Announcements:

- Quiz #2 back now or in section
  - Many people had trouble
  - Surprisingly many students didn't take it
  - Good prep for the prelim
- PS3 due Thursday 10/6 11:59PM
- Prelim #1 Tue 10/4 in Goldwin Smith Hall G64 7:30PM-9:00PM
  - Returned in section
  - Coverage through today's lecture
  - Review in section on Monday
  - Be sure to talk to me if you have a conflict
    - No makeup, but can take the exam at 6PM instead.
- What's on the prelim?
  - Eval
  - Zardoz
  - Dijkstra
  - Induction
  - Fold
  - Simple problems with modules and signature
- You will get (this weekend, probably):
  - A previous exam
    - Warning: coverage changes slightly year-to-year
  - A "full credit" induction proof

- You've seen modules, structures, signatures
    - The support in OCaml for abstraction, for hiding implementation details, and for namespace management
    - Other languages have similar tools, some better, some worse

- Today we are going to talk about a small abstraction example, give a more interesting example, then return to the general issue.

- Small example:
  - Graphics package in C, or a similar language
  - Points package for 2D points
  - TA implements, Ramin uses

- Spec contains the obvious functions:
  - Predicate (is this a point?)
  - Constructor (make a point from x y coordinates)
  - Accessors (get the x or y coordinates from a point)
  - Invariants, like point_x(make_point(*x*,*y*)) = *x*, point_y(make_point(*x*,*y*))=*y*

- TA will create an implementation of the spec

- The user (Ramin) is only supposed to rely on the documented properties of the implementation
  - And NOT on the accidental ones

- In real life, this often goes awry and results in a catastrophic software failure
  - such as Windows

- Let's suppose that TA's initial implementation is the obvious one,
    - she uses 2 consecutive memory locations to represent the X and Y coordinates

- In a language like C, this information "leaks" very easily
    - Partly because it is not strongly typed
    - OCaml prevents this (hard to make a good example in ML!)

- Ramin can take advantage of TA's implementation to do "clever" things
    - For example, suppose he has a bunch of points in an array and wants to check if any lie on an axis (X or Y)
        - Just multiply them all together, check for 0
    - Or to determine quadrant, multiply locations and check the sign
        - Without calling point_x or point_y

- This can be genuinely faster, in terms of execution time, and sometimes this really matters
- But usually it leads to a disaster
    - Premature emphasis on performance!

- TA changes her implementation, for example by using polar coordinates, or switching X and Y
    - Or adding a “magic number” to the beginning of each point, as a primitive form of type checking
    - After all, her default implementation of point is too weak
        - any pair of numbers is a point!
- Why do this? For example, determining that a pair of points lie along a line through the origin is trivial in polar coordinates, and perhaps TA wants to add this to the spec since she can support it efficiently

- A bigger disaster can occur if the change TA makes is more subtle
- Suppose she keeps most points in (x,y) order, but some in (y,x)? Or suppose that on some computers they are in (x,y) and some they are in (y,x)?
    - “big-endian” vs “small-endian” hardware
- Ramin’s code might not break, but his users (or his users users) might suddenly become flaky
- Tracking down this kind of bug is a total nightmare

- Examples: In Windows 3.0, a message was sent to a window if it was being resized or moved that its size was potentially changing.
    - In Windows 3.1, they fixed this so the message was only sent on resizing, not on moving
    - But the most popular application (Lotus 123) depended on getting this message even on a move, so it broke
    - See: “Make Compatible” on Wikipedia
- It is well-known within Microsoft that certain system calls would try to figure out what program was calling them (by looking up the call stack) and change their behavior to try to do what was expected
- Maintaining compatibility is hard is a huge, painful effort
- It’s no wonder that Intel and Microsoft are huge, and sometimes lumbering, companies!

- Typically failures come from:
    - Eagerness to get to work writing code
    - Too early focus on performance
        - There is no Moore’s law for programmers!
    - Inexperience with writing code big enough to need specs

- Spec design itself is something of an art form
- Remove any un-needed details, only describe what the implementation is required to do
    - What it does versus how it does it
    - Leave implementor (TA) freedom to improve and innovate!
    - Potentially might specify resources needed (time, space)
        - This is a somewhat controversial point, but there are situations where it is clearly needed
        - Such as most data structures (consider finite maps)
- You want these to be designed by both sides
    - Just like a good business contract

- You need a spec to decouple the user of (something) from the builder of (something)
    - User and builder agree on the spec
    - User can freely assume the implementation meets the spec
    - Builder can implement the spec as they see fit!
- Good examples of complex specs: standards
    - 802.11b/bluetooth, HTML/XML, MP3/JPEG, TCP/IP
        - Also: electricity, construction, drugs, DICOM, etc.
    - Complex design process w/ competing commercial interests
    - Reference implementation for debugging
- Implementation is not the spec!
    - Huge issue for, e.g., Windows, HTML, Office, etc
    - Retrofitting a spec is a huge pain
        - For example, Office file formats switched to XML