CS 420 - Compilers

Dr. Chen-Yeou (Charles) Yu

Interpreter & Compilers

- Interpreter
  - A program that reads an source program and produces the results by executing that program
- Compiler
  - A program that translates a program from one language (the *source*) to another (the *target*)

But they are facing the common issue

- Compilers and interpreters, they both need to read the input
  - A stream of characters and “understand” it
  - *This is called --- analysis*

Interpreter

- Program execution is interleaved with analysis
- i.e.

```
running = true;
while (running) {
    analyze next statement;
    execute that statement;
}
```

- Usually need repeated analysis of statements (particularly in loops, functions)

Compiler

- Read and analyze entire program
- Translate to semantically equivalent program in another language
  - Presumably easier to execute or more efficient
  - Should “improve” the program in some fashion

Typical Implementations (languages)

- Use compilers
  - C, C++, Fortran, Pascal
  - Needs optimization in many cases
- Use Interpreters
  - Perl, Ruby, Linux/Unix shells, Python

Typical Implementations (languages)

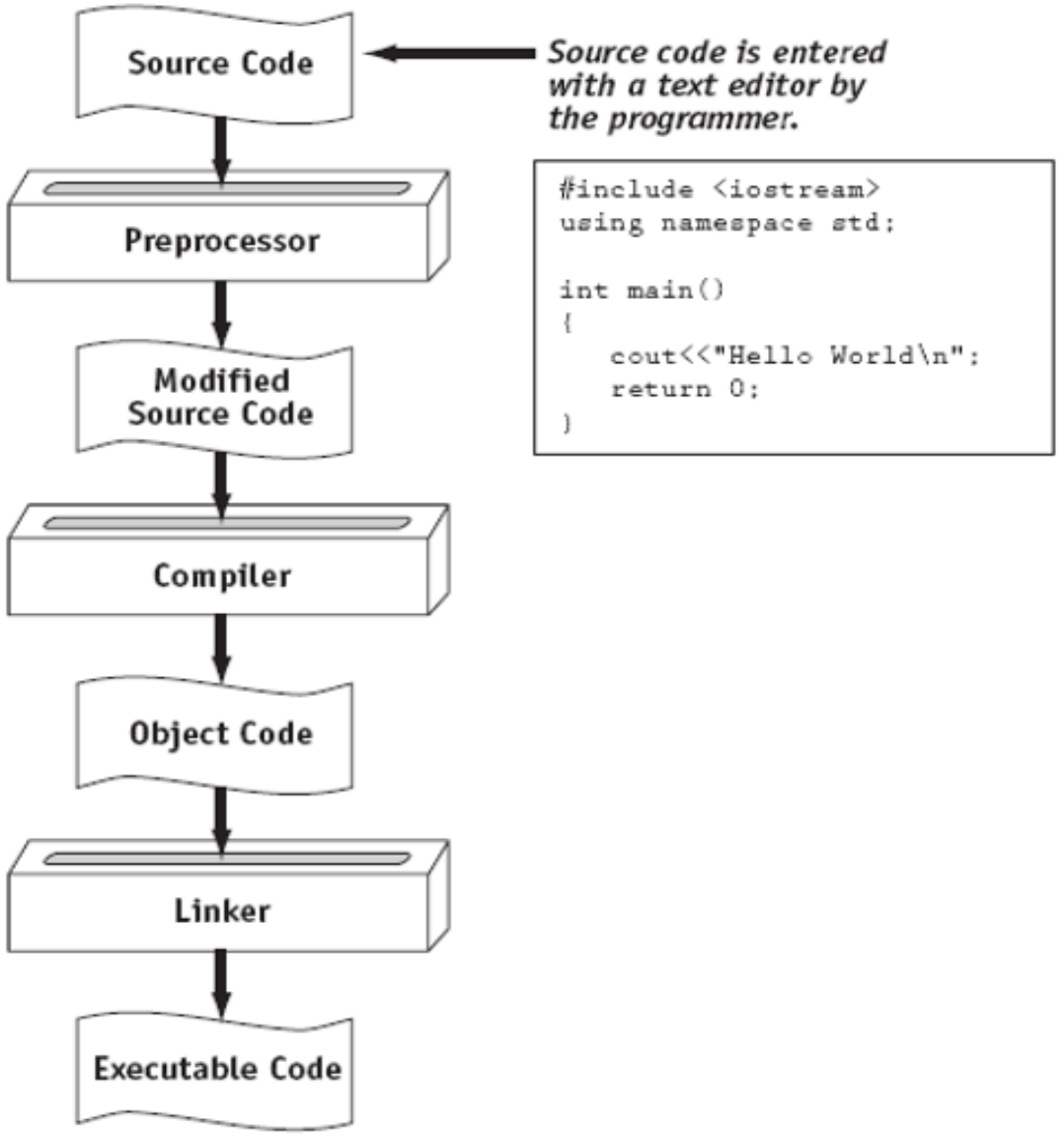
- Use Hybrid approach
  - Very famous one: Java
  - Compile Java source to byte codes – Java Virtual Machine language (.class files)
  - Execution:
    - Interpret byte codes directly, or
    - Compile some or all byte codes to native code
      - Just-In-Time compiler (JIT) – detect hot spots & compile on the fly to native code

