

```
DEFINE-NEW-INSTRUCTION move4 AS
        BEGIN
                move;
                move;
                move;
                move;
        END;
```

This is essentially another Refinement of our Algorithm

Refine Algorithm Again:

```
        Define turnright as 3 turnleft instructions
        Define turn-around as 2 turnleft instruction
        Define move4 as 4 move instructions
        ------
        2 move4 instructions
        1 move instruction
        Turnright
        1 move4 instruction
        3 moves
        Pickup beeper
        turn-around
        1 move4 instruction
        3 moves
        turnleft
        2 move4 instructions
        1 move instruction
        turn-around
        drop off beeper
```

While our Refined Algorithm may look a bit longer, the actual program will be much shorter.

This is what our revised program would look like.

Extending the Primitive Commands

```
beginning-of-program
   DEFINE-NEW-INSTRUCTION move4 AS
   BEGIN
      move;
      move;
      move;
      move;
   END;
   DEFINE-NEW-INSTRUCTION turnright AS
   BEGIN
      turnleft;
      turnleft;
      turnleft;
   END;
   DEFINE-NEW-INSTRUCTION turn-around AS
   BEGIN
      turnleft;
      turnleft;
   END;
   beginning-of-execution
      move4;
      move4;
      move;
      turnright;
      move4;
      move;
      move;
      move;
      pickbeeper;
      turn-around;
      move4;
      move;
      move;
      move;
      turnleft;
      move4;
      move4;
      move;
      turn-around;
      putbeeper;
      turnoff;
   end-of-execution
end-of-program
```

This “main” program has **21** statements

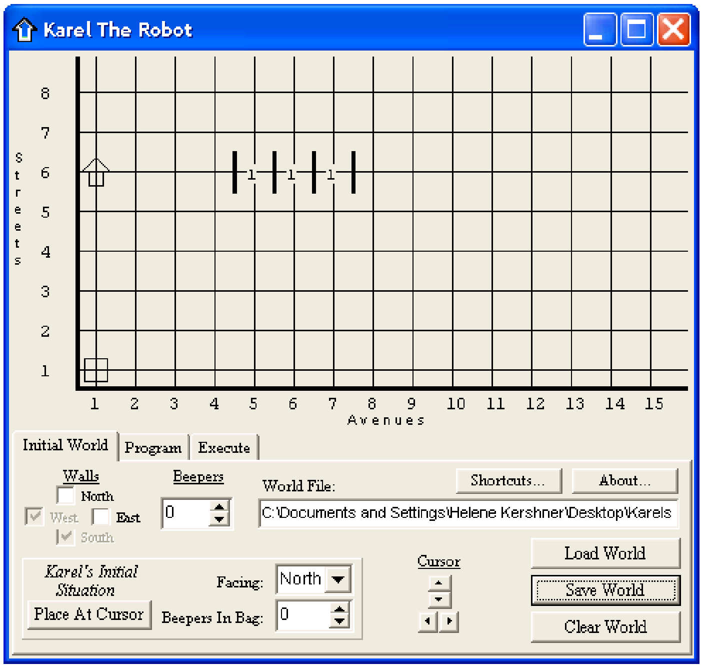
By using the move4 instruction, our “main” program (the part of the program without the definitions) has been reduced from 39 instructions down to 21 instructions. More important, this new program is easier to read by humans and thus easier to understand.

To simplify the “main” program we needed to carefully define three new instructions for Karel to use. We defined, turnright, turn-around and move4.

Let’s begin a totally new problem.

Karel the Robot
Extending the Primitive Commands

Karel's new World looks like this:



Beepers: 0

World File: C:\Documents and Settings\Helene Kershner\Desktop\Karels

Facing: North

Beepers In Bag: 0

Problem Statement: Karel is to go past each Hurdle, pick up the 3 beepers and place them behind the 4th wall.

