Lecture 11

Recursion

Announcements for This Lecture

Readings

- Read: pp. 403-408
  - but SKIP sect. 15.1.2
- ProgramLive, page 15-3
  - many recursive examples
- Play with today's demos

Assignment A3

- To be graded by Sunday

Prelim 1

- Info on course web site
  - Which room to go to
  - Prelim study guide
  - Past sample prelims
- Review session Sunday
  - 1:30-3:30 pm
  - Room TBA
  - Run by one of your TAs

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

A Mathematical Example: Factorial

- Non-recursive definition:

  $n! = n \times n\text{-}1 \times \ldots \times 2 \times 1$

  $= n\ (n\text{-}1 \times \ldots \times 2 \times 1)$

- Recursive definition:

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

  $0! = 1$ **Base case**

What happens if there is no base case?

Example: Fibonnaci Sequence

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

  a_0 a_1 a_2 a_3 a_4 a_5 a_6

  - Get the next number by adding previous two
  - What is a_8?

A: $a_8 = 21$
B: $a_8 = 29$
C: $a_8 = 34$
D: None of these.

Example: Fibonnaci Sequence

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

  a_0 a_1 a_2 a_3 a_4 a_5 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?

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);
}
```

Base case(s)

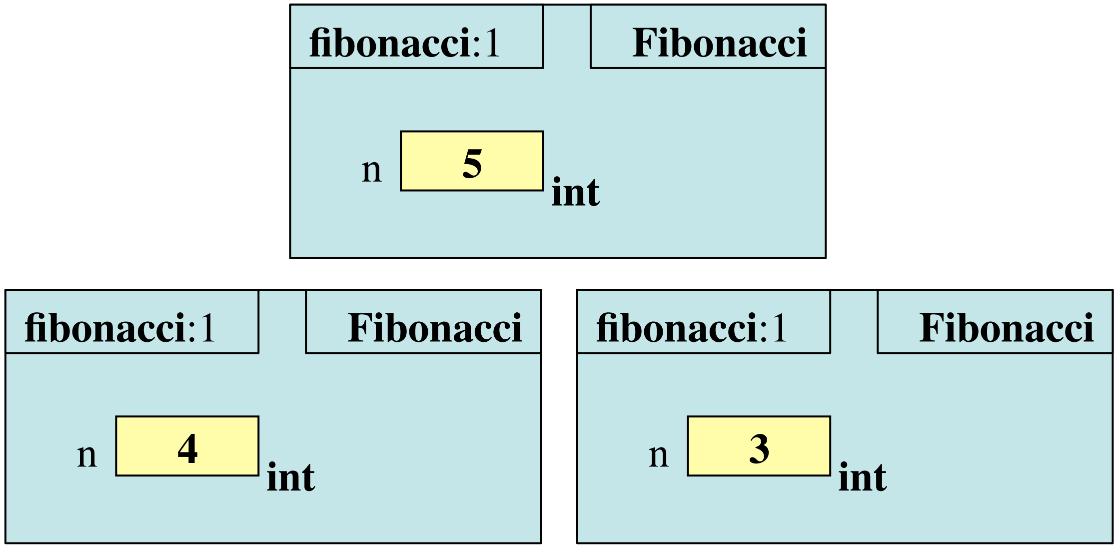
Recursive case

What happens if we forget the base cases?

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



Recursion as a Programming Tool

- Later in course, we will see iteration (loops)
- But recursion is often a good alternative
  - Particularly over lists of things
  - Examples: String, Vector<Animals>
- Some languages have no loops, only recursion
  - "Functional languages"; topic of CS 3110

A5: Recursion to draw fractal snowflakes

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));
}
```

Imagine s.length() does not exist

```
/** 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));
}
```

Two Major Issues with Recursion

- **How are recursive calls executed?**
  - We saw this with the Fibonacci example
  - Use the method frame model of execution
- **How do we understand a recursive method (and how do we create one)?**
  - You cannot use execution to understand what a recursive method does – too complicated
  - You need to rely on the **method specification**

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

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));
}
```

Base case

Recursive case

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

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..]

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));
}
```

Base case

Recursive case

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.

