

**Read: pp. 403-408** CS1110 30 September. Recursion but SKIP sect. 15.1.2

**ProgramLive CD, page 15-3, has interesting recursive methods.**

**Download presented algorithms from the website**

**Prelim I 7:30-9:00PM Thursday, 7 Oct**

**Recursive definition:** A definition that is defined in terms of itself.

**Recursive method:** a method that calls itself (directly or indirectly).

Recursion is often a good alternative to iteration (loops), which we cover later. Recursion is an important programming tool. Purely functional languages have no loops —only recursion.

**Recursion:** If you get the point, stop; otherwise, see Recursion.

**Infinite recursion:** See Infinite recursion.

**== versus equals** Read pp 154-155

```

public boolean equals(Object ob) {
    return this == ob;
}

```

In this situation:  
b == c is false

**A:** d.equals(c) is false  
**B:** d.equals(c) is true  
**C:** d.equals(c) gives an error

d == c ?  
d.equals(c) ?

```

/** = "p is a Point and
it describes the same point as this Point. */
public boolean equals(Object p)

```

b == c is false      Use b == c to test whether b and c contain the name of the same object.

b.equals(c) is true

```

/** = the number of 'e's in s */
public String noe(String s) {
    if (s.length() == 0) {
        return 0;
    }
    // { s has at least one char }
    return (s[0] == 'e' ? 1 : 0) + noe(s.substring(1));
}

```

Called the *base case*

Called the *recursive case*

**Notation:**  
s[i] shorthand for s.charAt[i].  
s[i..] shorthand for s.substring(i).

Express the answer with the same terminology as the specification, but on a smaller scale:  
number of 'e's in s = (if s[0] = 'e' then 1 else 0) + number of 'e's in s[1..]

**Two issues in coming to grips with recursion**

- 1. How are recursive calls executed?**
- 2. How do we understand a recursive method and how do we create one?**

We discussed the first issue earlier. If you execute a call on a recursive method carefully, using our model of execution, you will see that it works. Briefly, a new frame is created for each recursive call. We do this in the next lecture.

**DON'T try to understand a recursive method by executing its recursive calls!** Use execution only to understand how it works.

**Understanding a recursive method**

**Step 1: HAVE A PRECISE SPECIFICATION**

```

// = number of 'e's in s
public static int noe(String s) {
    if (s.length() == 0) {
        return 0;
    }
    // {s has at least one character}
    return (s[0] == 'e' ? 1 : 0) + noe(s.substring(1));
}

```

**base case**

**recursive case (has a recursive call)**

**Step 2:** Check the base case. When s is the empty string, 0 is returned. So the base case is handled correctly.

**Notation:**  
s[i] shorthand for s.charAt[i].  
s[i..] shorthand for s.substring(i).

**Understanding a recursive function**

```
s = ""
s has at least one character
```

base case  
recursive case

**Step 3:** Recursive calls make progress toward termination.

```
// = number of 'e's in s
public static int noe(String s) {
    if (s.length() == 0) {
        return 0; // base case
    }
    // {s has at least one character}
    return (s[0] == 'e' ? 1 : 0) + noe(s.substring(1));
}
```

argument s[1..] is smaller than parameter s, so there is progress toward reaching base case 0

**Step 4:** Recursive case is correct.

parameter s  
argument s[1..]

7

**Creating a recursive method**

**Task:** Write a method that removes blanks from a String.

**0. Specification:**

```
/** = s but with its blanks removed */
public static String deblank(String s)
```

precise spec!

**1. Base case:** the smallest String s is "".

```
if (s.length() == 0)
    return s;
```

Notation:  
s[i] shorthand for s.charAt[i].  
s[i..] shorthand for s.substring(i).

**2. Other cases:** String s has at least 1 character.

```
return (s[0] == ' ' ? "" : s) + s[1..] with its blanks removed
```

8

**Creating a recursive method**

```
// = s but with its blanks removed
public static String deblank(String s) {
    if (s.length() == 0) return s;
    // {s is not empty}
    if (s[0] is a blank)
        return s[1..] with its blanks removed
    // {s is not empty and s[0] is not a blank}
    return s[0] + (s[1..] with its blanks removed);
}
```

The tasks given by the two English, blue expressions are similar to the task fulfilled by this function, but on a smaller String! Rewrite each as

```
deblank(s[1..]).
```

Notation:  
s[i] shorthand for s.charAt[i].  
s[i..] shorthand for s.substring(i).

