

CS 106B

Lecture 8: Fractals

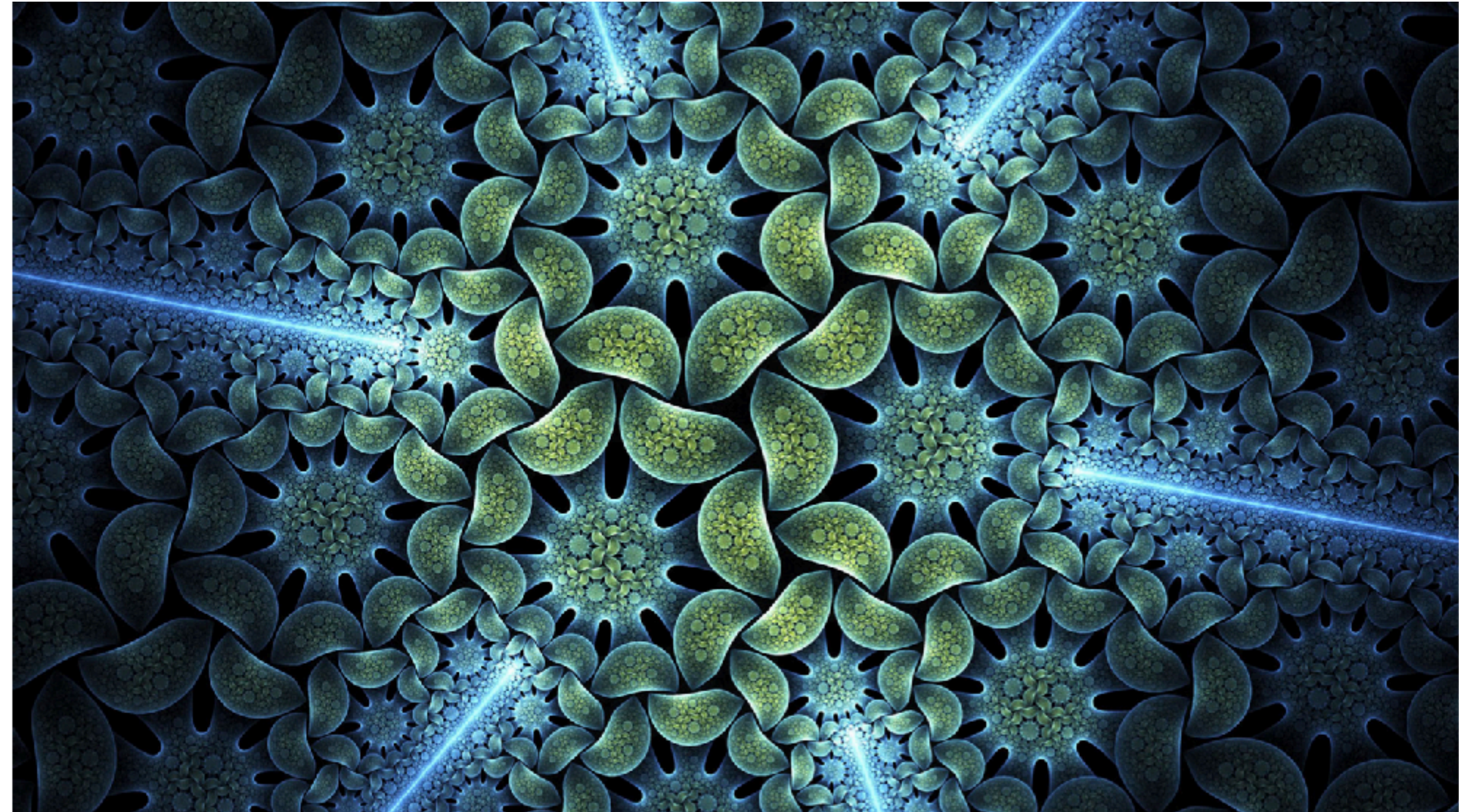
Monday, July 10, 2017

Programming Abstractions
Summer 2017
Stanford University
Computer Science Department

Lecturer: Chris Gregg

reading:

Programming Abstractions in C++, Chapter 5.4-5.6



Today's Topics

- Logistics:
 - ADTs Due Wednesday July 12th, noon
 - Towers of Hanoi video featuring Keith Schwartz: <https://www.youtube.com/watch?v=2SUvWfNJSsM>
 - Handout for today: <http://web.stanford.edu/class/cs106b/lectures/8-Fractals/code/handout.pdf>
- Assignment 3: Recursion — will be posted tomorrow, due *next Tuesday*.
 - Fractals
 - Grammar Solver
 - A more detailed recursion example
 - Fractals



Three Musts of Recursion 🔑

1. Your code must have a case for all valid inputs


2. You must have a base case that makes no recursive calls

3. When you make a recursive call it should be to a simpler instance and make forward progress towards the base case.



Recursion Example

The Google logo is displayed in its standard multi-colored font: 'G' is blue, the first 'o' is red, the second 'o' is yellow, 'g' is blue, 'l' is green, and 'e' is red.

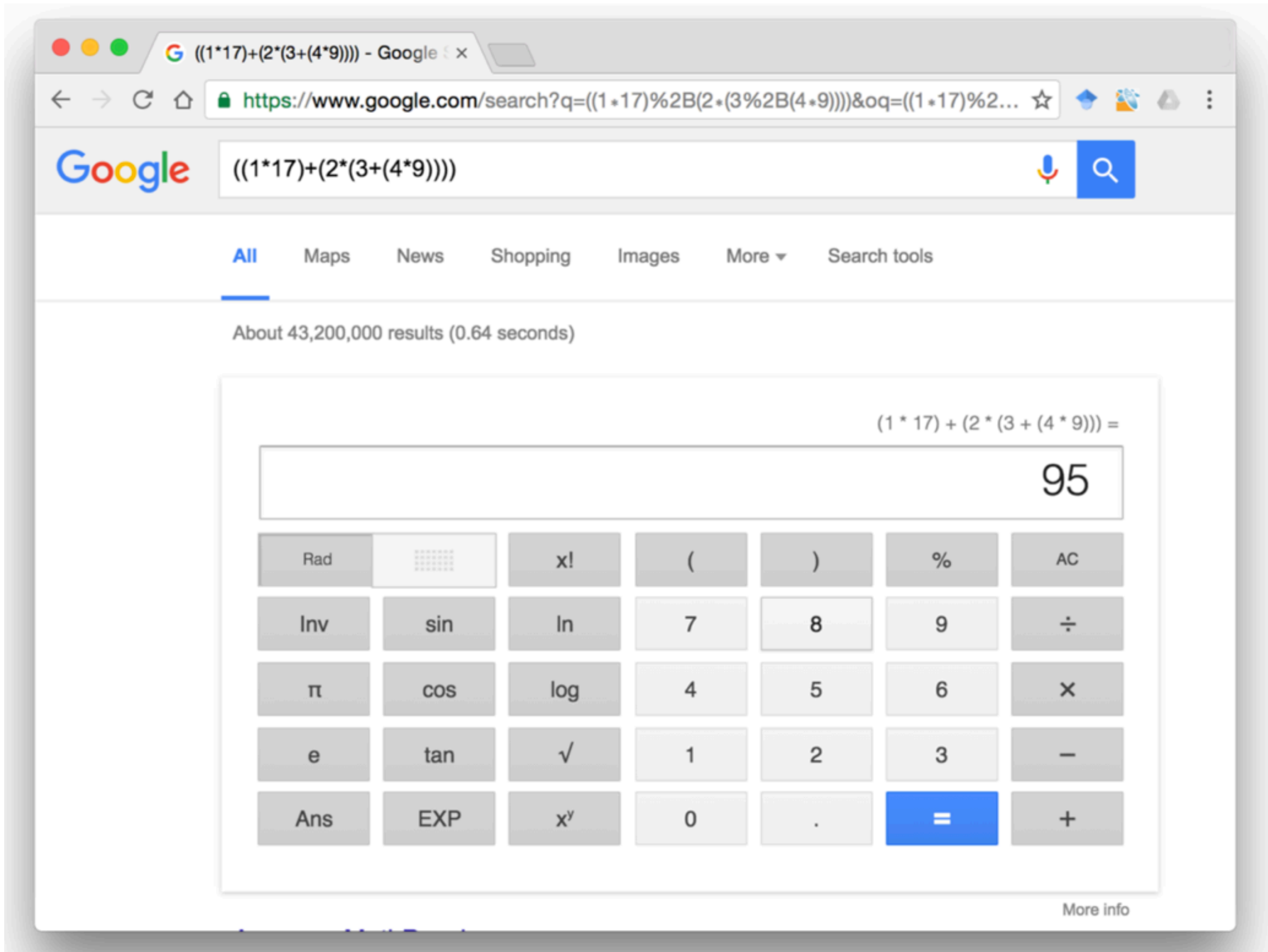
$((1+3) * (2 * (4+1)))$ 

Google Search

I'm Feeling Lucky



Recursion Example



$$((1 * 17) + (2 * (3 + (4 * 9))))$$

↓

$$95$$



Challenge

Implement a function which evaluates an expression string:

"((1+3)*(2*(4+1)))"

"(7+6)"

"(((4*(1+2))+6)*7)"

(only needs to implement * or +)



