We derive recursive functions and look at execution of recursive calls.

CS100J 22 Feb 2006
More on Recursion

Study Sect 15.1, p. 415. Watch activity 15-2.1 on the CD. In DrJava, write and test as many of the self-review exercises as you can (disregard those that deal with arrays).

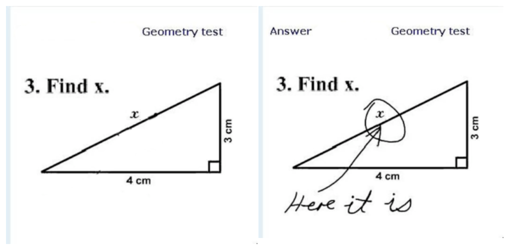
My first job was working in an orange juice factory, but I got canned: couldn't concentrate.
Then I worked in the woods as a lumberjack, but I just couldn't hack it, so they gave me the axe.
After that I tried to be a tailor, but I just wasn't suited for it. Mainly because it was a so-so job.
Next I tried working in a muffler factory but that was exhausting.
I worked as a pilot but eventually got grounded for taking off too much.
Then I tried teaching but I couldn't make the grade.

Prelim tonight in Olin 155 at 7:30 PM.

Get more of these from the course website

1

Geometry test



2

Recursive functions

/** = a copy of s in which s[0..1] are swapped, s[2..3] are swapped, s[3..4] are swapped, etc. */

public static String swapAdjacent(String s)

/** = b ^{c}. Precondition: c ≥ 0*/
public static int exp(**int** b, **int** c)

Properties:

(1) $b^c = b * b^{c-1}$

(2) For c even

$b^c = (b*b)^{c/2}$

e.g 3*3*3*3*3*3*3*3

= (3*3)*(3*3)*(3*3)*(3*3)

3

Recursive functions

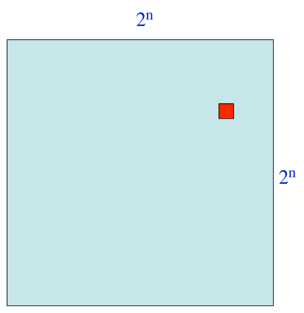
```
/** = b^c. Precondition: c ≥ 0*/
public static int exp(int b, int c) {
   if (c = 0)
      return 1;
   if (c is odd)
      return b * exp(b, c–1);
   // c is even and > 0
   return exp(b*b, c / 2);
}
```

c	number of calls
0	1
1	2
2	2
4	3
8	4
16	5
32	6
2^n	n + 1

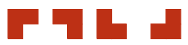
32768 is 2^{15}
so b^{32768} needs only 16 calls!

4

Tiling Elaine’s kitchen

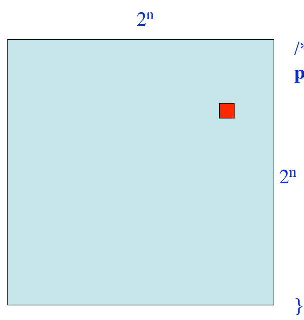


Elaine has a 2^n by 2^n kitchen. One square of it is covered by a 1 by 1 refrigerator. Tile the kitchen with these kinds of tiles:



5

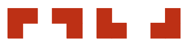
Tiling Elaine’s kitchen



```
/** tile a 2^n by 2^n kitchen. */
public static void tile(int n) {
   if (   )



}
```

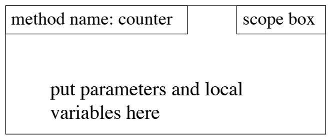


6

Executing recursive calls

Steps in executing a call:

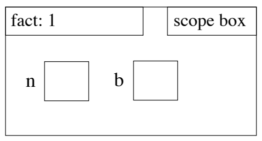
1. Draw a frame for the call, including the parameters and local variables and scope box.
2. Assign argument values to the parameters.
3. Executed the method body.
4. Erase the frame —and give value of function call to caller.



7

Executing recursive calls

```
/** = n!. Precondition: n ≥ 0 */
public static int fact(int n) {
   if (n <= 1)
     return 1;
   int b = fact(n-1);
   return n * b;
}
```



$0! = 1$. $n! = n * (n-1) * (n-2) * \ldots * 2 * 1$

1. Draw frame
2. Assign argument values to parameters
3. Execute body
4. Erase frame —and give value of function call back to caller.

8