The First Year
Beyond Language Issues

David G. Kay

University of California
Irvine, California

kay@uci.edu

Departmental goals

(interdependent, not strictly prioritized)

- Do good research and publish it
- Get extramural funding for research programs
- Increase national reputation
- Provide a creditable (if not excellent) undergraduate program
- Fulfill obligations to the rest of the campus (e.g., courses for non-majors)
- Maintain/improve working environment for faculty

Goals of the undergraduate program

- Provide well-trained graduates to industry
  - Technical knowledge and skills
  - Ability to adapt to changes
  - Ability to communicate and work with others
- Provide well-schooled candidates to graduate schools
- Produce well-educated citizen decision-makers
- Attract good students to campus, major
- Increase number of students served
- Operate within constraints on budgets and faculty time

Goals of a first-year course for CS majors — Students should:

- Be prepared for further course work
- Learn fundamental CS principles *
- Know basic computing concepts **
- Know how to use computing tools
  - Word processing, spreadsheets
  - Net tools: Email, news, WWW
- Develop programming skills ***
- Decide whether CS is the field for them
- Receive placement or credit for AP CS

Fundamental CS principles

- Algorithms and data structures
- Analysis — design — tradeoffs
- Abstraction and information hiding
- Breadth of the field of CS

Basic computing concepts

- Data representation (bits and bytes)
- Basic computer architecture
- Secondary storage; peripherals
- Software engineering
- Social impact

Programming skills

- Programming "in the small"—single routines
  - Algorithms: maximum, linear search, sort, …
  - Data structures: Array/table, queue, tree, …
- Abstraction and information hiding
  - Data and procedural abstraction
  - Distinguishing interface from implementation
  - Objects/classes/ADTs
  - Combining routines, using predefined "API"s
  - Programming paradigms (especially OOP and functional)
- Reading, understanding, debugging, and modifying code
- Analyzing code's performance and correctness; mathematical tools
- Adaptability to new concepts, tools, skills, paradigms, and languages
- Mechanics of entering and running code in some environment