Advanced programming techniques

Andrew Koenig

ark@research.att.com
http://www.research.att.com/info/ark

Today's topics

- Who am I?
- What is the purpose of this course?
- What will it cover?
- What will be expected of you?

Who am I?

- Programming since 1967
- Used lots of machines and languages
- Coauthor of Columbia Computer Chess Program, 1970–71
- Project editor of C++ standards committee

The purpose of the course

- Learn ideas in programming and system design that transcend any one language
- Learn that programming is not just coding
- Learn by doing

Underlying philosophy

- The software universe changes fast
- It is easy to drown in details of one system or another
- Nevertheless, there are things we can learn that can endure

The most important idea

- Abstraction
  - "the act or process of separating in thought, of considering a thing independently of its associations; or a substance independently of its attributes; or an attribute or quality independently of the substance to which it belongs" (OED)
  - "leaving out of a number of resembling ideas what is peculiar to each" (attributed to Locke by Priestly, 1782)
  - Selective ignorance

Abstraction in practice

- Reversing an array

```
i = 0; j = n-1;
while (i < j) {
 swap(a[i], a[j]);
 ++i; --j;
}
```

- What extraneous dependencies are there in this fragment?

Generalizing the algorithm

- Reversing a sequence

```
p = &a[0]; q = &a[n];
while (p != q) {
 swap(*p, *q);
 ++p;
 if (p != q)
     --q;
}
```

- What dependencies have we removed?

Other ways of reversing

- Copy from one sequence to another, reversing as you go.
- Attach a tag to every element, then sort the sequence.
- Push the elements on a stack, then pop them.

Other forms of abstraction

- Every programming language is an abstraction of a computer
- A file system abstracts a particular style of information storage and retrieval
- A relational database abstracts a different style

Abstractions create barriers

- You can't take advantage of what you're ignoring.
- An exam question from another course:
  - The purpose of an operating system is to keep users away from the computer. Discuss.
- "Good fences make good neighbors."

Why are barriers good?

- Information flow across barriers is controlled, and thereby reduced.
- When we design a large system, we can avoid having to learn about what is on the other side of a barrier: We care only about what crosses it.

Abstraction is rarely free

- Constrained information flow is usually less efficient.
- "Why can't I just reach in and tweak that variable? It's sitting right there…"

So what do we do?

- A key to successful programming is knowing how abstract to be and when.
  - Totally concrete programs take too long to write, and don't work.
  - Totally abstract programs take too long to run, and don't do enough.
- A sense of perspective is important.
  - $O(n \log n)$ is nearly $O(n)$, but $O(n^2)$ isn't.

The point of these examples

- Abstraction is useful
  - It is better to solve the reversing problem once and be done with it
  - More generally, we need a way to cope with problems that are too big to handle all at once
- Total abstraction is impossible
  - Ignoring everything leaves us with nothing

What is expected of you?

- Come to class. Ask questions. Be critical. Think for yourself.
- Form project teams (3–6 people). Propose a project; get approval; do it.
- Watch the calendar.

Schedule

- Class: M, W 1:30–2:50
- Project deadlines:
  - Proposals due Friday, February 27
  - Presentations March 2 (and 4?)
  - Revised proposals due Monday, March 23
  - Projects due Monday May 18 (during finals)
- I will be away March 9, 11; expect a guest lecturer and an exam

Course grades

- Project counts 40%; all team members get the same project grade
- Homework counts 40%; exams 20%
- Grades will usually be based on medians, not means
- Project must be complete to receive a grade at all!

Calibration

- Last year's grades:
  - A (including ±): 16
  - B (including ±): 11
  - C (± not allowed): 2
  - D, F: none

Program grades

- Does it work?
  - Does it do what it is supposed to do?
  - For the project: Does it do what the proposal said it would do? Does it do more? How ambitious is it?
- How easy is it to tell that it works?
- How well is it written?

Mechanics of programming

- I will probably use C++ for most examples, explaining as I go.
- You can do homework in any language.
- You know more than I do about the local computing facilities.
- You may wish to consider language preferences when choosing project partners.

Project teams

- Form your own teams (3–6 people)
- The team picks the project (try to choose something fun and useful)
- Get started early; ask if you need help

Project proposals

- Pretend that I'm the CEO of a venture capital company
- The proposal is what you will use to convince me to fund your startup
- You *must* do what you proposed!
  - Don't be too ambitious
  - Make it work correctly; then add to it

Project essentials

- Design—even (especially!) if it changes during development
- Test plan
  - You can develop the test facilities while you're developing the system
  - One person should probably work exclusively on testing
- Documentation (external and internal)
- Organization (who is doing what?)

Homework, part 1
(due Monday)

- Write a program that reads a file containing text and "fills" the lines to make them roughly the same length.
- Assume that a line with no text on it begins a new paragraph.

Homework, part 2

- Assume you're using a programming language that supports strings, in which evaluating `s+t` takes *O(len(s)+len(t))* time. How long does this loop take?

```
s = "";
while (--n >= 0)
    s = s + x;
```

- Prove it.

Suggested reading (part 1)
(All published by Addison-Wesley)

- Bentley: *Programming Pearls* and *More Programming Pearls*
- Brooks: *The Mythical Man-Month*
- Gamma, Helm, Johnson, and Vlissides: *Design Patterns*
- Koenig: *C Traps and Pitfalls*
- Koenig and Moo: *Ruminations on C++*
- Lippman: *C++ Primer*

Suggested reading (part 2)

- Reade: *Elements of Functional Programming*
- Sethi: *Programming Languages—Concepts and Constructs*
- Stroustrup: *The C++ Programming Language*