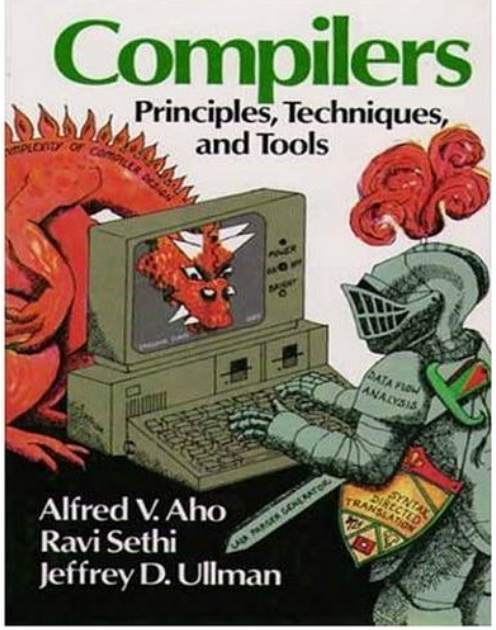
THEORY OF COMPUTATION AND COMPILERS

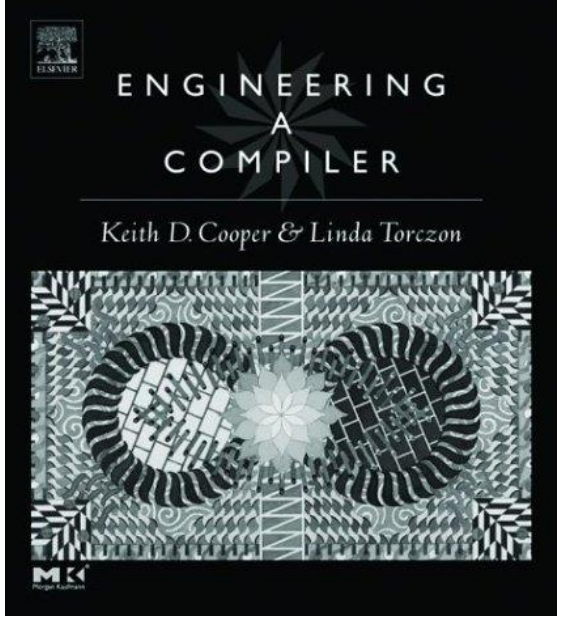
Unit - I | Part-2

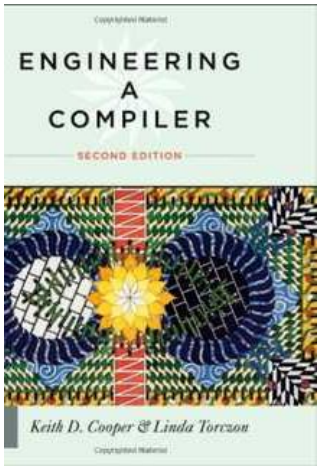
OVERVIEW OF COMPILATION

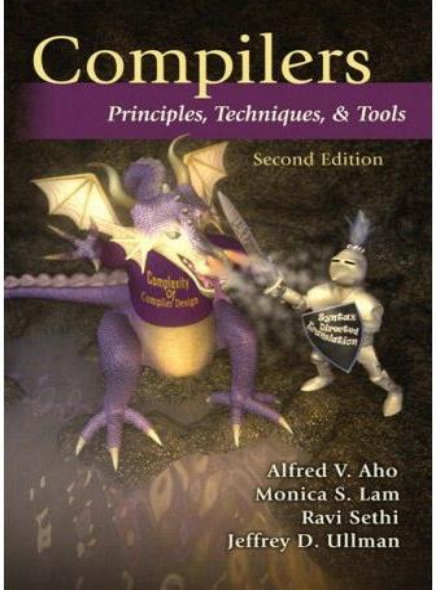
- Phases
- Lexical Analysis
- Lex Specifications
- Structure of a Lex Specification File
- Regular Grammar and Regular Expression for Common Programming Language Features
- Pass and Phases of Translation
- Interpretation
- Bootstrapping
- Data Structures of Compiler
- LEX tool