9

```
// = s but with its blanks removed
public static String deblank(String s) {
    if (s.length() == 0)
        return s;
    // {s is not empty}
    if (s.charAt(0) is a blank)
        return deblank(s.substring(1));
    // {s is not empty and s[0] is not a blank}
    return s.charAt(0) +
        deblank(s.substring(1));
}
```

**Check the four points:**

0. Precise specification?
1. Base case: correct?
2. Recursive case: progress toward termination?
3. Recursive case: correct?

10

**Check palindrome-hood**

A String with at least two characters is a palindrome if

- (0) its first and last characters are equal and
- (1) chars between first and last form a palindrome:

have to be the same  
 e.g. AMANAPLANACANALPANAMA  
has to be a palindrome

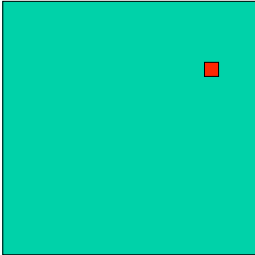
```
/** = "s is a palindrome" */
public static boolean isPal(String s) {
    if (s.length() <= 1)
        return true;
    // {s has at least two characters}
    return s.charAt(0) == s.charAt(s.length()-1) &&
        isPal(s.substring(1, s.length()-1));
}
```

11

A man, a plan, a caret, a ban, a myriad, a sum, a lac, a liar, a hoop, a pint, a catalpa, a gas, an oil, a bird, a yell, a vat, a caw, a pax, a wag, a tax, a nay, a ram, a cap, a yam, a gay, a tsar, a wall, a car, a luger, a ward, a bin, a woman, a vassal, a wolf, a tuna, a nit, a pall, a fret, a watt, a bay, a daub, a tan, a cab, a datum, a gall, a hat, a fag, a zap, a say, a jaw, a lay, a wet, a gallop, a tug, a trot, a trap, a tram, a torr, a caper, a top, a tonk, a toll, a ball, a fair, a sax, a minim, a tenor, a bass, a passer, a capital, a rut, an amen, a ted, a cabal, a tang, a sun, an ass, a maw, a sag, a jam, a dam, a sub, a salt, an axon, a sail, an ad, a wadi, a radian, a room, a rood, a rip, a tad, a pariah, a revel, a reel, a need, a pool, a plug, a pin, a peek, a parabola, a dog, a pat, a cud, a nu, a fan, a pal, a rum, a nod, an eta, a lag, an eel, a batik, a mug, a mot, a nap, a maxim, a mood, a leek, a grub, a gob, a gel, a drab, a citadel, a total, a cedar, a tap, a gag, a rat, a manor, a bar, a gal, a cola, a pap, a yaw, a tab, a raj, a gab, a nag, a pagan, a bag, a jar, a bat, a way, a papa, a local, a gar, a baron, a mat, a rag, a gap, a tar, a decal, a tot, a led, a tic, a bard, a leg, a bog, a burg, a keel, a doom, a mix, a map, an atom, a gum, a kit, a baleen, a gala, a ten, a don, a mural, a pan, a faun, a ducat, a pagoda, a lob, a rap, a keep, a nip, a gulp, a loop, a deer, a leer, a lever, a hair, a pad, a tapir, a door, a moor, an aid, a raid, a wad, an alias, an ox, an atlas, a bus, a madam, a jag, a saw, a mass, an anus, a gnat, a lab, a cadet, an em, a natural, a tip, a caress, a pass, a baronet, a minimax, a sari, a fall, a ballot, a knot, a pot, a rep, a carrot, a mart, a part, a tort, a gut, a poll, a gateway, a law, a jay, a sap, a zag, a fat, a hall, a gamut, a dab, a can, a tabu, a day, a batt, a waterfall, a patina, a nut, a flow, a lass, a van, a mow, a nib, a draw, a regular, a call, a war, a stay, a gam, a yap, a cam, a ray, an ax, a tag, a wax, a paw, a cat, a valley, a drib, a lion, a saga, a plat, a catnip, a pooh, a rail, a calamus, a dairyman, a bater, a canal ---Panama!

12


$2^n$



$2^n$

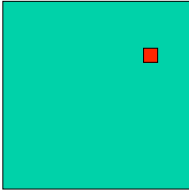
**Tiling Elaine's kitchen**

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:



13

$2^n$




$2^n$

**Tiling Elaine's kitchen**

*/\*\* tile a  $2^n$  by  $2^n$  kitchen with 1 square filled. \*/*

```
public static void tile(int n) {
    if (
        )
    }
}
```



14