Karel the Robot

1. Nicolas Wirth ... “Algorithms + Data Structures => Programs”
   - deal with each separately ... algorithms first
   - algorithm ... “a finite set of unambiguous, executable instructions that will ultimately terminate if followed”

2. Concept of Karel... a language ... similar to standard procedural programming languages like Pascal, C, or BASIC
   - no data ... just algorithms
   - invented for teaching purposes ... has the essential elements of a commercial language ... but its fun

3. Karel’s World
   - streets and avenues
- walls

Fig. A: North/South Wall Segments

Fig. C: East/West Wall Segments

- beepers

- the beeper bag ... used to store beepers ... may be filled at start of program
- navigating in the world ... doing tasks given by the Robot Master (the program)

**Figure 2-8** The Mountain-Climbing Task

**Figure 2-10** The Rearrange the Beepers Task
5. Karel’s Capability to Act ... primitive actions
   - Move;
   - TurnLeft;
   - PickBeeper;
   - PutBeeper;
   - TurnOff;

6. Karel’s Capability of Sensing the Environment (Sensory Perceptions)
   ... these are Karel’s “tests”
   - Radar (Echo Locator)
     - front-is-clear/blocked, left-is-clear/blocked, right-is-clear/blocked
   - Hearing (Microphone)
     - next-to-a-beeper, not-next-to-a-beeper
   - Sensor Arm
     - any-beepers-in-beeper-bag, no-beepers-in-beeper-bag
   - Direction (Compass)
     - facing-north, facing-south, facing-east, facing-west
     - not-facing-north, not-facing-south, not-facing-east, not-facing-west

7. Karel’s Capability of Making Decisions
   IF <TEST> THEN <ACTION 1> ELSE <ACTION 2>;
   <TEST> is one of the boolean “tests” in item 6 above.

8. Karel’s Capability to Do Repetitive Things ... like a good Robot should
   - ITERATE N TIMES <ACTION>;
• WHILE <TEST> DO <ACTION>;

9. Karel’s Capability to Learn New Actions
   • DEFINE-NEW-INSTRUCTION <NAME> AS <ACTION>;

10. Karel’s Capability to do Complex Tasks
    • BEGIN ... END;

11. Karel has a specific program structure as follows:

BEGINNING-OF-PROGRAM

DEFINE-NEW-INSTRUCTION <name1> AS
    BEGIN
        <instruction 1>;
        <instruction 2>;
        ...
        <instruction n>;
    END;

DEFINE-NEW-INSTRUCTION <name2> AS
    BEGIN
        <instruction 1>;
        <instruction 2>;
        ...
        <instruction n>;
    END;

BEGINNING-OF-EXECUTION
    <instruction 1>;
    ...
    ...
    <instruction n>;
END-OF-EXECUTION

END-OF-PROGRAM
12. Karel Programming
   • Similar to DRC ... write program in an editor. Execute in a simulator.
   • Karel has hearing only (can’t read) and therefore can’t distinguish upper case, lower case letters
   • Karel needs a semi-colon to terminate instructions (;).
   • Karel is a very formal robot ... needs a specific structure to the programs
   • Karel’s syntax is very exact! It permits no errors.
   • To make programs more readable to humans, we:
     ♦ skip spaces, and
     ♦ indent

13. An example of a Karel problem. Karel is on 2\textsuperscript{nd} Street and 2\textsuperscript{nd} Avenue facing east. On the corner of 5\textsuperscript{th} Street and 4\textsuperscript{th} Avenue is a beeper. Karel’s job is to get the beeper and move it two blocks east and 3 blocks north.

14. Steps in the solution
   • Sketch the initial situation
   • Sketch the solution
   • Get the main idea ... algorithm ... “walk it through” with a partner or on paper
   • Write the program
   • Test the program
   • Fix errors
   • Repeat the “test and fix” until the program works correctly.
15. We are using a new simulator from Otterbein College. The link is at:

- [http://math.otterbein.edu/Class/Csc100/Karel/web/Karel/karel.htm](http://math.otterbein.edu/Class/Csc100/Karel/web/Karel/karel.htm)

- The link has a reasonably good tutorial.