Anatomy of an Expression

An expression is always one of these three things

number

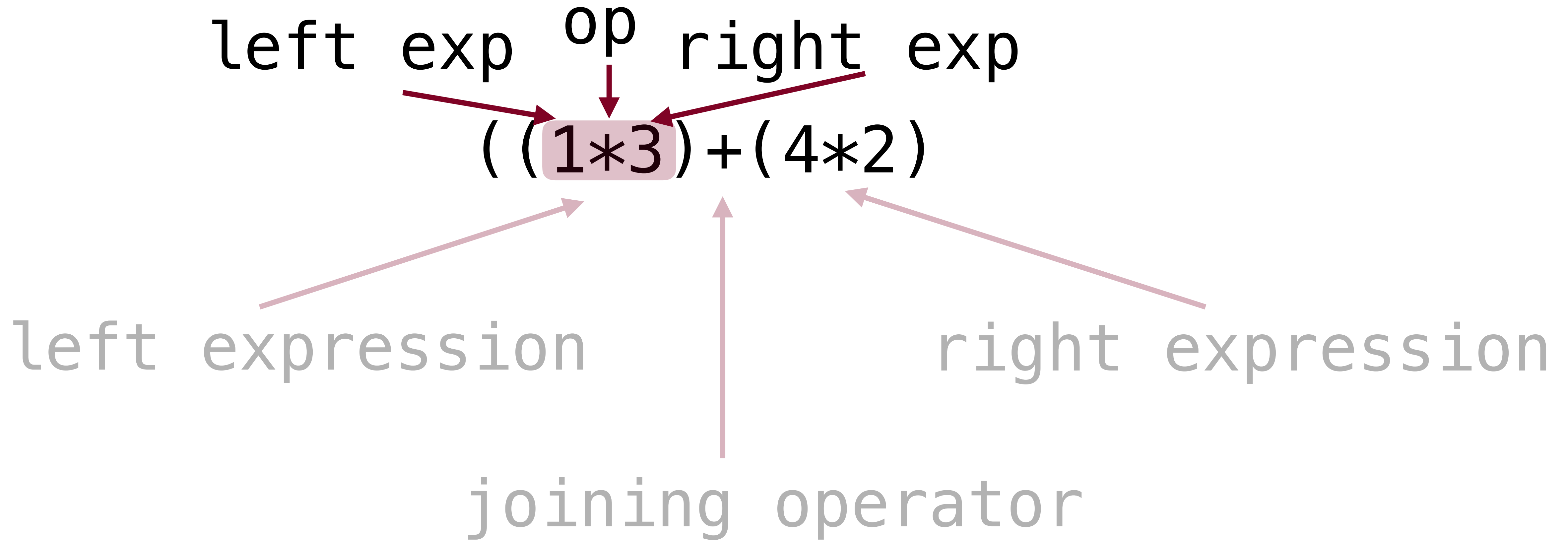
expression

(expression + expression)

(expression * expression)

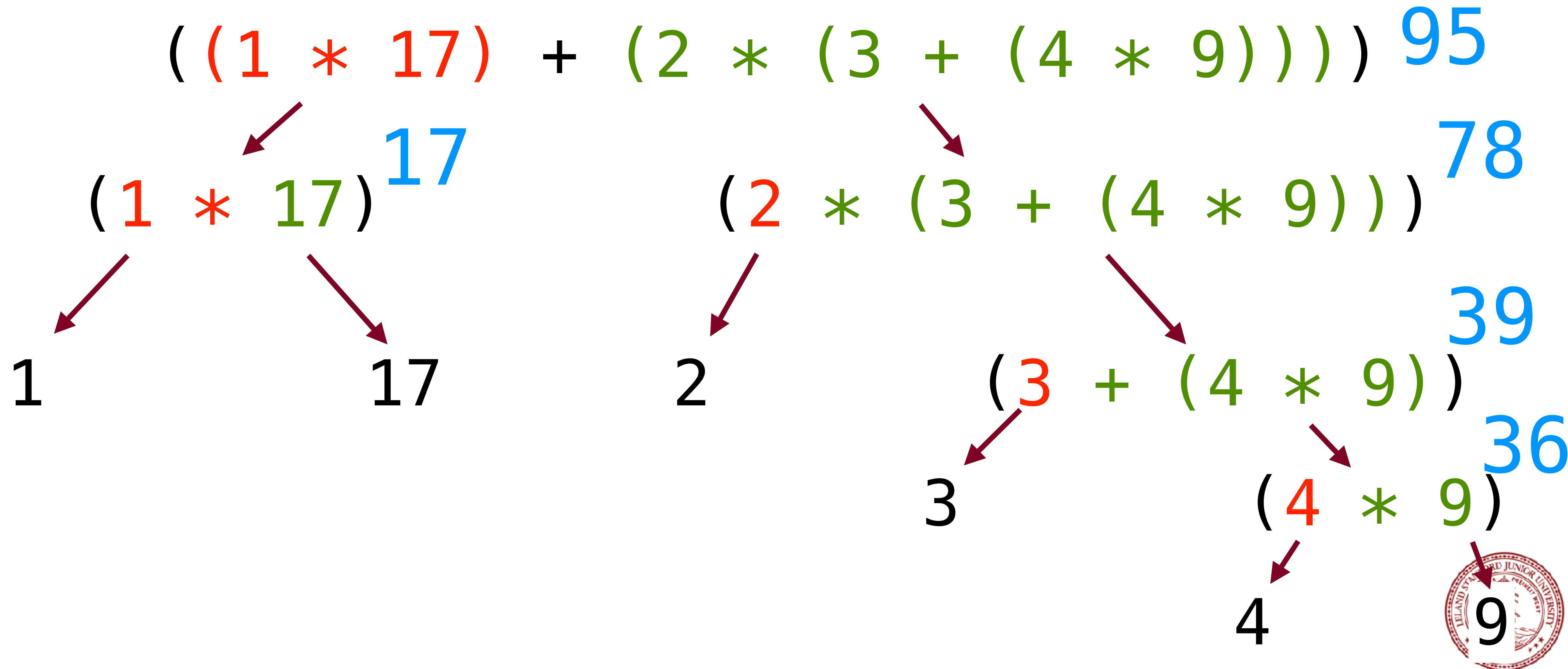


Anatomy of an Expression



Anatomy of an Expression

How do we evaluate $((1 * 17) + (2 * (3 + (4 * 9))))$?



Is it Recursive? Yes!

$$((1*3) + (4+2))$$

The big instance of this problem is:

$$((1*3) + (4+2))$$

The smaller instances are:

$$(1*3) \text{ and } (4+2)$$



Task

Write this function: `int evaluate(string exp);`

```
"((1*3)+(4+2))" // returns 9
```

Using these library functions:

```
stringIsInteger(exp)  
stringToInteger(exp)
```

And these exp helper functions:

```
//returns '+'  
char op = getOperator(exp);  
//returns "(1*3)"  
string left = getLeftExp(exp);  
//returns "(4+2)"  
string right = getRightExp(exp);
```



Solution (Pseudocode)

"((1*3)+(4+2))"

int evaluate(expression):

- if *expression* is a number, return *expression*
- Otherwise, break up *expression* by its operator:
 - *leftResult* = evaluate(leftExpression)
 - *rightResult* = evaluate(rightExpression)
 - return *leftResult* operator *rightResult*



Solution

```
int evaluate(string exp) {  
    if (stringIsInteger(exp)) {  
        return stringToInteger(exp);  
    } else {  
        char op = getOperator(exp);  
        string left = getLeftExp(exp);  
        string right = getRightExp(exp);  
        int leftResult = evaluate(left);  
        int rightResult = evaluate(right);  
        if (op == '+') {  
            return leftResult + rightResult;  
        } else if (op == '*') {  
            return leftResult * rightResult;  
        }  
    }  
}
```

`exp = "((1*3)+(4*5)+2)"`

`op = '+'`

`left = "(1*3)"`

`right = "((4*5)+2)"`

`leftResult = 3`

`rightResult = 22`



Helper Methods

Here is the key function behind the helper methods:

```
int getOppIndex(string exp){
    int parens = 0;
    // ignore first left paren
    for (int i = 1; i < exp.length(); i++) {
        char c = exp[i];
        if (c == '(') {
            parens++;
        } else if (c == ')') {
            parens--;
        }
        if (parens == 0 && (c == '+' || c == '*')) {
            return i;
        }
    }
}
```



By the way...

We could also have solved this with a stack!



