CS321 Languages and Compiler Design I
Winter 2012
Lecture 1

COURSE GOALS
• Improve understanding of languages and machines.
• Learn practicalities of translation.
• Learn “anatomy” of programming languages.
• Apply computer science theory to practical problems (using tools).
• Do large programming project.

COMPILERS
A compiler is a translator from “high-level” language to assembly code/object language.
Language L → TRANSLATOR → Language L’
Examples of translators:
- Pascal, C, etc. → Compiler → Machine Code
  - Java → Compiler → Byte Code
- Ratfor → Preprocessor → Fortran
- Tex → Text Formatter → Postscript
- SQL → DB Optimizer → Query plan

We study common features of translators, by building one.

LANGUAGE DESIGN
We study languages mainly from an implementor’s viewpoint.
• How do compilation feasibility and runtime efficiency affect language design?
(There are more “theoretical” approaches to studying programming languages, and there are interesting and useful languages that don’t compile easily...)
“VON NEUMANN” MACHINE

“HIGH-LEVEL” LANGUAGES

E.g., Fortran, Pascal, C, Cobol, Java, JavaScript, Python...

Example

```plaintext
func rev (a: @real, n:int) {
  var i := 0;
  var j := n - 1;
  while i < j do {
    var x := a[i];
    a[i] := a[j];
    a[j] := x;
    i := i + 1;
    j := j - 1
  }
}
```

FEATURES OF LOW-LEVEL CODE

- Sequential control flow + labels + jumps
- Small set of built-in data types and operators (e.g., byte, integer, floating point)
- Flat linear address space.
- Memory hierarchy (registers faster than memory faster than disk).

FEATURES OF HIGH-LEVEL CODE

- Expressions (arithmetic, logical)
- Control structures (loops, conditionals, etc.)
- Type declarations and type checking
- Composite types (arrays, records, etc.)
- Procedures/Functions, with private scope
- Abstraction facilities!
How can we make high-level language run on a Von Neumann machine?

Answer:

• Translate HLL into lower-level code  
  (in traditional compiler, to machine code.) 
and/or

• Build a “higher level” virtual machine  
  (in traditional interpreter, perhaps a stack machine.)

In practice, we do some of both, even in a compiler, since generated machine code makes use of a runtime library and operating system.

Syntax is easy.

• Well-understood.
• Good theory: regular and context-free languages and automata.
• Good tools, even for complex cases.

Semantics are hard.

• Inherently complex.

• Variety of choices:
  
  Informal — Reference Manual  
  Operational — Definitional interpreter  
  (↑ we will focus here)  
  Axiomatic — Logic  
  Denotational — Mathematical functions etc.

• Few tools.