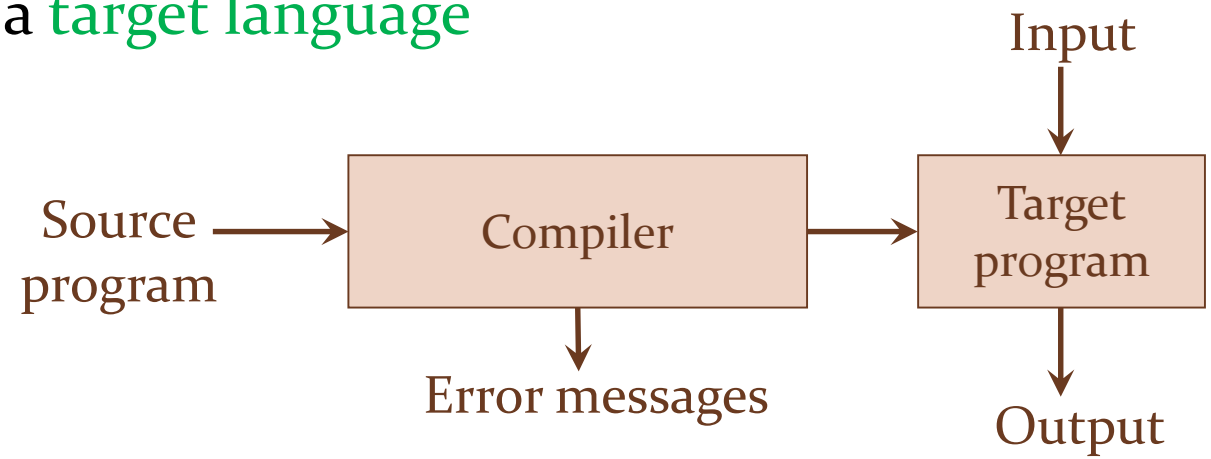
Chapter 1

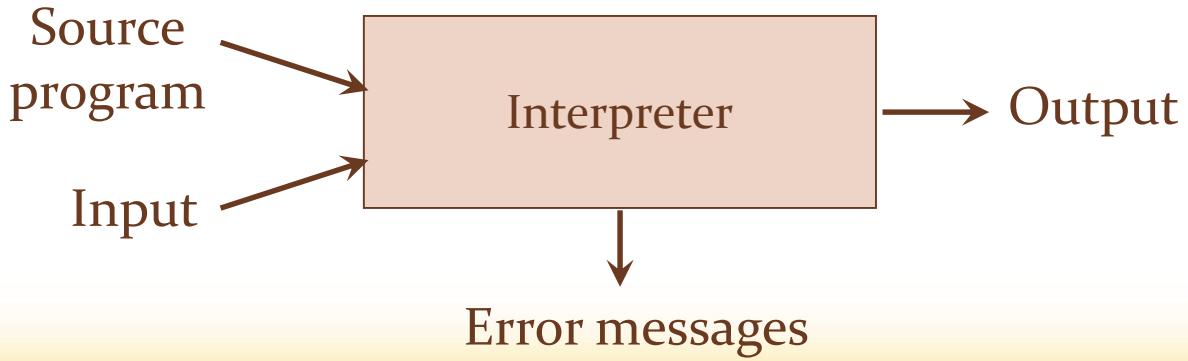
Introduction to Compiler Construction

Compilers and Interpreters

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



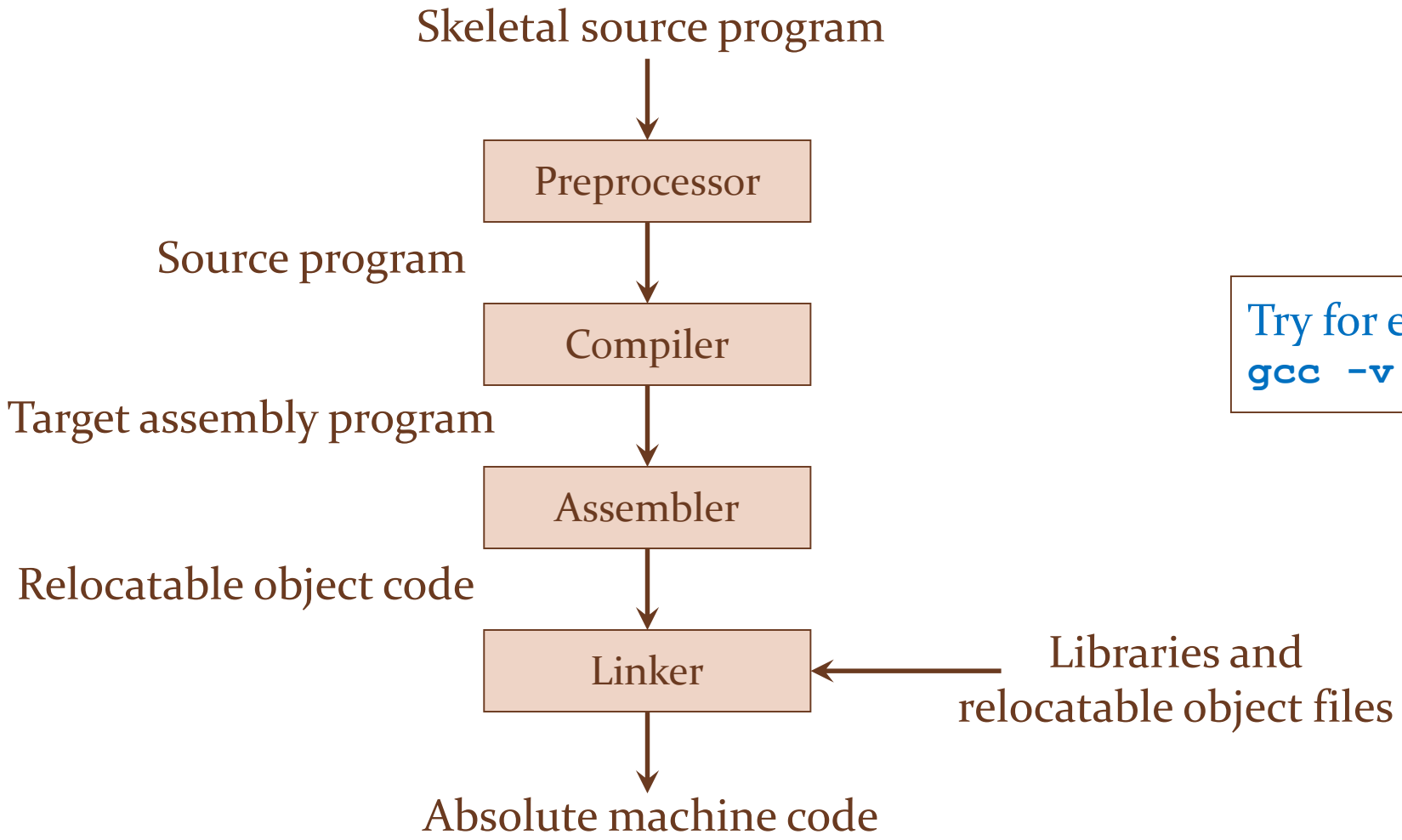
- Interpretation
  - Performing the operations implied by the source program



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
- 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)

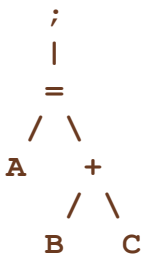
Preprocessors, Compilers, Assemblers, and Linkers



Try for example:

```
gcc -v myprog.c
```

The Phases of a Compiler

Phase	Output	Sample
Programmer (source code producer)	Source string	`A=B+C;`
Scanner (performs *lexical analysis*)	Token string	`'A', '=', 'B', '+', 'C', ';'` And *symbol table* with names
Parser (performs *syntax analysis* based on the grammar of the programming language)	Parse tree or abstract syntax tree	
Semantic analyzer (type checking, etc)	Annotated parse tree or abstract syntax tree	

