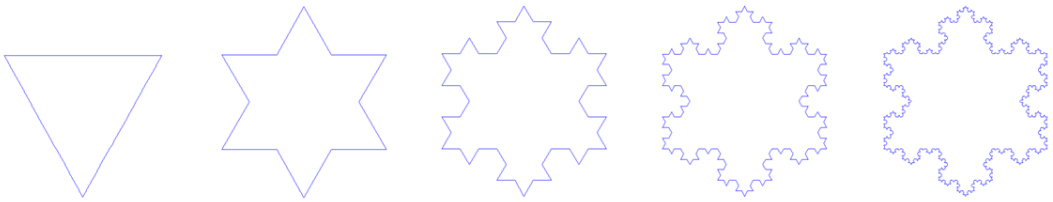
CS1114 Section 2/4 Exercises: Robot arena, recursion, and while loops



1 Robot arena

1. We've set up a virtual robot arena to test your code when all of the physical robots are in use. The first exercise is to start the robot arena, create a robot, and issue commands to it.

   (a) Download the robot arena zip file from the CS1114 Assignments Page into your `myfiles` directory.

   (b) Open a terminal, change to the `myfiles` directory and unzip the file with the command `unzip robotArena.zip`

   (c) Change to the robotArena directory (`cd robotArena`) and start the virtual arena by typing:

   ```
   > ./run.sh
   ```

   on the command prompt.

   This will open up a window containing an empty robot arena.

   (d) Next, in Matlab, use the `robotInit` function to create a virtual robot and connect to it. For the robot arena, this function takes two arguments. In Matlab, type:

   ```
   >> r = robotInit('localhost', 8000);
   ```

   (e) Now you can issue the virtual robot any of the normal robot commands. Try this by making the robot drive in a straight line:

   ```
   >> robotDriveStraight(r, 500, 500);
   ```

2 Recursion

In lecture last time we saw an example of *recursion* with the quicksort algorithm. A recursive algorithm (or function) is one that calls itself one or more times. In this exercise, you will explore a few simples examples of recursion, and see how algorithms can sometimes be implemented using either loops or recursion (you will also use a while loop in the second section).

2.1 Computing sums with recursion

For your first task, we will compute the "sumall" function. The sumall function takes a positive integer n, and is defined as the sum of all numbers between 1 and n:

$$\texttt{sumall}(n) = \sum_{i=1}^{n} i.$$

2. Write a Matlab function `sumall1` that takes a positive integer n and computes `sumall`(n). This function should use a **for loop**, and its header should be:

   ```
   function [ sum ] = sumall1(n)
   % Compute the sum of all numbers between 1 and n using a for loop
   ```

   This function should be saved in a file called `sumall1.m` (remember that you can type `edit sumall1.m` in Matlab to open an editor). Test out your function by calling `sumall1` with several numbers (remember from class that this sum is also just $n(n+1)/2$; you can use this fact to verify that your function is correct).

3. Now you will implement the same function using recursion. To do so, notice that `sumall`$(n) = n +$ `sumall`$(n-1)$ (at least for $n > 1$). Thus, `sumall` can be written as a function of itself with a slightly smaller input, a classic sign that we can use recursion. Use this fact to implement a **recursive** Matlab function `sumall2` that, just like `sumall1`, takes a positive integer n and computes `sumall`(n). The header of this function should be:

   ```
   function [ sum ] = sumall2(n)
   % Compute the sum of all numbers between 1 and n using recursion
   ```

   Be careful to write your code so that it terminates eventually! (Remember the `return` statement we talked about in class, and think about how you should use it here.)

2.2 Euclid's algorithm

Given two positive integers a and b where $a > b$, the *greatest common denominator* of a and b (called $\texttt{gcd}(a, b)$) is the largest number that evenly divides both a and b (for instance, $\texttt{gcd}(12, 8) = 4$, $\texttt{gcd}(10, 5) = 5$, and $\texttt{gcd}(17, 10) = 1$). One of the first algorithms ever invented—called Euclid's algorithm—computes the `gcd` of two numbers. (You might remember Euclid from high school geometry class—he wrote this algorithm down in 300 BC, although it was probably invented earlier. He also wrote down a lot of other things.)

The algorithm is very simple, and can be stated as follows: if $b = 0$, then return a—we're done. Otherwise, let $a' = b$, and let $b' = \texttt{mod}(a, b)$, and repeat the algorithm with a' and b' in place of a and b (the function $\texttt{mod}(a, b)$ returns the remainder of a/b, and is a built-in Matlab function; it also appears in assignment 1). Thus, a and b will be replaced with smaller numbers, and we run the algorithm again. (Eventually, b will be zero, and we'll return a).

4. Write a Matlab function `gcd1` that takes two numbers and computes their gcd using Euclid's algorithm. This function should use a **while loop**. The header of this function should be:

   ```
   function [ g ] = gcd1(a, b)
   % Compute the greatest common denominator of a and b using a while loop
   ```

5. The `gcd` function can also be written using recursion, since, when b is not equal to 0, $\texttt{gcd}(a, b) = \texttt{gcd}(b, \texttt{mod}(a, b))$ (again, we see that `gcd` can be written as a function as itself with slightly smaller inputs).

   Use this fact to implement a **recursive** Matlab function `gcd2` that computes the greatest common denominator of two numbers. The header of this function should be:

   ```
   function [ g ] = gcd1(a, b)
   % Compute the greatest common denominator of a and b using recursion
   ```

   Again, be careful to write your code so that it terminates.