YEAH - Recursion!
Anton Apostolatos

Source: The Office
A3: Recursion!

- Sierpinski
- Recursive Tree
- Mandelbrot Set
- Grammar Solver
Sierpinski
Order 7
Order 8
Order 9
Write the recursive function

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

- **gw**: where to draw the triangle (see C++ docs!)
- **(x, y)**: top-left corner of the triangle
- **size**: length of triangle side
- **order**: the order of the triangle to draw
\[(\text{size} / 2)^2 + H^2 = \text{size}^2\]

\[H = (\sqrt{3} / 2) \times \text{size}\]
Questions?
Recursive Tree
Write the recursive function

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

gw: where to draw the triangle
(x, y): top-left corner of the bounding box
size: length of bounding box
order: the order of the tree to draw
Every order \( n \) tree has seven trunks of order \( n-1 \)

**Lengths:** each trunk has half their parent’s length

**Angles:** Each subtree extends from the tip of the previous trunk at angles ±45, ±30, ±15, and 0 degrees

**Colors:** Branches order >=2 are drawn in a color of `BRANCH_COLOR` and the leafy fringe branches of the tree (branches drawn at level 1) are drawn in a color of `LEAF_COLOR`
Useful functions

<table>
<thead>
<tr>
<th>Member</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>gw.drawPolarLine(x0, y0, r, theta);</code></td>
<td>draws a line the given starting point, of the given length $r$, at the given angle in degrees $\theta$ relative to the origin</td>
</tr>
<tr>
<td><code>gw.drawPolarLine(p0, r, theta);</code></td>
<td></td>
</tr>
<tr>
<td><code>gw setColor(color);</code></td>
<td>sets color used for future shapes to be drawn, either as a hex string such as &quot;#aa00ff&quot; or an RGB integer such as 0xaa00ff</td>
</tr>
</tbody>
</table>

**Note:** recursive helper functions are sometimes required!
Recursive helper functions are friends, not food

```c
void drawPrettyTrees(GWindow& gw)

void drawTree(GWindow& gw, double x, double y, double size, int order)
```
Questions?
Mandelbrot Set
Definition of a complex number

\[ Z = a + bi \]

Real part

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

$$Z_{n+1} = Z_n^2 + C$$
**CS106B’s Mandelbrot Set Definition:** A complex number $C$ is in the Mandelbrot set if, after $\text{maxIterations}$, $Z^{\text{maxIterations}}$ is not greater than 4 (diverging) where $Z_0 = 0$ and:

$$Z_{n+1} = Z_n^2 + C$$
Complex Plane

Conversion: \((row, col) \rightarrow [\text{minX} + col \times \text{incX}] + [\text{minY} + row \times \text{incY}] \times i\)
Let’s say only (1, 3), (1,4) and (2,4) are in the Mandelbrot Set
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Complex(double (a), double (b))</td>
<td>Constructor that creates a complex number in the form (a + bi)</td>
</tr>
<tr>
<td><code>cpx.abs()</code></td>
<td>returns the absolute value of the number (a double)</td>
</tr>
<tr>
<td><code>cpx.realPart()</code></td>
<td>returns the real part of the complex number</td>
</tr>
<tr>
<td><code>cpx.imagPart()</code></td>
<td>returns the coefficient of the imaginary part of the complex number</td>
</tr>
<tr>
<td><code>cpx1 + cpx2</code></td>
<td>returns a complex number that is the addition of two complex numbers</td>
</tr>
<tr>
<td><code>cpx1 * cpx2</code></td>
<td>returns a complex number that is the product of two complex numbers</td>
</tr>
</tbody>
</table>
Write the recursive function

```cpp
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?
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
\[ \text{cat} ::= \text{Siamese} \mid \text{Bobtail} \]
\texttt{\textlt{household-pet}\texttt{\textop{::=}} \textlt{cat} \mid \textlt{dog}}

\texttt{\textlt{cat}\texttt{::=} \textlt{Siamese}\mid \textlt{Bobtail}}

\texttt{\textlt{dog}\texttt{::=} \textlt{Labrador}\mid \textlt{Xoloitzcuintle}}
The fat university laughed
Elmo kissed a green pretentious television
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!)

```cpp
stringSplit("hello;there", ";");  // {"hello", "there"}
trim(" hello there ", ";");  // "hello there"
```
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?