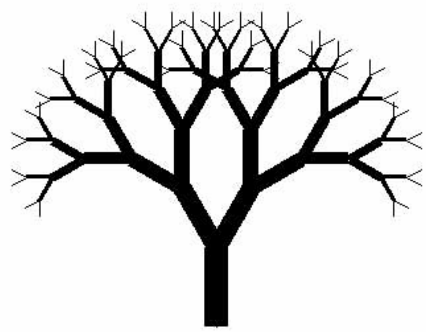
Recursion
(part 2)

October 27, 2006

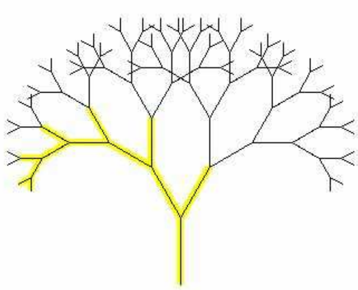
ComS 207: Programming I (in Java)
Iowa State University, FALL 2006
Instructor: Alexander Stoytchev

Examples of Recursion

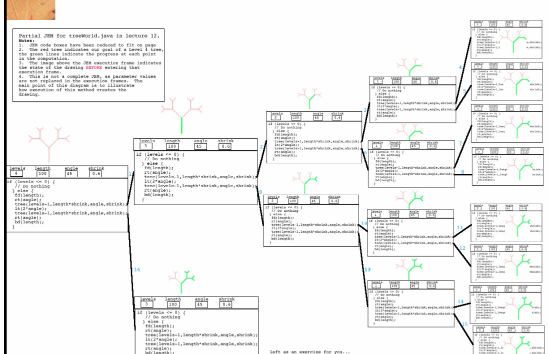
[http://www.math.ubc.ca/~jbryan/]



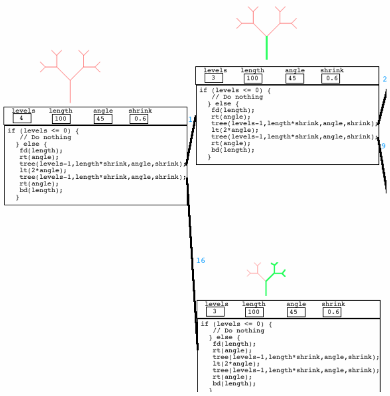
http://www.bfoit.org/Intro_to_Programming/TT_Recursion.html]



http://www.bfoit.org/Intro_to_Programming/TT_Recursion.html]

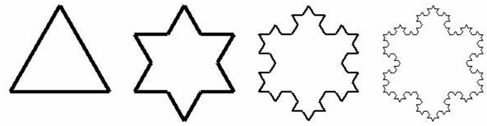


[http://cs.wellesley.edu/~cs111/spring03/unravel.gif]



[http://cs.wellesley.edu/~cs111/spring03/unravel.gif]

The von Koch Curve and Snowflake

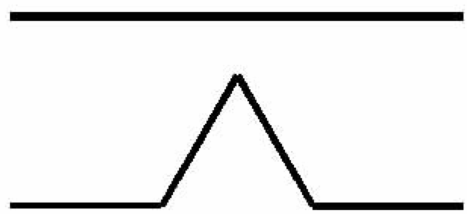


[http://www.bfoit.org/Intro_to_Programming/TT_Recursion.html]

Divide it into three equal parts

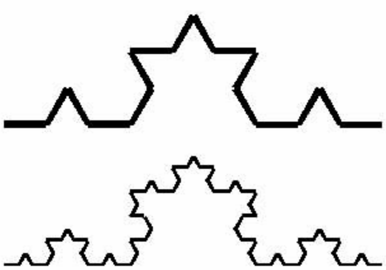
[http://www.bfoit.org/Intro_to_Programming/TT_Recursion.html]

Replace the inner third of it with an equilateral triangle

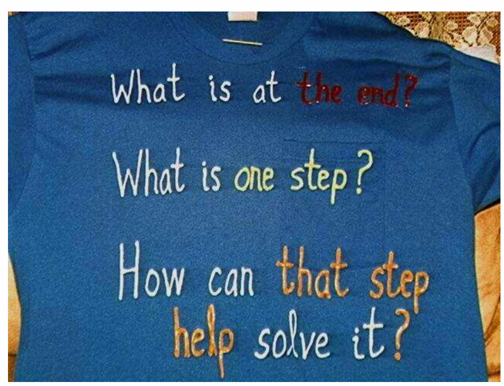


[http://www.bfoit.org/Intro_to_Programming/TT_Recursion.html]

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



[http://www.bfoit.org/Intro_to_Programming/TT_Recursion.html]



