Compiler Design

Lecture 1:
Course Overview

Dr. Momtazi
momtazi@aut.ac.ir

based on the slides of the course book

Outline

- **Administrative Information**
- Introduction to the Course
- Overview of the Semester

Course Home Page

- Administrative information
- Slides
- Exercises

Assessment

- Regular attendance in the class
- Final exam (40%)
- Midterm exam (30%)
- Exercises and project (30%)

Contact

- Email: momtazi@aut.ac.ir
- Phone: 021-64542737

Text Book

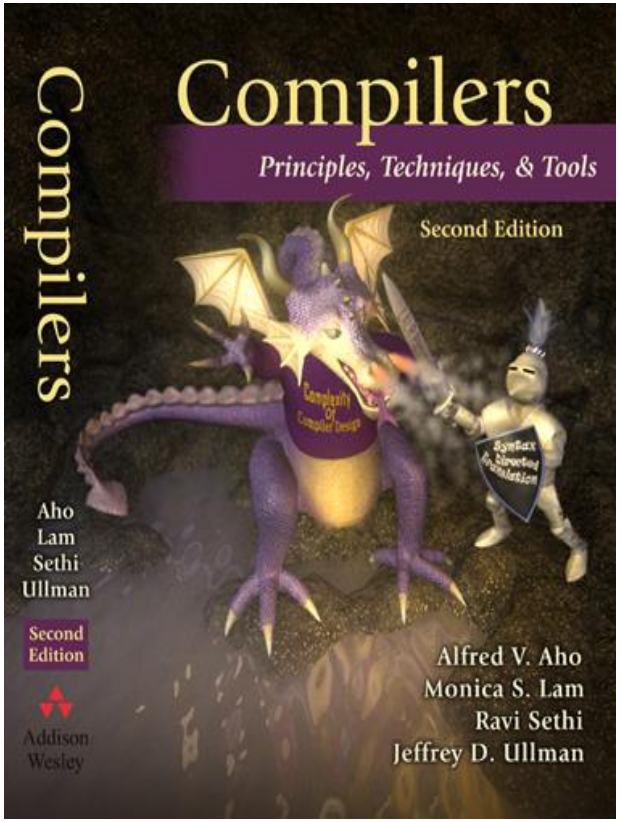
Compilers:

Principles, Techniques & Tools

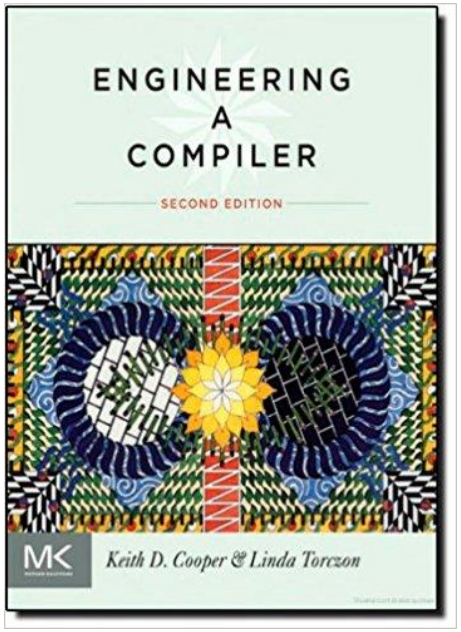
By Aho, Lam, Sethi, and Ullman

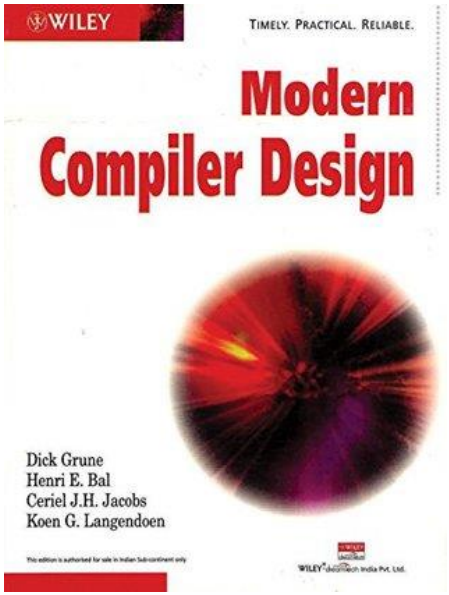
2nd ed., 2007

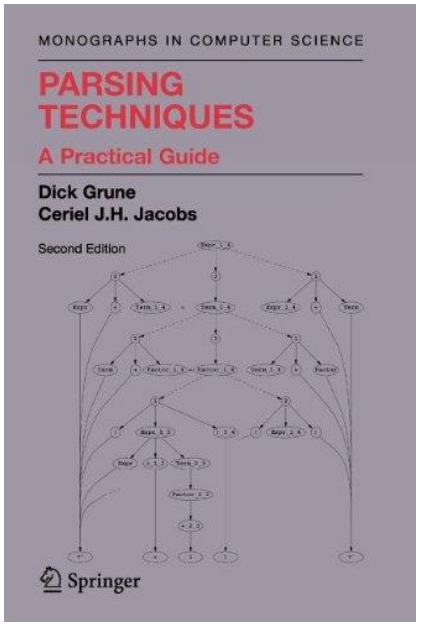
Publisher: Addison-Wesley



Other Related Books







- "*Engineering a Compiler*", Cooper & Torczon, 2nd edition, 2011, Elsevier
- *"Modern Compiler Design*" , Grune *et al.* , 2000, Wiley
- *"Parsing Techniques: A Practical Guide*", Grune and Jacobs, 2nd edition, 1998, Springer

Rules of the Game

- In case you don’t understand something:
  - Ask!!!
  - Ask!!!
  - Ask!!!

Outline

- Administrative Information
- **Introduction to the Course**
- Overview of the Semester

Course Goal

- Any program written in a programming language must be translated before it can be executed.
- This translation is typically accomplished by a software system called compiler.
- This course aims to introduce students to the principles and techniques used to perform this translation and the issues that arise in the construction of a compiler.

Learning Outcomes

- Understanding the principles governing all phases of the compilation process.
- Understanding the role of each of the basic components of a standard compiler.
- Showing awareness of the problems and methods and techniques applied to each phase of the compilation process.
- Applying standard techniques to solve basic problems that arise in compiler construction.
- Understanding how the compiler can take advantage of particular processor characteristics to generate good code.

Terminology

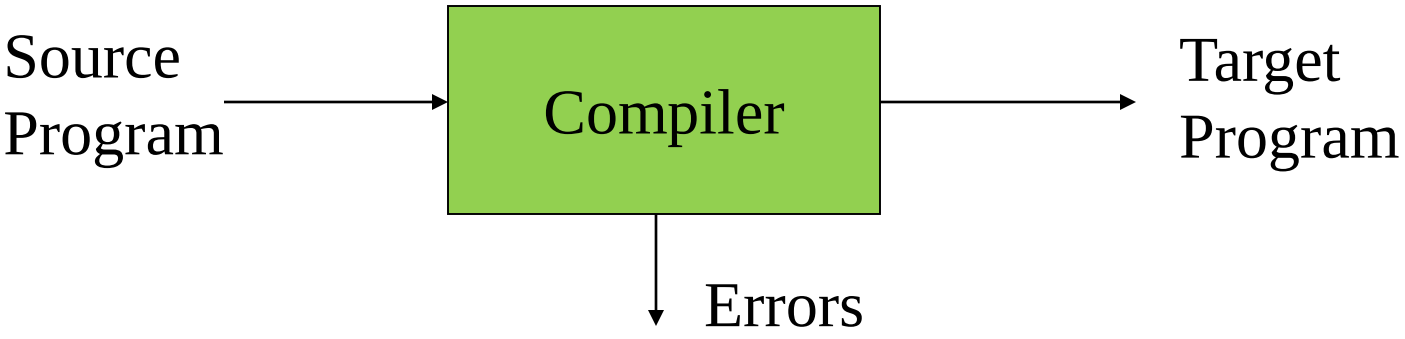
- Compiler:
  - A program that translates an *executable* program in a *source language* (usually high level)
    into an equivalent *executable* program in a *target language* (usually low level) while preserving the meaning of that text

Terminology

- Interpreter:
  - A program that reads an *executable* program and produces the results of running that program
  - Usually, this involves executing the source program in some fashion

