Compiler Principles and Techniques

ISSWHU, 2015

武漢大學 国际软件学院
International school of software

Lecturer

LI Li（李莉），
lli@whu.edu.cn

Office hours: 10:30-11:30am, Monday, 508, ISS Building

QQ Group: (pls change your name card): **333845131**

🕮 Prerequisites

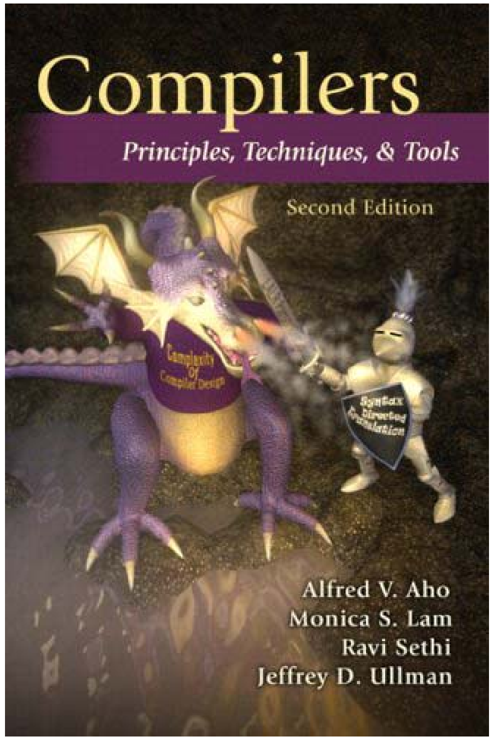
- ➢ **Data structure, Discrete Mathematics**

🕮 Required Textbook

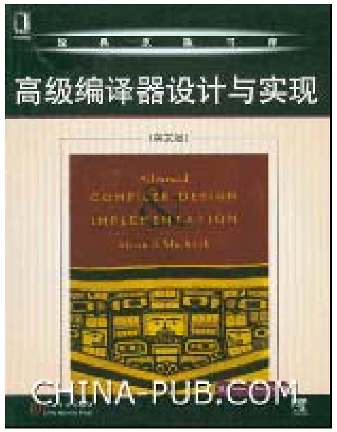
- ➢ **Compilers Principles, Techniques & Tools (Second Edition)**

  Alfred Aho., Monica S. Lam, Ravi Sethi, Jeffrey D. Ullman,

  Addison-Wesley, 2007


Compilers
Principles, Techniques, & Tools
Second Edition
Complexity of Compiler Design
Syntax Directed Translation
Alfred V. Aho
Monica S. Lam
Ravi Sethi
Jeffrey D. Ullman

🕮 Additional References



Compiler Construction Principles and Practice,
Kenneth C. Louden

Modern Compiler Implementation in Java, Second Edition, Andrew W. Appel, Cambridge University Press, 2002

Keith Schwarz , Stanford University,
http://web.stanford.edu/class/cs143/

Course Objectives

Programming Language Design

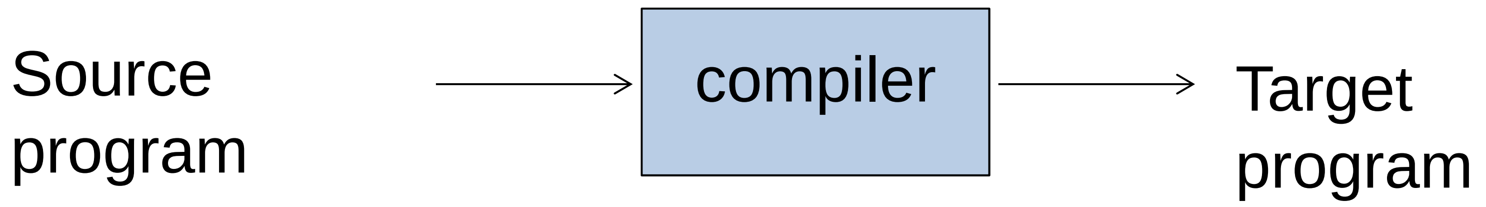
- Strengthens the understanding of the key concepts of programming languages
- Provides the theoretical foundation and implementation skills for designing and implementing programming languages.

Compilers

- principles
- techniques
- tools

What are Compilers

Compilers are computer programs that translate one language to another.



A compiler is a fairly complex program that can be anywhere from 10,000 to 1,000,000 lines of code.

Compilers are used in almost all forms of computing, and anyone professionally involved with computers should know the basic organization and operation of a compiler

Programming Languages

A programming language is a notation for specifying computational tasks that a person can understand and a computer can execute.

