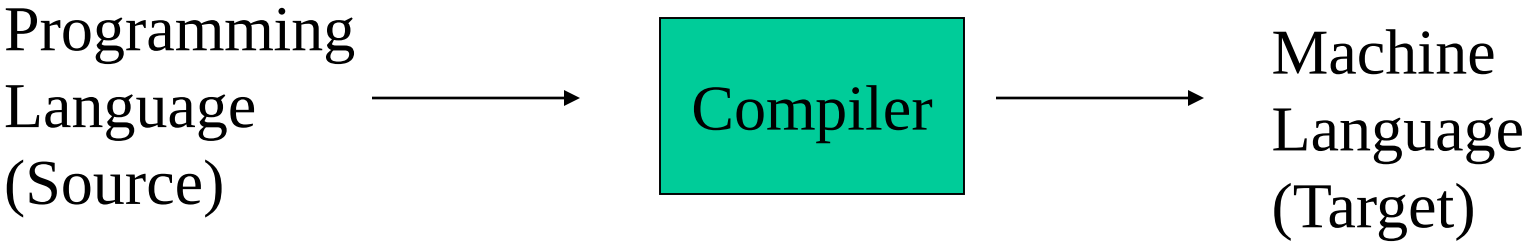
Chapter 1 Introduction

Outlines

- 1.1 Overview and History
- 1.2 What Do Compilers Do?
- 1.3 The Structure of a Compiler
- 1.4 The Syntax and Semantics of Programming Languages
- 1.5 Compiler Design and Programming Language Design
- 1.6 Compiler Classifications
- 1.7 Influences on Computer Design

Overview and History

- Compilers are fundamental to modern computing.
- They act as translators, transforming human-oriented programming languages into computer-oriented machine languages.



Overview and History (Cont'd)

- The first real compiler
  - FORTRAN compilers of the late 1950s
  - 18 person-years to build
- Compiler technology is more broadly applicable and has been employed in rather unexpected areas.
  - Text-formatting languages, like nroff and troff; preprocessor packages like eqn, tbl, pic
  - Silicon compiler for the creation of VLSI circuits
  - Command languages of OS
  - Query languages of Database systems

What Do Compilers Do?

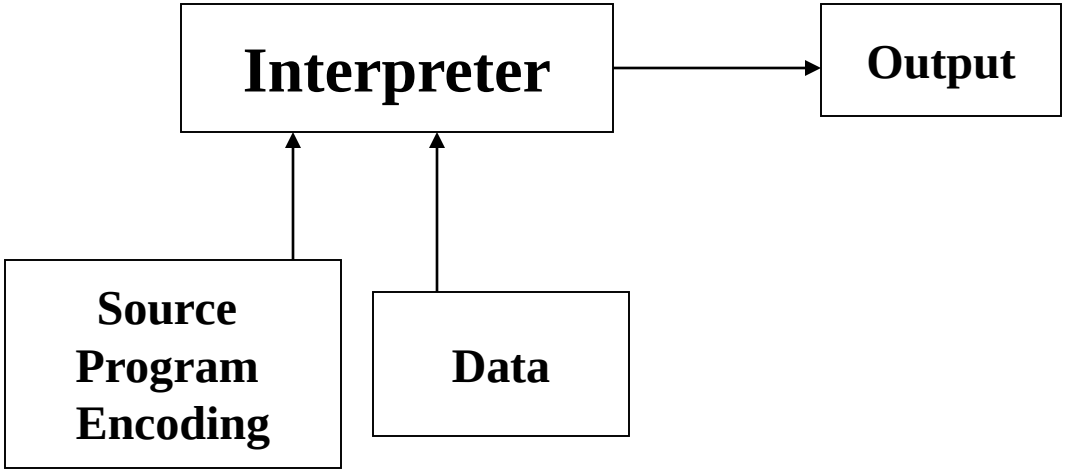
- Compilers may be distinguished according to the kind of target code they generate:
  - Pure Machine Code
  - Augmented Machine Code
  - Virtual Machine Code
    - JVM, P-code

What Do Compilers Do? (Cont'd)

- Another way that compilers differ from one another is in the format of the target machine code they generate
  - Assembly Language Format
  - Relocatable Binary Format
    - A linkage step is required
  - Memory-Image (Load-and-Go) Format

What Do Compilers Do? (Cont'd)

- Another kind of language processor, called an ***interpreter***, differs from a compiler in that it executes programs without explicitly performing a translation

