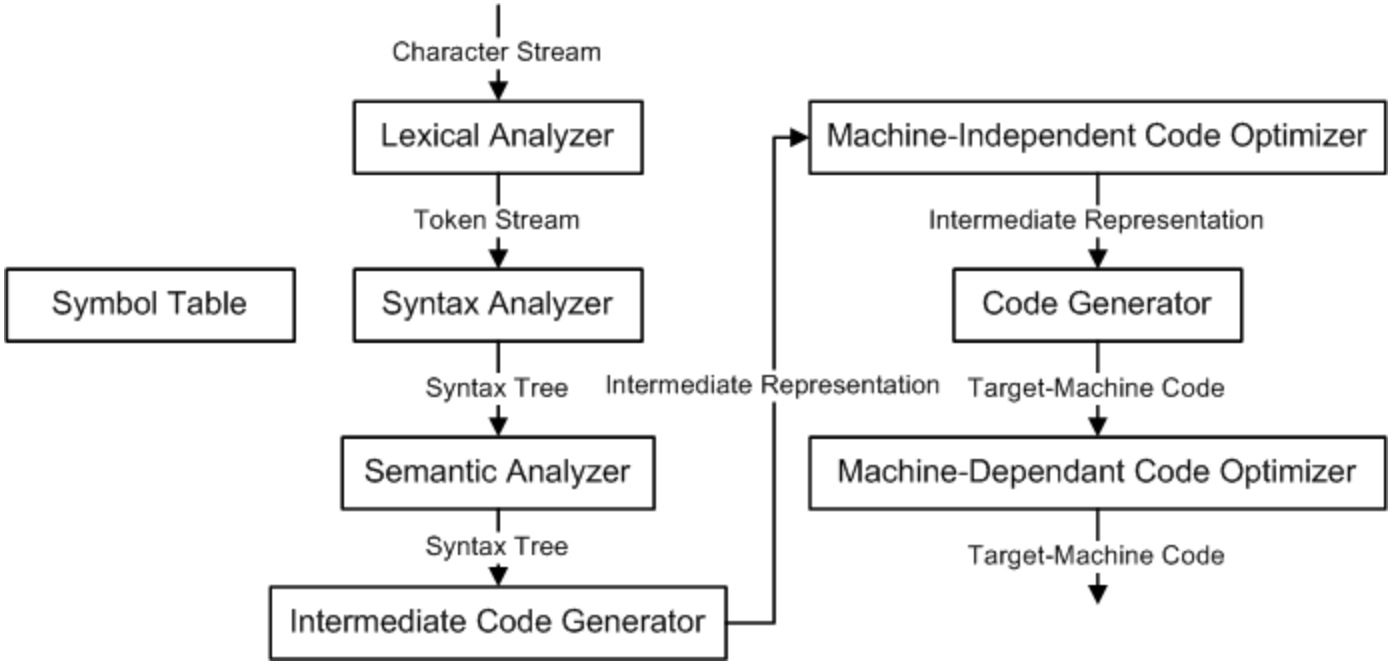
Compiler Construction

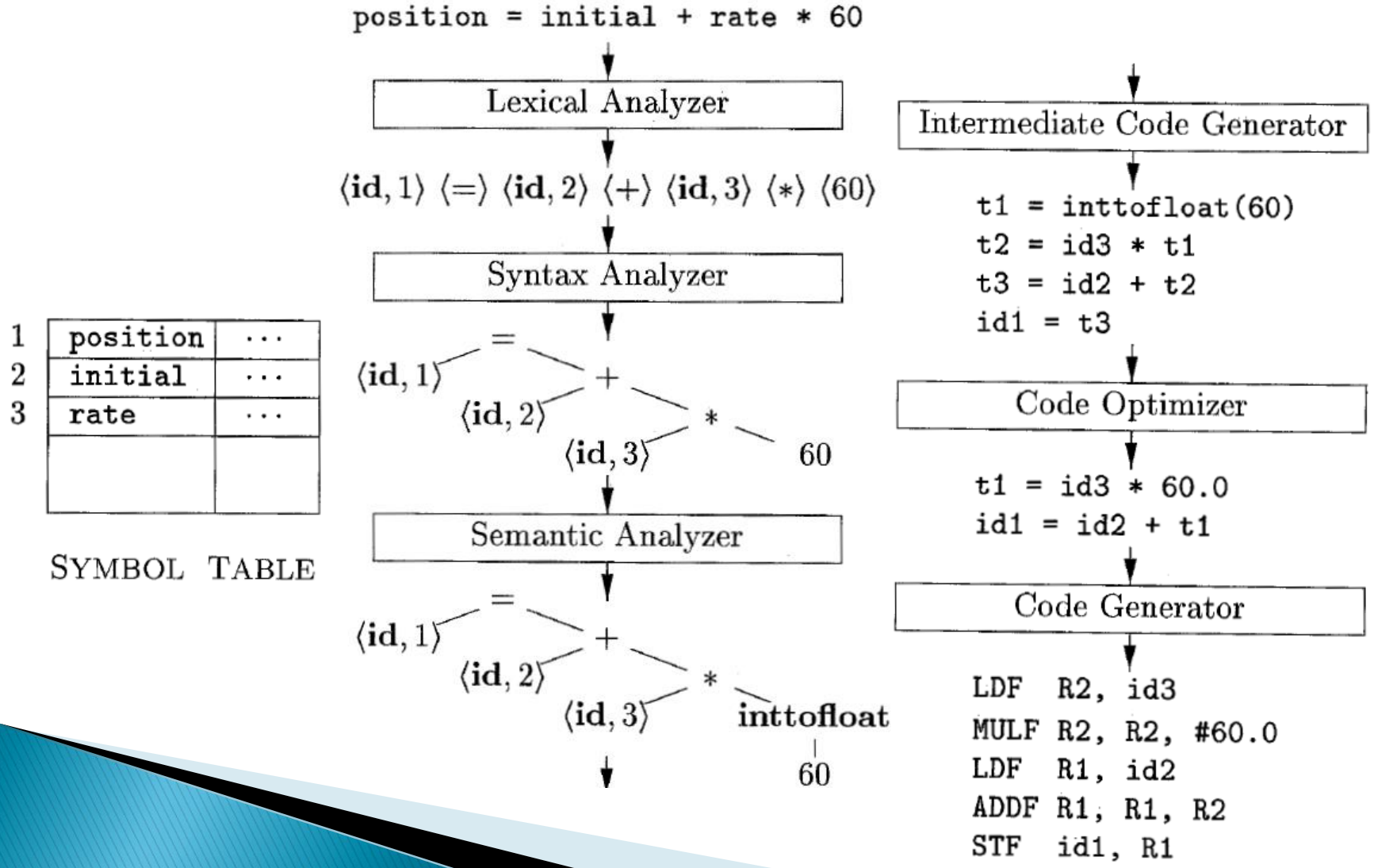
(Week 1, Lecture 2)

