CS 2401: Elementary Data Structures and Algorithms

Chris Kiekintveld

Email: cdkiekintveld@utep.edu
Office: COMP 207
Office Hours: MW 2:00-3:00 and by appointment

http://www.cs.utep.edu/vladik/cs2401.11a
Early "computers" at work, summer 1949. In the terminology of that period, computers were employees--typically female--who performed the arduous task of transcribing raw data from rolls of celluloid film and strips of oscillograph paper and then, using slide rules and electric calculators, reducing it to standard engineering units.

Source: Dryden Flight Research Center Photo Collection
Discussion Questions

- What is computer science?
- What got you interested in studying CS?
- List at least 5 exciting things you can do with a CS degree.
- What do you want to learn as a CS student?
What this course is about

- Learning to program (build on CS1)
  - Specific skills (e.g., Java)
  - General skills (e.g., debugging)
- Abstract problem solving and analysis
  - Break down complex problems
  - Design and analyze possible solutions (algorithms)
  - Understand tradeoffs in programming choices
- **Memorization is not enough**
Developing a toolkit

♦ Home Builder
  ♦ Architectural blueprint
  ♦ Wood, stone, nails, screws, pipes, shingles…
  ♦ Hammer, jigsaw, band saw, screwdriver, ratchet…

♦ Programmer
  ♦ Array, list, tree, hash table, stack…
  ♦ Binary search, quicksort, pattern matching…
  ♦ Information hiding, object-oriented design, recursion, design patterns…
Topic Areas

- **Data structures**
  - How is information organized?
  - How can it be accessed?

- **Algorithms**
  - Answer questions about data
  - Generate new data

- **Complexity Analysis**
  - How much time does an algorithm take?
  - How much space (computer memory)?
Example: Phone Book

- What information is stored?
- How long does it take to look up a number, given a name?
- How long to look up a name, given a phone number?
  - Best case?
  - Worst case?
  - Average case?
- Can I tell how far away I am from a business using this information?
Success in this course

♦ Put in the time: 8-12 hrs per week outside of class
  ♦ Read the textbook
  ♦ Work on assigned labs, homework
  ♦ Additional practice problems, studying
♦ Labs are critical!
Success in this course

♦ Need to learn a different way of thinking
  ♦ Problem solving; applying tools to new problems
  ♦ Breaking down complex problems into simpler ones
  ♦ Meticulous, methodical thinking
  ♦ Abstraction/generalization

♦ Don’t panic!
  ♦ We will give you new problems you haven’t seen
  ♦ Step back and work through it; apply your tools
Learning to program

- Practice is key
  - Learn by doing, not just through lectures
  - 10,000 hours to expert performance

- Expert programmers
  - Know specific languages/technologies well
  - Have a solid understanding of concepts
  - Can solve novel problems by applying concepts
  - Can quickly pick up new languages, etc.
First: Understand the Problem

- You can’t write a program if you don’t know what it is supposed to do!

- What are the inputs? What is known?
- What should the output be?
- Your task: figure out how to get from inputs to output

- Don’t expect to be able to sit down and write the code immediately
Techniques

- Work through examples
  - Start with simple cases
  - Make up your own examples
  - What steps did you need to do? Generalize!
- Draw figures/diagrams
  - How is data organized?
  - What is the flow of control?
- Write pseudocode in increasing detail
- Look up classes/methods you haven’t seen (or don’t remember)
Debugging

- This is a fact of life, but it does get easier with practice
- Try not to write your entire program and test it all at once
- Test small pieces of code or methods as you write them, in isolation from the main program
- Try using new built-in classes or libraries in a test program first so you know how they behave
Debugging

♦ General procedure
  ♦ Identify a specific behavior that doesn’t match what you expect (i.e., a bug)
  ♦ Write a test case that reproduces the bug
  ♦ TRACE!
  ♦ Narrow down the exact line in the code where the program deviates from what you expect
    ♦ Intuition/experience
    ♦ SOP or other logging methods
    ♦ Debuggers