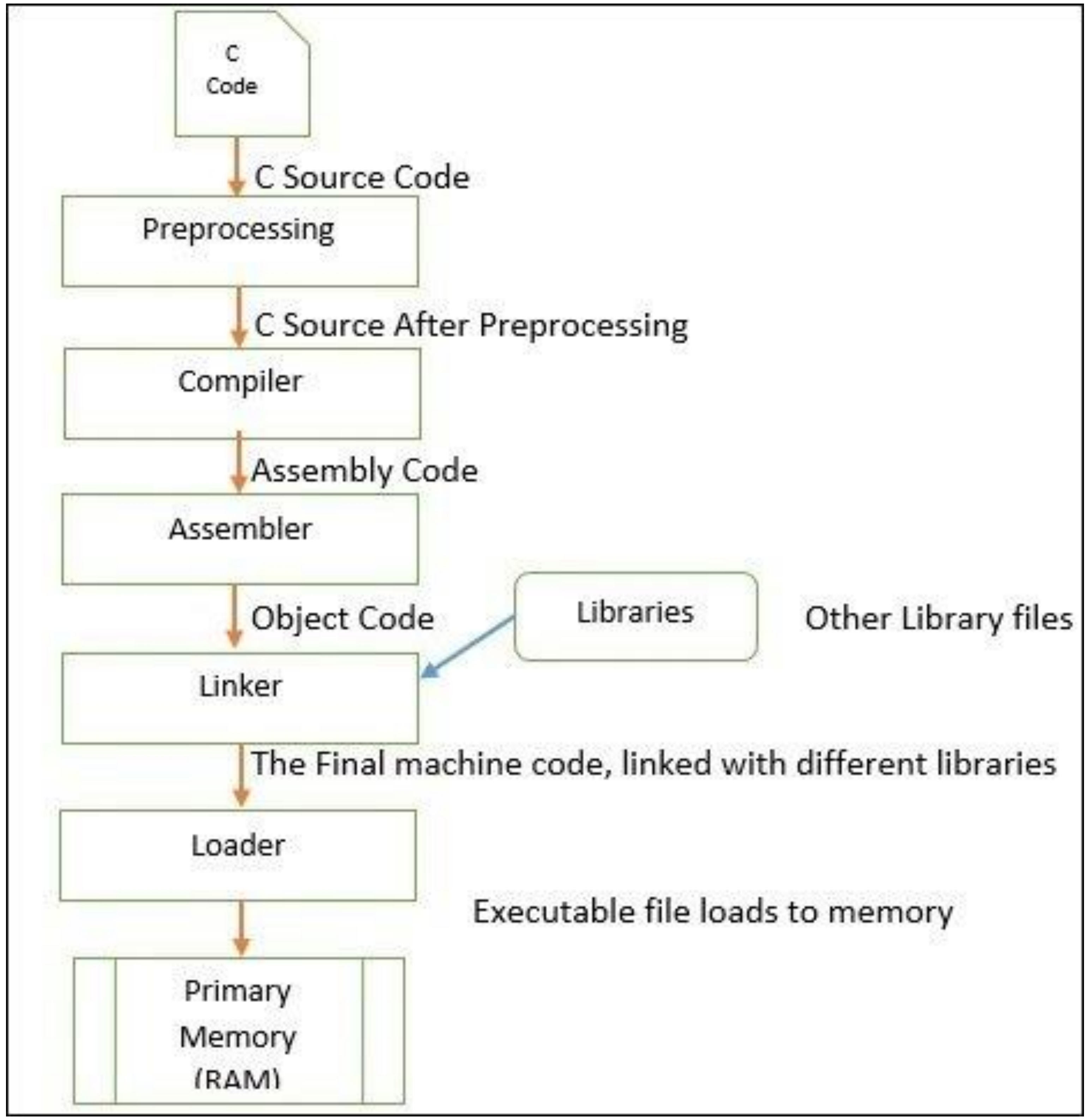
INTRODUCTION

What is a compiler

- A system software to convert source language program to target language program
- Compiler design started with FORTRAN in 1950s
- Validates input program to the source language specification – produces error message / warnings

Steps in Creating and Running Code



System Software: Program Development Environment

Compiler

Translates programming language (usually high-level, such as C/C++, Java, Pascal) to object code or machine code

Assembler

Translates assembly language programs to object programs or machine code

Linker

Combines and resolves references between object programs

Loader

Loads an executable program and starts its execution

(Low-level) Debugger

Used to debug executable programs and their associated object and source programs (trace variables, set breakpoints, etc.)

