Thinking Recursively
Part II
Recursive Problem-Solving

if (problem is sufficiently simple) {
    Directly solve the problem.
    Return the solution.
}
else {
    Split the problem up into one or more smaller problems with the same structure as the original.
    Solve each of those smaller problems.
    Combine the results to get the overall solution.
    Return the overall solution.
}
A fractal is an object that contains a smaller copy of itself.
A fractal is an object that contains a smaller copy of itself.
A fractal is an object that contains a smaller copy of itself.
What differentiates the smaller tree from the bigger one?

1. It's at a different position.
2. It's at a different orientation.
3. It has a different size.
4. It has a different order.
What differentiates the smaller tree from the bigger one?

1. It’s at a different *position*.
What differentiates the smaller tree from the bigger one?

1. It’s at a different position.
2. It’s at a different size.
What differentiates the smaller tree from the bigger one?

1. It’s at a different position.
2. It’s at a different size.
3. It has a different orientation.
What differentiates the smaller tree from the bigger one?

1. It’s at a different position.
2. It’s at a different size.
3. It has a different orientation.
4. It has a different order.
What differentiates the smaller tree from the bigger one?

1. It’s at a different **position**.
2. It’s at a different **size**.
3. It has a different **orientation**.
4. It has a different **order**.

Recursive fractals are often described in terms of some parameter called the **order** of the fractal.
What differentiates the smaller tree from the bigger one?

1. It’s at a different position.
2. It’s at a different size.
3. It has a different orientation.
4. It has a different order.

Recursive fractals are often described in terms of some parameter called the order of the fractal.

An order-0 tree.
What differentiates the smaller tree from the bigger one?

1. It’s at a different position.
2. It’s at a different size.
3. It has a different orientation.
4. It has a different order.

Recursive fractals are often described in terms of some parameter called the order of the fractal.
Recursive fractals are often described in terms of some parameter called the order of the fractal.

What differentiates the smaller tree from the bigger one?

1. It’s at a different position.
2. It’s at a different size.
3. It has a different orientation.
4. It has a different order.
Recursive fractals are often described in terms of some parameter called the order of the fractal.

An order-3 tree.

What differentiates the smaller tree from the bigger one?

1. It’s at a different position.
2. It’s at a different size.
3. It has a different orientation.
4. It has a different order.
What differentiates the smaller tree from the bigger one?

1. It’s at a different \textit{position}.
2. It’s at a different \textit{size}.
3. It has a different \textit{orientation}.
4. It has a different \textit{order}.

Recursive fractals are often described in terms of some parameter called the \textit{order} of the fractal.
What differentiates the smaller tree from the bigger one?

1. It’s at a different position.
2. It’s at a different size.
3. It has a different orientation.
4. It has a different order.

Recursive fractals are often described in terms of some parameter called the order of the fractal.
An order-3 tree.

What differentiates the smaller tree from the bigger one?

1. It’s at a different position.
2. It’s at a different size.
3. It has a different orientation.
4. It has a different order.

Recursive fractals are often described in terms of some parameter called the order of the fractal.
An order-0 tree is nothing at all.

An order-\( n \) tree is a line with two smaller order-(\( n-1 \)) trees starting at the end of that line.

What differentiates the smaller tree from the bigger one?

1. It’s at a different **position**.
2. It’s at a different **size**.
3. It has a different **orientation**.
4. It has a different **order**.

Recursive fractals are often described in terms of some parameter called the **order** of the fractal.
An order-0 tree is nothing at all.

An order-\(n\) tree is a line with two smaller order-\((n-1)\) trees starting at the end of that line.

What differentiates the smaller tree from the bigger one?

1. It’s at a different \textit{position}.
2. It’s at a different \textit{size}.
3. It has a different \textit{orientation}.
4. It has a different \textit{order}.

Recursive fractals are often described in terms of some parameter called the \textit{order} of the fractal.
An order-0 tree is nothing at all.

An order-$n$ tree is a line with two smaller order-$(n-1)$ trees starting at the end of that line.

What differentiates the smaller tree from the bigger one?

1. It’s at a different position.
2. It’s at a different size.
3. It has a different orientation.
4. It has a different order.

Recursive fractals are often described in terms of some parameter called the order of the fractal.

We can draw lines in the window by calling

```javascript
window.drawPolarLine(x, y, r, θ);
```

with $θ$ specified in degrees.
```c
int main() {
    GWindow window(kWindowWidth, kWindowHeight);

    double treeRootX = /* Here be dragons */;
    double treeRootY = /* Dragons, dragons, dragons */;
    double treeHeight = /* I like dragons! */;

    drawTree(window, treeRootX, treeRootY, treeHeight, 90, 8);

    return 0;
}
```
```c
int main() {
    GWindow window(kWindowWidth, kWindowHeight);

double treeRootX  = /* Here be dragons */;
double treeRootY  = /* Dragons, dragons, dragons */;
double treeHeight = /* I like dragons! */;

drawTree(window, treeRootX, treeRootY, treeHeight, 90, 8);

return 0;
}
```
int main() {
    GWindow window(kWindowWidth, kWindowHeight);

    double treeRootX  = /* Here be dragons */;
    double treeRootY  = /* Dragons, dragons, dragons */;
    double treeHeight = /* I like dragons! */;

    drawTree(window, treeRootX, treeRootY, treeHeight, 90, 8);

    return 0;
}
```c
int main() {
    GWindow window(kWindowWidth, kWindowHeight);

    double treeRootX  = /* Here be dragons */;
    double treeRootY  = /* Dragons, dragons, dragons */;
    double treeHeight = /* I like dragons! */;

    drawTree(window, treeRootX, treeRootY, treeHeight, 90, 8);

    return 0;
}
```
int main() {
    GWindow window(kWindowWidth, kWindowHeight);
    