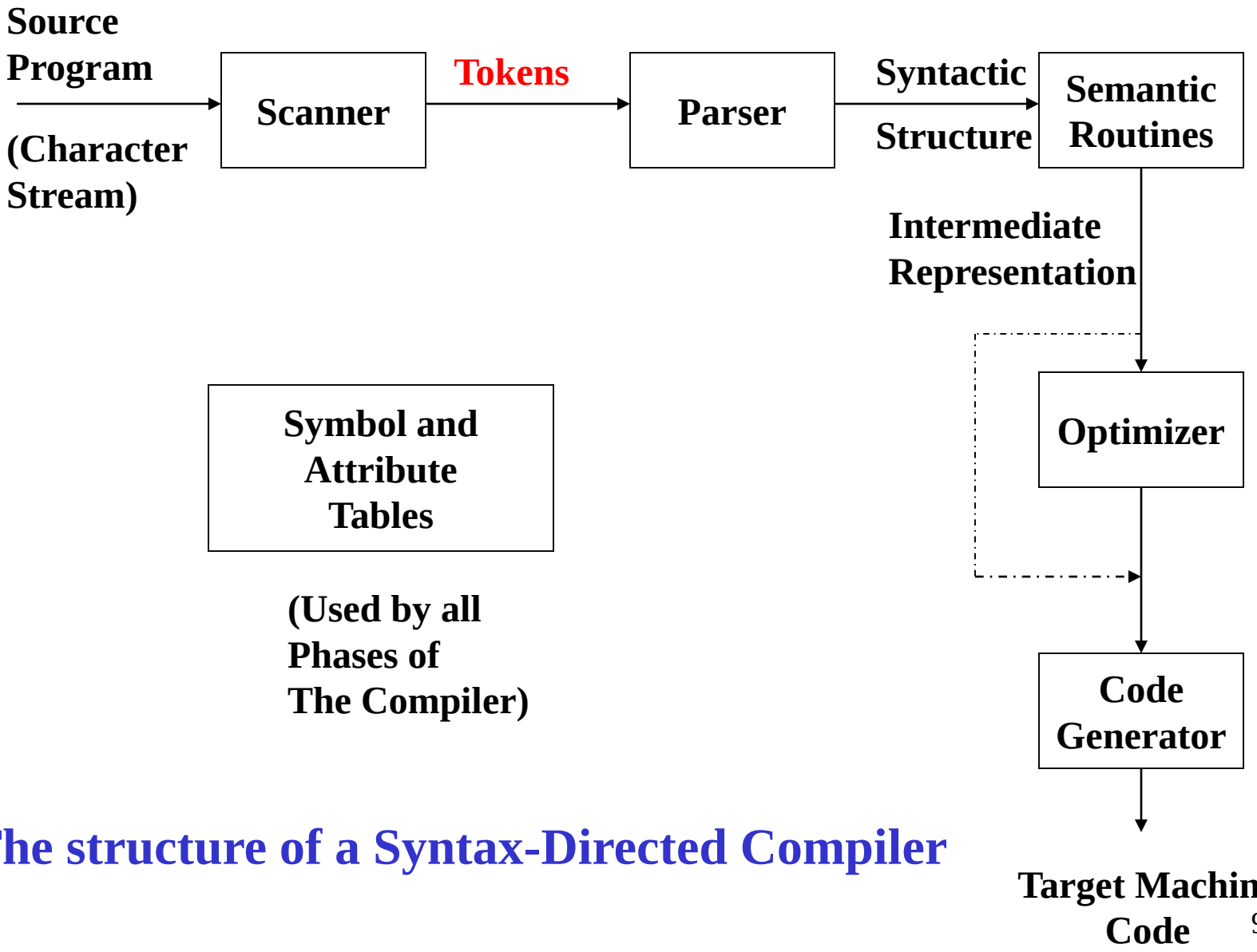


- Advantages and Disadvantages of an interpreter
  - See page 6 & 7

The Structure of a Compiler

- Any compiler must perform two major tasks
  - ***Analysis*** of the source program
  - ***Synthesis*** of a machine-language program

The Structure of a Compiler (Cont'd)



The structure of a Syntax-Directed Compiler

The Structure of a Compiler (Cont'd)

- Scanner
  - The scanner begins the analysis of the source program by reading the input, character by character, and grouping characters into individual words and symbols (**tokens**)
  - The **tokens** are encoded and then are fed to the parser for syntactic analysis
  - For details, see the bottom of page 8.
- Scanner generators

The Structure of a Compiler (Cont'd)

- Parser
  - Given a formal syntax specification (typically as a context-free [CFG] grammar), the parse reads tokens and groups them into units as specified by the productions of the CFG being used.
  - While parsing, the parser verifies correct syntax, and if a syntax error is found, it issues a suitable diagnostic.
  - As syntactic structure is recognized, the parser either calls corresponding semantic routines directly or builds a syntax tree.

The Structure of a Compiler (Cont'd)

- Semantic Routines
  - Perform two functions
    - Check the static semantics of each construct
    - Do the actual translation
  - The heart of a compiler
- Optimizer
  - The IR code generated by the semantic routines is analyzed and transformed into functionally equivalent but improved IR code.
  - This phase can be very complex and slow
  - Peephole optimization

The Structure of a Compiler (Cont'd)

- One-pass compiler
  - No optimization is required
  - To merge code generation with semantic routines and eliminate the use of an IR
- Compiler writing tools
  - Compiler generators or compiler-compilers
    - E.g. scanner and parser generators

Compiler Design and Programming Language Design

- An interesting aspect is how programming language design and compiler design influence one another.
- Programming languages that are easy to compiler have many advantages
  - See the 2nd paragraph of page 16.

Compiler Design and Programming Language Design (Cont'd)

- Languages such as Snobol and APL are usually considered noncompilable
- What attributes must be found in a programming language to allow compilation?
  - Can the scope and binding of each identifier reference be determined before execution begins
  - Can the type of object be determined before execution begins?
  - Can existing program text be changed or added to during execution?

Compiler Classifications

- Diagnostic compilers
- Optimizing compilers
- Retargetable compiler