Introduction to Language Processor: Language Processing activities and fundamentals of Language Processing.

Aditya Rajmane
Assistant Professor
Department of Computer Engineering
Sardar Patel Institute of Technology, Mumbai

Language Processor

----a compiler is a program that can read a program in one language -- the source code -- and translate it into an equivalent program in another language.

An important role of the compiler is to report any errors in the source program that it detects during the translation process.

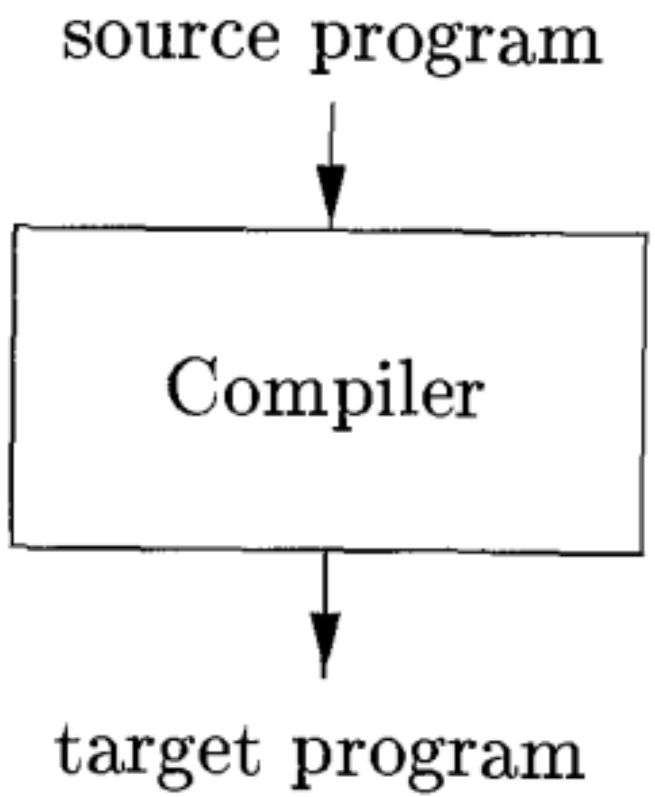

Figure 1.1: A compiler

In the target programme is an executable machine language program it can then be called by the user to process inputs and produce outputs.

Figure 1.2: Running the target program

An Interpreter

In your own words, what's an interpreter?

Have you read about an interpreter before?

If yes where and in what context?

An interpreter is another common kind of language processor. Instead of producing a target program as translation and interpreter appears to directly execute the operations specified in the source program on inputs supplied by the user.

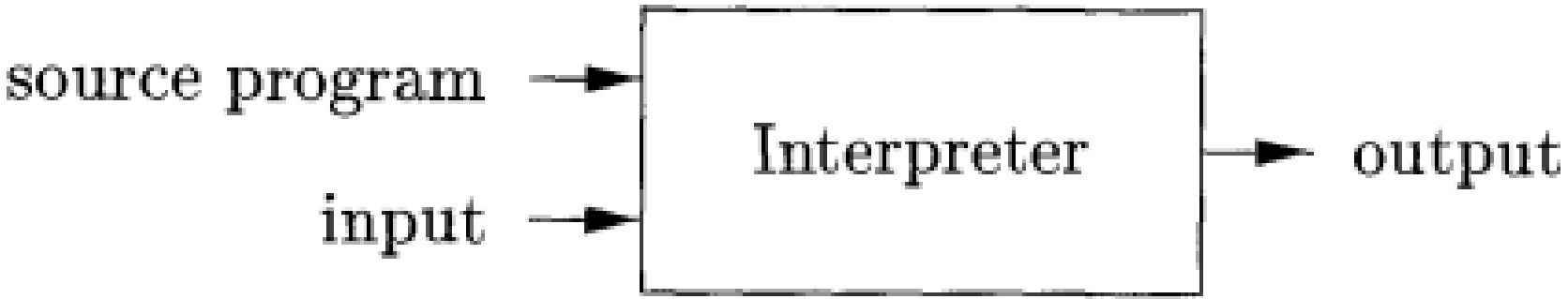

Figure 1.3: An interpreter

The machine language target program produced by 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

Java language processors combined compilation and interpretation. A Java source program will first be compiled into an intermediate form called *bytecodes*. The bytecodes are then interpreted by virtual machine. The benefit of this arrangement is that byte code is compiled on one machine can be interpreted on another machine. This makes Java platform independent.

