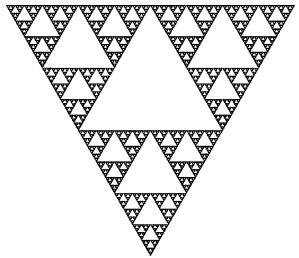
YEAH - Recursion!

Jason Chen
Original slides by: Anton Apostolatos

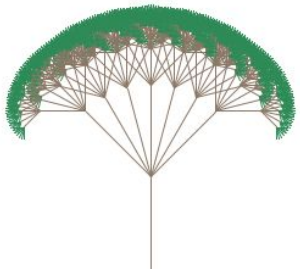
Source: The Office

A3: Recursion!

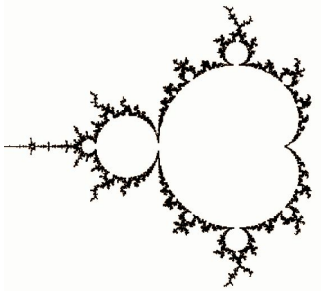
Sierpinksi



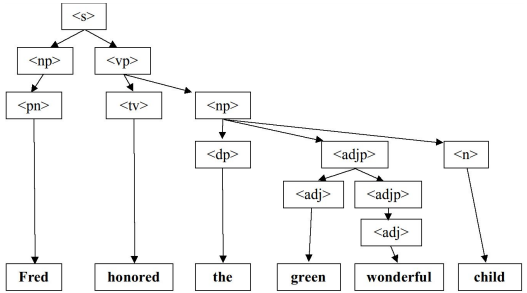
Recursive Tree (extension)



Mandelbrot Set



Grammar Solver



Two things to be clear on:

1. what your target or goal is.

2. what stuff you have to work with.

State

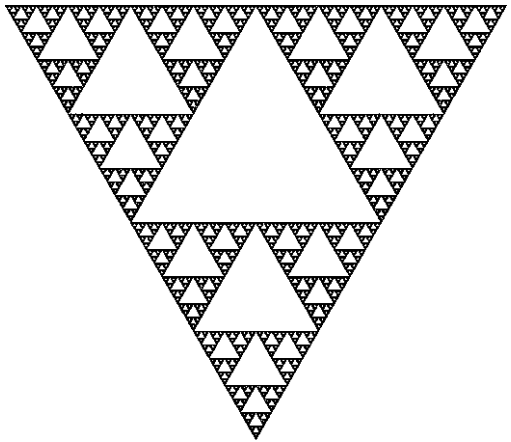
1. current state vs target state (are we there yet?)

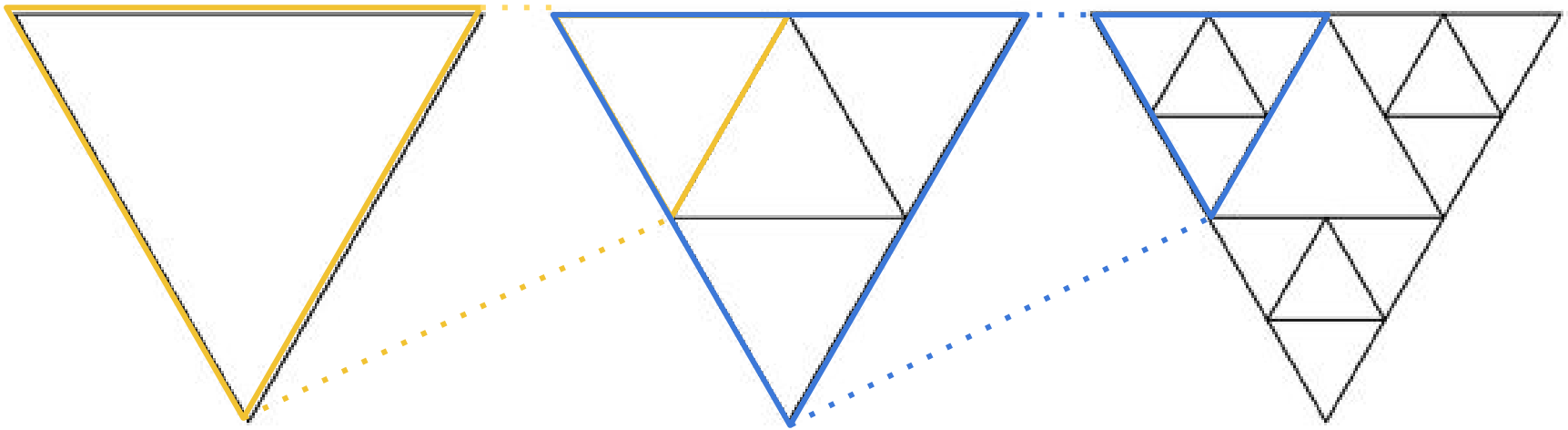
2. given this state, what can you change to move towards your target?