Steps in Creating and Running Code

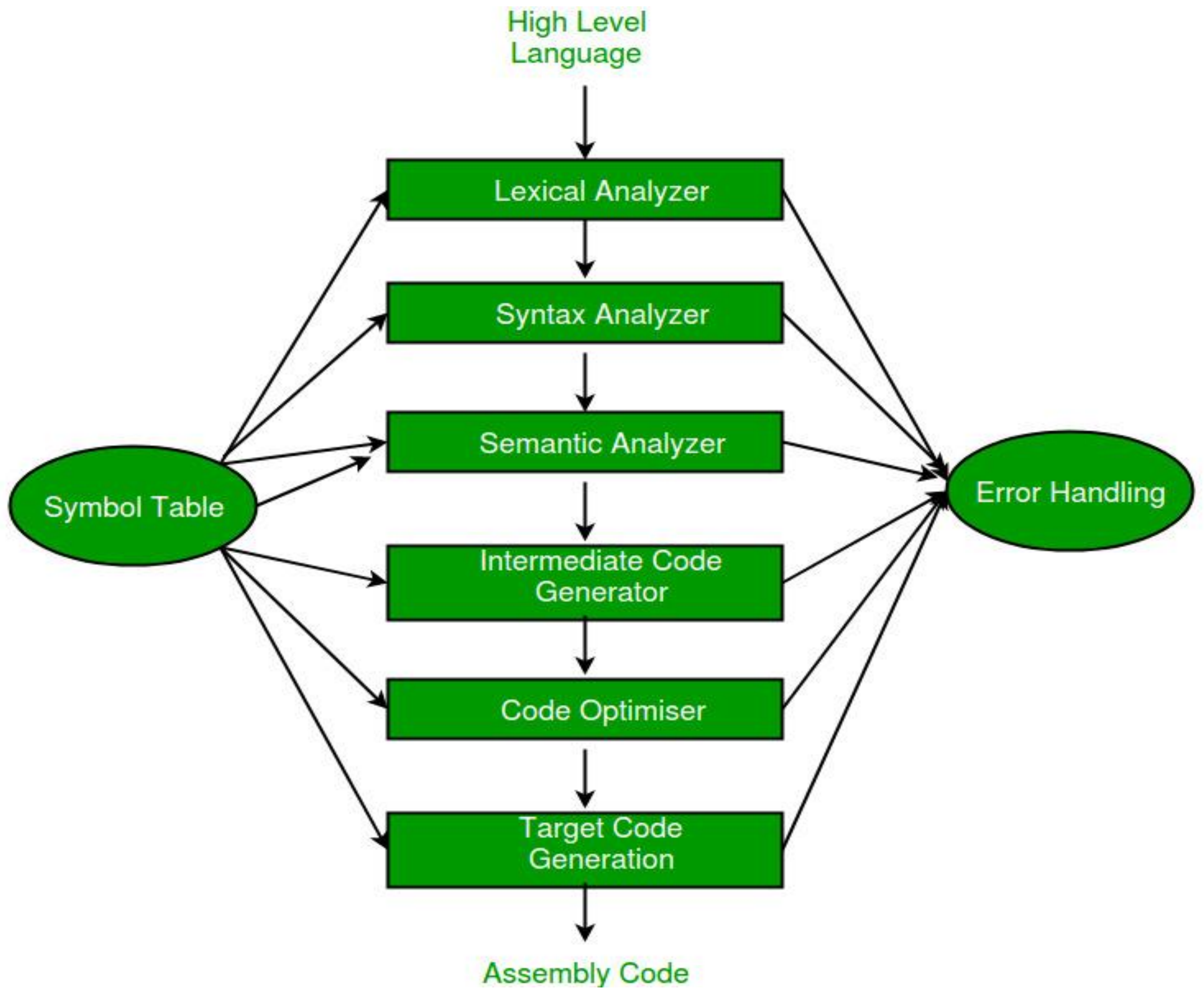
- **Source Program:** Human-readable program specification (e.g. C++, Assembly program)Usually created using a text editor (ASCII file)
- **Object Program:** Produced from a source program by compiling/assembling to “intermediate” machine code

“Intermediate” machine code augmented by:

- References (possibly undefined)
- Additional instructions related to combining the object program with other object programs, and/or executing the object program

- **Executable Program:** Instruction sequence that a computer can directly execute (“machine code”)
    - May be produced directly by a compiler/assembler
    - Often produced by combining object programs

Phases of a Compiler



Lexical Analyzer (Scanner)

- The lexical phase (scanner) groups characters into lexical units or tokens (Keyword, identifier, number,..etc.)
- The input to the lexical phase is a character stream. The output is a stream of tokens.
- Regular expressions are used to define the tokens recognized by a scanner (e.g. digit -> 0|1|..|9 and letter -> [A..Za-z], and identifier -> letter (letter|digit)*.
- The scanner can be implemented as a finite state machine.
- Lexeme: Position := initial + rate * 60 ;

Blanks, Line breaks, etc. are scanned out

Syntax Analyzer (Parser)

- The parser recognizing whether a program (or sentence) is grammatically well formed. It groups tokens into syntactical units.
- The output of the parser is a parse tree representation of the program.
- Context-free grammars are used to define the program structure recognized by a parser.

