Jerry Cain
CS 106AJ

Handout #9S
October 3 - 4, 2018

Section 1 Solution

Karel problem constructed by Eric Roberts and Chris Piech.

```
/*
 * File: UnitedNationsKarel.k
 * --------------------------
 * The UnitedNationsKarel program builds houses at corners
 * marked by rubble.
 */

import "turns";

function buildHouses() {
   while (frontIsClear()) {
      if (beepersPresent()) {
         pickBeeper();
         backUp();
         buildHouse();
      }
      if (frontIsClear()) {
         move();
      }
   }
}

/*
 * Function: buildHouse
 * --------------------
 * Builds a beeper house on stilts.
 * Precondition: Karel facing east at bottom of left stilt
 * Postcondition: Karel facing east at bottom of right stilt
 */

function buildHouse() {
   turnLeft();
   putThreeBeepers();
   move();
   turnRight();
   move();
   turnRight();
   putThreeBeepers();
   turnAround();
   move();
   turnRight();
   move();
   turnRight();
   putThreeBeepers();
   turnLeft();
}
```

```
/*
 * Function: putThreeBeepers
 * -------------------------
 * Creates a line of three beepers.
 * Precondition: Karel is in the first square in the line
 * Postcondition: Karel is in the last square in the line
 */
function putThreeBeepers() {
   repeat (2) {
      putBeeper();
      move();
   }
   putBeeper();
}

/*
 * Function: backUp
 * ----------------
 * Backs up one corner, leaving Karel facing in the same direction.
 * If there is no space behind Karel, it will run into a wall.
 */
function backUp() {
   turnAround();
   move();
   turnAround();
}
```

Try running this program on your own computer (you can cut/paste the code from this handout into JSKarel). What would you do if you wanted the houses to be taller? How would you make them 5 beepers high? Is there any way you could improve the style of the solutions?

Style Focus for Section 1:

Comments: Make sure to comment every function you write, and describe what the function does, and what the assumptions are before and after it is called. Write your comments so that your program could easily be understood by another person.

Good Function Names: Part of good style is good naming. You want your function name to succinctly describe what it does. Never call a function **`doStuff`**; rather, give it a good specific name like **`backUp`.** Be consistent in how you name your functions. In our solutions, we will use lower camel case naming conventions.

Short Functions: We could have written our whole program in the `buildHouses` function, but it is not good style and is difficult to follow. Try and break it down into functions that are small, easily understood snippets of code that accomplish one small task.