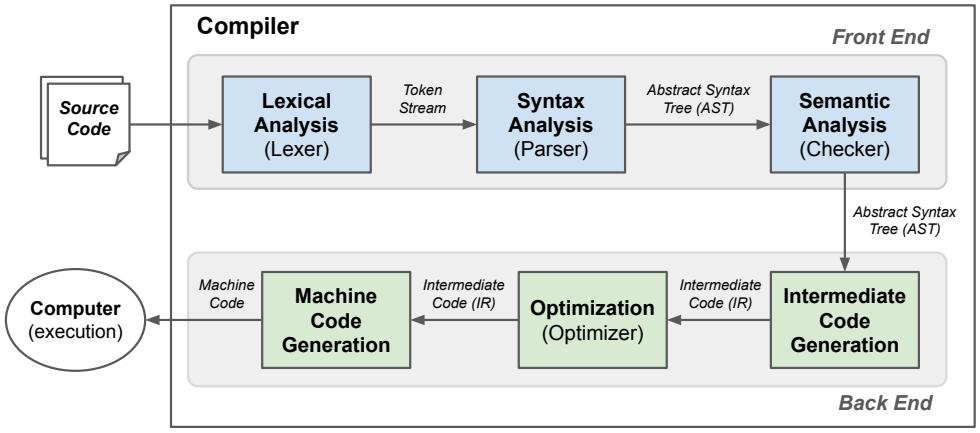
Lecture 3:

- Compilation and Interpretation (basics)

Announcements:

- HW-0 out (finish before Fri)

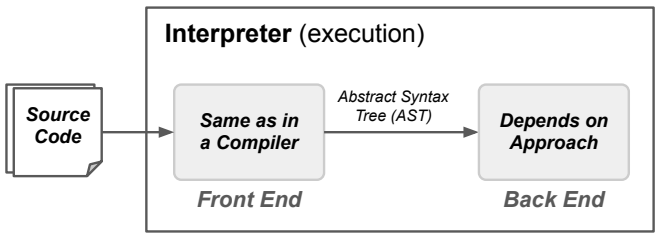
Typical Steps (Phases) of a Compiler



Example of "separation of concerns" ... design strategy

- each phase / step has a specific task
- too complex to do all steps "all at once" ... note: single-pass vs multi-pass
- each phase makes it easier to maintain, optimize, extend, reuse system

Interpreters



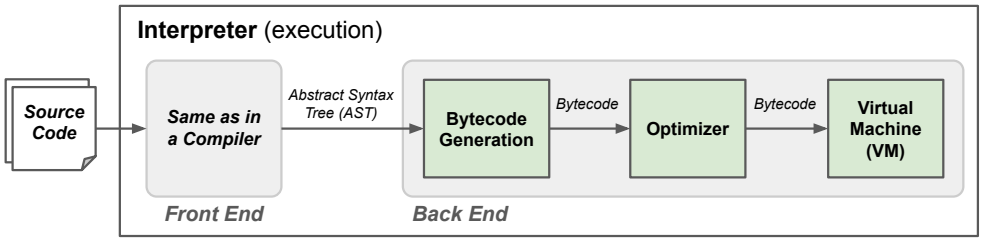
- instead of producing machine code ...
- the interpreter is a program that executes the source code directly

There are many types of interpreters

1. Abstract Syntax Tree (AST) Interpreters

- execute the program over the AST directly
- may involve an optimization pass over the AST first

Interpreters (cont)



2. Bytecode Interpreters (aka VMs) ... what we'll do

- intermediate representation is bytecode
- interpreter runs bytecode directly ... "write once run anywhere"

3. Just-in-time Compiler (JIT) ... "hybrid" approach

- instead of just interpreting bytecode, also generates and runs machine code
- monitor running code (e.g., frequent "hot spots") and optimize accordingly

Transpilers and Additional Notes

Transpilers

- translate from one high-level language to another (e.g., Python to C)
- includes same "front end" compilation steps

Compilers vs Transpilers:

- compilers translate from a high-level to a low-level language
- transpilers translate from a high-level to high-level language

AOT vs JIT: ... AOT = Ahead-of-Time

- AOT: generate low-level code (intermediate or machine) and execute it
- JIT: generate low-level code (machine language) as program executes
- JIT early examples: LISP (60's), Smalltalk (80's), Java (90's)
- more JIT: .NET, LLVM, PHP, V8 (javascript), CPython (experimental)