CS 106B, Lecture 10
Recursion and Fractals
Plan for Today

• Introduction to **fractals**, a powerful tool used in graphics
Fractals

- **fractal**: A self-similar mathematical set that can often be drawn as a recurring graphical pattern.
  - Smaller instances of the same shape or pattern occur within the pattern itself.
  - When displayed on a computer screen, it can be possible to infinitely zoom in/out of a fractal.
Many natural phenomena generate fractal patterns:

- earthquake fault lines
- animal color patterns
- clouds
- mountain ranges
- snowflakes
- crystals
- DNA
- shells
- ...
Example fractals

- **Sierpinski triangle**: equilateral triangle contains smaller triangles inside it (your next homework)

- **Koch snowflake**: a triangle with smaller triangles poking out of its sides

- **Mandelbrot set**: circle with smaller circles on its edge
Coding a fractal

• Many fractals are implemented as a function that accepts x/y coordinates, size, and a level parameter.
  – The level is the number of recurrences of the pattern to draw.
  – The position and size change in the recursive call; level decreases by 1

• Example, Koch snowflake:
  \texttt{snowflake}(\texttt{window, x, y, size, 1});

  \texttt{snowflake}(\texttt{window, x, y, size, 2});

  \texttt{snowflake}(\texttt{window, x, y, size, 3});
Boxy fractal

Where should the following lines be inserted in order to get the figure at right?

```java
gw.setFillColor("gray");
gw.fillRect(x, y, size, size);

void boxyFractal(GWindow& gw, int x, int y, int size, int order) {
    if (order >= 1) {
        // A
        boxyFractal(gw, x - size / 2, y - size / 2, size / 2, order - 1);
        // B
        boxyFractal(gw, x + size / 2, y + size / 2, size / 2, order - 1);
        // C
        boxyFractal(gw, x + size / 2, y - size / 2, size / 2, order - 1);
        // D
        boxyFractal(gw, x - size / 2, y + size / 2, size / 2, order - 1);
        // E
    }
}
```
#include "gwindow.h"

```c
GWindow gw(300, 200);
gw.setTitle("CS 106X Fractals");
gw.drawLine(20, 20, 100, 100);
```

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>gw.drawLine(x1, y1, x2, y2);</code></td>
<td>draws a line between the given two points</td>
</tr>
<tr>
<td><code>gw.drawPolarLine(x, y, r, t);</code></td>
<td>draws line from (x,y) at angle t of length r; returns the line's end point as a GPoint</td>
</tr>
<tr>
<td><code>gw.getPixel(x, y)</code></td>
<td>returns an RGB int for a single pixel</td>
</tr>
<tr>
<td><code>gw.setColor(&quot;color&quot;);</code></td>
<td>sets color with a color name string like &quot;red&quot;, or #RRGGBB string like &quot;#ff00cc&quot;, or RGB int</td>
</tr>
<tr>
<td><code>gw.setPixel(x, y, rgb);</code></td>
<td>sets a single RGB pixel on the window</td>
</tr>
<tr>
<td><code>gw.drawOval(x, y, w, h);</code></td>
<td>other shape and line drawing functions (see online docs for complete member list)</td>
</tr>
<tr>
<td><code>gw.fillRect(x, y, w, h);</code></td>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Axes</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0, 0)</td>
</tr>
<tr>
<td>x+</td>
</tr>
<tr>
<td>y+</td>
</tr>
</tbody>
</table>

![GWindow gw(300, 200);
  gw.setTitle("CS 106X Fractals");
  gw.drawLine(20, 20, 100, 100);](image-url)
The **Cantor Set** is a simple fractal that begins with a line segment.
- At each *level*, the middle third of the segment is removed.
- In the next *level*, the middle third of each third is removed.

Write a function `cantorSet` that draws a Cantor Set with a given number of levels (lines) at a given position/size.
- Place CANTOR_SPACING of vertical space between levels.

• How is this fractal *self-similar*?
• What is the *minimum amount of work* to do at each level?
• What's a good stopping point (base case)?
void cantorSet(GWindow& window, int x, int y,
                int width, int levels) {
    if (levels > 0) {
        // recursive case: draw line, then repeat by thirds
        window.drawLine(x, y, x + width, y);
        cantorSet(window, x, y + 20, width/3, levels-1);
        cantorSet(window, x + 2*width/3, y + 20, width/3, levels-1);
    }
    // else, base case: 0 levels, do nothing
}
Q: Which way does the drawing animate?  