Define Output: Three walls with beepers behind them, one wall at the end.
Define Input: Karel is at 6th Street and 1st Avenue. The Hurdles/Walls are to Karel's right

Initial Algorithm:

```
Turnright
Move 3 blocks
Go around first hurdle
Pickup beeper
Got around second hurdle
Pickup beeper
Go around third hurdle
Pickup beeper
Go around 4th hurdle
Drop off 3 beepers
```

As with the last program, while this algorithm solves the problem there are a lot of words in the algorithm that Karel cannot understand because there is nothing in his limited vocabulary to describe them.

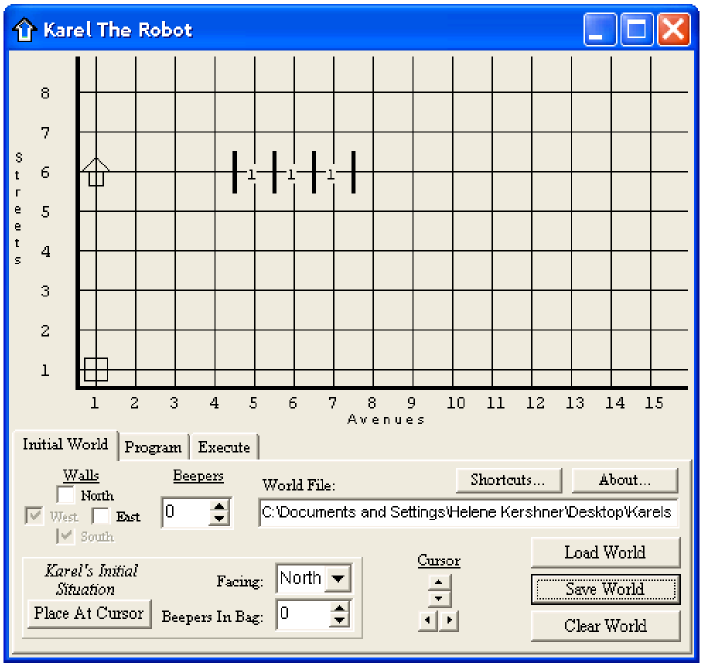
We will have to define turnright AND go around hurdle for a start.

Karel the Robot
Extending the Primitive Commands

Refine Algorithm:

```
        Define turnright
        Define go-around-hurdle
------------------
        Turnright
        Move 3 blocks
        Go around first hurdle
        Pickup beeper
        Got around second hurdle
        Pickup beeper
        Go around third hurdle
        Pickup beeper
        Go around 4th hurdle
        Drop off 3 beepers
```

Let's look at the definition section. Turnright we have created before. Go-around-hurdle requires a new definition. And, this definition is not simply a number of the same terms repeated.



Walls: ☑ West □ North □ East ☑ South

Beepers: 0

World File: C:\Documents and Settings\Helene Kershner\Desktop\Karels

Karel's Initial Situation

Facing: North

Beepers In Bag: 0

Let's pretend that Karel has moved from the starting location at 6th and 1st to being just before the first hurdle.

THINK!

What do we need Karel to do?

Creating definitions is often exactly the same as writing a program.