Phases of a Compiler



Phases of a Compiler

Translation of an Assignment Statement



Phases of a Compiler

- Lexical Analysis
  - Reads the stream of characters making up the source program and groups the characters into meaningful sequences called lexemes. For each lexeme, the lexical analyzer produces as output a token of the form <token-name, attribute-value>.

Phases of a Compiler

- Syntax Analysis (Parsing)
  - Uses the tokens produced by the lexical analyzer to create a tree-like intermediate representation that depicts the grammatical structure of the token stream.

Phases of a Compiler

- Semantic Analysis
  - Uses the syntax tree and the information in the symbol table to check the source program for semantic consistency with the language definition.
  - Also gathers type information and saves it in either the syntax tree or the symbol table, for subsequent use during intermediate-code generation.

Phases of a Compiler

- Intermediate Code Generation
  - May construct one or more intermediate representations.
  - Syntax trees are a form of intermediate representation, commonly used during syntax and semantic analysis.
  - Many compilers generate an explicit low-level or machine-like intermediate representation.

Phases of a Compiler

- Code Optimization
  - Attempts to improve the intermediate code so that better target code will result.
  - Usually better means faster, but other objectives may be desired, such as shorter code, or target code that consumes less power.

Phases of a Compiler

- Code Generation
  - Takes as input an intermediate representation of the source program and maps it into the target language.
  - If the target language is machine code, registers or memory locations are selected for each of the variables used by the program.
  - The intermediate instructions are translated into sequences of machine instructions that perform the same task.
  - Crucial aspect: Judicious assignment of registers.

Phases of a Compiler

- Symbol-Table Management
  - Record the variable names used in the source program and collect information about various attributes of each name.
  - These attributes may provide information about the storage allocated for a name, its type, its scope.
  - Procedure names: , number and types of arguments, the method of passing each argument and the type returned.

Phases of a Compiler

- Grouping of Phases into Passes
  - Logical organization of a compiler.
  - Activities from several phases may be grouped together into a pass that reads an input file and writes an output file.
    - Example:
      - Front-end phases (lexical analysis, syntax analysis, semantic analysis and intermediate code generation) – One pass.
      - Code optimization – An optional pass.
      - Back-end pass consisting of code generation for a particular target machine.

Compiler Construction Tools

- Scanner generators to produce lexical analyzers from a regular-expression description of the tokens of a language, e.g. Lex, Flex.
- Parser generators to automatically produce syntax analyzers from a grammatical description of a programming language, e.g. Yacc and Bison.

Compiler Construction Tools

- Syntax-directed translation engines that produce collections of routines for walking a parse tree and generating intermediate code. Syntax-directed translation engines that produce collections of routines for walking a parse tree and generating intermediate code.

Compiler Construction Tools

- Code-generator generators that produce a code generator from a collection of rules for translating each operation of the intermediate language into the machine language for a target machine.

Compiler Construction Tools

- Data-flow analysis engines that facilitate the gathering of information about how values are transmitted from one part of a program to each other part. Data-flow analysis is a key part of code optimization.
- Compiler-construction toolkits that provide an integrated set of routines for constructing various phases of a compiler.