Course Summary

- Programming Languages
- Translation of Programming Languages
- Tools
- Construction of Complex Software

Programming Languages

- Scope of declarations
- Scope vs. extent (lifetime) of variables
- Interactions between language design and runtime structures:
  - Function representation
    - Effects of recursion, variable-sized data, functional values
  - Inheritance
    - Single vs. multiple inheritance
    - Java-style interfaces
- Formal methods for describing languages: context-free grammars, type systems, formal definitions of operational semantics.

Translation of Programming Languages

- Lexical analysis
  - regular expressions, finite automata
- Context-free syntax
  - BNF
  - Top-down, recursive descent
  - Bottom-up, shift-reduce parsing
  - Terminology: derivation
  - Syntax-driven translation
- Static semantics
  - Symbol tables, relation to environment diagrams
  - Types, type inference

Translation of Programming Languages, contd.

- Code generation, intermediate forms
- Runtime representations for “special effects”
  - Exceptions
  - Procedure calls
  - Object-oriented method dispatch
  - Garbage collection
- Optimization
  - Terminology: basic blocks, control-flow graph
  - “Classical” optimizations
  - Structure of flow analysis

Other Static Analyses

- We've seen:
  - Static type analysis.
  - Program verification: axiomatic semantics.
  - Static pointer analysis (Andersen's Points-To Analysis).
- Each of these have a number of uses:
  - Check that program conforms to language rules.
  - Check that program conforms to desired specification.
  - Eliminating run-time checks.
  - Determining whether certain optimizations are possible.

Tools

- Lexer-generation, use of regular expressions and states.
- Parser generators, rule-based programming.
- A little more exercise in version-control concepts.

Construction of Complex Software

- Be familiar with project, including parts you didn't write.
- Concept of a "pass" or "phase".
- Use of object-orientation to partition task
- Importance of intermediate forms; how used to reduce work of porting compilers

Parting Remarks

- It's not just compilers:
  - Ideas in this course are general-purpose tools
  - Think domain-specific languages
- Opportunities for research
  - Parallelism and distributed computation
  - Static program analysis:
    - Supports compiling for parallelism & distributed computation.
    - Analyzing programs for security attacks/flaws
    - Formal analysis for program validation (e.g., avionics)