Slide 1

Administrivia

- Homework 6 design due today (11:59pm).

Slide 2

Recursion — Overview

- Basic approach:
  - Identify “base case” — something you can solve directly.
  - Figure out how to decompose non-base cases into “smaller” problems, and apply algorithm to smaller problems.
- How to think about “does it work?”
  - Does it work for base case(s)?
  - Assuming recursive calls work, does it work for other cases?
  - Does every recursive call get you at least one step closer to a base case?
- Implementation — conceptually (and usually in fact) involves a stack of calls-in-progress.
- Can be slower than iteration (though sometimes not), but can also be much easier to understand.

Slide 3

Recursion — Simple Examples

- Factorial function.
- Function to compute Fibonacci numbers (very slow!).

Slide 4

Recursion — Parsing an Arithmetic Expression

- “Fully parenthesized arithmetic expression” is one of two things:
  - A number n.
  - Something of the form
    $$(e\ op\ f)$$
    where e and f are expressions and op is one of the four arithmetic operators.
- How to evaluate one of these?
- Let’s write code for that . . .

Minute Essay

- Consider the following recursive function.

```
public static int mystery(int m, int n) {
    if (n == 0)
        return m;
    else
        return 1 + mystery(m, n-1);
}
```

- What does `mystery(5, 3)` return?
- Give a short description in general of what `mystery` accomplishes (not how it accomplishes it — e.g., we don't really care whether `Math.min(a, b)` uses `if` or something else, so long as it returns the smaller of `a` and `b`). Assume input `n` is non-negative, or also say what happens if `n` is negative.