# Recursion and Grammars for CS2

Viera K. Proulx
Northeastern University
College of Computer Science
Boston, MA 02115, USA

vkp@ccs.neu.edu

## Abstract

A programming exercise on recursion in which students create drawings of simple and bracketed Lindenmayer Systems provides a context for exploring additional computer science concepts. The resulting drawings give students a better understanding of the power of recursion as well as the rate of growth of time complexity with multiple successive recursive calls. We describe the exercise, the concepts that need to be addressed to solve the problems, and the results of using this exercise in our classes.

## Introduction

A programming exercise on recursion in which students draw Koch snowflake has become a staple of many textbooks. The drawing is based on replacing each line segment according to some rules and repeating the action recursively for several layers. Unfortunately, in the standard treatment, the replacement rules have to be hard-coded. That means that the function to draw a side of the snowflake has a one time utility - it only makes snowflakes. If we decide to do a different fractal curve, we have to rewrite the whole function.

Fractal curves such as Koch snowflakes may also be described by simple grammars (or rewriting systems) known as Lindenmayer systems [1]. The simple Lindenmayer systems are defined by a grammar describing the rewriting rules for different symbols. User decides how many times should the rewriting rules be applied which is equivalent to selecting the depth of recursion. At level 0 the grammar symbols represent one of the following three actions that generate the drawing:

- move forward a fixed distance while drawing a line all capital letters
- move forward a fixed distance while lifting the pen symbol f
- turn (left or right) by a fixed angle symbols + and –

Partial support for this work has been provided by the NSF Leadership in Laboratory Development, award #DUE- 9650552

Capital letter S represents the starting string. At level zero, the symbols in the rewriting rule for S generate the level 0 curve. The angle and the distance are fixed before the drawing begins and remains constant throughout the drawing process.

For example, the Koch snowflake is described by the following grammar:

angle = 60°
S -> F—F—F
    the starting string
F -> F+F—F+F
    rewriting rule for symbol F at recursion levels > 0

The snowflakes at the first three levels would then be represented by strings:

```
level 0: F       ——F       ——F
level 1: F+F——F+F——F+F——F+F——F+F——F+F
```

Spacing was designed to show the application of the rewriting rule. At the next level we notice the exponential explosion:

```
level 2:
      F+F——F+F+F+F——F+F——F+F——F+F+F+F——F
+F——F+F——F+F+F+F——F+F——F+F——F+F+F+F——F+F
——F+F——F+F+F+F——F+F——F+F——F+F+F+F——F+F
```

A bracketed Lindenmayer system adds two new symbols to the grammar, '[' and ']'. The left bracket '[' indicates that the state of the Turtle (current position and heading) should be saved, before proceeding with the next symbol. The right bracket ']' indicates that the Turtle state should be restored to the one saved with the matching left bracket.

## The Exercise

We designed a programming exercise, in which the students start by reading in the rules of the grammar. We allow only the following rewriting symbols: S for the starting string, and F, L and R and f. At level 0, the symbol f represents a move without drawing, while symbols L, R, and F represent one line segment. The recursive function that creates the drawing has as its arguments the string to traverse, the Turtle object, and the current level of recursion. It traverses the string, examining each symbol, and depending on the level of recursion either performing the desired action or calling itself recursively with the replacement string and a decreased level of recursion. The drawing is initiated by calling this function with the starting string as the argument.

For example, to draw the Koch snowflake at level zero, we supply as argument the string F—F—F and the drawing becomes a triangle. At level one, we again start with traversing the string F—F—F but each time we see the symbol F we call the function recursively with level 0 and string F+F—F+F, thus describing how each side of the triangle needs to be decomposed into four segments.

The recursive pattern used here is more complex than the typical binary tree traversal or divide and conquer technique used in quicksort. The degree of branching is much larger and provides a more powerful representation of the exponential growth of the number of steps. However, because the result is visual, most of the mistakes students make are quite obvious and easy to identify.

In order to be able to see the drawing, students and the instructor have to resolve several issues related to the display of the drawing as well as to the internal recording of the state of the system. The key concepts are the following:

- creation of the class Turtle that with member functions Move, Skip, TurnLeft, TurnRight
- resolving the problem of scaling the image to fit the display window
- saving the state of a process before a function call and restoring it on return

The class Turtle is modeled after the well-known Logo Turtle. Internally, for our purpose, it records its heading, the size of the step, and the angle of the turn. It is a nice example to illustrate the design and use of classes in a non-trivial context.

