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 the meaning (i.e. the *semantics* to such form.

- A translator for the programming language analyzes a given program, and then
  - transforms it into another semantically equivalent program, or
  - directly performs the semantic actions specified in the program.

- In other words, a translator *implements* a programming language.

- Two main classes of language translators:
  - Interpreter: Analyzes a statement and execute it immediately. (Example: HTML browsers, JVMs.)
  - Compiler: Analyzes the whole program, then generates an equivalent program for later execution. This approach results in more efficient and more reliable codes, especially useful for performance-critical programs which will be executed many times and have long life-time.

- Conventionally, a compiler analyzes a program written in a high-level language and generates an equivalent program in a low-level language, e.g. machine code, or intermediate code.

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, memory allocation, dataflow analysis)
- Code generation

In phase 1, the key is the *context-free grammar* (CFG)

In phase 3, the key is the machine model.

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.

Course Work

- Written homework (15%)
- Midterm exam (19%)
- Final exam (30%)
- Projects (36%)

The home page of this course will be hosted by Purdue's Vista facility at www.itap.purdue.edu/ecourses/ Read class policy carefully once the course shows up on Vista. Check the home page frequently for announcements and lecture slides.

The scopes of the exams are fully covered by the lecture contents.