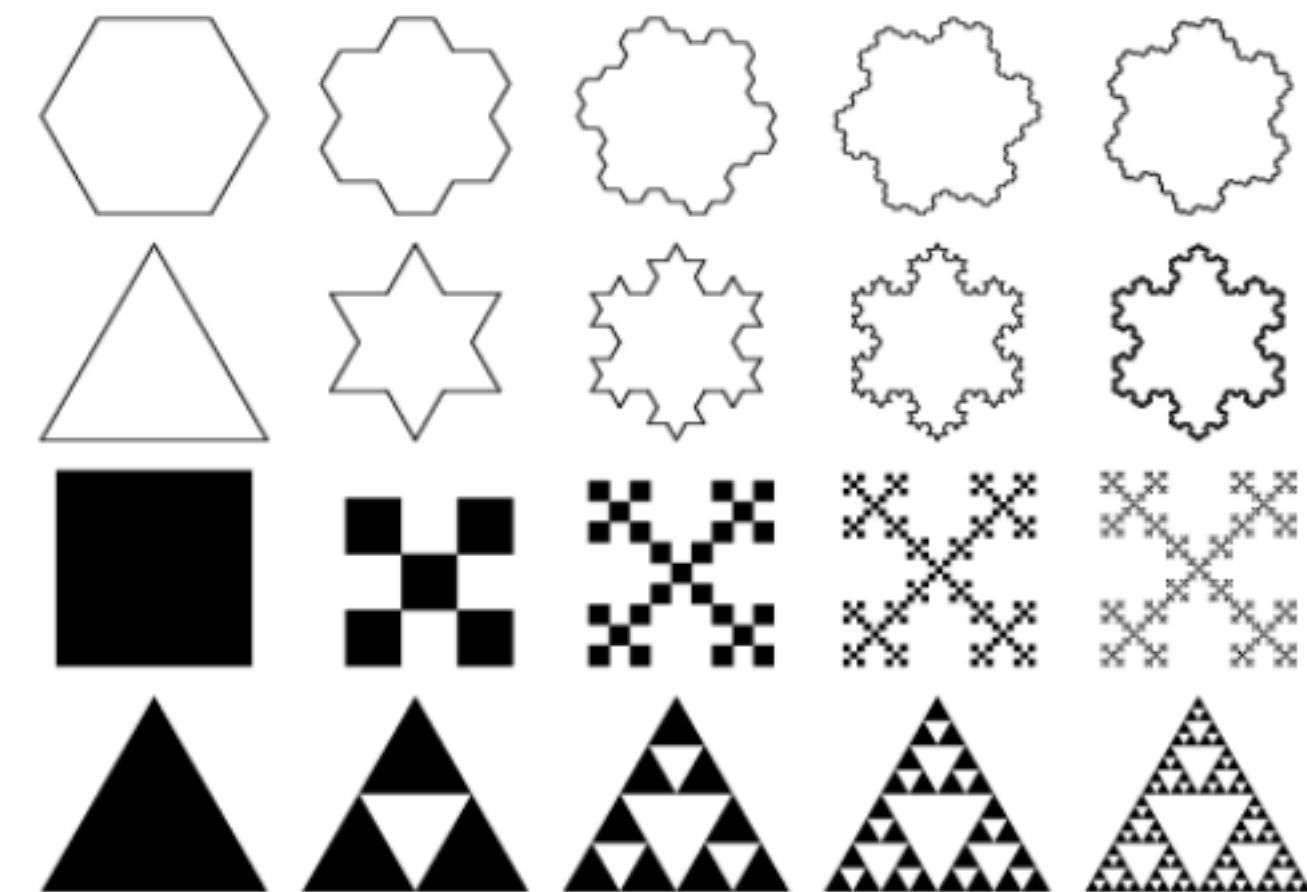
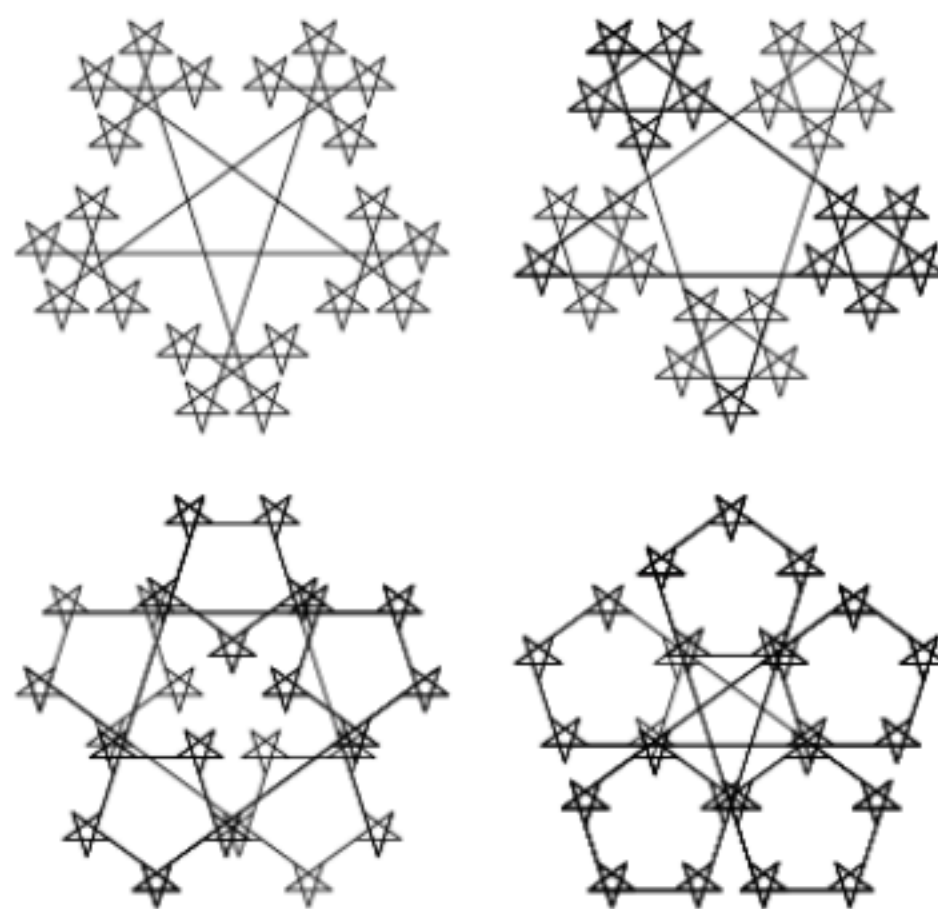
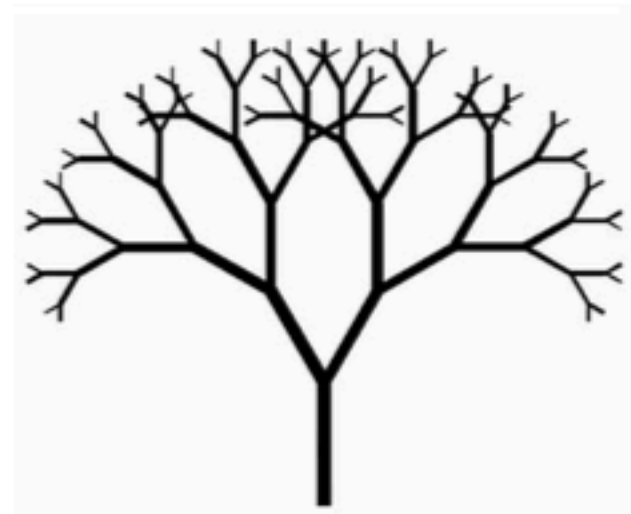
Today

Recursion you can see



Fractal

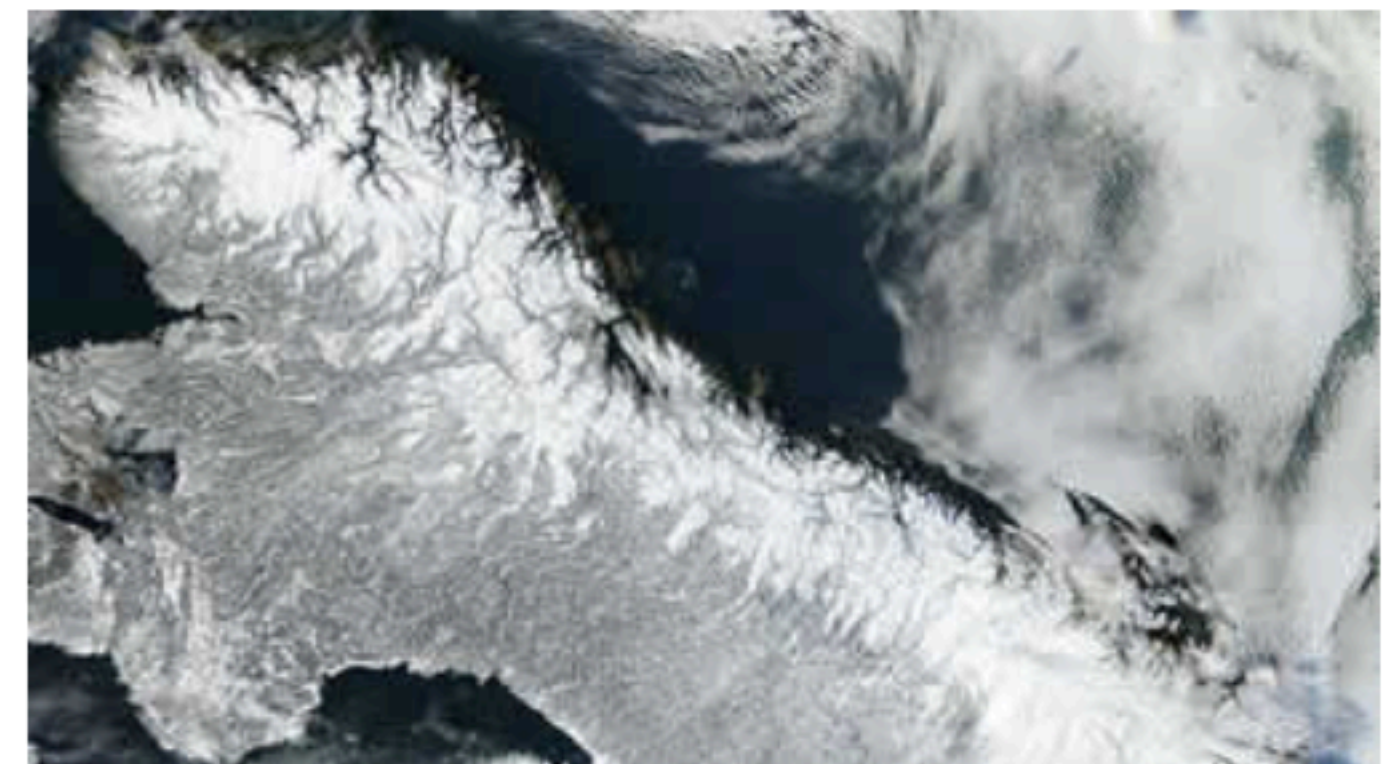
fractal: A recurring graphical pattern. Smaller instances of the same shape or pattern occur within the pattern itself.