Every programming language has a syntax and semantics.

–The syntax specifies how a concept is expressed.

–The semantics specifies what the concept means or does.

What does this C program do?

```
#include <stdio.h>
int main ( ) {
  int i, j;
  i = 1;
  j = i++ + ++i;
  printf("%d\n", j);
}
```

- “ The world of compiler design has changed significantly. Programming languages have evolved to present new compilation problems. Computer architectures offer a variety of resources of which the compiler designer must take advantage. Perhaps most interestingly, the venerable technology of code optimization has found use outside compilers. It is now used in tools that find bugs in software, and most importantly, find security holes in existing code. And much of the "front-end" technology - grammars, regular expressions, parsers, and syntax-directed translators – are still in wide use.”

- “ We recognize that few readers will build, or even maintain, a compiler for a major programming language. Yet the models, theory, and algorithms associated with a compiler can be applied to a wide range of problems in software design and software development . We therefore emphasize problems that are most commonly encountered in designing a language processor, regardless of the source language or target machine. ”

WHY study compilers ? (1)

- Parsers and interpreters are everywhere
- Seeing the development of a compiler gives you a feeling for how programs work that may help you understand the internal process of program execution deeply
- Compilers (ideally) have three parts:
  - Language-dependent frontend (parsing, type-checking)
  - Language and target independent middle end (optimization)
  - Target-dependent backend (code generation)

WHY study compilers ? (2)

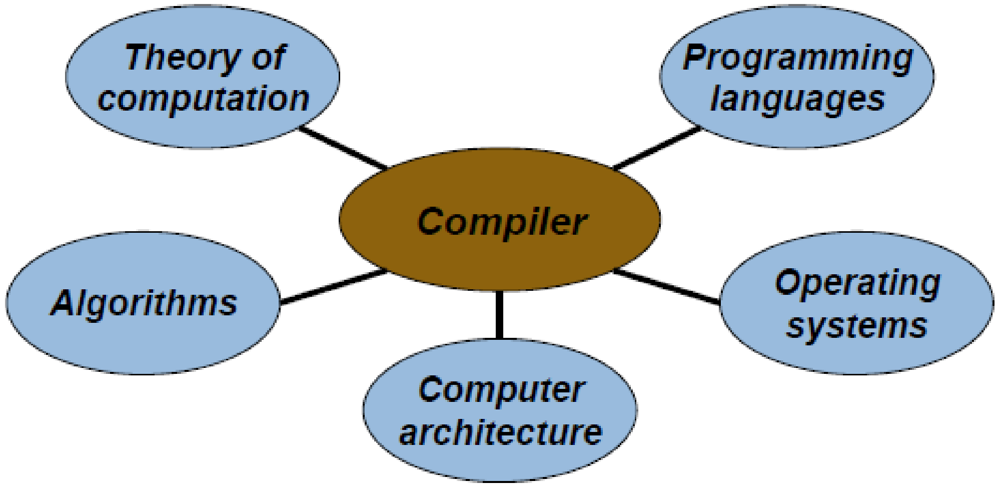
- Many algorithms and models you will use in compilers are fundamental, and will be useful elsewhere:
  - automata, regular expressions (scanning)
  - context-free grammars, trees (parsing)
  - hash tables (symbol table)
  - dynamic programming, graph coloring (code generation)

WHY study compilers ? (3)

- To program more efficient programs
  - Compiler writing spans programming languages, machine architecture, language theory, algorithms, and software engineering.
  - The ideas behind principles and techniques of a compiler writing can be used many times in the career of a computer scientist.
- You will be able to write faster code
- You will be able to write correct code

Hope you

- Understand how compilers work
- See how theory and practice work together
- Learn to think about tradeoffs
- Bring many parts of CS together



Course Content

- Introduction
- A simple syntax-directed translator
- Lexical analysis - Scanning
- Context-free Grammars and Parsing
- Syntax analysis
  - Top-down parsing
  - Bottom-up parsing
- Semantics Analysis
- Runtime Environments
- Code Optimization
- Code Generation

Programming Assignments

- **3 programming assignments corresponding with compiler phases learning**
- **Programs documentation and testing will be required for each assignment.**

Grading Policies

Assignments (30%)

Quizzes(10%) : 5~10 times in the class (Attendance)

Mid-Term Exam (10%)

Final Exam (50%)

Academic Policy

- **Plagiarism will be taken seriously**. Students found to be adopting unfair means of any kind will be severely dealt with.
- All assignments and projects must be submitted on time. **No late submission will be accepted**.
- If you are going to miss the midterm/final due to unavoidable circumstances, inform me before the exam.