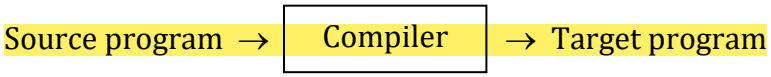
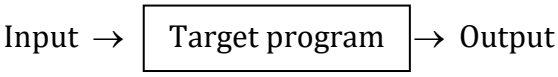
Introduction to Compilers

- Before a program can be run, it first must be translated into a form in which it can be executed by a computer.
- The software systems that do this translation are called ***compilers***.
- We will try to learn the different forms of language translators and a high-level overview of the structure of a typical compiler.

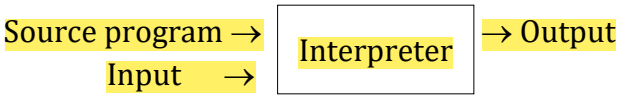
- **Compiler:** A compiler is a program that can read a program in one language – the *source language* – and translate it into an equivalent program in another language – the *target language*.
- The name *Compiler* is primarily used for programs that translate source code from a high-level programming language to a lower level language to create an executable program.
- An important role of the compiler is to report any errors in the source program that it detects during the translation process.



- If the target program is an executable machine-language program, it can then be called by the user to process inputs and produce output.

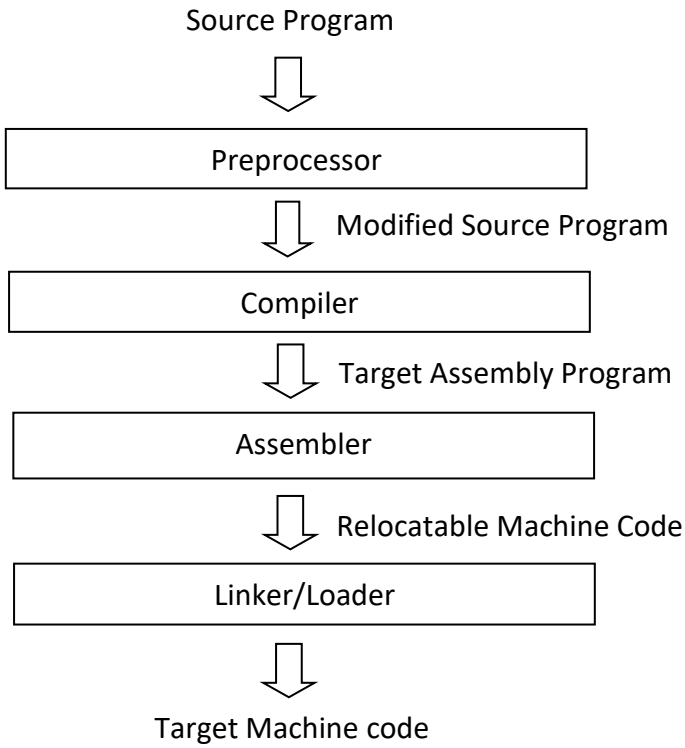


- **Interpreter**: It is another common kind of language processor. Instead of producing a target program as a translation, an interpreter appears to directly execute the operations specified in the source program on inputs supplied by the user.



