Compiler Construction

Chapter-1

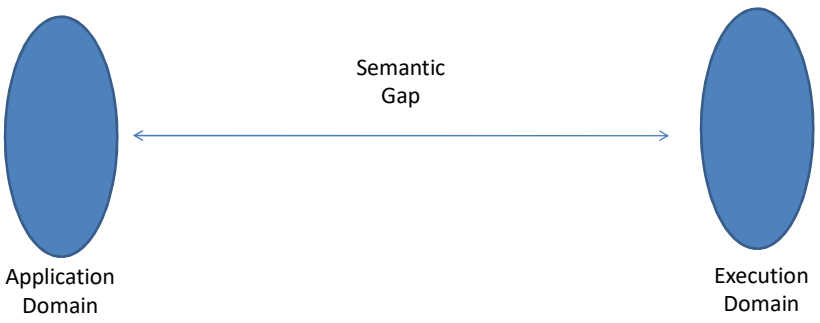
Text Books

- Compilers (Principles, Techniques & Tools)
  - **Second Edition, PEARSON**
  - Alfred V. Aho, Monica S. Lam
  - Ravi Sethi, Jeffrey D. Ullman
- Compiler Writing (20 % Syllabus)
  - Trembly & Sorenson

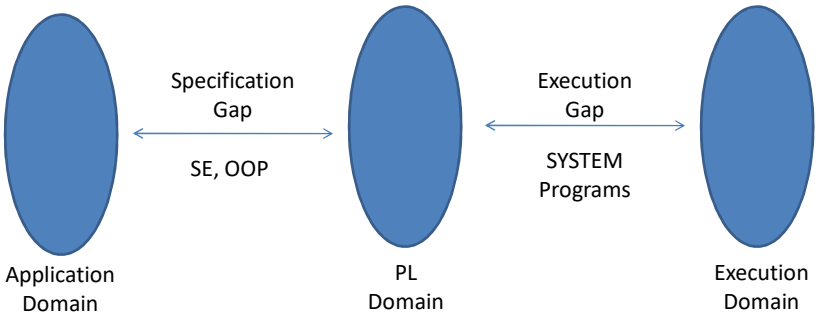
Language Processor

- Migraters
- Assemblers
- Pre-processors
- Interpreters (BASIC)
- Hybrid Compiler (Java)
- **Compilers**

Need: Language Processing

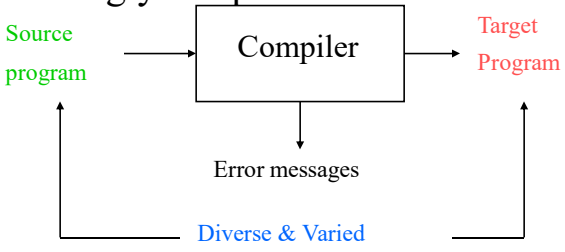


Need: Language Processing



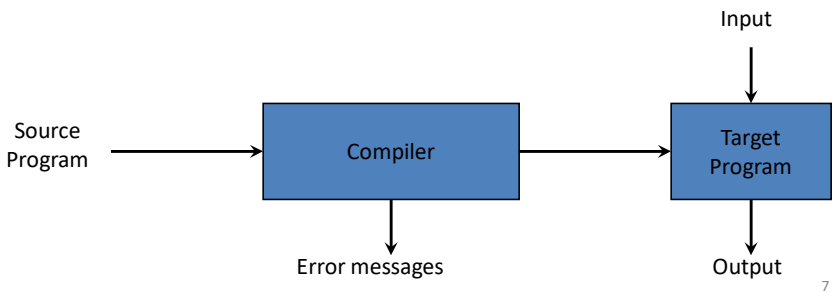
Language Processor

- As a Discipline, Involves Multiple CS&E Areas
  - Programming Languages and Algorithms
  - Theory of Computing & Software Engineering
  - Computer Architecture & Operating Systems
- Has Deceivingly Simplistic Intent:



Compilers and Interpreters

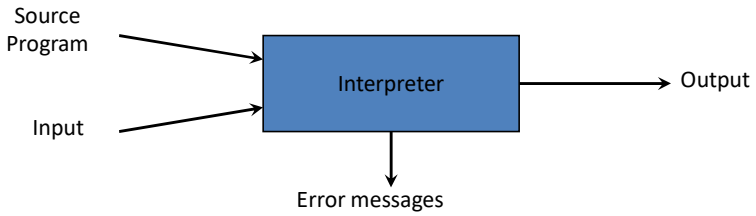
- "*Compilation*"
  - Translation of a program written in a source language into a semantically equivalent program written in a target language



