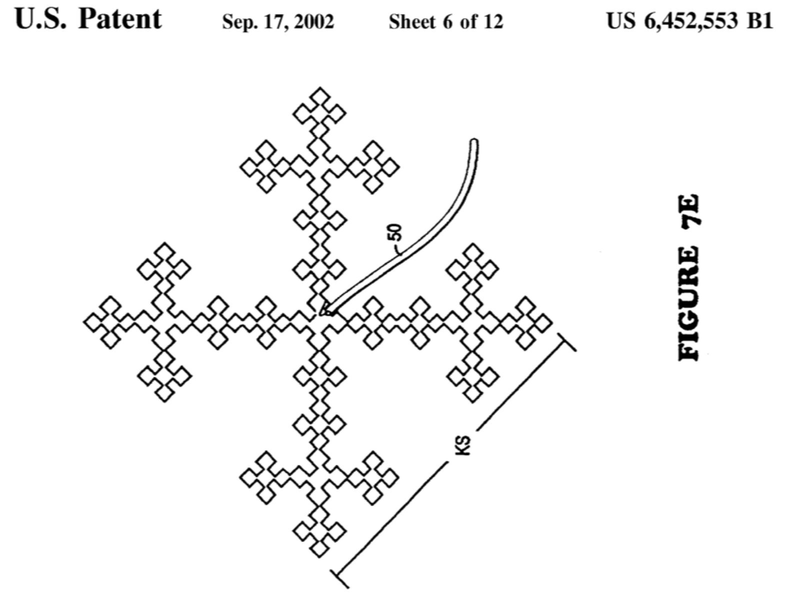
Computer Science AP/X CSCI-140/242
Fractal Antenna Lab 2

1 Problem

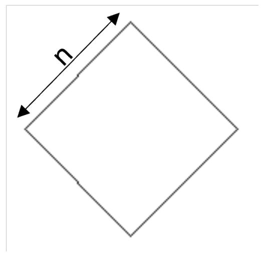
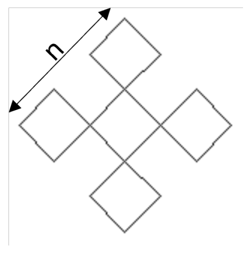
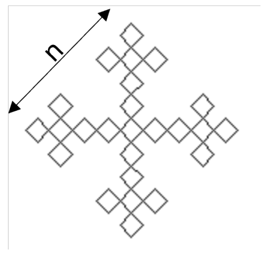
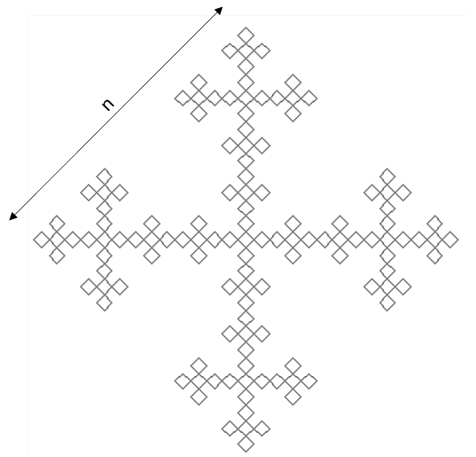
According to Wikipedia, a fractal antenna "is an antenna that uses a fractal, self-similar design to maximize the length, or increase the perimeter" and "are very compact, multi-band or wideband, and have useful applications in cellular telephone and microwave communications." One example of a fractal antenna is given as shown below:



In this lab, you will use Python's turtle package to draw fractal antennas.

2 Requirements

Table 1: Example Fractal Antenna Figures for `level` values 1-4

You will individually implement a program, `antenna.py`, which draws fractal antennas in a turtle's canvas using recursion. This file should contain the following:

1. A recursive function (strategy 1) that generates the antenna by drawing its perimeter as a single line. Note that this fractal antenna is similar to a square rotated 45 degrees, where each side of the square matches the pattern described in the problem-solving activity.
   To draw the fractal antenna, each side of length n, v levels, position the turtle in one of the antenna's corner (e.g. the bottom corner) and draw four "sides" of length n and level v.