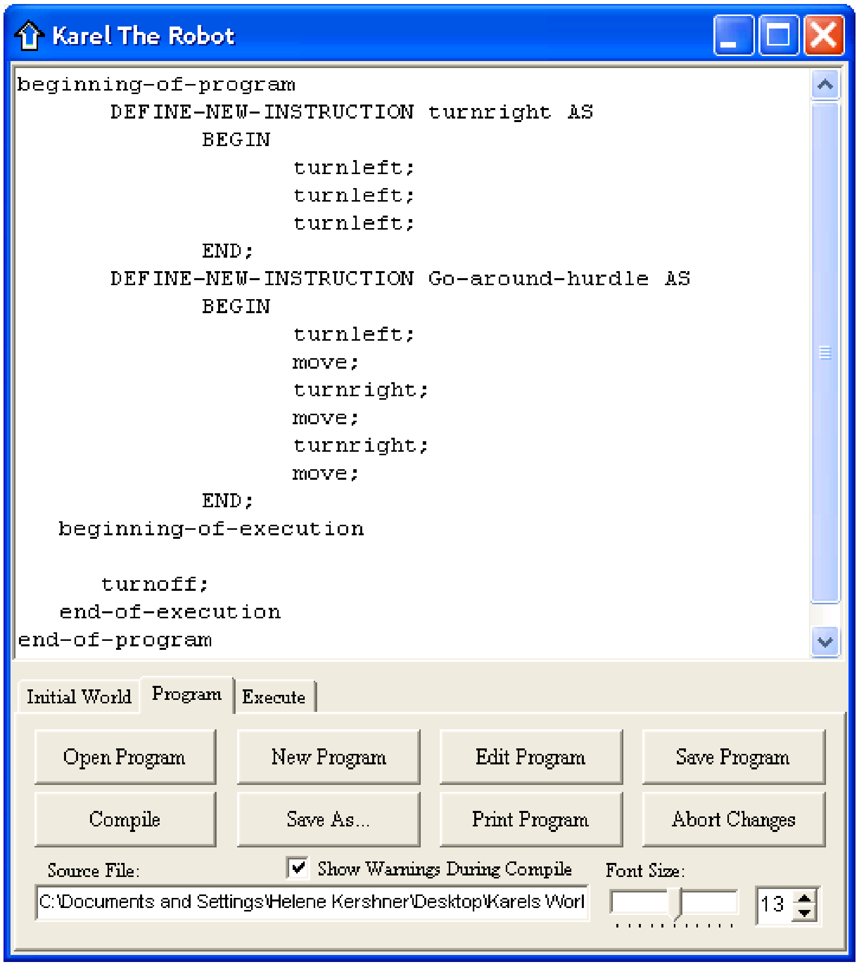
What do we need Karel to do to get past a hurdle?

```
        Turnleft
        Move 1 street
        Turnright
        Move 1 street
        Turnright
        Move 1 street
```

Karel the Robot

Extending the Primitive Commands

This set of instructions seems to get Karel around the wall and ready to pickup the beeper. Let’s enter the information into the Program section of Karel’s screen and see if we’ve given Karel enough information.



```
beginning-of-program
       DEFINE-NEW-INSTRUCTION turnright AS
             BEGIN
                    turnleft;
                    turnleft;
                    turnleft;
             END;
       DEFINE-NEW-INSTRUCTION Go-around-hurdle AS
             BEGIN
                    turnleft;
                    move;
                    turnright;
                    move;
                    turnright;
                    move;
             END;
   beginning-of-execution

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

Source File: C:\Documents and Settings\Helene Kershner\Desktop\Karels Worl

☑ Show Warnings During Compile

Font Size: 13

We’ve defined two new instructions.
We’ve defined turnright and Go-around-hurdle.

NOTICE: We defined a new instruction that makes use of another instruction.
IS this Legal?

As long as an instruction is defined before it is used, one new instruction can use another.

So, Go-around-hurdle could make use of the turnright instruction because it appears before the definition for Go-around-hurdle.