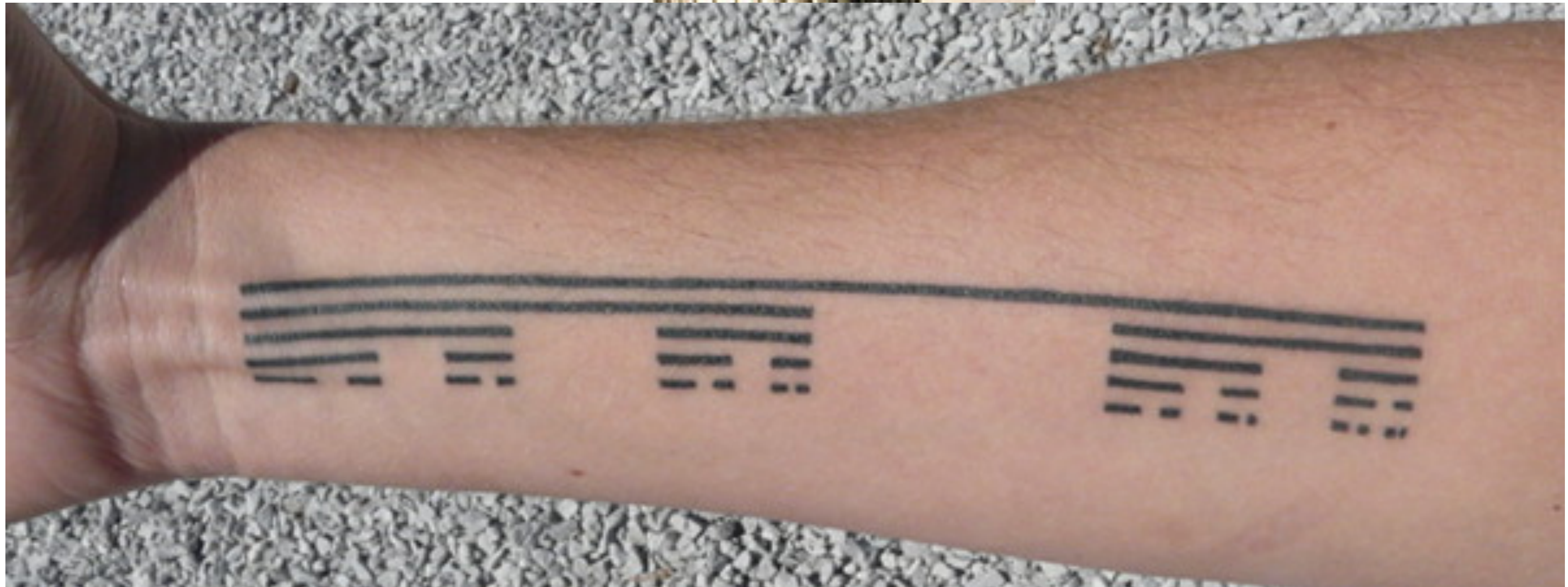
Fractal

Many natural phenomena generate fractal patterns:

1. earthquake fault lines
2. animal color patterns
3. clouds
4. mountain ranges
5. snowflakes
6. crystals
7. DNA
8. ...



The Cantor Fractal



Cantor Fractal



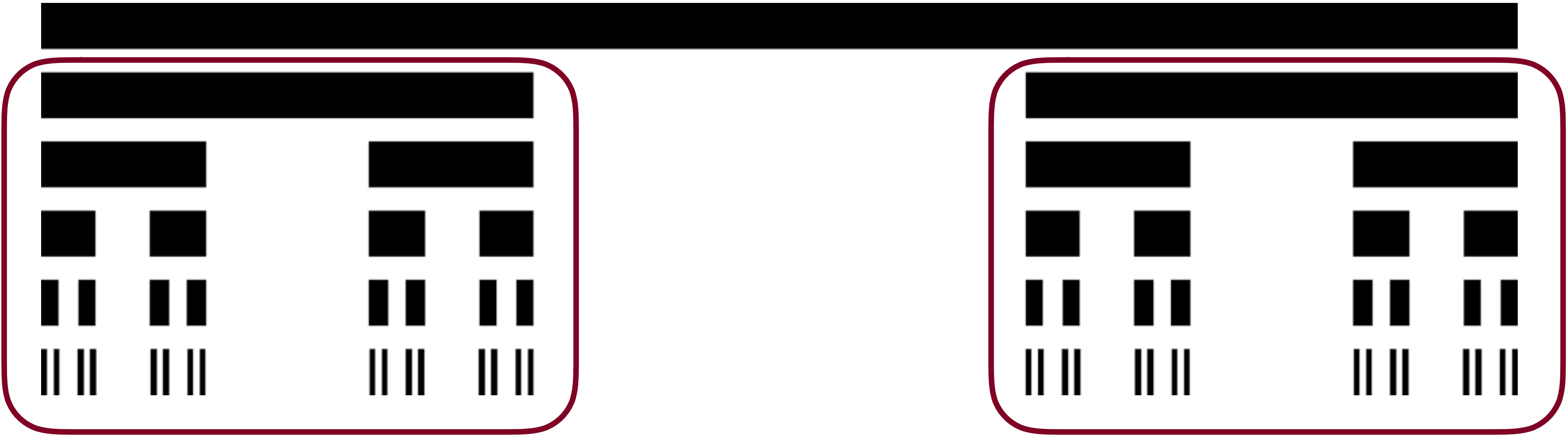
Parts of a cantor set image ... are Cantor set images



