

Announcements for This Lecture

<h3 style="text-align: center;">Readings</h3> <ul style="list-style-type: none"> • Read: pp. 403-408 <ul style="list-style-type: none"> ▪ but SKIP sect. 15.1.2 • ProgramLive, page 15-3 <ul style="list-style-type: none"> ▪ many recursive examples • Play with today's demos <h3 style="text-align: center;">Assignment A3</h3> <ul style="list-style-type: none"> • To be graded by Sunday 	<h3 style="text-align: center;">Prelim 1</h3> <ul style="list-style-type: none"> • Info on course web site <ul style="list-style-type: none"> ▪ Which room to go to ▪ Prelim study guide ▪ Past sample prelims • Review session Sunday <ul style="list-style-type: none"> ▪ 1:30-3:30 pm ▪ Room announced on web ▪ Run by one of your TAs
--	---

3/1/12
Recursion
1

Recursion

- **Recursive Definition:**
A definition that is defined in terms of itself
- **Recursive Method:**
A method that calls itself (directly or indirectly)

- **Recursion:** If you get the point, stop; otherwise, see Recursion
- **Infinite Recursion:** See Infinite Recursion

3/1/12
Recursion
2

A Mathematical Example: Factorial

- Non-recursive definition:

$$n! = n \times n-1 \times \dots \times 2 \times 1$$

$$= n (n-1 \times \dots \times 2 \times 1)$$
- Recursive definition:

$$n! = n (n-1)! \quad \text{for } n \geq 0 \quad \text{Recursive case}$$

$$0! = 1 \quad \text{Base case}$$

What happens if there is no base case?

3/1/12
Recursion
3

Example: Fibonacci Sequence

- Sequence of numbers: 1, 1, 2, 3, 5, 8, 13, ...

$a_0 \quad a_1 \quad a_2 \quad a_3 \quad a_4 \quad a_5 \quad a_6$

 - Get the next number by adding previous two
 - What is a_8 ?
- Recursive definition:
 - $a_n = a_{n-1} + a_{n-2}$ **Recursive Case**
 - $a_0 = 1$ **Base Case**
 - $a_1 = 1$ **(another) Base Case**

Why did we need two base cases this time?

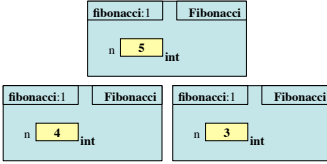
3/1/12
Recursion
4

Fibonacci as a Recursive Method

```

/** Yields: Fibonacci number a_n
 * Precondition: n >= 0 */
public static int fibonacci(int n) {
    if (n <= 1) {
        return 1;
    }
    return fibonacci(n-1)+
        fibonacci(n-2);
}
  
```

- Method that calls itself
 - Each call is new frame
 - Frames require memory
 - Infinite calls = infinite memory



3/1/12
Recursion
5

String: Two Recursive Examples

```

/** Yields: the number of characters in s. */
public static int length(String s) {
    if (s.equals("")) {
        return 0;
    }
    // { s has at least one character }
    return 1 + length(s.substring(1));
}

/** Yields: the number of 'e's in s. */
public static int numEs(String s) {
    if (s.length() == 0) {
        return 0;
    }
    // { s has at least one character }
    return (s.charAt(0) == 'e' ? 1 : 0) + numEs(s.substring(1));
}
  
```

Imagine s.length() does not exist

3/1/12
Recursion
6

How to Think About Recursive Methods

- 1. Have a precise method specification.**
- 2. Base case(s):**
 - When the parameter values are as small as possible
 - When the answer is determined with little calculation.
- 3. Recursive case(s):**
 - Recursive calls are used.
 - Verify recursive cases with the specification
- 4. Termination:**
 - Arguments of recursive calls must somehow get "smaller"
 - Each recursive call must get closer to a base case

3/1/12 Recursion 7

Understanding the String Example

```

/** Yields: the number of 'e's in s. */
public static int numEs(String s) {
    if (s.length() == 0) {
        return 0;
    }
    // { s has at least one character }
    return (s.charAt(0) == 'e' ? 1 : 0)
        + numEs(s.substring(1));
}
    
```

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

- Express using 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..]

0 1 s.length()
 s H ello World!

8

Understanding the String Example

- Step 1: Have a precise specification**

```

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

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

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

9

Understanding the String Example

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

```

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

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

argument s[1..]

- Step 4: Recursive case is correct**
 - Just check the specification

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

10

Exercise: Remove Blanks from a String

- 1. Have a precise specification**

```

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

- 2. 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)

- 3. Other Cases: String s has at least 1 character.**

```

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

3/1/12 Recursion 11

Exercise: Remove Blanks from a String

```

/** Yields: s but with 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 blanks removed
    }
    // { s is not empty and s[0] is not blank }
    return s[0] +
        (s[1..] with blanks removed);
}
    
```

- Write code in pseudocode
 - Mixture of English and code
 - Similar to top-down design
- Stuff in green looks like the method specification!
 - But on a smaller string
 - Replace with deblank(s[1..])

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

3/1/12 Recursion 12