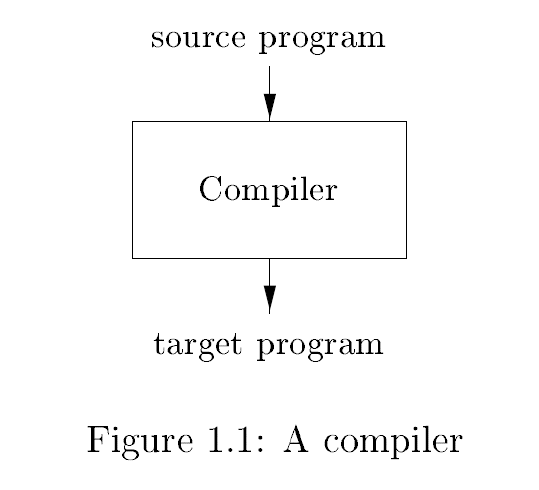
**1.1 Language Processors**

**Exercise 1.1.1. *What is the difference between a compiler and an interpreter?***

* **Compilers**: *Takes the whole source code as input and converts it into a target code. Input goes into the target code, which then gives the output.*
* *Reports any errors found in the source program during translation.*

**

* **Interpreter:** *Takes as input the source code as well as the input parameters and then gives the output directly.*

A diagram of a person's figure

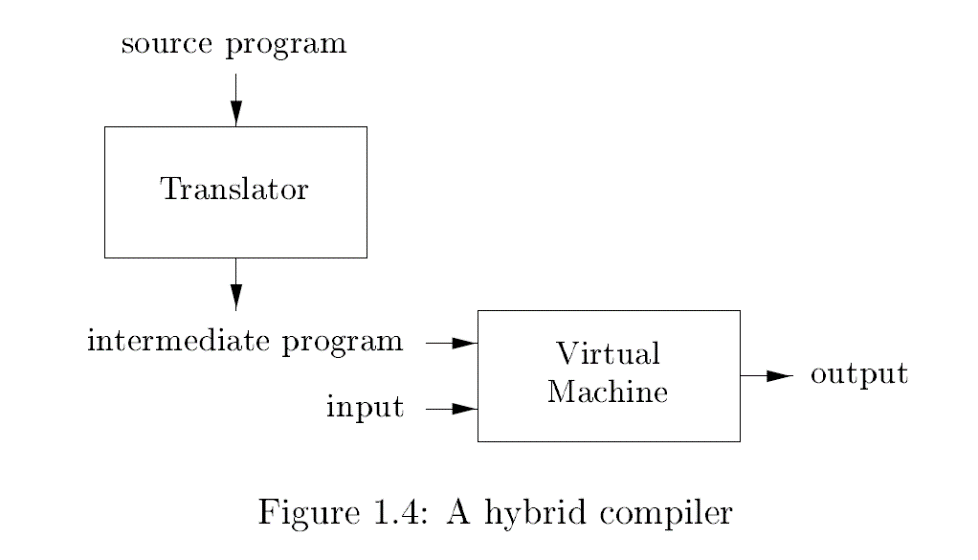
Description automatically generated

**Exercise1.1.2. *What are the advantages of (a) a compiler over an interpreter, (b) an interpreter over a compiler?***

1. *Compiler is way faster than an interpreter at mapping inputs to outputs.*
2. *Interpreter gives better error diagnostics than a compiler because it executes a source program statement by statement*.

**Exercise1.1.3. *What advantages are there to a language processing system in which the compiler produces assembly language rather than machine language?***

* *The produced code is machine-independent and hence, portable.*
* *A Java processor combines compiler and interpreter. The result is machine-independent interpreted code.*

**

* **Preprocessor:** Collects the source program. Expands macros into source language statements.

**Exercise1.1.4. *A compiler that translates a high-level Language to another high-level language is called a source-to-source translator. What advantages are there to using C as a target-language for a compiler?***

* *The advantages of using C as a target-language include –*
* *C is closer to the machine, yet a high-level language.*
* *C is an extremely fast language.*
* *C is widely supported and hence, portable.*

