CS 406: Compilers and Interpreters

Stefan D. Bruda

Winter 2016

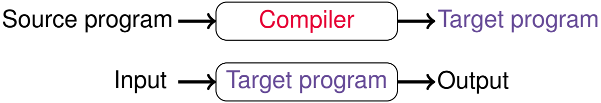
CS 406: COMPILERS AND INTERPRETERS

- Coordinates:
  - **Course Web page:**

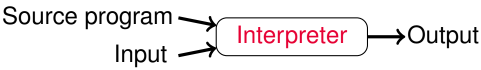
    http://cs.ubishops.ca/home/cs406
  - Instructor: Stefan Bruda
    (http://bruda.ca, stefan@bruda.ca, Johnson 114B, ext. 2374)
  - Office hours?
- Textbook (required): C. N. Fischer, R. K. Cytron, and R. J. LeBlanc Jr, Crafting a Compiler, Addison Wesley, 2009.

Compilation and interpretation

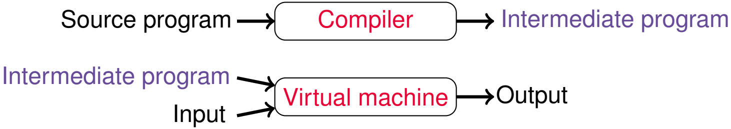
- Pure compilation: The compiler translates the high-level source program into an equivalent target program (typically in machine language), then goes away:



- Pure interpretation: The interpreter stays around for the execution of the program and becomes the locus of control during execution



- Compilation followed by interpretation:

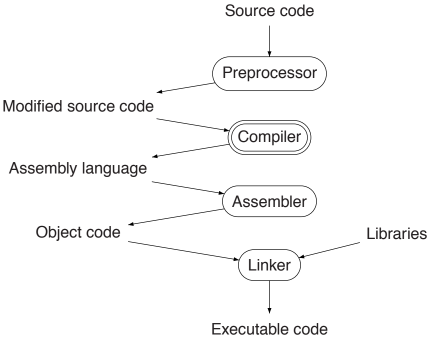


COMPILATION AND INTERPRETATION (CONT'D)

- Interpretation offers greater flexibility and better diagnostics, but compilation offers better performance
- Compilation does not have to produce machine language for some hardware
  - Compilation = translation from one language into another
  - Some compilers produce nothing but virtual instructions (Pascal P-code, Java byte code, Microsoft COM+)
- Compilation possibly preceded by a preprocessor

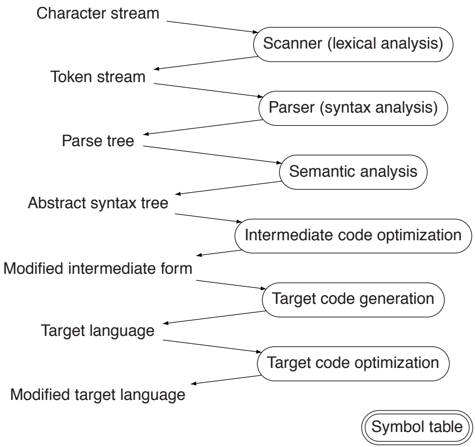
Compilation workflow

- For languages that compile to executable code:



- For languages that run on a virtual machine: the assembler and linker part are replaced by an interpreter (or virtual machine)

PHASES OF COMPILATION



- Scanner: divides program into "tokens" (smallest meaningful units)
  - Driven by regular expressions
- Parser: discovers the syntactic structure of a program
  - Driven by context-free grammar
- Semantic analysis: discovers the meaning of the program
  - Static analysis
  - Some other things can only be figured out at run time
- Intermediate form: tree-like structure and/or some machine-like language (but machine independent)
  - Often a form of machine language, but for an idealized machine

Phases of compilation (cont'd)

- Intermediate code optimization: produce code that does the same thing, only faster
  - Algorithmic optimization
- Code generation: produces assembly language for the target machine
- Code optimization: machine-specific optimizations (use of special instructions or addressing modes, reorder instruction to improve the load on superscallar architectures, etc.)

- Symbol table: all phases rely on a symbol table that keeps track of all the identifiers in the program and what the compiler knows about them
  - This symbol table may be retained (in some form) even after compilation has completed, for use by a debugger