7

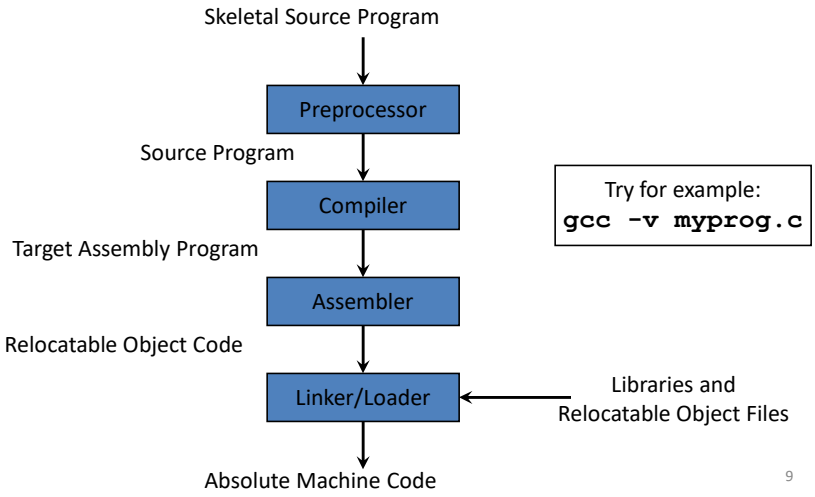
Compilers and Interpreters (cont'd)

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



8

Preprocessors, Compilers, Assemblers, and Linkers

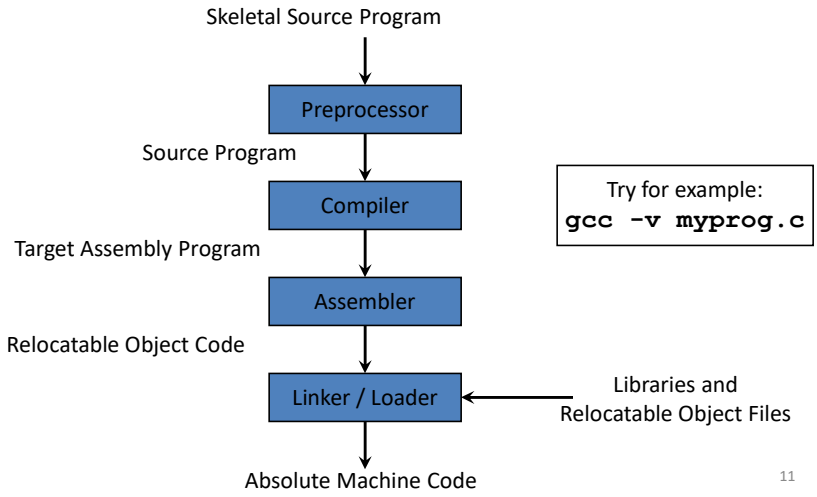


9

Tasks of Preprocessor

- Macro Processing
- File Inclusion
- Rational Preprocessors
- Language Extensions for a Database Systems

Preprocessors, Compilers, Assemblers, and Linkers



Tasks of Linker & Loader

- Link-editor: taking many (relocatable) machine code programs (with cross-references) and produce a single file.
  - Need to keep track of correspondence between variable names and corresponding addresses in each piece of code.
- Loader: taking relocatable machine code, altering the addresses and placing the altered instructions into memory.

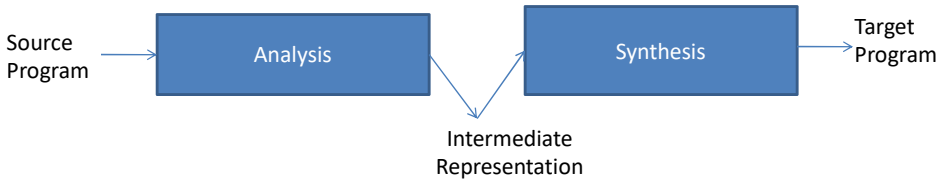
The Analysis-Synthesis Model of Compilation