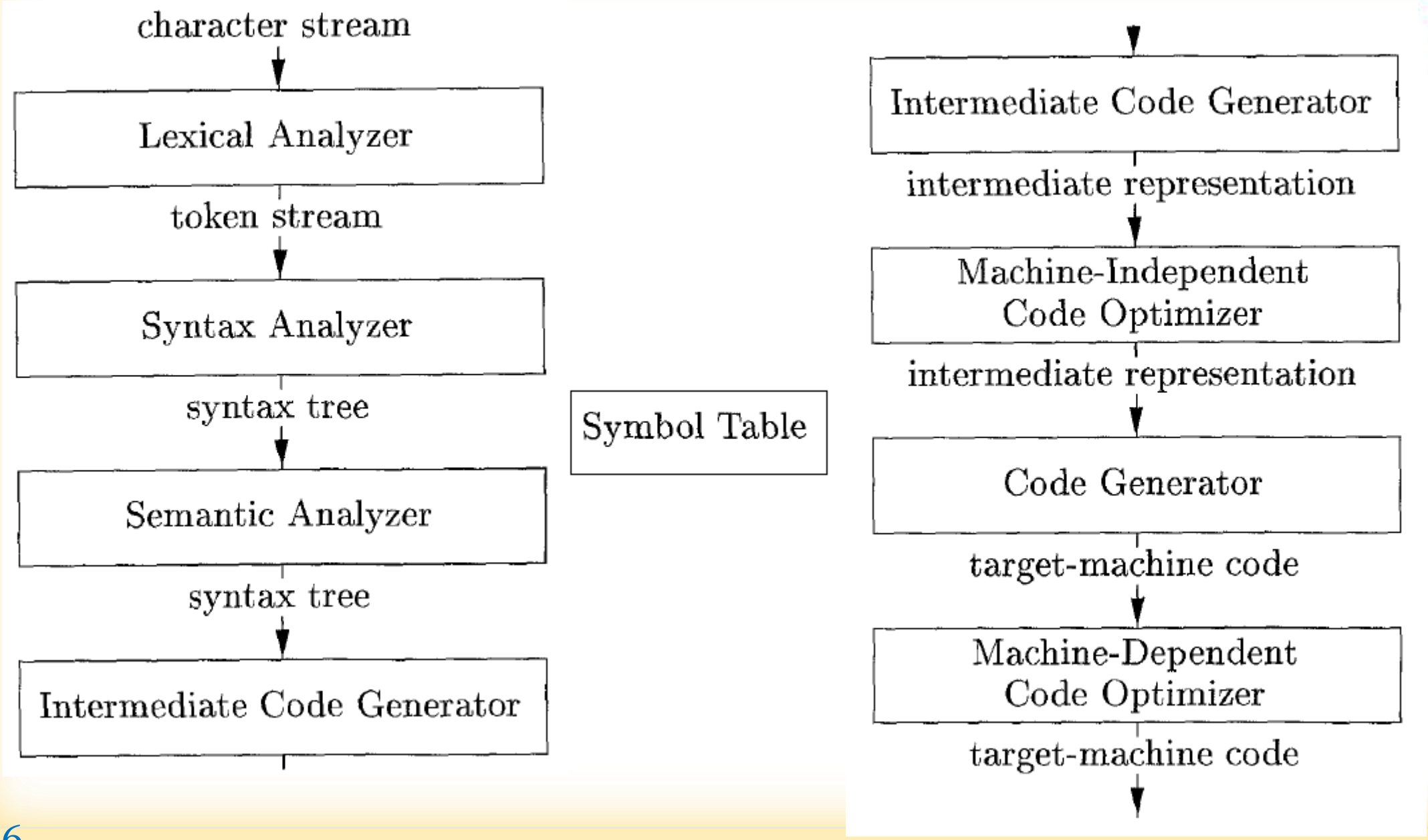
Intermediate code generator	Three-address code, quads, or RTL	`int2fp B t1`
		`+ t1 C t2`
		`= t2 A`