(How could we change it?)

```cpp
void cantorSet(GWindow& window, int x, int y, int width, int levels) {
    if (levels > 0) {
        // recursive case: draw line, then repeat by thirds
        pause(250);
        window.drawLine(x, y, x + width, y);
        cantorSet(window, x, y + 20, width/3, levels-1);
        cantorSet(window, x + 2*width/3, y + 20, width/3, levels-1);
    }
}
```

//   A.               B.               C.               D.
Announcements

• Homework 2 due today at **5PM**
• Homework 1 grades will be released by your section leader soon!
• Tyler does not have OH today (or tomorrow, since there is no class)
**Koch snowflake**

- **Koch snowflake**: A fractal formed by pulling a triangular "bend" out of each side of an existing triangle at each level.

- Start with an equilateral triangle, then:
  - Divide each of its 3 line segments into 3 parts of equal length.
  - Draw an eq.triangle with middle segment as base, pointing outward.
  - Remove the middle line segment.
Line segment replace

- Replace each line segment as follows:
Multiple levels

• How is this fractal self-similar?
Polar lines

// x   y    r   theta
window.drawPolarLine(20, 20, 113, -45);

-45 degrees

113 pixels
Triangle in polar

• Segment 1:  

Segment 2:  

Segment 3:
Segment in polar

- Think of a triangle side as 4 polar line segments, as below.
  - What are their angles, relative to the angle of this triangle side?
Snowflake solution

```
GPoint ksLine(GWindow& gw, GPoint pt, int size, int t, int levels) {
    if (levels == 1) {
        return gw.drawPolarLine(pt, size, t);
    } else {
        pt = ksLine(gw, pt, size/3, t, levels - 1);
        pt = ksLine(gw, pt, size/3, t + 60, levels - 1);
        pt = ksLine(gw, pt, size/3, t - 60, levels - 1);
        return ksLine(gw, pt, size/3, t, levels - 1);
    }
}

void kochSnowflake(GWindow& gw, int x, int y, int size, int levels) {
    GPoint pt(x, y);
    pt = ksLine(gw, pt, size, 0, levels);
    pt = ksLine(gw, pt, size, -120, levels);
    pt = ksLine(gw, pt, size, 120, levels);
}
```
• Write a recursive function `fib` that accepts an integer $N$ and returns the $N$th Fibonacci number.
  – The first two Fibonacci numbers are defined to be 1.
  – Every other Fibonacci number is the sum of the two before it.

```
  fib(1) => 1
  fib(2) => 1
  fib(3) => 2
  fib(4) => 3
  fib(5) => 5
  fib(6) => 8
  fib(7) => 13
  fib(8) => 21
  fib(9) => 34
  ...
```
// Returns the nth Fibonacci number.
int fib(int n) {
    if (n <= 2) {
        return 1;
    } else {
        return fib(n - 1) + fib(n - 2);
    }
}

// what does the call stack look like?
Bad fib solution

**Figure 7-2** Steps in the calculation of \( \text{fib}(5) \)

\[
\begin{align*}
\text{fib}(5) & \rightarrow \text{fib}(4) & \text{fib}(3) \\
\text{fib}(4) & \rightarrow \text{fib}(3) & \text{fib}(2) \\
\text{fib}(3) & \rightarrow \text{fib}(2) & \text{fib}(2) \\
\text{fib}(2) & \rightarrow \text{fib}(1) & \text{fib}(1) \\
\text{fib}(1) & \rightarrow 1 & \text{fib}(0) \\
\text{fib}(0) & \rightarrow 0 & 1 \\
\end{align*}
\]
Memoization

- **memoization**: Caching results of previous expensive function calls for speed so that they do not need to be re-computed.
  - Often implemented by storing call results in a collection.

- Pseudocode template:

```python
    cache = {}  # empty

    function f(args):
        if I have computed f(args) before:
            Look up f(args) result in cache.
        else:
            Actually compute f(args) result.
            Store result in cache.
        Return result.
```
Wrapper Functions

• We don't want the user to have to worry about the cache!
  – Alternative to the default parameters we saw yesterday
• Some recursive functions need extra arguments to implement the recursion
• A wrapper function is a function that does some initial prep work, then fires off a recursive call with the right arguments.
• The recursion is done in the helper function
// Returns the nth Fibonacci number.
// This version uses memoization.
int fib(int n) { // wrapper function
    Map<int, int> cache;
    return fibHelper(n, cache);
}

int fibHelper(int n, Map<int, int> &cache) {
    if (n <= 2) {
        return 1;
    } else if (cache.containsKey(n)) {
        return cache[n];
    } else {
        int result = fibHelper(n - 1) + fibHelper(n - 2);
        cache[n] = result;
        return result;
    }
}