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)

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

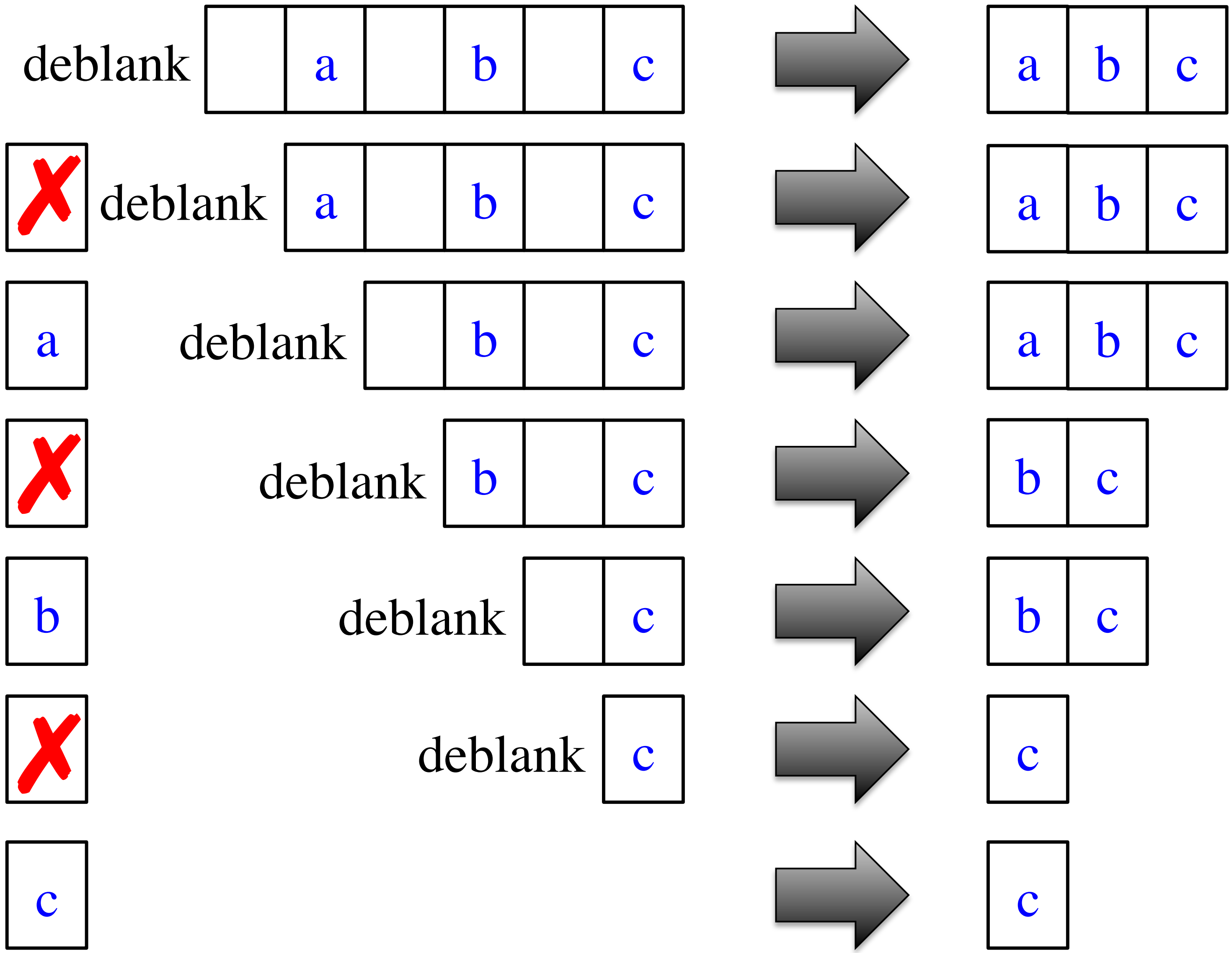
s[i] shorthand for s.charAt(i)

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)
   ```

What the Recursion Does



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)

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.charAt(0) == ' ') {
     return deblank(s.substring(1));
  }

  // {s is not empty and s[0] is not blank}
  return s.charAt(0) +
           deblank(s.substring(1));
}
```

- Check the four points:
  1. Precise specification?
  2. Base case: correct?
  3. Recursive case: progress toward termination?
  4. Recursive case: correct?

Notation

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

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

Next Time: A Lot of Examples