Optimizer	Three-address code, quads, or RTL	`int2fp B t1` `+ t1 #2.3 A`

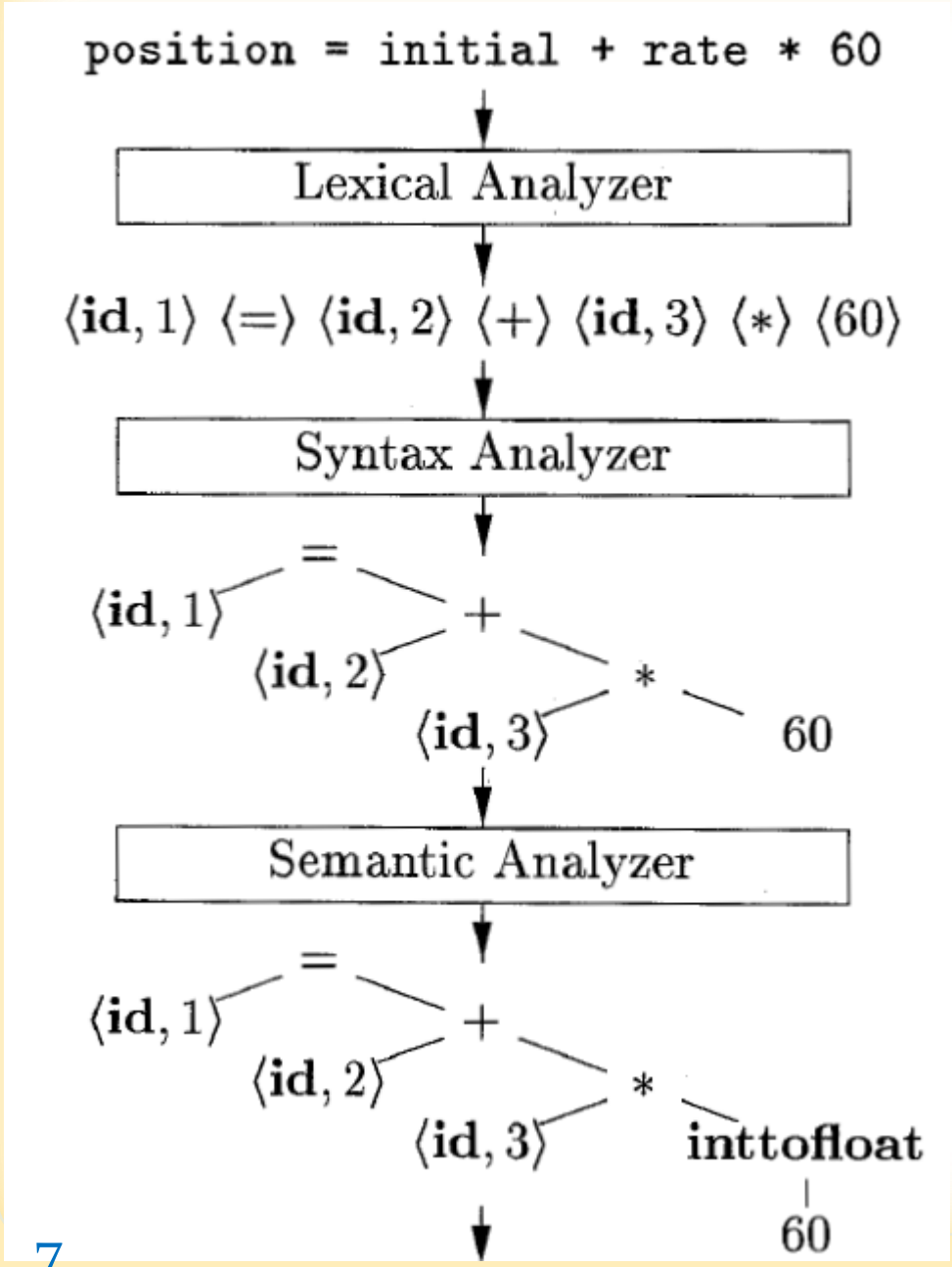
Code generator	Assembly code	`MOVF #2.3,r1`
		`ADDF2 r1,r2`
		`MOVF r2,A`