3. given this change, how does the state change?

[repeat]

Sierpinski

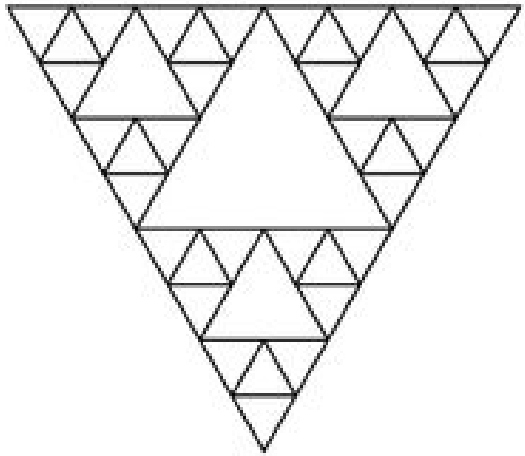




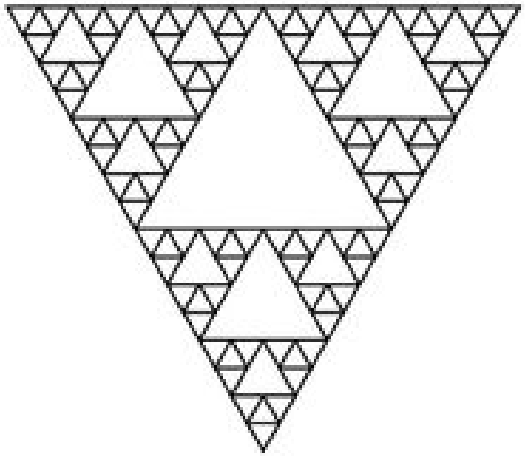
Order 1

Order 2

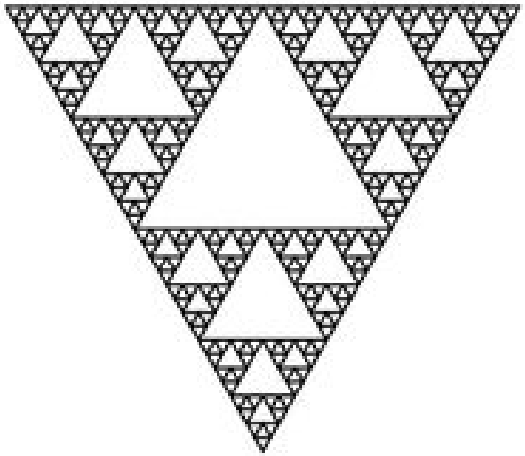
Order 3



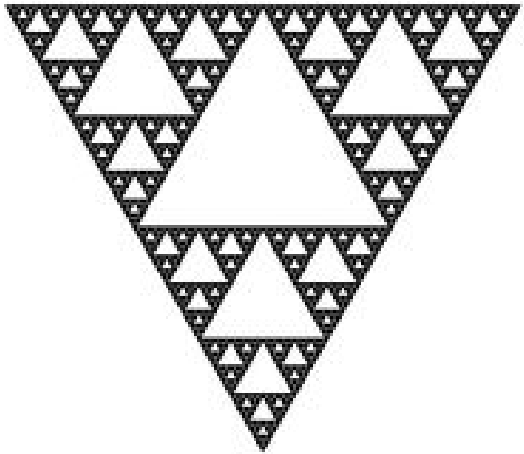
Order 4



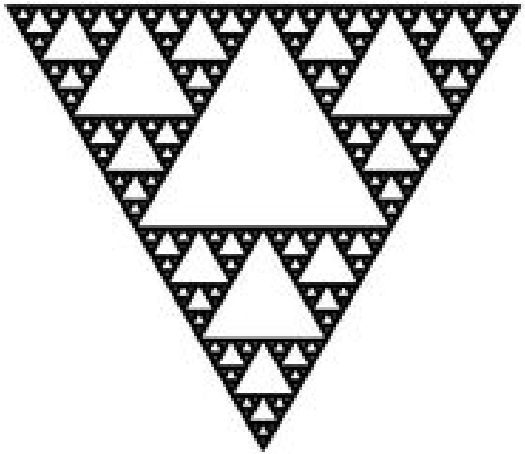
Order 5



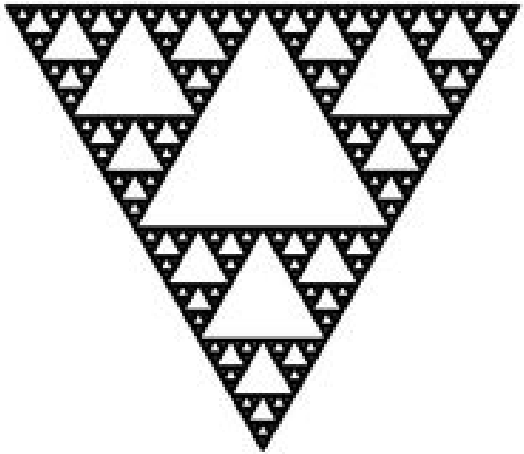
Order 6



Order 7



Order 8

