Programming Languages

Vitaly Shmatikov

http://www.cs.utexas.edu/~shmat/courses/cs345/

Course Personnel

- Instructor: Vitaly Shmatikov
  - Office: CSA 1.114
  - Office hours: Tuesday, 3:30-4:30pm (after class)
  - Open door policy – don't hesitate to stop by!
- TAs: Jeremy Stober and Austin Waters
  - Office: PAI 5.38 (TA station #1)
  - Office hours: Monday, 2-3pm and Wednesday, 1:30-2:30pm (Jeremy), TBA (Austin)
- Watch the course website
  - Assignments, reading materials, lecture notes

Course Logistics

- Lectures: Tuesday, Thursday 2-3:15pm
- Homeworks and programming assignments
  - 49% of the grade (7 assignments, 7% each)
- Three in-class exams (2 midterms and final)
  - 51% of the grade (17% each)

No make-up or substitute exams!

If you are not sure you will be able to take the exams in class on the assigned dates, **do not take this course!**

Code of Conduct

- UTCS Code of Conduct will be strictly enforced
- All assignments are strictly individual
  - Unless explicitly stated otherwise
  - “ We were just talkin’ ” is not an excuse
  - No Googling for answers!
- You do **not** want me to catch you cheating

Late Submission Policy

- Each take-home assignment is due in class at 2pm on the due date
- You have 3 late days to use any way you want
  - You can submit one assignment 3 days late, 3 assignments 1 day late, etc.
  - After you use up your days, you get 0 points for each late assignment
  - Partial days are rounded up to the next full day

Course Materials

- Textbook:

  Mitchell. "Concepts in Programming Languages."
  - Attend lectures! Lectures will cover some material that is not in the textbook – and you will be tested on it!
- Harbison, Steele. "C: A Reference Manual."

  (5th edition)
- Occasional assigned readings

Other Helpful Books

- Bison Manual
- Dybvig. “The Scheme Programming Language.”
- Harper. “Programming in Standard ML.”
- All of these are available for free online
  - See links on the course website

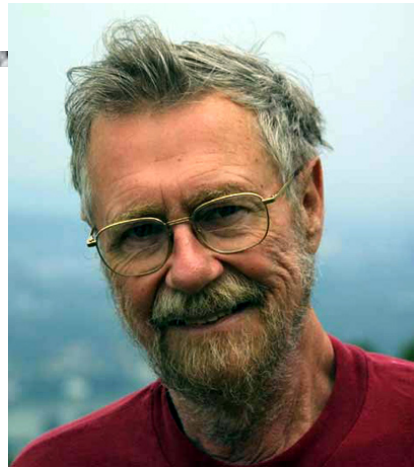
Syllabus

- Survey of fundamental concepts underlying modern programming languages
  - Goal: understand paradigms, not vocational training in any given language
  - Examples drawn from ANSI C, C++, Java, Scheme, ML, JavaScript ...
- Procedural / imperative
- Functional / applicative
- Object-oriented
- Concurrent

Some Course Goals

- Language as a framework for problem-solving
  - Understand the languages you use, by comparison
  - Appreciate history, diversity of ideas in programming
  - Be prepared for new methods, paradigms, tools
- Critical thought
  - Identify properties of language, not syntax or sales pitch
- Language and implementation tradeoffs
  - Every convenience has its cost
    - Recognize the cost of presenting an abstract view of machine
    - Understand tradeoffs in programming language design