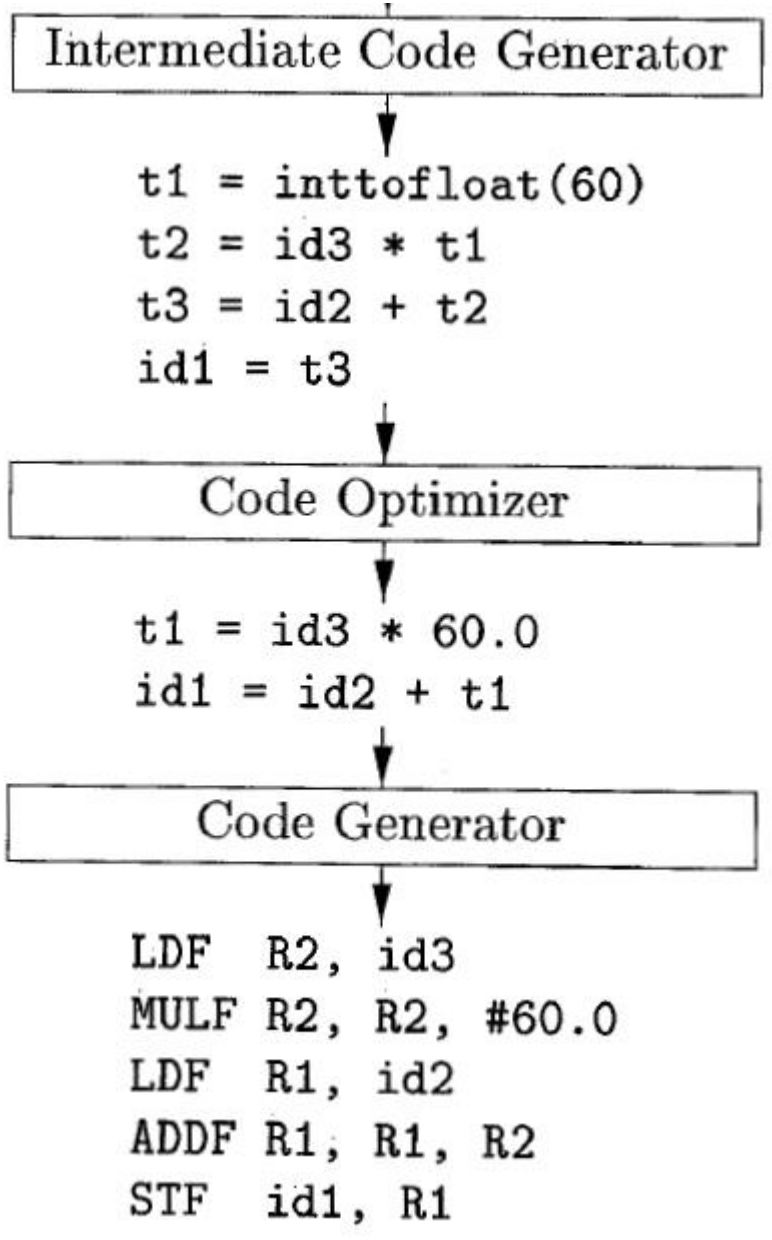
Peephole optimizer	Assembly code	`ADDF2 #2.3,r2` `MOVF r2,A`

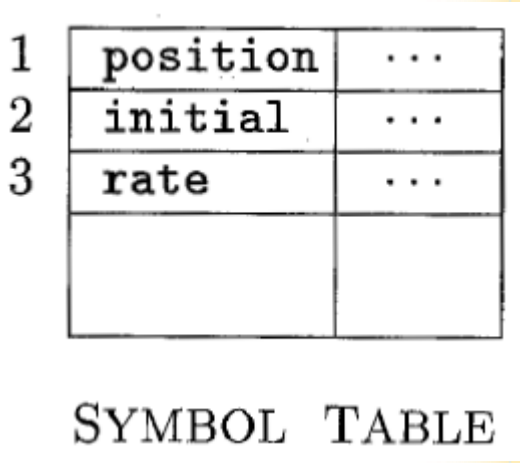
The Phases of a Compiler



The Phases of a Compiler







The Grouping of Phases

- Compiler *front* and *back ends*:
  - Front end: *analysis* (*machine independent*)
  - Back end: *synthesis* (*machine dependent*)
- Compiler *passes:*
  - A collection of phases is done only once (*single pass*) or multiple times (*multi pass*)
    - Single pass: usually requires everything to be defined before being used in source program
    - Multi pass: compiler may have to keep entire program representation in memory

Compiler-Construction Tools

- Software development tools are available to implement one or more compiler phases
  - *Scanner generators*
  - *Parser generators*
  - *Syntax-directed translation engines*
  - *Automatic code generators*
  - *Data-flow engines*