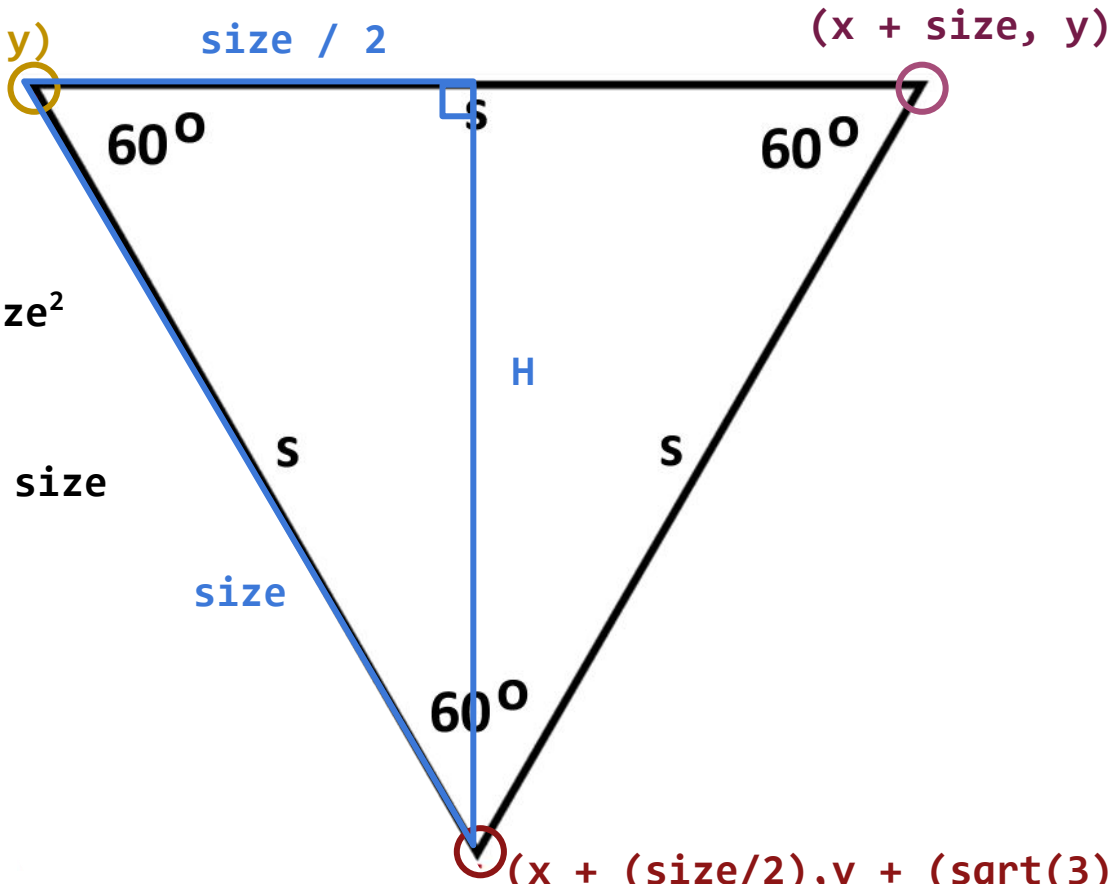


Order 9

Write the recursive function

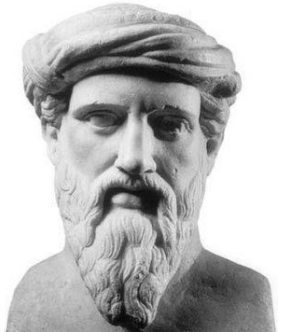
```
void drawSierpinskiTriangle(GWindow& gw,
   double x, double y, double size, int order)
```

gw: where to draw the triangle (see C++ docs! specifically the *drawLine* function)
(x, y): top-left corner of the triangle
size: length of triangle side
order: the order of the triangle to draw



