Chapter 8: Recursion

Presentation slides for

Java Software Solutions

for AP* Computer Science A

2nd Edition

by John Lewis, William Loftus, and Cara Cocking

Java Software Solutions is published by Addison-Wesley

Recursion

- Recursion is a fundamental programming technique that can provide elegant solutions certain kinds of problems
- Chapter 8 focuses on:
  - thinking in a recursive manner
  - programming in a recursive manner
  - the correct use of recursion
  - examples using recursion
  - recursion in sorting
  - recursion in graphics

Recursive Thinking

- *Recursion* is a programming technique in which a method can call itself to solve a problem
- A *recursive definition* is one which uses the word or concept being defined in the definition itself; when defining an English word, a recursive definition usually is not helpful
- But in other situations, a recursive definition can be an appropriate way to express a concept
- Before applying recursion to programming, it is best to practice thinking recursively

Recursive Definitions

- Consider the following list of numbers:

```
24, 88, 40, 37
```

- A list can be defined recursively

```
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, ultimately 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
```


Infinite Recursion

- All recursive definitions must have a non-recursive part
- If they don't, there is no way to terminate the recursive path
- A definition without a non-recursive part causes *infinite recursion*
- This problem is similar to an infinite loop with the definition itself causing the infinite "loop"
- The non-recursive part often is called the *base case*

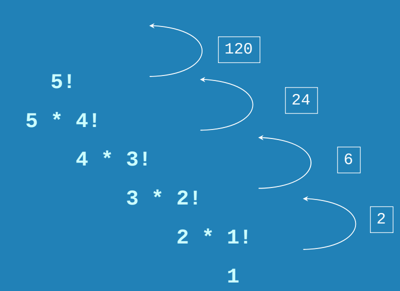
Recursive Definitions

- Mathematical formulas often are expressed recursively
- 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)!
```

- The concept of the factorial is defined in terms of another factorial until the base case of 1! is reached

Recursive Definitions



Recursive Programming

- A method in Java can invoke itself; if set up that way, it is called a *recursive method*
- The code of a recursive method must be structured to handle both the base case and the recursive case
- Each call to the method sets up a new execution environment, with new parameters and new local variables
- As always, when the method execution completes, control returns to the method that invoked it (which may be an earlier invocation of itself)

Recursive Programming

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

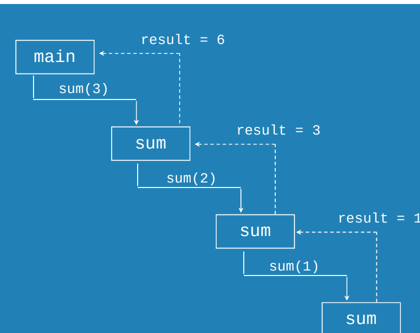
$$\sum_{i=1}^{N} = N + \sum_{i=1}^{N-1} = N + (N-1) + \sum_{i=1}^{N-2}$$

$$= \text{etc.}$$

Recursive Programming

```
public int sum (int num)
{
   int result;
   if (num == 1)
        result = 1;
   else
        result = num + sum (num - 1);
   return result;
}
```


Recursive Programming



Recursion vs. Iteration

- 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; in fact, there is a formula which is superior to both recursion and iteration!
- You must be able to determine when recursion is the correct technique to use

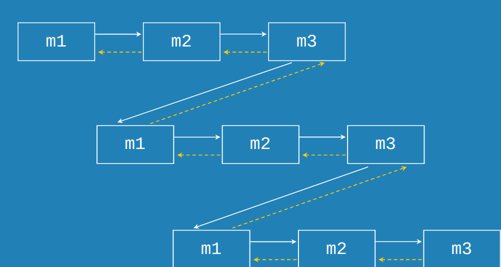
Recursion vs. Iteration

- Every recursive solution has a corresponding iterative solution
- For example, the sum (or the product) of the numbers between 1 and any positive integer N can be calculated with a for loop
- Recursion has the overhead of multiple method invocations
- Nevertheless, recursive solutions often are more simple and elegant than iterative solutions

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 until a base case is reached
- 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



Maze Traversal

- We can use recursion to find a path through a maze; a path can be found from any location if a path can be found from any of the location's neighboring locations
- At each location we encounter, we mark the location as "visited" and we attempt to find a path from that location's "unvisited" neighbors
- Recursion keeps track of the path through the maze
- The base cases are an prohibited move or arrival at the final destination

Maze Traversal

- See `MazeSearch.java` (page 473)
- See `Maze.java` (page 474)

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 disks on top
- The goal is to move all of the disks from one peg to another according to the following rules:
  - We can move only one disk at a time
  - We cannot place a larger disk on top of a smaller disk
  - All disks must be on some peg except for the disk in transit between pegs

Towers of Hanoi

- A solution to the three-disk Towers of Hanoi puzzle
- See Figures 8.5 and 8.6

Towers of Hanoi

- To move a stack of N disks from the original peg to the destination peg
  - move the topmost N - 1 disks from the original peg to the extra peg
  - move the largest disk from the original peg to the destination peg
  - move the N-1 disks from the extra peg to the destination peg
  - The base case occurs when a "stack" consists of only one disk
- This recursive solution is simple and elegant even though the number of move increases exponentially as the number of disks increases
- The iterative solution to the Towers of Hanoi is much more complex

Towers of Hanoi

- See `SolveTowers.java` (page 479)
- See `TowersOfHanoi.java` (page 480)

Recursion in Sorting

- Some sorting algorithms can be implemented recursively
- We will examine two:
  - Merge sort
  - Quick sort

Merge Sort

- Merge sort divides a list in half, recursively sorts each half, and then combines the two lists
- At the deepest level of recursion, one-element lists are reached
- A one-element list is already sorted
- The work of the sort comes in when the sorted sublists are merge together
- Merge sort has efficiency O(n log n)
- See `RecursiveSorts.java` (page 483)

Quick Sort

- Quick sort partitions a list into two sublists and recursively sorts each sublist
- Partitioning is done by selecting a pivot value
- Every element less than the pivot is moved to the left of it
- Every element greater than the pivot is moved to the right of it
- The work of the sort is in the partitioning
- Quick sort has efficiency O(n log n)
- See `RecursiveSorts.java` (page 483)

Recursion in Graphics

- Consider the task of repeatedly displaying a set of tiled images in a mosaic in which one of the tiles contains a copy of the entire collage
- The base case is reached when the area for the "remaining" tile shrinks to a certain size
- See `TiledPictures.java` (page 490)

Fractals

- A *fractal* is a geometric shape than can consist of the same pattern repeated in different scales and orientations
- The *Koch Snowflake* is a particular fractal that begins with an equilateral triangle
- To get a higher order of the fractal, the middle of each edge is replaced with two angled line segments

Fractals

- See Figure 8.9
- See `KochSnowflake.java` (page 493)
- See `KochPanel.java` (page 496)

Summary

- Chapter 8 has focused on:
  - thinking in a recursive manner
  - programming in a recursive manner
  - the correct use of recursion
  - examples using recursion
  - recursion in sorting
  - recursion in graphics