[http://www.cs.iastate.edu/~leavens/T-Shirts/227-342-recursion-front.JPG]

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:

```
number comma LIST
  24     ,   88, 40, 37

             number comma LIST
               88     ,   40, 37

                          number comma LIST
                            40     ,   37

                                       number
                                         37
```

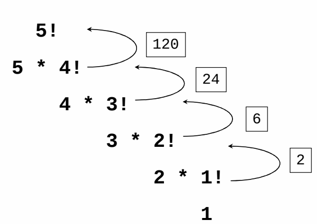

Recursive Definitions

- 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 * (N-1)!
```

- A factorial is defined in terms of another factorial
- Eventually, the base case of 1! is reached

Recursive Definitions



Recursive Execution

6!
(6 * 5!)
(6 * (5 * 4!))
(6 * (5 * (4 * 3!)))
(6 * (5 * (4 * (3 * 2!))))
(6 * (5 * (4 * (3 * (2 * 1!)))))
(6 * (5 * (4 * (3 * (2 * (1 * 0!))))))

(6 * (5 * (4 * (3 * (2 * (1 * 1))))))
(6 * (5 * (4 * (3 * (2 * 1)))))
(6 * (5 * (4 * (3 * 2))))
(6 * (5 * (4 * 6)))
(6 * (5 * 24))
(6 * 120)
720

Recursive Programming

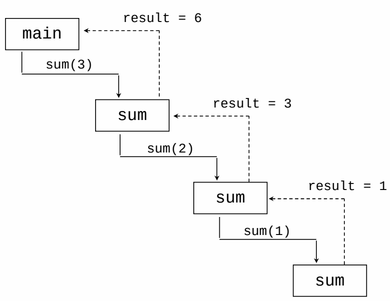
- Consider the problem of computing the sum of all the numbers between 1 and any positive integer N
- This problem can be recursively defined as:

$$\sum_{i=1}^{N} i = N + \sum_{i=1}^{N-1} i$$
$$= N + N-1 + \sum_{i=1}^{N-2} i$$
$$= N + N-1 + N-2 + \sum_{i=1}^{N-3} i$$

Recursive Programming

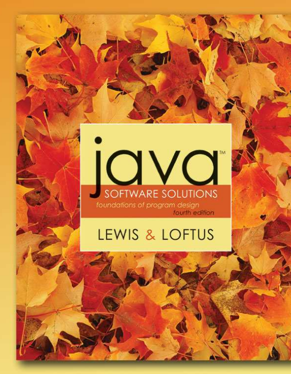
```
// This method returns the sum of 1 to num
public int sum (int num)
{
    int result;

    if (num == 1)
        result = 1;
    else
        result = num + sum (n-1);