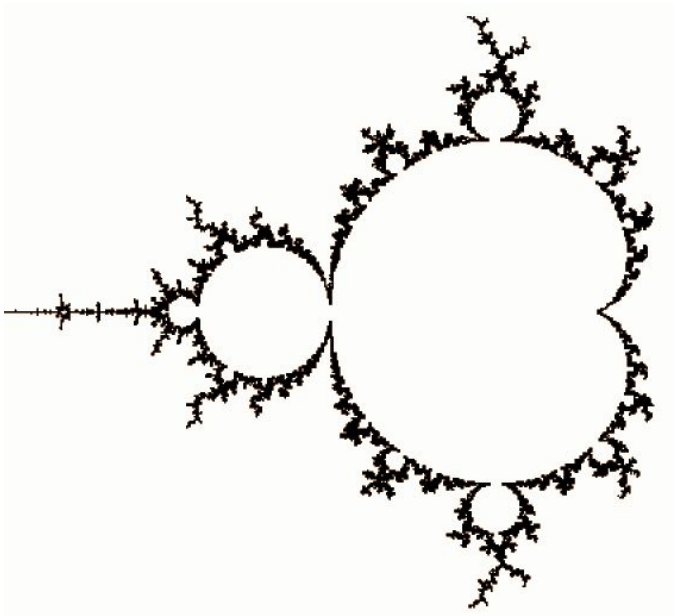
$(size / 2)^2 + H^2 = size^2$

$H = (sqrt(3) / 2) * size$



Questions?

Mandelbrot Set



Definition of a complex number

$$Z = a + bi$$

Mandelbrot Set Definition: A complex number **C** is in the Mandelbrot set if, as n approaches infinity, $\mathbf{Z_n}$ does not converge where $\mathbf{Z_0 = 0}$ and:

$$z_{n+1} = z_n^2 + c$$

CS106B's Mandelbrot Set Definition: A complex number **C** is in the Mandelbrot set if, after **maxIterations**, $\mathbf{Z_{maxIterations}}$ is not greater than **4** (diverging) where $\mathbf{Z_0 = 0}$ and:

$$z_{n+1} = z_n^2 + c$$

What is z and what is c?