2. A second recursive function (strategy 2) that draws the antenna as a series of squares, drawing one square at a time. This strategy works as follows:
   - If the level is 1, it draws a single square of length n like the top-leftmost figure from Table 1.
   - If the level is 2, it draws a square of length $n/3$ at the center, and then, four outer squares of the same length like the top-center figure from Table 1, and so on.

   At each recursive call, the length of the squares decreases by a third.

3. A `main` function that does the following:

```
Prompt the user for the length of one side in pixel (technically,
its level-1 equivalent length) (positive float). If the length is not a
valid float value, produce an error message and ask again

Prompt the user for number of levels (positive integer). If the
level is not a valid integer value, produce an error message and ask again

Set up the turtle window

Call the recursive function strategy 1 and print the antenna's total length

Pause the window and wait for the user to hit the enter key

Reset the turtle window

Call the recursive function strategy 2 and print the antenna's total length
```

4. Any other support/utility functions you write that promote function re-use (e.g. setting up the drawing window, drawing a single line segment of the antenna or drawing a square).

Click here to watch a video demonstrating the program.

2.1 Input Validation

All user input must be error-checked, with error messages of this form displayed when needed.

`Value must be a` *type*. `You entered` '*string-value*'.

The user is given an unlimited number of chances to enter a legal value. The program only need check for correct type, not correct range of value, e.g. non-negative.

To do the error-checking, look at the `fullmatch` function in the `re` package. Also study the form of *regular expression patterns*. For example, `r"[0-9]+"` is a pattern matched by all strings only containing decimal digits. (As an alternative, and if you have experience with exceptions, you can handle the exception thrown when numeric conversion fails.)

2.2 Contraints

- Both strategy functions must draw a fractal antenna with the given length and level, and return the total length of all the lines drawn. The resulting drawing and total length must be the same for both strategies.

- When your program draws the figure, it may be off center. This is valid.
- You are not allowed to use global variables to compute the antenna's total length.

2.3 Useful tips

- Use the `input` built-in function to pause the program between displaying the antenna using strategy 1 and 2.
- A variety of capabilities from the turtle library will help produce the drawing.
  - `speed()`: Set the turtle's speed to an integer values between 0 and 10. 0 is the fastest speed, 1 the slowest and 6 is normal speed.
  - `hideturtle()`: Hide the turtle so we can see the drawing. You should hide the turtle before starting to draw.
  - `reset()`: Reset the window by clearing everything and setting the default state. Use this in between the two drawings.
  - Turtle can very slow to animate, but it is useful when you are developing and debugging. To turn off the animation, use the function `tracer()` before starting to draw. After drawing, use the function `update()` to refresh the turtle's window and display the drawing.

    ```
    turtle.tracer(0, 0)
    # call the drawing recursive function here
    turtle.update()
    ```

  - `mainloop()`: Wait for the user to close the turtle window.

  Note: You are not allowed to use commands like `goto`, `setPos`, or similar. You must move the turtle by using the commands `left`, `right`, `forward`, and `backward`.

2.4 Sample Run

What follows is a sample run of your program for a specific set of parameter values. If you actually run your completed program, the following text should appear in the console. The only user-entered information is after the colons. The turtle will draw the figure shown in Table 1 (level 2) in a separate canvas window.

```
Length of initial side: 300
Number of levels: 2
Strategy 1 - Antenna's length is 2000.0 units.
Hit enter to continue...
Strategy 2 - Antenna's length is 2000.0 units.
Bye!
```

3 Grading

The assignment grade is based on these factors:

- 15%: Attendance at problem-solving and results of problem-solving teamwork
- 10%: In-lab preliminary assignment
- 10%: Good design practices
- 60% Functionality:
  - 5%: Proper user input (including order)
  - 5%: Error checking and re-prompting
  - 20%: Accurate drawing of the antenna using strategy 1 (15%) and compute its length (5%)
  - 20%: Accurate drawing of the antenna using strategy 2 (15%) and compute its length (5%)
  - 5%: The program pauses between drawing the two antennas
  - 5%: The program resets the turtle window when drawing the second antenna
- 5%: The code follows the style guidelines on the course website

4 Submission

Go to your project and zip your **src** directory, which should contain just your Python source file(s), to a file named **lab02.zip** and submit it to the MyCourses Assignments dropbox before the due date.