Overview

- This course is about:
  - Programming Languages
  - What they are
  - How they work
  - How we can reason about them
- This course is NOT about:
  - Programming
    - Not the place to learn Java, C#, etc.
  - Applications
    - Not the place to learn about GUIs, applets, etc.
- Brief history of this course
  - Pre-Java (the bad old days)
  - Java and the modern era
  - This year and beyond

Administrivia

- Web: http://www-edlab.cs.umass.edu/cs530
  - The definitive source for information; Check it often!
- Text: Programming Languages: Theory and Practice
  - Robert Harper
  - Working Draft (available on the web)
- People
  - Prof. Jack Wileden
  - Matthew Cheung, TA
- Misc
  - Video (PEEAS, NTU) presentation
  - Course schedule in syllabus

Administrivia (cont.)

- Grading
  - 35% homework assignments
  - 35% programming assignments
  - 25% project
  - 5% class participation (in-class quizzes and writing, etc.)
- Notes on these
  - Three (or so) homework assignments
  - Three (or so) programming assignments
  - Occasional in-class assignments, lowest grade dropped
  - Projects will involve web programming, “traditional” programming languages and/or formal foundations
  - More details soon

Remainder of Class

- Introductory discussion
  - Toward a Science of Programming Languages
- Homework 0
  - Due (by email) ASAP
  - See WWW
- Reading assignment 1
  - See WWW
- End early for on-site administrative issues

Toward a Science of Programming Languages
(borrowing from Frank Pfenning)

- Programming Languages: What? Why?
- Studying Programming Languages: How?
What Are Programming Languages For?

• Role of programming languages:
  – In the science of computing?
  – In the practice of computing?

Practical Issues

• Facets of Programmer Productivity
  – Initial development time
  – Program correctness and robustness
  – Software maintainability

• Crucial Contributing Factors
  – Programming language(s)
  – Development environment(s)
  – Software engineering practices

Why Languages are Crucial

• How do we implement data structures?
• How do we design and structure code?
• How do we express assumptions and guarantees?
• How do we read and analyze a program?

What Make a Language Good?

• “An ideal language allows us to express easily what is useful for the programming task and at the same time makes it difficult to write what leads to incomprehensible or incorrect programs.” - Nico Habermann
• “Good languages make it easier to establish, verify and maintain the relationship between code and its properties.” - Robert Harper
• What do you think makes a language good?

Why Are Languages Like They Are?

• Science?
• Engineering?
• History?
• Marketing?
• Accident?
• ?????

How Can We Study Programming Languages?

• Objective Criteria
  – Syntactic properties
  – Type safety
  – Performance?

• Subjective Criteria
  – Ease of use
  – Power
  – Performance?

Toward a Science of Programming Languages: Basic Tools

- Type Theory: Techniques for structuring languages to ensure safety and modularity of programs
- Operational Semantics: Techniques for describing the execution behavior of programs, at various levels of abstraction
- Mathematical Logic: Techniques for specifying and verifying programs

One Approach to Studying Programming Languages: Vivisection

- Take one or several living languages, preferably widely used
- Analyze it or them in minute detail
  - Syntax: Grammar and parsing
  - Semantics: Type-checking and operational semantics
  - Pragmatics: Programming methodology and implementation strategies

Another Approach to Studying Programming Languages: Autopsy

- Take one or several dead languages, preferably (once) widely used
- Analyze it or them in minute detail
  - Syntax: Grammar and parsing
  - Semantics: Type-checking and operational semantics
  - Pragmatics: Programming methodology and implementation strategies

Another Approach to Studying Programming Languages: Genesis

- Take a problem domain, preferably useful
- Design the ultimate language
  - Syntax: Grammar and parsing
  - Semantics: Type-checking and operational semantics
  - Pragmatics: Programming methodology and implementation strategies

Another Approach to Studying Programming Languages: Taxonomy

- Analyze many languages based on a few criteria
- Create taxonomy of (living or dead) languages

Our Approach to Studying Programming Languages: Basic Concepts and an Exemplar

- Ignore issues of syntax (largely)
- Isolate and investigate basic concepts like:
  - Functions, procedures and variables
  - Classes, objects and methods
  - Effect-free vs. imperative programming
  - Static vs. dynamic typing
  - Concrete vs. abstract types
  - Sequential vs. concurrent vs. parallel prog.
- Emphasize mathematical tools
- Investigate an interesting new language: Scala

All of these approaches can be interesting and instructive -- BUT we won’t follow any of them
**Our Approach to Studying Programming Languages: Virtues and Goals**

- Not bound by flaws or limits of actual languages
- But can draw conclusions about actual languages
- After this course you should be able to:
  - Confidently critique existing languages
  - Define and analyze your own language
  - Prove properties of languages
  - Avoid common mistakes and pitfalls
  - Reflect more deeply on programming style
  - Write better programs (?)
  - Carry out research on programming languages

**Our Approach to Studying Programming Languages: Core Topics**

- **Mathematical Foundations**
  - Judgments and inductive definitions
  - Variable renaming and substitution
  - Structural induction
- **Language Description Techniques**
  - Concrete and abstract syntax
  - Static semantics via type systems
  - Dynamic semantics via abstract machines
  - Type safety and its consequences

**Our Approach to Studying Programming Languages: Language Features (tentative)**

- Continuations
- Exceptions
- Mutable storage
- Monads
- Parallelism
- Polymorphism
- Data abstraction
- Interoperability
- Laziness
- Dynamic typing
- Subtyping
- Inheritance
- Concurrency
- Storage management
- Refinement types
- Security

**Beyond the end: administrivia**

- Video instruction and you
- Syllabus accuracy
  - The Web!
- Homework
  - Late homework policy
  - Use of your own computer
- In-class assignments
- Project groups
- Grade grubbing
- Collaboration, plagiarism, and honesty
- Honors section