CS1003/1004: Intro to CS, Spring 2004

Lecture #1: Introduction

Janak J Parekh
janak@cs.columbia.edu

What is this class?

- An introduction to Computer Science
- Two *required* components:
  - Weekly lecture covering the *theory* behind CS, common to both languages
  - Weekly lab covering a programming language, different one for each language
  - "Guinea pig" format
- Prerequisites: basic computer skills
- Which language is "better"?

Basic information

- Instructor: Janak J Parekh (janak@cs.columbia.edu)
  - Call me Janak, please
  - 9th year at Columbia (in various capacities)
  - OH: to be finalized once we get all our TAs
- Class website: http://www.cs.columbia.edu/~janak/cs10034
  - Make sure to check it regularly
  - Still setting up webboard and other sections…

Lab information

- C lab taught by TAs Suhit Gupta (suhit@cs.columbia.edu) and Java lab by Maryam Kamvar (mkamvar@cs.columbia.edu)
- Please register by end-of-week if possible
  - Difficulty in scheduling labs: who has a problem?
- Exception for this first week *only*: **no labs this week**
  - Instead, UNIX tutorial in this room this Thursday, 11-12:15pm

Textbooks

- Multiple textbooks
  - Brookshear, "Computer Science: An Overview", 7th Ed. required for theory
  - Oualline, "Practical C Programming", 3rd Ed., required for C lab
  - Bishop, "Java Gently", 3rd Ed., required for Java lab
  - Everyone must buy two textbooks (sorry!)
- Books can be obtained from Papyrus, SW 114th & Broadway; Amazon links & ISBN on website

Course structure

- 6 homeworks, 25 points each = 150 points
  - Roughly every 2 weeks
- 50 point midterm, 100-point final (open-book)
- Class participation (see next slide)
- In other words, homeworks are most important component of class
  - Learning programming is useless unless you actually do it hands-on

Class participation and attendance

- Attendance is expected; participation is beneficial
  - I won't take attendance, but the TAs might informally
  - Participation is useful for your grade at the end of the semester…
- If you miss class and/or lab, you're expected to catch up
  - I'll post slides and reading assignments to the schedule page to help

Homeworks

- Will consist of written and programming parts
  - Programming part will be submitted online
  - Programming to be done on CUNIX (or at least tested there)
- Late policy: you are given 3 grace days during the semester
  - A late day is exactly 24 hours
  - Can use up to two on any individual homework
  - After late days used up, late submissions will *not be accepted*

Homework 0

- It's up
- Basically, get your CUNIX account and make sure you can log into it
  - See if you can compile code
- Not to be submitted
- Thursday tutorial will cover most of these topics

Cheating

- Plagiarism and cheating: unacceptable
  - You're expected to do homeworks *by yourself*
  - Rest assured I have electronic tools to catch plagiarizers
    - I had five students last semester
    - Renaming variables, etc. doesn't help
- Results: instant zero on assignment, likely referral to dean
  - Columbia takes dishonesty *very seriously*
  - I'd much rather you come to me or the TAs for help

Feedback

- This is a "guinea-pig" course: I'm open to suggestions
- I can't promise I'll make your dreams come true, but I will take any constructive feedback seriously
  - Not just template-speak: ask my students from last semester
- I'm here to help you succeed!

Poll time!

- School
  - CC: 6
  - SEAS: 60
  - GS: 4
  - Other: 7
- Year
  - Freshman: 15
  - Sophomore: 15
  - Junior: 5
  - Senior: 7
  - Masters or later: 6

Poll (II)

- Have you programmed before?
  - No: 50
  - Yes (BASIC, VB): 6
  - Yes (C, C++, C#, Java): 4
- Have you used…
  - UNIX: 7
  - Windows command prompt: 10
- You're taking this class…
  - Because you want to: 15
  - Because you have to: 40

What is Computer Science?

What is Computer Science?

- We ask Google:
  http://www.google.com/search?q=define:Computer+Science
- I like this one best: "The systematic study of algorithmic processes that describe and transform information: their theory, analysis, design, efficiency, implementation, and application."
  - "Information age": we're presented with tons of information, and need tools to help organize it and manipulate it.

Who cares?

- "I'm taking this class because I have to know how to write code."
- "I'm taking this class because my advisor said I have to and I need an A."
- Several reasons:
  - Rising importance of computers in the world (and for your job)
  - A good coder does *not* necessarily make a good programmer or good computer scientist
  - Learning a programming language doesn't necessarily make a good coder
  - Brainteasers…

So what are we going to do?

- Study *algorithms*
  - An algorithm is a "set of steps that defines how a task is performed"
  - Not necessarily as intuitive as you may think
- Study *programs/software*
  - A program is machine-compatible representation of an algorithm, written in a *programming language*
- Study (the basics of) *hardware:* how does the software run?

Abstraction

- While we're studying all this, maintain the fundamental principle of abstraction
- What is abstraction?
  - http://www.google.com/search?q=define:abstraction
  - "Abstraction means ignoring many details in order to focus on the most important elements of a problem."
  - At any given time, we focus on one aspect of a problem, and abstract away the details of others
  - Lets us build a "big picture" of Computer Science, brick by brick

Topics we'll cover

- We'll start with the basics you need to start programming: language basics, algorithm design
- Then, we'll take a bottom-up approach to the computer
  - How is information stored in hardware?
  - How is information manipulated in hardware?
  - How do you tell the hardware to manipulate information?
  - How do you run this software in a reasonable fashion on a hardware?
- Finally, we'll look at some interesting directions for Computer Science
  - AI: the "future"?
  - Computation theory: what makes a computer a computer from a theoretical perspective?

And in the labs…

- A pragmatic approach to learning the programming language of your choice
- I'll work hard to synchronize the two parts of the class, although they won't always cover the same topics
  - You're *not* going to write an operating system!

Let's start thinking…

- You've got a five quart jug, a three quart jug, and a lake. How do you come up with exactly a gallon of water?
  - This is (was?) a brainteaser asked at Microsoft interviews

How to get a quart

- I'll model this as (x,y) where x == # of quarts in five-quart jug, y == # of quarts in three-gallon jug
  - 1. Fill three: (0, 3)
  - 2. Move three to five: (3, 0)
  - 3. Fill three: (3, 3)
  - 4. Move (as much as possible) three to five: (5, 1)
  - 5. Dump five: (0, 1)
  - 6. Move three to five: (1, 0)
  - 7. Fill three: (1, 3)
  - 8. Move three to five: (4, 0)

Something more pragmatic, perhaps?

- Given a map of the NYC subway system, design an algorithm that finds the "optimal route" between two stations
  - OK, this is not *that* easy, and you're *not* going to know enough to do this in this class
  - But we can think about it conceptually: got any ideas?
  - http://www.mta.info/nyct/maps/submap.htm

OK, how about something simpler?

- Given 10 numbers, sort them
  - Easy, you say?
  - Sort 100 numbers
  - Sort 1,000 numbers
  - Do it fast

Being a good programmer…

- Takes more than knowing how to write code
- It takes the ability to take a problem and break it down into small enough steps to write code that solves it
- It takes the ability of knowing enough of the field (and the language) to know what a "step" is
- Hopefully, that's what you'll learn this Spring

Before we go any further…

- Let me prove that I, unlike most professors, know how to program
  - All of us know C and Java, so don't hesitate to ask for help
- First program: always "Hello, world!"
- We'll go through the details next week…
- I'll put this code up; try running it for HW#0

Next class

- **NO LAB THIS WEEK!**
- Next class will be on Thursday, 1/20, 11am-12:15pm
  - UNIX tutorial