Recursion (part 2)

Examples of Recursion
The von Koch Curve and Snowflake

Divide it into three equal parts

Replace the inner third of it with an equilateral triangle

Repeat the first two steps on all lines of the new figure

What is at the end?
What is one step?
How can that step help solve it?
Quick review of last lecture

Recursive Definitions

• Consider the following list of numbers:
  24, 88, 40, 37

• Such a list can be defined as follows:
  A LIST is a: number
  or a: number comma LIST

• That is, a LIST is defined to be a single number, or a number followed by a comma followed by a LIST

• The concept of a LIST is used to define itself

Recursive Definitions

• The recursive part of the LIST definition is used several times, terminating with the non-recursive part:

  \[
  \text{number comma LIST}
  \]

  \[
  \begin{align*}
  24 & , 88, 40, 37 \\
  88 & , 40, 37 \\
  40 & , 37 \\
  37 &
  \end{align*}
  \]

Recursive Definitions

• N!, for any positive integer N, is defined to be the product of all integers between 1 and N inclusive

• This definition can be expressed recursively as:

  \[
  1! = 1 \\
  N! = N \times (N-1)! 
  \]

• A factorial is defined in terms of another factorial

• Eventually, the base case of 1! is reached

Recursive Definitions

\[
\begin{align*}
5! & = 5 \times 4! \\
5 & \times 4! \\
4 & \times 3! \\
3 & \times 2! \\
2 & \times 1! \\
1 &
\end{align*}
\]

Recursive Execution

\[
\begin{align*}
6! & = 6 \times 5! \\
& = 6 \times 5 \times 4! \\
& = 6 \times 5 \times 4 \times 3! \\
& = 6 \times 5 \times 4 \times 3 \times 2! \\
& = 6 \times 5 \times 4 \times 3 \times 2 \times 1! \\
& = 6 \times 5 \times 4 \times 3 \times 2 \times 1 \\
& = 720
\end{align*}
\]
Recursive Programming

- Consider the problem of computing the sum of all the numbers between 1 and any positive integer N.
- This problem can be recursively defined as:
  \[
  \sum_{i=1}^{N} i = N + \sum_{i=1}^{N-1} i
  \]
  \[
  = N + (N-1) + \sum_{i=1}^{N-2} i
  \]
  \[
  = N + (N-1) + (N-2) + \sum_{i=1}^{N-3} i
  \]

Recursive Programming

// This method returns the sum of 1 to num
public int sum(int num)
{
    int result;
    if (num == 1)
        result = 1;
    else
        result = num + sum(num - 1);
    return result;
}

Recursive Programming

- Note that just because we can use recursion to solve a problem, doesn’t mean we should.
- For instance, we usually would not use recursion to solve the sum of 1 to N problem, because the iterative version is easier to understand.
- However, for some problems, recursion provides an elegant solution, often cleaner than an iterative version.
- You must carefully decide whether recursion is the correct technique for any problem.

Maze Traversal

- We can use recursion to find a path through a maze.
- From each location, we can search in each direction.
- Recursion keeps track of the path through the maze.
- The base case is an invalid move or reaching the final destination.
- See MazeSearch.java (page 583).
- See Maze.java (page 584).
The Towers of Hanoi is a puzzle made up of three vertical pegs and several disks that slide on the pegs.

- The disks are of varying size, initially placed on one peg with the largest disk on the bottom with increasingly smaller ones on top.
- The goal is to move all of the disks from one peg to another under the following rules:
  - We can move only one disk at a time.
  - We cannot move a larger disk on top of a smaller one.
Towers of Hanoi

Original Configuration

Move 1

Move 2

Move 3

Move 4

Move 5

Move 6

Move 7 (done)

Animation of the Towers of Hanoi


Fractals

• A fractal is a geometric shape made up of the same pattern repeated in different sizes and orientations
• The Koch Snowflake is a particular fractal that begins with an equilateral triangle
• To get a higher order of the fractal, the sides of the triangle are replaced with angled line segments
• See KochSnowflake.java (page 597)
• See KochPanel.java (page 600)

Koch Snowflakes

Becomes

< x1, y1 >

< x2, y2 >

< x3, y3 >

< x4, y4 >

< x5, y5 >

< x6, y6 >

< x7, y7 >
Indirect Recursion

- A method invoking itself is considered to be direct recursion
- A method could invoke another method, which invokes another, etc., until eventually the original method is invoked again
- For example, method m1 could invoke m2, which invokes m3, which in turn invokes m1 again
- This is called indirect recursion, and requires all the same care as direct recursion
- It is often more difficult to trace and debug

THE END