CSCM12 – Software concepts and efficiency February 17th 2025
Manjiri Joshi & Cécilia Pradic

Lab 3: recursive functions

- The questions marked as **Challenge** are not required for signing off, but I would encourage you to look at them if and only if you have time to spare.

1. **Factorial, recursively** Write a java function taking as input an integer n and outputting the product of all integers from 1 to n using recursion.

2. **Playing with dice** We are looking at the following problem: If we roll k 6-sided dice, how many ways are there for the sum of the eyes to be n?

   (a) Calculate by hand how many ways there are to get a total of 7 by rolling 2 dice.

   (b) Describe a recursive process to obtain the general answer by distinguishing the potential outcomes for the first dice.

   (c) Implement the process in Java, and run it on a few examples.

3. **Fun with fractals** Fractals are geometric shapes that are often nice to describe using recursion. For this question, we will be drawing some fractals using the graphics library from java. First, download the file `Fractals.java` from canvas. It comes pre-filled with a bit of boilerplate code for displaying stuff and a `Turtle` class[1].

   The basic idea is that we may pretend that we are drawing thanks to a turtle walking on the screen. The turtle is always located somewhere, represented by the x and y attribute, and facing some direction represented by `orientation`, which stores the angle with the x-axis (in radians). The method `turnLeft` can be used to change orientation, and `walk` to make the turtle move forward in the current direction by a certain distance, printing a line in the process. The method `fly` is the same as `walk`, but it will not induce any drawing.

   In the boilerplate code, I expect you to modify the `paint` method of `MyCanvas` and to add methods to `Turtle`[2]. You should not need to modify attributes directly in those new methods.

   (a) First run the code and explain the `walkEquilateralTriangle` method.

   (b) Implement a `turnRight` method similar to `turnLeft`, but which makes the turtle turn to the right. Also implement a method `turnAround`.

   (c) Implement a method for `Turtle` with signature

   ```
   static void randomWalk(double dist, int nbSteps)
   ```

   which implements the following random walk: at each step, the turtle rotates randomly[3] in one of the four cardinal directions and move over `dist` pixels. Try out your function on reasonably large values. For `nbSteps = 50`, does you turtle ever loop?

[1] This is inspired from the Logo language `https://en.wikipedia.org/wiki/Logo_(programming_language)`.

[2] Unless there are some quick fixes to be made in the made function owing to screen size or something of that nature, but I expect things should work as-is.

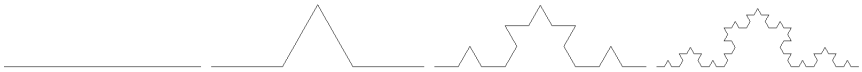
[3] If you do not remember how to draw a random number, please look at the provided code, it has some hints

(d) The *Koch* curve is perhaps the most popular example of a first fractal. It can be defined as the limit of the sequence of curves which can be defined recursively as follows:

- at order 0, just draw a segment
- at order $(n+1)$, consider the following shape

where the middle part is an equilateral triangle minus its base. Take that and replace every segment by the Koch curve at order n.

The first few iterations can be represented as follows:



This can easily be done using the turtle by rephrasing the above process recursively

i. First, write a method

```
static void walkKoch1(double dist)
```

that prints out the curve at order 1; each atomic segment should have size $\frac{\texttt{dist}}{3}$, so that the overall straight line distance covered by the turtle should be `dist` (hint: recall that all the inner angles of an equilateral triangle are $\frac{\pi}{3}$

ii. Now write the general function

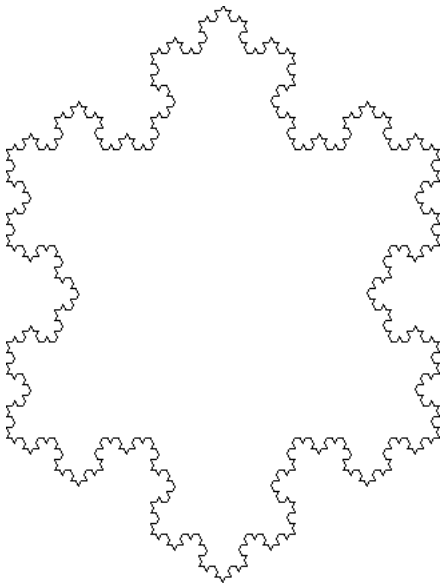
```
static void walkKoch(double dist, order n)
```

that prints out the Koch curve at an arbitrary order. For the recursive case, I suggest to start from the code from `walkKoch1`, which can presumably be adapted by replacing calls of the `walk` method by suitable recursive calls.

iii. Finally write a function

```
static void walkKochFlake(double dist, order n)
```

that outputs the following



(e) **Challenge** draw some other fractals; one nice one would be the dragon curve which may be obtained using a mutual recursion:

- At order 0, just draw a segment.
- At order $n+1$, draw a dragon curve at order n, turn $\frac{\pi}{2}$ to the left, and draw a *flipped* dragon curve at order n.

The flipped dragon curve of order n is drawn by:

- Drawing a segment at order 0
- At order $n+1$, first drawing a non-flipped dragon curve of order n, turning $\frac{\pi}{2}$ **to the right** and then drawing a flipped dragon curve of order n.

You may find inspiration from wikipedia, or you can have fun by varying the angles, the distance and adding a modicum of randomness! Feel free to look at the documentation of `Graphics` to do fancier stuff too.