

Recursion

CSE 114: Computer Science I
SUNY at Stony Brook

Software Reuse

- ☺ Laziness is a virtue among programmers
- ☺ Often, a given task must be performed multiple times
 - ☺ Ex. loops for local repetition
- ☺ Instead of (re)writing the code each time, it is more efficient to write the code once and reuse it as necessary

Methods

- ① A function is a small block of code that can be called from another point in a program
 - ② In Java, functions are called methods
- ① Functions/methods enable reuse, and can be used to abstract out common tasks
 - ② Ex. computing the tax on a purchase
- ① Method results can be changed by supplying different input values

Method Execution

- ④ Only one method can be active at a time
- ④ If a method is invoked, the calling method is put on hold while the new method executes.
- ④ When the called method completes (returns), execution returns to the calling method
- ④ A method may call other methods. Method calls can be nested.
 - ④ Ex., A calls B, which calls C, etc.

Variables Inside Methods

- ④ Every method invocation has its own copy of that method's variables
 - ④ Thus, if A() calls A() again, there are two copies of each variable declared inside A()
 - ④ Each copy can be changed independently of the others

Factorial Revisited

```
int factorial (int value)
{
    if (value <= 1)
        return 1;
    else
        return value * factorial(value - 1);
}
```

Recursive Functions

- ④ A recursive function is one that calls itself to solve a smaller version of the original problem
- ④ Ex. $\text{factorial}(n)$ calls $\text{factorial}(n - 1)$
- ④ A solution is put on hold until the solution to the smaller problem is computed

Recursion Requirements

- ④ In reaching a solution, the problem must first solve a smaller version of itself
- ④ There must be a version of the problem that can be solved without recursion (base case)
- ④ Ex. factorial(1) has a fixed value
 - ④ A recursive solution may have more than one base case

Recursion

- ④ Some problems lend themselves to elegant recursive solutions
- ④ All recursive solutions can also be restated in iterative terms
- ④ Recursion is not as efficient as iteration
 - ④ Need for increased storage overhead
 - ④ Increased time for function calls

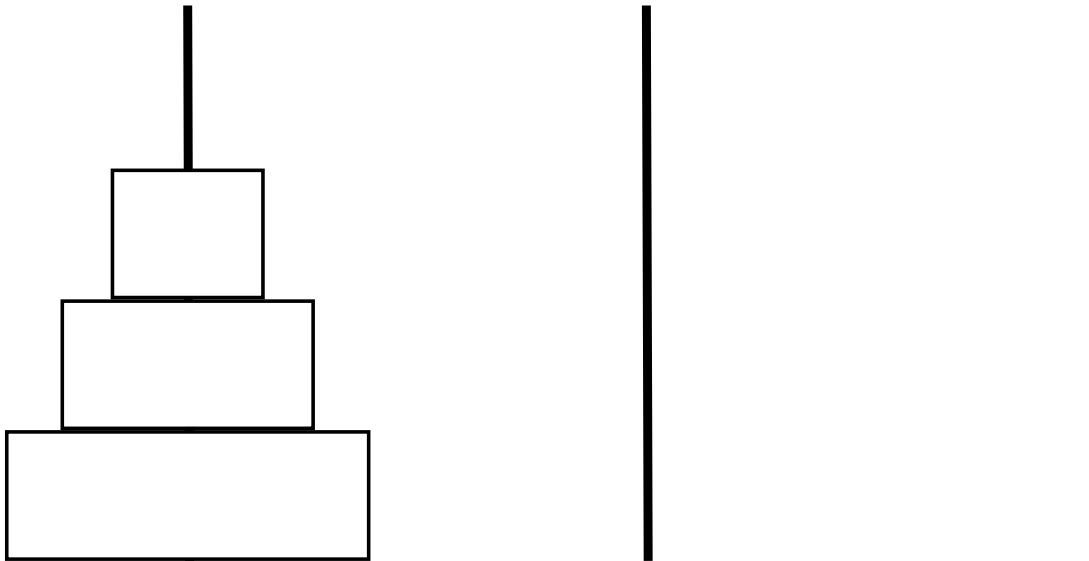
The Fibonacci Sequence

```
int fibonacci (int n)
{ // Assumption: n >= 0
    if (n == 0 || n == 1) // base case(s)
        return 1;
    else // recursive case
        return fibonacci(n-1) + fibonacci(n-2);
}
```

Seeing Stars

```
void printStars(int numStars)
{
    if (numStars > 1) // implied base case: numStars is 0
    {
        System.out.print("*");
        printStars(numStars - 1);
    }
}
```

The Towers of Hanoi



Towers of Hanoi

- ④ Given a set of discs stacked on one pole, move them to a second pole, subject to the following rules:
 - ④ Only one disc can be moved at a time
 - ④ A larger disc can never be placed on top of a smaller disc
 - ④ A third pole can be used as temporary storage

A Recursive Solution

- ④ Base case: 1 disc
 - ④ Move the disc from source to destination
- ④ Recursive case: n discs
 - ④ Move $n - 1$ discs from source to temp
 - ④ Move 1 disc from source to destination
 - ④ Move $n - 1$ discs from temp to destination

Solution Code

```
void hanoi (int n, int source, int dest, int temp)
{

if (n == 1) // base case (1 disc)
{

    System.out.print("Move 1 disc from " + source);

    System.out.println(" to " + dest);

}
```

Solution Code, part 2

```
else // recursive case
{
    hanoi (n-1, source, temp, dest);

    System.out.print("Move 1 disc from " + source)

    System.out.println(" to " + dest);

    hanoi (n-1, temp, dest, source);
} // end of else clause

} // end of method
```

Sample Exercise

Write a recursive method that takes an array of characters and an int (the index of the last array element) and prints the array in reverse order

Method header:

```
void reverse(char[] input, int index)
```

One Solution

```
if (index == 0)
    System.out.print(input[index]);
else
{
    System.out.print(input[index]);
    reverse (input, index - 1);
}
```

Solution Trace

- ⑤ reverse ("abc", 2);
 - ⑥ print 'c'
- ⑤ reverse ("abc", 1);
 - ⑥ print 'b'
- ⑤ reverse ("abc", 0);
 - ⑥ print 'a'