    return result;
}
```


Recursive Programming



Chapter 11

Recursion



PEARSON Addison Wesley

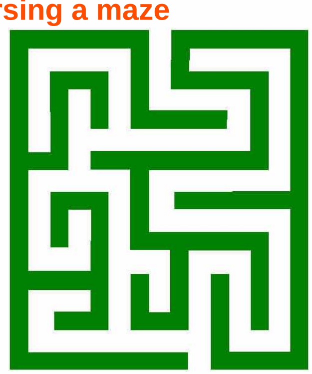
Recursive Programming

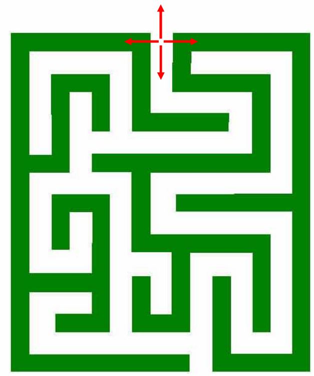
- Note that just because we can use recursion to solve a problem, doesn't mean we should
- For instance, we usually would not use recursion to solve the sum of 1 to N problem, because the iterative version is easier to understand
- However, for some problems, recursion provides an elegant solution, often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

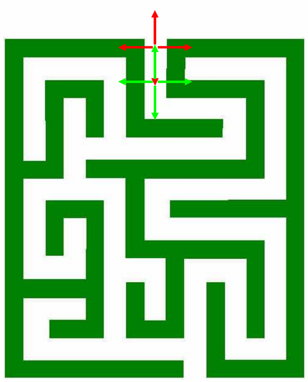
Maze Traversal

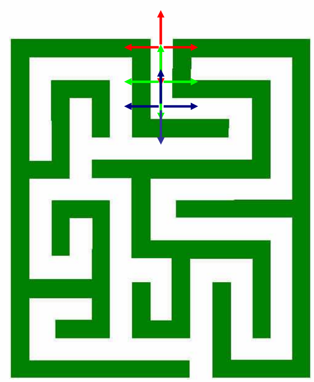
- We can use recursion to find a path through a maze
- From each location, we can search in each direction
- Recursion keeps track of the path through the maze
- The base case is an invalid move or reaching the final destination
- See `MazeSearch.java` (page 583)
- See `Maze.java` (page 584)

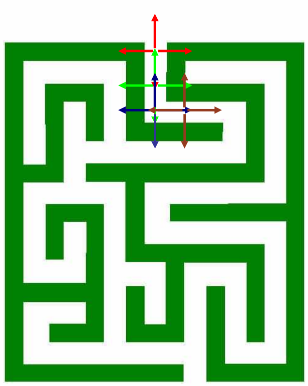
Traversing a maze







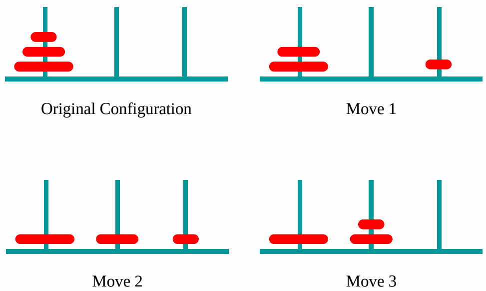




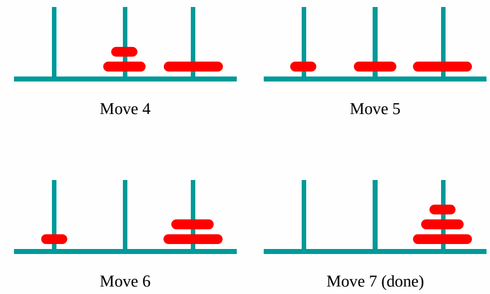
Towers of Hanoi

- **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



Towers of Hanoi



Animation of the Towers of Hanoi

http://www.cs.concordia.ca/~twang/WangApr01/RootWang.html

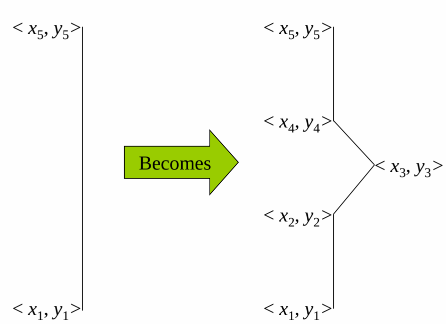
Towers of Hanoi

- **An iterative solution to the Towers of Hanoi is quite complex**
- **A recursive solution is much shorter and more elegant**
- **See `SolveTowers.java` (page 590)**
- **See `TowersOfHanoi.java` (page 591)**

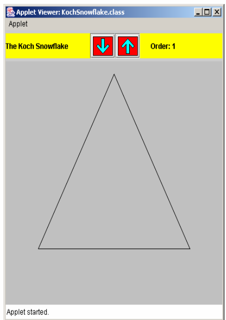
Fractals

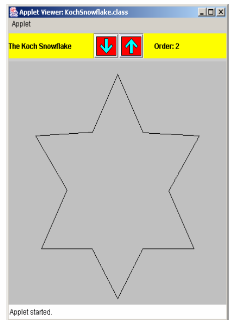
- **A *fractal* is a geometric shape made up of the same pattern repeated in different sizes and orientations**
- **The *Koch Snowflake* is a particular fractal that begins with an equilateral triangle**
- **To get a higher order of the fractal, the sides of the triangle are replaced with angled line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

Koch Snowflakes

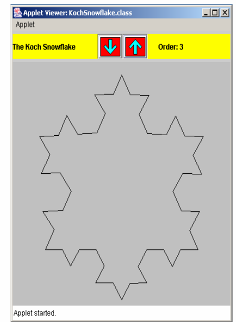


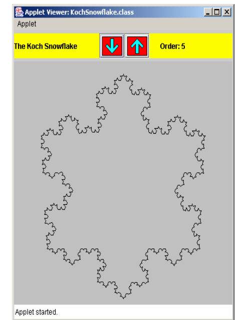
Koch Snowflakes





Koch Snowflakes

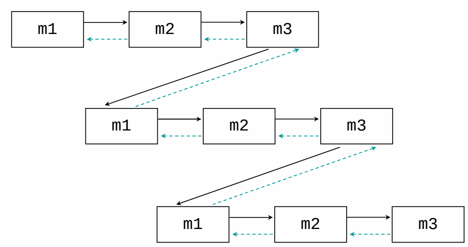




Indirect Recursion

- 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

Indirect Recursion



THE END