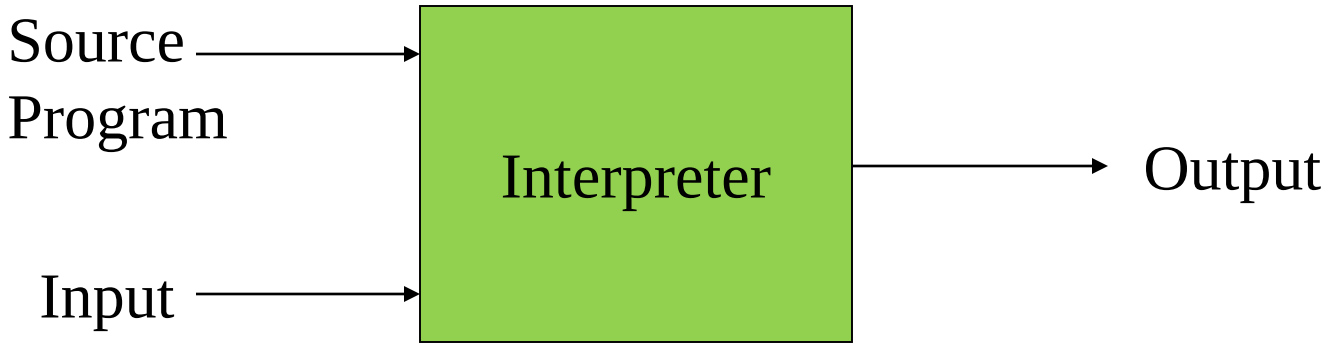
- Our course is mainly about compilers but many of the same issues arise in interpreters too.

