Measuring Performance

- Use analysis, e.g., big-Oh, to reason about performance
  - Good in practice, what about in theory?
  - The proof of a pudding is in the eating...

- Use coarse-granularity timing
  - `/bin/time` or shell time functions
  - Tapestry `CTimer` functions

- Use a profiling tool like `gprof`
  - See resources page

- What about our friend Heisenberg?
Does performance matter?

- **Obviously it depends**
  - Run it once, throw it away: probably don’t care
  - Route TCP/IP packets: millions/second: do care

- **Performance is one aspect of a system**
  - Correctness
  - Time to develop/deploy
  - If it’s not a bottleneck, don’t optimize

- **Premature optimization is the root of all evil (or similar): Knuth**
  - Remember, the compiler is your friend: try -O2
Time and Space trade-offs

- **Two principal performance resources: time and space**
  - Time is CPU time, wall-clock time isn’t always the same
  - Space is memory resources used by program

- **Often we can trade-off time for space and vice versa**
  - Consider the O(1) solution to itoa
    - \(2^{32}\) values x 8 bytes/value = 32 Gigabytes (conservative)
  - What else is expensive? (allocating memory, %)

- **There are sometimes subtle effects due to cache**
  - What’s on disk, what’s in memory?
  - Memory is “different”, cache, main memory, etc.
Modules, design, coding, refactor, XP

- Make it run, make it right, make it fast, make it small
- Do the simplest thing that can possibly work (XP)
  - Design so that refactoring is possible
  - Don’t lose sight of where you’re going, keep change in mind, but not as the driving force [it will evolve]

- Refactor: functionality doesn’t change, code does
  - Should mean that new tests aren’t written, just re-run
  - Depends on modularity of code, testing in pieces

- What’s a module in C++
  - Could be a class, a file, a directory, a library, a namespace
  - We should, at least, use classes, files, directories
See filterdemo.cpp

- Filter is an abstract base class, why?
  - What is filter designed to do?
  - Why make an abstract base class?

- Problem: We want to have MinFilter, MaxFilter, MinMaxFilter, etc., lots of different kinds of filters
  - We can’t make a new class for all, there are too many when combined with each other
  - We can use the decorator pattern to solve the problem

- We want to add responsibilities to objects (not classes)
  - Add dynamically, also remove
  - Extension by subclassing impractical (too many)
  - Create an interface, decorator both is-a and has-a
Decorator

- **Filter**: specifies an interface, other filters implement the interface
  - Chain filters together by forwarding queries

![Diagram of Filter, SizeFilter, and AndFilter]

- `Filter`
  - `ok()`
  - `operator()`

- `SizeFilter`
  - `ok()`
  - `operator()`

- `AndFilter`
  - `ok()`
  - `operator()`
Filter in scandir

- **What's the C approach to passing functions?**
  - What's a pointer to a function?
  - How to declare?
  - How to parse (by people)

- **What's the C++ approach to passing functions**
  - Same as C, but we get more too (of course)
  - Make an object that acts like a function
    - Function object
    - Advantages? What about state, parameters?
    - Ubiquitous in STL
Eric Raymond

- Open source evangelist
  - The Cathedral and the Bazaar
    [http://ot.op.org/cathedral-bazaar.html](http://ot.op.org/cathedral-bazaar.html)
  - How to construct software

  “Good programmers know what to write. Great ones know what to rewrite (and reuse).”

- How to convince someone that guns are a good idea? Put this sign up:

- THIS HOME IS A GUN-FREE ZONE