To achieve faster processing of inputs to outputs some Java compilers Just-in-time compilers translate the bytecodes into machine language immediately before they run intermediate program to process the input.

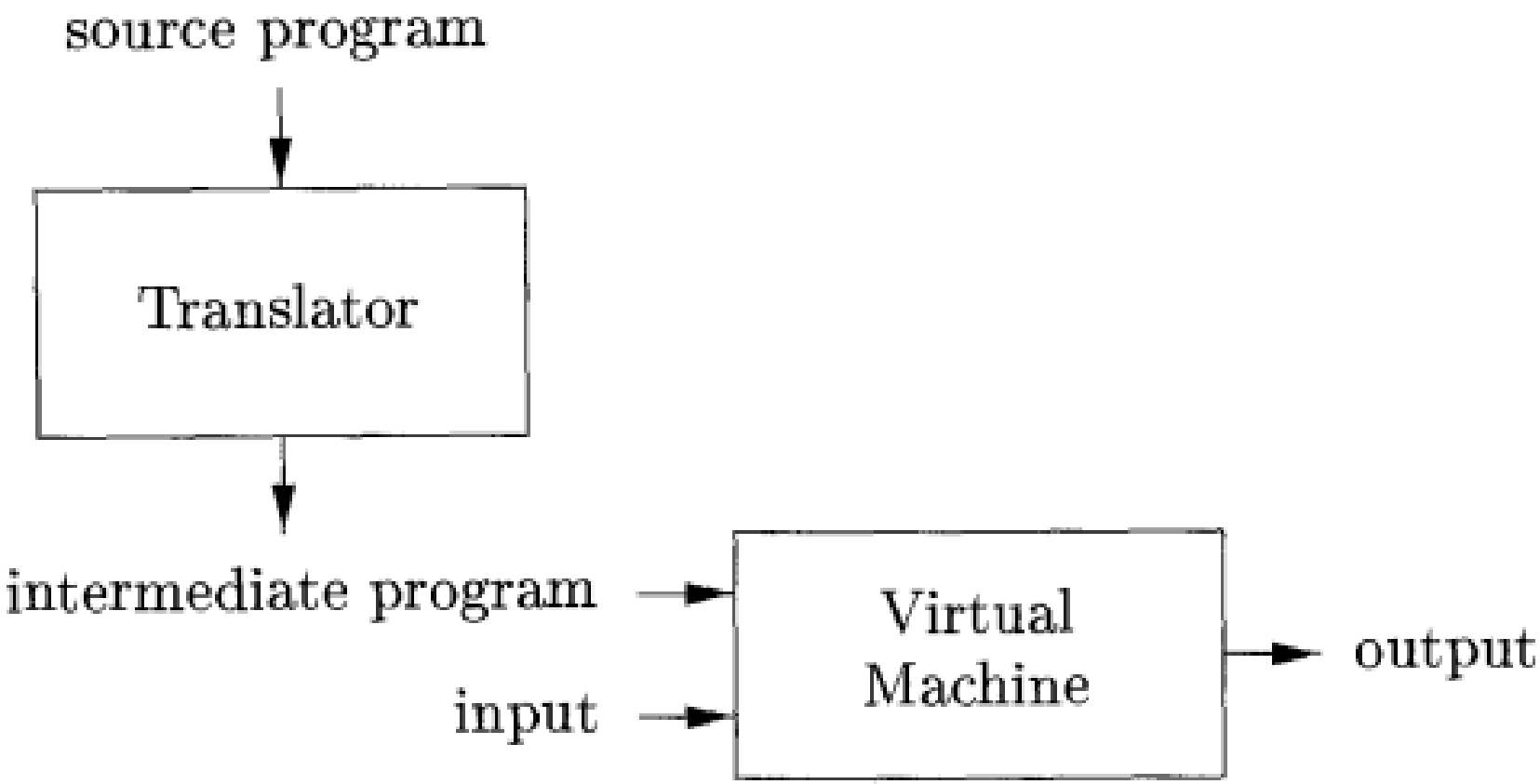

Figure 1.4: A hybrid compiler

A source program may be divided into modules stored in separate files. The task of collecting the source program is sometimes interested to a separate program, called a **preprocessor**. The preprocessor may also expand shorthands, call macros, into source language statement.

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 it's easier to debug the assembly language is then processed by program called **assembler** that produces relocatable machine code as its output.