$$z_{n+1} = z_n^2 + c$$

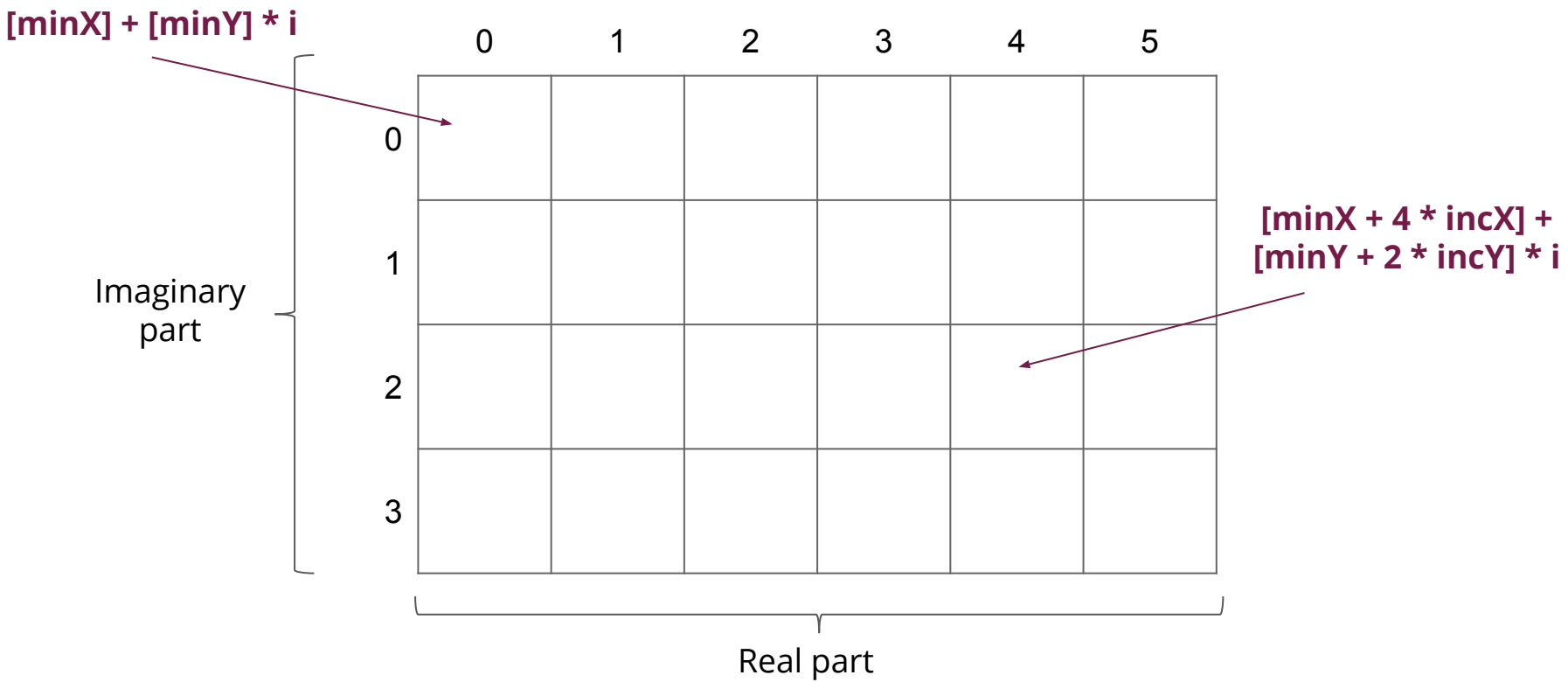
$$z_0 = 0, \quad n \to \infty$$

Zo = 0; Z1 = c; Z2 = Z1*Z1 + c

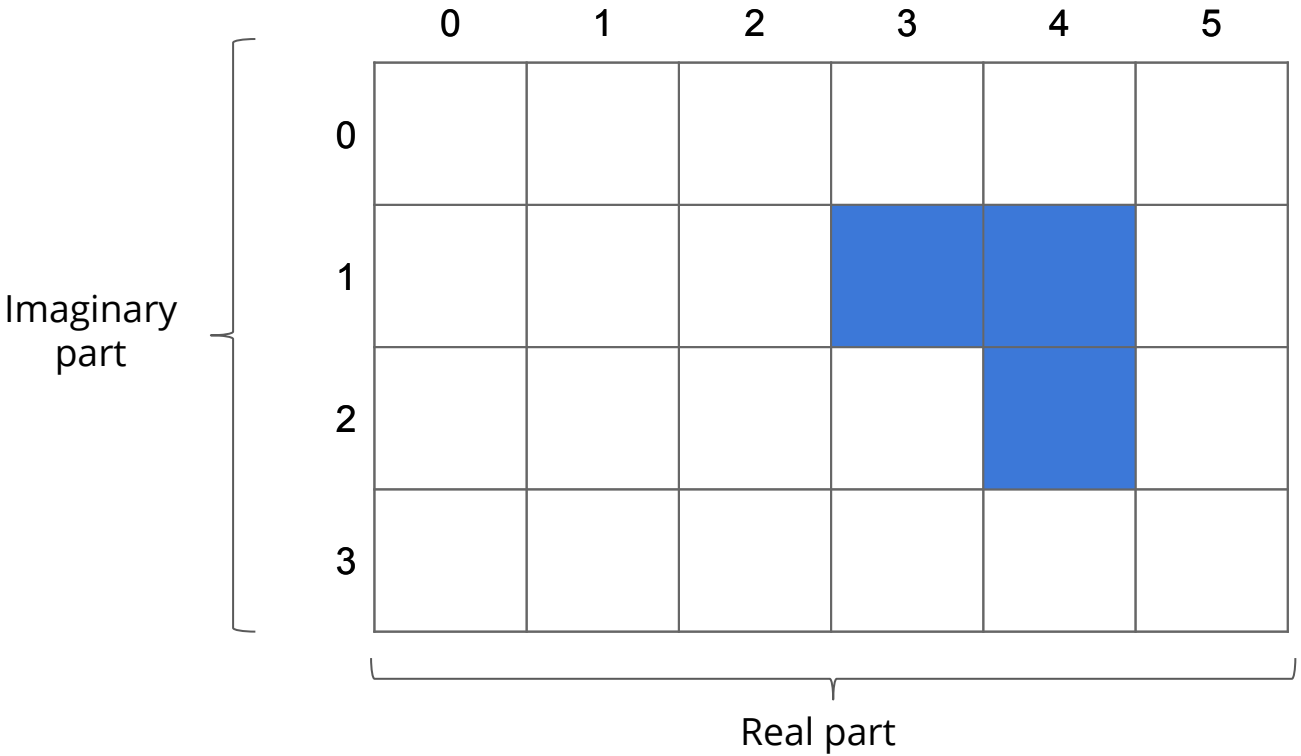
Function	Description
`Complex(double a, double b)`	Constructor that creates a complex number in the form a + b*i*
`cpx.abs()`	returns the absolute value of the number (a double)
`cpx.realPart()`	returns the real part of the complex number
`cpx.imagPart()`	returns the coefficient of the imaginary part of the complex number
`cpx1 + cpx2`	returns a complex number that is the addition of two complex numbers
`cpx1 * cpx2`	returns a complex number that is the product of two complex numbers