Why we need to study compilers

- Become a better programmer?
  - Have better insight into interaction between languages, compilers, and hardware
  - Better intuition about what your code does
- Compiler techniques are everywhere, we need to know how it works
  - Parsing: XML
  - AI: domain-specific languages
  - Trust me, for some part of compiler technology is related to NLP (Natural Language Processing) !

Why we need to study compilers

- C → Assembly (Translated!)
- Can you feel that?

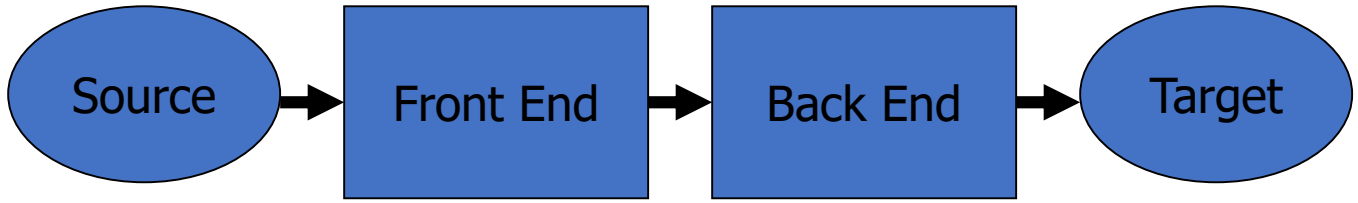


Why we need to study compilers

- Fascinating blend of theory and engineering
  - Parsing, scanning, static analysis
  - Difficult problems
    - Resource allocation
    - Approximations/heuristics

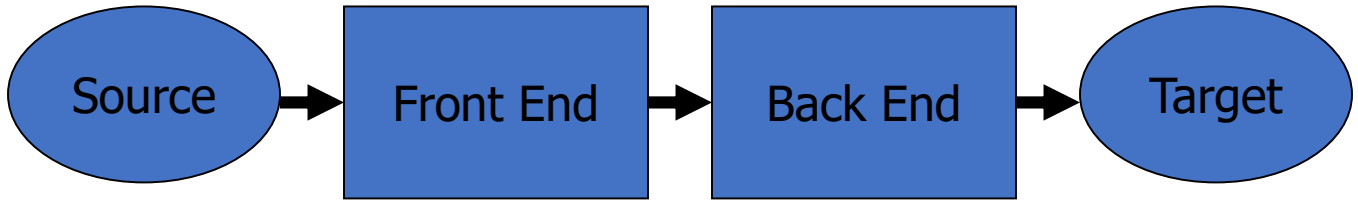
Structure of a Compiler

- A very high level description
  - Front end: analysis
    - Read source program and understand its structure and meaning
  - Back end: synthesis
    - Generate equivalent target language program



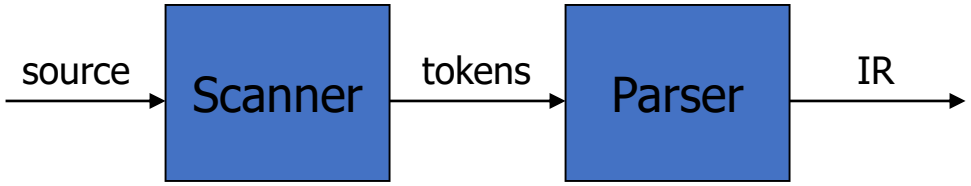
Structure of a Compiler

- Implications
  - Must recognize legal programs (& complain about illegal ones)
  - Must generate correct code
  - Must manage storage of all variables/data
  - Must agree with OS & linker on target format
  - Need some sort of Intermediate Representation(s) (IR)
  - Front end maps source into IR
  - Back end maps IR to target machine code



Front End

- Split into two parts (Remember the IPO model in CS 180 and 181?)
  - Scanner: Responsible for converting character stream to token stream
    - Also strips out white space, comments
  - Parser: Reads token stream; generates IR
- Both of these can be generated automatically
  - Source language specified by a formal grammar
  - We had software tools that can read the grammar and generate scanner & parser (either table-driven or hard-coded)



What is Token?

