CS1003: Intro to CS, Summer 2008

Lecture #01
Introduction

Instructor: Arezu Moghadam
arezu@cs.columbia.edu

Basic information

- Instructor: Arezu Moghadam
  - Call me ‘Arezu’ please...
  - arezu@cs.columbia.edu
  - Office hours: Mondays 5:10~6:10
  - TA: Abe Skolnik

- Class website:
  - www.cs.columbia.edu/~arezu/cs1003
  - please check it regularly
    - Assignments, solutions, lecture notes & etc.
    - In progress...
What is this class?

- An introduction to computer science
- Two required components
  - Weekly lectures covering the theory behind CS
  - Weekly labs covering C programming language
- Prerequisites: basic computer skills
- Which programming language is “better”?  
  - Why C?

Lab information

- Taught by TA, Abe Skolnik
  - as2635@columbia.edu
- Lab participation is strongly encouraged!
- Each class is divided in two halves.
  - The second half is the lab.
- Except this week only: No lab
- Instead UNIX tutorial
Textbooks

- Multiple textbooks;
  - Brookshear, “Computer Science: An Overview”, 10th Ed. required for theory
  - Oualline, “Practical C Programming”, 3rd Ed., required for C lab
  - Everyone must buy two textbooks (sorry!)
- Books can be obtained from Columbia bookstore, 115th & Broadway; Amazon links & ISBN on website

Course structure

- 5 homeworks, 30 points each = 150 points
  - Roughly every week
- 50 point midterm, 100-point final
- 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 TA 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

Assignments

- Will consist of written and programming parts
  - Programming part will be submitted online
  - Programming to be done on CUNIX (or at least tested there)
  - Before submitting your code make sure it can be compiled successfully.
- 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
- UNIX tutorial to be given in the second half will cover most of these topics

Cheating

- Plagiarism and cheating: unacceptable
  - You’re expected to do assignments 100% by yourself
  - Please be assured we have electronic tools to catch plagiarizers
    - Renaming variables, etc. doesn’t help
- Results: instant zero on assignment, likely referral to dean
  - Columbia takes dishonesty extremely seriously
  - I’d much rather you come to me or the TA for help
Feedback

- I’m totally open to any suggestion
- I can’t promise I’ll make your dreams come true, but I will take any constructive feedback seriously
  - Not just template-speak
- I’m here to help you succeed!

Poll time!

- School/company
- Major
- Year/level
- Have you programmed before?
  - No
  - Yes (BASIC, VB)
  - Yes (C, C++, C#, Java)
- Have you used...
  - UNIX
  - Windows command prompt
- You’re taking this class...
  - Because you want to
  - Because you have to
What is computer science?

Let’s ask Wikipedia:

“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.
What is a computer?

- An electronic machine that can process information according to a set of instructions (*an algorithm*).
- *Algorithm*: a finite sequence of steps that solves a problem.

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.”
- “I can learn to code myself, so why do I need this class?”
- 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
  - How to solve a problem is important
  - How to think
  - Learning how to program doesn’t necessarily make a good coder
So what are we going to do?

- **Study algorithms – how to think?**
  - 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 – how to implement our thought?**
  - A program is machine-compatible representation of an algorithm, written in a *programming language*

- **Study (the basics of) hardware: how does the hardware understand the software?**

Abstraction

- **While we’re studying all this, maintain the fundamental principle of abstraction**

- **What is 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
  - How to layer our thinking and problem solving
  - 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 lab…

- 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!
Algorithm = structured thinking

- So 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?
  - 1 liquid quart = 1/4 liquid gallons

How to get one quart?

- We’ll model this as (x, y) pair where x == # of quarts in five-quart jug, y == # of quarts in three-quart jug
- Input: (x, y)
- Output: gallon of water
- Algorithm:
  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)
Another example, Shortest path

- The map of the NYC subway system is given
- There are multiple routes to travel between two destinations
- But 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](http://www.mta.info/nyct/maps/submap.htm)

Something simpler – sorting

- Given 10 numbers, sort them
  - Easy, you say?
  - Sort 100 numbers
  - Sort 1,000 numbers
  - Do it fast
How to successfully solve a problem?

- Specify problem requirements
- Analyze the problem
- Design the algorithm
- Implement the algorithm
- Test and verify
- Maintain and update code

How to become 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 Summer
Before we go any further…

- First program: always “Hello, world!”
- We’ll go through the details...
- I’ve put this code up; try running it for HW#0