Complex Plane

Conversion: **(row, col) → [minX + col * incX] + [minY + row * incY] * i**



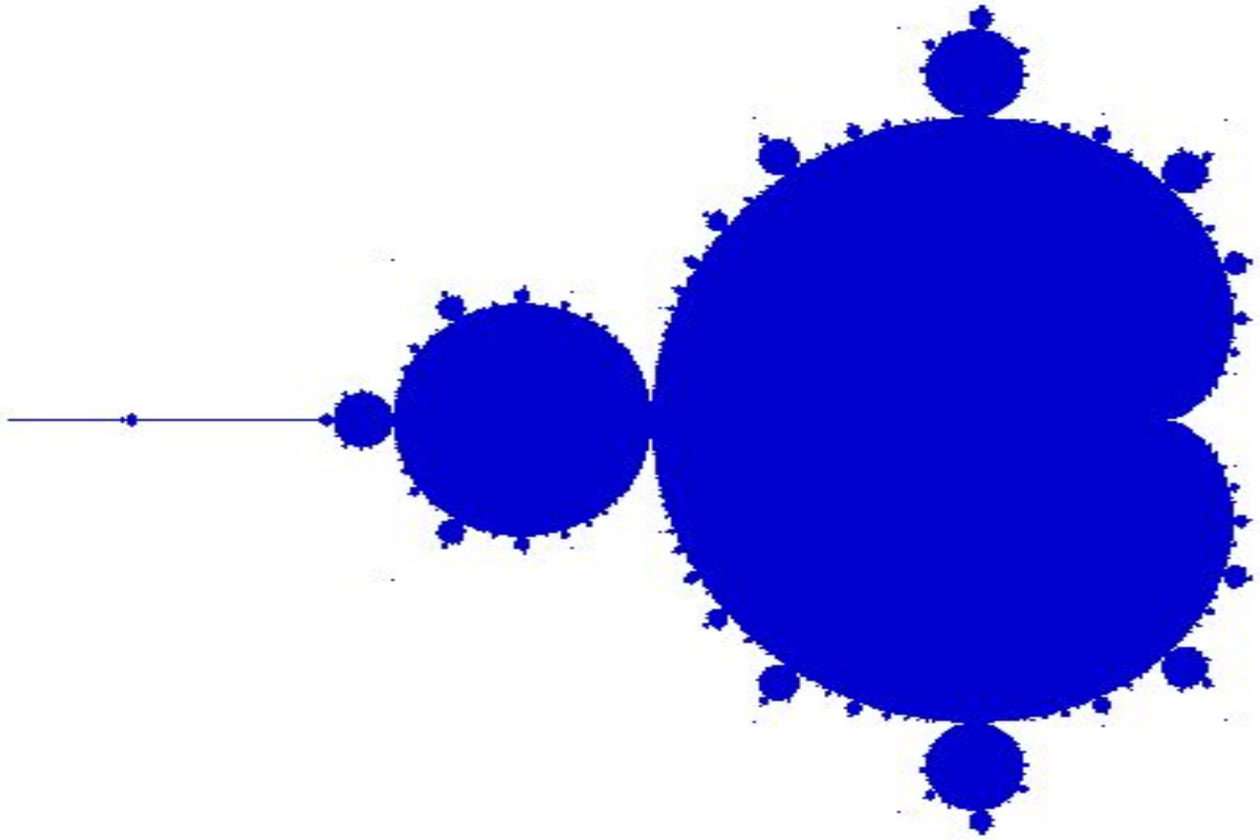
Let’s say only (1, 3), (1,4) and (2,4) are in the Mandelbrot Set



