CS352 — Compilers: Principles and Practice

Zhiyuan Li

http://www.cs.purdue.edu/homes/li/cs352

This course studies how to *mechanically translate programs* which are written in a certain *programming language*.

- A programming language defines the components of programs.
- It defines the syntax form for each of such components.
- It assigns *semantics* to such forms.

A translator *implements* a programming language by analyzing a program written in that language and then

- either directly performing the semantic actions specified in the program
- or *transforming* it into a new version which is more suitable for a certain purpose, e.g.
  - being in a form ready to be loaded to the hardware and executed,
  - delivering higher performance when executed,
  - requiring less memory to store,

There exist two main classes of language translators:

- Interpreter: Analyzes a statement and execute it immediately. (Example: Java virtual machines, Unix shells)
- Compiler: Analyzes the whole program (or a whole “program unit”) before generating an equivalent but different version. Since the entire program or program unit is analyzed and “optimized”, a compiled program usually executes more efficiently than an interpreted one.

Most compilers analyze a program written in a high-level language and generate an equivalent program in a low-level language, e.g. the machine code.

Can you give examples of programming languages and language translators?

Abstract Syntax Tree

At the center of a modern compiler, there is the *internal representation* (IR) of the program, which takes the form of *abstract syntax trees* (ASTs), or in short, *syntax trees*.

The ASTs define the *operations* of a program.

Information about the *identifiers* is stored in the *symbol table*.

Let us look at at Figure 1.4. (in the textbook) for an example of AST.

Three Main Phases in a Modern Compiler

- Syntax analysis (converting the source code into ASTs and the symbol table)
- Semantic analysis (type checking, dataflow analysis)
- Code generation (including memory and register allocation)

In phase 1, the key concept is *context-free grammars* (CFG)
In phase 3, the key concept is machine and memory models.

Let us look at Figure 1.3 to see how the CFG is written and Figure 1.1 to get an overview of compiler phases.

∞

Why Studying Compiler Techniques?

- To better understand the designs of programming languages.
- To better understand program execution mechanism.
  - How does my code interact with the library routines?
  - Does the program error occur in my code or elsewhere?
- To be able to develop sophisticated user interfaces for software tools.