Last 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 runs on the machine. The **linker** resolves external memory addresses, with the code in one file may refer to location another file. the **loader** then puts together the executable object files into memory for execution.

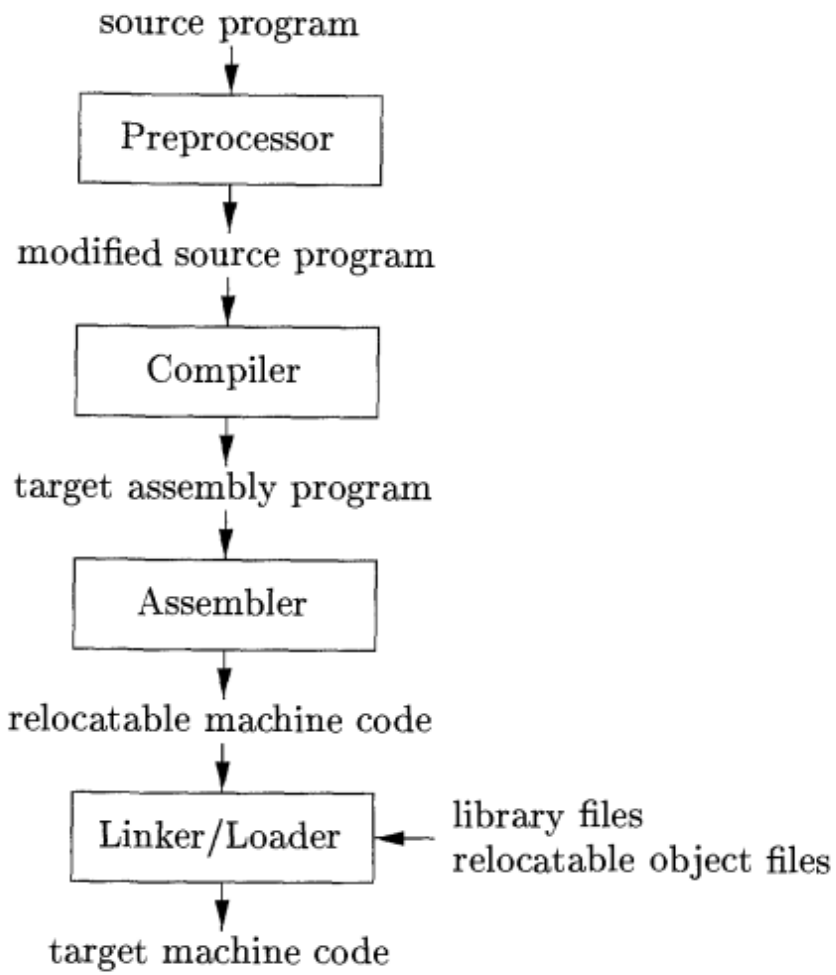

Figure 1.5: A language-processing system

The Compiler

We have defined compiler as a program that translates what language into an equivalent program in another language. Let us look at the structure of a compiler is much more detail. There are essentially 2 parts to this mapping: analysis and synthesis.

The **analysis** part breaks of the source program is to constituent pieces and imposes a grammatical structure on them. It then uses the structure to create an intermediate representation of the source program.

