Course wrap-up

csc148, Introduction to Computer Science
Diane Horton
Winter 2015

Administrative stuff

- We will post when A3 results are available.
- We will contact you about any outstanding remarks etc.
- Pre-exam office hours will be posted.
- I will also hold some "exam review" problem-solving sessions in a large classroom.

- Aside: Please complete your course evaluations. We want your feedback!

Preparing for the final

- Re-solve parts of the assignments that a partner did.
- Re-do / complete lab exercises.
- Edit and run examples from class.
- Test out your "what if" questions / theories.
- Make up your own problems.
- Focus on topics you aren't fully confident in.
- Solve old tests and finals.
- Come to the exam review sessions.

The final

- Comprehensive (covers the whole term), including:
  - class design with inheritance
  - ADTs, stack, queue
  - trees, linked lists
  - recursion
  - Python-specific techniques such as list comprehensions, ternary if, propery
  - Time efficiency and big-oh

The final

- No aids allowed this time.
- But you don't need to memorize function/ method APIs.
  - We'll provide them and/or be forgiving when marking those details.
- And there is very little else to memorize. It's about understanding, not regurgitation.
- Helpers are always welcome.
- Comments are not necessary unless we say otherwise.

The final

- Prior to the exam, will will post the front page, so you can see the marks breakdown.

What have we learned?

Theme: abstraction

- If a function provides a consistent interface
  - client can ignore implementation details and think of the function abstractly
  - implementation details can be changed with no impact on client code
- The same holds for classes
- ADTs take abstraction up a notch
  - above the level of any implementation or language
  - we saw stack, queue, graph (only briefly discussed)

Theme: good design

- Good design is challenging!
- We can use inheritance to capture what several classes have in common.
    - Specifies what any future descendant class must do.
    - Specifies what every descendant class has.
    - Allows client code to call methods on an object without knowing which kind it is.
- Allows plug-out plug-in compatability.
- If the design is poor, you risk having to change the interface later, and lose all of this.

Theme: algorithms

- Recursion: a whole new way of thinking.
- Some algorithms require significant reasoning to invent / understand / assess for correctness.
- Assertions can help us express that reasoning.

Theme: analysis of algorithms and code

- Some algorithms are *much* faster than others.
- Some algorithms are so slow they are infeasible.
- Big-oh helps us express such things.
- Sometimes the problem itself is infeasible.

Theme: trade-offs

- Speed vs accuracy
  - E.g., full minimax vs myopic
- Time vs space
  - E.g., saving the game state tree costs space, but makes Option B functions faster.
- Time vs time
  - E.g., sorted lists require slower inserts and deletes, but have faster search.
  - E.g., balanced search trees require slower inserts and deletes but have faster search.

Theme: gaining confidence in our code

- doctest examples are very important
- but we need to design thorough test cases
- for complex code like minimax, choosing good test cases is not easy
- tools like unittest help implement testing
- assertions and proofs are helpful for very tricky algorithms

Building your set of tools

- You've begun to build a set of tools for how to think about:
    - data
    - algorithms
    - code design
    - analysis of correctness
    - analysis of efficiency
- Knowing when and how to use these tools is what makes you a computer scientist.
- The journey is just beginning!

fin