A Compiler


Source
Program
Compiler
Target
Program
Errors

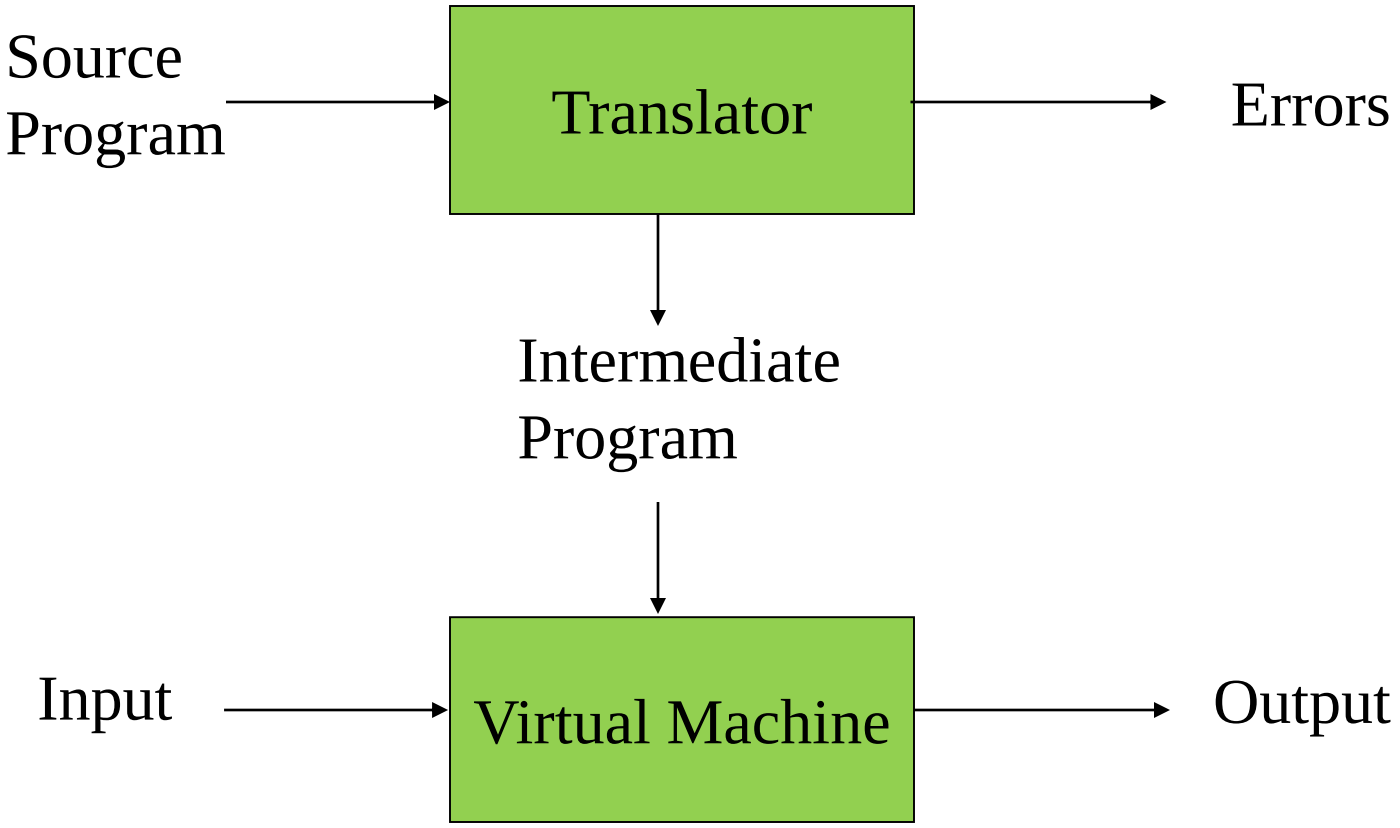

Input
Target Program
Output

An Interpreter



- Translates line by line
- Executes each translated line immediately
- Execution is slower because translation is repeated
- But, usually give better error diagnostics than a compiler

A Hybrid Compiler



Examples

- C is typically compiled
- Lisp is typically interpreted
- Java is compiled to bytecodes, which are then interpreted

Expected Qualities of a Compiler

- Generating correct code (first and foremost!)
- Generating fast code
- Conforming to the specifications of the input language
- Coping with essentially arbitrary input size, variables, etc
- Working well with the debugger

Expected Qualities of a Compiler (cont.)

- Compilation time (linearly) proportional to size of source
- Good diagnostics
- Consistent optimizations

Principles of Compilation

- *The compiler must*:
  - Preserve the meaning of the program being compiled.
  - Improve the source code in some way.

Principles of Compilation

- *Other issues (depending on the setting):*
  - Speed (of compiled code)
  - Space (size of compiled code)
  - Feedback (information provided to the user)
  - Debugging (transformations obscure the relationship source code vs target)
  - Compilation time efficiency (fast or slow compiler?)

Compiler Learning

- Isn’t it an old discipline?
  - Yes, it is a well-established discipline
  - Algorithms, methods and techniques were developed in early stages of computer science
  - There are many compilers around, and many tools to generate them automatically

Compiler Learning

- So, why we need to learn it?
  - Although you may never write a full compiler
  - But the techniques we learn is useful in many tasks like:
    - Writing an interpreter for a scripting language
    - Validation checking for forms, and so on

Compiler Learning

- Success stories (one of the earliest branches in CS)
  - Applying theory to practice (scanning, parsing, static analysis)
  - Many practical applications have embedded languages (eg, tags)

- Practical algorithmic & engineering issues:
  - Approximating really hard (and interesting!) problems
  - Emphasis on efficiency and scalability
  - Small issues can be important!

Compiler Learning

- Ideas from different parts of computer science are involved:
  - AI: Heuristic search techniques
  - Algorithms: graph algorithms
  - Theory: pattern matching
  - Also: Systems, Architecture

- Compiler construction can be challenging and fun:
  - New architectures always create new challenges;
  - Success requires mastery of complex interactions; results are useful;
  - Opportunity to achieve performance.

Uses of Compiler Technology

- Most common use: translate a high-level program to object code
  - Program Translation: binary translation, hardware synthesis, …
- Optimizations for computer architectures:
  - Improve program performance, take into account hardware parallelism, etc…
- Automatic parallelization or vector representation

Uses of Compiler Technology

- Performance instrumentation
  - e.g., -pg option of cc or gcc
- Interpreters
  - e.g., Python, Ruby, Perl, Matlab, Shell, …
- Software productivity tools
  - Debugging aids: e.g, purify

Uses of Compiler Technology

- Security: Java VM uses compiler analysis to prove "safety" of Java code.

- Text formatters, just-in-time compilation for Java, power management, global distributed computing, …

Key: Ability to extract properties of a source program (analysis) and transform it to construct a target program (synthesis)

Summary

- A compiler is a program that converts some input text in a source language to output in a target language.

- Compiler construction poses some of the most challenging problems in computer science.

Outline

- Administrative Information
- Introduction to the Course
- **Overview of the Semester**

Syllabus

- Introduction
- Lexical Analysis (scanning)
- Syntax Analysis (parsing)
- Semantic Analysis
- Intermediate Representations
- Storage Management
- Code Generation
- Code Optimisation

Question?