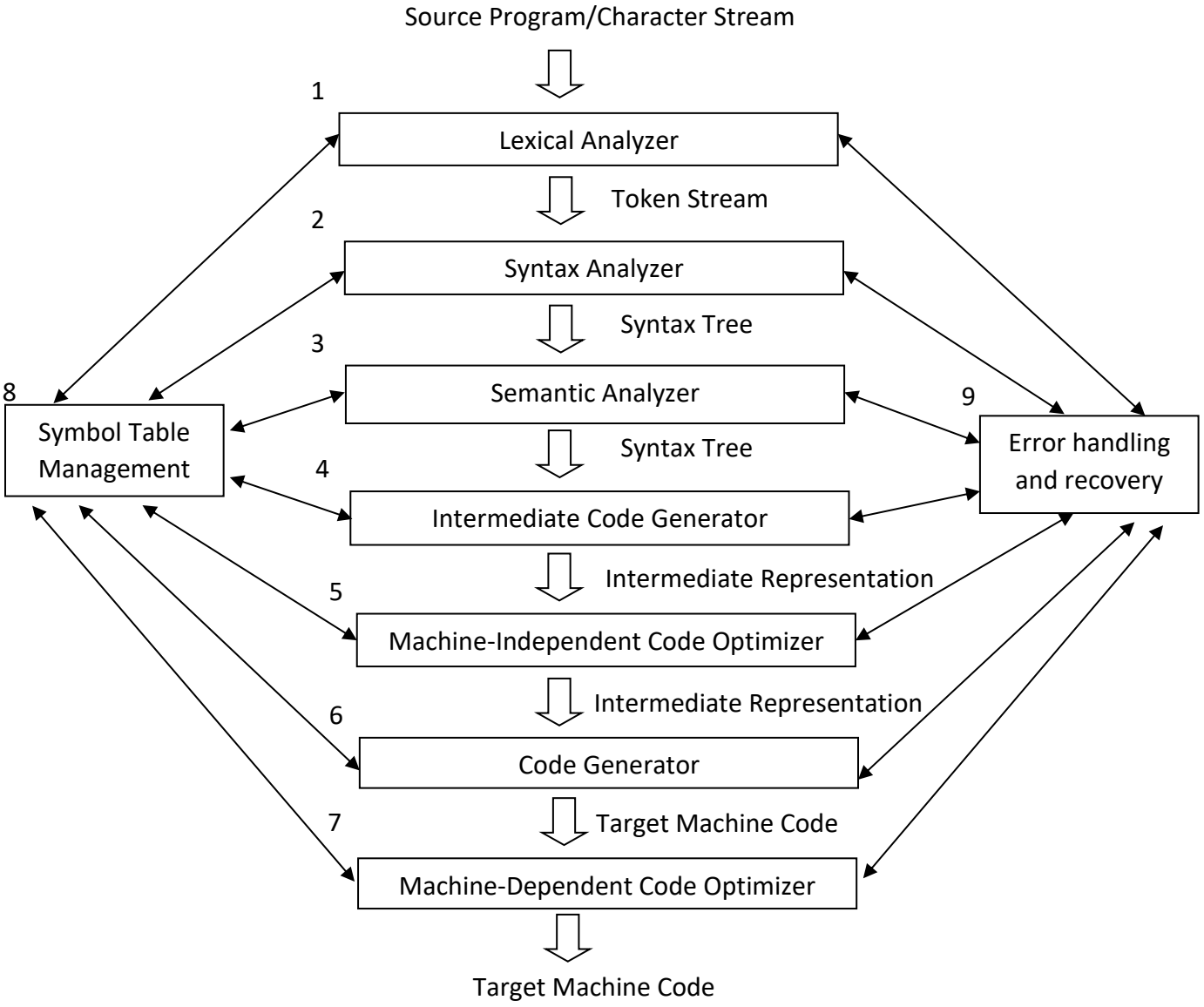
- The machine-language target program produced by a compiler is usually much faster than an interpreter at mapping inputs to outputs.
- An interpreter, however, can usually give better error diagnostics than a compiler, because it executes the source program statement by statement.

- In addition to a compiler, several other programs may be required to create an executable target program.
- A source program may be divided into modules stored in separate files. The task of collecting the source program is sometimes entrusted to a separate program, called a **preprocessor**.
- The modified source program is then fed to a compiler. The compiler may produce an assembly-language program as its output, because assembly language is easier to produce as output and is easier to debug.
- The assembly language is then processed by a program called an **assembler** that produces *relocatable machine code* as its output.
- Large programs are often compiled in pieces, so the relocatable machine code may have to be linked together with other relocatable object files and library files into the code that actually runs on the machine. The **linker** resolves external memory addresses, where the code in one file may refer to a location in another file. The **loader** then puts together all of the executable object files into memory for execution.



A Language Processing System

Structure of a Compiler Enigma - 1c



Major Phases of a Compiler with phase inputs and outputs

-The structure of a Compiler can be mapped into two parts: Analysis (Front End) and Synthesis (Back End).

Analysis Phase:

- Analysis phase reads the source program and splits it into multiple tokens and constructs the intermediate representation of the source program.

- It also checks and indicates the syntax and semantic errors of a source program.

- It collects information about the source program and prepares the symbol table. Symbol table will be used all over the compilation process.

- This is also called as the front end of a compiler.

Synthesis Phase:

-It will get the analysis phase input (intermediate representation and symbol table) and produces the targeted machine level code.

-This is also called as the back end of a compiler.

Questions:

1. What is the difference between a compiler and an interpreter?
2. What are the advantages of (a) a compiler over an interpreter (b) an interpreter over a compiler?