Write the recursive function

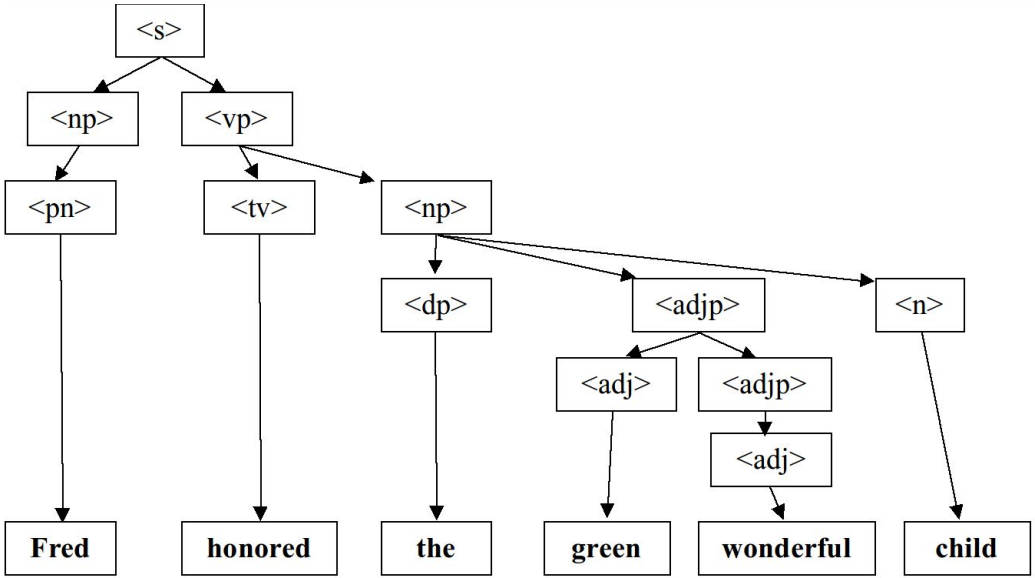
```
int mandelbrotSet(GWindow& gw, double minX,
   double incX, double minY, double incY,
   int maxIterations, int color)
```

gw: where to draw the Mandelbrot Set
(minX, minY): values of top-left corner of the grid
(incX, incY): how much to increment per row/col
maxIterations: the maximum number of iterations
color: the color of the Mandelbrot set



Questions?

Grammar Solver

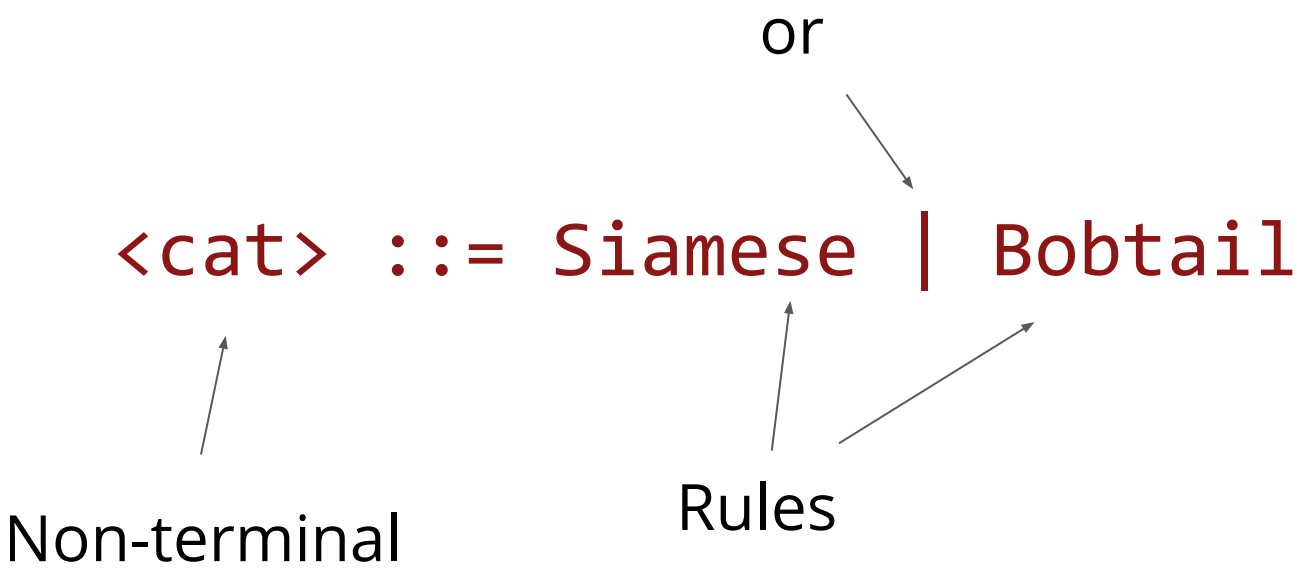


Definitions

Formal language: set of words or symbols along with a set of rules, called syntax of a language