Semantic Analyzer (Semantic)

- The semantic analysis phase analyzes the parse tree for context-sensitive information often called the static semantics.
- Type Checking - Legality of Operands
  - Real := int + char ;
  - A[int] := A[real] + int ;
  - while char <> int do
- The output of the semantic analysis phase is an annotated parse tree (augmented with semantic actions

Symbol Table / Error Handling

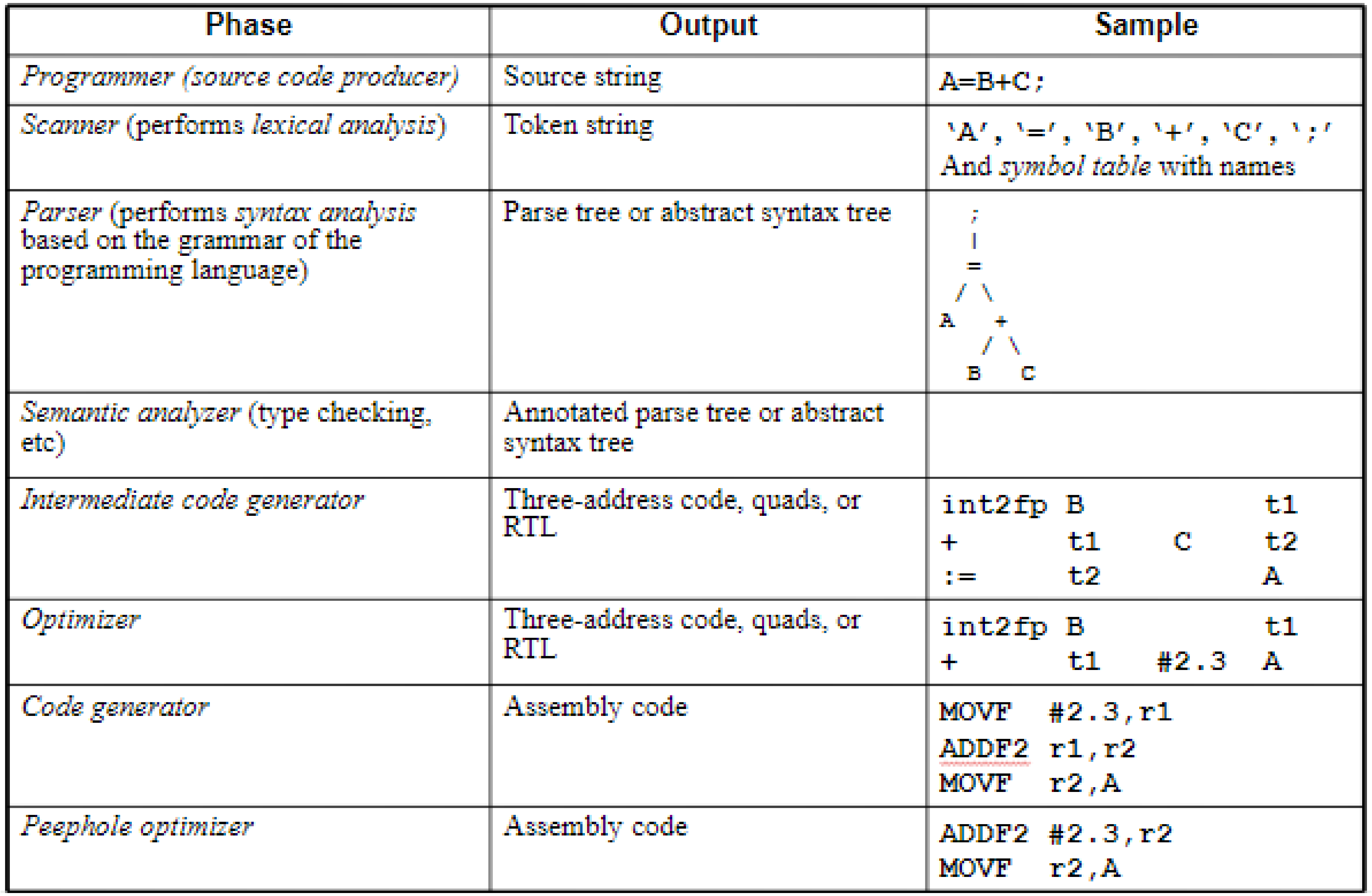
- Symbol Table Creation / Maintenance
  - Contains Info on Each “Meaningful” Token, Typically Identifiers
  - Data Structure Created / Initialized During Lexical Analysis
  - Utilized / Updated During Later Analysis & Synthesis
- Error Handling
  - Detection of Different Errors Which Correspond to All Phases
  - What Kinds of Errors Are Found During the Analysis Phase or Synthesis Phase?
  - What Happens When an Error Is Found?

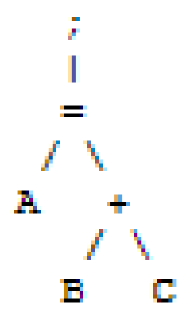
Intermediate Code Generation

- It uses Abstract Machine Version of Code - Independent of Architecture
- Easy to Produce and Do Final, Machine Dependent Code Generation
- Three-Address Code: “Portable” assembly-like language
  - Every memory location can act like a register
  - At most three operands per instruction

```
temp1 := inttoreal(60)
temp2 := id3 * temp1
temp3 := id2 + temp2
id1 := temp3
```

Example



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`