Dijkstra on Language Design



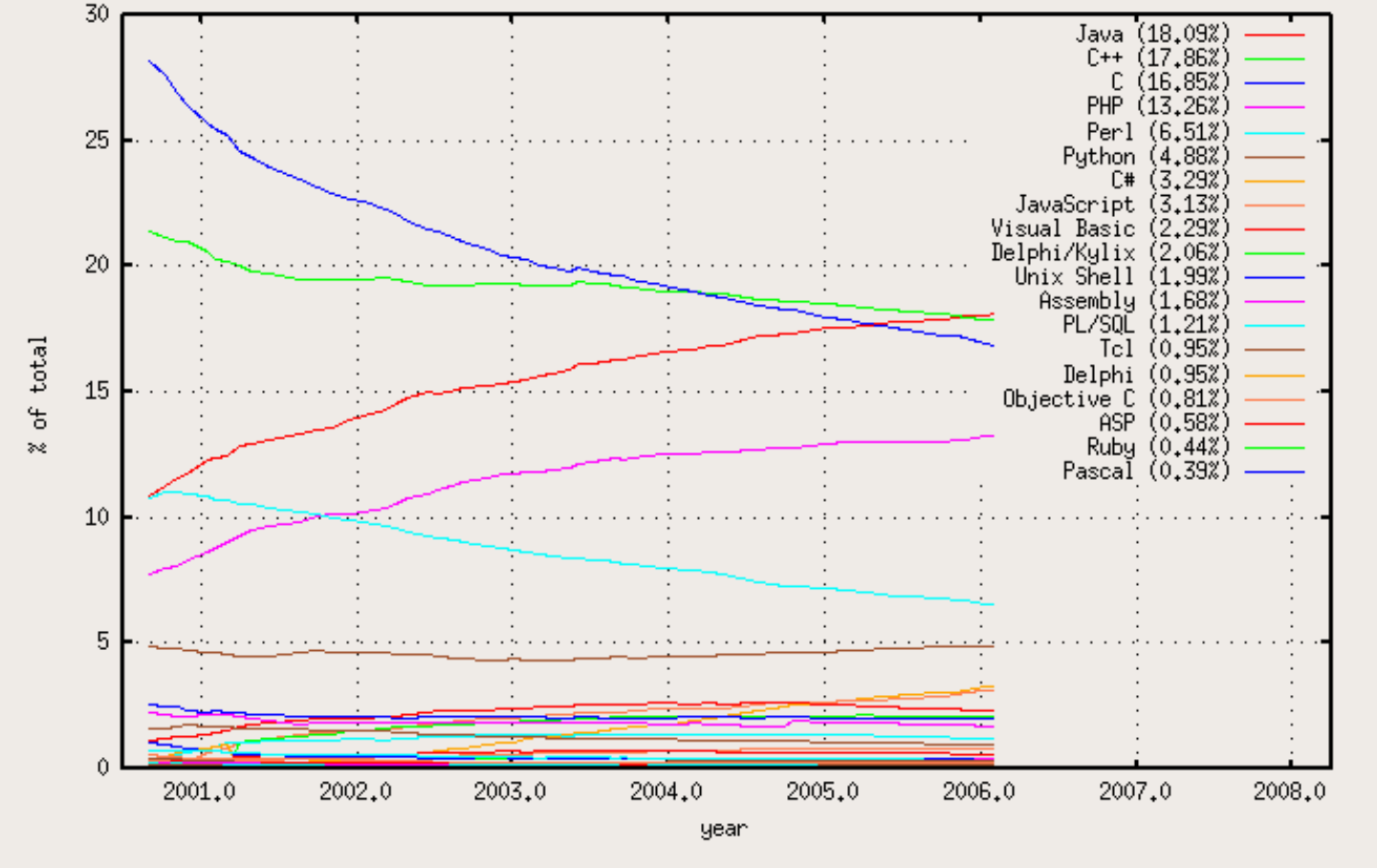
- “The use of COBOL cripples the mind; its teaching should, therefore, be regarded as a criminal offence.”
- “APL is a mistake, carried through to perfection. It is the language of the future for the programming techniques of the past: it creates a new generation of coding bums.”
- “FORTRAN, 'the infantile disorder’ ... is hopelessly inadequate for whatever computer application you have in mind today: it is now too clumsy, too risky, and too expensive to use.”
- “It is practically impossible to teach good programming to students that have had a prior exposure to BASIC: as potential programmers they are mentally mutilated beyond hope of regeneration.”

What's Worth Studying?

- Dominant languages and paradigms
  - C, C++, Java... JavaScript?
  - Imperative and object-oriented languages
- Important implementation ideas
- Performance challenges
  - Concurrency
- Design tradeoffs
- Concepts that research community is exploring for new programming languages and tools

Languages in Common Use

[F. Labelle]



Based on open-source projects at SourceForge

Flon's Axiom

"There is not now, nor has there ever been,
nor will there ever be,
any programming language in which
it is the least bit difficult to write bad code."

- Lawrence Flon

Latest Trends

- Commercial trends
  - Increasing use of type-safe languages: Java, C#, ...
  - Scripting and other languages for Web applications
- Teaching trends: Java replacing C
- Research and development trends
  - Modularity
  - Program analysis
    - Automated error detection, programming environments, compilation
  - Isolation and security
    - Sandboxing, language-based security, ...

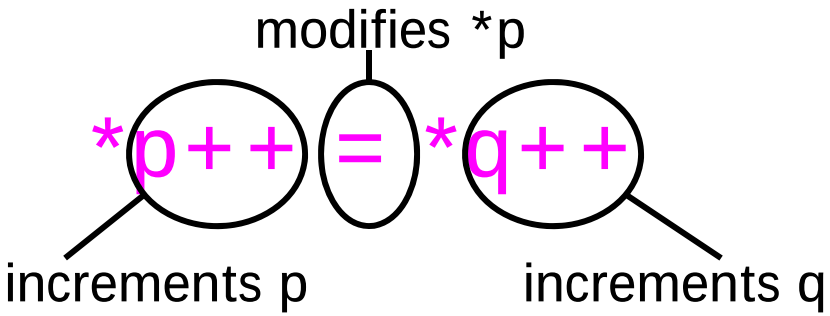
Support for Abstraction

- Data
  - Programmer-defined types and classes
  - Class libraries
- Procedural
  - Programmer-defined functions
  - Standard function libraries

Reliability

- Program behavior is the same on different platforms
  - E.g., early versions of Fortran
- Type errors are detected
  - E.g., C vs. ML
- Semantic errors are properly trapped
  - E.g., C vs. C++
- Memory leaks are prevented
  - E.g., C vs. Java

What Does This C Statement Mean?



Does this mean...

```
*p = *q;
++p;
++q;
```

... or

```
*p = *q;
++q;
++p;
```

... or

```
tp = p;
++p;
tq = q;
++q;
*tp = *tq;
```

Orthogonality

- A language is orthogonal if its features are built upon a small, mutually independent set of primitive operations.
- Fewer exceptional rules = conceptual simplicity
  - E.g., restricting types of arguments to a function
- Tradeoffs with efficiency

Efficient Implementation

- Embedded systems
  - Real-time responsiveness (e.g., navigation)
  - Failures of early Ada implementations
- Web applications
  - Responsiveness to users (e.g., Google search)
- Corporate database applications
  - Efficient search and updating
- AI applications
  - Modeling human behaviors