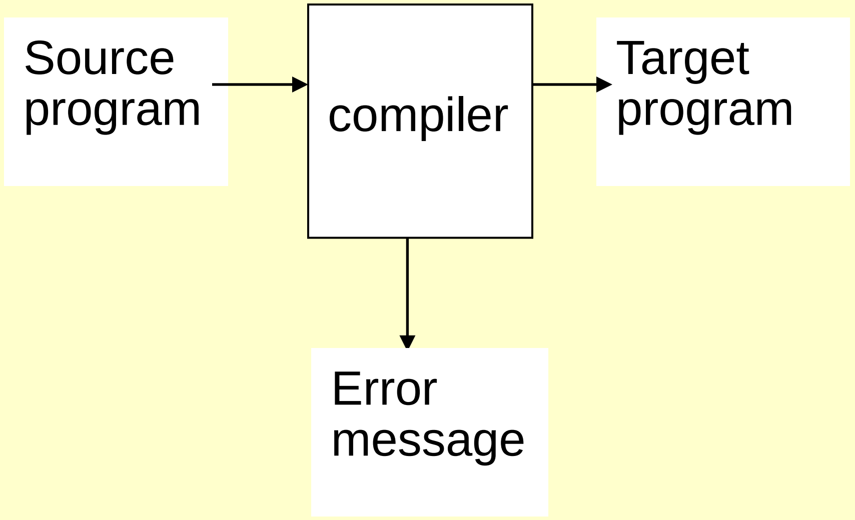
Introduction

Jaruloj Chongstitvatana
Department of Mathematics and Computer Science

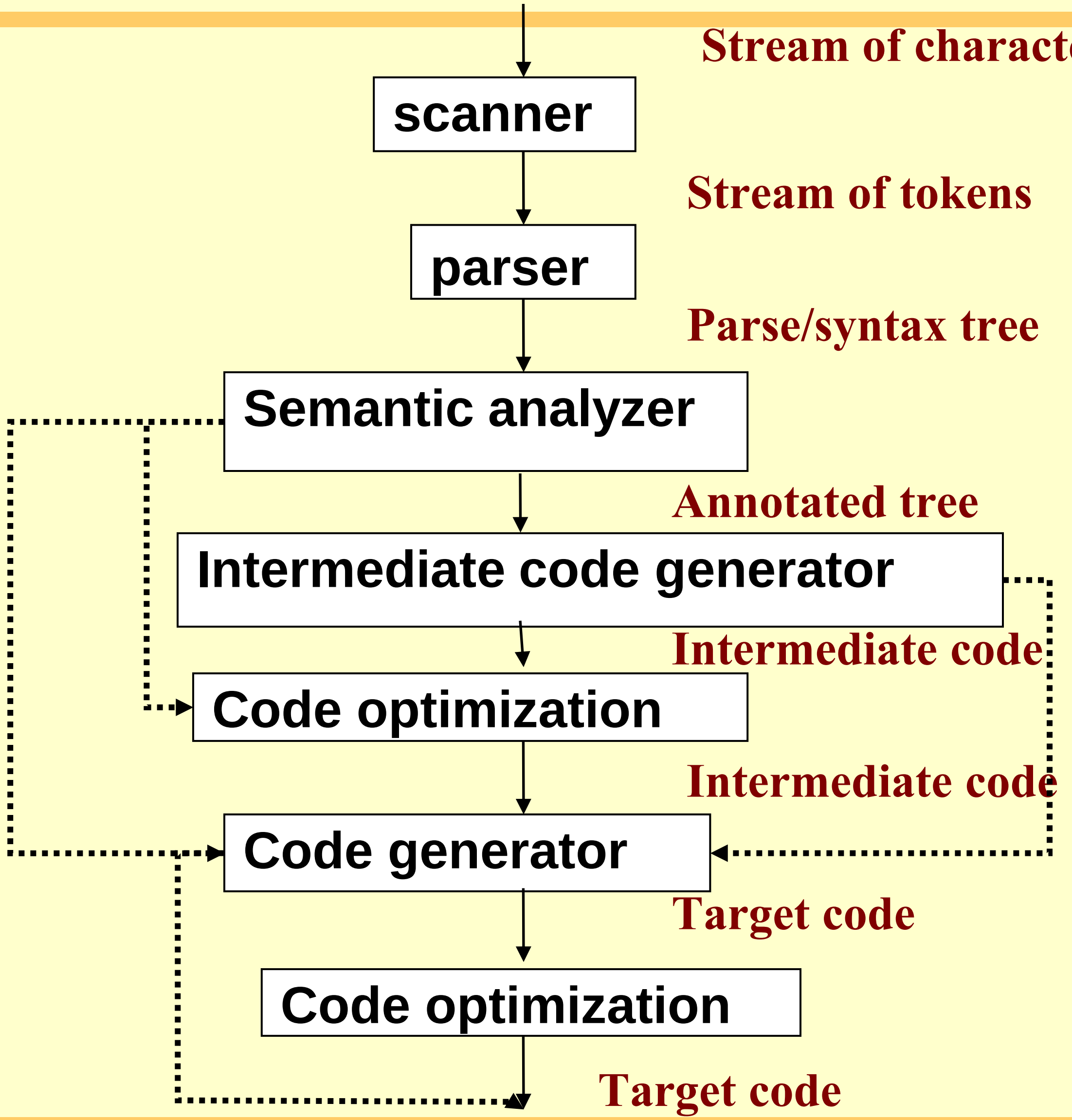
Chulalongkorn University

What is a Compiler?

- A **compiler** is a computer program that translates a program in a *source language* into an equivalent program in a *target language*.
- A **source program/code** is a program/code written in the source language, which is usually a high-level language.
- A **target program/code** is a program/code written in the target language, which often is a machine language or an intermediate code.



Process of Compiling



Some Data Structures

- Symbol table
- Literal table
- Parse tree

Symbol Table

- Identifiers are **names** of variables, constants, functions, data types, etc.
- Store information associated with identifiers
  - Information associated with different types of identifiers can be different
    - Information associated with variables are name, type, address,size (for array), etc.
    - Information associated with functions are name,type of return value, parameters, address, etc.

Symbol Table (cont'd)

- Accessed in every phase of compilers
  - The scanner, parser, and semantic analyzer put names of identifiers in symbol table.
  - The semantic analyzer stores more information (e.g. data types) in the table.
  - The intermediate code generator, code optimizer and code generator use information in symbol table to generate appropriate code.
- Mostly use hash table for efficiency.

Literal table

- Store constants and strings used in program
  - reduce the memory size by reusing constants and strings
- Can be combined with symbol table

Parse tree

- Dynamically-allocated, pointer-based structure
- Information for different data types related to parse trees need to be stored somewhere.
  - Nodes are variant records, storing information for different types of data
  - Nodes store pointers to information stored in other data structure, e.g. symbol table

Scanning

- A scanner reads a stream of characters and puts them together into some meaningful (with respect to the source language) units called ***tokens***.
- It produces a stream of tokens for the next phase of compiler.

Parsing

- A parser gets a stream of tokens from the scanner, and determines if the syntax (structure) of the program is correct according to the (context-free) grammar of the source language.
- Then, it produces a data structure, called a *parse tree or an abstract syntax tree*, which describes the syntactic structure of the program.