Cantor Fractal

Start

End



Another cantor set

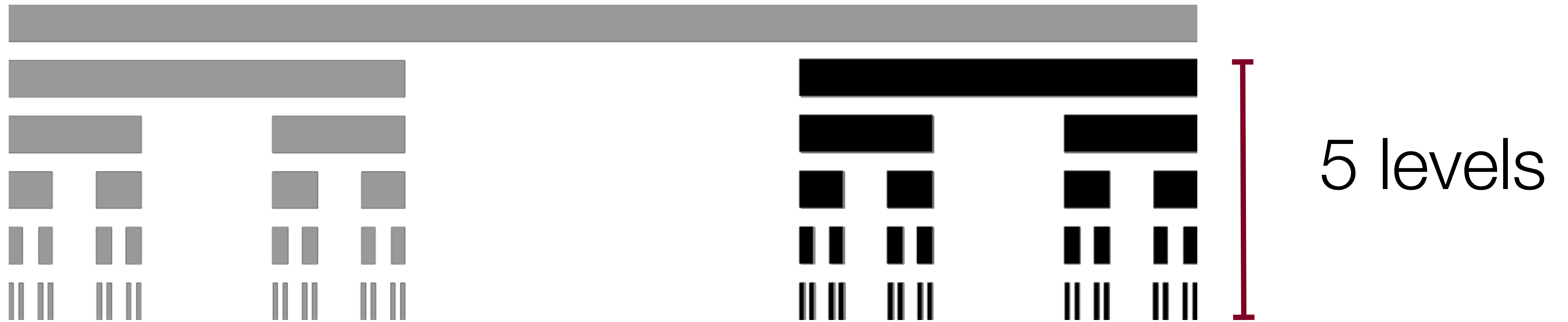
Also a cantor set



Levels of Cantor



Levels of Cantor



Levels of Cantor



1 level

I

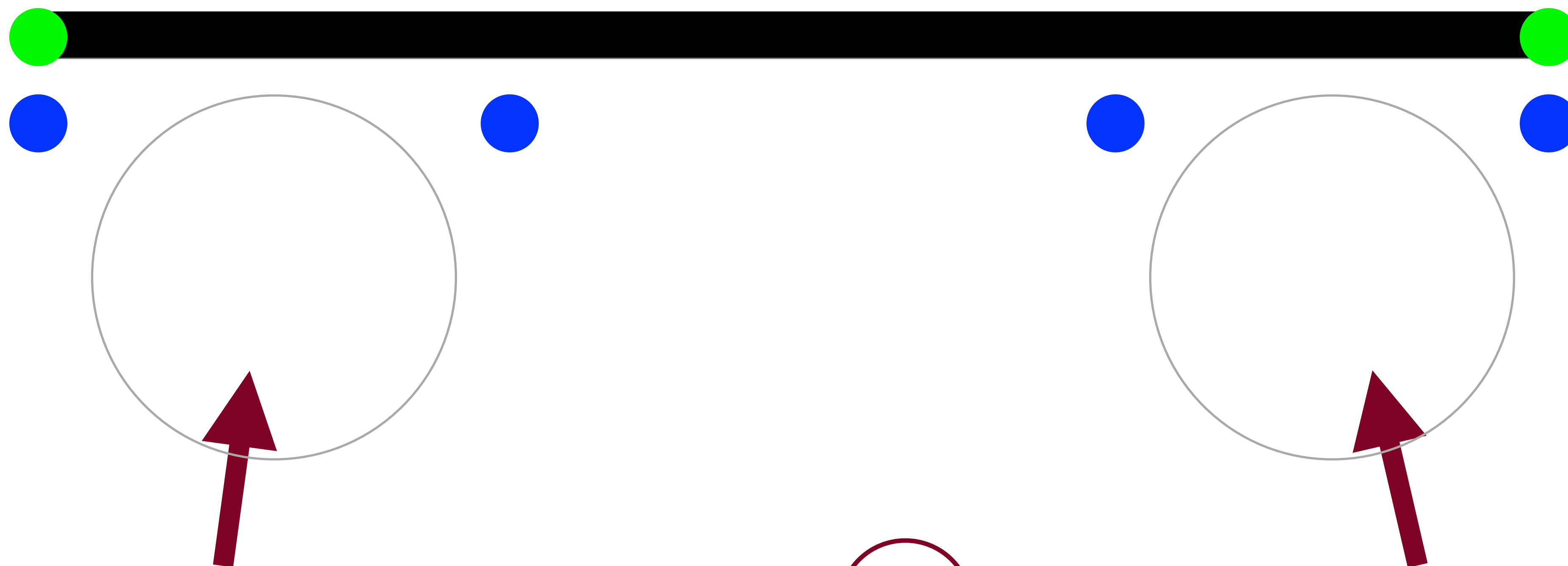


How to Draw a Level 1 Cantor



How to Draw a Level n Cantor

1 Draw a line from start to finish.

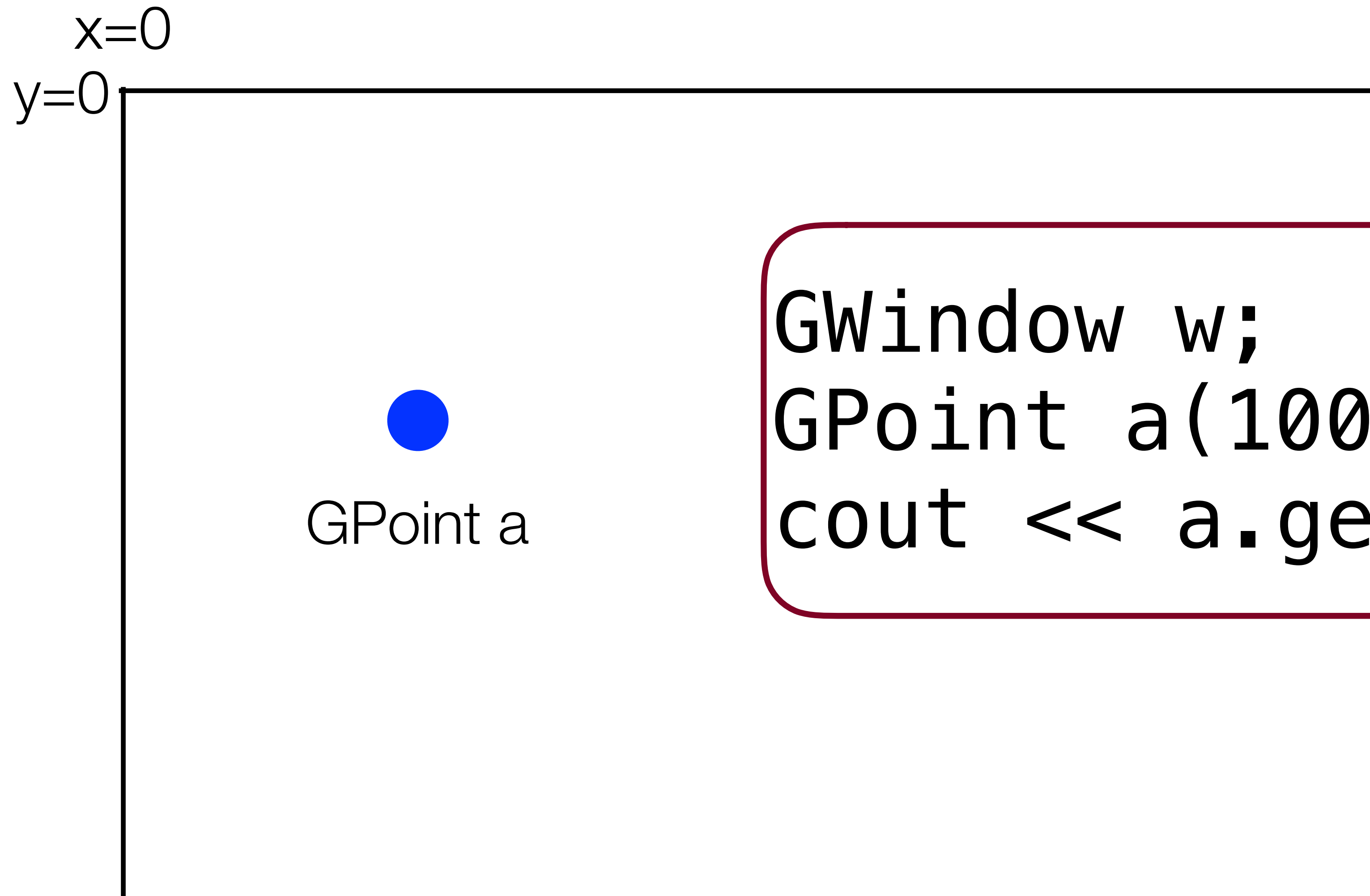


2 Draw a Cantor of size $n-1$

2 Draw a Cantor of size $n-1$



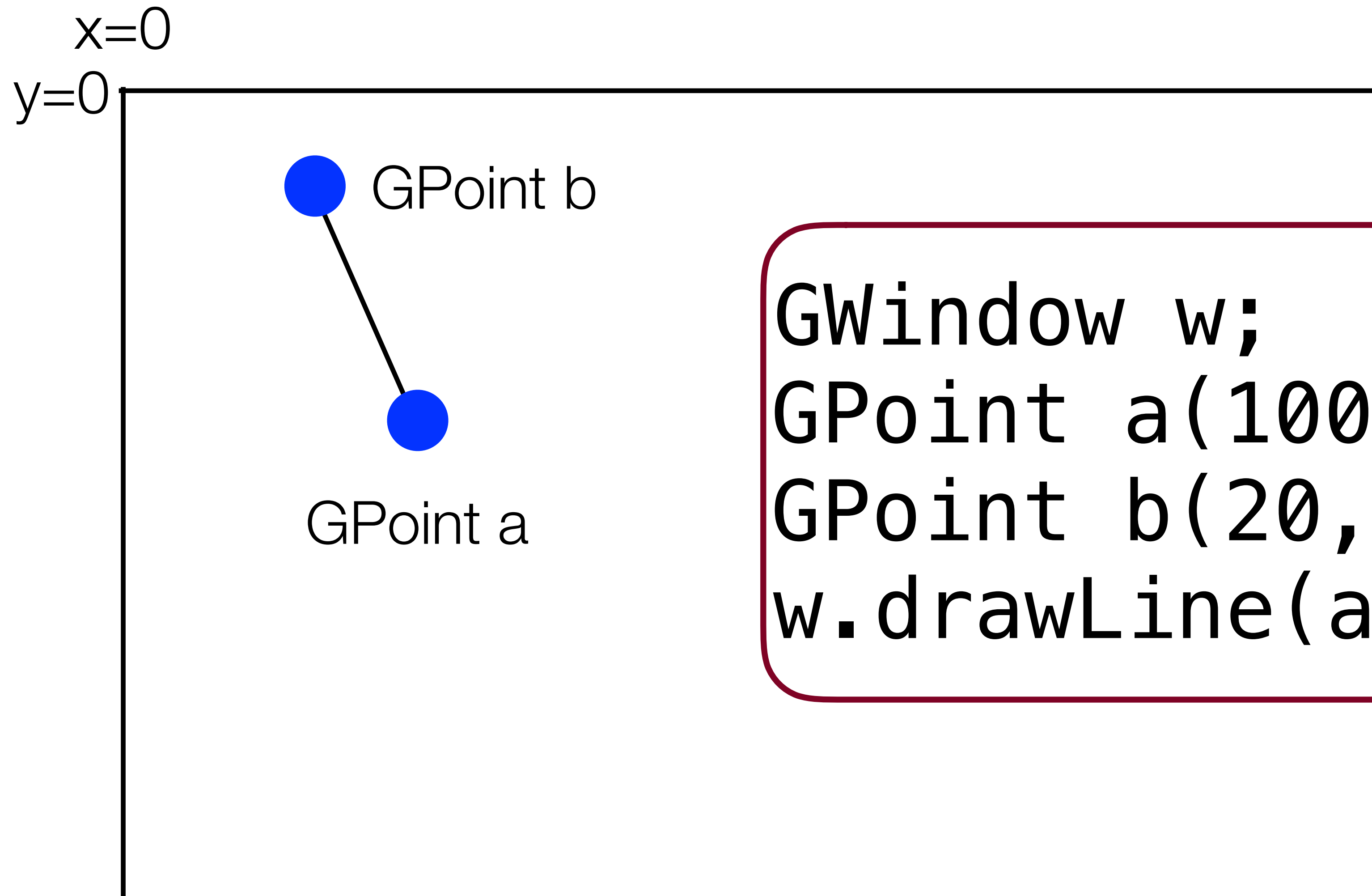
Graphics in C++ with the Stanford Libs: GPoint



```
GWindow w;  
GPoint a(100, 100);  
cout << a.getX() << endl;
```