Dr. R. Madana Mohana
Professor, Artificial Intelligence & Data Science | I/c-Head, Artificial Intelligence & Machine Learning
CHAITANYA BHARATHI INSTITUTE OF TECHNOLOGY
Hyderabad - 500 075, Telangana, INDIA
www.cbit.ac.in









Course Texts

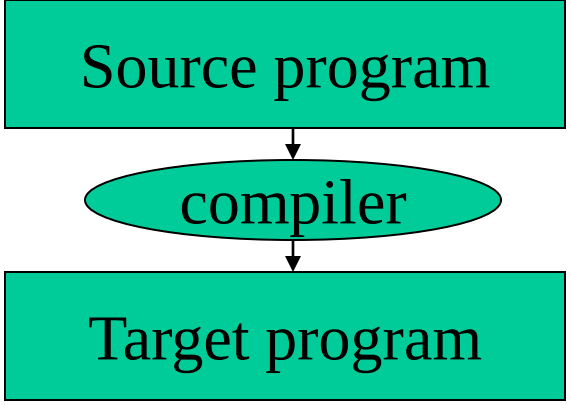
- **Aho, Lam, Sethi, Ullman. "Compilers: Principles, Techniques and Tools", 2nd edition**. (Aho2) The **1st edition** (by Aho, Sethi, Ullman – Aho1), the "Dragon Book", has been a classic for over 20 years.
- **Cooper & Torczon. "Engineering a Compiler"** – an earlier draft has been consulted when preparing this module. The 2nd edition is now available and being assessed (pointers will be provided to the 1st and hopefully to the 2nd edition).
- Other books:
  - Hunter *et al*. "The essence of Compilers" (Prentice-Hall)
  - Grune *et al*. "Modern Compiler Design" (Wiley)

Definitions

(compile: collect material into a list, volume)

- What is a compiler?
  - A program that accepts as input a program text in a certain language and produces as output a program text in another language, while preserving the meaning of that text (Grune *et al*, 2000).
  - A program that reads a program written in one language (source language) and translates it into an equivalent program in another language (target language) (Aho *et al*)
- What is an interpreter?
  - A program that reads a source program and produces the results of executing this source.
- *We deal with compilers! Many of these issues arise with interpreters!*

Examples



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

Also:

- C++ to Intel Core 2/.../Assembly
- C++ to C
- High Performance Fortran (HPF) to Fortran (parallelising compiler)
- C to C (or any language to itself)

In the general sense:

- What is LaTeX?
- What is ghostview? (PostScript is a language for describing images)

Qualities of a Good Compiler

What qualities would you want in a compiler?

- generates correct code (first and foremost!)
- generates fast code
- conforms to the specifications of the input language
- copes with essentially arbitrary input size, variables, etc.
- compilation time (linearly)proportional to size of source
- good diagnostics
- consistent optimisations
- works well with the debugger

Principles of Compilation

The compiler must:

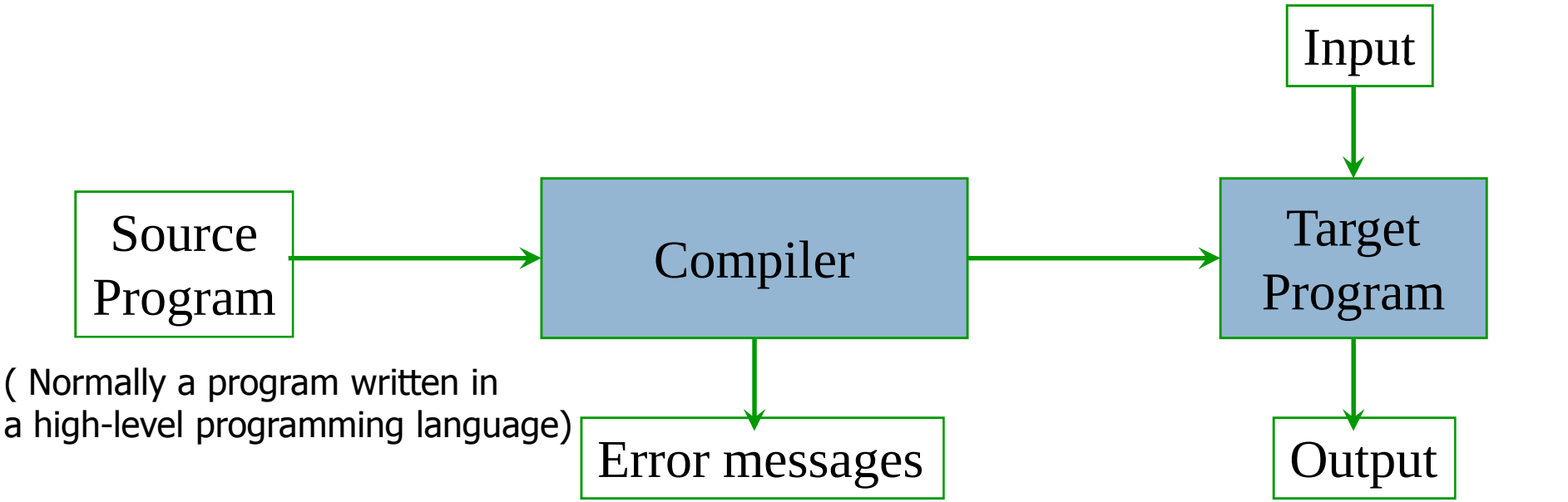
- *preserve the meaning of the program being compiled.*
- *"improve" the source code in some way.*

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?)

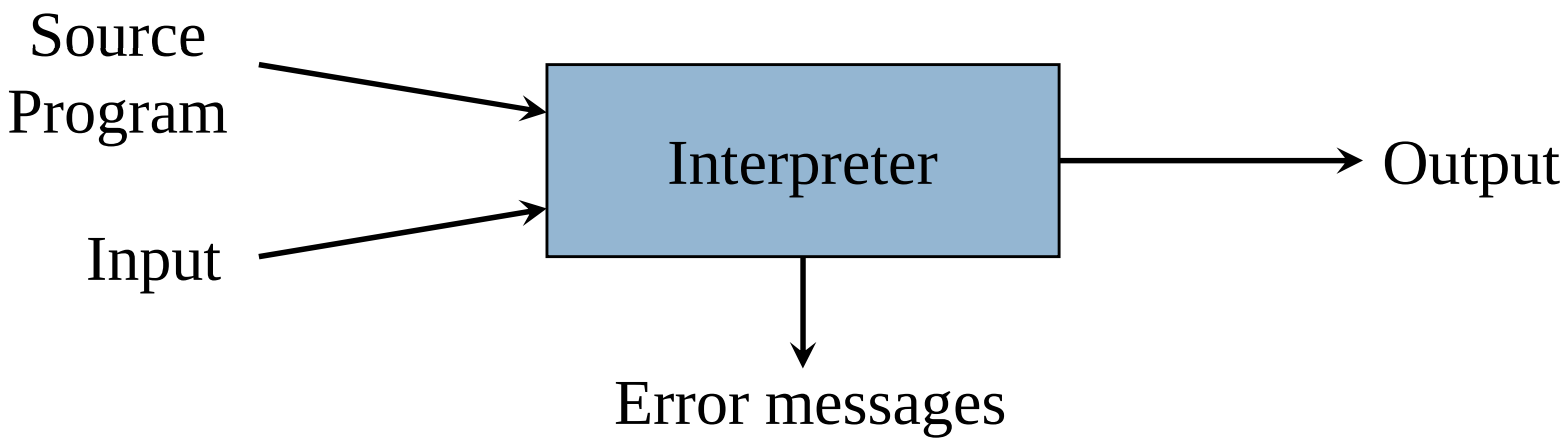
Compilers and Interpreters

- "Compilation"
  - Translation of a program written in a source language into a semantically equivalent program written in a target language