Grammar: way of describing the syntax of a language

Backus-Naur Form (BNF): set of rules where each rule names a **symbol** and the symbol's **legal transformations**

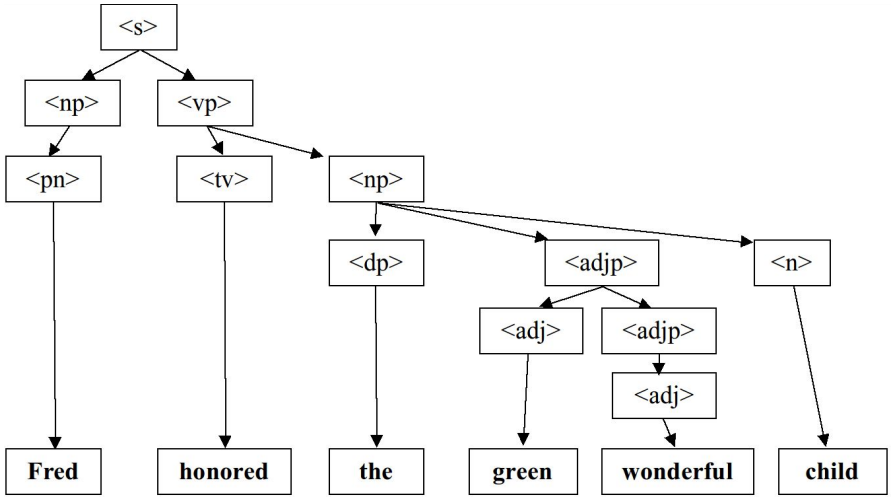

or
<cat> ::= Siamese | Bobtail
Non-terminal
Rules

```
<household-pet> ::= <cat> | <dog>
<cat> ::= Siamese | Bobtail
<dog> ::= Labrador | Xoloitzcuintle
```

```
<s>::=<np> <vp>
<np>::=<dp> <adjp> <n>|<pn>
<dp>::=the|a
<adjp>::=<adj>|<adj> <adjp>
<adj>::=big|fat|green|wonderful|faulty|subliminal|pretentious
<n>::=dog|cat|man|university|father|mother|child|television
<pn>::=John|Jane|Sally|Spot|Fred|Elmo
<vp>::=<tv> <np>|<iv>
<tv>::=hit|honored|kissed|helped
<iv>::=died|collapsed|laughed|wept
```

The fat university laughed

Elmo kissed a green pretentious television



How does this compare to N-Grams?

Write the function

```
Vector<string> grammarGenerate(istream& input,
    string symbol, int times)
```

input: input stream with file in BNF form
symbol: symbol to generate
times: number of times to generate symbol

Sample run

```
Symbol to generate (Enter to quit)? <s>
How many to generate? 7

 1: a green green big dog honored Fred
 2: the big child collapsed
 3: a subliminal dog kissed the subliminal television
 4: Fred died
 5: the pretentious fat subliminal mother wept
 6: Elmo honored a faulty television
 7: Elmo honored Elmo
```

Step 1: Reading Input File

- Store contents of the grammar into a **Map**
  - Think about what key/value data types or collections you want to use!
- The **stringSplit** and **trim** functions can be very helpful from **strlib.h** (Read the documentation!)

```
stringSplit(“hello;there”, ";");   // {"hello", "there"}

trim(“   hello  there ”);   // "hello  there"}
```

<s>::=<np> <vp>
<np>::=<dp> <adjp> <n>|<pn>
<dp>::=the|a
<adjp>::=<adj>|<adj> <adjp>
<adj>::=big|fat|green|wonderful|faulty|subliminal|pretentious
<n>::=dog|cat|man|university|father|mother|child|television
<pn>::=John|Jane|Sally|Spot|Fred|Elmo
<vp>::=<tv> <np>|<iv>
<tv>::=hit|honored|kissed|helped
<iv>::=died|collapsed|laughed|wept

E::=T|E OP T
T::=x|y|42|0|1|92|(E)|F1 (E)|- T|F2 (E , E)
OP::=+|-|*|%|/
F1::=sin|cos|tan|sqrt|abs
F2::=max|min|pow

Recursively

Step 2: Generating Random Expressions

- **If S is a terminal symbol:** result is symbol
- **If S is a non-terminal symbol:** choose random rule for S and explore it

Questions?