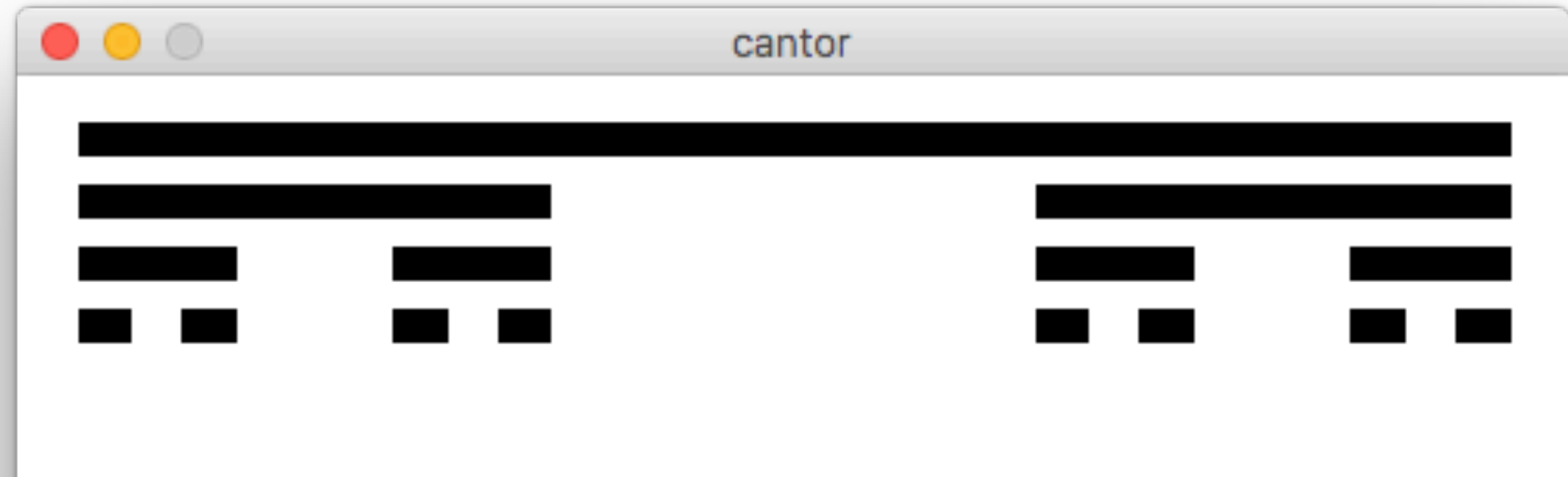
Graphics in C++ with the Stanford Libs: GPoint



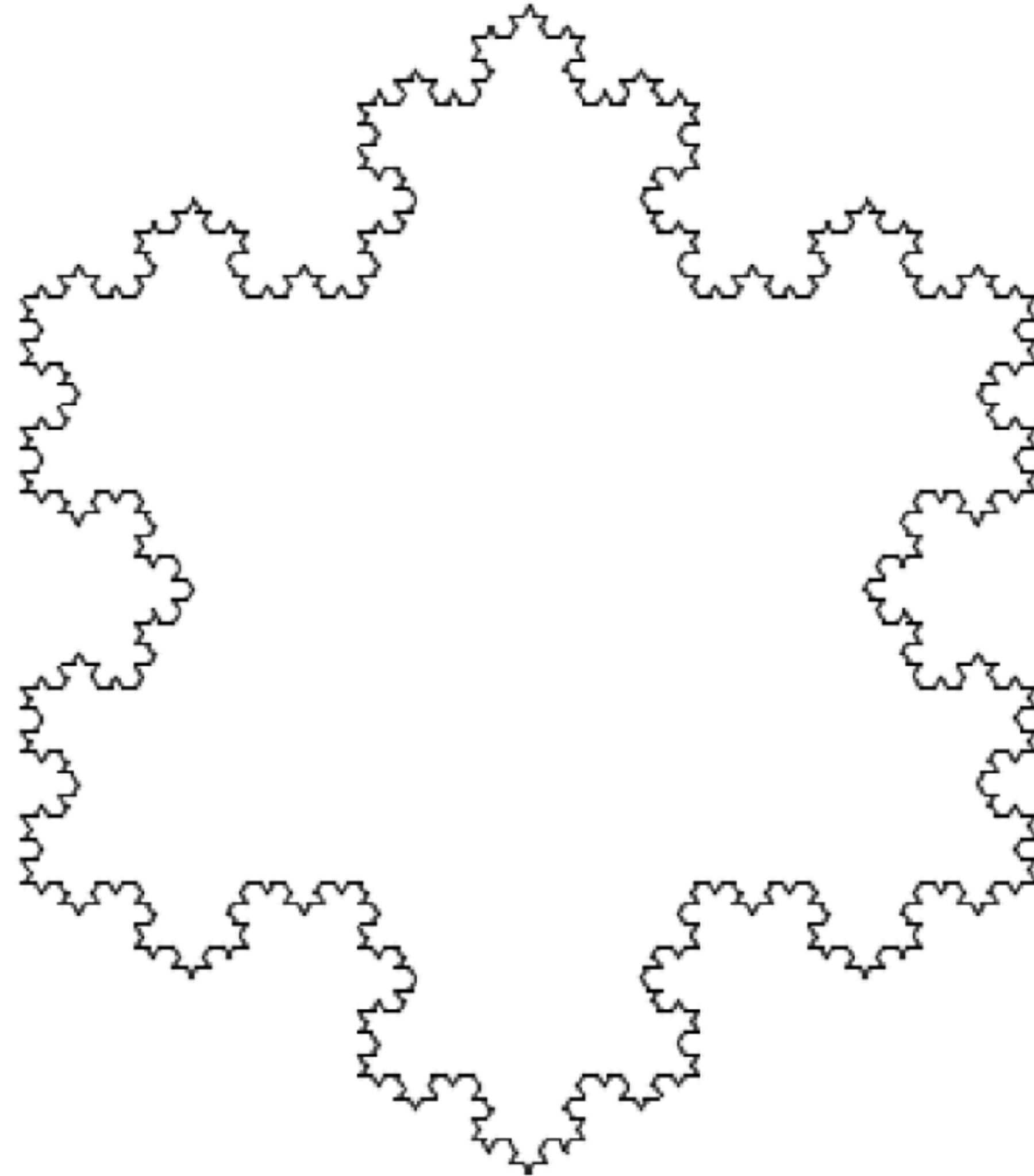
```
GWindow w;  
GPoint a(100, 100);  
GPoint b(20, 20);  
w.drawLine(a, b);
```