```
beginning-of-program
```

Put the definition of new instruction here

```
    beginning-of-execution
```

Definitions MUST be in the right location of the program or Karel will not be able to find them.

Karel the Robot

Extending the Primitive Commands

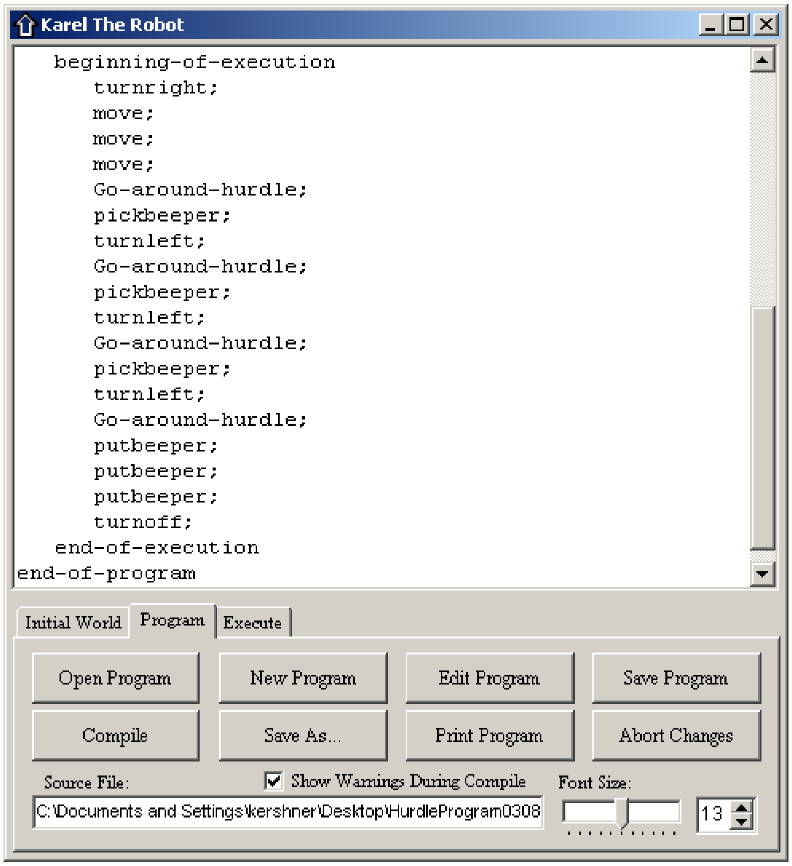
Now that we have our definitions in place (turnright and Go-around-hurdle) we can write our program.

Let's got back to our Refined Algorithm:

Refine Algorithm:

```
        Define turnright
        Define go-around-hurdle
------------------
        Turnright
        Move 3 blocks
        Go around first hurdle
        Pickup beeper
        Got around second hurdle
        Pickup beeper
        Go around third hurdle
        Pickup beeper
        Go around 4th hurdle
        Drop off 3 beepers
```

Looking at the vocabulary in our algorithm, is there anything remaining that we cannot translate directly into Karel's language. There does not seem to be. So let's just type in the code using the primitive commands. In other words, there the algorithm says, move 3 blocks, we would have to type: move; move; move;



```
   beginning-of-execution
      turnright;
      move;
      move;
      move;
      Go-around-hurdle;
      pickbeeper;
      turnleft;
      Go-around-hurdle;
      pickbeeper;
      turnleft;
      Go-around-hurdle;
      pickbeeper;
      turnleft;
      Go-around-hurdle;
      putbeeper;
      putbeeper;
      putbeeper;
      turnoff;
   end-of-execution
end-of-program
```

Source File: C:\Documents and Settings\kershner\Desktop\HurdleProgram0308

☑ Show Warnings During Compile

Font Size: 13

Our total program includes what is visible in this screen shot and, the previously created definitions.

Karel the Robot

Extending the Primitive Commands

Here is the entire program, which compiles without errors in spelling, punctuation or misunderstood words.

```
beginning-of-program
   DEFINE-NEW-INSTRUCTION turnright AS
   BEGIN
      turnleft;
      turnleft;
      turnleft;
   END;
   DEFINE-NEW-INSTRUCTION Go-around-hurdle AS
   BEGIN
      turnleft;
      move;
      turnright;
      move;
      turnright;
      move;
   END;
   beginning-of-execution
      turnright;
      move;
      move;
      move;
      Go-around-hurdle;
      pickbeeper;
      turnleft;
      Go-around-hurdle;
      pickbeeper;
      turnleft;
      Go-around-hurdle;
      pickbeeper;
      turnleft;
      Go-around-hurdle;
      putbeeper;
      putbeeper;
      putbeeper;
      turnoff;
   end-of-execution
end-of-program
```

When you Execute your program, watch Karel. You will notice that Karel really doesn’t “remember” the new vocabulary we have defined. Rather, each time Karel encounters a defined instruction, Karel looks it up does what it says and promptly forgets it again. So Karel doesn’t really KNOW how to turnright. Karel does know how to look up the definition, do three turnleft instructions. However, by creating these instructions, our programs become easier for people to read.