Compilers and Interpreters (cont'd)

- "Interpretation"
  - Performing the operations implied by the source program



Difference Between Compiler & Interpreter

Compiler	Interpreter
1. Compiler scans the entire HLL program at once.	1.Interpreter translate one statement at a time.
2. The translation process carried out by the compiler is called as Compiler.	2.The translation process carried out by the interpreter is termed
3. If program is error free, it compiles the program in to machine language which has to be executed by an interpreter. i.e, object program executed is not carried out by the compiler.	3.If program is error free, it executes the program and continues till the last statement. i.e., object program execution is carried out by the interpreter.
4. If process the program statements in their physical input sequence	4. It process according to logical flow of control through the program
5. If process each statement exactly once	5. Some statements may execute repeatedly
6. Processing time is less	6. Processing time is more
7. Compilers are large in size and they occupy more memory.	7. Interpreter are smaller than compilers.
8. Error identification is difficults.	8. easy
9. This is also called as s/w translator	9.This is also called as s/w simulation
10. Ex:- FORTRAN, PASCAL, C,C++	10. LISP, Prolog, Smalltalk etc.

Other Applications of Compiler Design

- In addition to the development of a compiler, the techniques used in compiler design can be applicable to many problems in computer science.
  - Techniques used in a lexical analyzer can be used in text editors, information retrieval system, and pattern recognition programs.
  - Techniques used in a parser can be used in a query processing system such as SQL.
  - Many software having a complex front-end may need techniques used in compiler design.
    - A symbolic equation solver which takes an equation as input. That program should parse the given input equation.
  - Most of the techniques used in compiler design can be used in Natural Language Processing (NLP) systems.

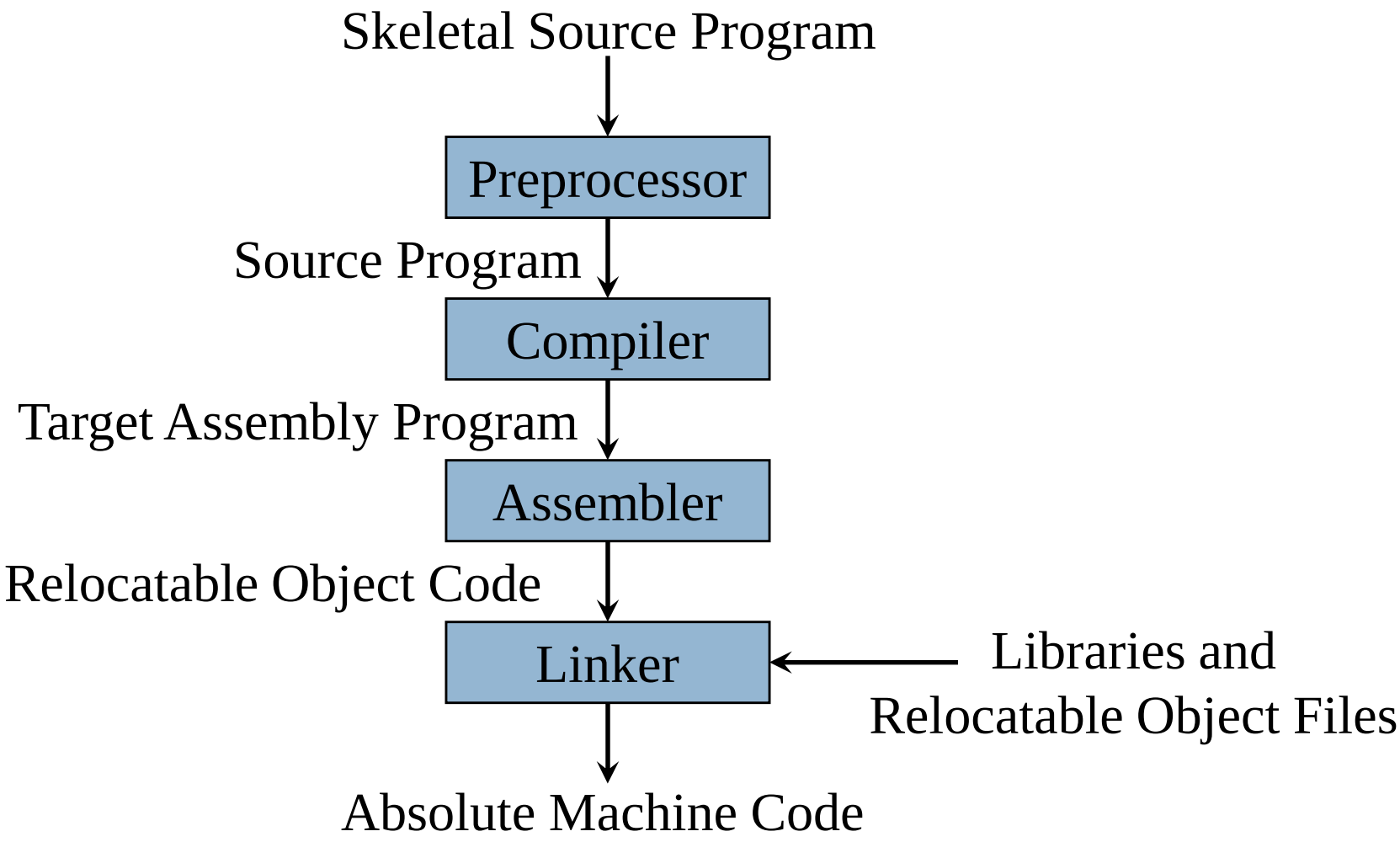
The Analysis - Synthesis Model of Compilation