**Exercise1.1.5. *Describe some of the tasks that an assembler needs to perform.***

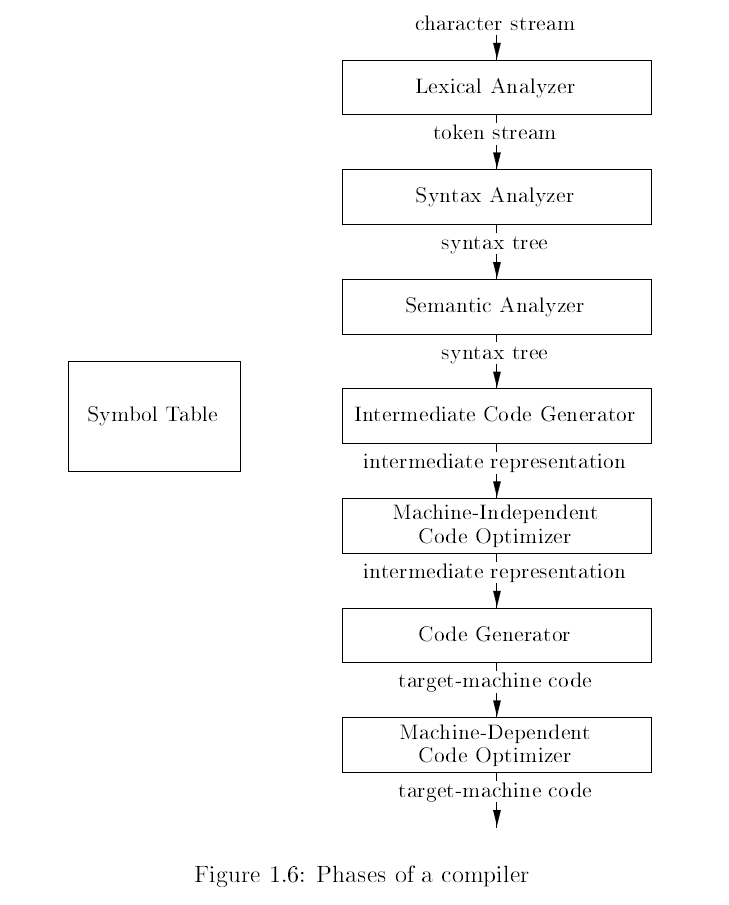
* **Assembler:** A compiler may produce an assembly language as output. An assembler produces relocatable machine code from the produced assembly language.
* Large programs are often compiled in pieces, so the relocatable machine code must be linked together with other relocatable object files and library files into the code that runs on the machine.
* **Linker:** Resolves external memory addresses where the code in one file may refer to a location in another file.
* **Loader:** Puts together all the executable object files in memory for execution.

A diagram of a language processing system

Description automatically generated

**1.2 The Structure of a Compiler**

* The actions of a compiler can be divided into the following two phases –
* **Analysis:** Breaks up source program into constituent pieces and imposes a grammatical structure on them.
* Uses the grammatical structure to create an intermediate representation of the program.
* If the program is syntactically ill-formed or semantically unsound, then it provides informative message to the user for correction.
* Collects information about the source program and stores in a data structure called ***symbol table***.
* **Synthesis:** Constructs the desired target program from the intermediate representation and the symbol table.



**1.2.1 Lexical Analysis**

* **Lexical Analyser:** Reads the stream of characters making up the source program and groups the characters into meaningful sequences called ***lexemes***. For each lexeme, it produces as output a ***token*** of the form –

<token\_name, attribute\_value>

* Let’s suppose a program contains the assignment statement –

position = initial + rate \* 60

1. The lexeme ***position*** would be mapped into a token <id, 1> where id is an abstract symbol for identifier and 1 point to the symbol table entry for position.
   * The symbol table entry for an identifier holds information such as its name and type.
2. ***=***  is mapped into the token <=>. This token needs no attribute value.
3. ***initial*** is mapped into the token <initial, 2>.
4. ***rate*** is mapped into the token <rate, 3>.
5. ***\**** is mapped into the token <\*>.
6. ***60*** is mapped into the token <60>.

* Blanks separating the lexemes are discarded by the lexical analyser.
* After lexical analysis the assignment statement would look something like –

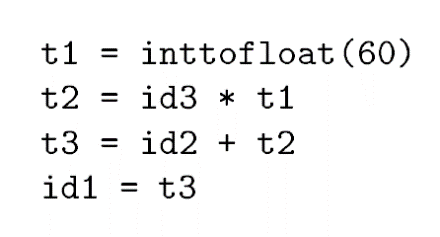
<position, 1> <=> <initial, 2> <+> <rate, 3> <\*> <60>

**1.2.2|1.2.3 Syntax and Semantic Analysis**

* **Syntax analysis or parsing:** Uses the first components of the tokens to form a tree-like intermediate representation that depicts the grammatical structure of the token stream.
* **Syntax tree:** Each interior nodes represents an operation and the children of the represents the arguments of the operation.
* **Semantic analyser:** Uses the syntax tree and symbol table to check the source program for semantic consistency.
* Gathers type information and saves it in syntax tree/symbol table for subsequent use.
* **Type checking:** The compiler checks that each operator has matching operands.
* E.g. the compiler must report an error if floating point number is used to index an array.
* **Coercions:** If a binary arithmetic operator is applied to an integer and a float, then the compiler may convert the integer into a float.

**1.2.4|1.2.5 Intermediate Code Generation and Code Optimisation**

* **Intermediate Code Generation:** Many compilers generate an explicit low-level or machine-like **intermediate representation**.
* It can be thought of as a program for an abstract machine.
* **Three-address code:** Consists of a sequence of assembly-like instructions with three or fewer operands per instruction.



* **Code Optimisation:** This phase attempts to improve the intermediate code so that better target code will result.
* Usually better means faster, but may also mean –
* shorter code
* less power-consuming code