    double treeRootX = /* Here be dragons */;
    double treeRootY = /* Dragons, dragons, dragons */;
    double treeHeight = /* I like dragons! */;

    drawTree(window, treeRootX, treeRootY, treeHeight, 90, 8);

    return 0;
}

Tell you parameters like the Order and Initial Angle? Most unorthodox!
int main() {
    GWindow window(kWindowWidth, kWindowHeight);
    double treeRootX  = /* Here be dragons */;
    double treeRootY  = /* Dragons, dragons, dragons */;
    double treeHeight = /* I like dragons! */;

    drawTree(window, treeRootX, treeRootY, treeHeight);

    return 0;
}
```c
int main() {
    GWindow window(kWindowWidth, kWindowHeight);
    double treeRootX  = /* Here be dragons */;
    double treeRootY  = /* Dragons, dragons, dragons */;
    double treeHeight = /* I like dragons! */;
    drawTree(window, treeRootX, treeRootY, treeHeight);
    return 0;
}
```

This is more acceptable in polite company!
Wrapper Functions

• Some recursive functions need extra arguments as part of an implementation detail.
  • In our case, the order of the tree is not something we want to expose.
• A wrapper function is a function that does some initial prep work, then fires off a recursive call with the right arguments.
• We’ll use wrapper functions extensively over the next couple of lectures, and they’ll (hypothetically speaking) be something useful to know. 😊
The beautiful thing is that the distribution of the sizes of individual trees in the forest appears to exactly match the distribution of the sizes of individual branches within a single tree [...] 

[S]tudying a single tree will make it easier to predict how much carbon dioxide an entire forest can absorb.

- *Hunting the Hidden Dimension*
An Amazing Website

http://recursivedrawing.com/
Recursive Enumeration
You need to send an emergency team of doctors to an area. Each doctor has different skills (primary care, surgery, infectious disease, etc.). Many have several skills.

How do you find the smallest team you can send in that has all the essential skills covered?

1. List off every possible team of doctors you could form.
2. Pare the list down to just the teams that have all the skills you need.
3. Choose the smallest of these teams.
1. List off every possible team of doctors you could form.

2. Pare the list down to just the teams that have all the skills you need.

3. Choose the smallest of these teams.
1. List off every possible team of doctors you could form.

2. Pare the list down to just the teams that have all the skills you need.

3. Choose the smallest of these teams.
Recursive Enumeration

• Recursion is a powerful tool for enumeration: listing off all the possible ways something can be done.

• As you get more and more comfortable with recursive enumeration, you’ll find yourself able to solve a larger and larger class of problems.

• You’ll see a lot of examples of this over the next few lectures, section handouts, and programming assignments.
Subsets and Power Sets

- Mathematically, we are looking to choose a **subset** of the group of physicians.
- A subset of a group $S$ is a collection of zero or more objects chosen out of $S$.
- Sets have tons of crazy and counterintuitive properties – take CS103 for details!
- In the meantime, how might we go about listing off all the subsets of our group?
\{ A, H, I \}
\{ A, H, I \}

\{ H, I \}

\{ I \}

\{ A \}

\{ A, H \}

\{ A, I \}

\{ A, H, I \}
To generate all subsets of \{A, H, I\}:

Generate all subsets of \{H, I\}.

For each of those subsets:
   Make two copies.
   Leave one copy unmodified.
   Add A into the other.

Return what you find.
To generate all subsets of \{A, H, I\}:

Generate all subsets of \{H, I\}.

For each of those subsets:
  Make two copies.
  Leave one copy unmodified.
  Add A into the other.

Return what you find.
What’s the smallest set we can make?

**Answer:** The *empty set* of no elements!
Generating Subsets

• **Base Case:**
  • The only subset of the empty set is the empty set itself.

• **Recursive Step:**
  • Fix some element \( x \) of the set.
  • Generate all subsets of the set formed by removing \( x \) from the main set.
  • These subsets are subsets of the original set.
  • All of the sets formed by adding \( x \) into those subsets are subsets of the original set.
Tracing the Recursion
Tracing the Recursion

\{ A, H, I \}
Tracing the Recursion

\{ A, H, I \}

\{ H, I \}
Tracing the Recursion

\{ A, H, I \}

\{ H, I \}

\{ I \}
Tracing the Recursion

{ A, H, I }

{ H, I }

{ I }

{ }
Tracing the Recursion

{ A, H, I }

{ H, I }

{ I }
Tracing the Recursion

\{ A, H, I \}

\{ H, I \}

\{ I \}

\{ \{ I \}, \{ \} \}

\{ \{ \} \}
Tracing the Recursion

\{ \{ A, H, I \} \}

\{ \{ H, I \} \}

\{ \{ I \} \}

\{ \{ \} \}

\{ \{ H, I \}, \{ H \}, \{ I \}, \{ \} \}
Tracing the Recursion

{ A, H, I }  { {A, H, I}, {A, H}, {A, I}, {A}, {H, I}, {H}, {I}, { } } }

{ H, I }  { {H, I}, {H}, {I}, { } } }

{ I }  { {I}, { } }

{ }  { { } } }
Your Action Items

• **Complete Assignment 2**
  - It’s due on Monday. If you haven’t started it yet, you are behind where you need to be right now!

• **Read Chapter 8 of the Textbook**
  - There’s a ton of goodies in there! It’ll help you solidify your understanding.
Next Time

- *Exhaustive Recursion II*
  - What other structures can we generate?
  - How do we do so efficiently?
- *Recursive Backtracking*
  - How do you find a needle in a haystack?