- There are two parts to compilation:
  - *Analysis* determines the operations implied by the source program which are recorded in a tree structure
  - In analysis phase, an intermediate representation is created from the given source program.
    - Lexical Analyzer, Syntax Analyzer and Semantic Analyzer are the parts of this phase
  - *Synthesis* takes the tree structure and translates the operations therein into the target program
  - In synthesis phase, the equivalent target program is created from this intermediate representation.
    - Intermediate Code Generator, Code Generator, and Code Optimizer are the parts of this phase.

Other Tools that Use the Analysis-Synthesis Model

- *Editors* (syntax highlighting)
- *Pretty printers* (e.g. Doxygen)
- *Static checkers* (e.g. Lint and Splint)
- *Interpreters*
- *Text formatters* (e.g. TeX and LaTeX)
- *Silicon compilers* (e.g. VHDL)
- *Query interpreters/compilers* (Databases)

Preprocessors, Compilers, Assemblers, and Linkers



Why study Compilation Technology?

- 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!
- Ideas from different parts of computer science are involved:
    - AI: Heuristic search techniques; greedy algorithms - 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 parallelisation or vectorisation
- Performance instrumentation: e.g., -pg option of cc or gcc
- Interpreters: e.g., Python, Ruby, Perl, Matlab, sh, …
- Software productivity tools
  - Debugging aids: e.g, purify
- 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)

Applications of Compiler Technology & Tools

- Processing XML/other to generate documents, code, etc.
- Processing domain-specific and device-specific languages.
- Implementing a server that uses a protocol such as http or imap
- Natural language processing, for example, spam filter, search, document comprehension, summary generation
- Translating from a hardware description language to the schematic of a circuit
- Automatic graph layout (graphviz, for example)
- Extending an existing programming language
- Program analysis and improvement tools

Other Applications

- In addition to the development of a compiler, the techniques used in compiler design can be applicable to many problems in computer science.
  - Techniques used in a lexical analyzer can be used in text editors, information retrieval system, and pattern recognition programs.
  - Techniques used in a parser can be used in a query processing system such as SQL.
  - Many software having a complex front-end may need techniques used in compiler design.
    - A symbolic equation solver which takes an equation as input. That program should parse the given input equation.
  - Most of the techniques used in compiler design can be used in Natural Language