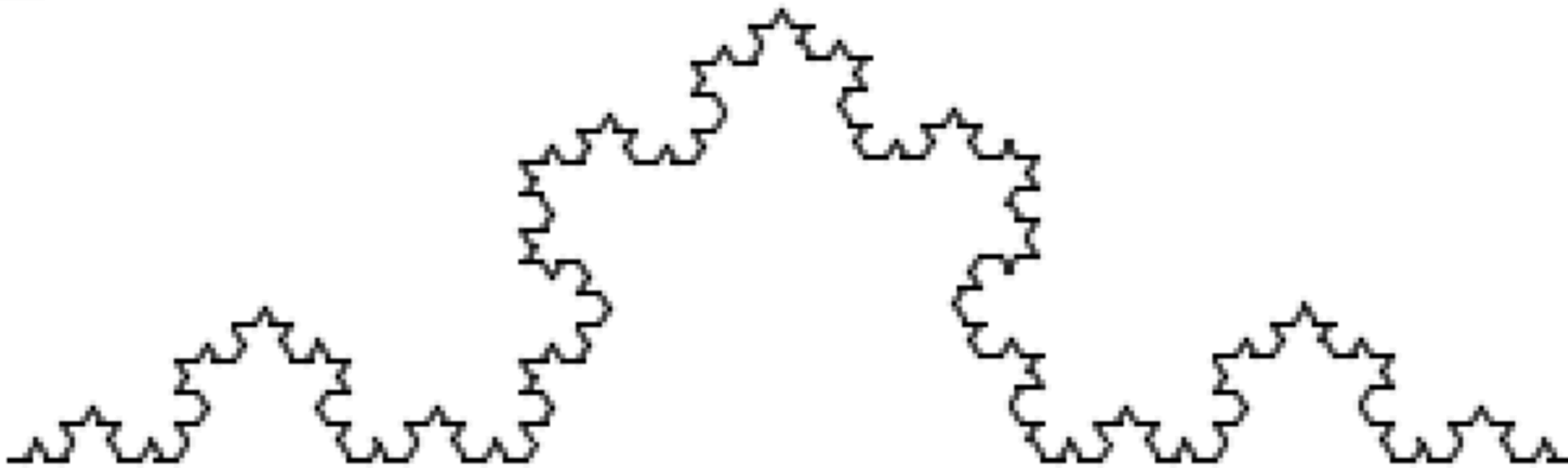
Cantor Fractal



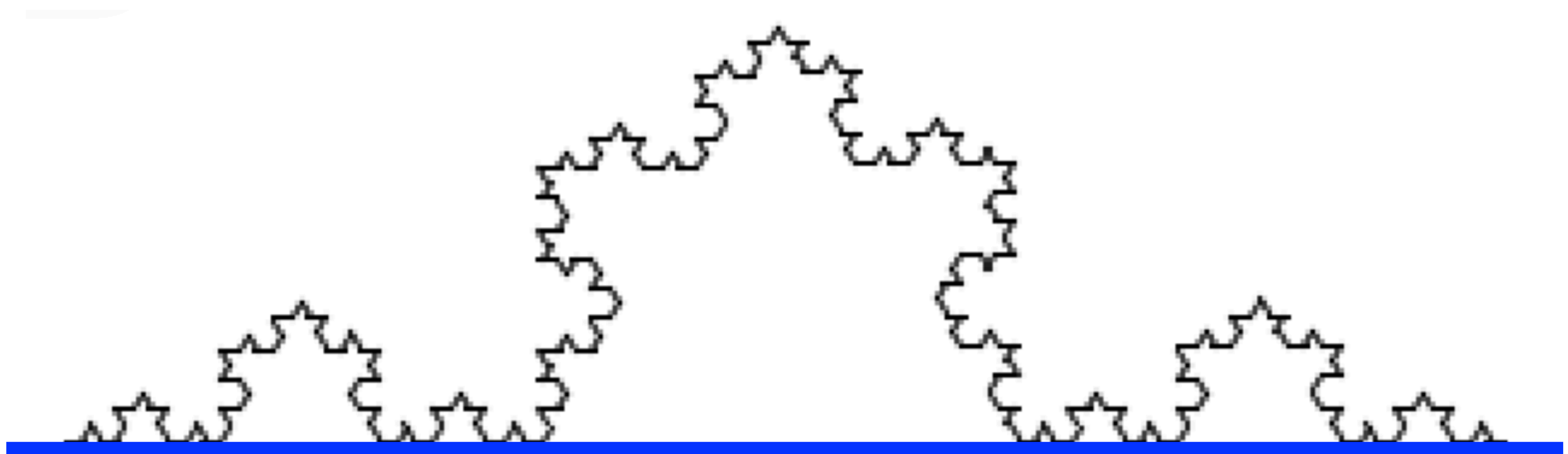
Snoflake Fractal



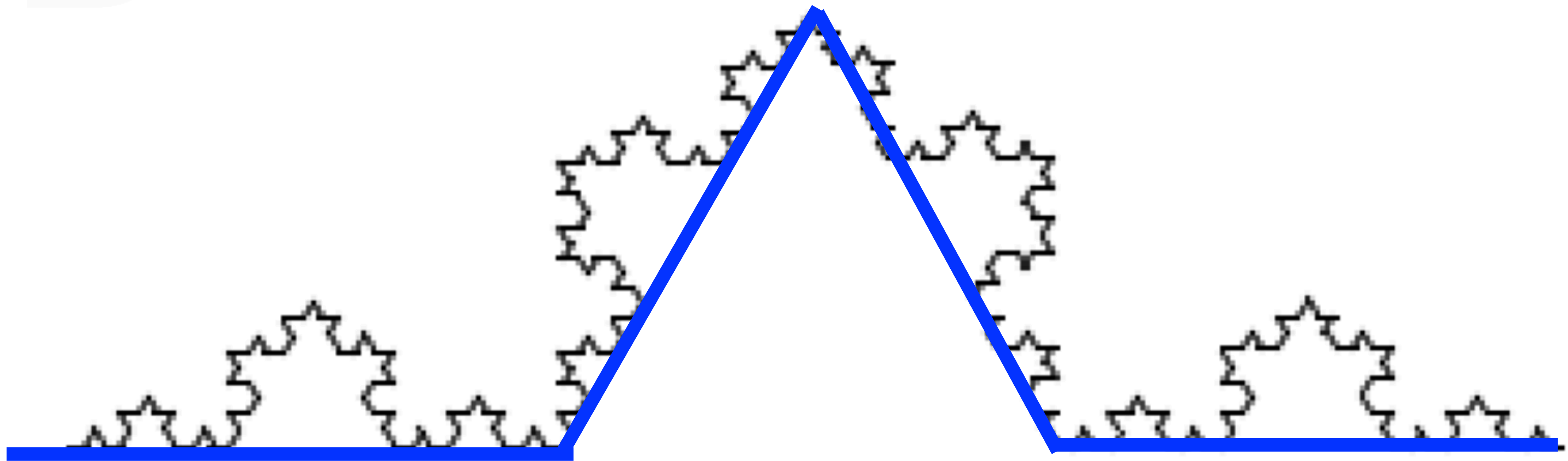
Snowflake Fractal



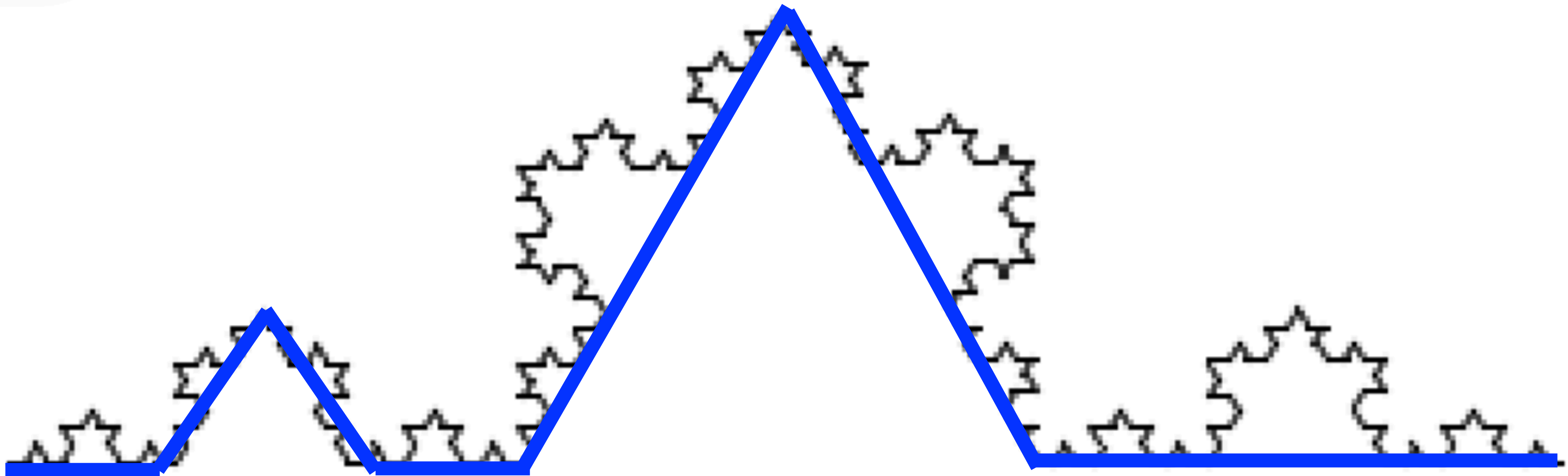
Depth 1 Snowflake Line



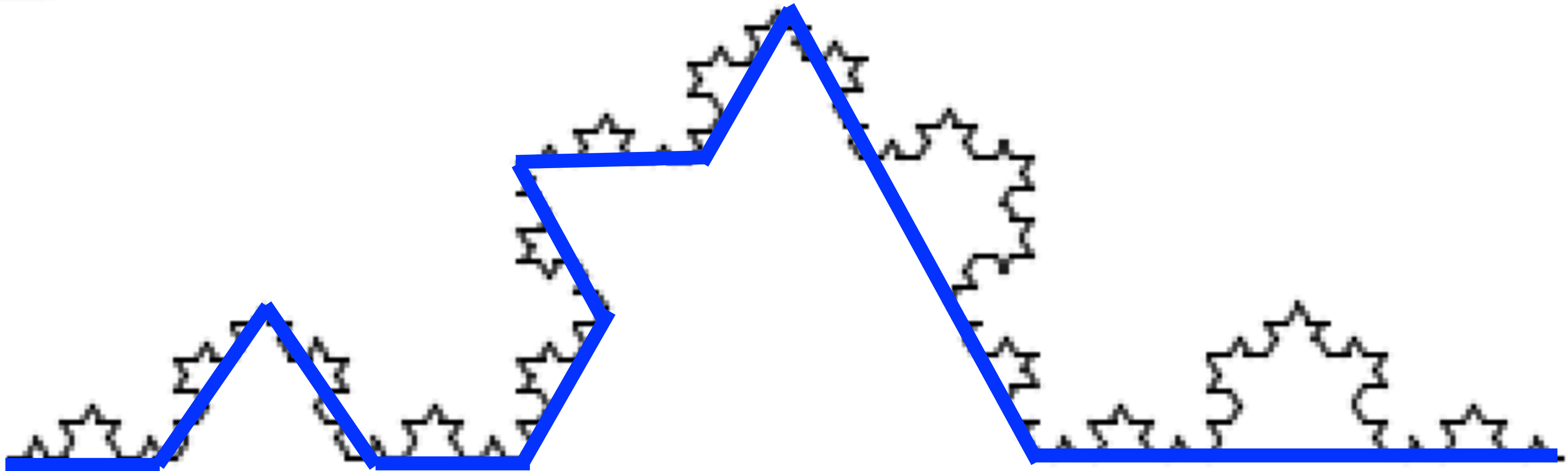
Depth 2 Snowflake Line



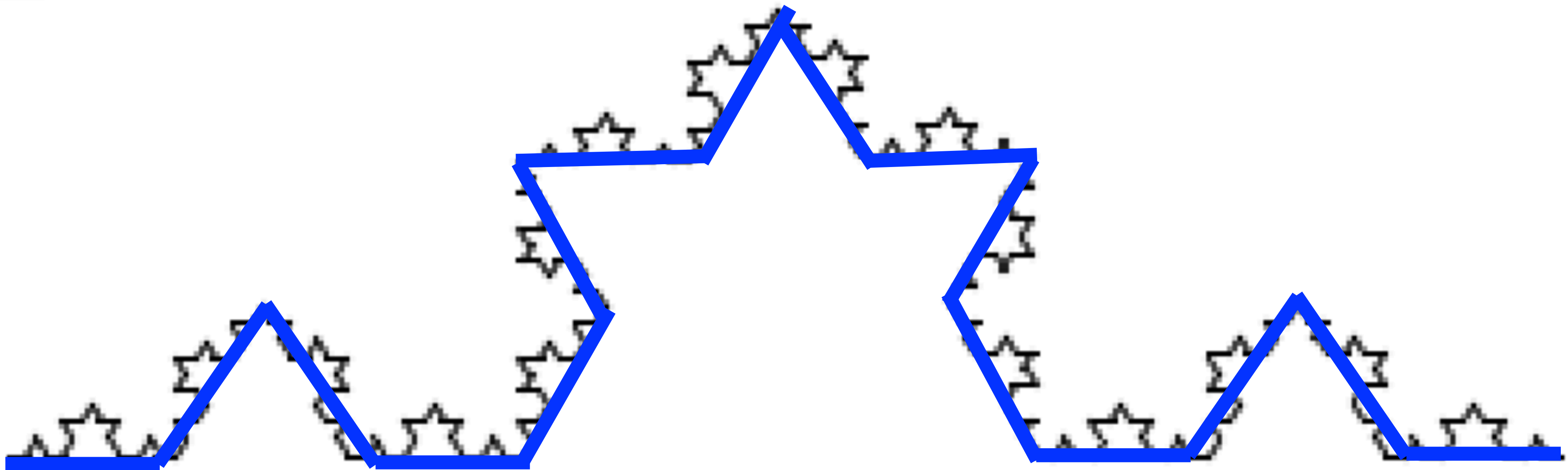
Depth 3 Snowflake Line (in progress)



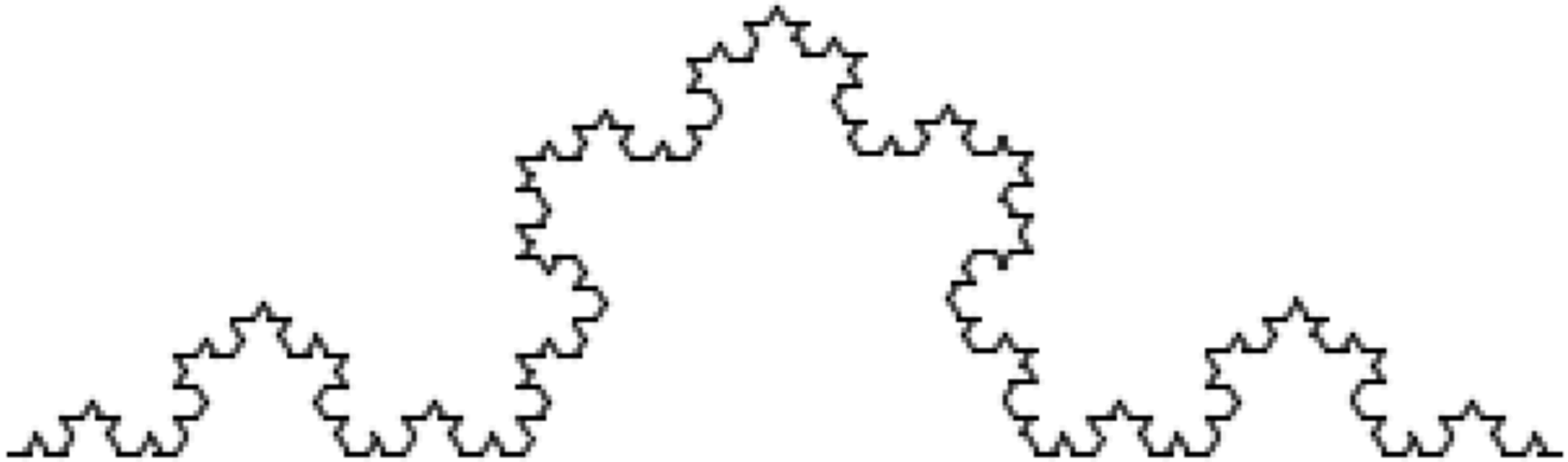
Depth 3 Snowflake Line (in progress)



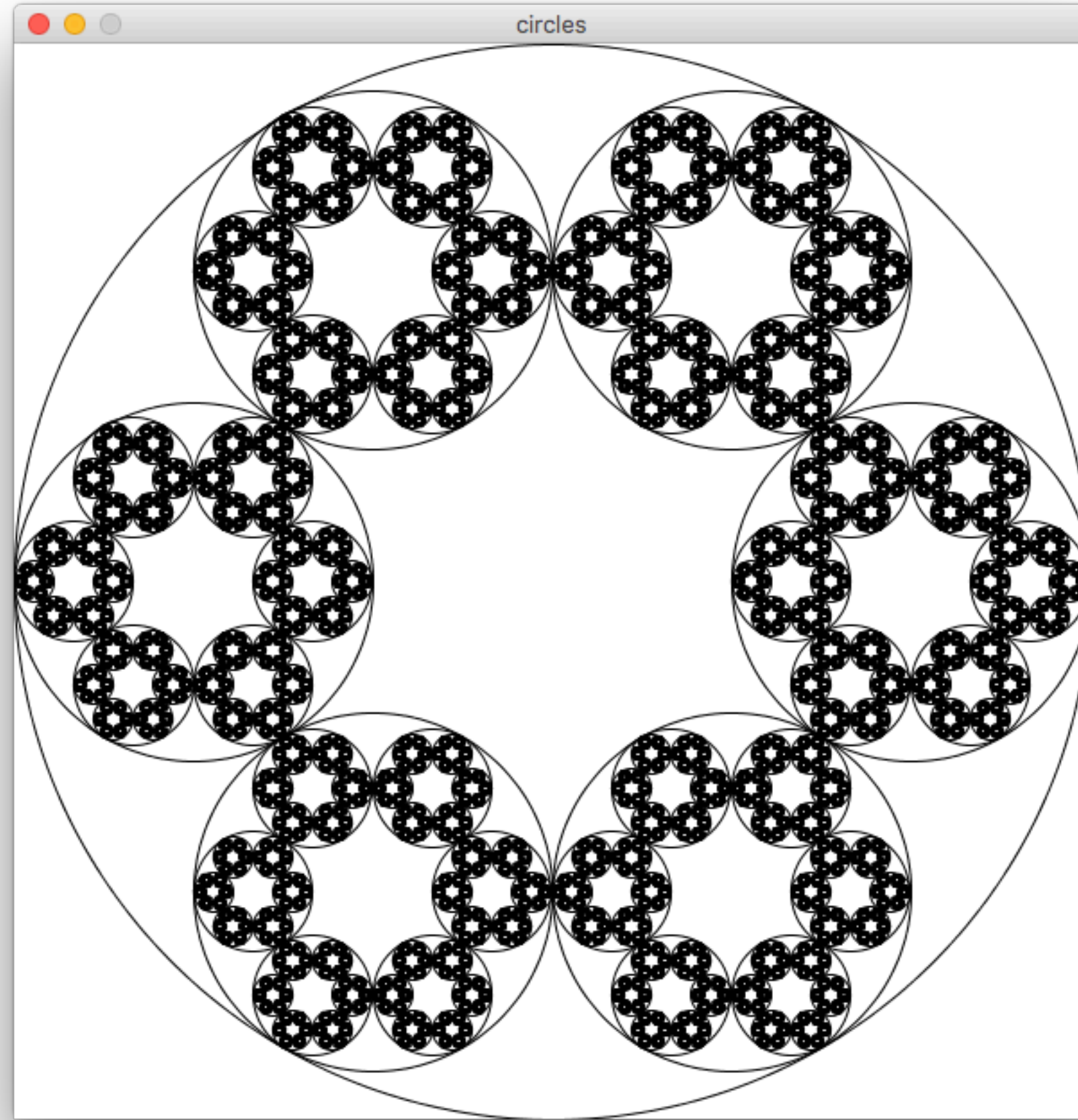
Depth 3 Snowflake Line (in progress)



Depth 3 Snowflake Line (in progress)



Another Example On the Website



Recap

- **Fractals**

- Fractals are self-referential, and that makes for nice recursion problems!
- Break the problem into a smaller, self-similar part, and don't forget your base case!



References and Advanced Reading

- **References:**

- <http://www.cs.utah.edu/~germain/PPS/Topics/recursion.html>
- Why is iteration generally better than recursion? <http://stackoverflow.com/a/3093/561677>

- **Advanced Reading:**

- Tail recursion: <http://stackoverflow.com/questions/33923/what-is-tail-recursion>
- Interesting story on the history of recursion in programming languages: <http://goo.gl/P6Einb>



Extra Slides