The scaling issue is more complex and should be examined in several laboratories leading up to this exercise. Scaling is at the heart of any graphics display exercise and should be one of the core concepts students become familiar with during their first two years of CS study.

In our solution to this problem, the recursive drawing is 'created' twice. The first time we record the bounds of the region traversed by the Turtle. We use the result to determine the scaling factors, which are then used when the drawing is actually made in the second call to the drawing function.

To implement the bracketed L-systems the left bracket symbol '[' activates saving of the current Turtle object and passing a new copy to the recursive function call. The right bracket ']' returns to using the saved Turtle object, which includes moving the current 'pen location' to the location recorded in the saved Turtle object. The design of this save/restore operation allows us to highlight the issues that need to be addressed during any save/restore operation in a stack-like context: creation of a copy of the object being saved and making sure that the all relevant information including the graphics state is restored correctly.

## The User Interface

We believe that to get the greatest rewards from running this exercise, the user interface should include a number of options. First of all, the user should be allowed to use the same grammar several times with different levels of recursion to see the growth of complexity.

Next, the user should have the option to see the string generated by the grammar. This is essential during the debugging stage to locate errors that cannot be detected by viewing the drawing. For example, if the angle is erroneously set to 90° when drawing Koch snowflake, user sees only a straight line at level 0. By viewing the generated string, the error is easily detected. Observing the generated strings is also helpful in illustrating the full impact of the recursion (see below).

Finally, user should be also allowed to run the program with several different grammars in a row - to make comparisons as well as experiment with minor modifications of grammars.

## The Rewards

The exercise is very exciting. The recursion used to implement the drawing is a bit more complex than most of the straightforward exercises seen in standard textbooks. Students see, that recursion can be a very powerful tool when used in a non-trivial way.

Students also learn something about drawing complex images. It is crucial for the success of this exercise that the Turtle record the exact position in real coordinates and that the scaling is applied only at the time when a line is drawn or a move or turn is made. Otherwise, the round-off error quickly accumulates and destroys the symmetry of the drawing.

When students run their program with different levels of recursion, they observe the growth in the number of steps needed. The rate of growth with the increasing the levels of recursion becomes very concrete. Students can also print out the strings generated by the grammar and count the number of moves the turtle makes at each of the successive levels.

Another lesson is in seeing a large number of different objects generated from the same underlying formal system. Students seldom experience situations that illustrate the power of describing concrete objects by a formal set of rules. Here, they may begin to understand, how the definitions of other formal systems help understand and manage these complex situations.

## References

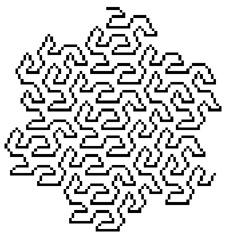
1. Przemyslaw Prusinkiewicz and Aristid Lindenmayer, *The Algorithmic Beauty of Plants*, Springer Verlag, New York, 1990.

2. Jack Ryder, "Using Parallel String Rewriting Systems to Generate Fractal Images", Proceedings of the Twelfth Annual Eastern Small College Computing Conference, October 25-26, 1996, pp. 76-78.

## Sample Drawings

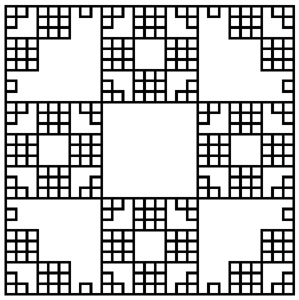
### *Hexagonal Gosper Curve:*

angle = 60°
S -> L
L -> L+R++R-L--LL-R+
R -> -L+RR++R+L--L-R



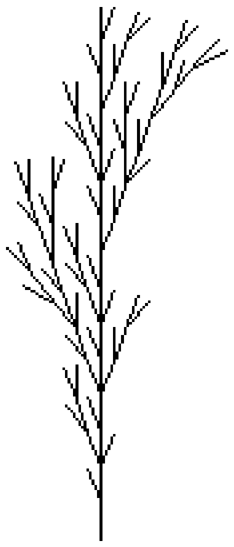
### *Quadratic Koch Island*

angle = 90°
S -> F-F-F-F
F -> F+FF-FF-F-F+F+FF-F-F+F+FF+FF-F



### *Bracketed OL-system*

angle = 20°
S -> F
F -> F[+F]F[-F][F]
N = 4



angle = 20°
S -> F
F -> F[+F]F[-F][F]
N = 6