Semantic analysis

- It gets the parse tree from the parser together with information about some syntactic elements
- It determines if the semantics or meaning of the program is correct.
- This part deals with *static semantic*.
  - semantic of programs that can be checked by reading off from the program only.
  - *syntax of the language which cannot be described in context-free grammar*.
- Mostly, a semantic analyzer does type checking.
- It modifies the parse tree in order to get that (static) semantically correct code.

Intermediate code generation

- An intermediate code generator
  - takes a parse tree from the semantic analyzer
  - generates a program in the intermediate language.
- In some compilers, a source program is translated into an intermediate code first and then the intermediate code is translated into the target language.
- In other compilers, a source program is translated directly into the target language.

Intermediate code generation (cont'd)

- Using intermediate code is beneficial when compilers which translates a single source language to many target languages are required.
  - The front-end of a compiler – *scanner to intermediate code generator* – can be used for every compilers.
  - Different back-ends – *code optimizer and code generator*– is required for each target language.
- One of the popular intermediate code is *three-address code*. A three-address code instruction is in the form of $x = y \ op \ z$.

Code optimization

- Replacing an inefficient sequence of instructions with a better sequence of instructions.
- Sometimes called code improvement.
- Code optimization can be done:
  - after semantic analyzing
    - performed on a parse tree
  - after intermediate code generation
    - performed on a intermediate code
  - after code generation
    - performed on a target code

Code generation

- A code generator
  - takes either an intermediate code or a parse tree
  - produces a target program.

Error Handling

- Error can be found in every phase of compilation.
  - Errors found during compilation are called *static* (or *compile-time*) *errors.*
  - Errors found during execution are called *dynamic* (or *run-time*) *errors*
- Compilers need to detect, report, and recover from error found in source programs
- Error handlers are different in different phases of compiler.

Reading Assignment

- Louden, K.C., Compiler Construction: Principles and Practice, PWS Publishing, 1997. ->Chapter 1