Computer Science 203
Programming Languages
Fall 2004
Cormac Flanagan
University of California, Santa Cruz

Administrivia

• Who am I
  - Cormac Flanagan
  - cormac@cs.ucsc.edu
  - http://www.soe.ucsc.edu/~cormac/

• Office hours
  - Baskin 353A (till mid-October, then E2)
  - Tuesdays, 2:00-3:30
  - and by appointment

• Teaching assistant
  - Aaron Tomb (atomb@soe.ucsc.edu)

• Web page
  - http://www.soe.ucsc.edu/classes/cmps203/Fall04/
  - lecture slides, notes, homeworks, project ideas

Prerequisites

• Programming-language background
  - some programming experience
  - exposure to several languages (e.g., C, Java, ML, Lisp, Prolog)
  - Computer Science 112 or equivalent

• Some mathematical sophistication
  - ability to follow and do proofs
    - eg, proofs by induction
  - some acquaintance with mathematical logic
  - ease with formal notation and manipulation

Prerequisites (cont.)

• Please ask me if:
  - you are not sure that you meet the prerequisites, or
  - you are an undergraduate, or
  - a graduate student from outside CS or CE, or
  - if you did not enroll in the course but want to attend

Course Goals

• Learn about language and program analysis
  - semantics (operational, denotational, axiomatic)
  - type systems

• Learn about some languages and language features
  - lambda calculus, objects, concurrency

• Learn a little about research topics and applications
  - security of module code, information flow

Reading

• Required reading
  - "The Formal Semantics of Programming Languages" by Glynn Winskel
  - Several papers, indicated during the course
  - Language documentation

• Some optional, additional books
  - "Foundations for Programming Languages" by John Mitchell
  - "Types and Programming Languages" by Benjamin Pierce

• All books on reserve in the library
Course work

- Reading
- Class participation

Homework
- concentrated in first half of course
- announced and explained in class
- usually due at the start of class one week later (strictly)

A small final project
- concentrated in second half of course

The final project

- Three kinds
  - small survey of recent work on a relevant topic (individual)
  - programming project
  - research paper

- Team (1-4 people) or individual projects
- You select the project, I will suggest some
- Scale
  - 20-40+ hours of work per person
  - short report
  - short presentation
- More later

Grades

- Grades are determined as follows
  - homework: 60 - 70%
  - project (including report and presentation): 20 - 30%
  - class participation: the rest

- Regular class attendance is required.

Who are you?

The Landscape

- Programming languages is one of the oldest fields in computer science.
- It remains an active, vibrant field.

- Some trends:
  - The web renews interest in language design (see Java).
  - Type safety gains acceptance as playing a role in security.
  - Program analysis becomes a significant component of what is called software engineering.

Language Design in Practice

- Languages are adopted to fill a void.
  - To enable a previously difficult/impossible application.
  - To run on a new platform.
- Language quality is sometimes secondary.

- Programmer training is a huge cost.
  - Languages with many users are replaced rarely.
  - Popular languages become ossified.
  - Conversely, new applications and new platforms present opportunities for innovation.
Why So Many Languages?

- Many languages were created for specific applications
- Application domains have distinctive (and conflicting) needs.
- Examples:
  - Artificial intelligence: symbolic computation (Lisp, Prolog)
  - Scientific computing: high performance (Fortran)
  - Business: report generation (COBOL)
  - Systems programming: low-level access (C)
  - Customization: scripting (Perl, Javascript)
  - Distributed systems: mobile computation (Java, C#)
  - And more (LaTeX, awk, Hancock, …)

Language Paradigms

- Imperative, procedural
  - Fortran, Algol, Cobol, C, Pascal
- Functional (mostly assignment-free)
  - Lisp, Scheme, ML, Haskell
- Object oriented
  - Simula, Smalltalk, Eiffel, Self, C++, Java
- Logic
  - Prolog, JProlog
- Concurrent
  - Fortran90

What Makes a Good Language?

- There are no universally accepted metrics for design.
- "A good language is one people use"?
  - Is COBOL the best language?

Good Language Features

- Simplicity (of syntax and semantics)
- Readability and writability
- Familiarity
- Safety
- Machine independence
- Support for programming in the large
- Efficiency (of execution and compilation)

What is Science?

- "When you can measure what you are speaking about, and express it in numbers, you know something about it; but when you cannot measure it, when you cannot express it in numbers, your knowledge of it is of a meager and unsatisfactory kind; it may be the beginning of knowledge, but you have scarcely, in your thoughts, advanced it to the stage of science."
  - Lord Kelvin (1824-1907)

Good Language Features (cont.)

- These goals almost always conflict.
  - Safety checks cost something in either compilation or execution time.
  - Safety and machine independence may exclude efficient low-level operations.
- Type systems restrict programming style in exchange for strong guarantees.
- Compromises are hard.
- Simplicity helps with other goals, but is rare.
- Good language design is hard!
Languages vs. Features

- Popular languages are isolated points in a large space.
- The language designer should (according to Hoare)
  - be familiar with many features,
  - have good judgment in choosing and reconciling features,
  - know the application domain of the new language,
  - know how hairy the language should get,
  - have resources for implementations, documentation, and environment.
- The language designer should not include new, untried ideas
  - "Goal is consolidation, not innovation."
  - e.g. Java

Studying Features

- A language concept or construct can be studied in the context of a large (real) language
  - good for understanding pragmatics in practice
- But is often easier to study a language feature by considering a small language that embodies it
  - easier experimentation
  - deeper, more precise reasoning and understanding

Considering Features in Small Languages

- Easier to:
  - experiment with the use of the language
  - define the language rigorously
    - syntax
    - compile-time checks
    - run-time behavior
  - study the properties of the language
    - is it deterministic?
    - what does typechecking guarantee?
      - applications: no information leakage? no viruses?

Considering Features in Small Languages

- Easier to:
  - experiment with the use of the language
  - define the language rigorously
    - syntax
    - compile-time checks
    - run-time behavior
  - study the properties of the language
    - is it deterministic?
    - what does typechecking guarantee?
      - applications: no information leakage? no viruses?

What is Science?

- "When you can measure what you are speaking about, and express it in numbers, you know something about it; but when you cannot measure it, when you cannot express it in numbers, your knowledge of it is of a meager and unsatisfactory kind; it may be the beginning of knowledge, but you have scarcely, in your thoughts, advanced it to the stage of science."

- Lord Kelvin (1824-1907)
Homework 1

- Due at start of class, Sept 30 (strictly)

- Download the Objective Caml system from http://caml.inria.fr/ocaml/.

- Write a function $f$ that takes an integer argument $n$ and outputs the string “Hello world” followed by $|n|$ “!”s, where $|n|$ is the absolute value of $n$.
  - For example, $f(0)$ should output “Hello world”,
  - and $f(2)$ and $f(-2)$ should both output “Hello world!!”.
  - Include some test runs in your homework.