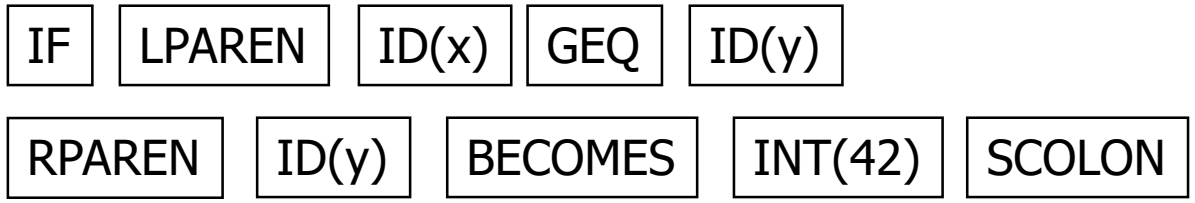
- Token stream: Each significant lexical chunk of the program is represented by a token
  - Operators & Punctuation: {}[]!+-=*;: …
  - Keywords: if while return goto
  - Identifiers: id & actual name
  - Constants: kind & value; int, floating-point character, string, …

Scanner Example

- Input text

```
// this statement does very little
if (x >= y) y = 42;
```

- Token Stream



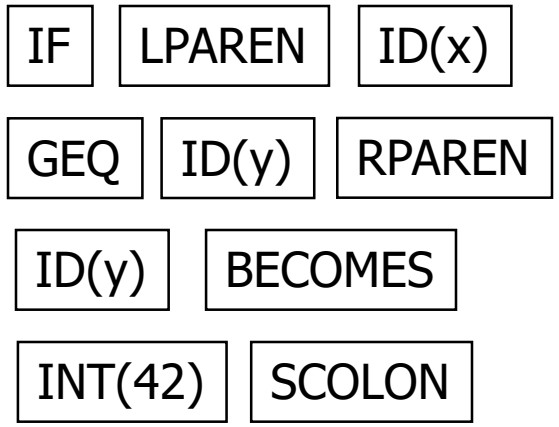
  - Notes: tokens are atomic items, not character strings; comments & whitespace are *not* tokens (in most languages)

Parser Output (IR)

- Many different forms
- But the common output from a parser is an abstract syntax tree (AST)
  - This is the essential meaning of the program without the syntactic noise
- Different forms may have different memory requirements

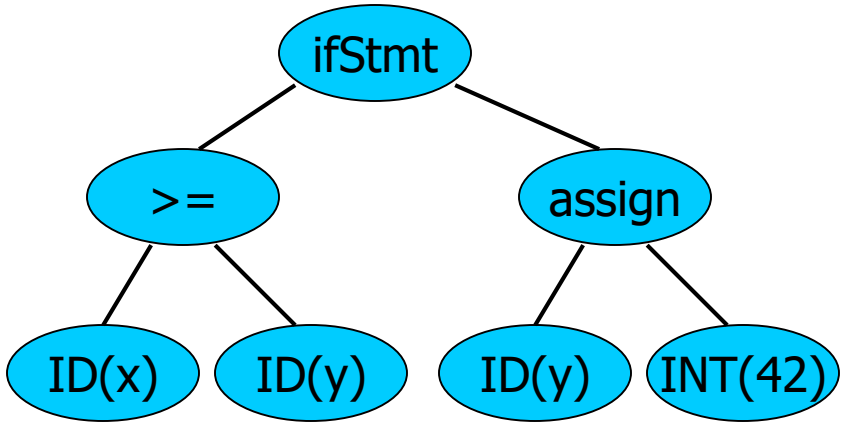
Parser Example

- Token Stream Input



Parser Example (Cont.)

- AST



During (more common) or After Parsing, what are the steps we need to do?

- Static Semantic Analysis
  - Type checking
  - Check language requirements like proper declarations
  - Preliminary resource allocation
  - Collect other information need to be used by “back end analysis” and “code generation” (later stages)