The analysis part also detects anomaly in the source program, and it must provide informative messages so the user can take corrective action. The analysis part also collects information about the source program and stores it in data structure called a **symbol table**, which is passed along with the intermediate representation to the synthesis part. Which is used by all the steps the compilation process.

The **synthesis** part constructs the desired target program from the intermediate representation and the information in the symbol table. The analysis is often called **front end** of the compiler; The synthesis part is the **back end**.

Big compilation process is often looked at something that operates as a sequence of **phases** each of which transforms one representation of source program to another.

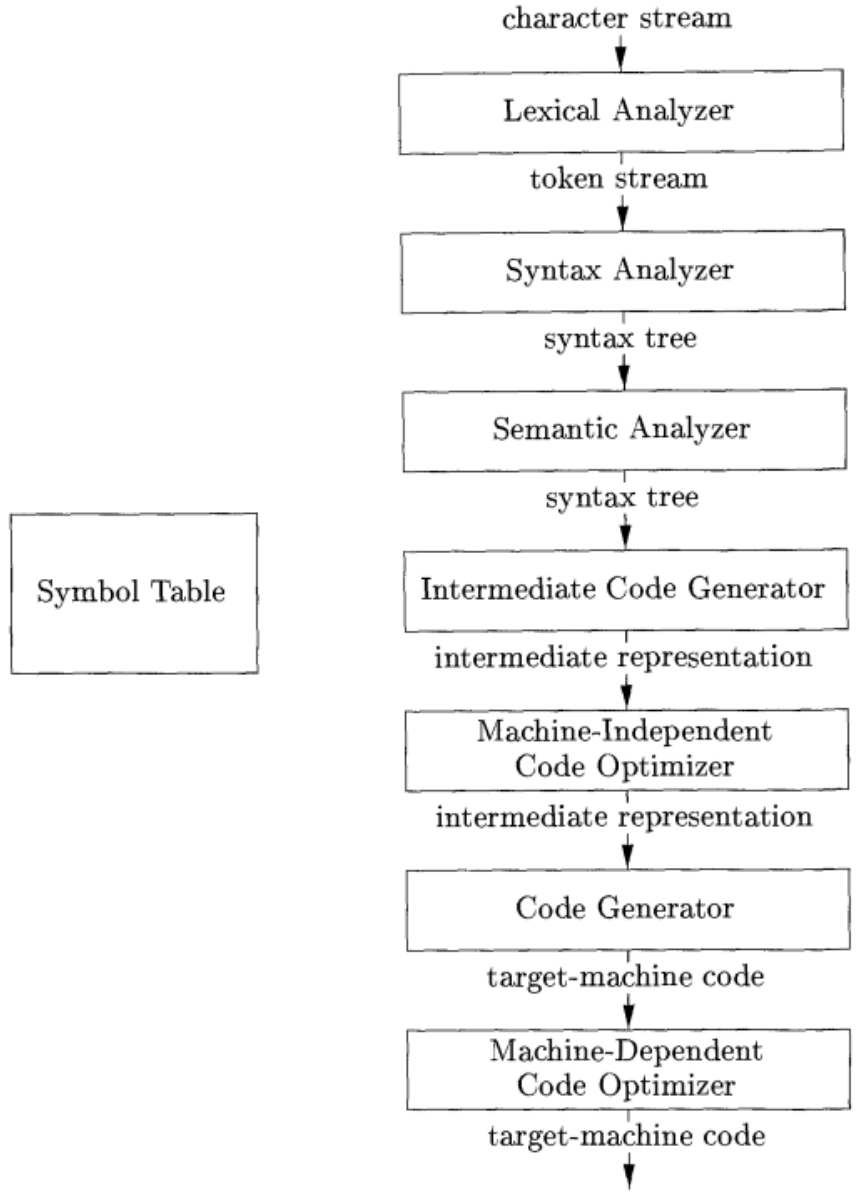

Figure 1.6: Phases of a compiler