CS1110 Lec. 4 October 2011
More on Recursion

Today we develop recursive functions and look at execution of recursive functions.

Study: Sect 15.1, p. 415.

Watch: Activity 15-2.1 on the CD.

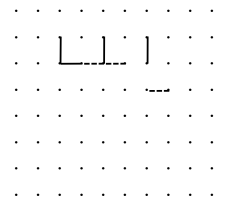
In DrJava: Write and test as many of self-review exercises as you can (disregard those that deal with arrays).

In lab today: Write many recursive functions. Ask for help! Don't waste 1 hour mulling over 1 function. Remember—you need:

0. **A good function specification**
1. **Base case(s) that are correct**
2. **Progress toward termination**
3. **Recursive case(s) that are correct**

For recursive call, think of **what** it does in terms of the function spec, not **how** execution happens

A game



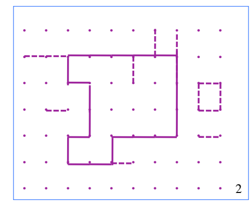
while there is room
A draws — or | ;
B draws --- or ¦ ;

A and B alternate moves

A wants to get a solid closed curve.
B wants to stop A from getting a solid closed curve.
Who can win? What strategy to use?

Board can be any size: m by n dots, with m > 0, n > 0

A won the game to the right because there is a solid closed curve.




```
/** = non-negative n, with commas every 3 digits
       e.g. commafy(5341267) = "5,341,267" */
public static String commafy(int n) {

}
```


Recursive functions

/** = b^c. Precondition: $c \geq 0$*/
public static int exp(**double** 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)

Recursive functions

/** = b^c. Precondition: $c \geq 0$*/
public static int exp(**double** b, **int** c) {

}

c	number of recursive calls
0	0
1	1
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!

Binary arithmetic

Decimal	Binary	Octal
00	00	00
01	01	01
02	10	02
03	11	03
04	100	04
05	101	05
06	110	06
07	111	07
08	1000	10
09	1001	11
10	1010	12

	Binary
$2^0 = 1$	1
$2^1 = 2$	10
$2^2 = 4$	100
$2^3 = 8$	1000
$2^4 = 16$	10000
$2^5 = 32$	100000
$2^6 = 64$	1000000
$2^{15} = 32768$	1000000000000000

Test c odd: Test last bit = 1

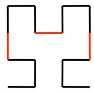
Divide c by 2: Delete the last bit

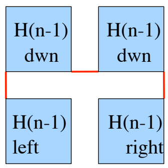
Subtract 1 when odd: Change last bit from 1 to 0.

Exponentiation algorithm processes binary rep. of the exponent.

Hilbert’s space-filling curve

Hilbert(1): 

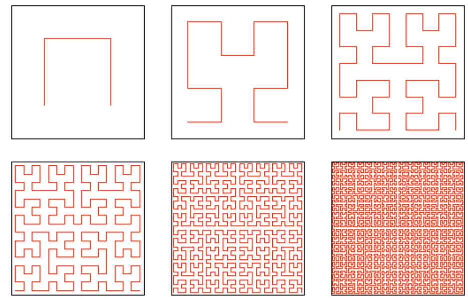
Hilbert(2): 

Hilbert(n): 

As the size of each line gets smaller and smaller, in the limit, this algorithm fills every point in space. Lines never overlap.

All methods used in today’s lecture will be on course website

Hilbert’s space-filling curve



Executing recursive function calls

```
/** = non-negative n, with commas every 3 digits
      e.g. commafy(5341267) = “5,341,267” */
public static String commafy(int n) {
   1: if (n < 1000)
         2: return “” + n;
   // n >= 1000
   3: return commafy(n/1000) + “,” + to3(n%1000);
}
/** = p with at least 3 chars —
     0’s prepended if necessary */
public static String to3(int p) {
   if (p < 10) return “00” + p;
   if (p < 100) return “0” + p;
   return “” + p;
}
```

commafy(5341266 + 1)