- The new simulator introduces two changes in the syntax of Karel
  
  * **BEGIN / END is used to “block structure” the action in:**
    
    - **IF** <test>
      
      ```
      THEN BEGIN <action1> END
      ELSE BEGIN <action2> END;
      ```
      
    - **ITERATE N TIMES BEGIN** <action> **END;**
    
    - **WHILE** <test> **DO BEGIN** <action> **END;**
    
    - **DEFINE-NEW-INSTRUCTION** <name> **AS**
      
      ```
      BEGIN <actions> END;
      ```
  
  * **Statements are terminated with a semi-colon (;) not separated by a semicolon. This is a change from the version in the standard Karel textbook and is being done because of a new simulator we are using. It corresponds to the structure of C/Java rather than Pascal.**
BNF for Karel

BNF is the Backus-Naur Form of describing a programming language

 ::= this symbol is called a production. Each definition in BNF is called a production.

 words within the angle braces are names of objects

 a symbol which represents alternative choices

 anything within square braces is repeated 0 or more times

program ::= BEGINNING-OF-PROGRAM
          [<new-instruction>]
          BEGINNING-OF-EXECUTION
          <instruction> [<instruction>]
          END-OF-EXECUTION
          END-OF-PROGRAM

<new-instruction> ::= DEFINE-NEW-INSTRUCTION <name> AS
                    BEGIN <instruction> END ;

(instruction) ::= <primitive> | <conditional> | <repeat>

(primitive) ::= move; | turnleft; | pickbeeper; | putbeeper; | turnoff;

(conditional) ::= IF <test > THEN BEGIN <instruction> END;
                ELSE BEGIN <instruction> END;
                IF <test > THEN BEGIN <instruction> END;

(repeat) ::= WHILE <test > DO BEGIN <instruction> END;
           ITERATE <integer> TIMES BEGIN <instruction> END;

(test) ::= <side>-is-clear | <side>-is-blocked | next-to-a-beeper |
         not-next to a beeper | facing-<direction> |
         not-facing-<direction> | any-beepers-in-beeper-bag |
         no-beepers-in-beeper-bag
<side> ::= front | left | right
<direction> ::= north | south | east | west
<name> ::= <letter> [<character>]
<letter> ::= a..z | A..Z
<character> ::= <letter> | <digit> | -
<digit> ::= 0..9
<integer> ::= <digit> [<digit>] // practical limit of 5 digits
Karel Programming Summary

Primitive Instructions

- move;
- turnleft;
- pickbeeper;
- putbeeper;
- turnoff;

Conditional Instructions

- IF <test>
  THEN BEGIN <actions> END;
- IF <test>
  THEN BEGIN <actions1> END
  ELSE BEGIN <actions2> END;

Repetition Instructions

- ITERATE N TIMES BEGIN <actions> END;
- WHILE <test> DO BEGIN <actions> END;

Mechanism for Block Structuring

- BEGIN <actions> END;

Mechanism for Defining a New Instruction

- DEFINE-NEW-INSTRUCTION <name> AS
  BEGIN <actions> END;

Program Structure

BEGINNING-OF-PROGRAM

DEFINE-NEW-INSTRUCTION <name> AS
  BEGIN <actions> END;
DEFINE-NEW-INSTRUCTION <name> AS
  BEGIN <actions> END;
...
DEFINE-NEW-INSTRUCTION <name> AS
  BEGIN <actions> END;

BEGINNING-OF-EXECUTION

<instruction>
<instruction>
...
<instruction>

END-OF-EXECUTION

END-OF-PROGRAM

Tests

- 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
- facing-north, not-facing-north,
  facing-south, not-facing-south,
  facing-east, not-facing-east,
  facing-west, not-facing-west