CS1003/1004: Intro to CS, Spring 2004
Lecture #6: Algorithms II
Janak J Parekh
janak@cs.columbia.edu

Administrivia
- HW#2 is out
  - You *really* should start earlier for this one…
- HW#1 being graded
  - Most people seemed to do well on the programs
  - If you couldn’t do the HW#1 programming, come see me and let’s straighten it out – future homeworks will only be harder
- Questions? Feedback?
- Yet another ACM UNIX session this Wednesday (more advanced stuff), 7:30, 252 ET

Agenda
- Sidebar: good homework practices
- Continue algorithms discussion
Homework notes

- As I suggest, make sure you know what you want to do first, conceptually, before programming it
- How to debug your code?
  - First - recognize if your error is syntax or semantics
  - Learn how to understand the compiler's error messages
  - Try going through the code by hand and make sure it makes sense
  - Put debugging statements in your code
  - If you are truly stuck, go to a TA's office hours or email them a detailed bug report
  - Don't send code!

Homework notes (II)

- Commenting your code
  - I didn't require it for HW#1, but I want you to start for HW#2
  - /* ... */ and // conventions
  - What to comment?
    - Put your name and a brief description at the top of your source file
    - Put a comment before things that are non-obvious
    - Put a comment before non-obvious functions
    - Wherever else you feel appropriate
  - Look at my examples...

Review of last class

- Strategies with coming up with algorithms...
  - "Get foot in the door": try to get an intuitive grasp on the problem first, conceptually
  - Stepwise refinement: take the big picture and break into smaller pieces
  - Determine if there are any iterative structures to be implemented
  - Keep boundary conditions in mind!
Iterative structures, cont’d.

- Two more types of loop constructs
- for: useful for situations where we’re doing a loop N times
  - for(i=0; i < 10; i++) { … } runs exactly 10 times
  - Three parts: initialize, condition, increment
- for(; i < 10;) { … } == while(i < 10) { … }
- Java: can put declaration inside for loop, e.g.,
  - for(int i=0; i < 10; i++) { … }

Iterative structures, cont’d.

- do-while: almost the same as while, but it does one run first
  - do { … } while (i>1); will run how many times?
- Less used
  - Another paradigm: use the break keyword
    - Will break out of loop, sometimes useful if you find you don’t need to run through every step
  - while(true) { … break; … } is sometimes used – not usually good form

Let’s revisit our examples

1. Print out the first n numbers, and keep a running total… using a for loop
2. Print out the first n Fibonacci numbers
3. Write a function that calculates \( x^y \) (i.e., raise x to the y power)
4. Reverse a list (array) of numbers
Here’s another way to look at repetition
- \( \text{fib}(n) = \text{fib}(n-1) + \text{fib}(n-2) \), right?
- We can actually encode that in a computer
  - **Recursion**: Define a solution in terms of a smaller version of itself
  - Must have **stopping** (base) case(s)
  - What’s the base case for the above recursion?
  - How about doing \( x^y \) using recursion?

Another recursive example
- Binary search: works for a sorted list of information
  - Basic idea: pick the middle element
    - If that’s what we’re looking for, done
    - If it’s larger, recursively search the “top half”
    - Otherwise, recursively search the “bottom half”
    - If we’re stuck with an empty list, we failed

HW#2
- Asks you to check a *palindrome*
- I’m not going to do the homework for you, but let’s think, conceptually, what needs to be done…
Next time

- Finish up intro to algorithms