- There are two parts to compilation:
  - *Analysis* determines the operations implied by the source program which are recorded in a tree structure
  - *Synthesis* takes the tree structure and translates the operations therein into the target program

13

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
  - *Synthesis* takes the tree structure and translates the operations therein into the target program

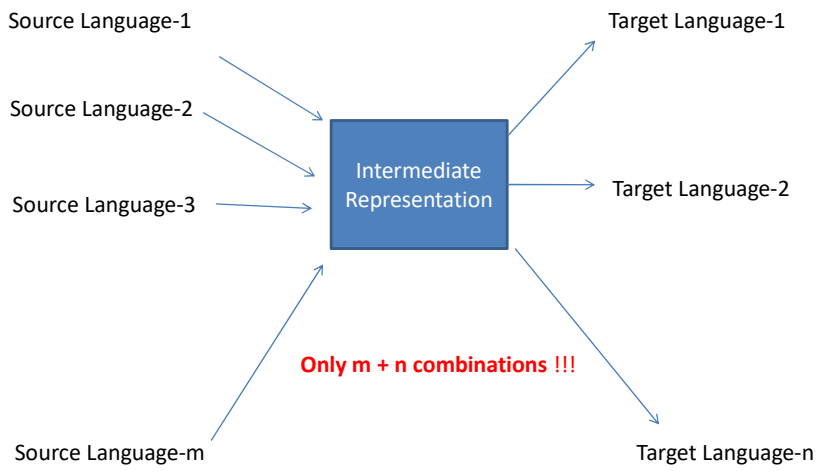


14

What is the Main Advantage of Analysis-Synthesis Model ?

- Let's consider Python Language as source Lang.
- Let's consider Assembly languages as target Languages: 8086,8085,80386 ……(n assemblers)
- We require n compilers for one source langauge.
- In this way, for m different source languages we require **m*n different compilers** !!!

What is the Main Advantage of Analysis-Synthesis Model ?



Analysis Part of Compilation

- Three Phases:
  - Linear / Lexical Analysis:
    - L-to-R Scan to Identify Tokens
      token: sequence of chars having a collective meaning
  - Hierarchical Analysis (Syntax Analysis):
    - Grouping of Tokens Into Meaningful Collection
  - Semantic Analysis:
    - Checking to ensure Correctness of Components

Synthesis Part of Compilation

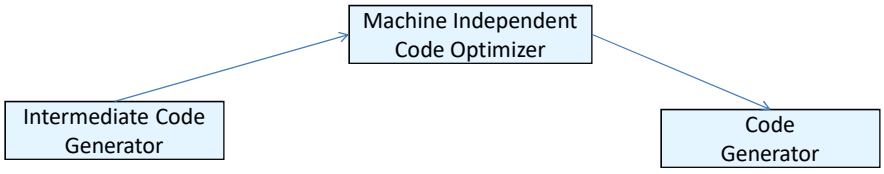
- Intermediate Code Generation
  - Abstract Machine Version of Code - Independent of Architecture
    - Easy to Produce and
    - Easy to translate into target program
- Code Optimization
  - Find More Efficient Ways to Execute Code
  - Replace Code With More Optimal Statements
- Final Code Generation
  - Generate Relocatable Machine Dependent Code

Synthesis Part of Compilation

- Code Optimizer (Optional Phase)
  - Machine Independent Code Optimizer
  - Machine Dependent Code Optimizer
- Code Generation

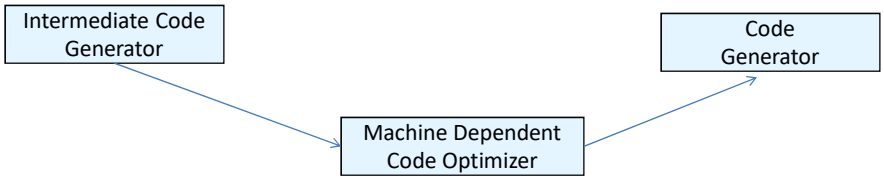
Synthesis Part of Compilation

- Code Optimizer (Optional Phase)
  - Machine Independent Code Optimizer
  - Machine Dependent Code Optimizer
- Code Generation



Synthesis Part of Compilation

- Code Optimizer (Optional Phase)
  - Machine Independent Code Optimizer
  - Machine Dependent Code Optimizer
- Code Generation



Phases of Compiler